

# Curso de Iniciação à Programação em Python para Profissionais de Saúde



Luís Vieira e José Ferrão

Instituto Nacional de Saúde Doutor Ricardo Jorge

17-26 Março 2025

# Objetivos

- Introduzir os conceitos de pensamento computacional e de programação
- Ensinar a programar utilizando como modelo a linguagem *Python*
- Construir pequenos programas em *Python* para automatização de tarefas
- Demonstrar a potencialidade do *Python* na resolução de diferentes tipos de problemas biológicos
- Promover a utilização da programação como ferramenta corrente no desenvolvimento de estudos, projectos ou actividades diárias
- Incentivar a auto-aprendizagem do *Python* para desenvolver as capacidades de programação a nível pessoal

# Calendarização

- O curso é composto por 6 sessões de 3h30m cada, distribuídas pelas seguintes datas:
  - 1ª semana: Dias 17, 18 e 19 de Março
  - 2ª semana: Dias 24, 25 e 26 de Março
- Cada sessão será estruturada da seguinte forma:
  - Componente teórica
  - Exercícios a realizar no computador
  - *Coffee-break* (~10 minutos)
- As sessões têm um carácter contínuo, ou seja, os conhecimentos adquiridos numa sessão serão utilizados na sessão seguinte.

# Programa geral

- Sessão 1: Iniciação ao *Python*
- Sessão 2: Expressões condicionais e iteração
- Sessão 3: Objectos estruturados
- Sessão 4: Funções
- Sessão 5: Módulos
- Sessão 6: Números aleatórios

# Sessão 1: Iniciação ao *Python*

- **Conteúdos:**

- Conceitos básicos sobre pensamento computacional e programação
- A programação em linguagem *Python*:
  - O interpretador do *Python*
  - Sintaxe e semântica
  - Tipos de objectos (números inteiros, *floats* e *strings*)
  - Operações matemáticas básicas
  - Valores booleanos e operadores
  - Funções *built-in*
  - Variáveis e atribuição
  - Operações com *strings* (concatenação)

# Pensamento computacional

- **Declarativo:** Composto por declarações de facto
  - Ex: “A raiz quadrada de  $x$  é um número  $y$  tal que  $y*y=x$ .”
- **Imperativo:** Composto por instruções (“receitas”) para obter informação
  - Ex: Calcular a raiz quadrada de um número  $x$ :
    - Começar com um número  $y$  qualquer
    - Se  $y*y$  está próximo de  $x$ , terminar e dizer que  $y$  é a resposta
    - Caso contrário criar um novo número  $y$  fazendo a média de  $y$  e  $x/y$ , ou seja,  $(y + x/y)/2$
    - Usando este novo número  $y$ , repetir o processo até que  $y*y$  esteja próximo de  $x$

# Algoritmo

- Um algoritmo é uma lista finita de instruções que descrevem uma **computação**:

1. Conjunto de dados fornecidos (“input”)



2. Conjunto bem definido de estados (“program”)  
(inclui testes e ordem de execução das instruções)



3. Conjunto de resultados/respostas (“output”)

# O que é a programação?

- A programação é o processo de interagir com o *hardware* do computador e dar instruções para que este execute uma determinada tarefa
- A programação é uma componente fundamental de vários dispositivos electrónicos, por exemplo, telemóveis, dispositivos GPS, *smart TVs*, consolas de jogos, etc.
- Para criar um programa é necessário usar uma **linguagem de programação** para descrever a sequência de instruções, ou seja, uma forma de dizer ao computador o que deve fazer



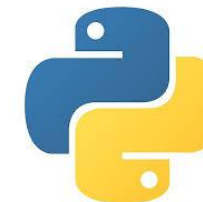
# Linguagens de programação

- Existem muitas linguagens de programação, algumas das quais estão direccionadas para certos tipos de aplicações (por exemplo, o Java é muito utilizado em páginas *web* e em televisões, enquanto os jogos usam frequentemente o C++)
- Uma linguagem de programação é como outra linguagem qualquer, que também usa números, letras e sinais de pontuação
- As linguagens de programação têm um vocabulário reduzido e são mais restritivas e objectivas que as linguagens faladas

