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 3 Functions

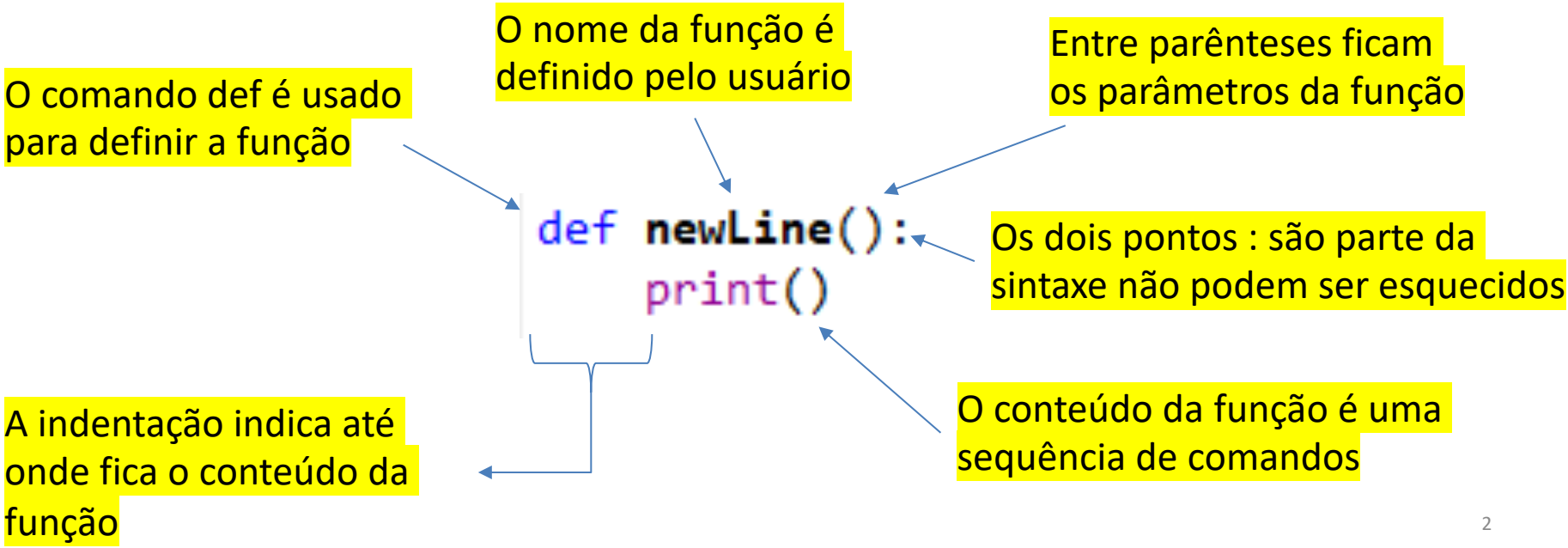
3.6 Adding new functions

("Adicionando novas funções")

Na aula anterior vimos as funções matemáticas fornecidas pelo *math module* (*https://docs.python.org/3/library/math.html*).

Agora veremos como o usuário pode definir suas próprias funções.

Começando com um exemplo ilustrativo (pouco ou nada útil ☺)



As funções são definidas antes do corpo principal do programa onde serão utilizadas

teste_fun1.py

```
1
2   def newLine():
3       print()
4
5   #---------------------
6
7   print("First Line")
8   newLine()
9   print("Second Line")
```

O que essa função faz é inserir uma linha na saída do código quando é chamada

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_fun1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
First Line

Second Line
```

teste_fun2.py

```
1
2   def newLine():
3       print()
4
5   #--------------------
6
7   print("First Line")
8   newLine()
9   newLine()
10  newLine()
11  print("Second Line")
```

Aumentando o número de chamadas o número de linhas correspondentes aumenta

```
In [4]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_fun2.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
First Line



