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 4 Conditionals and recursion

4.9 Recursion

("Recursão")

Como vimos anteriormente funções podem chamar outras funções. Uma possibilidade que a linguagem Python tem é a recursão, quando a função chama a si mesma.

Essa possibilidade é uma funcionalidade muito útil em Ciência da Computação mas requer atenção devido ao potencial para erros. Considere o exemplo:

teste_rec1.py

```
1
2   def countdown(n):
3       if (n == 0):
4           print ("Lift-off!")
5       else:
6           print (n)
7           countdown(n-1)
8
9
10  countdown(10)
```

A função tem como entrada o valor n

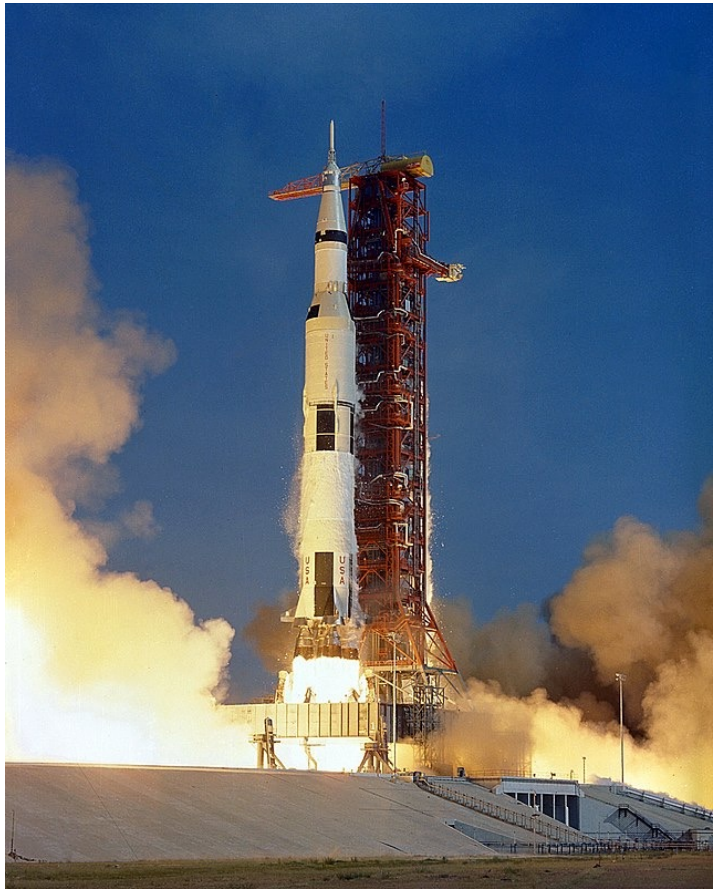
O valor n é checado se é nulo, se for o print é executado e a execução da função se encerra.

Caso n seja não nulo seu valor é impresso.

A função chamada agora um valor n – 1 em relação a chamada anterior.

A execução resulta na contagem regressiva conforme desejado.

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_rec1.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
10
9
8
7
6
5
4
3
2
1
Lift-off!
```



Um outro exemplo análogo ao exemplo didático anterior:

teste_rec2.py*

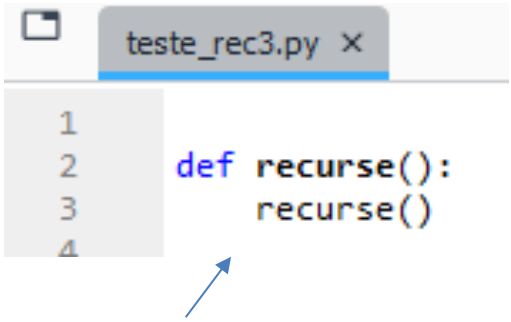
```
1
2   def nLines(n):
3       if (n > 0):
4           print()
5           nLines(n-1)
6
7
8   print('First Line')
9   nLines(9)
10  print('Second Line')
```

A recursão foi usada como uma forma de definir na execução através de um parâmetro, quantas vezes uma ação seria realizada (existem outras formas).

