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])
```

Bote 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))
```

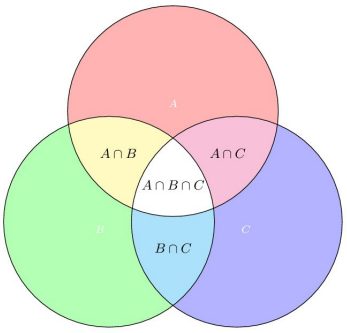
Cujos tempos de execução são: F(20)=6 milésimos de segundo; F(30)=0.42 segundos, F(40)=49 segundos, F(41)=80 segundos e F(42)= 131 segundos e paramos por aqui. O problema desta implementação é o recálculo do mesmo valor para a função muitas vezes. Acompanhe: para calcular fibo(3) fazem-se 2 chamadas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