POO II
Revisão da Linguagem Python

Prof. Romuere Silva

Listas

- Uma lista é uma sequencia de valores onde cada valor é identificado por um índice iniciado por 0.
- São similares a *strings* (coleção de caracteres) exceto pelo fato de que os elementos de uma lista podem ser de qualquer tipo.
- A sintaxe é simples, listas são delimitadas por colchetes e seus elementos separados por virgula:

```
>>> lista1 = [1, 2, 3, 4]
>>> lista1
[1, 2, 3, 4]
>>>
>>>lista2 = ['python', 'java', 'c#']
>>> lista2
['python', 'java', 'c#']
```

```
>>> lista = [1, 2, 'python', 3.5, 'java']
>>> lista
[1, 2, 'python', 3.5, 'java']
```

Listas

- Também é possível usar a função list() para criar uma lista passando um tipo que pode ser iterável como uma *string*:

```
>>> lista = list('python')
>>> lista
['p', 'y', 't', 'h', 'o', 'n']
```

```
    meses = ['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Junho', 'Julho', 'Agosto', 'Setembro', 'Outub
ro', 'Novembro', 'Dezembro']

    n = 1

    while(n < 4):
        mes = input("Escolha um mês (1-12): ")
        if 1 <= mes <= 12:
            print('O mês é {}'.format(mes[mes-1]))
        n += 1
```

Listas

- As listas também possuem funcionalidades prontas e podemos manipulá-las através de funções embutidas.
- A lista tem uma função chamada *append()* que adiciona um dado na lista:

```
>>> lista = []
>>> lista.append('zero')
>>> lista.append('um')
>>> lista
['zero', 'um']
```

```
>>> lista = ['zero', 'um']
>>> lista.extend(['dois', 'três',])
>>> lista += ['quatro', 'cinco']
>>> lista + ['seis']
['zero', 'um', 'dois', 'três', 'quatro', 'cinco', 'seis']
>>> lista * 2
['zero', 'um', 'dois', 'três', 'quatro', 'cinco', 'seis', 'zero', 'um', 'dois', 'três', 'quatro',
'cinco', 'seis']
```

Tuplas

- Uma tupla é uma lista **imutável**, ou seja, uma tupla é uma sequência que não pode ser alterada depois de criada.
- Uma tupla é definida de forma parecida com uma lista com a diferença do delimitador.
- Enquanto listas utilizam colchetes como delimitadores, as tuplas usam parênteses:

```
>>> dias = ('domingo', 'segunda', 'terça', 'quarta', 'quinta', 'sexta', 'sabado')
>>> type(dias)
<class 'tuple'>
```

```
>>> texto = 'python'
>>> tuple(texto)
('p', 'y', 't', 'h', 'o', 'n')
>>> lista = [1, 2, 3, 4]
>>> tuple(lista)
(1, 2, 3, 4)
```

Tuplas

- As regras para os índices são as mesmas das listas, exceto para elementos também imutáveis.
- Como são imutáveis, uma vez criadas não podemos adicionar nem remover elementos de uma tupla.
- O método append() da lista não existe na tupla:

```
>>> dias.append('sabado2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'append'
>>>
>>> dias[0]
'domingo'
>>>
>>> dias[0] = 'dom'
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuplas

- Não é possível atribuir valores aos itens individuais de uma tupla, no entanto, é possível criar tuplas que contenham objetos mutáveis, como listas.

```
>>> lista = [3, 4]
>>> tupla = (1, 2, lista)
>>> tupla
(1, 2, [3, 4])
>>> lista = [4, 4]
>>> tupla
(1, 2, [4, 4])
>>> tupla[2] = [3, 4]
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

CONJUNTOS

- O Python também inclui um tipo de dados para conjuntos. Um conjunto, diferente de uma sequência, é uma coleção **não ordenada** e que **não admite elementos duplicados**.
- Chaves ou a função set() podem ser usados para criar conjuntos.

```
>>> frutas = {'laranja', 'banana', 'uva', 'pera', 'laranja', 'uva', 'abacate'}
>>> frutas
>>> {'uva', 'abacate', 'pera', 'banana', 'laranja'}
```

Conjuntos

- Operações:

```
>>> a =  set('abacate')
>>> b = set('abacaxi')
>>> a
{'a', 'e', 'c', 't', 'b'}
>>> b
{'a', 'x', 'i', 'c', 'b'}
>>> a - b                              # diferença
{'e', 't'}
>>> a | b                              # união
{'c', 'b', 'i', 't', 'x', 'e', 'a'}
>>> a & b                              # interseção
{'a', 'c', 'b'}
>>> a ^ b                              # diferença simétrica
{'i', 't', 'x', 'e'}
```

Dicionários

- Qualquer chave de um dicionário é associada (ou mapeada) a um valor.
- Os valores podem ser qualquer tipo de dado do Python.
- Portanto, os dicionários são pares de chave-valor não ordenados.

```
>>> pessoa = {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'}
>>> pessoa
'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'
```

