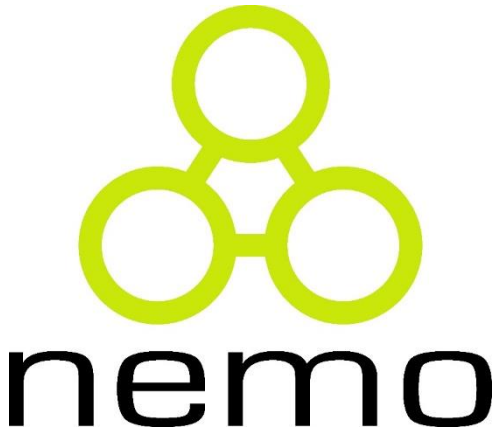

Programação I

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

LISTAS

REVISÃO

- Vimos que as tuplas são agrupamentos de tamanhos pré-definidos e heterogêneos.

- Em contrapartida, listas são tipos compostos para agrupar quantidades indefinidas de elementos de um mesmo tipo.

- Uma lista é uma sequência de zero ou mais elementos de um mesmo tipo. Entende-se por sequência uma quantidade qualquer de itens dispostos linearmente. Podemos representar uma lista pela enumeração dos seus elementos, separados por vírgulas e cercados por colchetes.

[*e1*, *e2*, ..., *en*]

nemo

Strings

- Em Python, strings podem ser tratadas como listas, possuindo algumas operações básicas semelhantes às operações de listas.

- Um caracter de uma string pode ser acessado através de sua posição, tal qual fazemos com elementos de uma lista.

- Exemplo:

```
>>> nome = "jordana"
>>> nome[4]
'a'
```

nemo

Strings

- Mas não podemos alterar seu valor atribuindo um valor através do índice.

- Exemplo:

```
>>> nome[4] = 'x'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    nome[4] = 'x'
TypeError: 'str' object does not support item assignment
```

nemo

Strings

- String é um objeto iterável.

- Assim, podemos percorrê-la da mesma maneira que fazemos com listas.

- Exemplo:

```
>>> for letra in nome:
        print(letra)
```

```
j
o
r
d
a
n
a
```

nemo

Strings

- Podemos fatiar a string.

- Assim, podemos obter substring da mesma maneira que obtemos sublistas.

- Exemplo:

```
>>> nome[1:5]
'orda'
>>> nome[0:6]
'jordan'
>>> nome[-2]
'n'
>>> nome[:]
'jordana'
```

nemo

Operações básicas

- Concatenamos strings com o sinal +, assim como concatenamos listas e elementos a listas.

- Exemplo:

```
>>> "jordana" + " ou " + "morgana?"
'jordana ou morgana?'
```

- Quando concatenamos com um número, precisamos fazer a conversão do número.

- Exemplo:

```
>>> "O valor de pi eh: " + str(3.14)
'O valor de pi eh: 3.14'
```

Operações básicas

```
>>> "O valor de pi eh: " + 3.14
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    "O valor de pi eh: " + 3.14
TypeError: must be str, not float
```

- Escapamos (scape) caracteres com o sinal \
- Exemplo:

```
>>> 'There\'s a snake in my boot!'
"There's a snake in my boot!"
```

nemo

Operações básicas

- **in** : avalia se um determinado caracter é parte de uma string.

```
"b" in "abc"      # True
"d" in "abc"      # False
"d" not in "abc" # True
"b" not in "abc" # False
```

nemo

Métodos comuns

- **Método len()**
- Mostra o tamanho da string

```
>>> len(nome)
7
```

- **Método str()**
- Converte um valor para o tipo string.

```
num = 123
type(num)      # <class 'int'>
type(str(num)) # <class 'str'>
```

nemo

Métodos comuns

- **Método lower()**
- Mostra a string em caixa baixa.

```
>>> nome
'JORDANA'
>>> nome.lower()
'jordana'
```

- **Método upper()**
- Mostra a string em caixa alta.

```
>>> nome.upper()
'JORDANA'
```

Métodos comuns

- **Método isalpha()**
- **Retorna False se a string contiver algum caracter que não seja letra**

```
"abc".isalpha()   # True
"1fg".isalpha()   # False
"123".isalpha()   # False
"/+)".isalpha()   # False
```

- **Método strip()**
- Retira espaços em branco no começo e no fim

```
" sobrando espaços ".strip()      # 'sobrando espaços'
"  sobrando espaços   ".strip()   # 'sobrando espaços'
```

nemo

Métodos comuns

- **Método join()**
- **Junta cada item da string com um delimitador especificado.**
- **A operação join aceita lista como parâmetro.**

```
>>> ".".join(nome)
'J.O.R.D.A.N.A'
```

- **Método split()**
- Separa uma string conforme um delimitador.
- É o inverso do join.

```
>>> nome.split("A")
['JORD', 'N', '']
```

nemo

Interpolação de strings

- Podemos utilizar códigos para formatar uma string, assim como fazemos em C.

Símbolos usados na interpolação:

- %s: *string*.
- %d: inteiro.
- %o: octal.
- %x: hexacimal.
- %f: real.
- %e: real exponencial.
- %%: sinal de percentagem.

