MAP 2112 – Introdução à Lógica de Programação e Modelagem Computacional

1º Semestre - 2023

Prof. Dr. Luis Carlos de Castro Santos

lsantos@ime.usp.br

Chapter 9 Tuples

9.1 Mutability and tuples ("Mutabilidade e Tuples")

Vimos dois tipos de dados compostos, strings, e listas. O tipo tuple é semelhante a uma lista mas é imutável.

```
In [9]: my_tuple = ("a","b","c","d","e")

In [10]: type(my_tuple)
Out[10]: tuple
```

A convenção de criação de tuple é o uso de parênteses.

```
In [11]: t1 =("a")

In [12]: type(t1)
Out[12]: str

In [13]: t1 =("a",)

In [14]: type(t1)
Out[14]: tuple
```

Uma tuple de 1 elemento para ser criada necessita de uma vírgula, caso contrário o objeto assume o tipo do contéudo (o mesmo vale para int e float).

O tratamento de tuples é similar ao de listas:

```
In [19]: my_tuple = ("a","b","c","d","e")

In [20]: my_tuple[0]
Out[20]: 'a'

In [21]: my_tuple[1:3]
Out[21]: ('b', 'c')

In [22]: my_tuple[0] = "A"
Traceback (most recent call last):

  Cell In[22], line 1
    my_tuple[0] = "A"

TypeError: 'tuple' object does not support item assignment


In [23]: my_tuple = ("A",) + my_tuple[1:]

In [24]: my_tuple
Out[24]: ('A', 'b', 'c', 'd', 'e')


In [29]: len(my_tuple)
Out[29]: 5
```

Índice começa em 0.

Fatia usando índices.

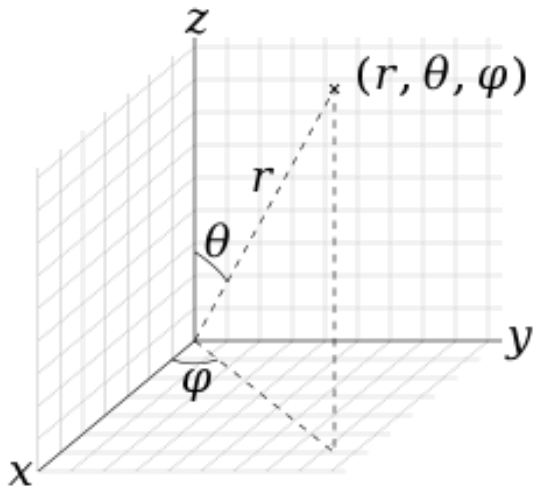
Tuples não aceitam atribuição: são imutáveis.

Tuples podem ser modificadas usando fatias e sobrescrevendo.

O comando len tem uso idêntico a strings e listas.

Tuples podem ser usadas como saídas de funções.

Exemplo: Coordenadas Cartesianas a partir de Coordenadas Esféricas




```python
import math

def cartesian(r,theta,phi):
    x = r*math.sin(theta)*math.cos(phi)
    y = r*math.sin(theta)*math.sin(phi)
    z = r*math.cos(theta)
    return x,y,z

r = 1
theta = math.pi/3
phi = math.pi/6

cart_coord = tuple()

cart_coord = cartesian(r, theta, phi)

print(cart_coord)
```

A saída da função é definida como tuple

O comando tuple cria uma tuple, nesse caso vazia

```
In [26]: runfile('C:/Users/win/Dropbox/USP/2023/rem
scripts/aula7_tp2.py', wdir='C:/Users/win/Dropbox/U
MAP2112_2023/scripts')
(0.75, 0.43301270189221924, 0.5000000000000001)
```

Chapter 10 Dictionaries

Dicionários é um tipo composto que aceita como índices não apenas valores inteiros. Como o próprio nome sugere, dicionários conectam índices a elementos, sendo os índices imutáveis.

Imagine um dicionário inglês espanhol, como exemplo:

```
eng2sp = {}

eng2sp["one"] = "uno"
eng2sp["two"] = "dos"
eng2sp["three"] = "tres"
eng2sp["fourteen"] = "catorce"


print(eng2sp)
```

Chaves criam um dicionário vazio

Os índices que são strings são associados a outros strings, nesse exemplo.

