Rafael Vieira Coelho

rafaelvc2@gmail.com

PARTE 1

9 - Tuplas

9.1 - Tuplas

Há um outro tipo em Python chamado tupla (*tuple*) que é similar a uma lista exceto por ele ser imutável.

Sintaticamente, uma tupla é uma lista de valores separados por vírgulas:
Embora não seja necessário, é convencional colocar tuplas entre parênteses:

```
tupla = ('a', 'b', 'c', 'd', 'e')
```

Para criar uma tupla com um único elemento, temos que incluir uma vírgula final:

```
t1 = ('a',)
type(t1)
```

<class 'tuple'>

Sem a vírgula, Python entende ('a') como uma string entre parênteses:

Mutabilidade e tuplas

Por exemplo, se tivermos um retângulo que sempre deva ter determinado tamanho, podemos garantir que seu tamanho não mudará colocando as dimensões em uma tupla:

```
dimensions = (200, 50)
print(dimensions[0])
200
print(dimensions[1])
50
dimensions[0] = 250
```

```
Traceback (most recent call last):
 File "dimensions.py", line 3, in <module>
  dimensions[0] = 250
TypeError: 'tuple' object does not support item
assignment
```

python™

Atribuições de tupla

De vez em quando, é necessário trocar entre si os valores de duas variáveis. Com operações de atribuição convencionais, temos que utilizar uma variável temporária.

Por exemplo, para fazer a troca entre a e b:

```
temp = a
a = b
b = temp
```

Se você tiver que fazer isso com frequência, esta abordagem se torna incômoda. Python fornece uma forma de atribuição de tupla que resolve esse problema elegantemente:

```
a, b = b, a
```

Atribuição de múltiplos valores

O lado esquedo é uma tupla de variáveis; o lado direito é uma tupla de valores.

Cada valor é atribuído à sua respectiva variável.

Todas as expressões do lado direito são avaliadas antes de qualquer das atribuições.

Esta característica torna as atribuições de tupla bastante versáteis.

Naturalmente, o número de variáveis na esquerda e o número de valores na direita deve ser igual:

```
 a, b, c, d = 1, 2, 3
ValueError: unpack tuple of wrong size
```

Tuplas como valores de retorno

Funções podem retornar tuplas como valor de retorno.

Por Exemplo, nós poderíamos escrever uma função que troca dois parâmetros entre si:

```
def troca(x, y):
        return y, x
```

Então nós poderíamos atribuir o valor de retorno para uma tupla com duas variáveis:

```
a, b = troca(a, b)
```

9.2 - Dicionários e tuplas

- Os tipos compostos que você aprendeu - strings, listas e tuplas - utilizam inteiros como indices. Se você tentar utilizar qualquer outro tipo como índice, você receberá um erro.

- Dicionários são similares a outros tipos compostos exceto por eles poderem usar qualquer tipo imutável de dados como índice.

- Como exemplo, nos criaremos um dicionário para traduzir palavras em Inglês para Espanhol. Para esse dicionário, os indices serão strings.

Dicionários e tuplas

Uma maneira de criar um dicionário é começando com um dicionário vazio e depois adiconando elementos. Um dicionário vazio é denotado assim {}:

```
ing2esp = {}
ing2esp['one'] = 'uno'
ing2esp['two'] = 'dos'
```

Dicionários e tuplas

Considere um jogo com alienígenas que possam ter cores e valores de pontuação diferentes.

O dicionário simples a seguir armazena informações sobre um alienígena em particular:

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0['color'])
print(alien_0['points'])
```

```
green
5
```

python™

Adicionando Novos Pares em Dicionários

Um dicionário em Python é uma coleção de pares chave-valor.

Cada chave é conectada a um valor, e você pode usar uma chave para acessar o valor associado a ela.

O valor de uma chave pode ser um número, uma string, uma lista ou até mesmo outro dicionário.

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'y_position': 25,' x_position': 0}
```

Modificando Valores em um Dicionário

```
alien_0 = {'color': 'green'}
print("The alien is " + alien_0['color'] + ".")
alien_0['color'] = 'yellow'
print("The alien is now " + alien_0['color'] + ".")
```

```
The alien is green.
The alien is now yellow.
```

Operações dos Dicionários

O comando **del** remove um par chave-valor de um dicionário.

Por exemplo, o dicionário abaixo contem os nomes de varias frutas e o numero de cada fruta em no estoque:

```
inventario = {'abacaxis': 430, 'bananas': 312, 'laranjas': 525, 'peras': 217}
print (inventario)
```

```
{'laranjas': 525, 'abacaxis': 430, 'peras': 217, 'bananas': 312}
```

python™

Operações dos Dicionários

Por exemplo, o dicionário abaixo contem os nomes de varias frutas e o numero de cada fruta em no estoque:

```
inventario = {'abacaxis': 430, 'bananas': 312, 'laranjas': 525, 'peras': 217}
```

Se alguém comprar todas as peras, podemos remover a entrada do dicionário:

```
del inventario['peras']
print (inventario)
```

```
{'laranjas': 525, 'abacaxis': 430, 'bananas': 312}
```

python™

Mais métodos de dicionários

```
ing2esp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

A função len também funciona com dicionários; retornando o número de pares chave-valor:

```
len(ing2esp)
```

3

O método keys retorna todas as chaves da relação chave-valor do dicionário.

