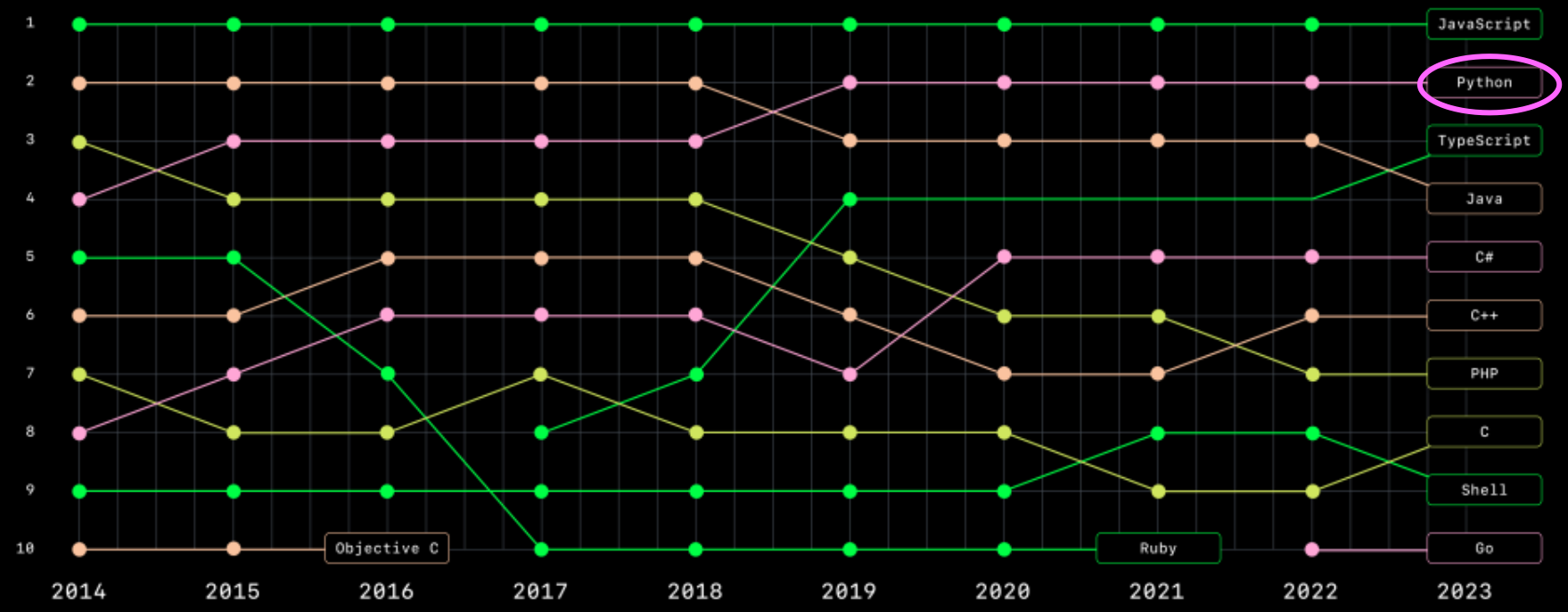
UMA BREVE INTRODUÇÃO AO PYTHON

(Com conteúdos baseados no livro “A Byte of Python”, de Swaroop [4])

