Nesta aula...

Conteúdo

1 Introdução à Programação

1.1 O que é?

O que é a programação de computadores?

- Concepção de métodos para resolver problemas usando computadores
- Análise e comparação de métodos diferentes
- Conjunção de várias competências:

  matemática linguagens formais para especificar ideias

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

  ciências naturais observar comportamento de sistemas complexos, tecer hipóteses, testar previsões

1.2 Para que serve?

Porquê aprender a programar?

- competência essencial em Ciência de Computadores
- as aplicações sofisticadas são programáveis (GNUplot, Matlab, Maple, Emacs, Mathematica, Excel...)
- trabalhos científicos necessitam de processamento de dados complexo
- necessidade de automatizar tarefas repetitivas
- estrutura o pensamento para resolver problemas
- é um desafio intelectual
- é uma competência transponível para outras áreas
- é divertido!

2 Linguagens de programação

Linguagens de programação

- notações *formais* para exprimir computação

  sintaxe: regras de formação (*gramática*)

  semântica: *significado* ou *acção* associados

- analogias: expressões aritméticas, símbolos químicos

	sintaxe	semântica
$1 + 2 = 3$	ok	verdade
$1 + 2 = 4$	ok	falsidade
$+ = 3, 2$	*erro*	—
H_2O	ok	água
$_2zZ$	*erro*	—

Linguagens de baixo nível: linguagem máquina

```
01010101 10001001 11100101 10000011 11101100 ...
55       89       e5       83       ec       ...
```

- linguagem nativa de um computador
- códigos numéricos associados a operações elementares
- incompreensível para humanos
- única linguagem directamente executável pelo computador

Linguagens de baixo nível: "*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 em mnemónicas (texto)
- traduzida para código máquina por um programa *assemblador*
- muito próxima da máquina: desenvolvimento lento, fastidioso, susceptível de erros

Linguagens de alto nível

E.g. 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
- independentes das características do *hardware*
- traduzidas para código máquina por programas especiais:

  interpretadores: tradução é efectuada passo-a-passo

  compiladores: tradução é efectuada uma só vez; produz um *executável* em código-máquina

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
1996	OCaml
1998	Scheme R5RS, C++(ISO), Haskell 98
2000	Python 2.0, C#
2004	C# 2.0(beta), Java 2 (beta)

Porquê tantas linguagens?

- diferentes *níveis de abstração*:

  mais alto nível: mais próximo da formulação do problemas; facilita a programação, detecção e correcção de erros

  mais baixo nível: mais próximo da máquina; potencialmente mais eficiente

- diferentes *problemas*:

  cálculo numérico: Fortran, C

  sistemas operativos: C, C++

  sistemas críticos: Ada

  scripting: Perl, Tcl, Python

Porquê tantas linguagens? (2)

- diferentes *paradigmas*:

  imperativo: Algol, Pascal, C

  funcional: Lisp, Scheme, ML, Caml, Haskell

  lógico: Prolog

  orientado a objectos: Smalltalk, C++, Java, C#

- subjectividade: estilo, elegância, legibilidade

2.1 Programação estruturada

Programação estruturada

- decompor um programa em pequenos *módulos* (analogia: blocos *Lego*)
- módulos podem ser *re-utilizados*
- podem-se *testar* e/ou *provar* que são correctos
- fáceis de *modificar*
- programas devem ser escritos para serem *lidos* por humanos
- *simples* e *correcto* primeiro, eficiente depois

2.2 Apresentação da linguagem Python

A linguagem *Python*

- Linguagem de alto nível
- Fácil de aprender, sintaxe simples
- Distribuída livremente
- Disponível em Linux, Windows, Mac OS (entre outros)
- Ligação com muitas bibliotecas: gráficos, computação numérica, web, etc.
- Usada no "mundo real": Microsoft, Yahoo!, NASA, Lawrence Livermore Labs, Industrial Light & Magic…
- `http://www.python.org`

A linguagem *Python*

Interpretada de forma "híbrida":

- compilador traduz Python para um código intermédio "*byte-code*"
- execução é feita por um interpretador de "*byte-code*"

Vantagens:
- desenvolvimento rápido
- fácil testar módulos
- mais eficiente do que um interpretador clássico

Desvantagens:
- é menos eficiente como e.g. C/C++ ou Fortran compilado
- operações elementares podem não ser executadas em tempo constante

Interpretador de *Python*

- Linux: comando `python`

  interactivo executa um comando de cada vez e imprime resultados

  batch executa todos os comandos num ficheiro

- Linux, Windows, Mac OS: ambientes de desenvolvimento: IDLE, Eric, Emacs
- Segue-se uma demonstração...

Usar o *Python* como uma calculadora

- prioridades entre operadores aritméticos
  1. Parênteses `(`, `)`
  2. Exponenciação `**`
  3. Multiplicação e Divisão `*`, `/`
  4. Adição e Subtracção `+` `-`
- expressões incorrectas: `SyntaxError`
- números inteiros e fraccionários são distintos: `2` $\neq$ `2.0` (mais na próxima aula)
- Ctrl-D ("end-of-file") para terminar

Na próxima aula

- Valores, expressões e nomes
- Tipos básicos