```
In [4]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_rec2.py', wdir='C:/Users/win/Dropbox/
USP/2023/remote/MAP2112_2023/scripts')
First Line


Second Line
```

4.11 Infinite recursion

O principal risco da recursão é que o código nunca atinja a condição de término levando a recursão infinita.



Produzindo recursão infinita

```
In [1]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_rec3.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts')

Windows fatal exception: stack overflow

Main thread:
Current thread 0x00000a14 (most recent call first):
  File "c:
\users\win\dropbox\usp\2023\remote\map2112_2023\scripts\teste_rec3.py",
line 11 in recurse
  File "c:
\users\win\dropbox\usp\2023\remote\map2112_2023\scripts\teste_rec3.py",
line 11 in recurse
                                .
                                .
                                .
\users\win\dropbox\usp\2023\remote\map2112_2023\scripts\teste_rec3.py",
line 11 in recurse
  File "c:
\users\win\dropbox\usp\2023\remote\map2112_2023\scripts\teste_rec3.py",
line 11 in recurse
  ...

Restarting kernel...

In [1]:
```

O Windows “crasha”

O kernel python é re-iniciado

4.12 Keyboard input

("Entrada pelo teclado")

A entrada pelo teclado pode ser feita pela função *input*.

teste_input1.py

```
1
2   meu_nome = input("Qual é o seu nome: ")
3
4   print("Meu nome é : ", meu_nome)
5
```

Uma instrução de preenchimento é necessária para que o usuário saiba o que preencher

```
In [1]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts/
teste_input1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
Qual é o seu nome: Luis Carlos
Meu nome é :  Luis Carlos
```

O usuário digita, aperta enter e o comando input segue o fluxo do código

O comando input espera como entrada uma string. Se a entrada for numérica ela será tratada como string.

teste_input1.py

```
1
2   meu_nome = input("Qual é o seu nome: ")
3
4   print("Meu nome é : ", meu_nome)
5
6   print(type(meu_nome))
7
```

```
In [3]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts/
teste_input1.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
Qual é o seu nome: 5
Meu nome é :  5
<class 'str'>
```

O tratamento dos tipos deve ser feito internamente pelo usuário no código para evitar erros.

Considere o seguinte código de exemplo:

teste_input2.py

```
1
2   segs_str = input("Qual o número de segundos que deseja converter: ")
3   segs_int = int(segs_str)
4
5   horas = segs_int // 3600
6   segs_restantes = segs_int % 3600
7   minutos = segs_restantes //60
8   segs_restantes_final = segs_restantes % 60
9
10  print(horas, "horas,", minutos,"minutos e", segs_restantes_final,"segundos.")
11
```

Conversão do string em int (comando simular para float)