O Python

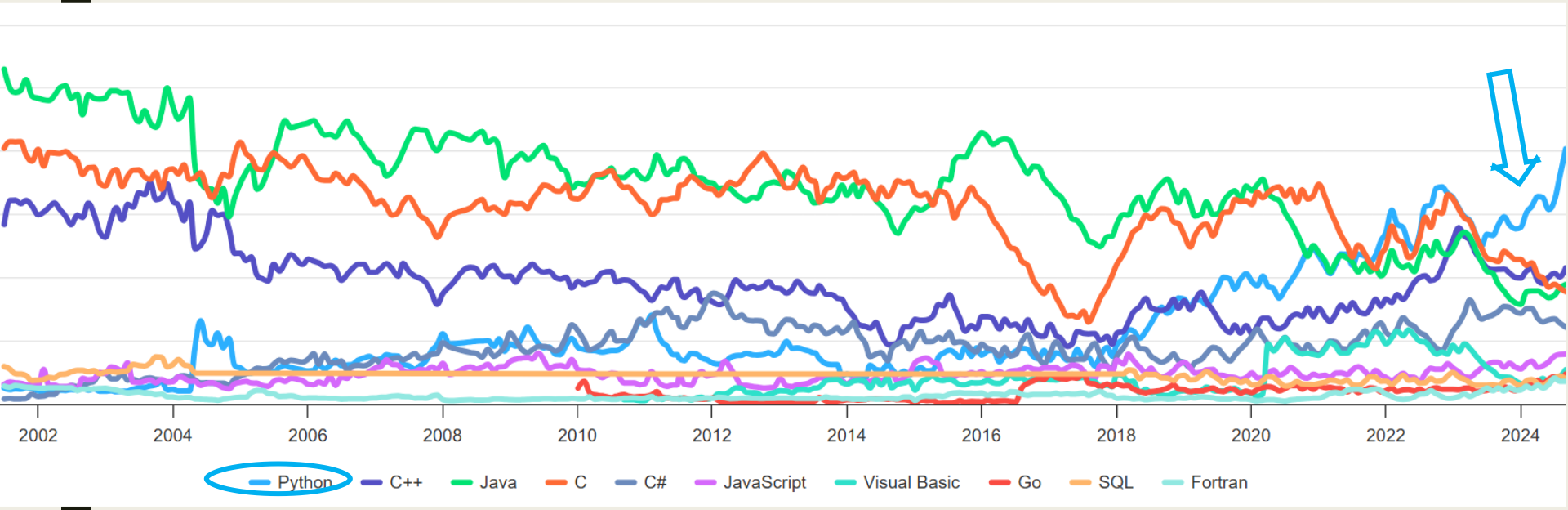
- Guido van Rossum, o criador da linguagem, inspirou-se no grupo de comediantes britânico Monty Python para lhe dar o nome pela qual é conhecida
  - *nada tem a ver com as pitons, até porque, segundo se sabe, Rossum nem é um grande apreciador de cobras…*
- Atualmente, o Python é, provavelmente, a linguagem mais popular e em maior crescimento
  - *desde 2019 é a 2ª linguagem mais utilizada dentro da plataforma GitHub – logo a seguir ao javascript*
  - *em Set/2024, e ao longo do último ano, destaca-se como a linguagem mais popular de acordo com os critérios da TIOBE*
- O Python é uma Linguagem de programação que consegue juntar o melhor de dois mundos: *simples (de aprender e usar)* e *poderosa*
- O Python é utilizado em todos os domínios e áreas de aplicação, incluindo a Data Science
  - *tem-se imposto cada vez mais como a linguagem de eleição da comunidade da Inteligência Artificial e da Machine Learning.*

Top 10 programming languages on GitHub



fonte: [ain.ua](ain.ua)

Índice de popularidade do Python segundo www.tiobe.com



Caraterísticas do Python

O Python é uma linguagem poderosa e empolgante, pois combina o desempenho a características que a tornam simples e divertida.

- É de acesso livre e de código aberto
- Tem um código Simples de interpretar (parecido com o pseudo-código)
- Fácil de aprender
- De alto nível
- De elevada portabilidade (se se evitarem recursos de sistema, um mesmo programa corre em múltiplas plataformas)
- É uma linguagem Interpretada (e convertível num código intermédio, o bytecode)
- Orientada a Objetos (suporta o paradigma de uma forma mais simples que o C++ e o Java)
- Permite programas facilmente extensíveis com código de outras linguagens (pode-se codificar trechos de código em C ou C++ e usá-lo no Python)
- Também facilmente incorporável em programas C/C++
- Linguagem estendida através da sua poderosa biblioteca standard e por inúmeras outras (pesquisáveis no Python Package Index)