Second Line
```

Qual é uso de funções :

- As funções encapsulam sequências de comandos sob um único nome (o nome da função) facilitando a leitura da parte principal do código.

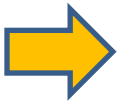
- Se essa sequência é usada em vários lugares da parte principal do código, o uso de funções deixa o código mais limpo e legível.

- Funções podem ser construídas usando os outras funções, tanto do uso usuário como módulos, o que promove a redução do código e aumenta sua legibilidade.

3.7 Definitions and use ("Definições e Uso")

teste_fun3.py

```
1
2   def newLine():
3       print()
4
5   #--------------------
6
7   def threeLines():
8       newLine()
9       newLine()
10      newLine()
11
12  #--------------------
13
14  def nineLines():
15      threeLines()
16      threeLines()
17      threeLines()
18
19  #--------------------
20
21  print("First Line")
22  nineLines()
23  print("Second Line")
```

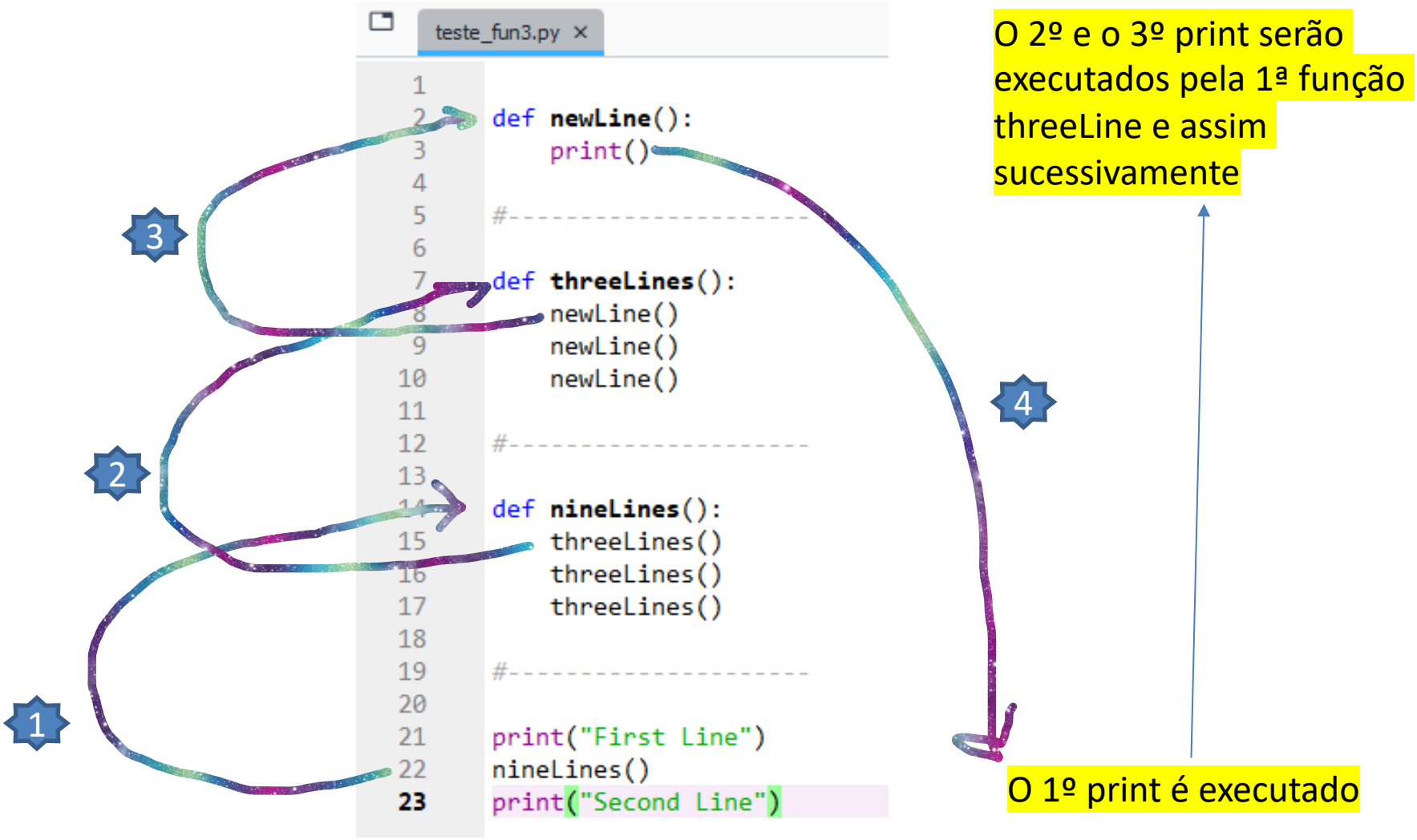


