UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 73990 ADRIANO LIMA DA SILVA 223FE112 - 1 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

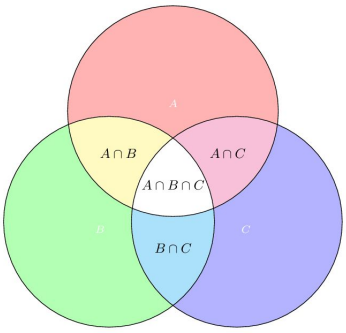
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	4	11	10	15	3	8	12	7
	B	4	7	2	8	5	1	3	15
	C	11	10	13	6	8	5	9	2
2	A	15	6	7	11	5	13	14	8
	B	3	8	9	7	6	1	15	11
	C	15	2	13	11	4	9	8	6
3	A	10	6	1	11	9	12	2	15
	B	3	10	9	11	8	13	7	2
	C	11	7	10	13	14	2	15	5

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-73990 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74005 ALANA KHRISSIE SILVA DE SOUZA 223FE112 - 2 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

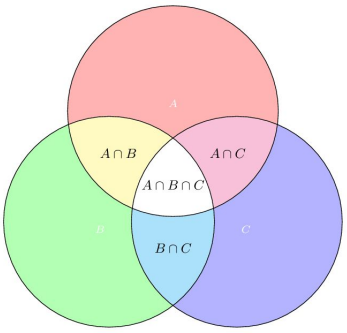
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	13	6	8	15	3	1	10
	B	9	4	15	13	8	1	7	11
	C	4	12	6	13	14	5	7	8
2	A	13	12	3	4	14	5	1	6
	B	4	10	14	9	7	13	5	12
	C	7	9	13	14	4	15	6	8
3	A	11	8	13	6	1	14	7	15
	B	2	12	7	15	3	14	11	4
	C	7	14	15	4	5	11	9	13

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74005 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74012 ARTHUR ADAIR VIEIRA 223FE112 - 3 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

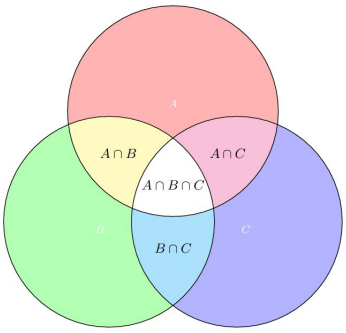
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	8	12	10	4	1	5	7
	B	15	2	8	1	10	9	5	13
	C	1	6	15	9	12	14	2	8
2	A	11	3	9	4	12	15	8	13
	B	7	4	2	15	3	13	8	10
	C	6	2	3	4	9	1	14	12
3	A	11	10	2	5	13	7	3	6
	B	2	14	13	10	5	1	15	11
	C	14	2	13	3	10	9	1	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74012 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74029 ARTHUR HIDEIO NOGUTI 223FE112 - 4 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

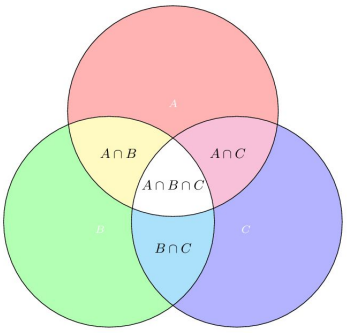
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	6	7	12	2	14	8	3	4
	B	13	2	1	9	3	10	11	14
	C	1	3	13	9	10	8	6	7
2	A	15	4	9	6	11	8	5	13
	B	9	12	6	7	3	1	13	5
	C	6	7	5	3	12	9	10	11
3	A	4	12	14	11	5	6	8	7
	B	3	9	10	5	4	15	6	7
	C	7	2	6	15	1	13	4	9

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74029 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74036 CAMILLA VIRGINIA RODRIGUES DA 223FE112 - 5 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

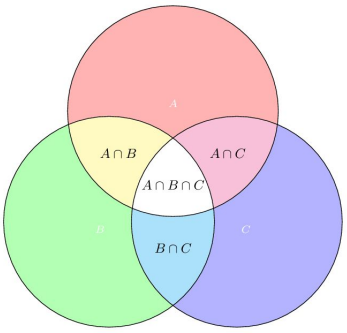
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	5	7	12	10	13	2	3	9
	B	10	2	4	15	6	8	12	7
	C	2	6	14	13	1	11	5	9
2	A	12	6	11	7	2	8	13	1
	B	4	13	8	9	7	12	11	15
	C	12	6	5	2	13	8	9	4
3	A	9	14	1	11	3	4	2	10
	B	13	12	7	10	6	15	9	4
	C	2	10	12	5	7	14	15	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74036 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74043 CARLOS CRISTIAN XAVIER FARIAS 223FE112 - 6 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