Os elementos são adicionados e o dicionário vai aumentando.

```
In [30]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/aula7_dic1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
{'one': 'uno', 'two': 'dos', 'three': 'tres', 'fourteen': 'catorce'}
```

Os índices do dicionário são chamados de chaves (“keys”)

10.1 Dictionary operations ("Operações em Dicionários")

```
In [31]: inventory = {"apples": 430, "bananas": 312, "oranges": 525,"pears": 217}

In [32]: inventory
Out[32]: {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}

In [33]: del inventory["pears"]

In [34]: inventory
Out[34]: {'apples': 430, 'bananas': 312, 'oranges': 525}

In [35]: inventory ["pears"] = 20

In [36]: inventory
Out[36]: {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 20}

In [37]: len(inventory)
Out[37]: 4
```

O dicionário pode ser criado com chaves e elementos associados

O conteúdo pode ser deletado através da chave

O conteúdo pode ser re-incluído simultaneamente com a chave

O comando len também pode ser aplicado a dicionários

De forma análoga a strings e listas existem os métodos para dicionários, alguns deles estão indicados na “folha-de-cola” (cheat sheet)

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

O uso é aplicado especificamente no objeto “dict”, veremos alguns exemplos em seguida.

Existe uma lista mais completa de métodos para listas disponível em :
https://www.w3schools.com/python/python_dictionaries_methods.asp

```
In [41]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [42]: x = car.keys()

In [43]: x
Out[43]: dict_keys(['brand', 'model', 'year'])

In [44]: type(x)
Out[44]: dict_keys

In [45]: x[0]
Traceback (most recent call last):

  Cell In[45], line 1
    x[0]

TypeError: 'dict_keys' object is not subscriptable


In [46]: klist = list(x)

In [47]: klist
Out[47]: ['brand', 'model', 'year']
```

```
In [48]: klist[0]
Out[48]: 'brand'
```

Extrai as chaves do dicionário

O objeto criado é do tipo dict_keys que não pode ser indexado

Para acessar as chaves por índices é necessário transformar em lista

```
In [49]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [50]: x = car.values()

In [51]: x
Out[51]: dict_values(['Ford', 'Mustang', 1964])

In [52]: x[0]
Traceback (most recent call last):

  Cell In[52], line 1
    x[0]

TypeError: 'dict_values' object is not subscriptable


In [53]: vlist = list(x)

In [54]: vlist[0]
Out[54]: 'Ford'
```

Extrai os valores do dicionário

O objeto criado é do tipo dict_values que não pode ser indexado

Para acessar os valores por índices é necessário transformar em lista

```
In [71]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [72]: x = car.items()

In [73]: x
Out[73]: dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

In [74]: x[0]
Traceback (most recent call last):

  Cell In[74], line 1
    x[0]

TypeError: 'dict_items' object is not subscriptable


In [75]: v = list(x)

In [76]: v[0]
Out[76]: ('brand', 'Ford')

In [77]: type(v[0])
Out[77]: tuple
```

Extrai os itens (chaves, valores) do dicionário

O objeto criado é do tipo dict_itens que não pode ser indexado

Para acessar os valores por índices é necessário transformar em lista

Os elementos da lista são tuples

```
In [78]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [79]: x = car.get("model")

In [80]: x
Out[80]: 'Mustang'
```

Extrai o valor associado a chave

```
In [81]: car.pop("model")
Out[81]: 'Mustang'

In [82]: car
Out[82]: {'brand': 'Ford', 'year': 1964}
```