```
In [6]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_fun3.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
First Line


Second Line
```

O exemplo é apenas ilustrativo do conceito

3.8 Flow of execution ("Fluxo de Execução")

A sequência de execução do código depende do encadeamento das chamadas das funções.



3.9 Parameters and arguments
("Parâmetros e Argumentos")

Vimos que algumas das funções nativas requerem argumentos.

teste_fun3.py × untitled1.py* ×

```
1
2   import math
3
4   a = math.pi/4
5
6   print(math.sin(a))
7
8   x = 2
9
10  y = 3
11
12  print(pow(x,y))
13
```

Argumento do seno é o ângulo (em radianos)

sin

Definition : sin(measured in radians)
Type : Function of math module

Return the sine of x (measured in radians).

Os argumentos da função "potência" são a base e o expoente.

pow

Definition : pow(...)
Type : Function of builtins module

Equivalent to base**exp with 2 arguments or base**exp % mod with 3 arguments

```
In [8]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/untitled1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
0.7071067811865476
8
```

As funções definidas por usuários também podem conter parâmetros aos quais os argumentos serão atribuídos para a realização da computação desejada.

Considere a função simples:

teste_fun3.py × teste_fun5.py ×

```
1
2   def printTwice(bruce):
3       print(bruce,bruce)
4
5   printTwice("spam")
6
7   printTwice(5)
8
9   printTwice(3.14159)
```

Os parâmetros são definidos na função.

Na chamada os argumentos serão processados conforme os parâmetros.

```
In [15]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_fun5.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
spam spam
5 5
3.14159 3.14159
```

Os argumentos são passados para as funções internas da função criada pelo usuário respeitando as mesmas propriedades.

operação de multiplicação de strings.

```
printTwice("spam"*4)

printTwice(math.cos(math.pi))
```

operação matemática “in place”.

```
In [17]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_fun5.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
spamspamspamspam spamspamspamspam
-1.0 -1.0
```

```
michael = "Eric, the half a bee"

printTwice(michael)
```

Variáveis podem ser argumentos

```
In [18]: runcell(0, 'C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_fun5.py')
Eric, the half a bee Eric, the half a bee
```

3.10 Variables and parameters are local

("Variáveis e parâmetros são locais")

As variáveis criadas dentro da função só existem dentro delas. Chamadas externas geram erros.

teste_fun3.py × teste_fun5.py × teste_fun6.py ×

```
1
2   def printTwice(bruce):
3       print(bruce,bruce)
4
5   def catTwice(part1, part2):
6       cat = part1 + part2
7       printTwice(cat)
8
9   chant1 = "Pie Jesu domine"
10  chant2 = "Dona eis requiem"
11
12  catTwice(chant1, chant2)
13
14  print(cat)
```

O editor já reclama

```
In [19]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_fun6.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
Pie Jesu domineDona eis requiem Pie Jesu domineDona eis requiem
Traceback (most recent call last):

  File C:\Program Files\Spyder\pkgs\spyder_kernels\py3compat.py:356
in compat_exec
    exec(code, globals, locals)

  File c:
\users\win\dropbox\usp\2023\remote\map2112_2023\scripts\teste_fun6.
py:14
    print(cat)

