

Laboratórios de Informática

Sebenta Prática 2017–2018

Docentes

João Paulo Barraca <jpbarraca@ua.pt>

António M. Adrego da Rocha <adrego@ua.pt>

Óscar Mortágua Pereira <omp@ua.pt>

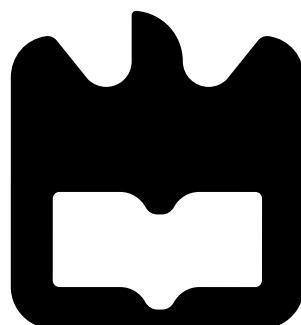
António Neves <an@ua.pt>

Pétia Georgieva <petia@ua.pt>

Vitor Cunha <vitorcunha@ua.pt>

José Duarte <hfduarte@ua.pt>

12 de Março de 2018



Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

Conteúdo

1	Introdução ao Ambiente Linux	1
1.1	Introdução	1
1.2	Instalação de sistemas operativos	2
1.3	A Linha de Comandos UNIX	3
1.4	Sistema de Ficheiros	9
1.5	Edição de ficheiros de texto	15
1.6	Processos	18
1.7	Administração básica	20
1.8	Para aprofundar o tema	24
2	Máquinas Virtuais	26
2.1	Introdução	26
2.2	Conceitos e terminologia	27
2.3	Instalação de uma máquina virtual	27
2.4	Instalação do sistema operativo na máquina virtual	31
2.5	Atualização e instalação de software	42
2.6	Partilha de ficheiros entre hospedeiro e convidado	44
2.7	Duplicação de máquinas virtuais	45
2.8	Para aprofundar o tema	46
3	Produção de documentos com L ^A T _E X	48
3.1	Introdução	49
3.2	Ações de preparação	49
3.3	Compilação de documentos L ^A T _E X	50
3.4	Estrutura obrigatória de um documento	51
3.5	Caracteres especiais do L ^A T _E X	52
3.6	Funcionalidades adicionais (linguísticas)	55
3.7	Dimensão das letras	57
3.8	Elementos estruturais de documentos	58
3.9	Inclusão de figuras	69
3.10	Índices de conteúdos, de figuras e de tabelas	71
3.11	Referências bibliográficas	72
3.12	Visão global da geração de documentos L ^A T _E X	76

3.13	Para aprofundar	78
4	Redes de Comunicações	79
4.1	Introdução	79
4.2	Configuração de rede de um computador	80
4.3	Tabela de Endereços Físicos	83
4.4	Tradução de nomes em endereços IP	83
4.5	Conectividade e rotas	85
4.6	Identificação da entidade responsável por uma máquina	87
4.7	Transmissão de informação em redes: ping	87
4.8	Transmissão de informação em redes: conteúdo HTTP	88
4.9	Acesso a servidores remotos	89
4.10	Para aprofundar o tema	94
5	Colaboração em Projetos	96
5.1	Introdução	96
5.2	A plataforma CodeUA	97
5.3	Sistemas de controlo de versões	106
5.4	Para aprofundar	120
6	Ferramentas de Automação	121
6.1	Automação em Projetos	121
6.2	GNU make	123
6.3	GNU Automake e Autoconf	130
7	Conceitos elementares de HTML	139
7.1	Protocolo HTTP	139
7.2	Documentos HTML	141
7.3	Utilização de um servidor HTTP	154
7.4	Para aprofundar	155
8	Conceitos elementares de CSS	156
8.1	Introdução	156
8.2	Sintaxe e aplicação de estilos	156
8.3	Seletores, identificadores e classes	162
8.4	Margens, Bordas e Espaçamentos	167
8.5	Texto	170
8.6	Pacote de estilos <i>Twitter Bootstrap</i>	174
8.7	Para aprofundar	178
9	Conceitos elementares de JavaScript	179
9.1	JavaScript	179
9.2	Inclusão numa página	180
9.3	Sintaxe	181