Remove chave e valor associado

```
In [83]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [84]: car.update({"color": "White"})

In [85]: car
Out[85]: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}

In [86]: car.clear()

In [87]: car
Out[87]: {}

In [88]: car = {"brand": "Ford", "model": "Mustang", "year": 1964}

In [89]: x = car.copy()

In [90]: x
Out[90]: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Inclui chave e valor associado

Apaga todo o conteúdo (chaves e valores)

Copia todo o conteúdo (chaves e valores)

Os métodos de cópia são a melhor prática para evitar a associação indesejada (aliasing)

Chapter 11 Files

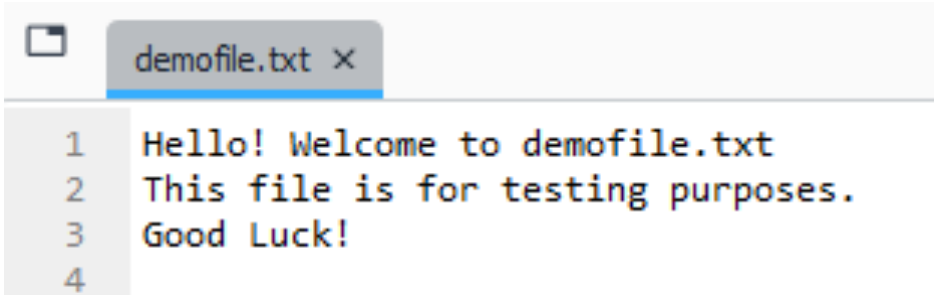
Manipulação de Arquivos

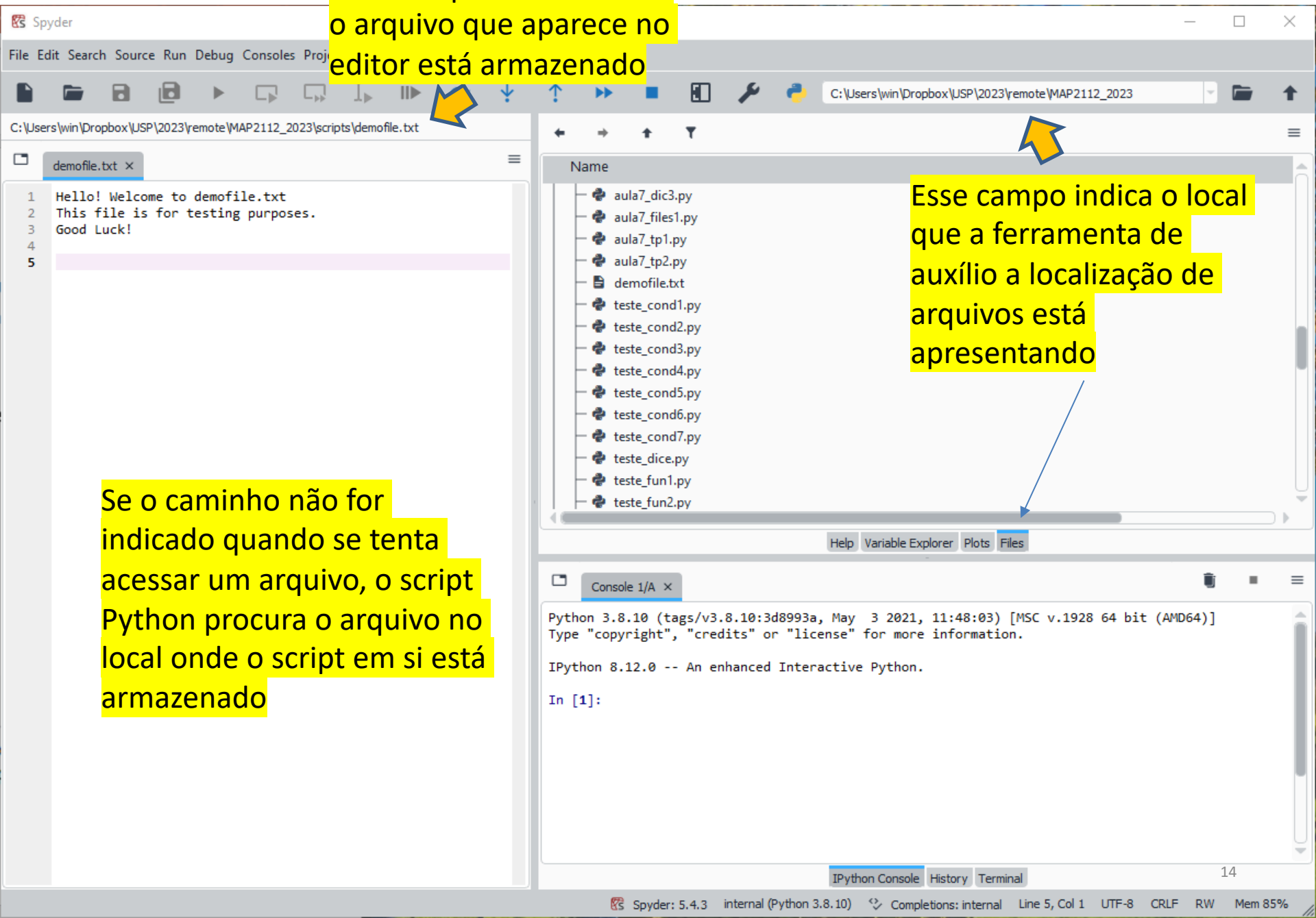
Ao invés de seguir a referência (ThinkCSpy) , o material relativo a manipulação de arquivos será adaptado de:
https://www.w3schools.com/python/python_file_handling.asp

Os arquivos são meios de armazenamento de informação que devem ser lidos, processados e escritos. Dessa forma existem diversos comandos que auxiliam nessa manipulação.