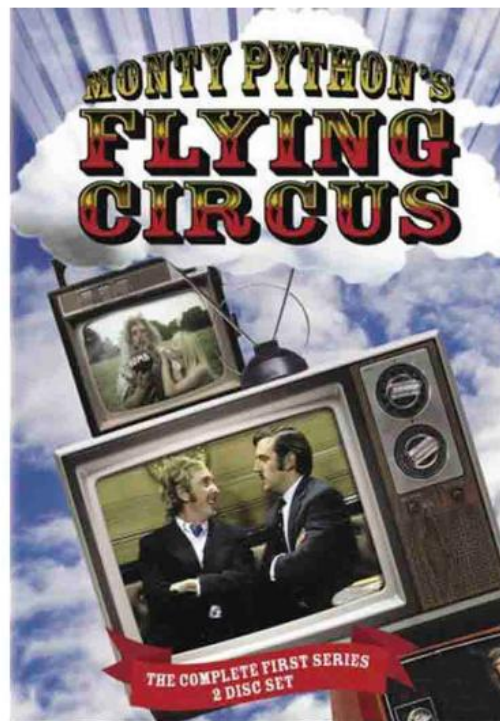
# Linguagem natural vs. programação

Linguagem natural	Linguagem de programação
Vocabulário extenso	Vocabulário reduzido
Subjectividade (ex., “O céu está meio nublado”)	Objectividade (ex: “O céu está nublado”)
Existência de sinónimos (ex: imprimir, publicar, estampar)	Palavras únicas (ex: ‘print’)
Inexactidão (ex: “O candidato teve cerca de 50% dos votos”)	Exactidão (ex: “O candidato teve 50% dos votos”)
Erros ortográficos não bloqueiam a leitura (ex: “Noz adoramos o Python!”)	Erros ortográficos não são permitidos
Letra maiúscula ou minúscula não altera o significado da palavra	Natal, natal e NATAL são 3 palavras com significados distintos

# O que é o *Python*?



- Linguagem de programação concebida no início dos anos 90 por Guido van Rossum do *Stichting Mathematisch Centrum* (Holanda)
- Linguagem de utilização transversal e considerada ideal para aprender a programar computadores
- Caracterizado como uma “linguagem de alto nível”, i.e., uma linguagem fácil de ler e de compreender
- O *Python* é de acesso livre e pode ser instalado em diferentes plataformas (Windows, macOS e Linux)
- Existem 2 versões activas do *Python* (2.X e 3.X), que podem ser obtidas a partir da página oficial do *Python*: <https://www.python.org/downloads/>

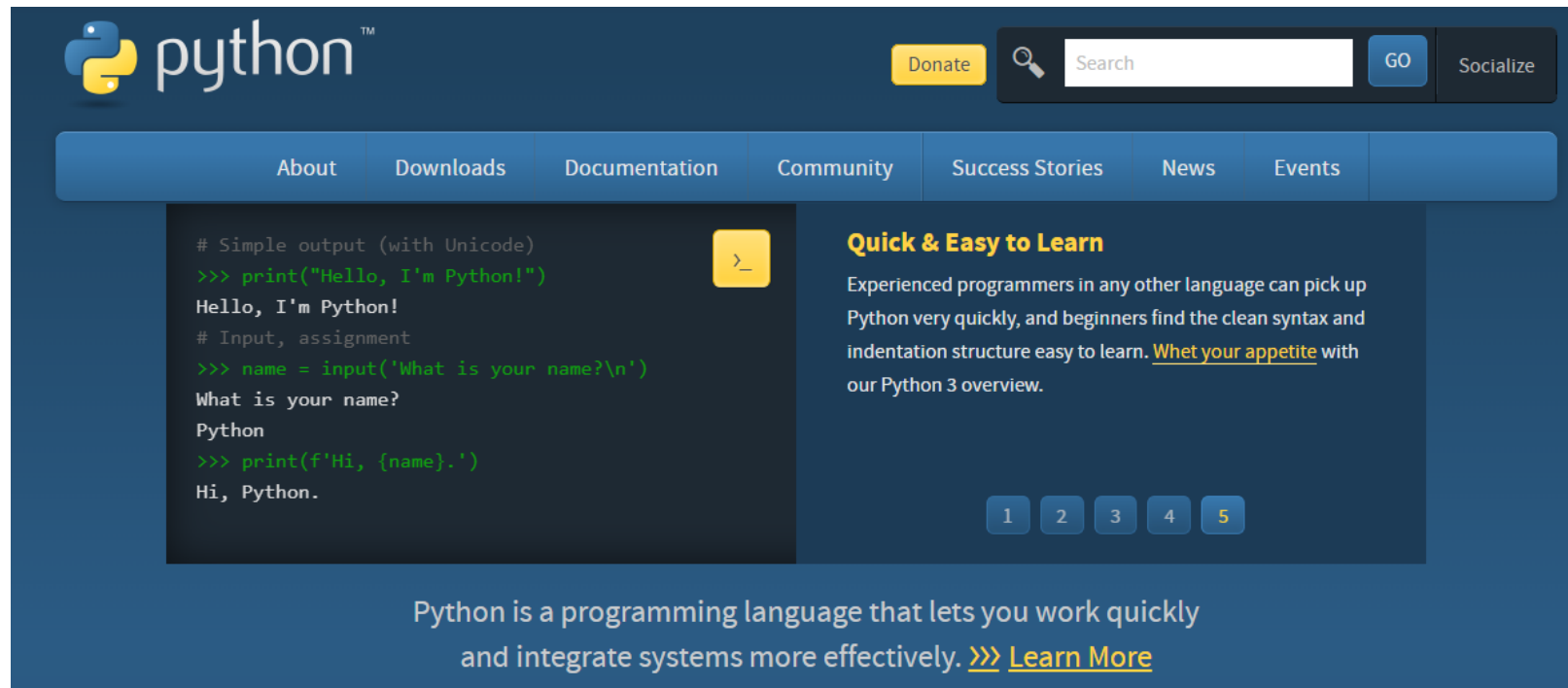


<http://www.montypython.com>



[https://en.wikipedia.org/wiki/Ball\\_python](https://en.wikipedia.org/wiki/Ball_python)

