

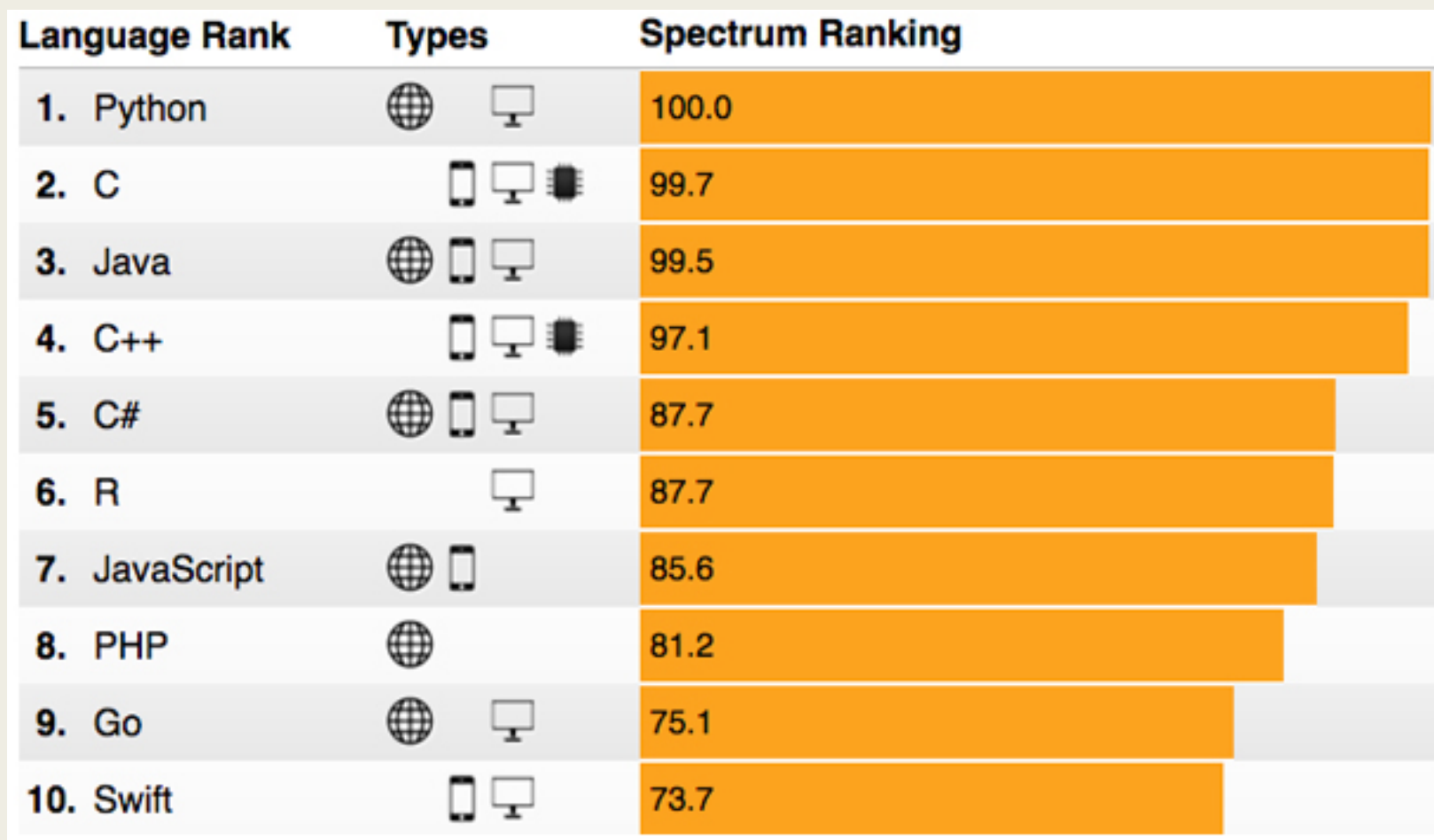
LINGUAGENS DE PROGRAMAÇÃO

Engenharia Informática

Porquê python

- Desenvolvimento muito rápido
 - *Extensa lista de bibliotecas.*
 - *Muito flexível*
- Ideal para prototipagem
 - *Visualmente limpa*
 - *Não compete em termos de performance.*
- Aplicações:
 - *GUIs, scripting, análise de dados, pequenos serviços web...*
- Linguagem pura em POO