O uso de IDE´s, como o Spyder, facilitam a localização e manipulação dos arquivos numa mesma interface.

Criando um arquivo de exemplo, com o conteúdo:




Esse campo indica o local o arquivo que aparece no editor está armazenado
Spyder
File Edit Search Source Run Debug Consoles Proj
C:\Users\win\Dropbox\USP\2023\remote\MAP2112_2023
C:\Users\win\Dropbox\USP\2023\remote\MAP2112_2023\scripts\demofile.txt
demofile.txt
1 Hello! Welcome to demofile.txt
2 This file is for testing purposes.
3 Good Luck!
4
5
Name
aula7_dic3.py
aula7_files1.py
aula7_tp1.py
aula7_tp2.py
demofile.txt
teste_cond1.py
teste_cond2.py
teste_cond3.py
teste_cond4.py
teste_cond5.py
teste_cond6.py
teste_cond7.py
teste_dice.py
teste_fun1.py
teste_fun2.py
Esse campo indica o local que a ferramenta de auxílio a localização de arquivos está apresentando
Help Variable Explorer Plots Files
Console 1/A
Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 8.12.0 -- An enhanced Interactive Python.
In [1]:
IPython Console History Terminal
Se o caminho não for indicado quando se tenta acessar um arquivo, o script Python procura o arquivo no local onde o script em si está armazenado