# Python software foundation



The screenshot shows the Python.org homepage with a dark blue header. The Python logo and 'python' text are on the left. On the right, there is a 'Donate' button, a search bar with a magnifying glass icon, a 'GO' button, and a 'Socialize' button. Below the header is a navigation bar with links: About, Downloads, Documentation, Community, Success Stories, News, and Events. The main content area is split into two columns. The left column features a code editor with a yellow prompt icon, showing a Python script that prints 'Hello, I'm Python!' and asks for a name. The right column has the heading 'Quick & Easy to Learn' followed by a paragraph about Python's ease of learning and a link to 'Whet your appetite'. Below this is a row of five numbered buttons (1-5). At the bottom, a blue banner contains the text: 'Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)'.

# Aplicações do *Python*

- Desenvolvimento da *web* e da *internet*
  - desenvolvimento de aplicações *web* em servidores, organização e publicação de conteúdos, etc.
- Computação científica e numérica
  - matemática, geometria, estatística, *machine-learning*, *data mining*, modelação, etc.
- Educação
  - ensino da programação em escolas primárias e secundárias, e universidades
- Desenvolvimento de *software*
  - ferramentas para construção e compilação de *software*, detecção de *bugs*, etc.
- Aplicações comerciais
  - *software* para gestão de empresas

# Requisitos necessários

- Computador com sistema operativo Windows ( $\geq$ XP), Apple MAC ou Linux
- *Integrated Developer Environment* (IDE), para escrever e executar o código *Python*
- Um editor de texto (editor de texto do *Python* – *IDLE* ou o *Notepad++*, por exemplo)
- Acesso à internet (para actualizações, acesso a bibliotecas de programas, tutoriais, fóruns de utilizadores, etc.)



# A linguagem *Python*

- A linguagem *Python* é caracterizada por um conjunto de construtores bem definidos e 3 níveis distintos de construção da linguagem.
- A utilização de outros construtores ou a inclusão de erros sintáticos ou semânticos produzem erros nos programas.
- **Construtores primitivos:** Constituem as “palavras” da linguagem
  - Literais: Um literal é um valor que é fixo e não se altera com o programa
    - Números inteiros, decimais (“floats”) e complexos
    - Cadeias de caracteres (“strings”), colocadas entre aspas ou plicas. Exemplo: “víbora”
    - Booleanos (*true* e *false*)
    - Especiais (*none* – indica a ausência de um valor ou uma referência nula)
  - Operadores (ex., sinais + e /)

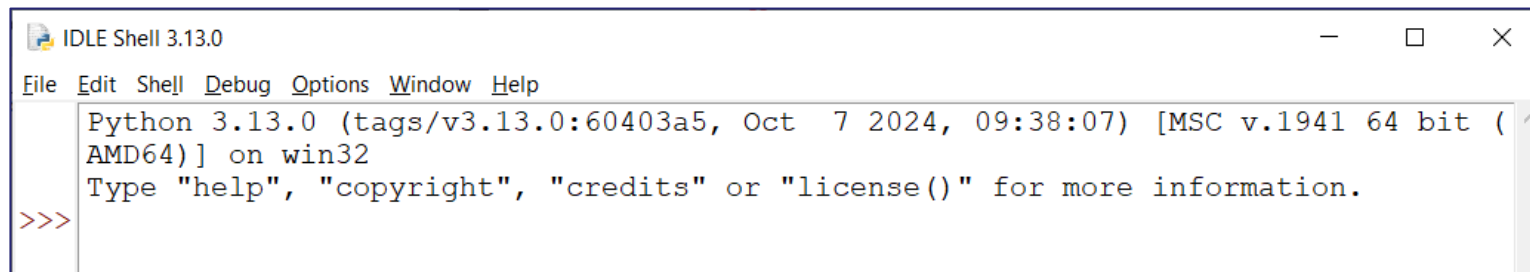


# A linguagem *Python*

- **Sintaxe:** Define quais as cadeias de caracteres e símbolos que estão bem formadas
  - Ex: A frase “A víbora uma cobra venenosa” não está sintaticamente correta
- **Semântica estática:** Define as cadeias de caracteres e símbolos que são sintaticamente válidas e têm um sentido
  - Ex: A frase “A víbora são uma cobra venenosa” não tem sentido
- **Semântica:** Associa um significado com cada cadeia de caracteres e símbolos que é sintaticamente correta e que não possui erros de semântica estática
  - Ex: A frase “Esta é uma víbora” pode referir-se a uma víbora ou a uma pessoa que é muito “venenosa”