NameError: name 'cat' is not defined
```

A variável não definida produz um erro

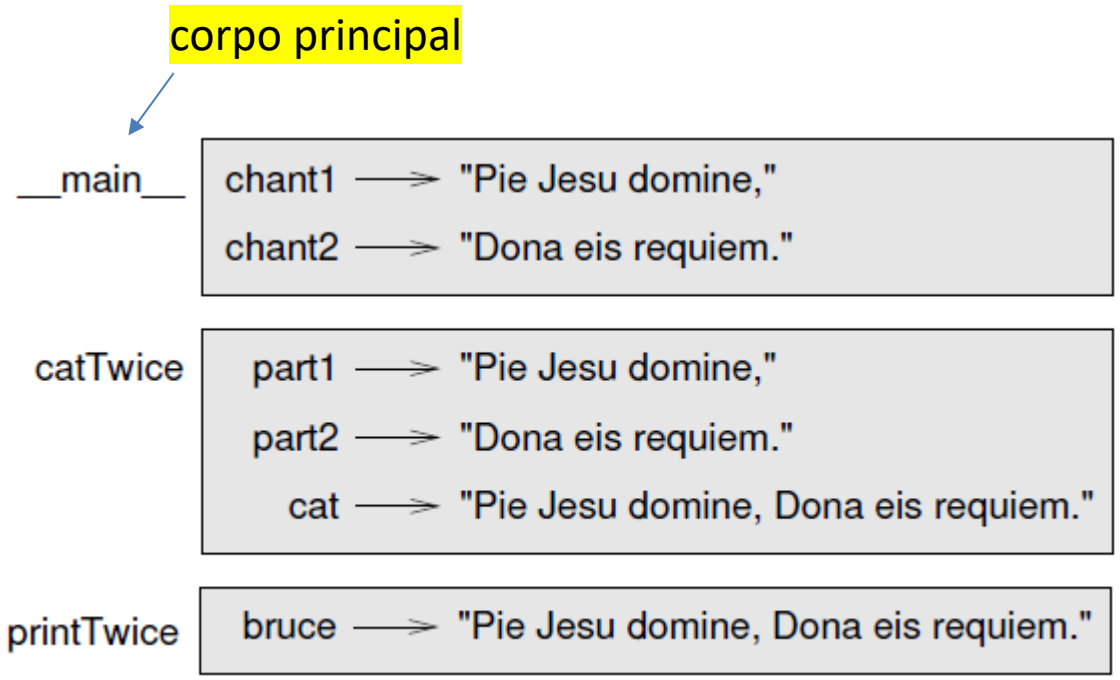
3.11 Stack diagrams

("Diagrama em pilha")

Uma forma de visualizar o fluxo de execução e as variáveis é o chamado "stack diagram")

teste_fun3.py × teste_fun5.py × teste_fun6.py ×

```
1
2   def printTwice(bruce):
3       print(bruce,bruce)
4
5   def catTwice(part1, part2):
6       cat = part1 + part2
7       printTwice(cat)
8
9   chant1 = "Pie Jesu domine"
10  chant2 = "Dona eis requiem"
11
12  catTwice(chant1, chant2)
13
14  print(cat)
```

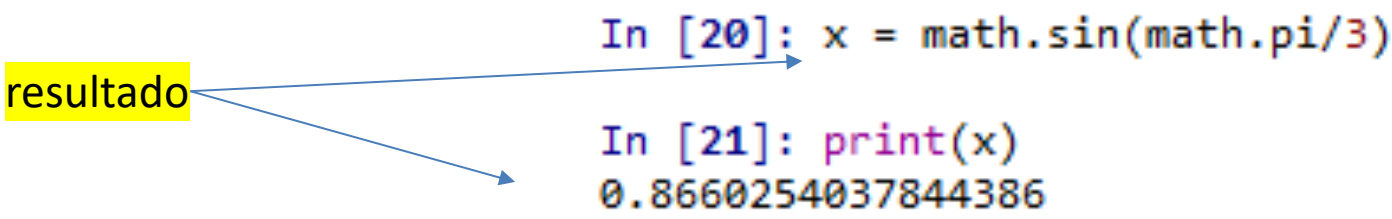


3.12 Functions with results

("Funções com resultados")

Os exemplos utilizados nesse capítulo são de funções sem resultados, mas claramente o uso de funções não se restringe a isso.

A chamada das funções matemáticas tem como saída os resultados, resultados esses que podem ser atribuídos a variáveis.



Esse tópico será tratado em mais detalhe no capítulo 5.

Chapter 4
Conditionals and recursion

("Condicionais e recursão")

4.1 The modulus operator
("operador módulo")