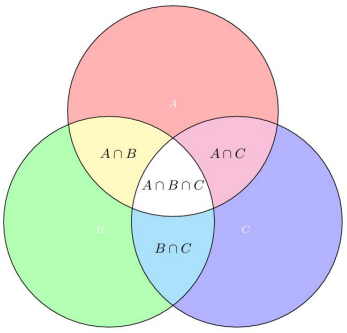
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	7	9	4	10	8	11	3
	B	6	3	13	15	11	10	4	2
	C	1	11	13	14	9	7	2	3
2	A	12	9	15	3	5	6	13	7
	B	15	14	11	7	3	5	6	9
	C	6	9	10	12	14	3	11	8
3	A	2	4	3	11	5	14	7	12
	B	11	3	4	8	5	13	12	15
	C	10	1	7	15	6	11	5	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74050 CASSIA BRUEL MORO DE OLIVEIRA 223FE112 - 7 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

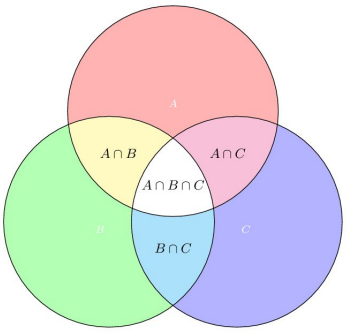
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	13	11	12	5	2	6	1
	B	5	2	15	3	6	1	7	4
	C	8	7	10	3	13	4	15	1
2	A	7	10	8	9	14	6	4	3
	B	13	15	6	14	11	8	5	2
	C	1	9	5	8	12	11	7	13
3	A	6	13	9	3	12	7	2	14
	B	2	11	1	8	6	7	15	12
	C	15	5	11	14	6	2	4	3

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74050 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74067 CRISTIAN PESSATTI DOS ANJOS 223FE112 - 8 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

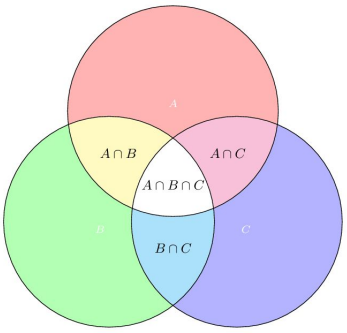
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	8	4	12	11	5	14	13	2
	B	9	11	7	10	14	3	13	6
	C	3	10	2	15	12	11	13	9
2	A	9	13	10	6	5	14	1	7
	B	3	8	5	11	1	6	14	7
	C	3	4	9	11	2	6	7	1
3	A	4	7	5	3	1	14	6	9
	B	1	11	15	3	5	2	7	9
	C	9	4	12	1	8	15	13	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74067 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74074 DIOGO BAYER CAVALLI 223FE112 - 9 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

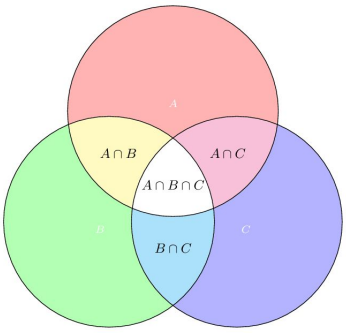
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	1	13	12	10	4	7	8
	B	7	11	9	13	6	14	3	12
	C	10	2	13	15	6	1	4	7
2	A	15	10	7	4	12	5	1	2
	B	15	7	6	1	2	8	14	3
	C	3	10	7	2	4	9	8	12
3	A	6	13	11	7	2	8	15	4
	B	4	10	7	14	1	15	5	9
	C	7	2	9	8	10	13	5	4

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74074 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74081 EDILSON ADRIANO ROMANETCH 223FE112 - 10 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

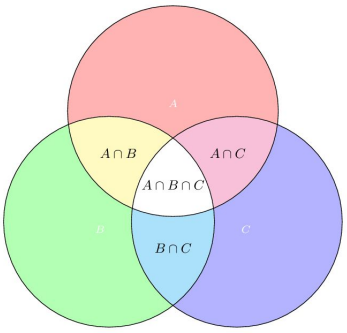
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	15	14	12	10	2	1	4	6
	B	14	13	15	12	2	9	7	3
	C	11	13	14	1	4	10	2	9
2	A	11	13	14	12	5	3	15	9
	B	4	13	15	3	10	12	7	11
	C	4	9	15	11	12	1	8	13
3	A	3	15	7	2	12	13	11	10
	B	12	13	8	9	2	14	11	15
	C	7	9	6	15	4	3	12	14

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74081 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74098 ERICK CINQUINI
223FE112 - 11 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

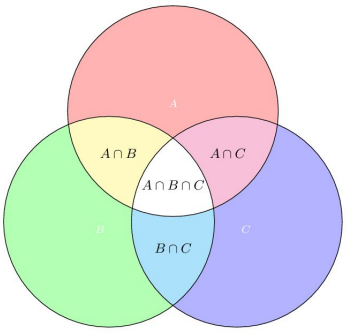
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	7	1	4	3	8	15	13	11
	B	12	5	1	9	15	8	10	3
	C	5	7	2	13	4	14	8	11
2	A	3	15	8	7	12	13	11	14
	B	8	13	12	9	6	2	11	5
	C	13	15	7	9	4	14	12	6
3	A	5	15	11	9	8	10	2	13
	B	6	12	8	7	9	15	11	10
	C	4	15	12	2	6	1	11	10

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74098 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74100 GABRIEL SOUZA BRAGA 223FE112 - 12 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