```
ing2esp.keys()
```

['one', 'three', 'two']

O método values é parecido; retorna a lista de valores de um dicionário:

```
ing2esp.values()
```

['uno', 'tres', 'dos']

O método items

O método items retorna os dois, na forma de uma lista de tuplas - cada tupla com um par chave-valor:

```
ing2esp.items()
[('one','uno'), ('three','tres'), ('two','dos')]
```

- A sintaxe fornece uma informação util.

- Os colchetes indicam que isso é uma lista.

- Os parênteses indicam que os elementos da lista são tuplas.

Como percorrer um dicionário?

Você pode usá-lo em um loop for, desta forma:

```
dicionário = {‘c’: 2, ‘a’:0, ‘b’:1}
for chave, valor in dicionario.items():
...         print(chave, valor)
```

```
c 2
a 0
b 1
```

Em um dicionário, os itens não respeitam uma ordem. Somente são identificados pela chave correspondente!

Como percorrer um dicionário ordenado?

Você pode usá-lo em um loop for, desta forma:

```
dicionario = {‘c’: 2, ‘a’:0, ‘b’:1}
for chave in sorted(dicionario.keys()):
...         print(dicionario[chave])
```

```
a 0
b 1
c 2
```

Como percorrer um dicionário ordenado sem repetição?

Você pode usá-lo em um loop for, desta forma:

```
favorite_languages = { 'jen': 'python', 'sarah': 'c', 'edward': 'ruby', 'phil': 'python', }
print("The following languages have been mentioned:”)
for language in set(favorite_languages.values()):
    print(language.title())
```

The following languages have been mentioned:
Python
C
Ruby

Método get

o método get recebe uma chave e retorna o valor associado se a chave existe no dicionário, e nada (**None**) caso contrário:

```
 ing2esp.get('one')
'uno'
 ing2esp.get('deux')
```

Se você tentar chamar um método sem especificar em qual objeto, você obterá um erro. Nesse caso, a mensagem de erro não é muito útil:

```
 get('one')
NameError: get
```

Operador in

in é um operador lógico que testa se uma chave está no dicionário.

Nós o utilizamos com strings e com listas, mas ele também funciona com dicionários:

```
 'one' in ing2esp
True

 deux' in ing2esp
False
```

Ordenação de Dicionários

```python
def main():
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    print('original:', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=True))
    print('decrescente pelo nome(chave):', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=False))
    print('crescente pelo nome(chave):', dicionario)
```

```
original: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pelo nome(chave): {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
crescente pelo nome(chave): {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
```

Ordenação de Dicionários

```
def main():
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    print('original:', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=True))
    print('decrescente pelo nome(chave):', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=False))
    print('crescente pelo nome(chave):', dicionario)
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[0], reverse=True))
    print('decrescente pela nota:', dicionario)
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[0], reverse=False))
    print('crescente pela nota:', dicionario)
```

```
original: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pelo nome(chave): {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
crescente pelo nome(chave): {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pela nota: {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
crescente pela nota: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
```

Ordenação de Dicionários

```python
def main():
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    print('original:', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=True))
    print('decrescente pelo nome(chave):', dicionario)
    dicionario = dict(sorted(dicionario.items(), reverse=False))
    print('crescente pelo nome(chave):', dicionario)
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[0], reverse=True))
    print('decrescente pela nota:', dicionario)
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[0], reverse=False))
    print('crescente pela nota:', dicionario)
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[1], reverse=True))
    print('decrescente pela turma:', dicionario)
    dicionario = {'Joao':(10, 'Turma B'), 'Maria':(9, 'Turma A')}
    dicionario = dict(sorted(dicionario.items(), key=lambda item: item[1], reverse=False))
    print('crescente pela turma:', dicionario)
```

```
original: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pelo nome(chave): {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
crescente pelo nome(chave): {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pela nota: {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
crescente pela nota: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
decrescente pela turma: {'Joao': (10, 'Turma B'), 'Maria': (9, 'Turma A')}
crescente pela turma: {'Maria': (9, 'Turma A'), 'Joao': (10, 'Turma B')}
```

Uma Lista de Dicionários

O dicionário alien_0 contém várias informações sobre um alienígena, mas não há espaço para armazenar informações sobre um segundo alienígena, muito menos para uma tela cheia deles.

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}

aliens = [alien_0, alien_1, alien_2]

for alien in aliens:
....    print(alien)
```

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

python™

Exemplo

```
# Cria uma lista vazia para armazenar alienígenas
aliens = []
# Cria 30 alienígenas verdes
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
# Mostra os 5 primeiros alienígenas
for alien in aliens[:5]:
    print(alien)
    print("...")
# Mostra quantos alienígenas foram criados
print("Total number of aliens: " + str(len(aliens)))
```

```
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
...
Total number of aliens: 30
```

Uma Lista em um Dicionário

- Em vez de colocar um dicionário em uma lista, às vezes é conveniente colocar uma lista em um dicionário.