Instalação do Anaconda

- Pretendendo instalar apenas o Python, sem integrar, com ele, outros módulos específicos interessantes, bastaria aceder a https://www.python.org/downloads/ e descarregar a versão mais recente, que à data de atualização deste texto era o Python 3.12.6.
- No entanto, em vez de se instalar o Python isoladamente, sugere-se a instalação do Anaconda, uma distribuição do Python que integra, para além do core do Python, um conjunto de bibliotecas muito úteis para a Machine Learning (ML),
  - *caso contrário, teríamos que instalar essas bibliotecas posteriormente de forma separada.*
  - *O Anaconda pode ser descarregado de https://www.anaconda.com/download/.*

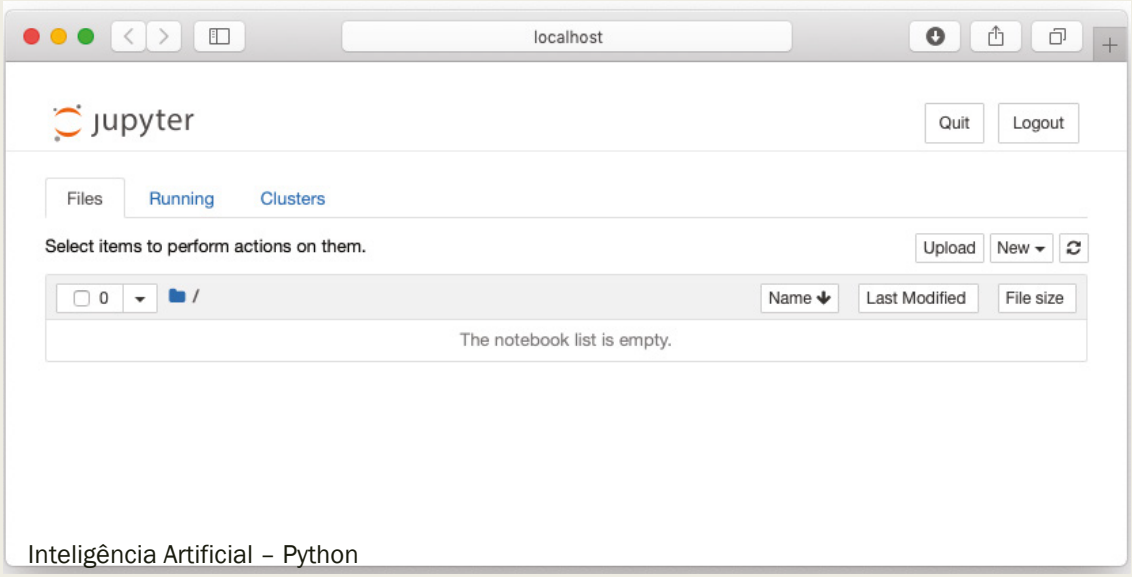
    (na instalação, aceitar configurações por defeito)

- Todos os modelos de ML serão desenvolvidos com recurso à biblioteca Scikit-learn, a qual implementa os vários tipos de algoritmos de ML: de regressão, classificação e clustering.
  - *Para além da Scikit-learn, vão também revelar-se úteis para a ML outras bibliotecas adiconais do Python: NumPy, Pandas, matplotlib, e seaborn.*
  - *Felizmente, a distribuição Anaconda já inclui todas estas 5 bibliotecas.*

Executando o Jupyter Notebook

- Uma vez o Anaconda instalado, já temos disponível o *Jupyter Notebook*, uma aplicação web que nos permite, num ambiente gráfico e interativo, criar e editar documentos conhecidos como *notebooks,* que para além de código, integram documentação, gráficos e outros recursos.
  - *É esta aplicação que é usada em [1] para criar e editar, de uma forma interativa e didática, todas as solução ML apresentadas.*
- *Para arrancar com o Jupyter Notebook* em Windows,
  - *executar a 'Anaconda Prompt'*
  - *dentro da respetiva janela de consola, escrever 'jupyter notebook'*