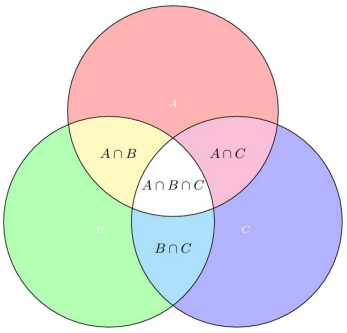
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	6	12	5	10	15	7	14	13
	B	15	6	4	14	8	7	12	9
	C	13	3	7	14	4	9	12	5
2	A	13	4	9	10	2	15	3	7
	B	8	14	15	5	11	7	13	4
	C	8	7	1	13	6	3	5	15
3	A	11	7	10	3	4	1	5	15
	B	1	14	10	12	6	3	7	11
	C	7	6	14	2	1	12	10	5

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74100 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74117 GABRIEL VINICIUS RITT 223FE112 - 13 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

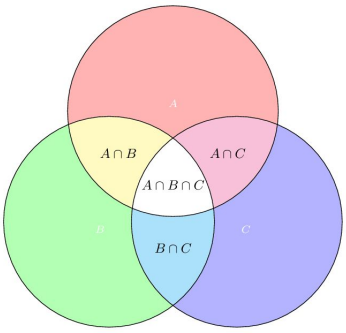
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	5	15	8	7	1	9	12	14
	B	14	6	13	4	3	15	7	12
	C	15	6	14	4	1	13	10	7
2	A	10	2	9	3	14	4	12	15
	B	5	8	13	9	12	15	10	6
	C	6	13	3	14	8	2	9	11
3	A	12	6	3	2	15	5	8	14
	B	14	7	3	2	13	9	1	4
	C	1	10	3	14	13	11	8	15

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74117 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74124 GIULIA TEIXEIRA DUARTE 223FE112 - 14 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

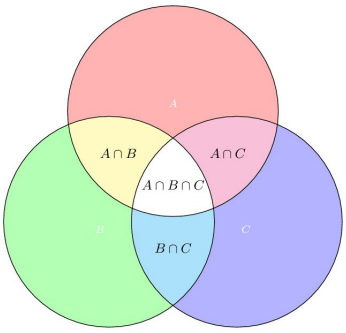
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	11	3	6	12	5	7	10	14
	B	14	3	6	13	2	7	15	4
	C	3	8	6	15	5	10	12	14
2	A	7	8	1	5	3	10	12	4
	B	15	13	2	1	11	12	10	8
	C	9	8	12	2	3	5	7	15
3	A	10	2	15	7	3	8	9	14
	B	2	6	3	14	13	11	10	8
	C	6	14	4	2	10	15	8	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74131 HENRIQUE PINTO MOREIRA MARCOND 223FE112 - 15 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

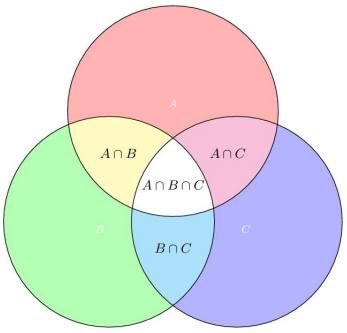
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	3	15	2	13	8	11	7	4
	B	9	5	13	2	15	11	4	14
	C	13	2	1	3	8	12	9	10
2	A	13	8	4	1	7	2	12	14
	B	14	10	15	12	4	13	1	2
	C	3	6	14	7	2	10	5	15
3	A	8	15	13	7	1	6	14	2
	B	3	11	6	2	13	5	9	7
	C	2	5	4	12	3	13	9	8

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74131 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74148 JANAINA KOVALSKI DOS SANTOS 223FE112 - 16 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

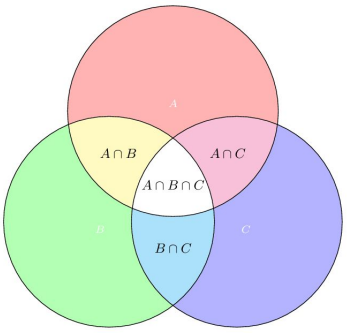
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	12	5	4	9	3	1	13
	B	11	6	13	3	4	15	5	14
	C	14	4	9	8	12	10	11	7
2	A	10	4	14	2	12	8	5	3
	B	2	11	8	9	12	6	13	3
	C	1	15	4	6	11	8	12	10
3	A	11	8	4	12	15	10	6	13
	B	4	10	5	6	9	8	12	7
	C	2	6	14	10	11	5	1	7

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74148 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74898 JHONATAN MARAFIGO M DA SILVA 223FE112 - 17
/ _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

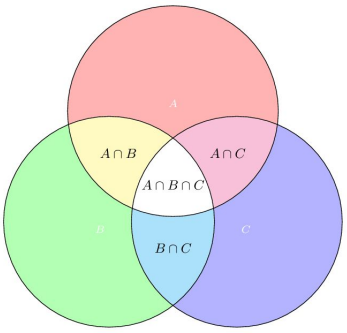
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	9	6	13	8	5	11	7	4
	B	1	14	6	5	15	13	4	11
	C	4	14	11	5	2	1	9	10
2	A	14	6	13	1	4	3	11	10
	B	13	8	14	12	5	15	10	9
	C	14	12	7	13	9	8	11	15
3	A	6	10	11	13	8	7	14	15
	B	2	6	7	3	5	1	13	4
	C	6	3	15	5	9	8	14	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74898 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74155 JONATAS ROBERTO KNAUBER 223FE112 - 18 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