# Iniciar o *Python*

- Na caixa de pesquisa do *Windows*, escrever “IDLE”; clicar sobre “IDLE (Python 3.13.0)” para abrir a *Shell* do *Python*
- A *Shell* é o local onde se pode introduzir o código do *Python* e ver o resultado da execução desse código
- A *Shell* está desenhada para a introdução de comandos curtos e não para programas compostos por múltiplas linhas



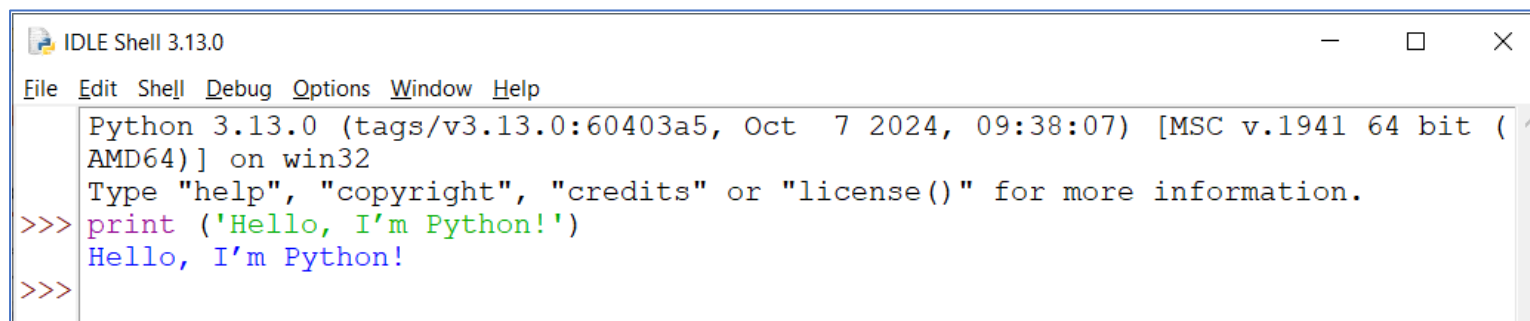
```

IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
    
```

# Primeiro código *Python*

- O código é introduzido após os 3 operadores “>>>” (*prompt*)
- O resultado da execução do código é apresentado na linha imediatamente abaixo
- Vamos introduzir a primeira linha de código e ver o resultado:

```
>>> print ('Hello, I'm Python!')
```



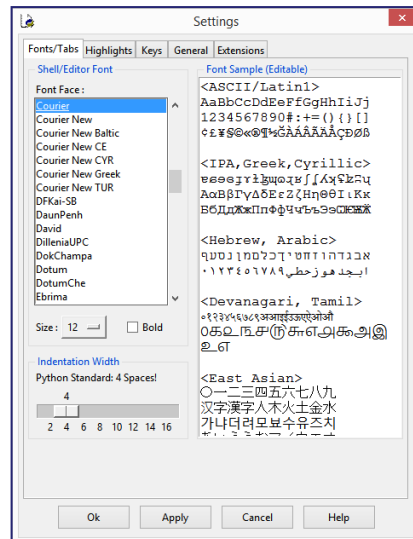
```

IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print ('Hello, I'm Python!')
Hello, I'm Python!
>>>
    
```

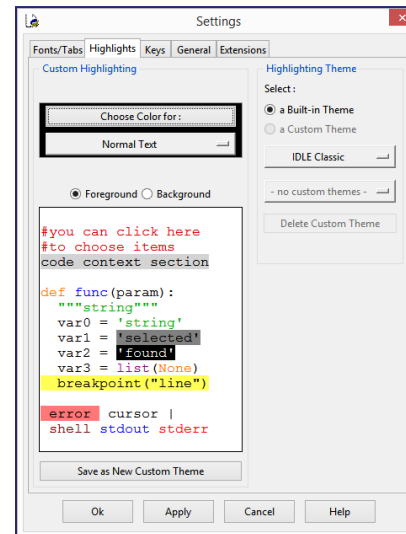
# Configurar o *Python*

- Aceder a *Options > Configure IDLE*, para configurar o ambiente de trabalho:
  - Separador “Fonts/Tabs”: Alterar o tipo e tamanho de letra
  - Separador “Highlights”: Alterar a codificação de cores
  - Separador “Keys”: Para ver/alterar as teclas de atalho do *Python*

