UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75789 ARIEL GUSTAVO RODRIGUES DA S 24PRG409 - 1 limite entrega 10/10/24 _____ / _____ / _____

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))
```

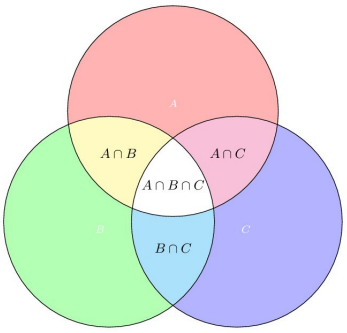
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	12	11	6	14	5	9	8	13
	B	14	1	6	7	4	13	5	9
	C	12	4	14	6	13	15	5	1
2	A	1	10	12	8	5	3	2	9
	B	8	3	15	6	14	11	5	13
	C	13	11	3	8	5	12	1	7
3	A	15	9	2	13	6	4	12	10
	B	12	4	7	11	10	14	1	3
	C	14	3	8	7	5	11	10	6

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										

409-75789 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75796 CASSIANO MAGNO CHAGAS E SA 24PRG409 - 2 limite entrega 10/10/24 _____ / _____ / _____

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))
```

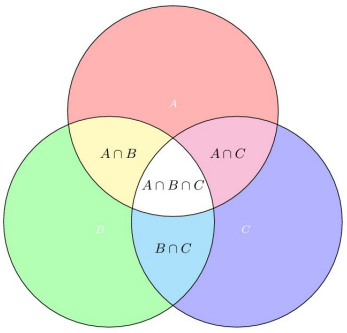
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	10	9	7	12	15	14	5
	B	11	9	4	15	3	6	12	10
	C	12	14	10	15	8	4	13	11
2	A	5	6	12	11	14	9	7	13
	B	9	7	5	3	8	12	2	13
	C	4	3	2	6	9	15	14	11
3	A	1	10	11	13	3	5	6	8
	B	7	3	1	11	10	9	5	2
	C	6	2	1	8	15	3	9	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										

409-75796 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75808 CHRISTOPHER BUCH FILIPAK 24PRG409 - 3 limite entrega 10/10/24 _____ / _____ / _____

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))
```

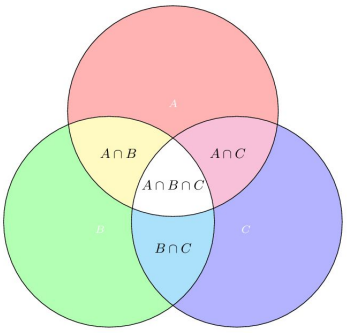
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	12	5	1	8	11	7	10	4
	B	12	2	3	4	10	14	8	13
	C	9	12	7	2	1	5	3	10
2	A	5	7	6	2	11	8	15	12
	B	6	15	12	10	9	8	5	14
	C	6	13	3	8	10	9	14	7
3	A	5	7	15	8	6	10	9	3
	B	8	11	14	3	9	13	5	6
	C	3	13	4	6	15	14	9	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										

409-75808 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75815 ERAN MARTINEZ RAMOS 24PRG409 - 4 limite entrega 10/10/24 / _____ / _____

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))
```

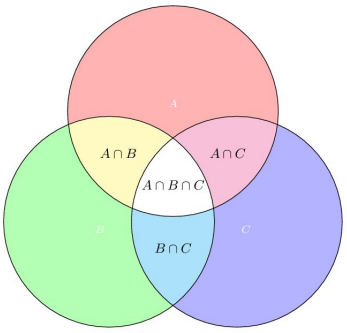
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	4	6	13	9	5	1	15	12
	B	13	10	12	9	11	2	15	8
	C	6	4	3	13	8	7	5	1
2	A	15	13	7	8	14	11	12	1
	B	4	2	10	3	1	11	13	5
	C	15	14	5	12	13	10	9	7
3	A	14	6	11	2	4	7	10	8
	B	14	9	2	5	11	4	7	12
	C	8	5	1	12	4	11	13	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										

409-75815 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75822 HENRIQUE IVANKIO GUSSE 24PRG409 - 5 limite entrega 10/10/24 / _____ / _____