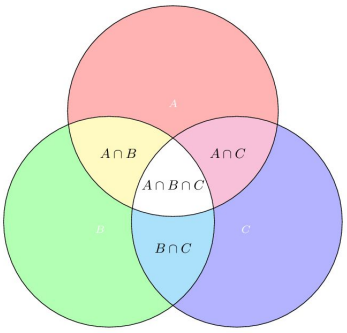
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	11	1	13	10	9	8	4	7
	B	13	8	12	10	6	4	7	14
	C	4	7	15	9	3	8	11	13
2	A	8	5	4	10	3	7	2	1
	B	2	4	10	5	7	3	1	12
	C	6	9	10	7	1	4	8	12
3	A	13	10	2	14	3	1	4	7
	B	15	5	4	9	12	11	2	13
	C	9	5	2	4	15	6	1	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74162 JULIANA MOREIRA FELIX 223FE112 - 19 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

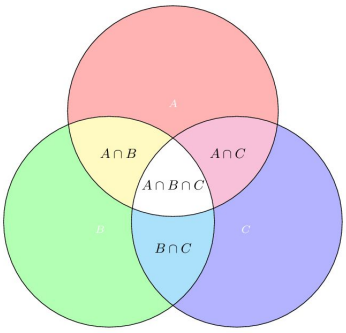
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	1	13	2	12	4	15	14	3
	B	12	13	11	14	3	7	1	6
	C	14	6	5	1	11	10	9	2
2	A	5	1	15	14	10	4	2	13
	B	4	3	2	13	14	11	8	9
	C	11	2	7	9	5	6	13	8
3	A	2	15	4	5	9	7	10	14
	B	1	7	15	12	14	9	8	6
	C	13	14	8	15	12	9	11	10

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74162 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74179 LEONARDO VIDAL BHERING 223FE112 - 20 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

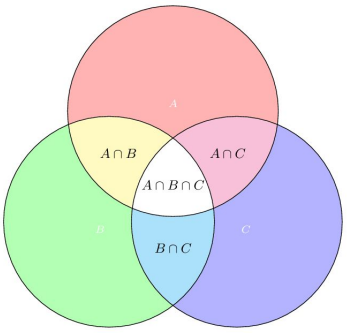
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	6	15	8	3	11	7	9	14
	B	7	11	3	12	2	5	10	14
	C	4	10	11	15	1	3	12	8
2	A	10	11	6	3	1	12	5	4
	B	2	4	12	15	7	3	6	1
	C	8	7	14	5	15	3	1	13
3	A	10	2	5	13	7	6	14	8
	B	6	15	9	3	14	10	12	1
	C	13	14	2	8	15	7	9	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74179 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74186 LUCAS FERREIRA EPIFANIO MELO M 223FE112 - 21 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

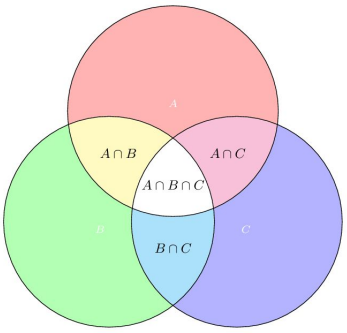
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	15	6	4	3	10	11	1	7
	B	10	7	14	13	4	6	9	11
	C	12	10	4	7	13	8	6	11
2	A	10	7	14	15	4	12	3	9
	B	6	10	7	11	13	8	1	15
	C	7	15	5	9	1	11	2	13
3	A	8	12	2	11	10	14	6	3
	B	13	6	9	8	11	4	7	12
	C	12	4	14	8	15	2	10	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74186 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74193 MARCIANO CARLOS BARRETO 223FE112 - 22 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

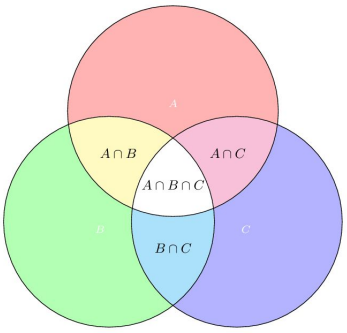
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	5	3	1	13	9	2	15	14
	B	6	3	14	9	12	7	8	11
	C	9	2	8	7	11	10	12	6
2	A	11	14	3	10	13	1	7	9
	B	8	1	11	2	6	10	12	3
	C	3	14	15	6	13	12	5	7
3	A	8	3	10	4	1	5	12	11
	B	10	8	14	11	7	15	12	1
	C	15	12	9	2	3	11	10	14

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74193 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74205 MARIA EDUARDA MATIAS MORALES 223FE112 - 23 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

