Programação I / Introdução à Programação

Capítulo 1, "The way of the program"

João Pedro Pedroso

2024/2025

Programação 1 / Introdução à Programação

Aula passada:

- Programa
- Avaliação
  - Avaliação contínua (necessária para obtenção de frequência):
    - "quizzes" das aulas teóricas
    - exercícios das aulas práticas
  - Exame final
- Exercícios com avaliação automática (Codex):
  - teóricas $\rightarrow$ corrigidos de forma assíncrona
  - práticas $\rightarrow$ corrigidos de imediato
- Exame
  - Feito no computador, com correção automática
  - Salas com a mesma configuração das práticas
  - Revisão pelo docente $\rightarrow$ pontuação pela *elegância* dos programas
- Linguagem utilizada: Python
- Ambiente nos laboratórios: Linux
- Editor de texto recomendado: Pyzo

Introdução aos computadores

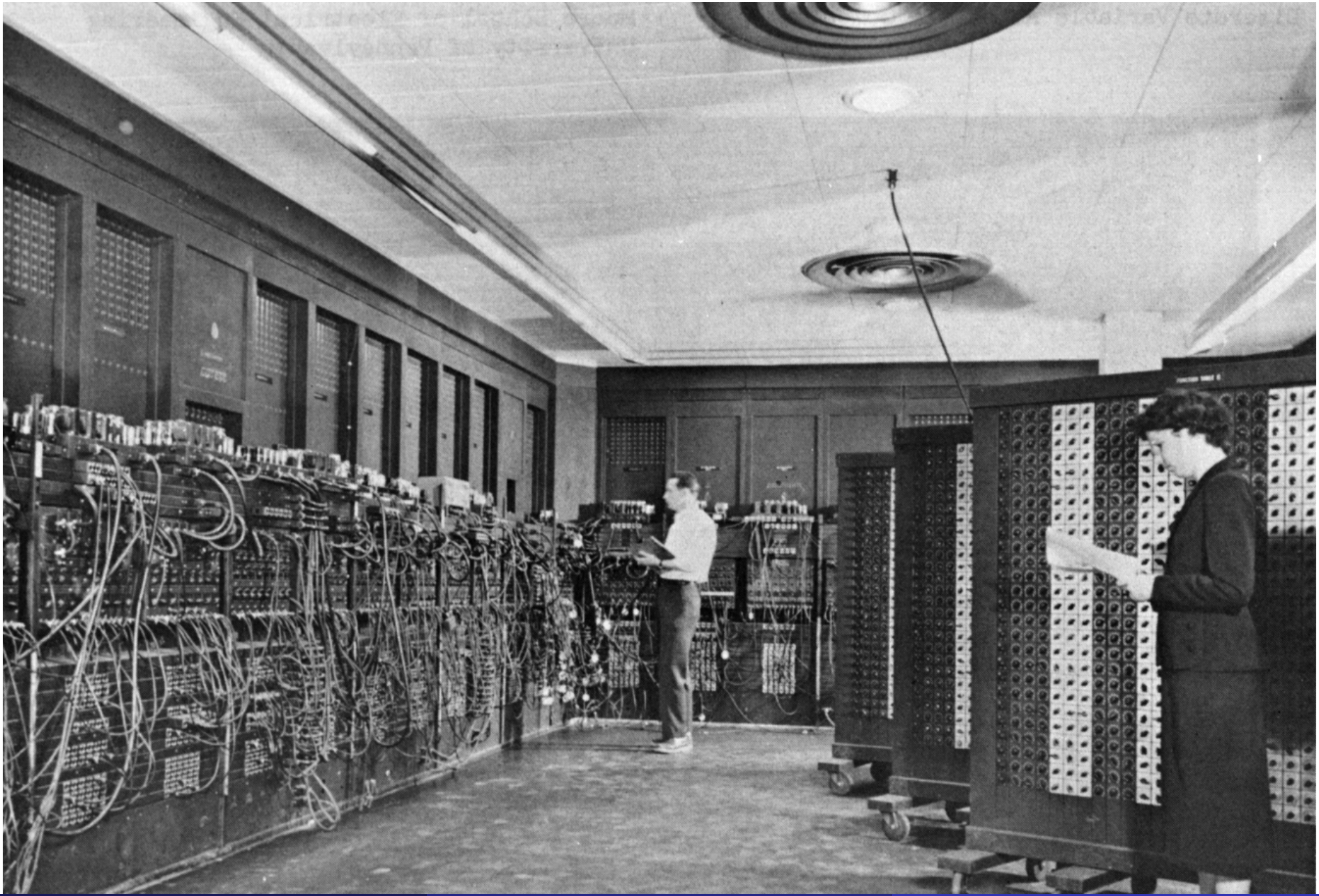
Introdução aos computadores

- Computador: processador de informação (dados) segundo uma lista de instruções (programa)
- efetua operações simples (aritméticas/lógicas) depressa:
  - *humano:* aproximadamente 1 operação/segundo
  - *computador atual:* >1 milhão de operações/segundo
- qualquer tipo de dados quantificáveis (números, textos, sons, imagens...)
- universal: utilizável para diversos fins com diferentes programas

Breve cronologia dos computadores

- 1940s → Colossus, Harvard Mk I, ENIAC — 2ª Guerra Mundial
- 1950s → UNIVAC I — computadores comerciais
- 1960s → IBM System/360
- 1970s → PDP-11, DEC VAX, VMS, UNIX
- 1977 → Apple II — computadores pessoais
- 1981 → IBM PC
- 1984 → Apple Mac — interfaces gráficas
- 1986 → Intel 386, Windows 1.0
- 1990 → Windows 3.0
- 1991 → WWW, GNU/Linux — internet, open-source
- 1993 → Intel Pentium
- 1995 → Windows 95
- 2001 → Windows XP, MacOS X
- 2007 → iOS (iPhone OS)
- 2008 → Android