Spyder: 5.4.3 internal (Python 3.8.10) Completions: internal Line 5, Col 1 UTF-8 CRLF RW Mem 85%

Manipulação de Arquivos

O comando que dá acesso ao script Python ao conteúdo do arquivo é o *open()*

Os parâmetros de entrada do comando *open()* são o nome do arquivo e a opção (*mode*). As opções (*modes*) são as seguintes:

"r" - Read – Valor default. Abre o arquivo para leitura, dá erro se o arquivo não existe.

"a" - Append – Abre o arquivo para adicionar conteúdo a partir do seu fim, cria o arquivo se ele não existe.

"w" - Write – Abre o arquivo para escrita, cria o arquivo se ele não existe.

"x" - Create - Cria o arquivo, dá erro se o arquivo já existe.

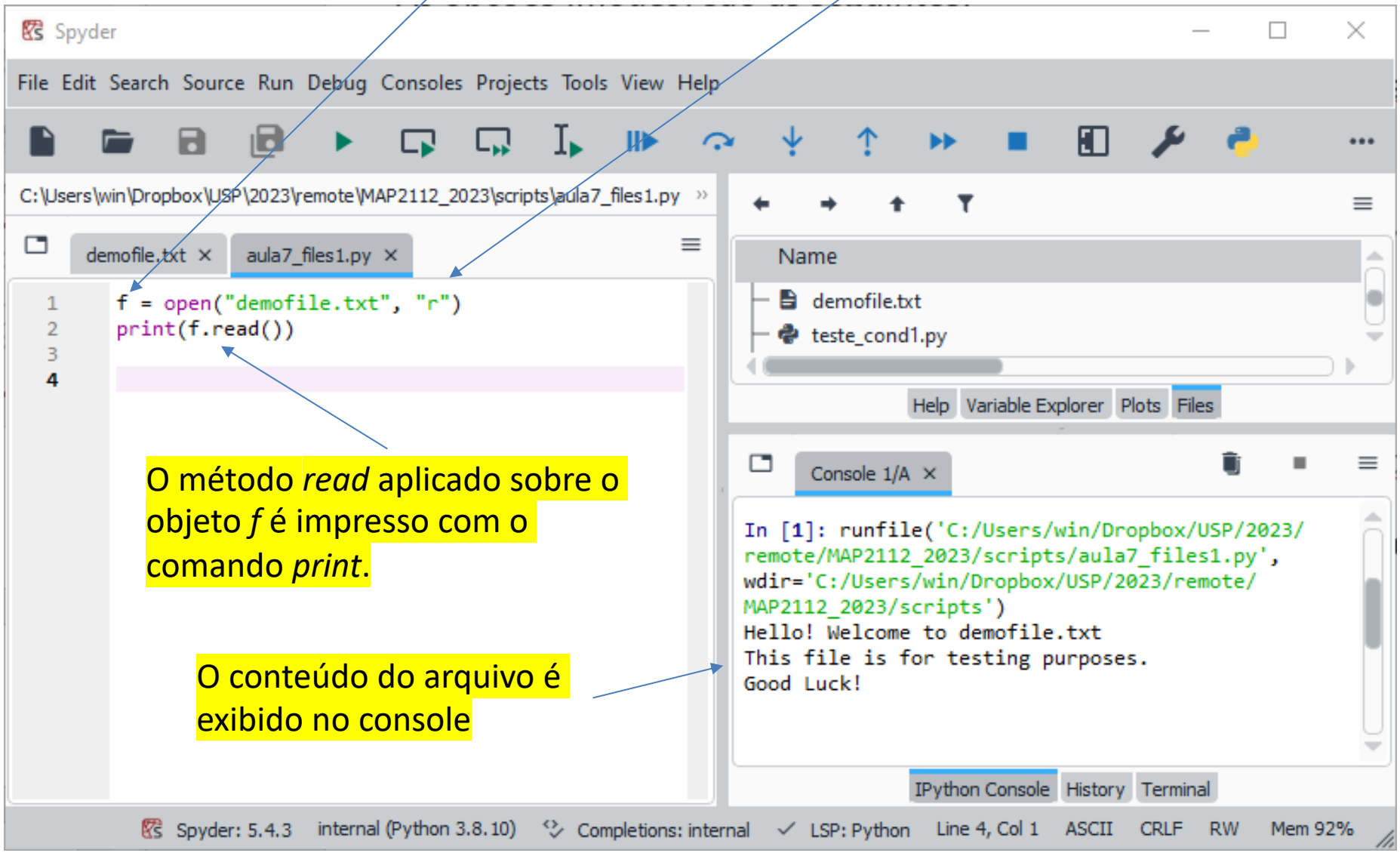
Em adição a essas opções pode-se especificar o tipo de conteúdo:

"t" - Text - Valor default. Modo texto

"*b*" – Binary – Modo binário (ex. imagens)

O comando open associa o arquivo ao objeto “f)

O arquivo é aberto no modo leitura



O método *read* aplicado sobre o objeto *f* é impresso com o comando *print*.

O conteúdo do arquivo é exibido no console

A opção do método *file.read* é a quantidade de caracteres a serem lidos. O default é o arquivo todo.