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))
```

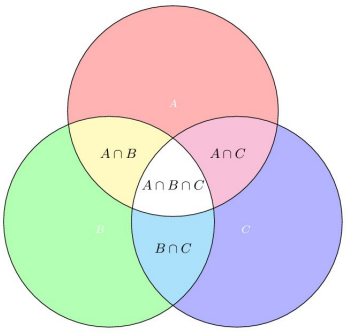
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	6	1	4	14	3	9	8
	B	12	3	6	10	4	14	2	7
	C	7	6	11	2	4	5	14	1
2	A	13	8	15	7	5	3	14	1
	B	12	6	5	13	8	9	4	3
	C	12	8	15	13	4	3	2	5
3	A	2	1	4	12	10	15	11	3
	B	14	13	15	11	7	2	9	5
	C	6	11	1	5	12	2	3	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										

409-75822 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75839 IAN DOLABELLA OHATA 24PRG409 - 6 limite entrega 10/10/24 / _____ / _____

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))
```

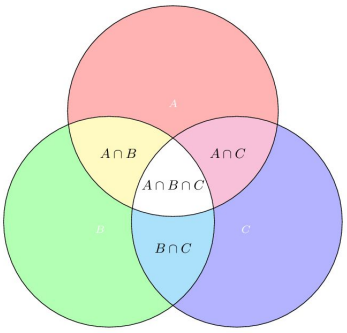
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	2	7	11	10	5	1	8
	B	3	11	9	15	2	4	14	6
	C	9	7	14	4	11	3	2	15
2	A	2	15	10	3	14	5	12	4
	B	14	1	10	5	11	8	13	12
	C	6	1	11	10	2	5	14	15
3	A	10	7	11	14	8	12	13	1
	B	6	8	11	14	2	4	5	9
	C	7	12	1	6	11	15	3	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										

409-75839 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75846 JOAO BONFIN LINO
24PRG409 - 7 limite entrega 10/10/24 _____ / / _____

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))
```

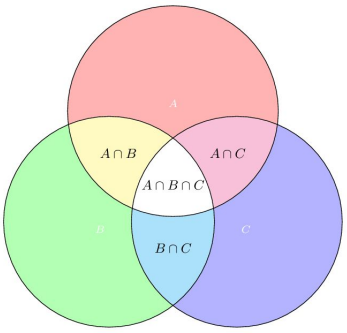
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	11	12	5	9	10	4	7	2
	B	11	4	8	5	7	6	15	9
	C	9	15	11	7	12	6	1	3
2	A	8	7	5	12	15	9	3	1
	B	4	9	3	13	12	15	14	5
	C	8	6	4	12	1	9	3	14
3	A	5	13	1	15	12	9	2	11
	B	9	7	8	13	15	14	10	12
	C	15	6	11	14	8	7	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										

409-75846 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75853 JOAO LUCA FERNANDES DE OLIVE 24PRG409 - 8 limite entrega 10/10/24 _____ / _____ / _____

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))
```

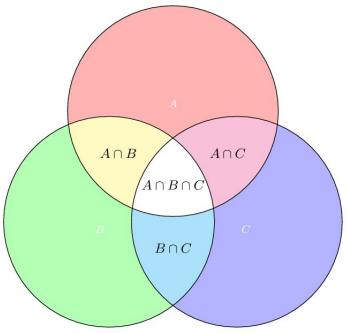
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	3	14	4	13	15	7	6	5
	B	15	2	13	9	14	5	1	11
	C	5	6	2	13	14	10	7	15
2	A	1	9	3	4	2	15	12	13
	B	8	9	5	3	2	15	7	14
	C	5	4	14	6	2	1	7	11
3	A	2	8	15	9	12	10	14	1
	B	2	3	5	7	6	12	1	9
	C	14	6	15	5	8	11	3	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										

409-75853 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75860 JOAO PEDRO DE BRITO DUARTE 24PRG409 - 9 limite entrega 10/10/24 _____ / _____ / _____