```
In [6]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts/teste_input2.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
Qual o número de segundos que deseja converter: 17345
4 horas, 49 minutos e 5 segundos.
```

Chapter 5 Fruitful functions

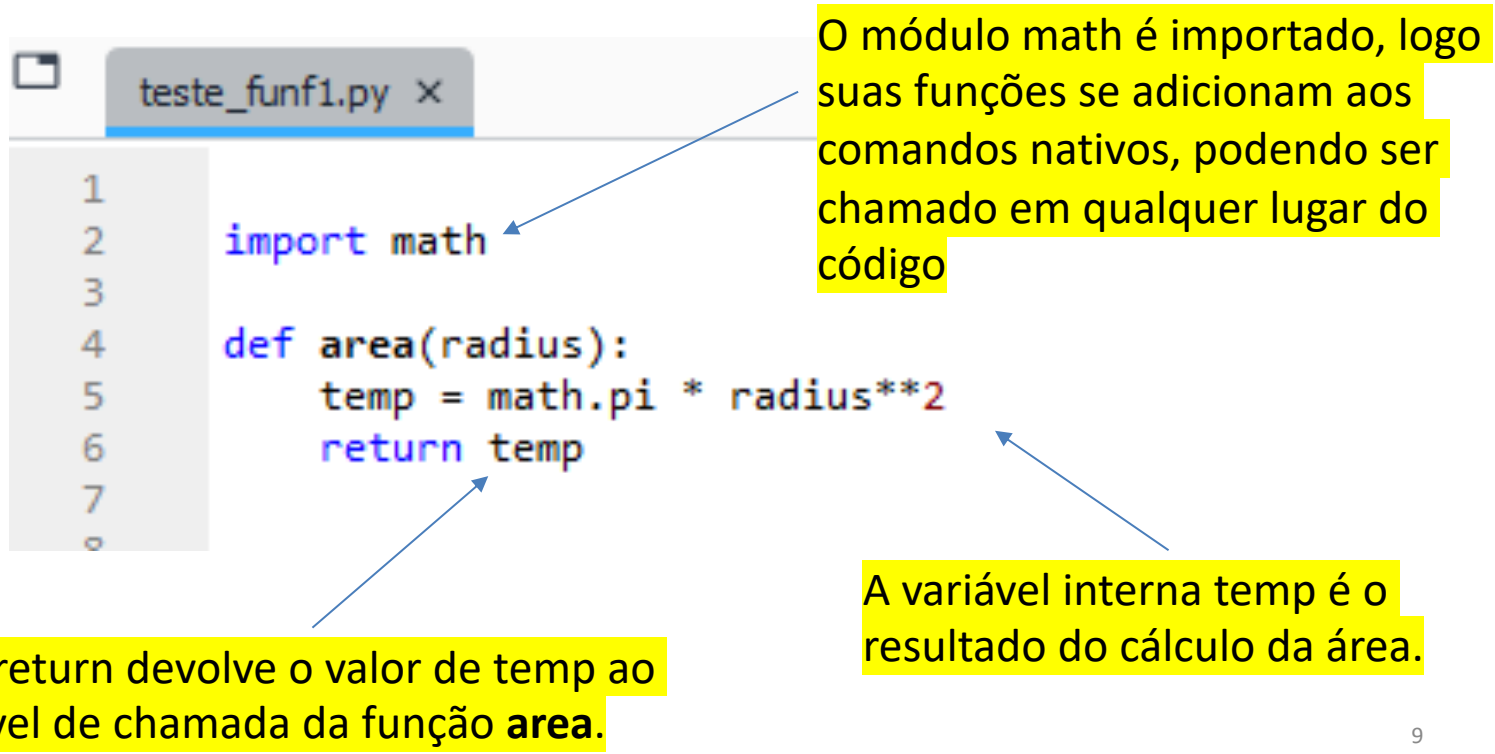
("Funções Frutíferas" ou Úteis ☺)

5.1 Return values

("Valores de retorno")

As funções são trechos de código que por motivos de organização e re-uso são definidos e nomeados.

De forma semelhante às funções nativas e aos módulos (por exemplo, math) as funções definidas pelo usuário podem ter saídas definidas.



```
2   import math
3
4   def area(radius):
5       temp = math.pi * radius**2
6       return temp
7
8   r = 5
9
10  x = area(r)
11
12  print(x)
13
14  print(temp)
15
16  print(radius)
```

O IDE já reclama das variáveis que não estão definidas no nível externo

```
In [7]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts/teste_funf1.py',
wdir='C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
78.53981633974483
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_funf1.py:14
    print(temp)

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

resultado

indicação do erro

Um recurso da linguagem é a compactação da execução “in place” (reduz o código mas em alguns casos dificulta a legibilidade)

```
def area(radius):
    return math.pi * radius**2
```

teste_funf2.py

```
1   def absoluteValue(x):
2       if x < 0:
3           return -x
4       elif x > 0:
5           return x
6
7   r = 0
8
9   a = absoluteValue(r)
10
11  print(a)
12
```

```
In [10]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts/
teste_funf2.py', wdir='C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
None
```

Funções que envolvem condicionais devem prever todas as opções para evitar erros na hora da chamada.