Breve cronologia dos computadores: ENIAC



Mainframes

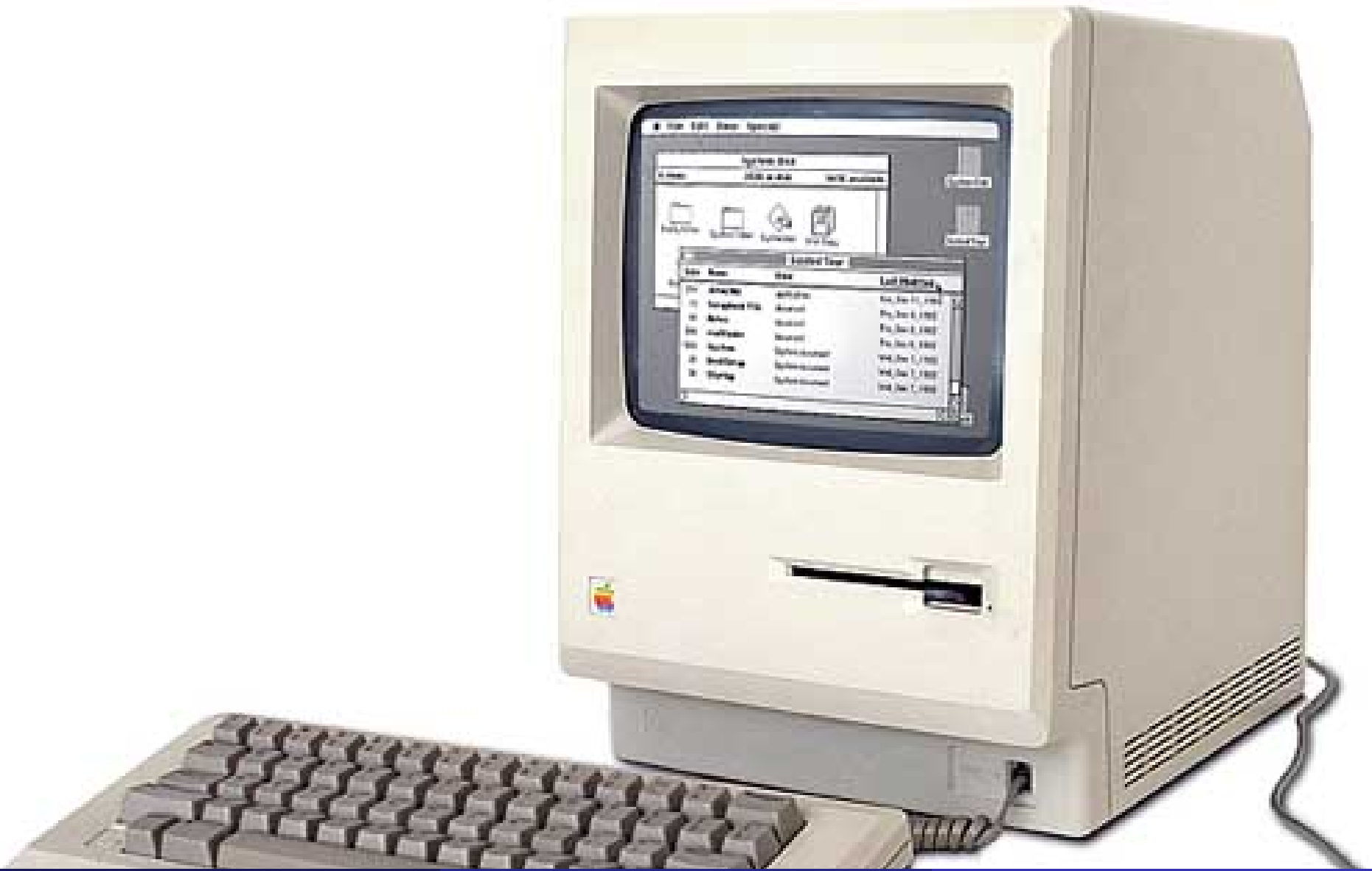
PDP-11

Computadores pessoais