9.4	Interação com o DOM	186
9.5	Temporizadores	191
9.6	Para aprofundar	194
10	Integração de componentes em páginas Web	195
10.1	Páginas Web	195
10.2	Componentes	197
10.3	Gráficos	201
10.4	Mapas	204
10.5	Comentários	208
10.6	Para aprofundar	209
11	Programação em Python	210
11.1	Linguagem Python	210
11.2	Características Básicas	211
11.3	Ficheiros	226
11.4	Para aprofundar o tema	229
12	Criptografia em Python	231
12.1	Introdução	231
12.2	Funções de síntese	231
12.3	Biblioteca pycrypto	234
12.4	Cifras simétricas	236
12.5	Cifras Assimétricas	241
12.6	Para Explorar	245
13	Testes e Depuração	246
13.1	Introdução	246
13.2	Desenvolvimento guiado por testes	246
13.3	Testes Unitários	248
13.4	Testes Funcionais	252
13.5	Depuração	253
13.6	Para Aprofundar	258
14	Comunicação entre aplicações	259
14.1	Introdução	259
14.2	Conceitos de comunicação	260
14.3	Sockets não orientados à ligação (UDP)	262
14.4	Sockets orientados à ligação (TCP)	265
14.5	Servidor de Mensagens Instantâneas	268
14.6	Para Aprofundar	270
15	Documentos	272
15.1	Introdução	272

15.2 CSV	273
15.3 JSON	276
15.4 XML	278
15.5 Para Aprofundar	284
16 Aplicações e Serviços Web	286
16.1 Introdução	286
16.2 Servidores Web	287
16.3 Serviços Web	294
16.4 Para Aprofundar	296
17 Bases de Dados	297
17.1 Bases de Dados	297
17.2 Acesso Programático	304
17.3 Para Aprofundar	308
18 Representação de Som	310
18.1 Princípios de acústica	310
18.2 Representação de informação sonora	312
18.3 Operações sobre som	319
18.4 Para aprofundar	327
19 Aplicações Móveis Web	328
19.1 Introdução	328
19.2 Uma aplicação simples	329
19.3 Localização	331
19.4 Comunicação com serviços externos	332
19.5 Acesso a imagens	334
20 Representação de Imagem	338
20.1 Introdução	338
20.2 Imagens	339
20.3 Formatos de ficheiros	341
20.4 Representação de cor	343
20.5 Efeitos sobre imagens	346
20.6 Marcação de imagens	355
21 Microcontroladores	359
21.1 Introdução	359
21.2 A linguagem de programação C	363
21.3 Programação de microcontroladores	365
21.4 Periféricos e portos de entrada/saída	368
21.5 Contadores e tempo	369
21.6 Entradas digitais	372

21.7 Entradas analógicas	373
22 Micro-controladores e a plataforma MicroRato	376
22.1 Introdução	376
22.2 Linguagem <i>C</i>	379
22.3 Sensores Analógicos e Digitais	381
22.4 Atuadores	386
22.5 Tarefas	388

Listas de Figuras

1.1	Algumas das distribuições mais populares de Linux.	3
1.2	Prompt apresentado pela bash	6
1.3	Indicação de erro pela bash	6
1.4	Estrutura do sistema de ficheiros <i>Linux</i>	10
1.5	Resultado dos comandos ls e pwd	11
1.6	Listagem completa dos ficheiro na área pessoal	12
1.7	Resultado da execução do comando ps Tu para mostrar os processos associados com a sessão actual.	18
1.8	Resultado da execução do comando whoami	21
1.9	Resultado da execução do comando sudo , inspecionando qual o utilizador em uso.	22
1.10	Resultado da execução do comando id , inspecionando qual o utilizador em uso (neste exemplo o utilizador tem o nome linux).	22
1.11	Processo de instalação da aplicação cowsay	25
3.1	Legenda da figura	64
3.2	Novo logotipo da Universidade de Aveiro	70
3.3	Logotipo da Universidade de Aveiro: a) na dimensão real, b) com 3cm de altura, c) com 20mm de largura, d) com altura e largura reduzidas a 1/2 e simultaneamente rodado 90° e e) com uma modificação anamórfica da altura e da largura.	71
3.4	Exemplo de ficheiro de registos BibTeX contendo dois registos.	73
3.5	Exemplo do aspeto final da bibliografia consoante o estilo usado. A etiqueta de cada registo é exatamente igual ao que aparecerá ao longo do texto, no local onde for feita a respetiva referência.	75
3.6	Ficheiros e comandos envolvidos na geração de um documento Portable Document Format (PDF) a partir de fontes em L ^A T _E X e BibTeX	77
4.1	Resultado do comando arp -an	83
4.2	Resultado do comando traceroute www.google.pt executado a partir de um servidor na Universidade de Aveiro.	86
5.1	Página de entrada da plataforma CodeUA	98
5.2	Página de introdução de uma notícia.	100
5.3	Página de apresentação das notícias.	101
5.4	Página de criação de uma nova tarefa.	102