Top used languages



<https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>

Instalação

- Download de www.python.org
 - *Windows*
 - *MACOS*
 - `brew install python3`
 - *Linux*
 - `apt-get install python3`
- Vamos seguir a versão 3.6+
 - *Há diferenças entre 2.x e 3.0!!!*
- IDE
 - *pyCharm*
 - www.jetbrains.com/pycharm
 - Download Community Version
 - Syntax highlight
 - Autocomplete
 - Run step-by-step (debug)

Python tutorial

- <https://www.tutorialspoint.com/python3/index.htm>
- David Goodger, Code Like a Pythonist: Idiomatic Python
 - <http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.htm>
- Allen B. Downey, Think Python
 - <http://greenteapress.com/wp/think-python/>
 - pdf

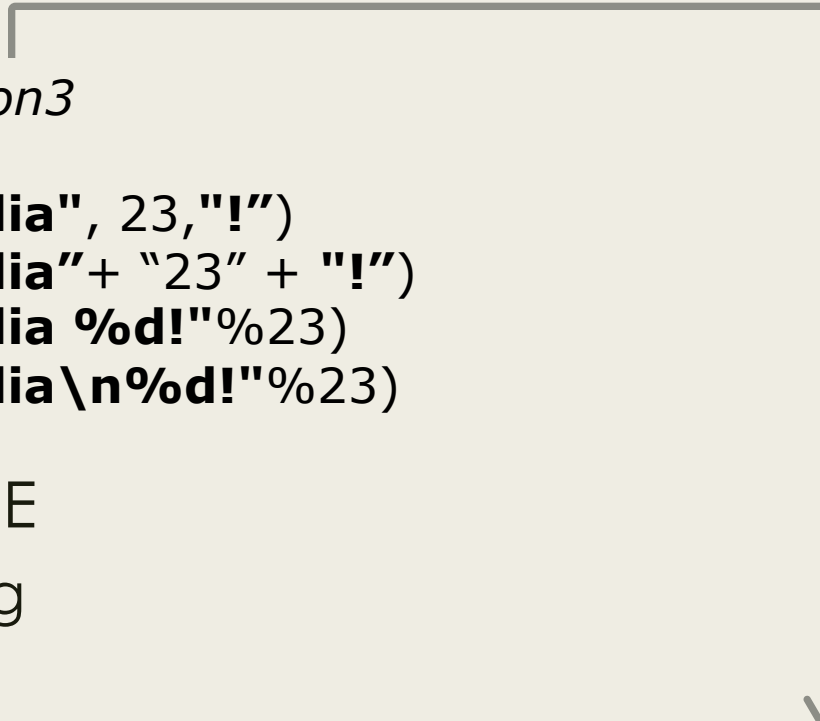
Olá Mundo

■ Editar main.py

```
#!/usr/bin/python3
```

```
print ("Hoje é dia", 23, "!")  
print ("Hoje é dia" + "23" + "!")  
print ("Hoje é dia %d!"%23)  
print ("Hoje é dia\n%d!"%23)
```

- Executar no IDE
 - Run ou Debug
- Shell:
 - python3 main.py



```
>chmod +x main.py  
>./main.py
```

Identificadores

■ Nomes de variáveis e funções

```
Manpower=3  
manpower=3
```

– *Estas duas variáveis são diferentes.*

■ Palavras reservadas:

```
and      exec    not    as    finally  or  
assert  for    pass  break  from    print  
class   global  raise  continue  if        return  
def     import  try    del    in       while  
elif    is     with   else    lambda  yield  
except
```

#Comentários

- (#) comenta o resto da linha

#Isto é um comentário - não é processado pelo interpretador
`print ("Isto é executado")` *#Isto não é executado*

""" Isto também é um Comentário """

''' E isto também '''

Identação

- Python não utiliza {} para designar um bloco execução, como em Java, C, C++, C#...