Apple II

IBM PC

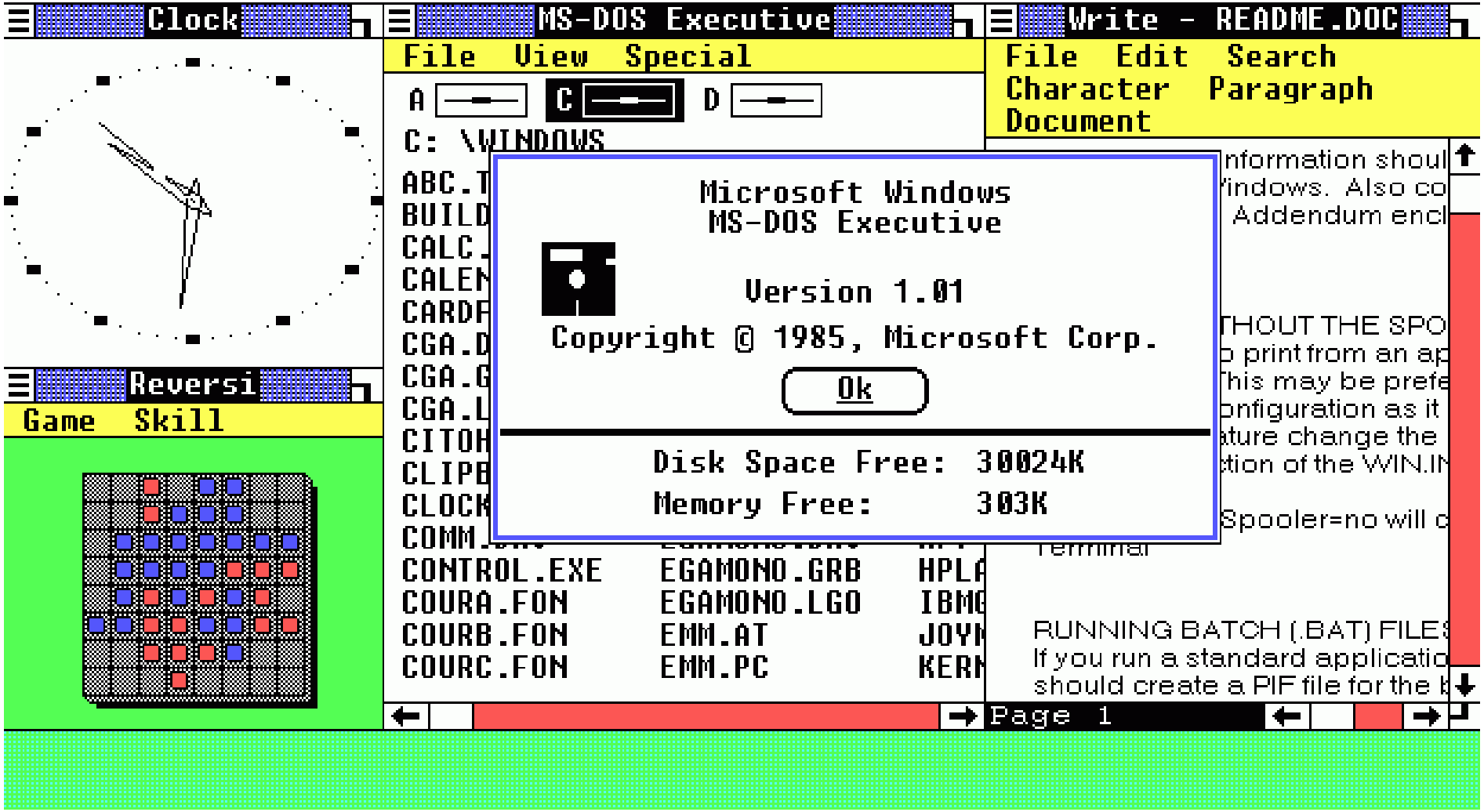
Computadores pessoais com interfaces gráficas: Apple Macintosh



