QUÍMICA COMPUTACIONAL

Introdução à Linguaem de Programação Python

Iuliia Voroshylova

voroshylova.iuliia@fc.up.pt
Departamento de Química e Bioquímica
Faculdade de Ciência da Universidade do Porto, Portugal

2023/2024

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org)

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org).
- **Operações Matemáticas:** +, -, *, /, //, %, **.

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org).
- **Operações Matemáticas:** +, -, *, /, //, %, **.
- **Interacção com o utilizador**: print para imprimir dados, e input para pedir dados ao utilizador.

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org).
- **Operações Matemáticas:** +, -, *, /, //, %, **.
- **Interacção com o utilizador**: print para imprimir dados, e input para pedir dados ao utilizador.
- **Execução Condicionada**: Blocos if/elif/else.

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org).
- **Operações Matemáticas:** +, -, *, /, //, %, **.
- **Interacção com o utilizador**: print para imprimir dados, e input para pedir dados ao utilizador.
- **Execução Condicionada**: Blocos if/elif/else.
- **Repetições:** Ciclos for (quando sabemos quantas vezes repetir).

Resumo da última aula

- **Shell:** Programa do interpretador do Python, útil para tarefas interactivas (dispon´ıvel em www.python.org).
- **Operações Matemáticas:** +, -, *, /, //, %, **.
- **Interacção com o utilizador**: print para imprimir dados, e input para pedir dados ao utilizador.
- **Execução Condicionada**: Blocos if/elif/else.
- **Repetições:** Ciclos for (quando sabemos quantas vezes repetir).
- **Repetições:** Usar o comando break quando não precisamos de mais iterações dentro de um ciclo.

Função Range – ciclo For

- A função range() pode ser representada de três maneiras diferentes:
  - range(stop_value): por defito, considera o ponto inicial como zero.
  - range(start_value, stop_value): gera a sequência com base no valor inicial e final.
  - range(start_value, stop_value, step_size): gera a sequência incrementando o valor inicial usando o tamanho do passo até atingir o valor final.
  - O valor final (stop_value) nunca é incluido!

Exemplo:

```
>>> for i in range(5):
... print(i)
0
1
2
3
4
>>> for i in range(2,6):
... print(i)
2
3
4
5
>>> for i in range(1,10,2):
... print(i)
1
3
5
7
9
```

Função Range – ciclo For

- A função range() também pode ser usada como argumento para uma lista, o que resulta em uma lista de números com comprimento igual ao stop_value
- A função len() retorna o comprimento da lista

Exemplo:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> len(list(range(10)))
10
```

Se quisermos que o valor 10 também faça parte da lista?

Ciclos while

- Usamos a instruçãoo while para repetir um bloco de instruções até que uma dada condição seja verdadeira.

Exemplo

```
# Primeiro número cujo quadrado
# é superior a 117
>>> n = 1
>>> while(n**2 <= 117):
...       n = n + 1
...
>>> print(n, n**2)
11 121
```

Ciclos while

- Usamos a instruçãoo while para repetir um bloco de instruções até que uma dada condição seja verdadeira.
- As instruções break e continue permitem converter ciclos for em while (e vice-versa).
- No entanto, a opção pelo o while marca claramente a intenção derepetição condicionada.

Exemplo

```
# Primeiro número cujo quadrado
# é superior a 117
>>> n = 1
>>> while(n**2 <= 117):
...     n = n + 1
...
>>> print(n, n**2)
11 121
>>> for i in range(100):
...     if(i**2 > 117):
...         n = i
...         break
...
>>> print(n, n**2)
11 121
```

Ciclos while

- Usamos a instruçãoo while para repetir um bloco de instruções até que uma dada condição seja verdadeira.
- As instruções break e continue permitem converter ciclos for em while (e vice-versa).
- No entanto, a opção pelo o while marca claramente a intenção derepetição condicionada.
- O bloco else no final da instrução while (facultativo) é executado no final das iterações.

Estrutura Geral da Instrução while