```
In [1]: quotient = 7/3

In [2]: print(quotient)
2.3333333333333335

In [3]: quotient = 7//3

In [4]: print(quotient)
2

In [5]: remainder = 7 % 3

In [6]: print(remainder)
1
```

No python 3.X a divisão de inteiros com barra simples (/) converte para float

Para manter a divisão de inteiro deve usar o operador barra dupla (//)

O operador módulo (%) apresenta o resto da divisão de inteiros

Aplicando os operadores sobre floats:

```
In [7]: 7.0 // 3.0
Out[7]: 2.0

In [8]: 7.0 / 3.0
Out[8]: 2.3333333333333335

In [9]: 7.0 % 3.0
Out[9]: 1.0
```

A barra dupla (//) sobre floats produz um float

A barra simples (/) é a operação natural sobre floats

O resto da divisão sobre floats produz um float

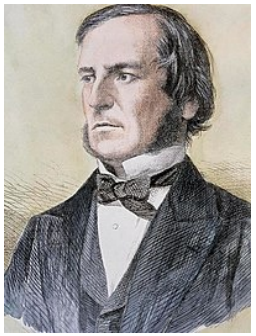
Para extrair os últimos d'igitos de um inteiro um recurso é o resto da divisão por 10, 100, etc...

```
In [10]: x = 1234567

In [11]: x % 10
Out[11]: 7

In [12]: x % 100
Out[12]: 67

In [13]: x % 1000
Out[13]: 567
```

4.2 Boolean expressions

("Expressões Booleanas")

Expressões booleanas são expressões cujos resultados são verdadeiro (TRUE) ou falso (FALSE)



George Boole
(1815-1864)

Esses resultados são produzidos pelos operadores lógicos:

```
In [14]: 5 == 5
Out[14]: True

In [15]: 5 == 6
Out[15]: False

In [16]: x = (5 == 5)

In [17]: type(x)
Out[17]: bool

In [18]: print(x)
True
```

O operador duplo igual (==) é um operador lógico de comparação

Variáveis são resultados de operações booleanas são do tipo booleano

The == operator is one of the **comparison operators**; the others are:

```
x != y          # x is not equal to y
x > y           # x is greater than y
x < y           # x is less than y
x >= y          # x is greater than or equal to y
x <= y          # x is less than or equal to y
```

Importante prestar atenção na sintaxe

```
In [19]: x = 5

In [20]: y = 10

In [21]: x <= y
Out[21]: True

In [22]: x =< y
  Cell In[22], line 1
    x =< y
      ^
SyntaxError: invalid syntax
```

4.3 Logical operators

("Operadores Lógicos")

São três os operadores lógicos:
and (e) ; or (ou) e not (não)

Operador	Função
and	lógico E
or	lógico OU
not	lógico de negação

Exemplos	
exp1 and exp2	É verdadeira se as duas **exp1** e **exp2** forem verdadeiras.
exp1 or exp2	É verdadeira se uma das duas **exp1** ou **exp2** for verdadeira ou se as duas forem verdadeiras.
not exp1	É verdadeira se **exp1** for falsa.