Computadores pessoais com interfaces gráficas: Commodore Amiga

Computadores pessoais com interfaces gráficas: Atari ST

Computadores pessoais com interfaces gráficas: "IBM PC clones" e a arquitetura "Wintel¨



World Wide Web

1991: GNU/Linux

Plataformas móveis: iOS/Android/...

Smart watches

Níveis conceptuais de um computador

- Hardware: CPU, memória, unidades de disco, ecrã, teclado, rato...
- Software: sistema operativo, aplicações, jogos, ficheiros de dados (imagens, músicas, filmes, folhas de cálculo, bases de dados...)

Tendências:

- *hardware* mais barato
- *software* mais complexo e caro
- importância do desenvolvimento de *software*:
  - utilização de linguagens de alto nível
  - reutilização de componentes (bibliotecas)

Sistemas de operação

- Conjunto de *software* de base para gerir recursos do computador
- Proporciona funcionalidades para as aplicações:
  - gestão de utilizadores
  - gestão de memória
  - gestão de ficheiros
  - gestão de input/output (I/O): terminais, impressoras, interfaces gráficas, ligações de rede

Cronologia do sistemas de operação (1)

- Primeiros sistemas de operação (1950s):
  - um trabalho de cada vez (*batch*)
  - apenas supervisiona as transições entre trabalhos
- *Mainframes* (1960-1970s)
  - caros: necessário partilhar recursos
  - multi-utilizador: vários utilizadores em *terminais*
  - multi-tarefa: divisão o tempo de processamento entre os vários trabalhos (*time-sharing*)

Cronologia dos sistemas de operação (2)

- Mini-computadores: sistema UNIX (1970s)
  - multi-utilizador, multi-tarefa
  - portável para diferentes modelos de computadores
  - "código-fonte" em linguagem C distribuído com o sistema
  - popular na comunidade académica (Universidade de Berkeley)
  - variantes comerciais: Ultrix, System V, IRIX, Solaris

Cronologia dos sistemas de operação (3)

- Primeira geração de computadores pessoais (1970–1980)
  - recursos escassos: um utilizador, uma tarefa
  - interface textual (e.g. interpretador de comandos MS-DOS)
  - combinado com uma linguagem de programação (BASIC)
  - específicos de cada modelo de computador e.g. Apple II, IBM PC

Cronologia dos sistemas de operação (4)

- Primeiras interfaces gráficas (1984–1990)
  - aplicações associadas a janelas separadas
  - apresentação de programas e ficheiros usando ícones
  - seleção visual usando o "rato"
  - facilita o acesso aos utilizadores
  - um utilizador, multi-tarefa
  - associadas a um modelo de *hardware* específico

Cronologia dos sistemas de operação (5)