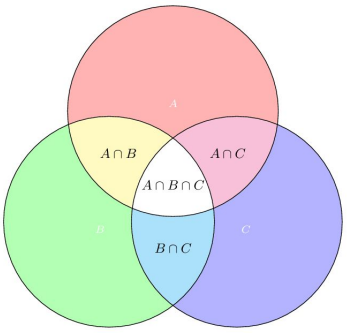
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	13	2	15	7	14	11	3	6
	B	9	13	10	8	5	6	1	11
	C	5	12	1	8	15	14	4	7
2	A	10	15	2	11	3	13	9	1
	B	4	5	1	8	2	9	10	15
	C	13	4	6	7	8	11	9	3
3	A	1	8	6	7	14	9	4	3
	B	8	13	3	4	12	7	11	2
	C	9	2	5	7	6	14	12	8

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74205 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74900 MARIA EDUARDA V MACHADO 223FE112 - 24 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

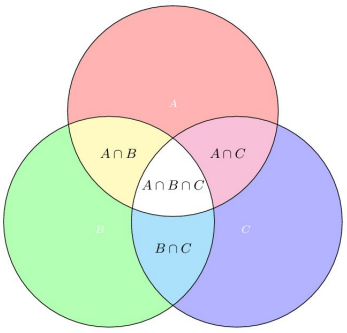
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	13	1	9	10	2	6	8	12
	B	15	12	6	9	14	10	11	3
	C	5	14	6	7	11	4	1	3
2	A	4	12	15	6	11	14	13	3
	B	3	14	8	4	13	7	6	11
	C	4	15	8	7	9	1	3	2
3	A	5	10	6	11	1	15	3	8
	B	5	4	13	11	14	6	3	8
	C	1	5	10	12	11	15	4	9

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74900 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74212 NEY KAWAN DA SILVA 223FE112 - 25 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

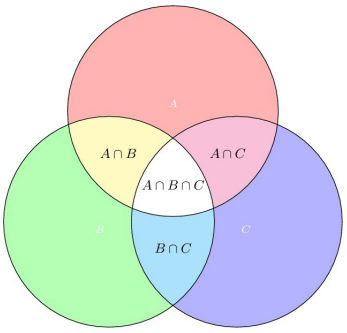
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	4	14	12	5	10	3	1	11
	B	2	12	1	4	15	10	8	5
	C	11	7	1	3	13	8	14	12
2	A	2	5	15	14	9	3	13	4
	B	13	12	15	9	10	8	3	5
	C	9	8	1	11	5	14	10	12
3	A	5	2	11	12	15	6	14	13
	B	10	6	7	1	2	3	4	14
	C	14	11	10	2	3	9	5	1

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74229 NICOLAS BAGGIO MADEIRA 223FE112 - 26 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

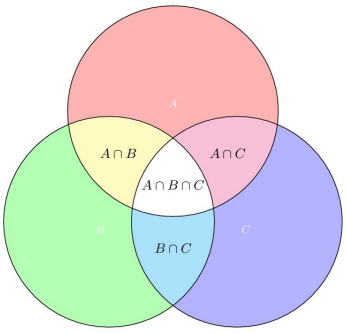
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	9	5	11	3	4	1	13	12
	B	5	12	15	14	10	8	6	11
	C	7	14	6	10	3	8	4	13
2	A	15	3	13	12	9	14	7	11
	B	4	9	8	14	12	15	13	6
	C	6	7	11	1	9	4	15	13
3	A	9	10	8	12	6	14	5	3
	B	4	13	9	15	10	3	1	8
	C	4	5	1	7	2	11	10	3

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74229 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74236 PAULO CEZAR PINZAN DE MATOS 223FE112 - 27 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

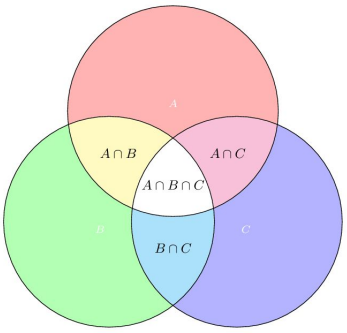
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	9	5	12	11	7	2	6	3
	B	8	15	13	10	1	5	3	4
	C	8	3	5	1	14	6	12	4
2	A	7	14	1	2	8	3	10	5
	B	4	8	11	13	12	15	10	5
	C	9	12	7	15	11	13	8	5
3	A	14	3	10	9	12	4	13	6
	B	3	9	8	11	1	5	10	6
	C	1	9	6	8	12	15	13	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74243 PEDRO HENRIQUE DANDREA 223FE112 - 28 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

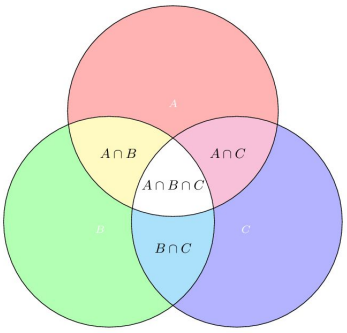
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	4	2	5	1	11	3	14	8
	B	13	3	10	9	8	7	12	15
	C	11	2	12	5	15	6	4	14
2	A	5	12	13	7	3	2	11	6
	B	8	15	6	2	13	12	10	4
	C	6	10	11	8	3	13	12	4
3	A	2	4	3	6	7	9	11	1
	B	14	12	10	15	1	7	4	8
	C	9	7	10	11	14	1	15	5