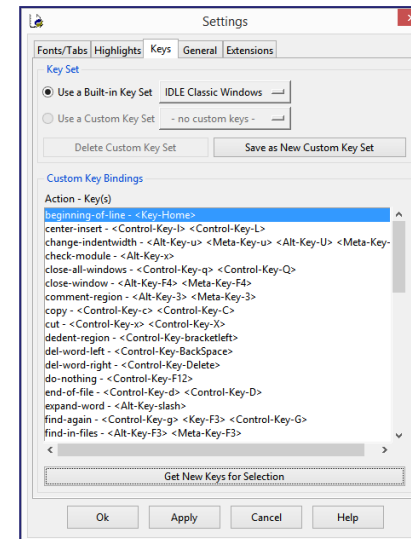
Fonts/Tabs



Highlights



Keys



# Objectos do *Python*

- Objectos escalares (sem estrutura interna)
  - Números
  - *Floats*
  - Valores booleanos
- Objectos estruturados
  - *Strings*
  - Tuplos
  - Listas
  - Dicionários

# Números e expressões matemáticas

- O *Python* consegue efectuar várias operações matemáticas, por exemplo:
  - Adição: `>>> 1+1`
  - Adição: `>>> 3528682+4691273`
  - Subtracção: `>>> 1-1`
  - Subtracção: `>>> 3528682-4691273`
  - Multiplicação: `>>> 1*1`
  - Multiplicação: `>>> 3528682*4691273`
  - Divisão: `>>> 1/1`
  - Divisão: `>>> 3528682/4691273`
- A divisão produz um número decimal, que se designa no *Python* por ***float*** ou ***floating-point arithmetic***

# Números e expressões matemáticas

- Para se obter um número inteiro na divisão tem de se usar a dupla barra à direita (//):
  - Divisão: `>>> 3//2`
  - Divisão: `>>> 10//5`
- O resto da divisão pode ser obtido da seguinte forma:
  - Divisão: `>>> 5/3`
  - Resto: `>>> 5%3`
- A exponencial (potência) é obtida usando duplo asterisco:
  - 3 elevado a 2: `>>> 3**2`

# Exercícios

- Efectue as seguintes operações matemáticas na *Shell* do *Python* e interprete os resultados:

```
>>> 3+2
```

```
>>> 3.0+2.0
```

```
>>> 8-5
```

```
>>> 2*5
```

```
>>> 2.0*5.0
```

```
>>> 456378967*823567892
```

```
>>> 6//3
```

```
>>> 6//4
```

```
>>> 6/4
```

```
>>> 6%4
```

```
>>> 4**2
```

```
>>> 4.0**2.0
```

```
>>> 'a'*3
```

```
>>> 'a'**2
```

```
>>> 2+2*3
```

```
>>> (2+2)*3
```



# Funções *built-in*

- O interpretador do *Python* contém um conjunto de funções e tipos que podem ser usadas sempre que necessário
- As palavras das funções *built-in* têm uma cor identificadora própria
- Os valores a colocar dentro dos parêntesis podem ser de vários tipos (números, *strings*, etc.) e designam-se como **argumentos** da função

Built-in Functions			
<b>A</b> <a href="#">abs()</a> <a href="#">aiter()</a> <a href="#">all()</a> <a href="#">anext()</a> <a href="#">any()</a> <a href="#">ascii()</a>	<b>E</b> <a href="#">enumerate()</a> <a href="#">eval()</a> <a href="#">exec()</a>	<b>L</b> <a href="#">len()</a> <a href="#">list()</a> <a href="#">locals()</a>	<b>R</b> <a href="#">range()</a> <a href="#">repr()</a> <a href="#">reversed()</a> <a href="#">round()</a>
<b>B</b> <a href="#">bin()</a> <a href="#">bool()</a> <a href="#">breakpoint()</a> <a href="#">bytearray()</a> <a href="#">bytes()</a>	<b>F</b> <a href="#">filter()</a> <a href="#">float()</a> <a href="#">format()</a> <a href="#">frozenset()</a>	<b>M</b> <a href="#">map()</a> <a href="#">max()</a> <a href="#">memoryview()</a> <a href="#">min()</a>	<b>S</b> <a href="#">set()</a> <a href="#">setattr()</a> <a href="#">slice()</a> <a href="#">sorted()</a> <a href="#">staticmethod()</a> <a href="#">str()</a> <a href="#">sum()</a> <a href="#">super()</a>
<b>C</b> <a href="#">callable()</a> <a href="#">chr()</a> <a href="#">classmethod()</a> <a href="#">compile()</a> <a href="#">complex()</a>	<b>G</b> <a href="#">getattr()</a> <a href="#">globals()</a>	<b>N</b> <a href="#">next()</a>	<b>T</b> <a href="#">tuple()</a> <a href="#">type()</a>
<b>D</b> <a href="#">delattr()</a> <a href="#">dict()</a> <a href="#">dir()</a> <a href="#">divmod()</a>	<b>H</b> <a href="#">hasattr()</a> <a href="#">hash()</a> <a href="#">help()</a> <a href="#">hex()</a>	<b>O</b> <a href="#">object()</a> <a href="#">oct()</a> <a href="#">open()</a> <a href="#">ord()</a>	<b>V</b> <a href="#">vars()</a>
	<b>I</b> <a href="#">id()</a> <a href="#">input()</a> <a href="#">int()</a> <a href="#">isinstance()</a> <a href="#">issubclass()</a> <a href="#">iter()</a>	<b>P</b> <a href="#">pow()</a> <a href="#">print()</a> <a href="#">property()</a>	<b>Z</b> <a href="#">zip()</a>  <a href="#">__import__()</a>