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))
```

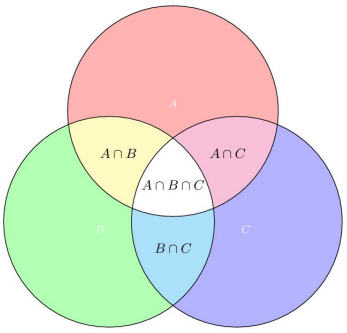
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	15	7	8	14	3	10	9
	B	2	6	3	5	1	14	7	15
	C	15	13	4	7	1	5	3	2
2	A	15	5	11	8	1	9	10	14
	B	2	3	9	8	12	5	10	1
	C	10	15	3	14	4	8	2	9
3	A	11	7	8	3	10	12	6	4
	B	2	9	1	4	12	6	10	15
	C	6	9	11	8	10	14	4	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										

409-75860 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75877 JOAO PEDRO MARIANO SUEKE 24PRG409 - 10 limite entrega 10/10/24 _____ / _____ / _____

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))
```

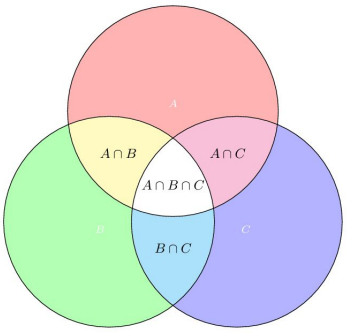
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	8	4	12	6	5	13	15	9
	B	8	15	11	5	3	10	2	4
	C	15	3	14	10	6	8	12	4
2	A	13	12	9	6	5	15	10	3
	B	15	9	14	10	5	1	2	4
	C	10	4	3	9	5	7	1	14
3	A	7	10	9	8	4	14	6	3
	B	9	8	5	14	13	6	11	3
	C	2	11	4	3	9	8	13	6

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										

409-75877 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75884 JOAO VICTOR GOMES VILELA GER 24PRG409 - 11 limite entrega 10/10/24 _____ / _____ / _____

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))
```

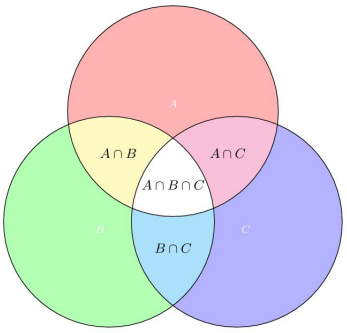
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	11	14	15	8	4	5	1	9
	B	15	5	1	6	9	8	13	10
	C	12	3	13	9	14	5	2	10
2	A	9	4	8	2	10	3	11	6
	B	1	2	7	10	12	15	11	6
	C	4	1	15	10	12	11	2	13
3	A	10	2	12	5	6	15	13	3
	B	8	12	7	3	1	2	13	9
	C	11	7	8	14	13	1	2	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										

409-75884 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75989 LUCAS ANTOSZCZYSZEN 24PRG409 - 12 limite entrega 10/10/24 _____ / _____ / _____

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))
```

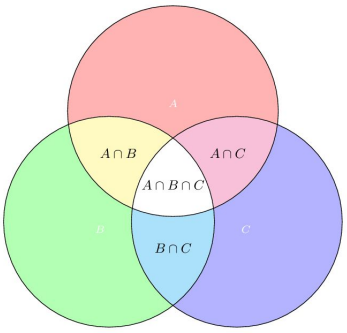
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	15	9	6	3	13	2	7	11
	B	6	3	4	13	2	15	1	9
	C	1	13	7	5	11	8	15	4
2	A	12	5	8	3	2	4	10	15
	B	1	8	9	11	5	10	13	14
	C	5	14	1	6	8	9	2	10
3	A	14	4	11	15	5	3	1	8
	B	10	12	15	9	2	3	8	14
	C	6	12	1	2	9	4	11	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										

409-75989 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75891 LUIZ FELIPE GUEDES BUCCHERI 24PRG409 - 13 limite entrega 10/10/24 _____ / _____ / _____

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))
```

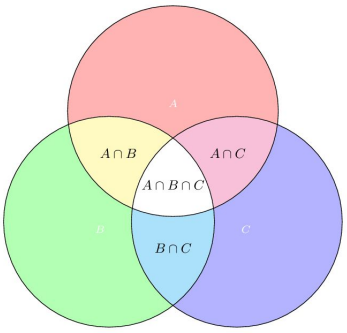
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	8	10	7	5	14	3	1	15
	B	9	13	5	8	2	10	15	1
	C	3	7	13	15	11	4	14	9
2	A	3	2	1	15	13	11	12	7
	B	10	11	2	13	14	7	15	8
	C	11	3	12	1	14	15	10	7
3	A	12	6	11	14	4	10	1	15
	B	10	3	9	2	8	11	12	15
	C	13	14	12	10	9	15	4	6

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										

409-75891 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75903 MANUELLA LEAL DE MEIRELLES F 24PRG409 - 14 limite entrega 10/10/24 _____ / _____ / _____

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))
```