5.2 Program development + 5.3 Composition

(“Desenvolvimento de Programas”) (“Composição”)

Vamos partir de um enunciado:

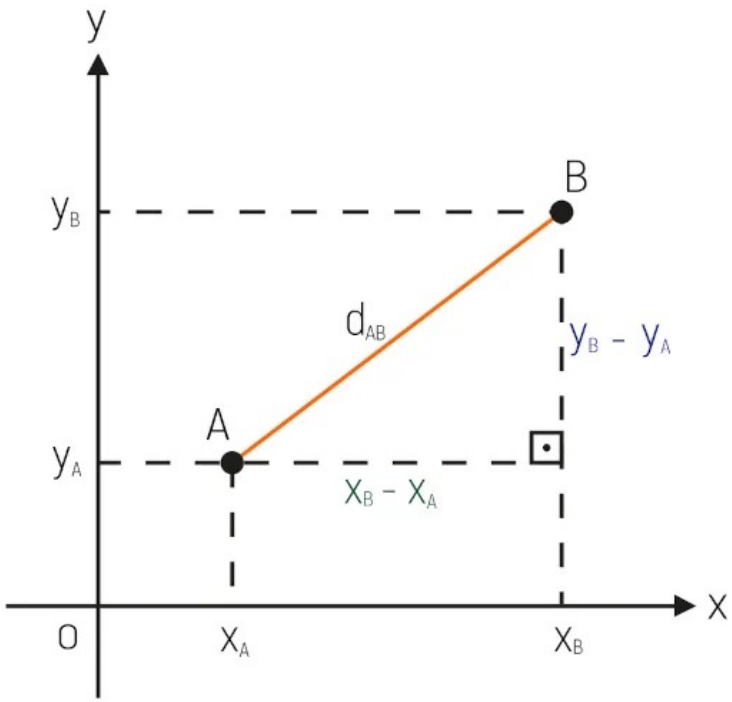
“Faça um programa em python que dados as coordenadas do centro e de qualquer ponto no perímetro de um círculo calcule sua área.”

Quais seriam os elementos necessários para a construção desse programa ?

Quais são os elementos necessários para o cálculo desejado ?

Área do Círculo

$$A = \pi . r^2$$



$$d_{AB} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Distância entre dois pontos

Entrada de dados:

teste_funf3.py

```
1
2   # Programa que calcula a área do circulo dado o centro e dos pontos
3   # no perímetro
4
5   # Entrada de dados
6
7   print("Entrada das coordenadas do centro do círculo")
8   xr_str = input("Entre com a coordenada x : ")
9   yr_str = input("Entre com a coordenada y : ")
10
11  xr = float(xr_str)
12  yr = float(yr_str)
13
14  print("x do centro",xr)
15  print("y do centro",yr)
16
17  print("Entrada das coordenadas de qualquer ponto no perímetro do círculo")
18  xp_str = input("Entre com a coordenada x : ")
19  yp_str = input("Entre com a coordenada y : ")
20
21  xp = float(xp_str)
22  yp = float(yp_str)
23
24  print("x de ponto sobre o perímetro",xp)
25  print("y de ponto sobre o perímetro",yp)
```

Entrada de strings

Conversão de formato

Impressão para checar se a leitura foi correta. Pode ser comentado quando o código final estiver pronto.

```
31  import math
32
33  def distance(xa,ya,xb,yb):
34      dist = math.sqrt((xa-xb)**2 + (ya-yb)**2)
35      return dist
36
37  radius = distance(xr, yr, xp, yp)
38
39  print()
40  print("Raio do círculo; ", radius)
41
42  # definição da função de cálculo de area
43
44  def area(r):
45      s = math.pi*(r**2)
46      return s
47
48  area_circ = area(radius)
49
50  print()
51  print("Area do círculo; ", area_circ)
```