# Funções *built-in*

- Algumas das funções matemáticas *built-in* do *Python* são as seguintes:
  - **abs** (**x**): devolve o valor absoluto de um número (inteiro ou *float*)  
`>>> abs (10.0)`
  - **divmod** (**x**, **y**): recebe 2 números (x e y) como argumentos e devolve um par de números composto pelo seu quociente e o resto da divisão inteira  
`>>> divmod (5, 3)`
  - **max** (**x**, **y**, **z**, ...): devolve o valor mais elevado de 2 ou mais argumentos  
`>>> max (3, 7, 9.1, 5.3, 9.2, 0.1)`
  - **min** (**x**, **y**, **z**, ...): devolve o valor mais baixo de 2 ou mais argumentos  
`>>> min (3, 7, 9.1, 5.3, 9.2, 0.1)`
  - **round** (**x** [, **ndigits**]): devolve o número x arredondado a *ndigits* após o ponto decimal (se *ndigits* não é especificado, devolve o inteiro mais próximo de x)  
`>>> round (3.41, 1)`

# Exercícios

- Execute as seguintes funções na *Shell* do *Python* e interprete os resultados:

```
>>> abs (-3)
```

```
>>> abs (-10.7235876)
```

```
>>> divmod (6,3)
```

```
>>> max (6.0, 4.5, 5.6, 3.6, 7.8, 10.2, 9.9, 10.5, 3.5)
```

```
>>> min (6.0, 4.5, 5.6, 3.6, 7.8, 10.2, 9.9, 10.5, 3.5)
```

```
>>> round (-10.7235876, 4)
```

```
>>> float (324)
```

```
>>> int (-10.7235876)
```

```
>>> help ()
```

```
>>> help (abs)
```

# Expressões booleanas

- Uma expressão booleana é uma declaração lógica que é avaliada como verdadeira (True) ou falsa (False)
- As expressões booleanas são construídas usando operadores matemáticos:
  - > (maior que)
  - >= (maior ou igual que)
  - < (menor que)
  - <= (menor ou igual que)
  - == (igual a)
  - != (diferente de)

# Exercícios

- Efectue as seguintes operações lógicas na *Shell* do *Python* e interprete os resultados:

```
>>> 5 > 2
```

```
>>> 4 < 3
```

```
>>> 5>=4
```

```
>>> 5>=5.0
```

```
>>> 3<=2
```

```
>>> 3==3
```

```
>>> 3==3.0
```

```
>>> 3!=3
```

```
>>> 3!=3.0
```

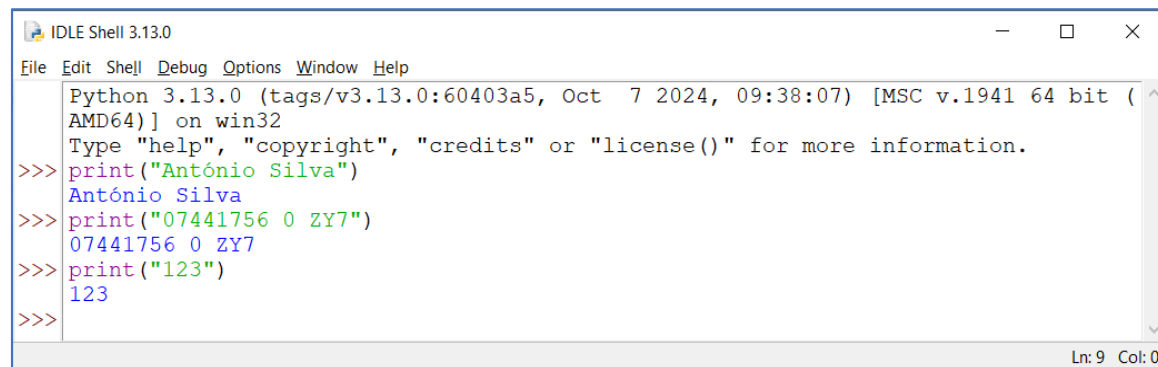
# Strings

- Uma *string* é uma variável composta por múltiplos caracteres, por exemplo:
  - Nome de pessoa (contém apenas letras)
  - N° do cartão de cidadão (contém números, letras e espaços)
  - Número
- As *strings* são colocadas entre aspas (“abc”) ou plicas (‘abc’)
- Exemplos:

```
>>> print (“António Silva”)
```

```
>>> print (“07441756 0 ZY7”)
```

```
>>> print (“123”)
```



```
IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("António Silva")
António Silva
>>> print("07441756 0 ZY7")
07441756 0 ZY7
>>> print("123")
123
>>>
```