```
In [28]: x = 5

In [29]: x > 0 and x < 10
Out[29]: True

In [30]: n = 8

In [31]: n % 2 == 0 or n % 3 ==0
Out[31]: True

In [32]: not (x > n)
Out[32]: True
```

Ambas as expressões são verdadeiras

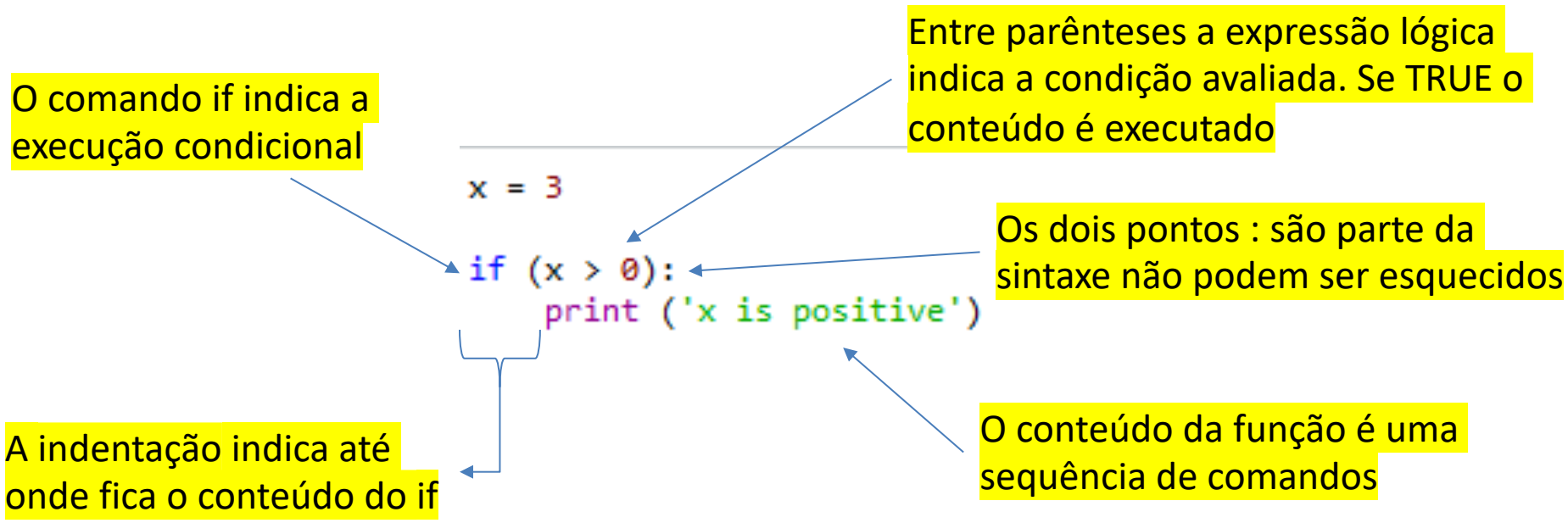
Basta que uma seja verdadeira

O operador negação inverte o resultado do conteúdo

4.4 Conditional execution

("Execução Condicional")

Para a implementação dos algoritmos de solução de diversos problemas é fundamental controlar o fluxo de execução verificando condições que definem esse fluxo.




```
In [33]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_cond1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')
x is positive
```

4.5 Alternative execution

("Execução Alternativa")

A estrutura do *if* pode conter uma sequencia de comandos para o ramo TRUE e outra para o ramo FALSE.

```
x = 44

if (x % 2 == 0):
    print (x, ' é par')
else:
    print (x, ' é ímpar')
```

O ramo TRUE é o 1º

O ramo FALSE fica depois do else:

```
In [35]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_cond2.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
44  é par
```

Naturalmente pode-se combinar estruturas de comando para criar funcionalidades reusáveis ao longo do código.

```
def printParidade(x):
    if (x % 2 == 0):
        print (x, ' é par')
    else:
        print (x, ' é ímpar')


printParidade(44)

printParidade(33)
```

A função pré-definida pode ser chamada ao longo do código.

```
In [36]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_cond3.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
44  é par
33  é ímpar
```

4.6 Chained conditionals

("Condicionais Encadeados")

Em alguns casos pode-se ter mais do que duas escolhas. Uma forma de estrutura de *if* que contempla essa possibilidade é o condicional encadeado.

```
x = 5
y = 5

if (x < y):
    print(x," é menor que ",y )
elif (x > y):
    print(x," é maior que ",y )
else:
    print(x," e ", y," são iguais" )
```

elif é uma abreviação de "else if".