The diagram illustrates Python's indentation rules. It shows three code snippets on the left, with arrows pointing from specific parts to explanatory text on the right. The first snippet is an 'if' statement where an arrow from the colon points to the text 'Cabeçaho do grupo terminado em ":"'. The second snippet is a 'for' loop where an arrow from the indented line points to the text 'Bloco indentado. <tab> ou 4 espaços'. The third snippet is a 'def' function definition where an arrow from the indented line points to the same text 'Bloco indentado. <tab> ou 4 espaços'.

```
if True:  
    print ("True")  
    print ("False")
```

Cabeçaho do grupo terminado em ":"

```
for i in range(5):  
    print(i)
```

Bloco indentado.
<tab>
ou
4 espaços

```
def funcao():  
    print ("Olá")
```

Multiple Statements on a Single Line

- Em Python a terminação das linhas com “;” não é necessária
 - *Mas a indentação é imperativa!*
- O “;” pode ser utilizado para compactar várias linhas numa só.

```
i=0  
j=0
```

```
i=0; j=0
```

Tipos de variáveis - Atribuição

```
contador = 100      # An integer assignment  
milhas    = 1000.0   # A floating point  
nome      = "João"  # A string
```

```
a = b = c = 1  
a, b, c = 1, 2, "john"
```

```
def multiReturn():  
    return 1, 2, "john"
```

```
a, b, c = multiReturn()
```

Tipos de dados

- Numéricos

- *int (signed integers)*
 - `x=1`
- *float (floating point real values)*
 - `x=1.0` `x=float(1)` `x=float("1")`
- *complex (complex numbers)*

- String

- `x="OOP is awesome!"`

- List []

- `x=[1,2,3,4, 1.1, 'string']` *#List com qualquer tipo de dados*

- Dictionary ou Dict {}

- `x={'key1': 1, 'key2': 2, 'key3': 3 }`

- Tuples

- Set

Operações Matemáticas

- Soma (+)
 - $1 + 1$
- Subtração (-)
 - $1 - 1$
- Divisão Real (/)
 - $3/2 == 1.5$
- Divisão Inteira (//)
 - $3//2 == 1$
- Resto da divisão inteira (%)
 - $3\%2 == 1$
- Multiplicação
 - $2 * 3 == 6$
- Potência de (**)
 - $2**3 == 8$

Booleanos

- True
- False

- x=True
- x=False

Decision Making

```
var = 100  
if var == 100:  
    print ("Value of expression is 100")
```

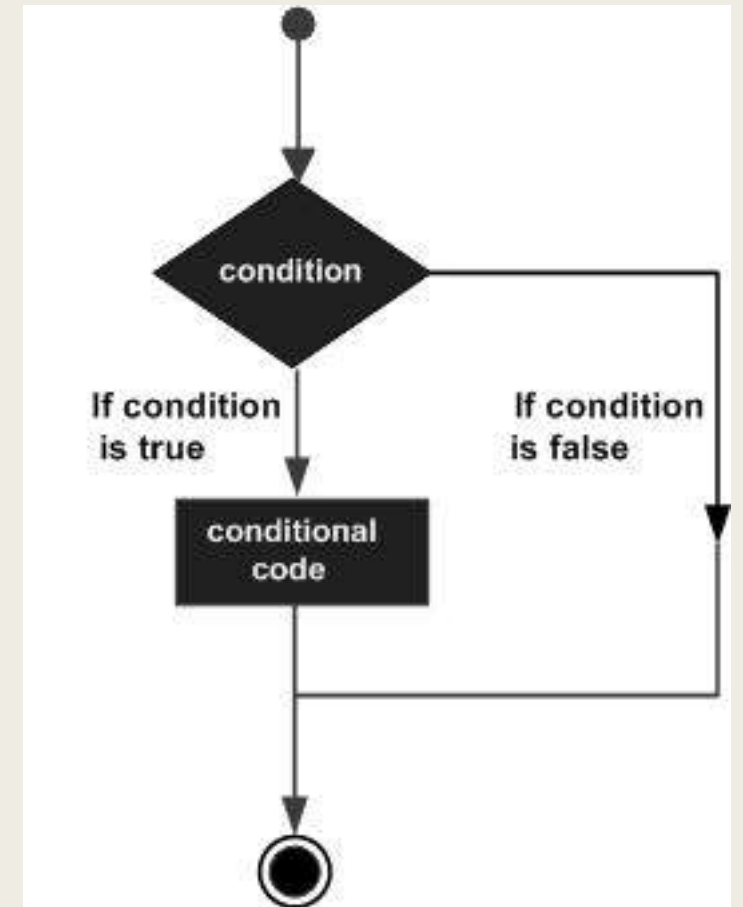
```
if var == 100:  
    print ("Value of expression is 100")  
else:  
    print ("Value of expression is NOT 100")
```

#One-liners...

```
if var == 100: print ("Value of expression is 100")  
else: print ("Value of expression is NOT 100")
```

#Atribuição Condicional

```
x = 10 if var == 100 else 20
```



Decision Making

#Atribuição Condicional

4 Linhas

```
if var == 100:  
    x = 10  
else:  
    x = 20
```

3 Linhas

```
x = 20  
if var == 100:  
    x = 10
```

1 Linha - The pythonic way!

```
x = 10 if var == 100 else 20
```

#Qual destas se lê melhor?