# Variáveis e atribuição

- Uma **variável** é uma forma de associar nomes com objectos. No *Python*, uma variável é **apenas um nome**
- O nome de uma variável pode conter letras (maiúsculas e minúsculas), números (mas não pode começar com um número) e o símbolo “\_”
- Uma declaração de **atribuição** associa o nome à esquerda do símbolo igual (=) com o objecto à direita desse símbolo. Por exemplo:

```
>>> mês = "Janeiro"
```

```
>>> ano = 2019
```

# Variáveis e atribuição

- Suponhamos que definimos as seguintes variáveis:

```
>>> nome = "Oswaldo"
```

```
>>> apelido = "Cruz"
```

```
>>> profissao = "médico de saúde pública"
```

```
>>> data_nascimento = 1872
```

- O tipo das variáveis pode ser determinado usando a função *type()*:

```
>>> type (nome)           —————→  string
```

```
>>> type (apelido)        —————→  string
```

```
>>> type (profissao)      —————→  string
```

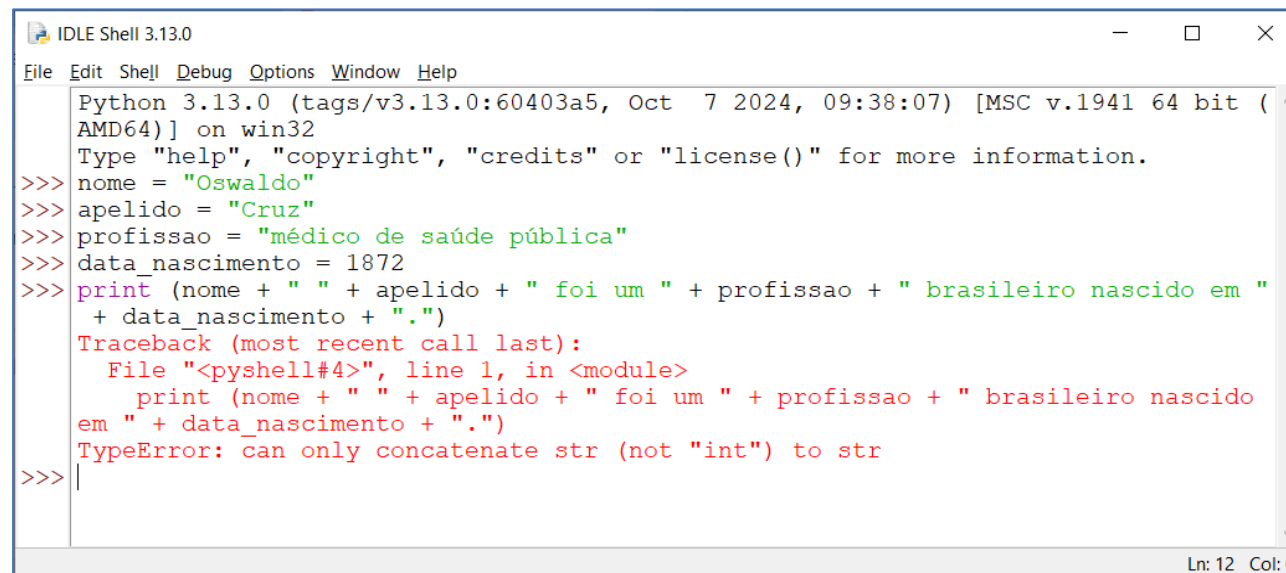
```
>>> type (data_nascimento) —————→  int
```



# Concatenação de *strings*

- As variáveis criadas anteriormente podem ser combinadas para apresentar a seguinte frase: "Oswaldo Cruz foi um médico de saúde pública brasileiro nascido em 1872.":

```
>>> print (nome + " " + apelido + " foi um " + profissao + " brasileiro
nascido em " + data_nascimento + ".")
```