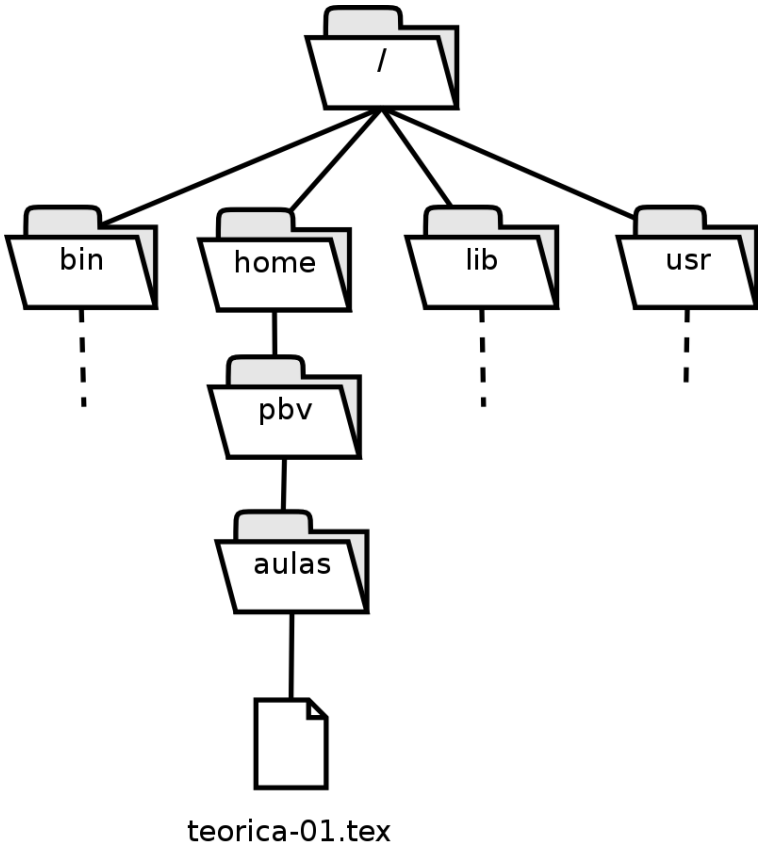
- Atualmente:
  - computadores pessoais com mais recursos do que os antigos "*mainframes*"
  - ligados em redes locais e globais (internet)
  - sistemas multi-utilizador, multi-tarefa
  - maior independência do *hardware*
  - separação entre o núcleo e a interface gráfica
  - UNIX em PCs: GNU/Linux, Free BSD, MacOS X

O sistema UNIX

- organizado num *núcleo* (*kernel*) e vários *processos*
  - núcleo: tem acesso direto ao *hardware*
  - processos: pedem recursos ao núcleo (e.g. consultar ficheiros)
- cada utilizador:
  - identificado por um "*login*" e.g. `pbv`
  - autenticado por uma *palavra-passe* (secreta)
  - área pessoal para ficheiros: "*home directory*"
- um utilizador especial: `root`
  - único que pode alterar configurações de sistema
  - único que pode acrescentar/remover utilizadores
  - alguns utilizadores podem executar comandos em *modo root*

Organização de ficheiros

- ficheiros: textos, imagens, programas...
- identificados por nomes
- estruturados em diretórios hierárquicos e.g. {`/home/pbv/aulas/teorica-01.tex`}
- permissões associadas a cada ficheiro: leitura, escrita, execução



Processos

- processo: execução dum programa num determinado contexto (utilizador e dados)
- aparência de vários processos a "correr" em simultâneo
- processos de utilizadores comuns:
  aplicações, editores, compiladores, interpretadores, etc.
- processos de sistema: interface gráfica, servidores (WWW, email, ssh), etc.
- filosofia UNIX: núcleo simples, tudo o resto são processos!