Decision Making

```
if var < 100:  
    print ("Escalão 1")  
else:  
    if var < 200:  
        print ("Escalão 2")  
    else:  
        print ("Escalão 3")
```

```
if var < 100:  
    print ("Escalão 1")  
elif var < 200:  
    print ("Escalão 2")  
else:  
    print ("Escalão 3")
```

→ elif - simplifica a visualização da sequência de decisões

Comparações

```
a = 3
```

```
print (a == 1)
```

```
print (a > 1)  
print (a >= 1)
```

```
print (a < 5)  
print (a <= 5)
```

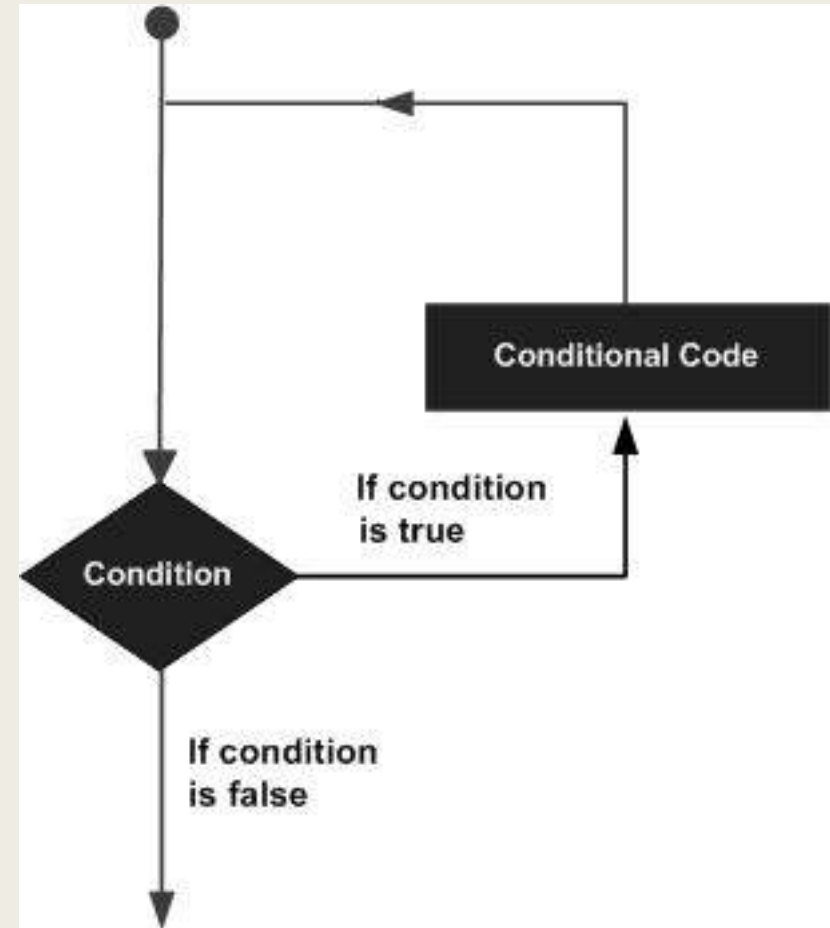
```
print (1 < a < 5) #print ( a > 1 and a < 5 )
```

Ciclos

- **for** iterating_var in sequence:
statements(s)

 for var in list(range(5)):
 print (var)
- **while** expression:
statement(s)

 count = 0
 while (count < 9):
 print ('**The count is:**', count)
 count = count + 1
- **break** interrompe o ciclo
- **continue** salta para o próximo passo



Importar bibliotecas

```
import math  
print(math.sin(60))
```

```
from math import sin  
print(sin(60))
```

```
from math import sin as sino  
print(sino(60))
```