5.5	Página de listagem das tarefas	103
5.6	Diagrama de Gantt para o primeiro trabalho de aprofundamento de conhecimentos	105
5.7	Fluxo de eventos no ciclo de vida de um projeto gerido por Git.	109
7.1	Organização típica de uma página Web.	151
7.2	Utilização de uma divisão para implementar um <i>popup</i>	153
8.1	Simples cabeçalho <h1> com um estilo.	159
8.2	Exemplo de uma tabela com séries de temperatura de vários anos	163
8.3	Exemplo de uma tabela com um elemento estilizado de forma individual.	164
8.4	Exemplo de uma tabela com um elemento estilizado por classes.	166
8.5	Modelo de caixas em CSS.	168
8.6	Modelo de caixas em CSS com padding.	169
8.7	Modelo de caixas em CSS sem padding.	169
8.8	Exemplo de escolha de tipos de letra.	171
8.9	Exemplo de utilização de um tipo de letra externo.	172
8.10	Exemplo de utilização do Twitter Bootstrap.	175
8.11	Botões em Twitter Bootstrap.	176
8.12	Painel em Twitter Bootstrap.	177
9.1	Estrutura hierárquica do DOM	186
10.1	Barra de navegação	199
10.2	<i>Popup</i>	200
10.3	Gráfico de linhas	202
10.4	Mapa utilizando Leaflet JS	205
13.1	Fluxo de processos na metodologia TDD.	247
14.1	Modelo Cliente-Servidor	260
14.2	Vários pontos de comunicação com um navegador Web.	261
14.3	Sequência de primitivas utilizadas num Socket UDP.	263
14.4	Sequência de primitivas utilizadas num Socket TCP.	266
18.1	Forma de onda de um som medida ao longo do tempo num ponto do espaço.	311
18.2	Várias notas de um piano capturadas num espetrograma.	312
18.3	Conversão analógico-digital: amostragem e quantização. O sinal analógico original é representado em cinzento. As linhas verticais azuis representam os instantes de amostragem. A quantização aproxima as amplitudes por valores de um conjunto finito.	313
18.4	Estrutura dum ficheiro WAVE.	315
18.5	Número de telefone codificado em DTMF	318
18.6	Exemplo de <i>Fade In</i> e <i>Fade Out</i>	322
20.1	Composição de uma imagem digital.	339
20.2	Compressão com JPG.	342
20.3	Valores RGB do logótipo da UA	344
20.4	Figura original em RGB e com os canais R e G trocados.	347
20.5	Figura original em RGB e em tons de cinza (L).	348
20.6	Figura com intensidade aumentada ($f=1.5$) ou diminuída ($f=0.5$).	349

20.7	Correção de gama para um CRT (Fonte Wikimedia Foundation)	350
20.8	Figura com gama de 2.2 ou de 0.5.	350
20.9	Figura com saturação aumentada por 1.5 ou reduzida para 0.5.	351
20.10	Figura convertida para tons Sépia ou Lomo.	352
20.11	Bordas detectadas numa imagem.	353
20.12	Vignette comum (esquerda) e deslocado para a direita (direita)	354
20.13	Fotografia da UA com o símbolo em marca de água.	356
20.14	Imagen com marca de água usando técnicas de estanografia e marca de água recuperada.	357
21.1	O Microchip PIC32MX795F512H e o Intel i5, aproximadamente à mesma escala. (As imagens pertencem aos seus fabricantes.)	360
21.2	Placa de desenvolvimento DETPIC32.	361
21.3	Kit fornecido para exploração de microcontroladores.	362
22.1	Final de uma prova do Micro Rato	377
22.2	O Microchip PIC32MX795F512H e o Intel i5. Imagens pertencem aos seus fabricantes.	378
22.3	Sensores digitais de chão.	382
22.4	Sensores de distância por medição de ângulo da reflexão.	385
22.5	LEDs e motores	386

