Lógica de Programação Orientada a Objetos

Bem-vindos!

Objetivos

- Listas
- Tuplas
- Dicionários

LET'S CODE

Criando listas

- lista = [ ]
- lista = ['O carro', 'peixe', 123, 111]
- nova_lista = ['pedra', lista]

Operadores de lista

- Acessando os itens da lista:
  - lista[0]
  - nova_lista[1][2]
- Comprimento de uma lista:
  - len(nova_lista)
- Concatenação e multiplicação:
  - lista + nova_lista
  - lista * 3
- Verificando a existência de itens em uma lista:
  - 'peixe' in lista
  - 'gato' in lista
- Valores mínimos, máximos e soma:
  - numeros = [14.55, 67, 89.88, 10, 21.5]
    - min(numeros)
    - max(numeros)
    - sum(numeros)

Métodos das listas

- Adicionar itens no fim da lista:
  - livros = ['Java', 'SqlServer', 'Delphi', 'Python']
    - livros.append('Android')
- Adicionar itens em índice específico:
  - livros = ['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
    - livros.insert(0,'Oracle')
- Remover itens de uma lista:
  - livros = ['Oracle', 'Java', 'SqlServer', 'Delphi', 'Python', 'Android']
    - livros.pop() - Remove último item da lista;
    - livros.pop(1) - Remove item da lista pelo índice;
    - livros.remove('Oracle') - Remove item pelo valor.
  - Erros:
    Traceback (most recent call last):
    File "<pyshell#34>", line 1, in <module>
    livros.remove('Ruby')
    ValueError: list.remove(x): x not in list

Métodos das listas

- Ordenar de forma reversa em relação a ordem atual:
  - livros.reverse()
- Ordenar em ordem crescente:
  - livros.sort()
- Contar quantas vezes determinado item aparece na lista:
  - livros.count('Python')

Listas X Tuplas

De fato, tecnicamente as diferenças mais claras são essas - tuplas se comportam como listas estáticas. Assim, ainda podemos deduzir (e confirmar) que, por conta disso, tuplas ocupam menos espaço na memória comparadas a listas.

- Teste:
  - ( ).__sizeof__()
  - [ ].__sizeof__()

Criando tuplas nomeadas

- from collections import namedtuple

```
coordenadas = namedtuple('Coordenadas', ['latitude', 'longitude'])
nova_coordenada = coordenadas(latitude=-23.588254, longitude=-46.632477)
print(nova_coordenada)
print(nova_coordenada[0])
print(nova_coordenada.latitude)
print(nova_coordenada[1])
print(nova_coordenada.longitude)
```

Atribuindo Tuplas

- a, b = b, a
- a, b = 1, 2, 3
  ValueError: too many values to unpack
- addr = 'monty@python.org'
  uname, domain = addr.split('@')

Tuplas como valores de retorno

- t = divmod(7, 3)
- quot, rem = divmod(7, 3)

Dicionários

x = {1: "aaa", 2: "bbb", 3: "ccc"}

- Verificar tamanho do dicionário:
  - len(x)
- Remover item do dicionário pela chave:
  - del(x[1])

Retornando itens aleatórios

```
retornar_ligacao = {
    "Perícles": 30301122,
    "Menelau": 33547877,
    "Atreu": 33381245,
    "Tiestes": 36458899
}

retornar_ligacao.popitem()
('Menelau', 33547877)

retornar_ligacao.popitem()
('Perícles', 30301122)

retornar_ligacao.popitem()
('Atreu', 33381245)

retornar_ligacao.popitem()
('Pessoa xyz', -5254)
```

Erro ao retornar item aleatório de um dicionário vazio

```
>>> retornar_ligacao.popitem()
Traceback (most recent call last):
  File "<pyshell#166>", line 1, in <module>
    retornar_ligacao.popitem()
KeyError: 'popitem(): dictionary is empty'
```

Acesso a chave inexistente

- a = {"aaa": 111, "bbb": 222, "ccc": 333}
  - a["zzz"]
  - if a["zzz"]:
    print(a["zzz"])

Atualizando dicionários

```
a = {"aaa": 10, "bbb": 20, "ccc": 30}
b = {"ddd": 40, "eee": 50, "ddd": 60}
a.update(b)
print(a)
```

Iterando dicionários

```
dicionario = {"aaa": 10, "bbb": 20, "ccc": 30}

for e in dicionario:
        print(e)

for e in dicionario.values():
        print(e)

for i, e in enumerate(dicionario):
        print(i, e, dicionario[e])

for k, v in dicionario.items():
    print(k, " - ", v)
```

Dicionários para listas

- dicionario = {"aaa": 10, "bbb": 20, "ccc": 30}

```
print(list(dicionario))
print(list(dicionario.keys()))
print(list(dicionario.items()))
print(list(dicionario.values()))
```

Lista para dicionário

-
```
d1 = ['aaa', 'bbb', 'ccc']
d2 = ['1', '2', '3']
print(list(zip(d1, d2)))
print(dict(zip(d1, d2)))
```

EXERCÍCIO 1

Escreva uma função chamada most_frequent que receba 3 strings, uma palavra em português, outra com sua tradução para o inglês e a terceira com a tradução em espanhol e exiba as letras em ordem decrescente de frequência.

EXERCÍCIO 2

Escreva um programa que leia uma lista de palavras de um arquivo (pesquise por 10 palavras que são anagramas, ponha uma em cada linha e adicione outras 10 palavras quaisquer que não são anagramas) e imprima todos os conjuntos de palavras que são anagramas.

Aqui está um exemplo de como a saída pode parecer:

['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
['retainers', 'ternaries']
['generating', 'greatening']
['resmelts', 'smelters', 'termless']

Dica: você pode querer construir um dicionário que mapeie uma coleção de letras a uma lista de palavras que podem ser soletradas com essas letras. A pergunta é: como representar a coleção de letras de forma que possa ser usada como uma chave?

EXERCÍCIO 3

Altere o programa anterior para que exiba a lista mais longa de anagramas primeiro, seguido pela segunda mais longa, e assim por diante.

No Scrabble, um “bingo” é quando você joga todas as sete peças na sua estante, junto com uma peça no tabuleiro, para formar uma palavra de oito letras. Que coleção de oito letras forma o maior número possível de bingos? Dica: há sete.

EXERCÍCIO 4

Duas palavras formam um “par de metátese” se você puder transformar uma na outra trocando duas letras, por exemplo, “converse” e “conserve”. Escreva um programa que descubra todos os pares de metátese no dicionário. Dica: não teste todos os pares de palavras e não teste todas as trocas possíveis.