```
In [37]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_cond4.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
5  e  5  são iguais
```

Não existe um limite para encadeamento, pode-se criar uma estrutura com diversas opções, como por exemplo a estrutura abaixo:

```
if choice == 'A':
  functionA()
elif choice == 'B':
  functionB()
elif choice == 'C':
  functionC()
else:
  print "Invalid choice."
```

4.7 Nested conditionals

("Condicionais Aninhados")

Condicionais podem abrigar outros condicionais em ramos para implementar opções mais complexas. Repetindo um caso anterior de forma equivalente para ilustração.

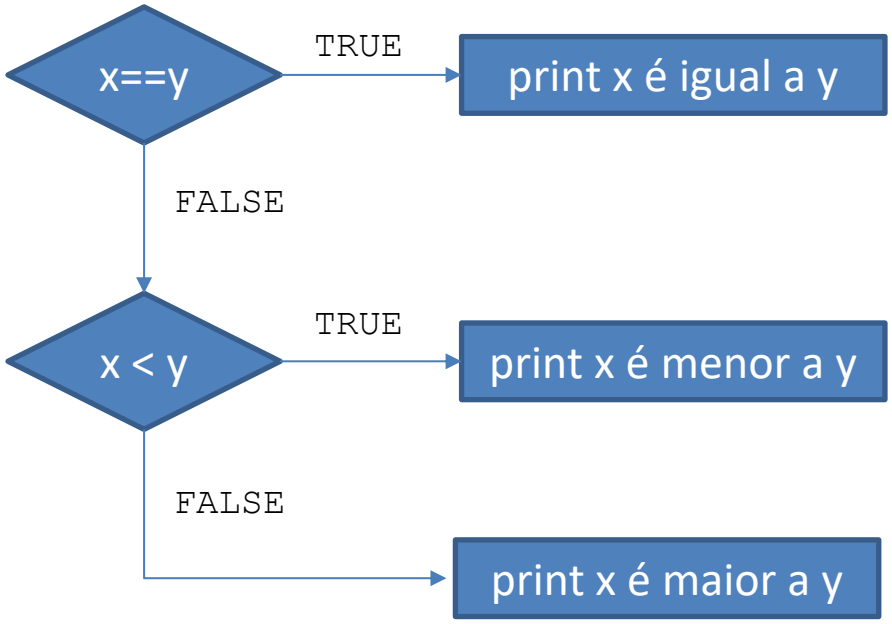
```
x = 7
y = 5

if (x == y):
    print(x," e ", y," são iguais" )
else:
    if (x < y):
        print(x," é menor que ",y )
    else:
        print(x," é maior que ",y )
```

O 1º if separa a opção de x e y iguais da opção x e y diferentes

O 2º if decide quais dos dois é maior.

Usando a visualização simbólica do fluxograma

```
if (x == y):
    print(x," e ", y," são iguais" )
else:
    if (x < y):
        print(x," é menor que ",y )
    else:
        print(x," é maior que ",y )
```



O uso de condicionais aninhados requer atenção especial na indentação.

Os operadores lógicos pode ser usados para simplificar condicionais aninhados.
Os dois exemplos a seguir são equivalentes:

```
if (0 < x):
    if (x < 10):
        print("x é positivo de um dígito")
```

```
if ((0 < x) and (x < 10)):
    print("x é positivo de um dígito")
```

O python oferece mais uma solução sintática equivalente:

```
if (0 < x < 10):
    print("x é positivo de um dígito")
```

4.8 The return statement

("o comando *return*")

O comando *return* é usado para quebrar a execução sem verificar todos os ramos. Seu uso é comum quando se deseja evitar uma condição de erro.

```
import math

def printLogarithm(x):
    if ( x <= 0 ):
        print("apenas números positivos")
        return
    result = math.log(x)
    print("o log de ",x," é ",result)

aa =  1
printLogarithm(aa)

aa =  -1
printLogarithm(aa)
```

O logaritmo não chega a ser calculado se o argumento for negativo.

```
In [44]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_cond7.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
o log de  1  é  0.0
apenas números positivos
```

A medida que o vocabulário de comandos aumenta torna-se difícil memoriza-los (e nem é esse o objetivo).

Um recurso natural em programação são as “folhas de cola” -> cheat sheets

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.

Imprima sua “cheat sheet” e localize as seções referentes ao material das aulas.

Para as provas a única consulta permita será a “cheat sheet”.

Fim Aula 03