Introdução

Esta sebenta engloba os guiões para cada aula Teórico-Prática de Laboratórios de Informática. O objetivo destes guiões é o de fomentar a aprendizagem de várias matérias, focadas na descoberta do mundo da informática, através de experiências práticas. Durante as mesmas, os alunos devem aplicar o seu raciocínio crítico e, desejavelmente, ir mais além na exploração de tópicos relacionados com os de cada tema.

Os temas estão divididas em dois blocos principais. O primeiro foca-se em questões mais genéricas, tais como ferramentas, escrita técnica e automação de tarefas. O segundo bloco foca-se em questões específicas do mundo informático, utilizando a linguagem Python como ferramenta para a exploração.

Introdução ao Ambiente Linux

Objetivos:

- Introdução ao Linux
- Interpretador de comandos
- Sistema de ficheiros
- Processos
- Utilizadores
- Gestão de aplicações

1.1 Introdução

O sistema operativo Linux, criado nos anos 90, é uma implementação livre de *UNIX*, um sistema operativo dos anos 70, que se popularizou devido à sua portabilidade para diferentes arquitecturas de hardware. Existem muitas distribuições de Linux disponíveis, mas todas têm em comum o *kernel* (núcleo), que foi desenvolvido por Linus Torvalds. Neste trabalho vão-se utilizar comandos *UNIX* para que o aluno se familiarize com o sistema operativo Linux. Por um lado, principalmente, vai-se trabalhar na linha de comandos; mas por outro lado, também iremos ver como alguns passos podem ser feitos quer na linha de comandos quer em ambiente gráfico. O sistema operativo Linux tem várias vantagens no seu kernel comparativamente com o sistema operativo Windows. O Linux é um sistema robusto e estável, que suporta tarefas de cálculo e actividades de peso com eficácia; é um sistema acessível, pois tem custo zero - é gratuito; é flexível, pois como o seu código fonte é aberto, o utilizador pode alterar variados aspectos à sua vontade, isto após ter várias distribuições por onde escolher; também é seguro, pois desde o *kernel* até à versão final a segurança é um aspecto trabalhado pelos co-autores de cada

distribuição; este sistema operativo não precisa de anti-virus, raramente bloqueia e é muito rápido e eficaz.

1.2 Instalação de sistemas operativos

O processo normal de instalação de um sistema operativo é feito do seguinte modo:

1. O sistema instalador é disponibilizado num suporte móvel (Compact Disk (CD), Digital Versatile Disk (DVD), memória *flash*, etc.), total ou parcialmente. Quando é disponibilizado parcialmente, a parte em falta é obtida de repositórios da Internet.
2. O sistema instalador é executado logo após o arranque da máquina, sendo ativado pelo sistema de controlo do arranque da máquina (*boot loader*).
3. O sistema instalador escolhe um disco rígido da máquina, ou um conjunto de partições de discos da máquina, para aí criar os sistemas de ficheiros que irão ser usados pelo sistema operativo que irá ser instalado. Normalmente usam-se duas partições diferentes, uma com os ficheiros que normalmente vemos no sistema de ficheiros, outra designada como *swap* que serve para apoio à gestão da memória virtual. Esta última partição pode ser substituída por um ficheiro.
4. Após a instalação do sistema operativo no disco rígido, a partição de arranque dessa instalação é marcada como sendo de arranque (*boot*) e o sistema está pronto para ser reiniciado. Por vezes no arranque de um sistema instalado é iniciado primeiro um sistema de controlo dos sistemas operativos a iniciar, de que é exemplo o **Grub**: arranque da BIOS → seleção do dispositivo de arranque → carregamento do módulo de arranque do dispositivo → seleção do sistema operativo a arrancar (opcional) → arranque do sistema operativo escolhido.