```
while (condição):
    (Bloco de instruções)
else:
    (Bloco de instruções final)
```

Exercício de Aplicação

Qual o primeiro inteiro n para o qual a série

$S_n = 1/n$

é inferior a 0.0025?

Scripts

- Até agora temos usado apenas a *Shell* do python, mas isso não permite re-utilizar o código!

Scripts

- Até agora temos usado apenas a *Shell* do python, mas isso não permite re-utilizar o código!
- Os programas em python são ficheiros de texto com a extensão **.py**

Scripts

- Até agora temos usado apenas a *Shell* do python, mas isso não permite re-utilizar o código!
- Os programas em python são ficheiros de texto com a extensão **.py**
- Podemos usar qualquer editor de texto (mas não processadores de texto!): nano, gEdit, Vim, Emacs... ou Bloco de Notas do Windows!

Correr *Scripts* no Linux

- O Linux pode correr ficheiros do Python como commandos num terminal.

Correr *Scripts* no Linux

- O Linux pode correr ficheiros do Python como commandos num terminal.
- Para isso é preciso preparar algumas coisas:

Correr *Scripts* no Linux

- O Linux pode correr ficheiros do Python como commandos num terminal.
- Para isso é preciso preparar algumas coisas:
  - O ficheiro tem que começar com a seguinte linha (*shebang*):
    - #!/bin/env python3

Correr *Scripts* no Linux

- O Linux pode correr ficheiros do Python como commandos num terminal.
- Para isso é preciso preparar algumas coisas:
  - O ficheiro tem que começar com a seguinte linha (*shebang*):
    - #!/bin/env python3
  - O ficheiro tem que ser classificado como executável:
    - [user@host] chmod 755 oMeuPrograma.py
    - Para correr basta colocar: ./oMeuPrograma.py

Correr *Scripts* no Linux

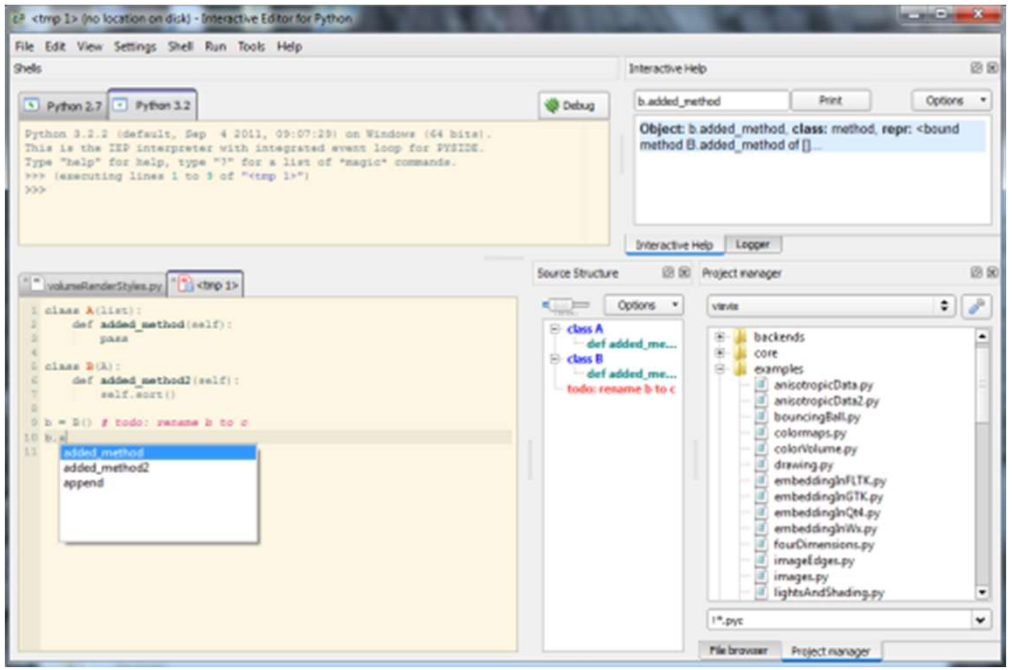
- O Linux pode correr ficheiros do Python como commandos num terminal.
- Para isso é preciso preparar algumas coisas:
  - O ficheiro tem que começar com a seguinte linha (*shebang*):
    - #!/bin/env python3
  - O ficheiro tem que ser classificado como executável:
    - [user@host] chmod 755 oMeuPrograma.py
    - Para correr basta colocar: ./oMeuPrograma.py
  - O caminho para o ficheiro tem que ser fornecido, ou figurar na variável PATH
    - [user@host] echo $PATH