Interpretador de comandos UNIX

- "*shell*": interface textual para executar comandos UNIX
- cada comando é (normalmente) um programa em `/bin` ou `/usr/bin`
- sintaxe típica:
  comando [*opções*] [*arg1 arg2* ...]
  - exemplo:

```
$ ls ~
Desktop   Pictures   Public
$
```

- cada comando é executado com um novo processo
- associado ao diretório corrente ("*working directory*")

Alguns comandos úteis

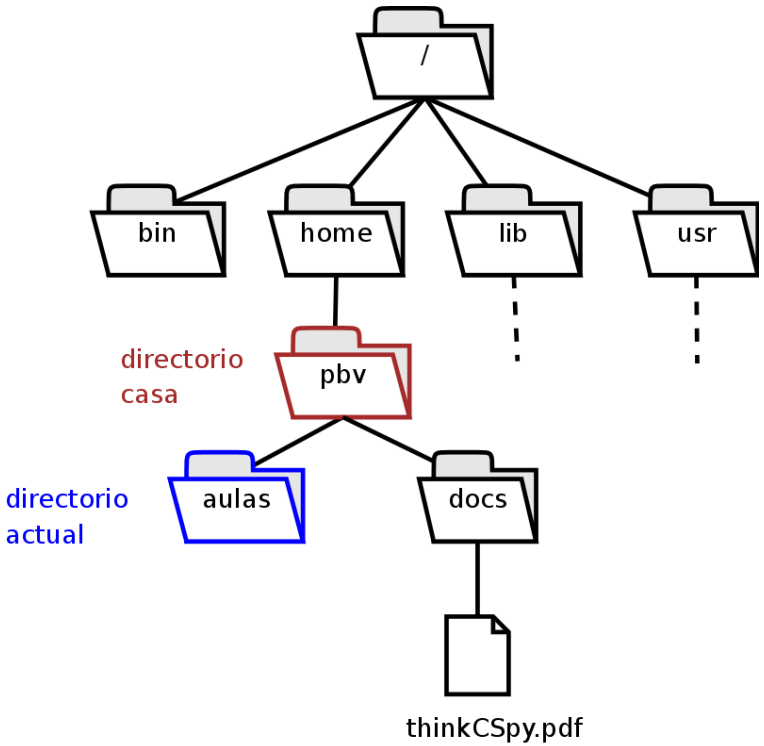
`ls`	listar ficheiros no diretório atual
`pwd`	imprime o diretório atual
`cd`	mudar o diretório atual
`mkdir`	criar um novo diretório
`rmdir`	remover um diretório (vazio)
`cp`	copiar ficheiro
`mv`	mover/mudar nome de um ficheiro
`rm`	remover um ficheiro
`less`	mostrar um ficheiro de texto página-a-página
`ps`	listar processos (do utilizador ou do sistema)
`man`	mostrar manual de um comando

Caminhos absolutos e relativos

. diretório actual

.. diretório pai

~ diretório casa



- Caminhos:
  - absoluto: `/home/pbv/aulas/docs/thinkCSpy.pdf`
  - relativo: `../docs/thinkCSpy.pdf`
  - relativo à casa: `~/docs/thinkCSpy.pdf`

"How to Think Like a Computer Scientist"

"How to Think Like a Computer Scientist"

- Competências:
  - matemática → utilização de linguagens formais para descrever ideias
  - engenharia → desenhar, construir sistemas a partir de componentes, avaliar alternativas
  - ciências → observar comportmento de sistemas complexos, formular hipóteses, testar previsões
- Competência mais importante: *resolução de problemas*
  - capacidade de formular problemas
  - criatividade para encontrar soluções
  - exprimir solução de forma clara e precisa

A linguagem de programação Python

- *Python*: linguagem de alto nível
  - forte abstração relativamente aos detalhes do computador
  - usa elementos da linguagem natural
  - outros exemplos: C++, C#, Java, ...
- Linguagens de *baixo nível:*
  - linguagens máquina/assembly
  - são as linguagens de os computadores executam
  - linguagens de alto nível têm de ser *traduzidas* para poderem ser executadas num computador
- Vantagens de linguagens de alto nível:
  1. mais fáceis de programar
     - rapidez de programação
     - programas curtos e fáceis de ler
     - menos erros
  2. portabilidade
     - podem ser executadas em computadores diferentes