Importação do módulo math para usar as funções matemáticas

Função de calculo da distância

Valor do raio para os dados de entrada

Impressão do valor do raio

Função de cálculo da área

Valor da área para o raio calculado

Impressão do valor da área

O código foi sendo construído e testado com o uso de prints


```
2   # Programa que calcula a área do circulo dado o centro e dos pontos
3   # no perímetro
4
5   # Entrada de dados
6
7   print("Entrada das coordenadas do centro do círculo")
8   xr_str = input("Entre com a coordenada x : ")
9   yr_str = input("Entre com a coordenada y : ")
10
11  xr = float(xr_str)
12  yr = float(yr_str)
13
14  #print()
15  #print("x do centro",xr)
16  #print("y do centro",yr)
17
18  print("Entrada das coordenadas de qualquer ponto no perímetro do círculo")
19  xp_str = input("Entre com a coordenada x : ")
20  yp_str = input("Entre com a coordenada y : ")
21
22  xp = float(xp_str)
23  yp = float(yp_str)
24
25  #print()
26  #print("x de ponto sobre o perímetro",xp)
27  #print("y de ponto sobre o perímetro",yp)
28
29  # definição da função de cálculo do perímetro
30
31  import math
32
33  def distance(xa,ya,xb,yb):
34      dist = math.sqrt((xa-xb)**2 + (ya-yb)**2)
35      return dist
36
37  radius = distance(xr, yr, xp, yp)
38
39  print()
40  print("Raio do círculo; ", radius)
41
42  # definição da função de cálculo de area
43
44  def area(r):
45      s = math.pi*(r**2)
46      return s
47
48  area_circ = area(radius)
49
50  print()
51  print("Area do círculo; ", area_circ)
```

```
In [18]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/s
win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
Entrada das coordenadas do centro do círculo
Entre com a coordenada x : 0
Entre com a coordenada y : 0
Entrada das coordenadas de qualquer ponto no perímetro do círculo
Entre com a coordenada x : 1
Entre com a coordenada y : 1

Raio do círculo;  1.4142135623730951

Area do círculo;  6.283185307179588
```

```python
# Programa que calcula a área do círculo dado o centro e dos pontos
# no perímetro

import math

# definição da função de cálculo do perímetro
def distance(xa,ya,xb,yb):
    dist = math.sqrt((xa-xb)**2 + (ya-yb)**2)
    return dist

# definição da função de cálculo de area
def area(r):
    s = math.pi*(r**2)
    return s

# Entrada de dados
print("Entrada das coordenadas do centro do círculo")
xr_str = input("Entre com a coordenada x : ")
yr_str = input("Entre com a coordenada y : ")

xr = float(xr_str)
yr = float(yr_str)

#print()
#print("x do centro",xr)
#print("y do centro",yr)

print("Entrada das coordenadas de qualquer ponto no perímetro do círculo")
xp_str = input("Entre com a coordenada x : ")
yp_str = input("Entre com a coordenada y : ")

xp = float(xp_str)
yp = float(yp_str)

#print()
#print("x de ponto sobre o perímetro",xp)
#print("y de ponto sobre o perímetro",yp)

radius = distance(xr, yr, xp, yp)

print()
print("Raio do círculo; ", radius)

area_circ = area(radius)

print()
print("Area do círculo; ", area_circ)
```

Importação dos módulos

Definição das funções

A estrutura natural dos códigos é outra.

Entrada dos dados

Parte principal

Saída dos resultados

5.4 Boolean functions
("Funções Boolenas")

Valores booleanos podem ser saídas de funções. O uso de funções booleanas permite encapsular longas cadeias lógicas favorecendo a legibilidade do código.

Um exemplo simples:

```
1    def isDivisible(x, y):
2        if x % y == 0:
3            return True
4        else:
5            return False
```

Mesma funcionalidade só que mais curta:

```
16   def isDivisible(x, y):
17       return x % y == 0
```

O resultado:

```
In [2]: isDivisible(6, 4)
Out[2]: False

In [3]: isDivisible(6, 3)
Out[3]: True
```

As funções podem ser chamadas em condicionais:

```
19  x = 6
20  y = 3
21
22  if isDivisible(x, y):
23      print ("x is divisible by y")
24  else:
25      print ("x is not divisible by y")
26
```

```
In [4]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_funf5.py', wdir='C:/Users/
win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
x is divisible by y
```

5.5 More recursion

("Mais recursão")

O "vocabulário" de comandos permite a construção de algoritmos com o potencial de resolver uma classe ampla de problemas.