    - [user@host] export PATH = /home/julia/programasPython :{PATH}

Correr *Scripts* no Linux

- O Linux pode correr ficheiros do Python como commandos num terminal.
- Para isso é preciso preparar algumas coisas:
  - O ficheiro tem que começar com a seguinte linha (*shebang*):
    - #!/bin/env python3
  - O ficheiro tem que ser classificado como executável:
    - [user@host] chmod 755 oMeuPrograma.py
    - Para correr basta colocar: ./oMeuPrograma.py
  - O caminho para o ficheiro tem que ser fornecido, ou figurar na variável PATH
    - [user@host] echo $PATH
    - [user@host] export PATH = /home/julia/programasPython :{PATH}
  - Altenativamente podemos correr o nosso programa assim:
    - python3 oMeuPrograma.py

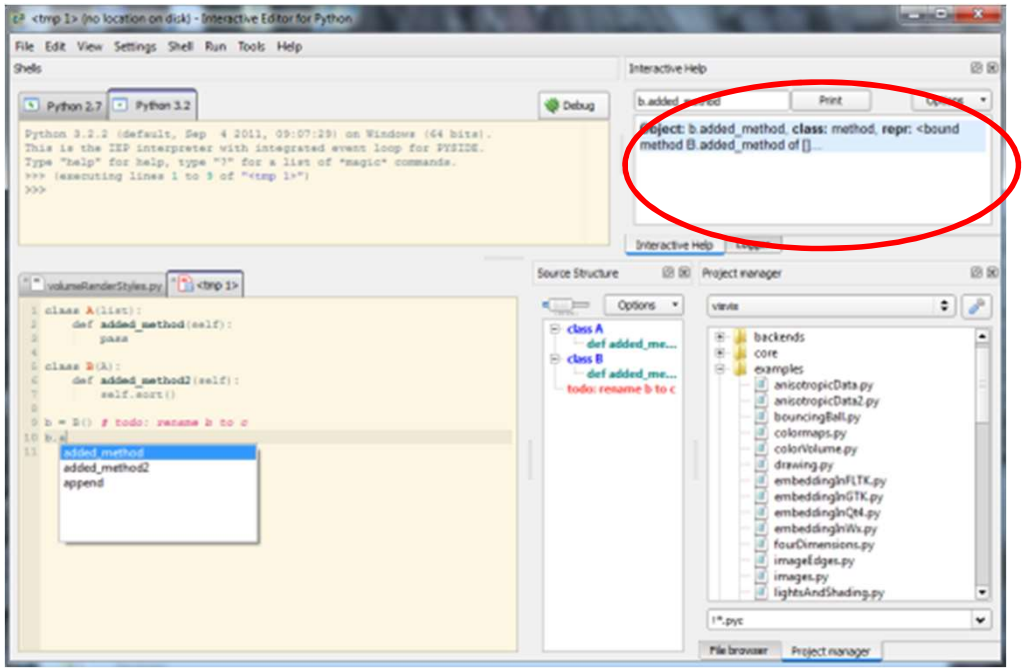
O Ambirnete Pyzo

- O ambiente Pyzo IDE (Integrated Development Environment) é um ambiente gráfico para o Python



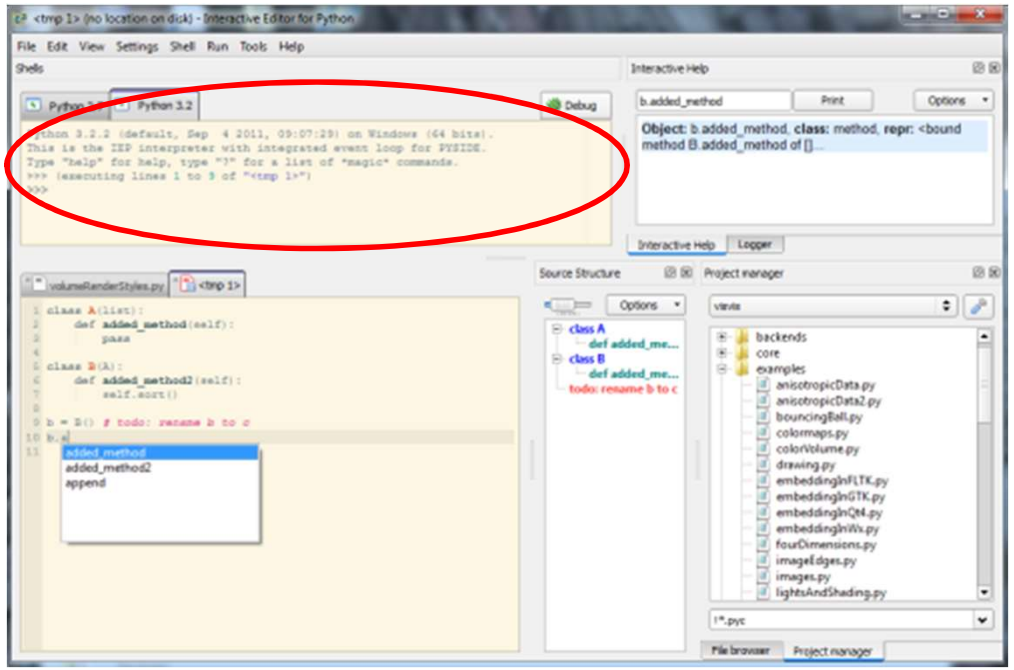
O Ambirnete Pyzo

- O ambiente Pyzo IDE (Integrated Development Environment) é um ambiente gráfico para o Python
- Inclui uma *shell* semelhante à que temos usado na linha de comandos



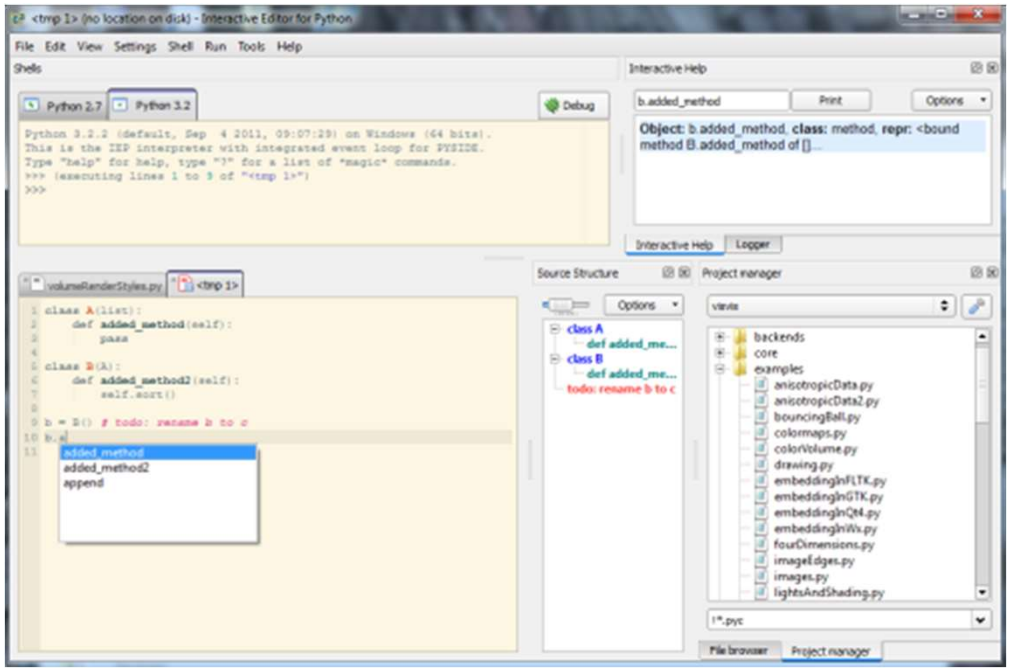