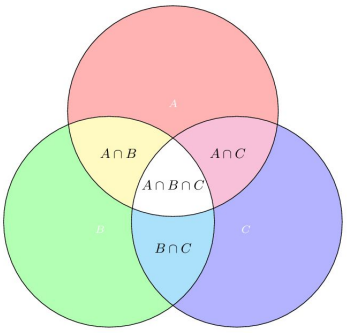
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	7	2	9	15	11	3	1	14
	B	6	7	3	4	2	13	8	14
	C	10	15	5	9	14	1	7	4
2	A	6	13	15	10	3	1	9	11
	B	6	14	4	13	10	12	11	7
	C	15	9	8	12	13	14	1	7
3	A	1	9	13	2	5	8	3	15
	B	2	7	15	14	3	4	6	9
	C	5	14	7	9	1	10	12	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										

409-75903 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75910 MARIA EDUARDA DA SILVA 24PRG409 - 15 limite entrega 10/10/24 _____ / _____ / _____

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))
```

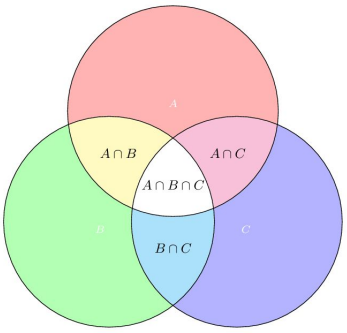
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	9	1	13	3	14	10	7	2
	B	13	5	1	8	7	9	4	2
	C	5	7	15	10	12	13	11	3
2	A	10	13	12	5	11	7	15	4
	B	6	4	2	10	15	13	8	9
	C	1	10	8	4	15	2	13	14
3	A	4	6	5	2	11	7	12	14
	B	1	3	15	6	7	9	12	13
	C	5	10	7	4	1	14	8	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										

409-75910 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75927 MATHEUS AITA FABRICIO DE CAR 24PRG409 - 16 limite entrega 10/10/24 _____ / _____ / _____

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))
```

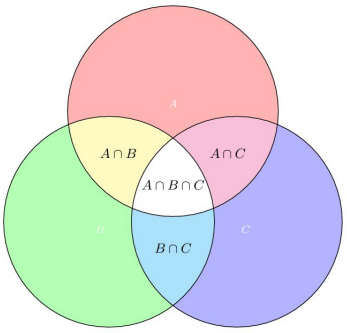
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	1	5	14	4	15	7	10	3
	B	2	10	3	8	14	6	1	4
	C	14	3	5	2	15	9	6	1
2	A	15	8	4	12	6	10	3	11
	B	14	6	10	4	15	1	11	5
	C	5	11	6	2	8	3	1	15
3	A	9	12	4	2	13	10	14	15
	B	7	5	13	11	6	2	3	1
	C	10	13	7	15	11	4	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										

409-75927 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75934 MATHEUS LEANDRO DE BITENCOUR 24PRG409 - 17 limite entrega 10/10/24 _____ / _____ / _____

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))
```

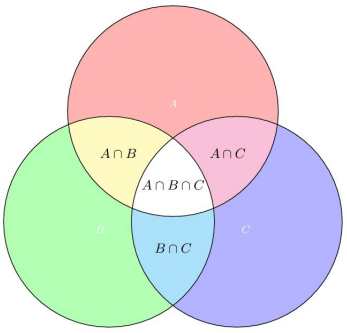
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	3	4	10	12	8	9	13	14
	B	9	4	2	8	6	10	7	11
	C	1	3	2	11	4	10	9	5
2	A	11	8	9	2	1	4	6	15
	B	6	11	4	5	15	14	9	12
	C	14	15	1	6	2	12	3	8
3	A	14	12	2	13	3	9	10	5
	B	5	3	2	10	9	11	14	13
	C	8	13	2	3	1	4	7	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										

409-75934 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75941 MAYCON TAVARES WOLPE 24PRG409 - 18 limite entrega 10/10/24 / _____ / _____

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))
```