- Você pode aninhar uma lista em um dicionário sempre que quiser que mais de um valor seja associado a uma única chave em um dicionário.

```
# Armazena informações sobre uma pizza que está sendo pedida
pizza = { 'crust': 'thick', 'toppings': ['mushrooms', 'extra cheese'], }
# Resume o pedido
print("You ordered a " + pizza['crust'] + "-crust pizza " + "with the following toppings:")
for topping in pizza['toppings']:
    print("\t" + topping)
```

```
You ordered a thick-crust pizza with the following toppings:
    mushrooms    extra cheese
```

Exemplo

```
favorite_languages = {
    'jen': ['python', 'ruby'],
    'sarah': ['c'],
    'edward': ['ruby', 'go'],
    'phil': ['python', 'haskell'],
}

for name, languages in favorite_languages.items():
    print("\n" + name.title() + "'s favorite languages are:")
    for language in languages:
        print("\t" + language.title())
```

```
Jen's favorite languages are:
    Python
    Ruby

Sarah's favorite languages are:
    C

Phil's favorite languages are:
    Python
    Haskell

Edward's favorite languages are:
    Ruby
    Go
```

9.4 Números aleatórios

A maioria dos programas de computador fazem a mesma coisa sempre que são executados, então, podemos dizer que eles são determinísticos.

Entretanto, para algumas aplicações queremos que o computador se torne imprevisível. Jogos são um exemplo óbvio, mas existem outros.

Python tem uma função nativa que gera números pseudo-aleatórios, os quais não são verdadeiramente aleatórios no sentido matemático, mas para os nossos propósitos eles são.

9.4 Números aleatórios

O módulo random contém uma função chamada random que retorna um número em ponto flutuante entre 0.0 e 1.0.

Cada vez que você chama random, você recebe o próximo número de uma longa série. Para ver uma amostra, execute este loop:

```
import random

for i in range(10):
        x = random.random()
        print (x)
```

```
0.9578680543553147
0.9181290234534996
0.6985652866953983
0.9037050266648651
0.3542870511850169
0.3168419787845531
0.7858910795081855
0.7832026942491465
0.1927941313344389
0.24420096119197732
```

Para gerar um número aleatório ente 0.0 e um limite superior, multiplique x por superior.

python™

9.4 Números aleatórios

```
import random

for i in range(10):
        x = random.random()
        print (int(x * 80))
```

```
44
65
6
39
12
1
0
75
14
40
```

python™

Tarefa para Casa

1) Considere um sistema onde os dados são armazenados em dicionários. Nesse sistema existe um dicionário principal e o dicionário de backup. Cada vez que o dicionário principal atinge tamanho 5, ele imprime os dados na tela e apaga o seu conteúdo.

Crie um programa que insira dados em um dicionário, realizando o backup a cada dado e excluindo os dados do dicionário principal quando ele atingir tamanho 5.

Exercícios

2) Crie um dicionário que é uma agenda e coloque nele os seguintes dados: chave (cpf),nome, idade, telefone (podem ser uma tupla ou lista). O programa deve ler um número indeterminado de dados, criar a agenda e imprimir todos os itens do dicionário no formato: nome-idade-fone.

3) Crie um programa que cadastre informações de várias pessoas (nome, idade e cpf) e depois coloque em um dicionário. Depois remova todas as pessoas menores de 18 anos do dicionário e coloque em outro dicionário.

Exercícios

4) Escreva uma função chamada **mais_frequentes** que receba uma string e escreva na tela as letras em ordem decrescente de ocorrência. Veja a variação de ocorrência de letras em diferentes linguagens e compare com os seus resultados:

https://pt.wikipedia.org/wiki/Frequência_de_letras

Letra	Francês [8]	Alemão [9]	Espanhol [10]	Português [11]	Esperanto [12]	Italiano[13]	Turco	Sueco[14]	Polonês[15]	Toki Pona [16]	Holandês [17]
a	7.636%	6.51%	12.53%	14.63%	12.12%	11.74%	11.68%	9.3%	8.0%	17.2%	7.49%
b	0.901%	1.89%	1.42%	1.04%	0.98%	0.92%	2.95%	1.3%	1.3%	0.0%	1.58%
c	3.260%	3.06%	4.68%	3.88%	0.78%	4.5%	0.97%	1.3%	3.8%	0.0%	1.24%
d	3.669%	5.08%	5.86%	4.99%	3.04%	3.73%	4.87%	4.5%	3.0%	0.0%	5.93%
e	14.715%	17.40%	13.68%	12.57%	8.99%	11.79%	9.01%	9.9%	6.9%	7.4%	18.91%
f	1.066%	1.66%	0.69%	1.02%	1.03%	0.95%	0.44%	2.0%	0.1%	0.0%	0.81%
g	0.866%	3.01%	1.01%	1.30%	1.17%	1.64%	1.34%	3.3%	1.0%	0.0%	3.40%