O Ambirnete Pyzo

- O ambiente Pyzo IDE (Integrated Development Environment) é um ambiente gráfico para o Python
- Inclui uma *shell* semelhante à que temos usado na linha de comandos
- Permite editar ficheiros .py com alguns confortos para o programador (indentação automática, código de cores, etc)



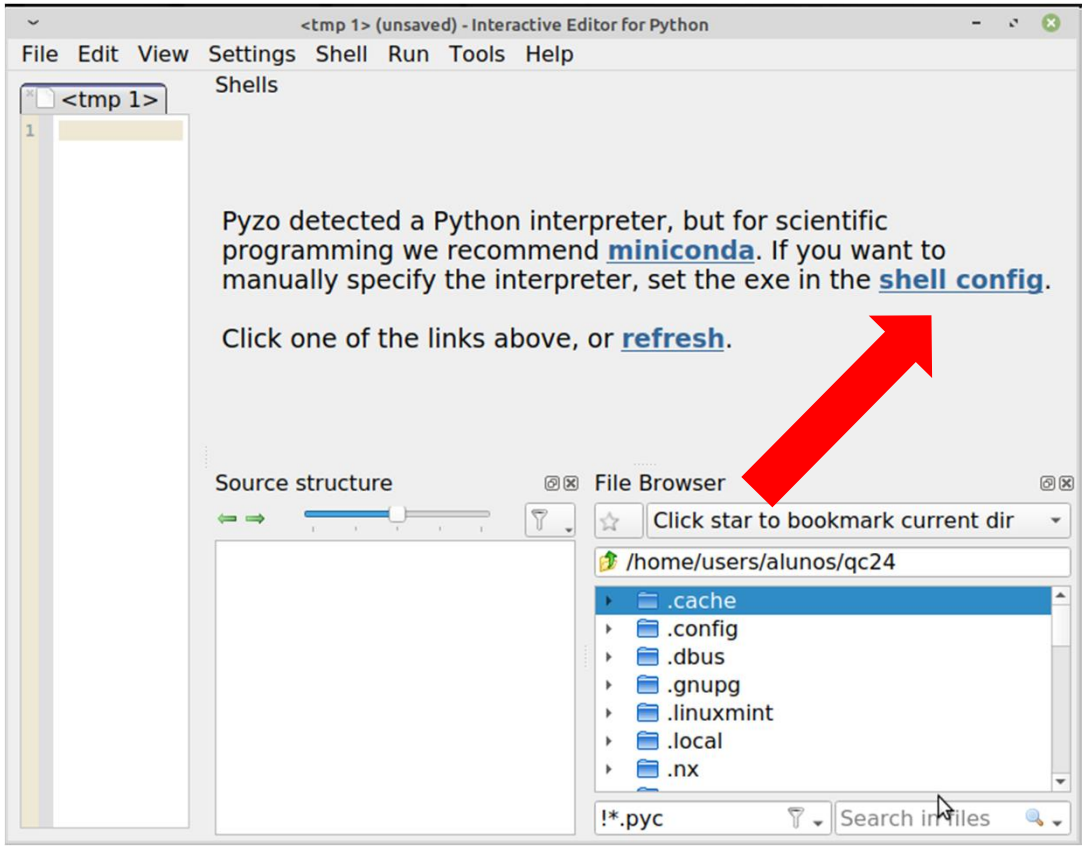
O Ambirnete Pyzo

- O ambiente Pyzo IDE (Integrated Development Environment) é um ambiente gráfico para o Python
- Inclui uma *shell* semelhante à que temos usado na linha de comandos
- Permite editar ficheiros .py com alguns confortos para o programador (indentação automática, código de cores, etc)
- Quando editamos um programa, podemos corre-lo directamente na *shell* premindo a Ctrl + E
- Mais sobre Pyzo: https://pyzo.org/



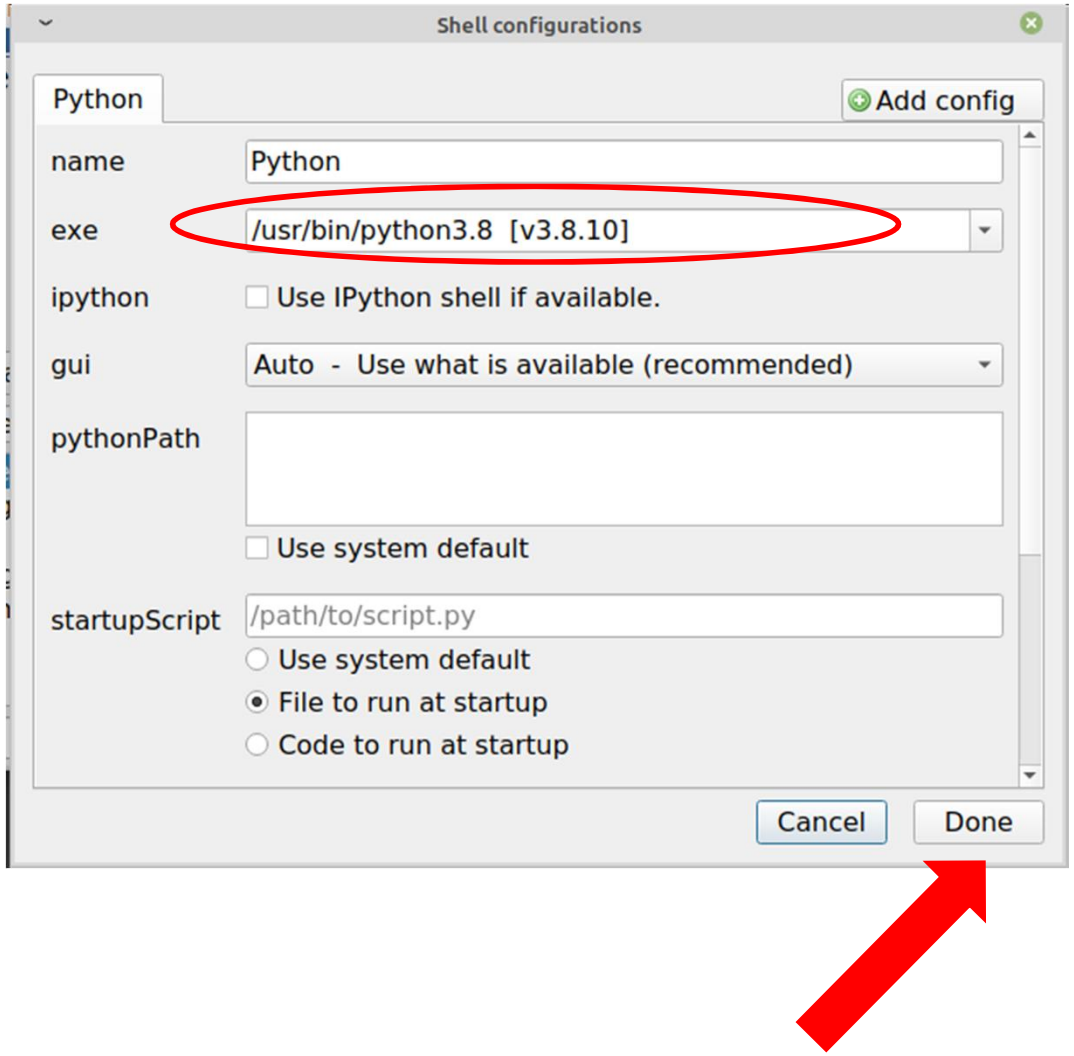
Ambiente Pyzo

- Para iniciar:
  - Escolher shell config



Ambiente Pyzo

- Para iniciar:
  - Escolher shell config
  - Escolher exe /usr/bin/python3.8
  - Carregar Done



Programa "Olá, mundo!"

Correr na shell do Pyzo e no terminal do Linux o programa "Olá, Mundo!"

Código