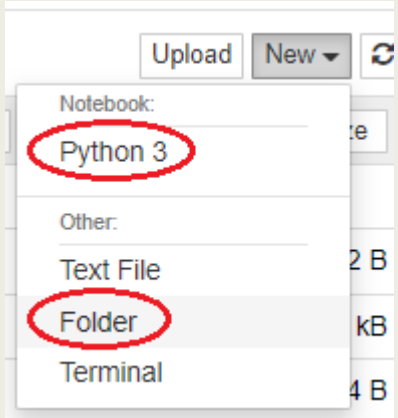
  (no Windows também é possível invocar diretamente o 'jupyter notebook')

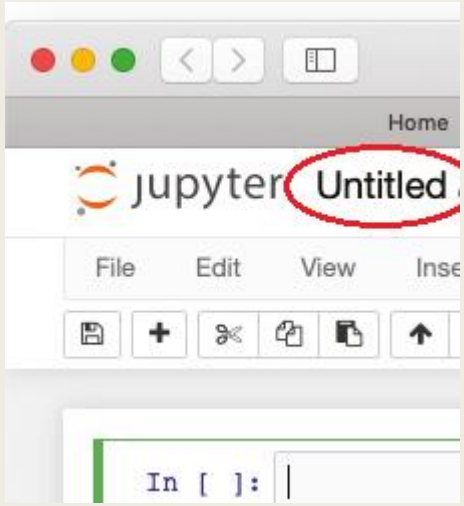


Primeiros passos com o Jupyter Notebook

- Criar uma pasta de trabalho, selecionando 'Folder' no botão 'New' que se encontra no lado direito
  - *depois, alterar o nome de 'Untitled Folder' para 'IA', por exemplo, e clicar nela para se tornar na pasta de trabalho.*

  (se pretender saber a localização exata da pasta de trabalha, basta usar o comando pwd*)*
- Para criar um novo notebook (“documento”), selecionar 'Python 3' no mesmo botão 'New' que se encontra no lado direito.
- Clicar em 'Untitled' para dar um nome adequado ao notebook que acaba de ser criado
  - *o mesmo é gravado com a extensão .ipynb, na nova pasta 'IA'*

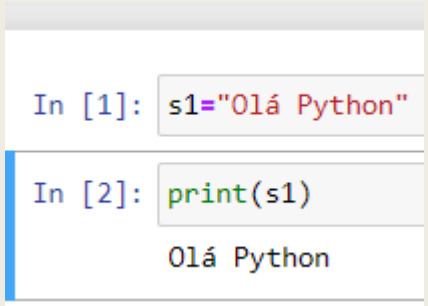


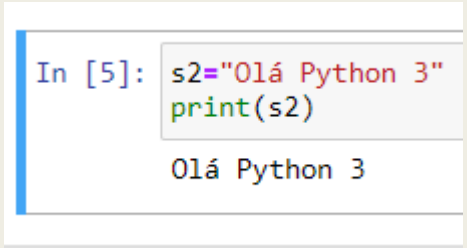


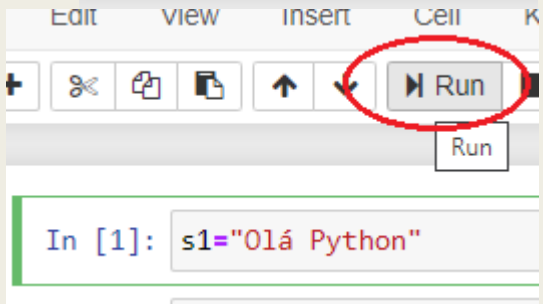
Primeiros passos com o Jupyter Notebook

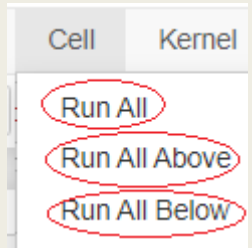
- Um notebook contém uma ou mais células de execução
  - *podemos digitar um comando Python em cada uma dessas células, e executá-los separadamente*
  - *mas podemos, igualmente, executar duma só vez vários comandos, inserindo numa só célula um trecho de várias instruções*
  - *para executar uma célula, clicar no botão 'run', com essa célula selecionada*