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74243 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74250 RAFAELLE RIBEIRO DIAS 223FE112 - 29 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

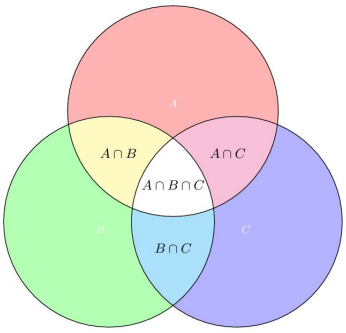
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	14	10	11	13	8	9	7	2
	B	9	7	12	13	4	10	2	3
	C	8	12	9	5	1	15	13	10
2	A	7	14	1	12	5	6	11	10
	B	9	6	8	1	7	12	15	5
	C	8	10	13	2	15	3	11	4
3	A	15	2	9	6	1	5	7	4
	B	14	11	8	7	4	13	15	5
	C	13	4	5	14	8	1	10	7

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74250 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74267 STEFANY PINHEIRO
223FE112 - 30 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

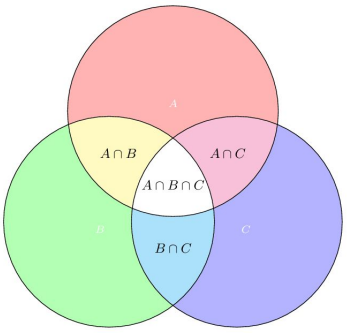
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	7	15	9	4	1	11	6	14
	B	5	12	3	6	9	7	4	2
	C	15	11	1	8	5	12	14	13
2	A	4	7	8	10	3	11	1	5
	B	12	9	7	5	1	6	10	3
	C	3	12	9	4	15	8	14	10
3	A	11	5	3	9	15	8	6	2
	B	9	11	13	2	5	12	10	14
	C	14	7	9	3	8	4	15	2

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74267 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74274 VICTOR PINHEIRO COSTA LOSS 223FE112 - 31 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

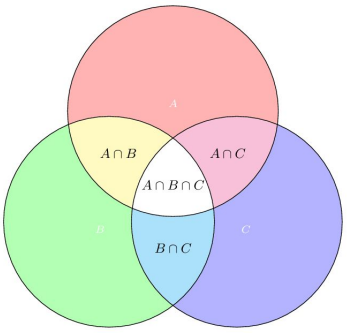
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	8	15	4	9	11	12	1	2
	B	9	7	4	14	12	13	5	1
	C	6	3	15	11	14	4	13	8
2	A	9	1	13	14	2	4	12	15
	B	4	1	10	8	7	9	13	2
	C	7	12	13	4	6	11	9	2
3	A	10	2	13	6	12	15	7	5
	B	9	6	8	5	2	15	4	11
	C	2	10	11	15	9	7	4	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74281 VITOR ALBERTI
LIMA 223FE112 - 32 _____ / _____ / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

a é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos :.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

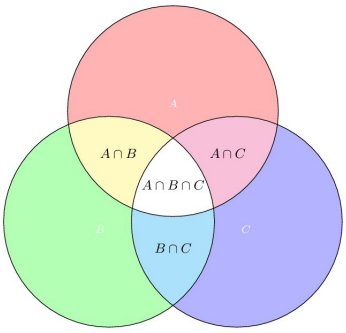
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	8	12	1	3	6	14	7	10
	B	11	9	12	8	14	15	1	10
	C	11	15	10	2	5	6	13	14
2	A	11	1	7	8	3	6	13	14
	B	6	4	13	3	9	1	11	7
	C	7	11	9	3	6	10	4	14
3	A	14	2	11	13	10	12	8	7
	B	10	4	7	12	13	1	5	2
	C	4	15	8	14	13	12	6	5

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74281 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74298 VITOR PRADO ZAMBALDI 223FE112 - 33 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

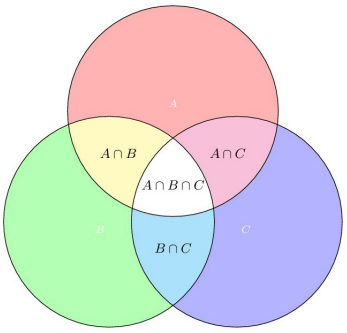
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	7	14	1	10	9	15	12	5
	B	5	7	6	11	14	2	13	12
	C	5	9	11	1	13	7	10	3
2	A	10	14	9	1	8	13	7	11
	B	12	13	11	4	5	6	3	1
	C	14	7	6	8	5	4	11	2
3	A	13	14	11	1	4	3	15	8
	B	3	1	6	5	10	2	12	7
	C	1	6	4	15	12	3	13	11

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74298 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74300 WELLITON DA SILVA OLIVERIO 223FE112 - 34 _____ / _____ /