```
#! /bin/env python3
# -*- coding: utf8 -*-

# O meu primeiro programa em Python

print("Olá Mundo!")
```

Funções

- As funções são instruções que desejámos repetir em vários pontos do programa.

Funções

- As funções são instruções que desejámos repetir em vários pontos do programa.
- Tal como na matemática, podemos usar um ou mais argumentos para modificar o comportamento da função.

Exemplo

```
>>> def quadrado(x):
...     return(x**2)
...
>>> quadrado(3)
9
>>> quadrado(1232)
1517824
>>>
```

Funções

- As funções são instruções que desejámos repetir em vários pontos do programa.
- Tal como na matemática, podemos usar um ou mais argumentos para modificar o comportamento da função.
- As funções são definidas pela instrução def.

Exemplo

```
>>> def quadrado(x):
...     return(x**2)
...
>>> quadrado(3)
9
>>> quadrado(1232)
1517824
>>>
```

Funções

- As funções são instruções que desejámos repetir em vários pontos do programa.
- Tal como na matemática, podemos usar um ou mais argumentos para modificar o comportamento da função.
- As funções são definidas pela instrução **def** com ":" no final.
- O valor de retorno é definido dentro da função pela instrução **return** (facultativo).

Exemplo

```
>>> def quadrado(x):
...     return(x**2)
...
>>> quadrado(3)
9
>>> quadrado(1232)
1517824
>>>
```

Funções

- As funções podem ser usadas para repetir bocados relativamente grandes de código.
- É boa prática dividir o código em várias funções pequenas.
  - Torna o código mais fácil de manter.
  - Podemos re-utilizar funções.

Exemplo

```
>>> def prettyPrint(i):
...     istr=str(i)
...     border = ""
...     for n in range(len(istr)+4):
...         border = border + "*"
...     print(border)
...     print("* "+istr+" *")
...     print(border)
...
>>> prettyPrint(666)
*******
* 666 *
*******
>>> prettyPrint("Olá Mundo")
*************
* Olá Mundo *
*************
```

Funções

- **Recursão:** As funções podem chamar outras funções, incluindo elas próprias.
- **docStrings** Uma cadeia de caracteres na primeira linha da função é usada para documentar a mesma.
- Na *shell*, usamos a função help() para consultar a documentação sobre cada função.

Exemplo

```
>>> def factorial(n):
...     """Retorna o factorial de n"""
...     if(n<=0):
...         return(0)
...     elif(n==1):
...         return(1)
...     else:
...         return(n*factorial(n-1))
...
>>> factorial(3)
6
>>> factorial(9)
362880
>>> help(factorial)
factorial(n)
    Retorna o factorial de n
>>>
```

Exercício de Aplicação

Criar e correr um programa que verifica se um número introduzido pelo utilizador pertence à série de Fibonacci:

$$F_0 = 0; F_1 = 1; F_n = F_{n-1} + F_{n-2}$$

Nota: a série é 1, 1, 2, 3, 5, 8, 13...

Módulos

- Módulos são extensões da linguagem básica para aplicações específicas.
- Alguns módulos comuns:
  - os e sys — Interacção com o sistema.
  - numpy — Aplicações numéricas (cálculo, álgebra, estatística, etc..)
  - matplotlib — Gráficos
  - pandas — Ler e trabalhar com tabelas.
- Cada módulo é carregado com a instrução import.

```
>>> import os
>>> os.system("ls -lh")
total 4424156
drwx------ 35 julia julia       4096 Sep 28 11:44 .
drwxr-xr-x  9     0     0         99 Sep 26 18:53 ..
-rw-rw-r--  1 julia julia          0 May  9 12:18 0_STEEP.mdp
-rw-rw-r--  1 julia julia          0 May  9 12:18 1_NPT_eq.mdp
-rw-rw-r--  1 julia julia          0 May  9 12:18 1_NVT_eq.mdp
drwxrwxr-x 28 julia julia       4096 May 31  2022 anaconda3
```

Módulos

- O conteúdo de cada módulo pode ser inspecionado com o comando dir:

```
>>> import sys
>>> dir(sys)
['__breakpointhook__', '__displayhook__', '__doc__',
(...)
'stdout', 'thread_info', 'version', 'version_info', 'warnoptions']
```

- O nome do módulo serve de prefixo para os seus conteúdos:

```
>>> os.mkdir("teste_python")
>>> sys.version
'3.7.4 (default, Jul 16 2019, 07:12:58) \n[GCC 9.1.0]'
```

- Podemos mudar o nome do módulo usando a instrução import ...as ...