Interpolação de strings

- Exemplos:

```
# -*- coding: latin1 -*-

# Zeros a esquerda
print 'Agora são %02d:%02d.' % (16, 30)

# Real (número após o ponto controla as casas decimais)
print 'Percentagem: %.1f%%, Exponencial:%.2e' % (5.333, 0.00314)

# Octal e hexadecimal
print 'Decimal: %d, Octal: %o, Hexadecimal: %x' % (10, 10, 10)
```

Saída:

```
Agora são 16:30.
Percentagem: 5.3%, Exponencial:3.14e-03
Decimal: 10, Octal: 12, Hexadecimal: a
```

nemo

Interpolação de strings

- A partir da versão 2.6, está disponível outra forma de interpolação além do operador "%", o método de string e a função chamados format().
- Exemplo:

```
# Parâmetros identificados pelo nome
msg = '{saudacao}, são {hora:02d}:{minuto:02d}'

print msg.format(saudacao='Bom dia', hora=7, minuto=30)
```

- Saída:

```
Bom dia, são 07:30
```

nemo

Interpolação de strings

- O Python 3.6 introduziu uma maneira mais simples de formatar strings: literais de string formatados, geralmente chamados de f-strings.

- Exemplo:

```
ram, region = 4, 'us-east'
f'This Linode has {ram}GB of RAM, and is located in the {region} region.'
```

```
'This Linode has 4GB of RAM, and is located in the us-east region.'
```

nemo

Dicionários

- Dicionários são um coleção desordenada de objetos representados na forma de (chave, valor) onde a chave é usada para referenciar um determinado valor.

- As chaves de um dicionário só podem ser de um tipo imutável como inteiros, floats e strings. Tuplas também podem ser aceitas desde que não contenham direta ou indiretamente um tipo mutável como listas.

- Dicionários não possuem uma noção de índice e não podem ser fatiados.

- Dicionários são mutáveis de forma que a qualquer momento você pode inserir ou remover itens.

Dicionários

- Para se criar um dicionário vazio, utilizamos:

```
>>> dic = {}
```

- Para criar um dicionário com alguns valores conhecidos, há duas maneiras:
- Dic = {chave1:valor1, chave2:valor2,....., $chave_n$:$valor_n$}
- Exemplo:

```
>>> dic = {'Julio': 'C', 'Jaime': 'Python', 'Ana': 'Ruby', 'Claudia':
'Java', 'Mauro': 'PHP'}
```

nemo

Dicionários

- Outra forma de gerar o mesmo dicionário acima é com dict(x) onde x pode ser uma lista de tuplas do tipo (chave, valor) como mostrado abaixo:

```
>>> di = dict([('Julio', 'C'), ('Jaime', 'Python'), ('Ana', 'Ruby'),
('Claudia', 'Java'), ('Mauro', 'PHP')])
```

Dicionários

- Para acessar elementos de um dicionário, existem duas formas:

- 1) Para acessar um item do dicionário devemos usar sua chave entre colchetes [ ]:

```
>>> dic['Ana']
'Ruby'
```

- 2) Para recuperar um valor no dicionário podemos usar o método get passando como argumento a chave do valor que queremos recuperar:

```
>>> dic.get('Ana')
'Ruby'
```

Dicionários

- Para verificar se uma chave existe no dicionário, utilizamos o operador in:

```
>>> def existeChave(chave, dicionario):
        return chave in dicionario

>>> existeChave('Ana', dic)
True
>>> existeChave('Pedro', dic)
False
```

nemo

Dicionários

- Para verificar se um valor existe no dicionário, utilizamos o operador in:

```
>>> def existeValor(valor, dicionario):
        return valor in dicionario.values()

>>> existeValor('C', dic)
True
>>> existeValor('C++', dic)
False
```

nemo

Dicionários

- Para alterar um valor em um dicionário, usamos o nome do dicionário com a chave entre colchetes e atribuímos o novo valor:

```
>>> di = {'Julio': 'C', 'Jaime': 'Python', 'Ana': 'Ruby', 'Cláudia': 'Java', 'Mauro': 'PHP'}
>>> di['Jaime'] = 'PHP'
>>> di
{'Cláudia': 'Java', 'Jaime': 'PHP', 'Julio': 'C', 'Mauro': 'PHP', 'Ana': 'Ruby'}
```

Dicionários

- Para inserir um item em um dicionário basta declarar o dicionário colocando entre colchetes a nova chave e atribuindo uma valor a ela:

```
>>> di = {'Portugal': 'Lisboa', 'Espanha': 'Madri', 'Itália': 'Roma'}
>>> print(di)
{'Espanha': 'Madri', 'Portugal': 'Lisboa', 'Itália': 'Roma'}
>>> di['França'] = 'Paris'
>>> print(di)
{'Espanha': 'Madri', 'Portugal': 'Lisboa', 'Itália': 'Roma', 'França': 'Paris'}
```

nemo

Dicionários

- Para remover um item em um dicionário basta usar a função del aplicada ao nome do dicionário com a chave entre colchetes:

```
>>> di = {'Claudia': 'Java', 'Jaime': 'Python', 'Julio': 'C', 'Mauro': 'PHP', 'Ana': 'Ruby'}
>>> del di['Mauro']
>>> di
{'Claudia': 'Java', 'Jaime': 'Python', 'Julio': 'C', 'Ana': 'Ruby'}
```

Dicionários

- Para obter o número de elementos em um dicionário, basta utilizarmos a função len:

```
>>> len(dic)
5
```

- Para mesclar dois dicionários, basta utilizar o método update:

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

nemo

That’s all Folks!