Programação I
Aula 2 — Apresentação da linguagem Python

Pedro Vasconcelos
DCC/FCUP

Nesta aula...

O que é a programação de computadores?

- Implementação de métodos computacionais para resolução de problemas
- Análise e comparação de métodos diferentes
- Conjunção de várias competências:

  matemática linguagens formais para especificar processos;

  engenharia juntar componentes para formar um sistema; avaliar prós/contras de alternativas

  ciências naturais observar comportamento de sistemas complexos; formular hipóteses; testar previsões

Porquê aprender a programar?

- Trabalhos científicos necessitam de processamento de dados
- Facilita a automatização de tarefas repetitivas
- Muitas aplicações científicas são programáveis (ex: Excel, GNUplot, Matlab, Maple, Mathematica)
- Estrutura o pensamento para resolver problemas
- Desenvolve o pensamento analítico
- É um desafio intelectual
- É divertido!

Porquê aprender a programar? (cont.)

Programar desenvolve competências de resolução de problemas:

- capacidade para descrever problemas de forma rigorosa;
- pensar de forma criativa em possíveis soluções;
- expressar as soluções de forma clara e precisa.

Linguagens de Programação

- Linguagens formais para exprimir computação
  - sintaxe: regras de formação de frases (gramática)
  - semântica: significado associado a cada frase
- Outras exemplos de linguagens: expressões aritméticas, símbolos químicos

	sintaxe	semântica
$3 \times (1 + 2)$	ok	9
$3 \times 1 + 2$	ok	5
$\times)1 + 2 + (3$	erro	—
H_2O	ok	água
$_2zZ$	erro	—

Código máquina

```
55 89 e5 83 ec 20 83 7d 0c 00 75 0f ...
```

- Códigos numéricos associados a operações básicas
- Linguagem específica de cada micro-processador
- Única linguagem directamente executável pelo computador
- Difícil escrever programas diretamente em código máquina
- Concebida para facilitar a implementação usando circuitos eletrónicos

Linguagem *assembly*

```
55                  push   %ebp
89 e5               mov    %esp,%ebp
83 ec 20            sub    $0x20,%esp
83 7d 0c 00         cmpl   $0x0,0xc(%ebp)
75 0f               jne    1b
...                 ...
```

- Representação do código máquina usando mnemónicas
- Mais legível do que a linguagem máquina
- Pode ser traduzida automaticamente para código máquina
- Continua a ser específica para cada micro-processador
- Exige programação lenta, fastidiosa e susceptível a erros
- Usada apenas em contextos muito específicos

Linguagens de alto-nível

Exemplo: calcular factorial de 10 em Python.

```
n = 10
p = 1
for i in range(2, n+1):
  p = p*i
print("factorial de ", n, " = ", p)
```

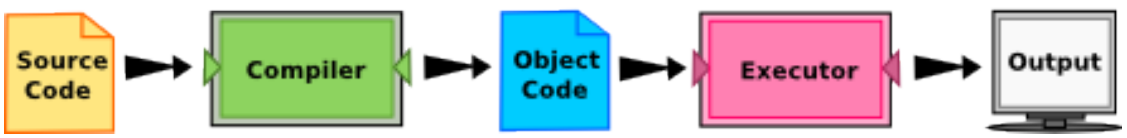
- Mais próximas da formulação matemática dos problemas
- Permitem o desenvolvimento de programas mais rápido e facilitam a deteção e a correção de erros
- Permitem desenvolver programas portáveis e.g. independentes do processador específico em cada computador

Interpretadores vs. compiladores

As linguagens de alto-nível são traduzidas para código máquina por programas especiais:

interpretador a tradução é efectuada de cada vez que o programa executa

compilador tradução é efectuada uma única vez



Interpretadores vs. compiladores (cont.)

Vantagens dos interpretadores:

- permitem uso interativo rápido
- facilitam testar fragmentos de programas
- são mais simples de implementar

Vantagens dos compiladores:

- permitem gerar código máquina mais eficiente;
- geram programas independentes
  (o compilador não tem de estar presente durante execução)

Cronologia de algumas linguagens

1956	Fortran I
1958	Lisp
1960	Cobol, Algol 60
1970	Pascal
1976	Fortran 77
1978	C (K&R)
1980	Smalltalk 80
1984	Common Lisp, C++
1986	Perl
1990	Fortran 90, Python, Haskell
1995	Java, JavaScript
2000	Python 2, C#
2008	Python 3

Porquê tantas linguagens?