```
f = open("demofile.txt", "r")
print(f.read(15))
```

```
In [3]: runfile('C:/Users/win/Dropbo
remote/MAP2112_2023/scripts/aula7_fi
wdir='C:/Users/win/Dropbox/USP/2023/
MAP2112_2023/scripts')
Hello! Welcome
```

As linhas podem ser lidas usando o método file.readline. O default é apenas 1 linha.

```
f = open("demofile.txt", "r")
print(f.readline())
```

```
In [4]: runfile('C:/Users/win/Dropbox/US
remote/MAP2112_2023/scripts/aula7_files2
wdir='C:/Users/win/Dropbox/USP/2023/remo
MAP2112_2023/scripts')
Hello! Welcome to demofile.txt
```

A repetição do método *file.readline* continua acessando as linhas em sequência.

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

```
In [6]: runfile('C:/Users/win/Dropbox/USP
remote/MAP2112_2023/scripts/aula7_files2.
wdir='C:/Users/win/Dropbox/USP/2023/remot
MAP2112_2023/scripts')
Hello! Welcome to demofile.txt

This file is for testing purposes.
```

A opção do método *readline* apresenta a quantidade de caracteres da linha corrente.

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline(10))
```

```
In [7]: runfile('C:/Users/win/Dropbox/USP/
remote/MAP2112_2023/scripts/aula7_files2.p
wdir='C:/Users/win/Dropbox/USP/2023/remote
MAP2112_2023/scripts')
Hello! Welcome to demofile.txt

This file
```

As linhas podem ser lidas em sequência com um laço de for.

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

```
In [8]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula7_files3.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!
```

O objeto x que varia no laço é do tipo string

```
In [9]: type(x)
Out[9]: str
```

Uma boa prática após o uso de um arquivo é fechá-lo usando o método *file.close*.

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Isso previne escritas e leituras indesejáveis

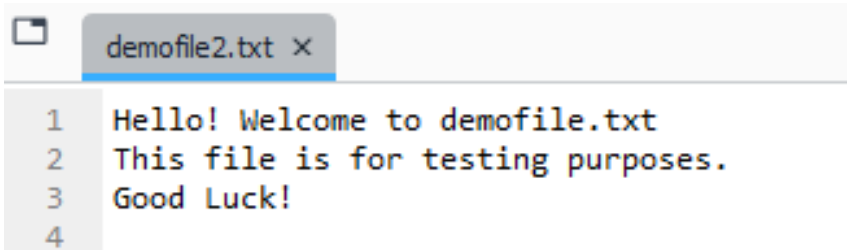
Para escrever em arquivos deve-se incluir na abertura do arquivo as opções:

"a" - Append – Abre o arquivo para adicionar conteúdo a partir do seu fim, cria o arquivo se ele não existe.

"w" - Write – Abre o arquivo para escrita, cria o arquivo se ele não existe.

para indicar como proceder no caso de escrita.

Usando um arquivo “demofile2.txt” com conteúdo idêntico ao demofile.txt.

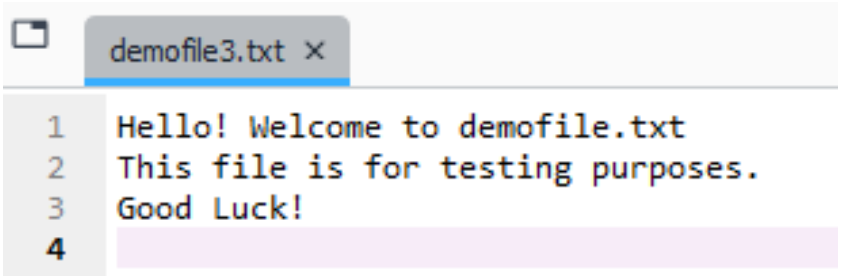


Usando o método *write* num arquivo aberto com a opção *append* permite a adição de conteúdo ao final do arquivo.

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()


f = open("demofile2.txt", "r")
print(f.read())
```

```
In [13]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula7_files5.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
Now the file has more content!
```

Usando agora um arquivo “demofile3.txt” com conteúdo idêntico ao demofile.txt.

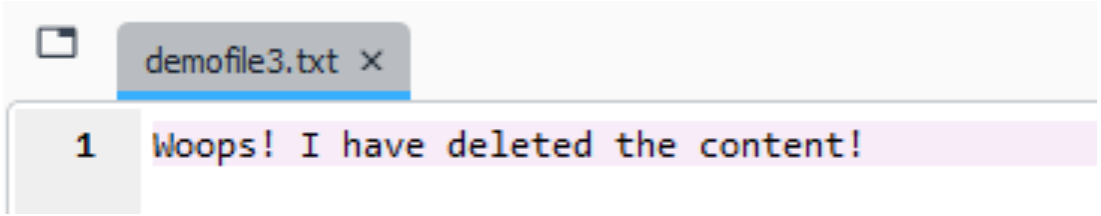


A opção “w” inicia a escrita no início do arquivo sobrescrevendo o conteúdo anterior

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()


f = open("demofile3.txt", "r")
print(f.read())
```

```
In [14]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula7_files6.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
Woops! I have deleted the content!
```

Abrindo o arquivo no editor



Chapter 12 Classes and objects

Chapter 13 Classes and functions

Chapter 14 Classes and methods

Classes: objetos, funções e métodos

O material dessas notas de aula irá combinar parcialmente o conteúdo desses capítulos da referência e também o material adicional de:

https://www.w3schools.com/python/python_classes.asp

Classes: Objetos, Funções e Métodos

A linguagem Python é uma linguagem orientada a objetos.