    (ou pressionar Shift+Enter)
  - *para executar todas as células ou as células anteriores ou posteriores à selecionada, aceder ao menu 'Cell'*









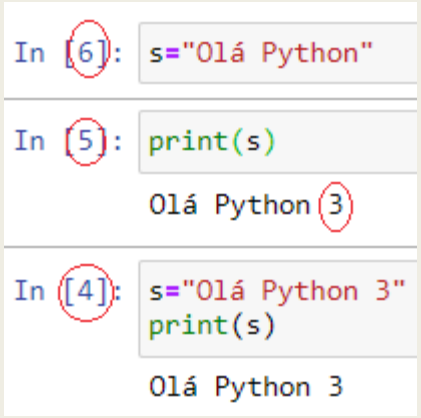
Primeiros passos com o Jupyter Notebook

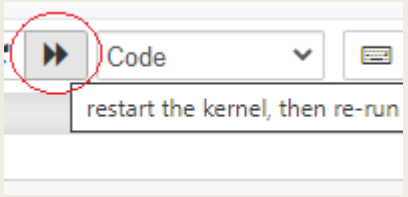
- As células podem ser executadas por uma qualquer ordem
  - *à esquerda de cada célula encontra-se o número de ordem com que foi executada*

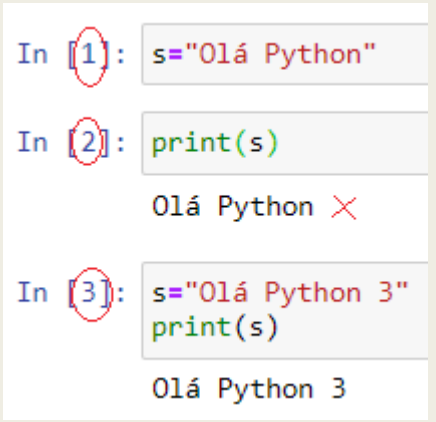
- Quando a ordem de execução é diferente da ordem com que se apresentam as células, é frequente obterem-se resultados inesperados ou ter-se alguma dificuldade em interpretar os resultados obtidos
  - *nessas situações pode dar jeito reiniciar o Kernel, para que todas as variáveis sejam reiniciadas, e voltar a executar todas as células pela ordem apresentada*

- Para se obter ajuda sobre uma função, posicionar o cursor no nome dessa função e pressionar Shift+Tab

- Depois de devidamente testada a solução criada, o código do notebook pode ser exportado para um ficheiro Python
  - *para isso, selecionar File>Download as>Python (.py)*







O editor Visual Studio Code (VS Code)

- *Ainda que se possa usar unicamente o Jupyter Notebook para construir totalmente os modelos de ML em Python (opção adotada em [1]),*
  - *nesta unidade curricular optaremos pelo IDE VS Code,*
  - *trata-se de um editor mais poderoso e versátil que nos dará outra flexibilidade e funcionalidades adicionais para criarmos projetos em Python.*
- Para usar o VS Code com o Python é necessário começar por instalá-lo com a extensão para o Python

  Seguir, por exemplo, as instruções disponibilizadas em

  https://medium.com/@joaolggross/como-configurar-o-vs-code-com-anaconda-e-jupyter-notebooks-b05258bf65c1

  para instalar e configurar devidamente o VS Code, ignorando o passo 1 caso já se tenha o Anaconda instalado.

  (Link alterativo, com as instruções de instalação e um pequeno tutorial de utilização: https://code.visualstudio.com/docs/python/python-tutorial)
- É também possível usar o Jupyter Notebook no próprio VS Code:
  - *https://code.visualstudio.com/docs/python/jupyter-support*
  - *Se necessário, instalar no VS Code a extensão para o Jupyter Notebook*