- Diferentes níveis de abstração:
  nível mais alto: mais próximo da formulação dos problemas; facilita a programação, deteção e correção de erros
  nível mais baixo: mais próximo da máquina; potencialmente mais eficiente
- Diferentes tipos de problemas:
  cálculo numérico: Fortran, C, C++, Python
  sistemas operativos: C, C++
  sistemas críticos: Ada, C, C++
  sistemas *web*: Java, JavaScript, Python

Porquê tantas linguagens? (cont.)

- Diferentes paradigmas:
  imperativo: Algol, Pascal, C
  funcional: Lisp, Scheme, ML, OCaml, Haskell
  lógico: Prolog
  orientado a objectos: Smalltalk, C++, Java, C#
- Preferências subjectivas (estilo, elegância, legibilidade)

A linguagem Python

- Linguagem de alto nível
- Sintaxe simples, fácil de aprender
- Pode ser usada em Windows, Linux, FreeBSD, Mac OS, etc...
- Implementação *standard* distribuida como código livre
- Suporta programação procedimental e orientada a objectos
- Muitas bibliotecas disponíveis
- Muito utilizada: Google, Microsoft, Dropbox, NASA, Lawrence Livermore Labs, Industrial Light & Magic...
- Sítio oficial: `http://www.python.org`

A linguagem Python

Implementada com um interpretador híbrido:

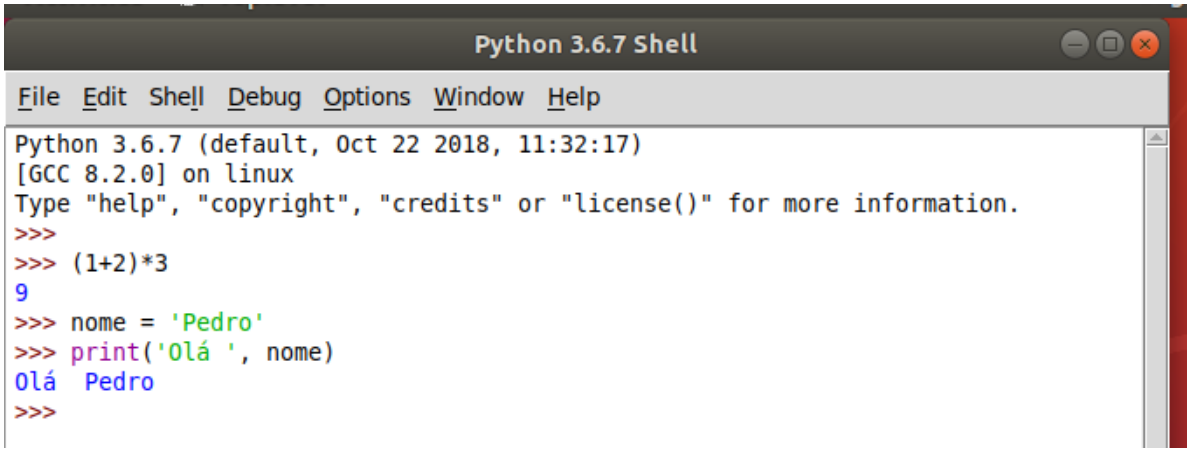
- o programa é traduzido para um "pseudo" código-máquina (*byte-code*)
- o *byte-code* é executado por um interpretador

Vantagens:
- fácil de usar interativamente
- fácil testar e modificar componentes
- mais eficiente do que um interpretador clássico

Desvantagem: não é tão eficiente como uma linguagem compilada (ex: C, C++ ou Fortran)

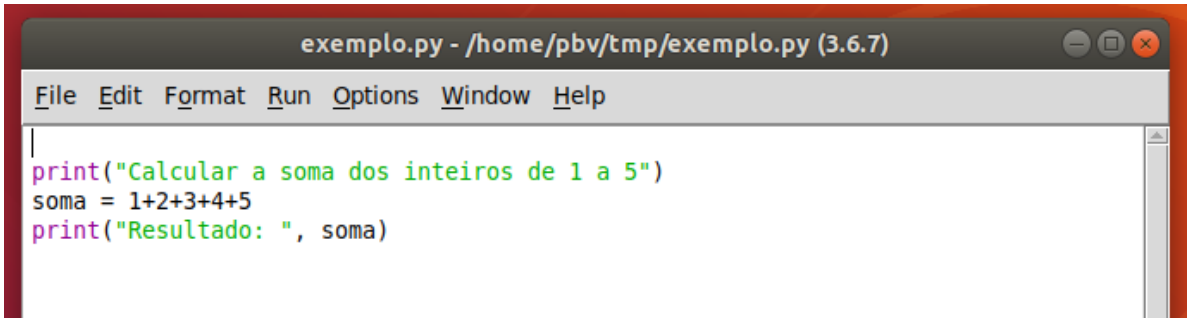
Utilização interativa

Executando o interpretador de Python (*IDLE*) podemos escrever comandos e ver os resultados imediatamente.