```
>>> import numpy as np
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> np.array([[2.3,5.0,6.0],[1.0,2.0,1.0],[0,5.0,6]])
array([[2.3, 5. , 6. ],
       [1. , 2. , 1. ],
       [0. , 5. , 6. ]])
```

O comando dir

- O comando dir é normalmente usado na *shell*.
- Sem argumentos, dir() mostra os objectos na memória (variáveis, funções definidas, móulos importados, etc...)
- Usando qualquer objecto como argumento, dir() mostra os métodos disponíveis para esse objecto.

Métodos das cadeias de caracteres (texto)

```
>>> a="texto de exemplo"
>>> dir(a)
'capitalize', 'casefold', 'center', (...),
['__add__', '__class__', (...),
'capitalize', 'casefold', 'center',(...),
'isspace', 'istitle', 'isupper', 'join',(...)
'replace', 'rfind', 'rindex', 'rjust', (...)
'title', 'translate', 'upper', 'zfill']
>>> a.isspace()
False
>>> a.upper()
'TEXTO DE EXEMPLO'
>>> a.capitalize()
'Texto de exemplo'
>>> a.replace("texto de","apenas um")
'apenas um exemplo'
```

O comando dir

- O comando dir é normalmente usado na *shell*.
- Sem argumentos, dir() mostra os objectos na memória (variáveis, funções definidas, móulos importados, etc...)
- Usando qualquer objecto como argumento, dir() mostra os métodos disponíveis para esse objecto.
- Também pode-se usar no Pyzo

Métodos das listas

```
>>> dir(b)
[(...), 'append', 'clear', 'copy',
'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>> b.sort()
>>> b
[1, 2, 5]
>>> b.append(-5)
>>> b
[1, 2, 5, -5]
>>> b.reverse()
>>> b
[-5, 5, 2, 1]
```

O Comando help

- O comando help imprime a mensagem de ajuda disponível para um dado objecto, função ou método.

Exemplo

```
>>> a="texto de exemplo"
>>> b = [1, 5, 2]
>>> help(a.title)
Help on built-in function title:

title() method of builtins.str instance
    Return a version of the string where each word is titlecased.

    More specifically, words start with uppercased characters and all remaining
    cased characters have lower case.
>>> help(b.append)
Help on built-in function append:

append(object, /) method of builtins.list instance
    Append object to the end of the list.
```

O Comando help

- O comando **help** imprime a mensagem de ajuda disponível para um dado objecto, função ou método.

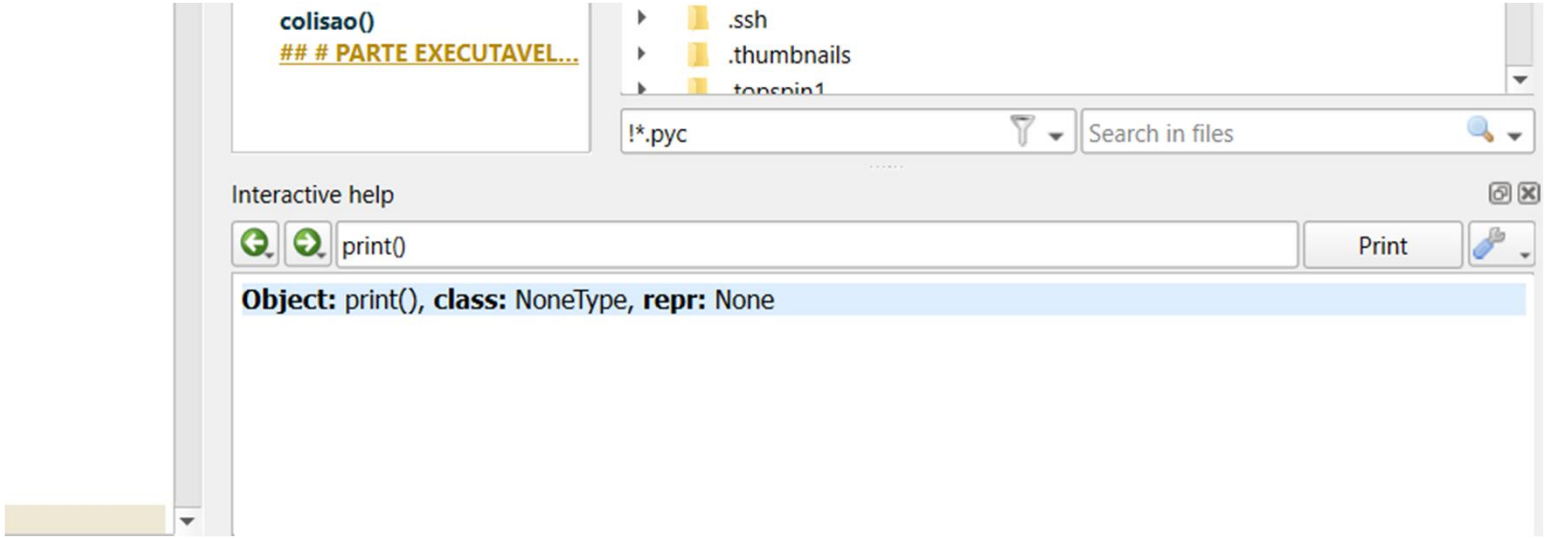
Exemplo

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Interactive help no Pyzo

- No Pyzo existe Interactive help:

  Tools/Interactive help



Listas e *Arrays* do módulo Numpy

- As listas são um tipo de variável que permite guardar uma sequência de valores de qualquer outro tipo (incluindo outras listas!)
- As *arrays* são um tipo de variável específico do módulo Numpy para representar vectores e matrizes.
- Podemos converter listas em *arrays* usando a instrução array do Numpy (numpy.array)
- As listas possuem métodos mais direccionados com a ordem, acréscimo e remoção de elementos.
- As *arrays* têm métodos mais relacionados com o processamento numérico.

Exemplo