Além dos objetos e métodos nativos (ou adicionados por módulos), o usuário de Python pode criar o seus próprios objetos e métodos usando o recurso de classes.

Criando uma classe de exemplo:

```
class MyClass:
    x = 5

p1 = MyClass()

print(p1.x)
```

A classe MyClass é definida com um atributo x que tem um valor inteiro 5

Um objeto p1 é criado como instância da classe MyClass, herdando seus atributos

Quando se pede um atributo de p1, no caso x o valor já está associado por pertencer a classe

```
In [1]: runfile('C:/Users/win/Drc
remote/MAP2112_2023/scripts/aula7
wdir='C:/Users/win/Dropbox/USP/20
MAP2112_2023/scripts')
5
```

As classes tem uma função de inicialização (que se não for definida fica vazia) que permite definir quais são os atributos inicialmente esperados.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age


p1 = Person("John", 36)


print(p1.name)
print(p1.age)
```

A classe Person é definida com uma função de inicialização que indica dois atributos, nome e idade

Um objeto p1 é criado como instância da classe Person, e valores são atribuídos aos atributos na sequência em aparecem

O uso de *self* para autodesignação é uma prática dos usuários de Python, poderia ser qualquer outra palavra.

```
In [2]: runfile('C:/Users/win/
remote/MAP2112_2023/scripts/au
wdir='C:/Users/win/Dropbox/USF
MAP2112_2023/scripts')
John
36
```

Além dos atributos as classes podem conter métodos, que são funções que pertencem ao objeto.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

p1.myfunc()
```

Além da função com os atributos, foi incluída uma outra função, no caso com o objetivo de impressão.

A função é chamada como um método aplicado sobre o objeto.

```
In [3]: runfile('C:/Users/win/Dropbox/
remote/MAP2112_2023/scripts/aula7_cls3
wdir='C:/Users/win/Dropbox/USP/2023/re
MAP2112_2023/scripts')
Hello my name is John
```

O valor dos objetos podem ser modificados ao longo da execução. Objetos podem ser deletados.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age


p1 = Person("John", 36)

print(p1.age)

p1.age = 40

print(p1.age)

del p1.age

print(p1.age)
```

O valor do atributo é modificado

```
In [5]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula7_cls4.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
36
40
Traceback (most recent call last):

⋮

AttributeError: 'Person' object has no attribute
'age'
```

Um atributo pode ser deletado e deixar e fazer parte do objeto

As classes podem ser criadas inicialmente sem atributos (vazias) e posteriormente os atributos podem ser associados.

```
class Point:
    pass

blank = Point()

blank.x = 3.0
blank.y = 4.0

print(blank)
```

A classe é criada vazia e o comando pass é necessário por definição de sintaxe.

O objeto blank é definido como membro da classe ponto

Os atributos são adicionado em seguida

```
In [10]: runfile('C:/Users/win/Dropbox/USP/
2023/remote/MAP2112_2023/scripts/
aula7_cls6.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
<__main__.Point object at 0x000001F3DFA34730>
```

A indicação é que blank é um objeto Point com um id em hexadecimal

Apesar de ser possível ir criando atributos posteriormente a definição da classe, essa prática não redefine as classes apenas os objetos referenciados.