Utilização com um *script*

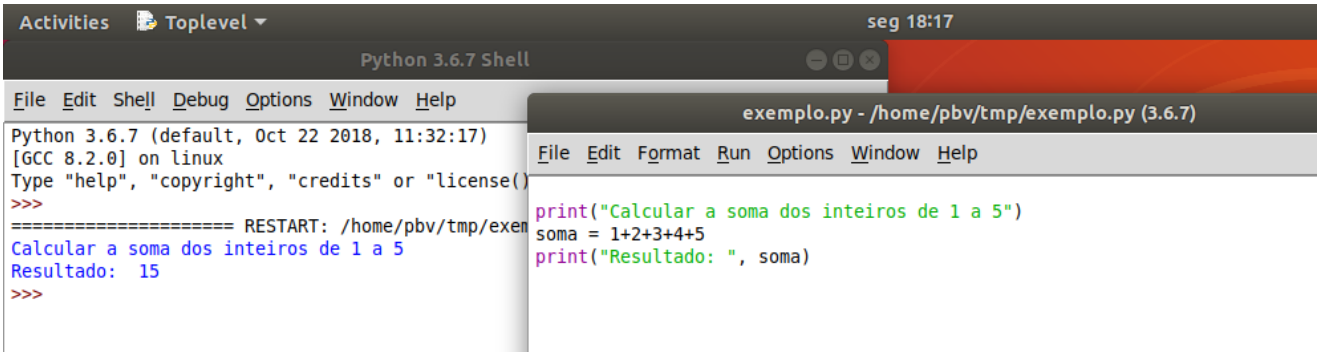
Em alternativa, podemos compor um programa num ficheiro de texto ("*script*") e executar de uma vez.



Convenção: programas em Python têm extensão `.py`

Utilização com um *script* (cont.)

- A forma interativa é usada para testar pequenas pedaços de código
- Para escrever programas com mais do que algumas linhas devemos editar um "*script*"
- O ambiente de desenvolvimento IDLE combina:
  - uma janela interativa para comandos ("*Python shell*")
  - uma ou mais janelas para ficheiros ("*scripts*")



Usar Python como uma calculadora

- Operadores aritméticos básicos:

  adição e subtração `+` `-`
  multiplicação e divisão `*` `/`
  exponenciação `**`
  parênteses `(` `)`
- Números inteiros e fracionários: `42` `-7` `3.1416`
- Expressões incorretas: `SyntaxError`

Usar Python como uma calculadora (cont.)

Prioridade entre os operadores (ordem de cálculo):

1. parêntesis `(` `)`
2. exponenciação `**`
3. multiplicação e divisão `*` `/`
4. soma e subtração `+` `-`

Operadores da mesma prioridade agrupam à esquerda.

Exemplos:

```
>>> 1+2-3+4
4
>>> 1+2-(3+4)
-4
```

```
>>> 2**3*4+1
33
>>> 2**3*(4+1)
40
```


Funções matemáticas

Muitas funções e constantes matemáticas estão disponíveis no módulo *math*.

Para usar devemos começar por importar o módulo.

```
>>> import math
```

Os nomes das funções começam com prefixo "`math`":

```
>>> math.sqrt(2)
1.4142135623730951
>>> math.pi
3.141592653589793
```

Funções matemáticas (cont.)

Algumas funções e constantes do módulo *math*:

raiz quadrada	`sqrt`
funções trignométricas	`sin`, `cos`, `tan`
funções trignométricas inversas	`asin`, `acos`, `atan`, `atan2`
exponencial e logaritmos	`exp`, `log`, `log10`
e, π	`e`, `pi`

Para obter mais informação:

```
>>> help(math)
>>> help(math.log)
```

informação geral
específica sobre uma função

Tipos

Os valores são classificados em diferentes tipos.

Algumas operações só são possíveis com determinados tipos:

```
>>> "Ola mundo!" + 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str
```

Tipos básicos

	tipo	exemplos
inteiros	`int`	`1 -33 29`
vírgula-flutuante	`float`	`1.0 -0.025 3.14156`
cadeias de texto	`str`	`"Ola mundo!" 'ABC' '1.23.99'`

Tipo de um resultado

No interpretador podemos usar `type(...)` para obter o tipo do resultado duma expressão:

```
>>> (1+2+3)*5-1
29

>>> type((1+2+3)*5-1)
<class 'int'>

>>> type(1.234)
<class 'float'>

>>> type('ABC')
<class 'str'>
```