```
>>> import numpy as np
>>> b = [1, 5, 2]
>>> myArray = np.array(b)
>>> myArray
array([1, 5, 2])
>>> dir(myArray)
['T', (...), 'all', 'any', 'argmax', (...),
'max', 'mean', 'min', 'nbytes', (...),
'prod', 'ptp', 'put', 'ravel', 'real', (...),
'sum', 'swapaxes', 'take', 'tobytes', (...),
'tolist', 'tostring', 'transpose',(...)]
>>> myArray.sum()
8
>>> myArray.prod()
10
>>> myArray.tolist()
[1, 5, 2]
```

Projecot Estatística

Usando o Pyzo, crie um programa (stats.py) em Python que calcule algumas estatísticas básicas com base em dados introduzidos pelo utilizador. Em particular o programa deve:

- Pedir um número de cada vez ao utilizador.
- Caso o utilizador escreva a letra "q", o programa deve parar de pedir novos dados e proceder ao calculo.
- O programa deverá imprimir no ecrã a média, variância e desvio padrão para os dados introduzidos.
- **Bónus**: O programa deverá também imprimir a mediana

Projecto Estatística

Média

$$A = \frac{1}{n}\sum_{i=1}^{n} a_i$$

A – média
a_i – valores individuais
n – número de valores

Variância

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n}$$

S^2 – variância
x_i – valores individuais
$\bar{x}$ – média
n – número de valores

Desvio padrão

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

σ – desvio padrão
x_i – valores individuais
μ – média
N – número de valores

Mediana

$$\mathrm{Med}(X) = \begin{cases} X[\frac{n+1}{2}] & \text{if n is odd} \\ \frac{X[\frac{n}{2}]+X[\frac{n}{2}+1]}{2} & \text{if n is even} \end{cases}$$

X – lista ordenada de valores
n – número de valores

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever código em Python.

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever código em Python.
- **Funções:** Blocos de código que podemos reutilizar (def e return).

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever código em Python.
- **Funções:** Blocos de código que podemos reutilizar (def e return).
- **Módulos**: Pacotes que extendem a funcionalidade básica do Python (import).

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever código em Python.
- **Funções:** Blocos de código que podemos reutilizar (def e return).
- **Módulos**: Pacotes que extendem a funcionalidade básica do Python (import).
- **Numpy:** Módulo para trabalhar com dados numéricos (import numpy as np).

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever código em Python.
- **Funções:** Blocos de código que podemos reutilizar (def e return).
- **Módulos**: Pacotes que extendem a funcionalidade básica do Python (import).
- **Numpy:** Módulo para trabalhar com dados numéricos (import numpy as np).
- **dir()**: Saber que objectos e métodos estão disponíveis.

Em resumo...

- **Repetições**: Os ciclos **while** são usados quando não sabemos quantas iterações são necessárias até que uma dada condição seja verdadeira.
- **Correr no Linux:** Verificar a *shebang*, o estatudo de executável e a variável $PATH!
- **Pyzo:** Uma boa ferramenta para escrever c´odigo em Python.
- **Funções:** Blocos de código que podemos reutilizar (def e return).
- **Módulos**: Pacotes que extendem a funcionalidade básica do Python (import).
- **Numpy:** Módulo para trabalhar com dados numéricos (import numpy as np).
- **dir()**: Saber que objectos e métodos estão disponíveis.
- **help()**: Imprime a documentação disponível para um dado método ou função.