```
class Point:
    pass

def printPoint(p):
    print ("(" + str(p.x) + "," + str(p.y) + ")")

blank = Point()

blank.x = 3.0
blank.y = 4.0

printPoint(blank)

p1 = Point()

print(p1.x)
```

Um função foi definida para impressão de atributos de um objeto ainda não definido

```
In [13]: runfile('C:/Users/win/Dropbox/USP/
2023/remote/MAP2112_2023/scripts/
aula7_cls7.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
(3.0,4.0)
Traceback (most recent call last):

  ⋮

AttributeError: 'Point' object has no
attribute 'x'
```

Um atributo pode ser deletado e deixar e fazer parte do objeto

O uso mais tradicional é construir a classe com todos os atributos e métodos requeridos.

```
class Point:
  def __init__(self, x = 0, y = 0):
    self.x = x
    self.y = y

  def __str__(self):
    return ("(" + str(self.x) +"," + str(self.y) + ")")

p = Point(3, 4)

print(p)
```

Pode-se definir na própria função - *__init__* valores default para os atributos

A função *__str__* define como será a saída do conteúdo da classe caso ela seja impressa (print)

```
In [16]: runfile('C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts/aula7_cls8.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
(3,4)
```

A escolha do formato de impressão da classe foram os atributos x e y.

Sobrecarga de Operadores (Operator Overloading)

Pode-se usar funções na classe que redefinem operadores quando aplicados sobre objetos definidos por elas.

```
class Point:
  def __init__(self, x = 0, y = 0):
    self.x = x
    self.y = y

  def __str__(self):
    return ("(" + str(self.x) +"," + str(self.y) + ")")

  def __add__(self, other):
      return Point(self.x + other.x, self.y + other.y)

p1 = Point(3, 4)

p2 = Point(5, 7)

p3 = p1 + p2

print(p3)
```

A definição da função *__add__* determina como o operador soma (+) deve ser interpretador quando for aplicado aos objetos da classe

```
In [17]: runfile('C:/Users/win/Dropb
remote/MAP2112_2023/scripts/aula7_cl
wdir='C:/Users/win/Dropbox/USP/2023/
MAP2112_2023/scripts')
(8,11)
```

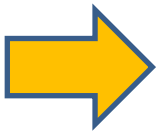
De forma análoga outras funções de operadores podem definidas

```
class Point:
  def __init__(self, x = 0, y = 0):
    self.x = x
    self.y = y

  def __str__(self):
    return ("(" + str(self.x) +"," + str(self.y) + ")")

  def __sub__(self, other):
      return Point(self.x - other.x, self.y - other.y)

  def __mul__(self, other):
      return Point(self.x * other.x, self.y * other.y)

  def __rmul__(self, other):
      return Point(other * self.x , other * self.y)

p1 = Point(3, 4)

p2 = Point(5, 7)

p3 = p1 - p2

p4 = p1 * p2

p5 = 2 * p1

print(p3)
print(p4)
print(p5)
```

operador subtração

operador multiplicação

operador multiplicação por escalar



```
In [19]: runfile('C:/Users/w
remote/MAP2112_2023/scripts/
wdir='C:/Users/win/Dropbox/U
MAP2112_2023/scripts')
(-2,-3)
(15,28)
(6,8)
```

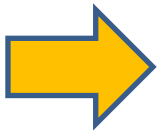
Polimorfismo

Uma vez que os operadores sobre os objetos estão definidos todas funções ao utilizarem operadores sobre os objetos irão carregar as definições.

Funções que podem operar sobre objetos de diferentes tipos são chamadas polimórficas.

```
class Point:
  def __init__(self, x = 0, y = 0):
    self.x = x
    self.y = y

  def __str__(self):
    return ("(" + str(self.x) +"," + str(self.y) + ")")

  def __add__(self, other):
      return Point(self.x + other.x, self.y + other.y)

  def __rmul__(self, other):
      return Point(other * self.x , other * self.y)

def multadd (x, y, z):
    return x * y + z

p1 = Point(3, 4)

p2 = Point(5, 7)

print(multadd(2,p1,p2))
```

Essa função pode ser aplicada a qualquer objeto uma vez definidas as operações



```
In [20]: runfile('C:/Users/win/D
remote/MAP2112_2023/scripts/aula
wdir='C:/Users/win/Dropbox/USP/2
MAP2112_2023/scripts')
(11,15)
```


The End

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict ...)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, s for a string, n for a number, L for a list where i,j are list indexes, D stands for a dictionary and k is a dictionary key.

Fim Aula 07