Considere alguns exemplos adicionais do uso de recursão.

A função fatorial é normalmente definida por:

$$n! = \prod_{k=1}^{n} k = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1, \qquad \forall n \in \mathbb{N}$$

Por exemplo, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

Como o fatorial de um número é uma multiplicação de 1 até n. n! pode ser definido pelo produto de n por (n-1)!

$$n! = n \times (n-1)!$$

Essa propriedade permite utilizar a recursão para implementação.

A função fatorial pode ser implementada na forma:

teste_funf6.py

```
1   def factorial(n):
2       if n == 0:
3           return 1
4       else:
5           recurse = factorial(n-1)
6           result = n * recurse
7           return result
8
9   n = 5
10  x = factorial(n)
11  print(x)
12
```

```
In [8]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_funf6.py', wdir='C:/Users/
win/Dropbox/USP/2023/remote/MAP2112_2023/scripts')
120
```

Para esclarecer o funcionamento da recursão vamos imprimir as variáveis internas:

```
1   def factorial(n):
2       if n == 0:
3           return 1
4       else:
5           recurse = factorial(n-1)
6           print("recurse",recurse)
7           result = n * recurse
8           print("result",result)
9           return result
10
11  n = 5
12  x = factorial(n)
13  print()
14  print("O fatorial de ",n," e´",x)
15
```

```
In [10]: runfile('C:/Users/win/D
MAP2112_2023/scripts/teste_funf6
win/Dropbox/USP/2023/remote/MAP2
recurse 1
result 1
recurse 1
result 2
recurse 2
result 6
recurse 6
result 24
recurse 24
result 120

O fatorial de  5  e´ 120
```

A recursão fica pendente até o valor mais profundo definido ser calculado, depois disso os passos são executados na ordem reversa.

5.7 One more example

("Mais um exemplo")

A função fatorial pode ser simplificada omitindo as variáveis intermediárias.

```
1   def factorial(n):
2       if n == 0:
3           return 1
4       else:
5           return n * factorial(n-1)
6
7   n = 5
8   x = factorial(n)
9   print()
10  print("O fatorial de ",n," e´",x)
```

$$n! = n \times (n-1)!$$

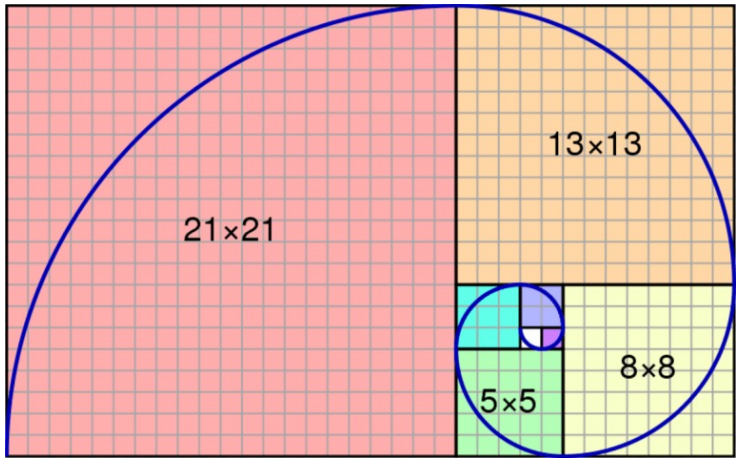
```
In [11]: runfile('C:/Users/win/D
MAP2112_2023/scripts/teste_funf7
win/Dropbox/USP/2023/remote/MAP2

O fatorial de  5  e´ 120
```

Estruturas análogas podem ser desenvolvidas da mesma forma. Tomando como exemplo a clássica sequência de Fibonacci.

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Leonardo Fibonacci
(1170-1250)

1, 2, 3, 5, 8, 13, 21 ...



A taxa de crescimento da sequência de Fibonacci e próxima do comprimento áureo.