List

```
x = [1, 2, 3, 4, 5, 6]
print (len(x))
print (x)
print (x[0])  #Primeiro elemento
print (x[1])  #Segundo elemento
print (x[-1]) #Último elemento
print (x[0:2]) #Slice do primeiro e segundo
print (x[3:])  #Slice do quarto elemto até ao fim
print (x[::2]) #Slice do Início ao fim de 2 em e
print (x[::-1])#Inverte a lista
```

```
x = x + [7, 8]  #ou x += [7,8]
print(x)
```

```
x.append(9)
```

```
x[1] = 9
print (x[1])
print (sorted(x))
```

Listas em Listas

```
x = [[1,2], [3,4], [5,6]]
```

```
print (len(x))
```

```
print (len(x[0]))
```

```
print (x[0][1])
```

Iterar Listas (for)

```
x = [3, 4, 3, 6, 10, 8]
```

```
for i in x:  
    print (i)
```

List

```
x = ['1','2','3','4','5','6','3']
```

```
print (x.pop(3)) #Remove indice 3 e devolve-o, neste caso para print  
print (x)
```

```
print (x.index('6')) #imprime o índice da primeira ocorrência de '6'  
print (x.index('10')) #Erro - não está na lista
```

```
x.append('00') #Adiciona ao fim da lista  
x.insert(0, '99') #Adiciona na posição índice 0  
print (x)
```

Tuple

```
x = () #tuple vazio
print (x)
x = ('physics', 'chemistry', 1997, 2000)
print (x)
x = (1, 2, 3, 4, 5 )
print (x)
x = (1,) #Com apenas um elemento necessita de uma vírgula no fim
print (x)
x = "a", "b", "c", "d"
print (x)
```

```
print (x[1]) #Acede ao segundo elemento
```

```
#Tuples são imutáveis - não podem ser modificados como listas
x[1] = "f" #Não é possível
x.append("f") #Não existe
```


Dictionary

Key:Value

```
x = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
print (x['Name'])
```

```
print (x['Age'])
```

```
x['NewKey'] = 'Value' #Adiciona um novo elemento se não existir
```

```
x['Age'] = 8 #Altera Valor da chave 'Age'
```

```
del x['Name'] #Remove Chave e Valor de 'Name'
```

```
len(x) #Número de elementos
```

```
x.clear() #Limpa todos os elementos
```

Funções

■ Passagem por valor ou referência

Function definition

```
def printme( str ):
    print (str)
return
```

Now we can call printme function

```
printme("LP is awesome!")
```

```
def apagalista( lista ):
    lista.clear()
return
```

Em python vai tudo por referência

```
mLista = [1,2,3,4,5,6]
apagalista(mLista)
print (mLista)  # A lista ficou vazia
```

Argumentos de Funções

```
def printinfo( name, age=35 ):
    print ("Name: ", name)
    print ("Age ", age)
    return
```

```
printinfo( "João", 50 )           #Required args
printinfo( age = 50, name = "João" ) #Keyword args
printinfo( name = "João" )       #default args
```

```
#Argumentos tamanho Variável
def printinfo( arg1, *vartuple ):
    print ("Output is: ")
    print (arg1)
    for var in vartuple: print (var)
    return
```

```
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Funções Anônimas

```
def sum(arg1, arg2):  
    return arg1 + arg2
```

```
sum1 = sum  
sum2 = lambda arg1, arg2: arg1 + arg2 #Função lambda....
```

```
print(sum1(1,2))  
print(sum2(1,2))
```

return

```
def sum( arg1, arg2 ):
    total = arg1 + arg2
    return total
```

```
def invert( arg1, arg2 ):
    return arg2, arg1 #Igual a return (arg2, arg1)
```

```
def retlist( arg1, arg2 ):
    return [arg1, arg2]
```

```
print(sum(1,2))
```

```
x_tuple = invert(1,2)
print(x_tuple[0],x_tuple[1])
```

```
x_arg1, x_arg2 = invert(1,2)
print(x_arg1, x_arg2)
```

```
print(retlist(1,2))
```

Argumentos e return

- Há uma grande flexibilidade no tipo de dados que se pode passar ou retornar de uma função.
- Cabe ao programador ser criterioso e saber o que quer passar/retornar.

Global vs. Local variables

- Variável local sobrepõem-se à global