Dicionários

- Não é possível acessar um elemento de um dicionário por um índice como na lista.
- Devemos acessar por sua chave:

```
>>> pessoa['nome']
'João'
>>> pessoa['idade']
25
```

Dicionários

- Como sempre acessamos seus elementos através de chaves, o dicionário possui um método chamado *keys()* que devolve o conjunto de suas chaves:

```
>>> pessoa1.keys()
dict_keys(['nome', 'idade', 'cidade', 'pais'])
```

- Assim como um método chamado values() que retorna seus valores:

```
>>> pessoa1.values()
dict_values(['João', 25, 'São Paulo', 'Brasil'])
```

Dicionários

- Também podemos criar dicionários utilizando a função **dict()**:

```
>>> a = dict(um=1, dois=2, três=3)
>>> a
{'três': 3, 'dois': 2, 'um': 1}
```

Exercícios

- Dada a lista=[12,-2,4,8,29,45,78,36,-17,2,12,8,3,3,-52] faça um programa que:
  - imprima o maior elemento
  - imprima o menor elemento
  - imprima os números pares
  - imprima o número de ocorrências do primeiro elemento da lista
  - imprima a média dos elementos
  - imprima a soma dos elementos de valor negativo

Exercícios

- Faça um programa utilizando um dict que leia dados de entrada do usuário. O usuário deve entrar com os dados de uma pessoa como nome, idade e cidade onde mora. Após isso, você deve imprimir os dados como o exemplo abaixo:

```
nome: João
idade: 20
cidade: São Paulo
```

- Utilize o exercício anterior e adicione a pessoa em uma lista. Pergunte ao usuário se ele deseja adicionar uma nova pessoa. Após adicionar dados de algumas pessoas, você deve imprimir todos os dados de cada pessoa de forma organizada.

Exercícios

```
Comparativo de Consumo de Combustível

Veículo 1
Nome: fusca
Km por litro: 7
Veículo 2
Nome: gol
Km por litro: 10
Veículo 3
Nome: uno
Km por litro: 12.5
Veículo 4
Nome: Vectra
Km por litro: 9
Veículo 5
Nome: Peugeout
Km por litro: 14.5

Relatório Final
 1 - fusca           -    7.0 -  142.9 litros - R$ 321.43
 2 - gol             -   10.0 -  100.0 litros - R$ 225.00
 3 - uno             -   12.5 -   80.0 litros - R$ 180.00
 4 - vectra          -    9.0 -  111.1 litros - R$ 250.00
 5 - peugeout        -   14.5 -   69.0 litros - R$ 155.17
O menor consumo é do peugeout
```

- Faça um programa que carregue uma lista com os modelos de cinco carros (exemplo de modelos: FUSCA, GOL, VECTRA etc). Carregue uma outra lista com o consumo desses carros, isto é, quantos quilômetros cada um desses carros faz com um litro de combustível. Calcule e mostre:
  - O modelo do carro mais econômico;
  - Quantos litros de combustível cada um dos carros cadastrados consome para percorrer uma distância de 1000 quilômetros e quanto isto custará, considerando um que a gasolina custe R$ 2,25 o litro. Abaixo segue uma tela de exemplo. O disposição das informações deve ser o mais próxima possível ao exemplo. Os dados são fictícios e podem mudar a cada execução do programa.

Exercícios

- Faça um programa que simule um lançamento de dados. Lance o dado 100 vezes e armazene os resultados em um vetor. Depois, mostre quantas vezes cada valor foi conseguido. Dica: use um vetor de contadores(1-6) e uma função para gerar números aleatórios, simulando os lançamentos dos dados.

Funções

```
def velocidade(espaco, tempo):
    v = espaco/tempo
    print('velocidade: {} m/s'.format(v))
```

```
def dados(nome, idade=None):
    print('nome: {}'.format(nome))
    if(idade is not None):
        print('idade: {}'.format(idade))
    else:
        print('idade: não informada')
```

```
def dados(nome, idade=None):
    if(idade is not None):
        return ('nome: {} \nidade: {}'.format(nome, idade))
    else:
        return ('nome: {} \nidade: não informada'.format(nome))
```

```
def calculadora(x, y):
    return x+y, x-y
```

NÚMERO ARBITRÁRIO DE PARÂMETROS (*ARGS)

```
def teste(arg, *args):
    print('primeiro argumento normal: {}'.format(arg))
    for arg in args:
        print('outro argumento: {}'.format(arg))

teste('python', 'é', 'muito', 'legal')
```

Saída

```
primeiro argumento normal:  python
outro argumento:  é
outro argumento:  muito
outro argumento:  legal
```

NÚMERO ARBITRÁRIO DE CHAVES (**KWARGS)

```
def minha_funcao(**kwargs):
    for key, value in kwargs.items():
        print('{0} = {1}'.format(key, value))

>>> minha_funcao(nome='caelum')
nome = caelum
```

```
dicionario = {'nome': 'joao', 'idade': 25}
minha_funcao(**dicionario)
idade = 25
nome = joao
```