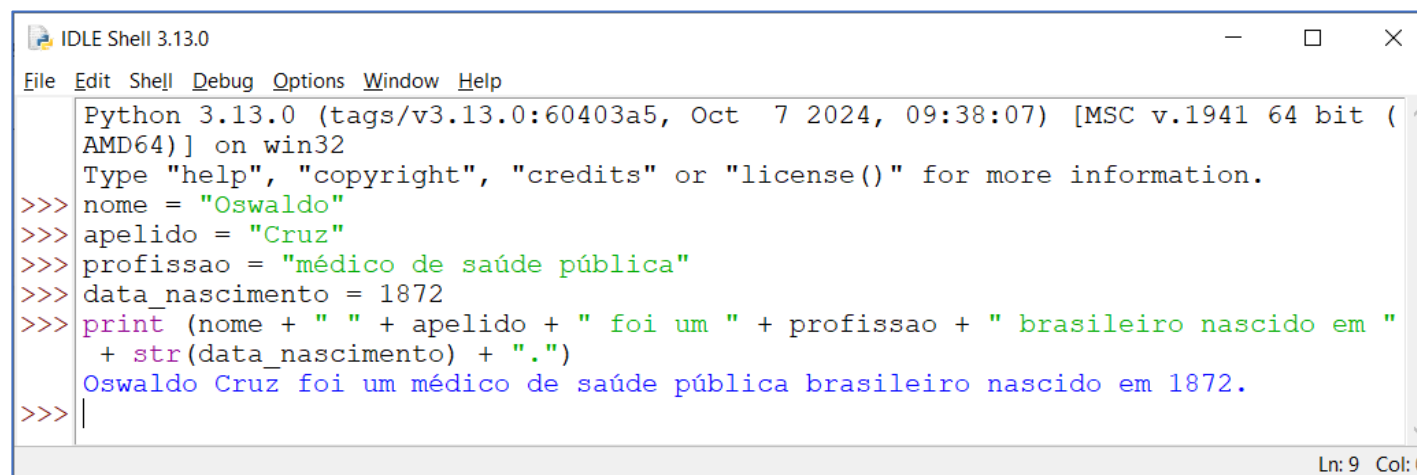


```
IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> nome = "Oswaldo"
>>> apelido = "Cruz"
>>> profissao = "médico de saúde pública"
>>> data_nascimento = 1872
>>> print (nome + " " + apelido + " foi um " + profissao + " brasileiro nascido em "
+ data_nascimento + ".")
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    print (nome + " " + apelido + " foi um " + profissao + " brasileiro nascido
em " + data_nascimento + ".")
TypeError: can only concatenate str (not "int") to str
>>>
```

# Concatenação de *strings*

- As variáveis de tipo inteiro e *string* não podem ser concatenadas no mesmo comando. A variável inteira tem de ser convertida numa *string* ou vice-versa, através de um processo designado *TypeCasting*. Neste caso, usa-se a função `str()` para transformar o número inteiro numa *string*:

```
>>> print (nome + " " + apelido + " foi um " + profissao + " brasileiro nascido em " +
str(data_nascimento) + ".")
```



```
IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> nome = "Oswaldo"
>>> apelido = "Cruz"
>>> profissao = "médico de saúde pública"
>>> data_nascimento = 1872
>>> print (nome + " " + apelido + " foi um " + profissao + " brasileiro nascido em "
+ str(data_nascimento) + ".")
Oswaldo Cruz foi um médico de saúde pública brasileiro nascido em 1872.
>>>
```

# Inicialização de variáveis

- Em algumas situações, as variáveis usadas num programa devem ser **inicializadas** para que possam ser usadas de forma apropriada e produzam o resultado correto no final
- A inicialização consiste em atribuir um valor inicial à variável, mas que pode ser alterado no decorrer do programa. Por exemplo, se pretendermos contar o número de caracteres 'o' na *string* “Oswaldo Cruz foi um médico de saúde pública brasileiro nascido em 1872”, devemos atribuir o valor 0 (zero) à respetiva variável antes da contagem ser iniciada:

```
>>> numero_de_o = 0
```

# Variáveis e atribuição

- Os objectos ligados às variáveis permanecem os mesmos até serem alterados manualmente ou como resultado de uma função ou operação matemática. No exemplo seguinte, o índice de massa corporal (IMC) não irá ser corrigido se o valor atribuído à variável *altura* for alterado após o cálculo:

$$\text{IMC} = \text{peso} / (\text{altura}^{**2})$$

```

IDLE Shell 3.13.0
File Edit Shell Debug Options Window Help
Python 3.13.0 (tags/v3.13.0:60403a5, Oct 7 2024, 09:38:07) [MSC v.1941 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> peso = 73
>>> altura = 1.70
>>> IMC = peso/(altura**2)
>>> IMC
25.25951557093426
>>> altura = 1.60
>>> IMC
25.25951557093426
>>> IMC = peso/(altura**2)
>>> IMC
28.515624999999993
    
```

# Exercícios

- Execute o comando *print()* com as seguintes *strings* na *Shell* do *Python* e interprete os resultados:

```
>>> print ("Charles")
```

```
>>> print ("Charles" + "Darwin")
```

```
>>> print ("Charles" + " " + "Darwin")
```

```
>>> print ("Charles" + "" + "Darwin")
```

```
>>> print ("Charles", "Darwin")
```

# Exercícios

- Introduza as seguintes variáveis na *Shell* do *Python*:

```
>>> nome = "Charles"
```

```
>>> apelido = "Darwin"
```

```
>>> actividade = "Naturalista"
```

```
>>> livro = "A Origem das Espécies"
```

```
>>> data = 1859
```

```
>>> data_nasc = 1809
```

- Use a função *print()* e a concatenação de *strings* para produzir as seguintes 3 frases:

"Charles Darwin: Naturalista"

"Charles Darwin publicou a Origem das Espécies em 1859"

"Charles Darwin publicou a Origem das Espécies aos 50 anos de idade"

# Fim da sessão 1