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

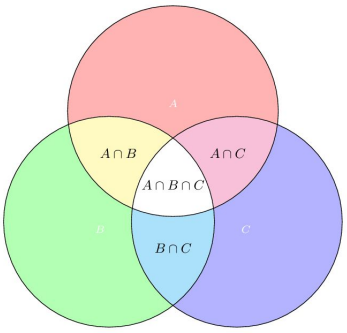
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	15	1	2	13	14	9	12	6
	B	2	3	14	8	4	11	6	7
	C	15	6	14	5	12	7	1	10
2	A	12	7	14	15	8	11	4	10
	B	6	2	14	11	5	9	13	10
	C	14	5	6	12	1	8	4	11
3	A	6	8	10	4	1	5	11	9
	B	13	8	1	2	10	5	4	12
	C	6	8	4	7	1	5	13	14

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74300 - /

UFPR - UP - UTFPR - PUCPr – 19/01/2023 - 17:02:38.1
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVO934a, V: 1.06 74317 WESLLEY ALEXANDRE DA SILVA 223FE112 - 35 _____ / / _____

Tuplas, Dicionários e Conjuntos em Python

Uma tupla é como se fosse uma lista, mas é uma lista imutável. Uma tupla é criada usando-se parênteses e dentro deles os elementos da tupla. Lembre que as listas são criadas com colchetes. Acompanhe

```
>>> a=[1,2,3,4]
>>> a[2]=55
>>> a
[1, 2, 55, 4]
>>> b=(1,2,3,4)
>>> b
(1, 2, 3, 4)
>>> b[2]=55
TypeError: 'tuple' object does not
support item assignment
```

Tuplas suportam a maior parte das operações de listas, como fatiamento e indexação (para leitura) e são bastante usadas para listas de constantes. Ao criar uma lista de elementos separados por vírgulas, sem nenhum delimitador, você estará criando uma tupla. Esta operação é chamada de empactotamento. Acompanhe

```
>>> viva = 1,2,3,4
>>> viva
(1, 2, 3, 4)
```

Dicionários

Uma estrutura de dados parecido a lista, mas com o detalhe: enquanto listas sao acessadas pelo índice, dicionários são acessados por uma parte de seu conteúdo, denominada chave. Assim, o dicionário é composto de pares: chave e conteúdo e o que o dicionário faz é associar uns a outros. Os dicionários são criados usando-se chaves {} O formato é

```
a = { chave: valor, chave: valor, ... }
```

`a` é o nome do diconário, e dentro dele há pares de chave:valor, separadas pelo caractere dois pontos `:`.

Depois de criado um dicionário é consultado escrevendo-se seu nome, colchete, a chave desejada e fecha-colchete. Acompanhe

```
>>> documentos={'rg':1234, 'motorista':34567,
'passaporte': 8998, 'coxa': 23454}
>>> documentos['passaporte']
8998
```

Quando se atribui um valor a uma chave, se a mesma não existir ainda, será criada com este valor. Se ela já existisse, teria seu valor alterado para aquele agora fornecido.

Um detalhe importante é que a ordem dos elementos dentro de um dicionário não pode ser importante pois não é garantida.

Se um acesso é feito a um dicionário para uma chave inexistente ocorre um erro de `Keyerror`. Para evitar tal erro, se necessário, deve-se fazer uma consulta com o operador `in`

```
if 'pis/pasep' not in documentos:
   documentos['pis/pasep'] = 555
```

As chaves e os conteúdos podem ser acessados como se fossem uma lista, veja

```
>>> documentos.keys()
dict_keys(['rg', 'motorista', 'passaporte',
   'coxa', 'pis/pasep'])
>>> documentos.values()
dict_values([1234, 34567, 8998, 23454, 555])
```

Note que eles voltam na forma de geradores. Podem ser usados diretamente em ciclos `for`, ou transformados em lista usando-se `list`. Para apagar uma chave, usa-se a instrução `del`. Veja

```
>>> del documentos['coxa']
>>> documentos
{'rg': 1234, 'motorista': 34567,
   'passaporte': 8998, 'pis/pasep': 555}
```

Nada impede que o dicionário associe uma chave a uma lista (ou a outro dicionário...). Suponha que para cada documento, quero saber o número e o ano em que foi emitido. Poderia fazer

```
>>> documentos={'rg':[1234, 1973],
  'motorista':[34567,1980],
'passaporte': [8998,2001],
  'coxa': [23454,1985]}
```

Conjuntos

Um conjunto é uma lista de coisas (um conjunto ou como se diz em Python: *set*), com 2 características e algumas operações específicas. As características:

- Não importa a ordem original dos elementos
- Não há elementos repetidos: se houver o Python deixa uma cópia só.

Em razão da primeira propriedade os conjuntos não suportam indexação nem fatiamento.

O conjunto pode ser criado oferecendo um conjunto de elementos à função *set* ou mais simplesmente, colocando os elementos entre chaves. Acompanhe

```
>>> a = {1,2,3,4,5,3,2}
>>> a
{1, 2, 3, 4, 5}
>>> b=set([1,2,3,4,5,1])
>>> b
{1, 2, 3, 4, 5}
```

As operações associadas a conjuntos, são aquelas da matemática básica e são:

Intersecção: Dados dois conjuntos A e B, a intersecção de A e B, $A \cap B$ é o conjunto dos elementos que estão em A e em B. A intersecção usa o símbolo `&`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A & B
{8, 6}
```

União: dados A e B, conjuntos, a união de A com B, denotada $A \cup B$ é o conjunto dos elementos que estão em A ou em B. Seu símbolo é `|`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A|B
{2, 4, 5, 6, 7, 8, 9, 10}
```

Diferença: Retorna os elementos de A que não estão em B. Seu símbolo `-`.