Noções Básicas de Python

- Comentários com o carater cardinal #
  - `print('hello world') # Note that print is a function`
- Constantes
  - `5, 1.23, 9.25e-3, 'Esta é uma string', "está é outra!"`
- Números
  - *inteiros (de qq tamanho), de vírgula flutuante, e complexos*
- As variáveis não precisam de ser declaradas ☺
  - *São criadas atribuindo-lhes simplesmente um valor*
  - *Nenhuma declaração ou definição de tipo de dados é necessária*
- O Python é case-sensitive
- Os identificadores seguem essencialmente as mesmas regras do C (usam os carateres alfanuméricos e o underscore)
  - *mas no Python 3 já é possível usar carateres acentuados e de alfabetos não anglo-saxónicos*

Outras particularidades do Python

- O Python é fortemente orientado a objetos no sentido de que tudo é um objeto, incluindo números, strings e funções.
- Linha Física vs Linha Lógica
  - *Uma linha física é aquela que se vê quando se escreve*
  - *Uma linha lógica é a que o Python vê como uma única instrução*
  - *Implicitamente, o Python incentiva o uso de uma única instrução por linha, para que uma linha física corresponda a uma lógica*
    - Mas se pretendermos que uma linha física inclua várias instruções (várias linhas lógicas), só temos que as separar por ponto-e-vírgula (;)
      - *exemplo:* i=5; print(i);
    - Já se pretendermos distribuir um instrução por várias linhas físicas, deveremos usar a barra invertida (\) para que a mudança de linha física seja ignorada
      - *exemplo:* s = 'Isto é uma \
        única string sem mudança de linha'

Strings

- são imutáveis
- usam a codificação Unicode UTF-8
- as constantes podem ser delimitadas por aspas ("…") ou por plicas ('…')
- ou por ("""…""") e ('''…'''), para delimitar strings com várias linhas ou que incluam no seu interior aspas e plicas

```
''' uma string com
"duas linhas"! '''
```

- para as concatenar basta separá-las por um espaço: dia= 'É' ' ' 'sexta' '-' 'feira'
- usam-se as sequências de escape habituais: `\  \n  \t  \\  \'  \" ...`
- raw strings, precedidas por r ou R: `r"sou uma string sem qq formatação"`
  - *tiram o significado especial ao carater* `\`
  - *especialmente indicadas para a manipulação de expressões regulares*
- não há o tipo char

Strings – o método format()

- Para contruir strings mais elaboradas, a partir de uma 'string de formatação':

```
age = 20
name = 'Swaroop'
print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
```

```
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

- Como funciona
  - *A string de formatação usa certos especificadores de formato (à semelhança do printf do C) e, posteriormente, o método format() substitui esses especificadores pelos valores dos seus parâmetros no formato indicado*
    - na posição {0} é colocado o 1º parâmetro, na posição {1} o 2º, na posição {2} o 3º, e assim sucessivamente.
- Mas a partir do Python 3.6 passou a ser possível a formatação de strings de uma forma mais cómoda, usando f-strings

```
frase = f‘{name} was {age} years old when he wrote this book’
```

- Podem ser usadas especificações mais detalhadas:

```
print('{0:.3f}'.format(1.0/3)) # decimal precision of 3 for float '0.3333...'
# fill with underscores (_) with the text 'hello' centered (^) to 11 width
print('{0:_^11}'.format('hello'))
# keyword-based 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

Indentação

- Os espaços de início de linha são importantes em Python
- É usada para criar blocos de instruções (em vez de {...})
  - *todas as instruções dum bloco devem ter a mesma indentação.*
- A indentação inadequada dá origem a erros, quer lógicos, quer sintáticos
  - *Exemplo:*

```
i = 5
# Error below! Notice a single space at the start of the line
 print('Value is', i)
```

Resultado:

```
    print('Value is', i)
    ^
IndentationError: unexpected indent
```

- Portanto, em Python o uso de indentação não é meramente cosmético.

Operadores

- A maioria dos operadores do Python coincidem com os do C/C++. Alguns dos que se diferenciam são os seguintes:
  - *not - Operador lógico NOT*
  - *and - Operador lógico AND*
  - *or - Operador lógico OR*
  - *** - potência (2**3=2^3=8)*
  - */ - divisão real (5/3=1.6666..., 9/1.81=4.9723...)*
  - *// - divisão inteira (5//3=1, 9//1.81=4.0)*