Linguagens de baixo nível: linguagem máquina

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

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

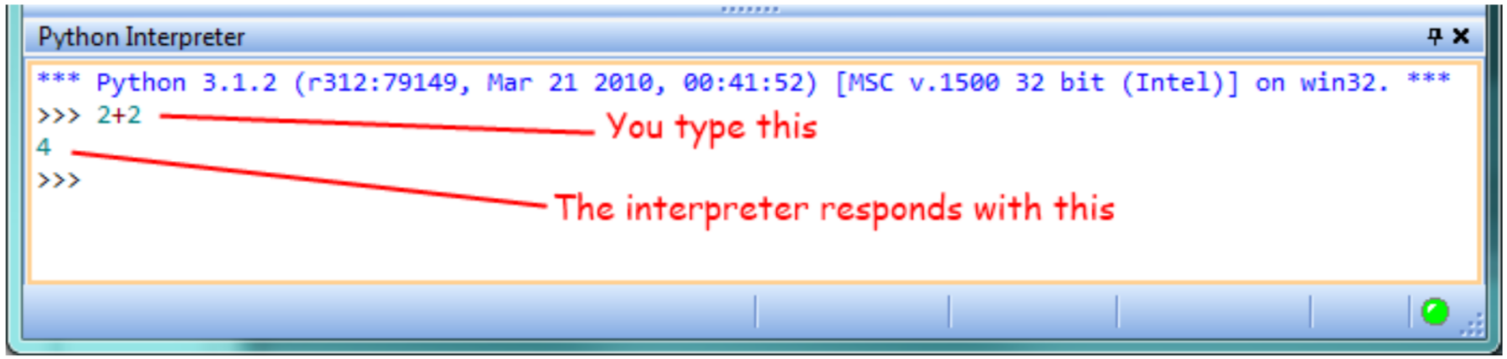
```
55                  push   %ebp
89 e5               mov    %esp,%ebp
83 ec 20            sub    $0x20,%esp
83 7d 0c 00         cmpl   $0x0,0xc(%ebp)
75 0f               jne    1b
...                 ...
```

  - representação do código máquina em mnemónicas (texto)
  - mais clara, para humanos, do que a linguagem máquina
  - traduzida para código máquina por um programa *assemblador*
  - continua a ser específica para cada computador/processador
  - muito próxima da máquina:
    - desenvolvimento lento, fastidioso, susceptível de erros

Interpretador de Python

Python é uma linguagem de alto nível.
Pode ser usada em

- *modo imediato/interativo* → escrevemos instruções e recebemos *feedback* imediato



  - >>> → *prompt*, usado para indicar que está à espera de instruções
  - conveniente para testar pequenos segmentos de código
  - Control-D ou `quit()` para terminar
- *modo de script* →
  - executa série de instruções escritas num ficheiro de texto
  - usado sempre, exceto para testar uma ou duas linhas de código

O que é um programa?

Um programa é uma sequência de instruções que determina como efetuar um cálculo

- matemático
- simbólico
- ...

Instruções básicas, presentes em praticamente todas as linguagens:

- input → obter dados do teclado ou de outro dispositivo
- output → escrever dados no ecrã ou noutro dispositivo
- cálculo matemático → operações como adição e multiplicação
- execução condicional → testar uma condição e decidir quais as instruções a executar de acordo com o resultado
- repetição → executar um conjunto de instruções repetidamente, geralmente com pequenas variantes

Mais detalhes quando virmos algoritmos

Debugging

- Programação é um processo complexo, suscetível a erros
- Erros de programação $\rightarrow$ bugs
  - sua correção $\rightarrow$ *debugging*
- Tipos de erro:
  - erros de sintaxe
  - erros de execução *(runtime errors)*
  - erros semânticos

Erros de sintaxe

Sintaxe

- regras de formação (gramática)
- num programa: *estrutura*, regras sobre essa estrutura
- Exemplo: expressões aritméticas
  - $3 \times (2 + 4) \rightarrow$ correcto
  - $3 + \times 24) \rightarrow$ erro de sintaxe

Erros de sintaxe:

- Linguagem natural $\rightarrow$ alguns erros são aceitáveis
- Em Python:
  - dão origem a uma mensagem de erro
  - impedem a execução do programa

Erros de execução *(runtime errors)*

- Aparecem durante a execução de um programa
- Indicam que algo excecional aconteceu $\rightarrow$ "exceções" (*exceptions*)
- Raros em programas simples

Erros semânticos

Semântica

- significado ou operação associados a expressão/programa
- Exemplo: expressões aritméticas
  - $3 \times (2 + 4) \rightarrow 18$

Erros semânticos:

- o programa é executado sem mensagens de erro
- mas não fará o que se pretende...
- o *significado/semântica* do programa não estão corretos
- exemplo: no caso anterior, escrever (incorretamente)
  - $3 \times 2 + 4$
- frequentemente, são difíceis de localizar e corrigir

Debugging experimental

- Uma das competências mais importantes que aqui desenvolveremos
- Por vezes frustante, mas intelectualmente interessante
- Trabalho de detetive
  - com base em indícios, inferir o que causa os resultados
- Ciência experimental:
  - com base numa hipótese sobre o que está errado, modificar o programa e tentar de novo
    - hipótese correta → um passo em frente para a versão final
    - hipótese errada → encontar nova ideia
- Alguns programadores:
  - programação e debugging ao mesmo tempo
  - debugging desde o princípio até o programa fazer o que se pretende
  - programa *faz qualquer coisa* desde o princípio
    - funciona sempre, funcionalidade/correção crescente

Linguagens formais e naturais

- Linguagens naturais
  - o que as pessoas falam: português, inglês, ...
  - não foram *desenhadas*; evoluíram naturalmente
- Linguagens formais
  - foram desenhadas explicitamente para fins específicos
  - exemplos:
    - *notação matemática*, para formular relações entre números e símbolos
    - *notação química*, para representar estrutura de moléculas
    - *linguagens de programação*, para exprimir cálculos/computação

Linguagens formais

Têm, normalmente, regras de sintaxe rígidas

área	expressão	sintaxe	semântica
matemática	$3 + 3 \times 6$	correta	21
matemática	$3 = \#6+$	erro	
química	H_2O	correta	água
química	$2R^Z$	erro	

Regras: baseadas em

- tokens (símbolos) → elementos básicos da linguagem
  - palavras, números, parêntesis, vírgulas, ...
  - exemplo:
    `print("Happy New Year for",2013)` → *6 tokens:*
    1. nome da função
    2. abertura de parêntesis
    3. cadeia de carateres (*string*)
    4. vírgula
    5. número
    6. fecho de parêntesis
- estrutura → a forma como os *tokens* são encadeados
  - exemplo: `print("Happy New Year for" 2013)` → *falta vírgula*
    - os tokens continuariam válidos mas a estrutura está incorreta

Parsing: determinar a estrutura de uma frase/instrução

- em linguagens naturais → inconsciente

Linguagens formais e naturais: diferenças

linguagens	naturais	formais
ambiguidade	elevada	reduzida
redundância	elevada	reduzida
literalidade	reduzida	elevada

O primeiro programa

1. Criar o ficheiro de texto `programa.py` com o seguinte conteúdo:

```
print("Ola, mundo!")
```

2. Executamos no terminal com

```
1 python3 programa.py
```

e obtemos o output

```
Ola, mundo!
```

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

Comentários

- Partes mais difíceis de um programa devem ser explicadas → comentários
- Permitem explicar em linguagem natural o que o programa/parte do código está a fazer
- Comentários são ignorados pelo interpretador de Python
- Em Python, comentários começam com #
- Tudo o que está à direita é ignorado

```
1  # Este programa imprime uma saudação
2  # A linguagem utilizada é Python
3  print("Ola, mundo!")     # é realmente simples...
```

O segundo programa

1. Criar o ficheiro de texto `inout.py` com o seguinte conteúdo:

```
name = input("What's your name? ")
print("Hello,", name, "!")
```

2. Executamos no terminal com

```
1 python3 inout.py
```

$\rightarrow$ verifique o que se obtém

O segundo programa

1. Criar o ficheiro de texto `inout.py` com o seguinte conteúdo:

```
name = input("What's your name? ")
print("Hello,", name, "!")
```

2. Executamos no terminal com

```
1 python3 inout.py
```

→ verifique o que se obtém

Nota: pode-se usar `input` sem o *prompt* como argumento:

```
x = input()
print("You entered", x)
```

Noções estudadas

algorithm
bug
comment
debugging
exception
formal language
high-level language
immediate mode
interpreter
low-level language
natural language
object code
parse
portability
print function
problem solving
program
Python shell
runtime error
script
semantic error
source code
syntax
syntax error
token

Próxima aula

- "Variáveis, expressões, instruções"