A implementação segue a estrutura da definição:

```
1   def fibonacci (n):
2       if ((n == 0) or (n == 1)):
3           return 1
4       else:
5           return fibonacci(n-1) + fibonacci(n-2)
6
7   n = 5
8   x = fibonacci(n)
9   print()
10  print("O elemento ",n," da sequencia de Fibonacci e´",x)
```

```
In [12]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_funf8.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')

O elemento  5  da sequencia de Fibonacci e´ 8
```

5.8 Checking types

("Checando tipos")

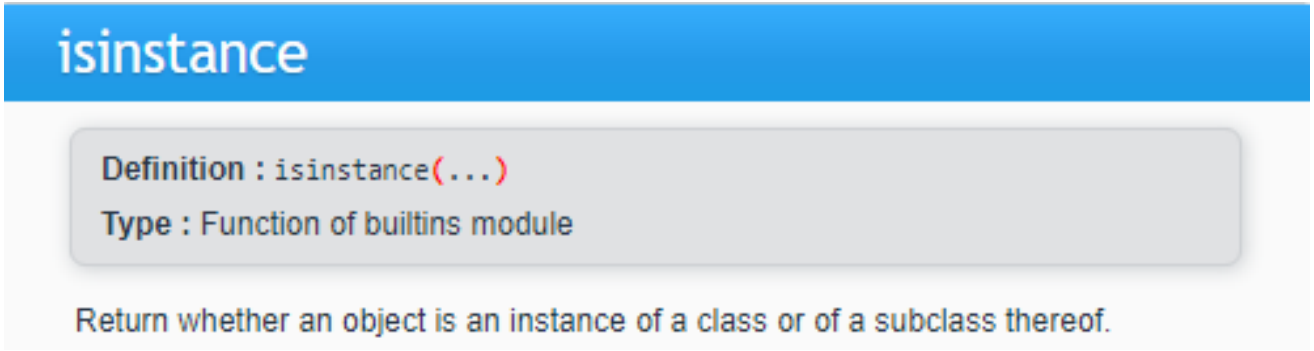
A função fatorial espera por inteiros. Se a entrada for um float temos um erro.

```
n = 1.5
x = factorial(n)
print()
print("O fatorial de ",n," e´",x)
```

```
In [14]: runfile('C:/Users/win/Dropbox/USP/2023/remote/
MAP2112_2023/scripts/teste_funf7.py', wdir='C:/Users/win/
Dropbox/USP/2023/remote/MAP2112_2023/scripts')


Windows fatal exception: stack overflow
```

Para evitar situações indesejadas deve-se "proteger" o código de condições inadequadas.

A função *isinstance* verifica se uma variável é de um tipo declarado. Ela pode ser servir para filtrar os casos indesejados.




```
In [2]: isinstance("João", str)
Out[2]: True

In [3]: isinstance(5, float)
Out[3]: False
```

```python
def factorial (n):
    if (not isinstance(n, int)):
        print ("Factorial is only defined for integers.")
        return -1
    elif (n < 0):
        print ("Factorial is only defined for positive integers.")
        return -1
    elif (n == 0):
        return 1
    else:
        return n * factorial(n-1)

n = 1.5
x = factorial(n)
print()
print("O fatorial de ",n," e´",x)
```

O usuário adotou -1 como código de erro.

```
In [4]: runfile('C:/Users/win/Dropbox/USP/2023/remote/MAP2112_2023/
scripts/teste_funf9.py', wdir='C:/Users/win/Dropbox/USP/2023/
remote/MAP2112_2023/scripts')
Factorial is only defined for integers.

O fatorial de  1.5  e´ -1
```

Experimente tentar resolver alguns exercícios simples citados no texto de referência:

As an exercise, write a `compare` *function that returns* `1` *if* `x > y`, `0` *if* `x == y`, *and* `-1` *if* `x < y`.

As an exercise, write a function `isBetween(x, y, z)` *that returns* `True` *if* $y \leq x \leq z$ *or* `False` *otherwise.*

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 04