```
total = 0 # This is a global variable.
```

```
def sum( arg1, arg2 ):
```

```
    total = arg1 + arg2; # Here total is local variable.
```

```
    print ("Local Total na função: ", total)
```

```
    return total
```

```
sum( 10, 20 )
```

```
print ("Global total fora da função: ", total )
```

- A variável global pode ser utilizada para leitura

```
total = 0 # This is a global variable.
```

```
def sum( ):  
    print ("Local Total na função: ", total)
```

```
sum( )  
print ("Global total fora da função: ", total )
```


- A variável global pode ser incorporada no scope/namespaces da função

```
total = 0 # This is a global variable.
```

```
def sum( arg1, arg2 ):
    global total
    total = arg1 + arg2; # Here total is local variable.
    print ("Local Total na função: ", total)
    return total
```

```
sum( 10, 20 )
print ("Global total fora da função: ", total )
```

Documentar funções

- Boa prática: Documentar as funções
 - *Descreve o que a função faz*
 - *Argumentos de entrada e saída*
 - Fundamental, quando no Python se pode passar/retornar o que se queira, sem haver pré declarações, ou um compilador que detecte erros de atribuição.

```
def sum( arg1, arg2 ):
    """ Devolve a soma dos argumentos
    :param arg1 - primeira parcela:
    :param arg2 - segunda parcela:
    :return soma:
    """
    return arg1 + arg2
```

```
help(sum)
```

Funções recursivas

```
def factorial_iter(n):  
    ret=1  
    for i in range(1, n+1): #1+(0..n)  
        ret *= i  
    return ret
```

```
def factorial_rec(n):  
    if n == 1:  
        return n  
    return n * factorial_rec(n-1)
```

```
print(factorial_iter(5))  
print(factorial_rec(5))
```

Nested Functions

```
def somaPares(par1, par2):  
  
    def soma(arg1, arg2):  
        return arg1 + arg2  
  
    return soma(par1[0],par1[1]),  
soma(par2[0],par2[1])  
  
print(somaPares( (1,2), (2,3) ))
```

Strings

```
x = "Eu sou uma string e sou imutável"  
x = x + "!!!" #Uma nova string  
print(x)  
#Esta string não foi modificada. Foi re-criada!
```

```
print("String " contatenada" + "!" )
```

```
print("String %s formatada!"%("bem") )
```

```
print("String %d estrelas%s Mesmo%s"%(5,"!","!") )
```

```
print("String {0} estrelas{1} Mesmo{1}".format(5,"!") )
```

```
#Mais info: https://pyformat.info/
```

Strings como listas

(ou como tuple uma vez que é imutável)

```
x = "Manel"
```

```
print(x[0])
```

```
for i in x:  
    print(i)
```

```
print(x[1:]) #Slicing
```

Strings, Chars e Comparações

`ord('a') == 97` *# código ascii decimal*

`chr(97) == 'a'`

`'A' == 'a'` *#False*

`'A' < 'a'` *#True*

#Comparações dependem do valor ascii

`'Pedro' == 'Pedro'` *#True*

`'Pedra' < 'Pedro'` *#True*

`'Pedro' < 'Pedros'` *#True*

'anTónio' == 'antónio' #False

'anTónio'.lower() == 'antónio'.lower() #True

'anTónio'.upper() == 'antónio'.upper() #True

'anTónio'.lower().startswith('ant') #True

'anTónio'.lower().endswith('ant') #False

IO - Ficheiros

```
ficheiro = open('ficheiro.txt', 'w')
```

```
# w - write - escreve desde o início
```

```
# r - read - lê desde o início
```

```
# a - write - escreve a começar do fim...
```

```
ficheiro.write("Estou a escrever uma linha")
```

```
ficheiro.write("\n") #Muda de linha
```

```
ficheiro.write("Estou a escrever uma linha\n")
```

```
listaDeLinhas = "\n".join(["Linha1", "Linha2"])
```

```
ficheiro.writelines(listaDeLinhas)
```

```
ficheiro.close()
```

IO - Ficheiros

```
ficheiro = open('ficheiro.txt', 'r')  
print (ficheiro.read()) #Lê o ficheiro completo  
ficheiro.close()
```

```
ficheiro = open('ficheiro.txt', 'r')  
for line in ficheiro: #ou for line in ficheiro.readlines():  
    print (line) #Lê uma linha  
ficheiro.close()
```

■ Questões?