Lógica de Programação Orientada a Objetos

Bem-vindos!

Objetivos

- Listas
- Lambda

LET'S CODE

LISTAS

- Uma coleção ordenada

- Cria uma lista vazia:

```
>>> l = list()
>>> type(l)
list
>>> l = []
>>> type(l)
list
```

MODIFICANDO

- Acrescentando valores na lista:

```
>>> l.append('apple')
>>> l
['apple']
>>> l.append('orange')
>>> l
['apple', 'orange']
```

- Criando uma lista populada:

```
>>> l = ["orange", "apple", "strawberry", "banana", "apricot"]
>>> l
['orange', 'apple', 'strawberry', 'banana', 'apricot']
```

- Adicionando em um índice específico:

```
>>> l.insert(3, 'melon')
>>> l
['orange', 'apple', 'strawberry', 'melon', 'banana', 'apricot', 'grapes']
```

- Removendo um item de um índice específico:

```
>>> del l[0]
>>> l
['apple', 'strawberry', 'melon', 'banana', 'apricot', 'grapes']
```

- Removendo um elemento (remove ele e retorna o seu valor):

```
>>> l.pop() # last element by default
'grapes'
>>> l
['apple', 'strawberry', 'melon', 'banana', 'apricot']
>>>l.pop(0) # by index
'apple'
>>> l
['strawberry', 'melon', 'banana', 'apricot']
```

- Todas as sequências de operações são válidas nas listas:

```
>>> l = ['orange', 'apple', 'strawberry', 'melon', 'banana', 'apricot', 'grapes']
>>> len(l)
7
```

- Loops for através das listas:

```
>>> for x in l:
...     print(x)
...
orange
apple
strawberry
banana
apricot
```

Recuperando elementos por índice, índice negativo e fatia:

```
>>> l[0]
'apple'
>>> l[8]
...
IndexError: list index out of range
>>> l[-1]
'grapes'
>>> l[2:4]
['strawberry', 'melon']
>>> [10,20,30,40][::-1]
[40, 30, 20, 10]
```

- O operador in escaneia por todos os elementos e retorna True ou False:

```
>>> 'apple' in l
True
>>> 'basketball' in l
False
```

- Listas podem ser concatenadas com + e multiplicadas por *:

```
>>> [25,3,14] + [5,4] + [101, 2]
[25, 3, 14, 5, 4, 101, 2]
>>> ["foo", "bar"] * 4
['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar']
```

- O método .reverse() altera a própria lista, e retorna None:

```
>>> fruit = ["orange", "apple", "strawberry", "banana", "apricot"]
>>> fruit.reverse()  # changes the list fruit!
>>> fruit
['apricot', 'banana', 'strawberry', 'apple', 'orange']
```

- O método .sort() organiza os próprios elementos, e retorna None:

```
>>> fruit = ["orange", "apple", "strawberry", "banana", "apricot"]
>>> fruit.sort()  # changes the list fruit!
>>> fruit
['apple', 'apricot', 'banana', 'orange', 'strawberry']
>>> fruit.sort(reverse=True)
>>> fruit
['strawberry', 'orange', 'banana', 'apricot', 'apple']
```

LISTAS E REFERÊNCIAS

- Enquanto estiver usando o operador de atribuição (=) em uma lista, a referência é criada para a lista original:

```
>>> l = [10,5,25,100,250,1,8]
>>> l
[10, 5, 25, 100, 250, 1, 8]
>>> l2 = l
>>> l2
[10, 5, 25, 100, 250, 1, 8]
>>> l.append("BOOM!")
>>> l
[10, 5, 25, 100, 250, 1, 8, 'BOOM!']
>>> l2
[10, 5, 25, 100, 250, 1, 8, 'BOOM!']
```

- Como demonstrado acima, l2 não é uma cópia de l, mas uma referência da mesma lista python na memória. Para criar uma cópia da lista, use o operador de fatia ([:]):

```
>>> l3 = l[:]
>>> l3.append(9876)
>>> l3
[10, 5, 25, 100, 250, 1, 8, 'BOOM!', 9876]
l
>>> [10, 5, 25, 100, 250, 1, 8, 'BOOM!']
```

- Tenha em mente é uma cópia rasa de l:

```
>>> numbers = [10,20,30]
>>> l = [numbers, "x", "y"]
>>> l
[[10, 20, 30], 'x', 'y']
>>> l2 = l[:]
>>> l2
[[10, 20, 30], 'x', 'y']
>>> l2.append("z")
>>> l2
[[10, 20, 30], 'x', 'y', 'z']
>>> l
[[10, 20, 30], 'x', 'y']
>>> # BUT:
>>> numbers.append(9999)
>>> l
[[10, 20, 30, 9999], 'x', 'y']
```

(copy.deepcopy deve ser usado para criar uma cópia real das listas)

Tuplas

- Imutável
- Tuplas são usados para agrupar dados ordenados
- Suporta sintaxe de índices e fatiamento
- Poderosa funcionalidade de atribuição (packing/unpacking)

CRIANDO UMA TUPLA

```
>>> t = tuple()
>>> type(t)
tuple

>>> t = (1, 2, 'hi')
>>> type(t)
tuple

>>> t = 1, 2, 'hi' # cuidado
>>> type(t)
tuple

>>> t = tuple([1,2,'yo'])
>>> t
(1, 2, 'yo')
```

PODEROSA ATRIBUIÇÃO

```
x, y = 'hi', 'man'
x, y = y, x
print(x, y)

# output
man hi
```

DICIONÁRIOS

- Um dicionário é um hash map:
  - Hash é uma chave para mapear valores
  - Chaves devem ser imutáveis para que o hash não mude
- dict() e {} são dicionários vazios.
- d[k] acessa o valor mapeado por k
- d[k] = v atualiza o valor mapeado por k

MÉTODOS

- len(), in, e del trabalha como listas
  d.keys() e d.values() Retorna uma lista correspondente das chaves e valores do dicionário.
- d.items() produz uma lista de tuplas (k,v)
- d.get(k,x) olha para o valor de k. Retorna x se k not
- in d
- d[k] = x cria uma chave ou altera um valor da respectiva chave.
- d.pop(k,x) retorna e remove o valor de k. Retorna x como padrão

ITERADORES BUILTINS

- len(x): Mostra número de elementos
- sum(x): Soma os elementos
- a in x: checa presença
- all(x)/any(x): retorna True quando toda/qualquer elemento retorna True

- max(x)/min(x): maior/menor elemento
- reversed(x): iterador de elementos em ordem reversa
- zip(x,x): lista de tuplas com um elemento de uma com outra lista
- sorted(x): retorna uma lista ordenada

LAMBDA

```
x = lambda a : a + 10
print(x(5))
```

```
x = lambda a, b : a * b
print(x(5, 6))
```

```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

LET'S CODE

Exercícios

- https://www.hackerrank.com/challenges/python-lists/problem
- https://www.hackerrank.com/challenges/python-tuples/problem
- https://www.hackerrank.com/challenges/defaultdict-tutorial/problem