Programação em Python

Introdução à linguagem Python

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

1. Porquê programar?

2. Linguagens de Programação

3. A linguagem Python

Porquê programar?

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 complexo 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 de Programação

- Linguagens formais para exprimir computação

  sintaxe: regras de formação (gramática)

  semântica: significado ou operação associados

- Outras 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	—

Linguagem máquina

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

- Linguagem específica de cada micro-processador
- Códigos numéricos associados a operações básicas
- Única linguagem directamente executável pelo computador
- Não é pensada para ser usada por programadores

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 da linguagem máquina em mnemónicas
- Mais legível por humanos do que a linguagem máquina
- Pode ser traduzida para linguagem máquina automaticamente por um programa assemblador
- Continua a ser específica para cada micro-processador
- Muito baixo-nível: desenvolvimento lento, fastidioso, susceptível de erros
- Usada apenas em contextos muito específicos

Linguagens de alto-nível

Exemplo: linguagem C

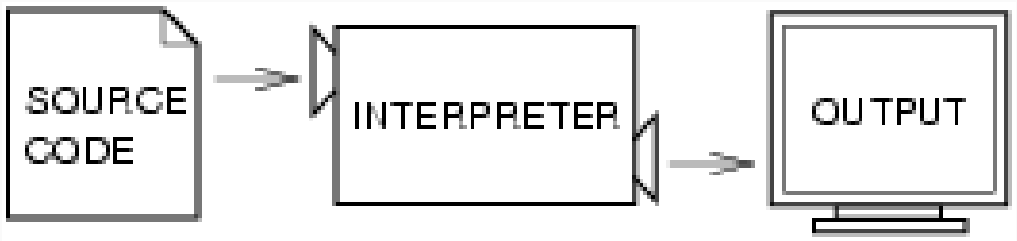
```
p = 1;
for(i=2; i<=n; i++)
  p = p*i;
printf("factorial %d = %d\n", n, p);
```

- Mais próximas da formulação matemática dos problemas
- Facilitam o desenvolvimento de programas
- Programas são portáveis (independentes da arquitetura dum computador específico)

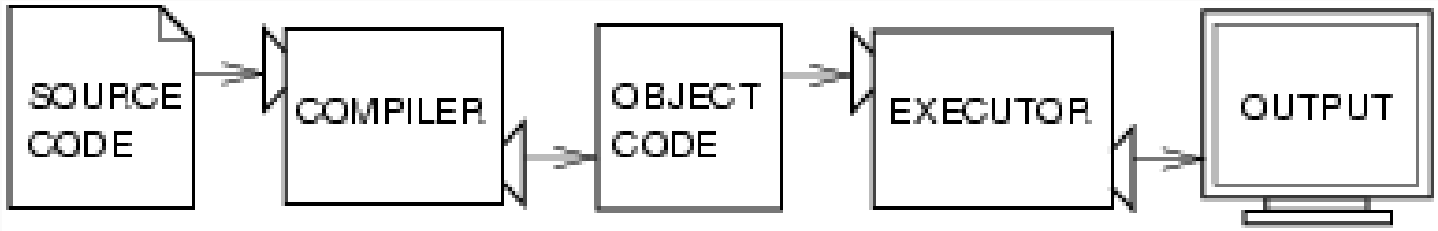
Interpretadores vs. compiladores

Linguagens de alto-nível são traduzidas para linguagem máquina por programas especiais:

interpretadores: tradução é efectuada sempre que o programa executa;



compiladores: tradução é efectuada uma só vez; produz um programa código-máquina independente



Cronologia de algumas linguagens

1956	Fortran I
1958	Lisp
1960	Cobol, Algol 60
1964	PL/I
1968	Smalltalk
1970	Pascal, Prolog
1974	Scheme
1976	Fortran 77, ML
1978	C (K&R)
1980	Smalltalk 80
1982	Ada 83
1984	Common Lisp, C++, SML
1986	Eiffel, Perl, Caml
1988	Tcl
1990	Fortran 90, Python, Java
1994	Ruby, Perl 5
1995	JavaScript
1996	OCaml
1998	Scheme R5RS, C++(ISO), Haskell 98
2000	Python 2.0, C#
2004	C# 2.0(beta), Java 2 (beta)
2008	Python 3.0

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++

  sistemas operativos: C, C++

  sistemas críticos: Ada, C, C++

  sistemas *web*: Java, JavaScript, Ruby, 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

A linguagem Python

- Linguagem de alto nível
- Sintaxe simples: fácil de aprender
- Pode ser usada em Windows, Linux, FreeBSD, Mac OS, etc...
- Implementação distribuida como *código livre*
- Suporta programação procedimental e orientada a objectos
- Muitas bibliotecas disponíveis
- Usada no "mundo real": Google, Microsoft, Yahoo!, NASA, Lawrence Livermore Labs, Industrial Light & Magic...
- Sítio oficial: `http://www.python.org`

A linguagem Python

Python é implementado com um interpretador híbrido:

- programa Python é traduzido para um código intermédio (*byte-code*);
- o *byte-code* é executado por um interpretador especial.

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 tradicional (ex: C, C++, Fortran)

Utilização interativa

Executando `python3` num terminal podemos escrever comandos Python e ver os resultados imediatamente.

```
Python 3.2.3 (default, Jul  5 2013, 08:29:02)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>> 1+1
2
>>> print("Ola, mundo!")
Ola, mundo!
>>>
```

Ctrl-D (“end-of-file”) ou `quit()` para terminar.

Utilização com um *script*

Em alternativa podemos escrever um programa completo num ficheiro de texto (*script*) e executar de uma só vez.

programa.py

```
print("Ola, mundo!")
print("1 + 1 = ", 1+1)
```

Executamos no terminal "`python3 programa.py`" e obtemos:

```
Ola, mundo!
1 + 1 = 2
```

Convenção: ficheiros de programas Python têm extensão `.py`

Utilização com um *script* (cont.)

- A forma interativa é usada para testar pequenas partes de código.
- Devemos escrever programas com mais do que algumas linhas num *script*.
- *Ambientes de desenvolvimento* como o IDLE e o Pyzo combinam:
  - uma janela para testes interativos;
  - uma ou mais janelas para *scripts*

(segue-se uma demonstração. . .)

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)
1.414213562373095 1
>>> 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:

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

importar a biblioteca `math`

informação geral

específica sobre uma função