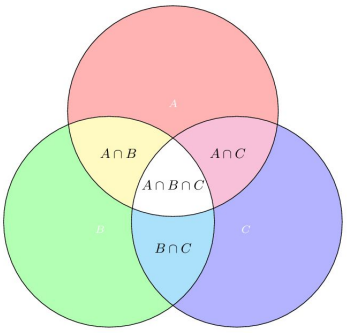
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	6	7	15	2	14	3	8	9
	B	1	5	13	3	15	8	14	7
	C	15	3	2	4	5	1	8	7
2	A	5	11	4	6	7	3	8	10
	B	3	11	9	5	13	8	10	2
	C	5	4	11	6	7	1	15	8
3	A	8	3	7	6	13	14	1	10
	B	10	5	1	13	3	7	4	2
	C	14	7	12	10	15	5	6	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										

409-75941 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75958 PEDRO TOLEDO LEAL 24PRG409 - 19 limite entrega 10/10/24 / _____ / _____

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))
```

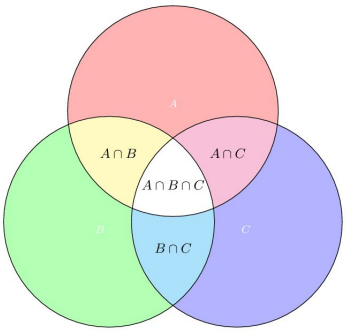
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	11	12	9	1	3	4	15	8
	B	11	5	7	1	10	4	8	13
	C	5	9	1	4	7	15	11	10
2	A	11	13	10	1	15	5	4	7
	B	14	5	8	10	11	6	4	1
	C	4	2	11	15	1	10	13	12
3	A	13	7	6	1	11	5	2	12
	B	12	15	2	13	6	5	4	9
	C	5	3	7	4	12	13	9	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										

409-75958 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75965 PHELIPE GABRIEL LIMA DA SILV 24PRG409 - 20 limite entrega 10/10/24 _____ / _____ / _____

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))
```

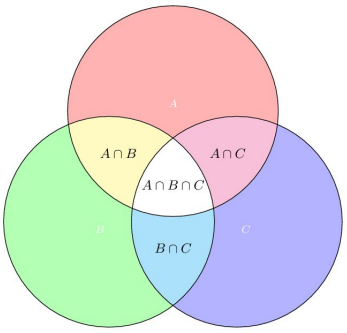
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	7	12	6	14	1	15	8	9
	B	7	6	8	9	14	5	13	15
	C	12	10	6	2	4	7	13	3
2	A	6	11	12	1	5	3	10	14
	B	6	15	4	11	14	13	8	9
	C	12	15	9	7	3	11	10	2
3	A	15	11	7	9	13	4	6	10
	B	7	13	14	9	11	12	2	10
	C	6	9	15	4	14	12	13	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										

409-75965 - e /nt

UFPR - UP - UTFPR - PUCPr – 23/09/2024 - 09:25:28.3
Prof Dr P Kantek (pkantek@gmail.com)
Tuplas, Dicionários e Conjuntos em Python
VIVX934a, V: 1.06 75972 REBECA CABRAL DOS SANTOS 24PRG409 - 21 limite entrega 10/10/24 _____ / _____ / _____

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))
```

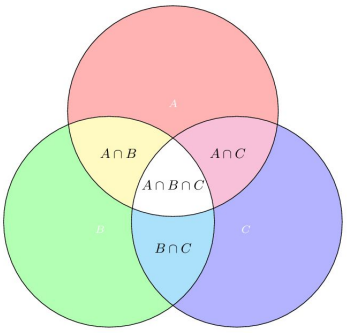
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas 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	8	14	15	11	5	13	7	3
	B	13	7	10	3	6	15	4	14
	C	13	11	4	5	15	1	10	6
2	A	10	4	8	3	13	1	7	2
	B	3	4	5	7	8	12	15	14
	C	10	11	12	1	14	6	9	5
3	A	2	4	11	9	5	8	6	13
	B	1	14	13	6	9	12	7	8
	C	10	6	11	1	3	9	12	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										

409-75972 - e /nt