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A-B
{2, 10, 4}
```

Diferença simétrica: Retorna todos os elementos de ambos os conjuntos que pertencem somente a um deles. O símbolo é `^`. Veja

```
>>> A={2,4,6,8,10}
>>> B={5,6,7,8,9}
>>> A^B
{2, 4, 5, 7, 9, 10}
```

Acompanhe agora alguns exemplos:

```
>>> a=[1,2,34,5,6,7]
>>> b=set([1,2,3,4,5,6])
>>> a
[1, 2, 34, 5, 6, 7]
>>> b
{1, 2, 3, 4, 5, 6}
>>> a=set(a)
>>> a
{1, 2, 34, 5, 6, 7}
>>> b
{1, 2, 3, 4, 5, 6}
>>> a|b       # uniao
{1, 2, 34, 3, 5, 6, 7, 4}
>>> a&b       # intersecçao
{1, 2, 5, 6}
>>> a-b       # diferença
{34, 7}
>>> a^b       # diferença simétrica
{34, 3, 4, 7}
```

Uma aplicação prática do uso de dicionários. Seja a sequência de Fibonacci que é $1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ e cuja formulação é:

$$\begin{cases} F(1) & = 1 \\ F(2) & = 1 \\ F(n) & = F(n-1) + F(n-2) \quad se\ n \geq 3 \end{cases}$$

Implementando diretamente esta definição em Python, fica-se com

```
def fibo(n):
  if n<3:
    return 1
  else:
    return fibo(n-1)+fibo(n-2)
xx = int(input('informe n '))
print(fibo(xx))
```

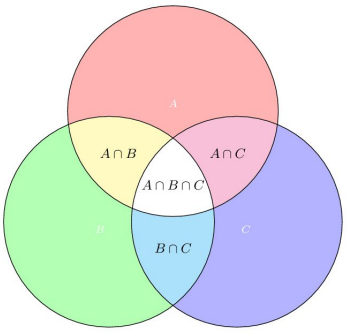
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas a fibo (fibo(1) e fibo(2)). Para calcular fibo(4) fazem-se 2 chamadas a fibo(2) e fibo(3). Como fibo(3) são 2 chamadas, no total são 4 chamadas. Fibo(5) são 6, fibo(6) são 10, fibo(7), 16; fibo(8) 26, fibo(9) 42... e as chamadas também compõe uma outra sequência de Fibonacci. O caso é que para calcular f(100) o computador deve demorar mais de 1 mês de processamento. A alternativa é o que se chama *algoritmo memoizável* que nada mais faz do que guardar o resultado já calculado. Onde ? Neste caso em um dicionário. Acompanhe

```
d = {1:1, 2:1}
def fibm(n):
  aa=d.get(n)
  if aa is None:
    d[n]=fibm(n-1)+fibm(n-2)
    return d[n]
  else:
    return aa
print(fibm(xx))
```

Olhe os tempos: fibm(30) demora 5 milésimos, fibm(40), 9 milésimos e fibm(100) demora 9 milésimos também.

☞ Para você fazer

Vamos brincar um pouco com a Teoria dos Conjuntos com a ajuda do Python. Suponha três conjuntos A, B e C, assim dispostos



Você deve achar a cardinalidade dos conjuntos a seguir descritos em 3 casos e deve somar os valores encontrados.

M= $A \cap B \cap C$
N= $A \cap B \cap C^c$
P= $A \cap B^c$
Q= $C \cap (A \cup B)^c$
R= $A \cap (B \cup C)$
S= $(B \cap (A \cup C)^c) \cup (A \cap C \cap (A \cap B \cap C)^c)$
T= $A \cup B \cup C$
U= $(A \cap B \cap C^c) \cup (A \cap C \cap B^c) \cup (B \cap C \cap A^c)$
V= $(A \cap B) \cup (A \cap C) \cup (B \cap C)$
W= $(A \cup B \cup C) \cap ((A \cap B) \cup (A \cap C) \cup (B \cap C))^c$
Lembre que $A \cap B^c \Leftrightarrow A - B$
Eis os valores a usar:

1	A	8	5	3	12	7	13	2	10
	B	1	13	4	5	7	10	6	11
	C	4	10	6	12	3	9	2	8
2	A	4	10	3	1	15	7	8	5
	B	7	2	12	13	6	10	5	15
	C	10	7	5	14	15	1	8	6
3	A	4	3	9	10	11	1	7	2
	B	8	11	9	4	5	13	14	3
	C	8	15	14	2	4	11	3	12

Para a resposta, preencha a cardinalidade (o número de elementos) de cada conjunto.

ex.	M	N	P	Q	R	S	T	U	V	W
1										
2										
3										
Σ										

112-74317 - /