Sistemas *live*

Há, contudo, variantes a este processo base. Um deles consiste no arranque dos sistemas ditos *live* (ou distribuições *live*). Os sistemas *live* são sistemas que arrancam como os demais mas não alteram nada na máquina de forma definitiva. Em particular, não usam qualquer repositório persistente da máquina (v.g. discos rígidos) para guardar qualquer informação. Portanto, estes sistemas podem-se executar em máquinas sem disco rígido.

Uma distribuição *live*, de que há inúmeros exemplos para Linux¹, é uma imagem de CD (ficheiro ISO) que pode ser usada para arrancar um sistema *live* numa máquina, a partir do seu leitor de CD.

¹http://en.wikipedia.org/wiki/List_of_live_CDs

Atualmente muitas das distribuições *live* possuem uma funcionalidade 2-em-1: permitem o arranque de uma versão *live* normal, mas essa permite depois criar uma instalação no disco rígido da máquina. Esta faceta será explorada neste trabalho.

Distribuições populares de Linux

No mundo linux há várias distribuições populares, tais como as listadas na Figura 1.1: debian, a partir da qual se desenvolveram ubuntu, lubuntu, linuxmint, etc.; redhat a partir da qual se desenvolveram fedora, centOS, etc., slackware a partir da qual vem o openSUSE, etc.; archlinux e gentoo linux; entre muitas outras.

Nos computadores dos laboratórios de aulas está instalada uma versão de *Ubuntu*. O aluno também pode instalar o Ubuntu ou outra versão Linux em casa ou numa aula OT. Se optar pelo Ubuntu terá uma distribuição completamente funcional, com todas as vantagens do mundo *Ubuntu* (repositórios, suporte, etc.) e uma maior semelhança com o ambiente das aulas.

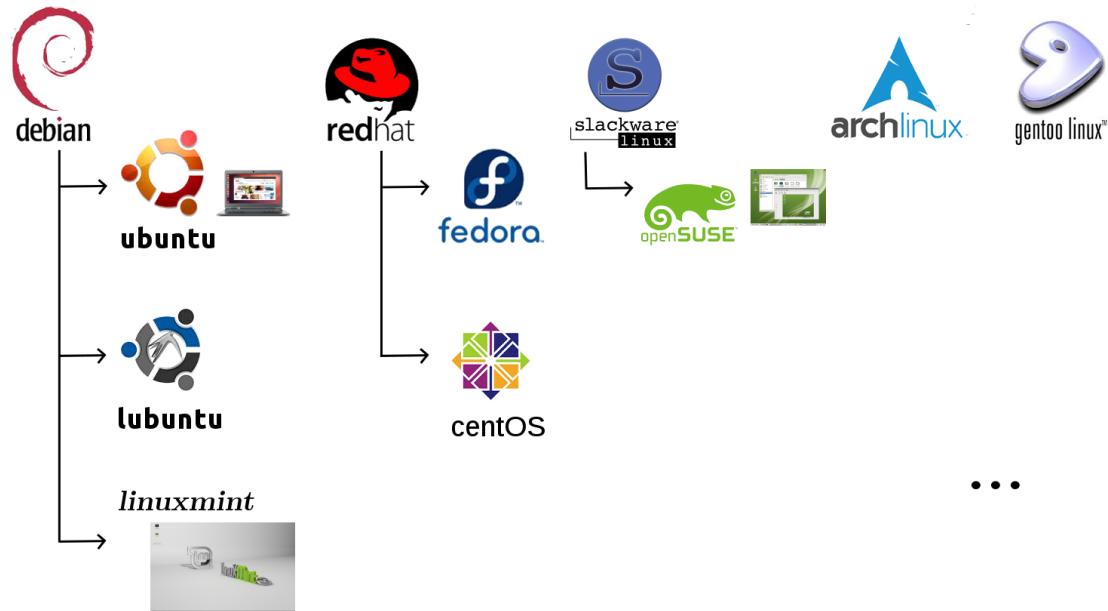


Figura 1.1: Algumas das distribuições mais populares de Linux.

1.3 A Linha de Comandos UNIX

Quando o sistema *UNIX* foi concebido, os computadores eram controlados essencialmente através de *consolas* ou *terminais* de texto: dispositivos dotados de um teclado e de um

ecrã onde se podia visualizar somente texto. A interação com o sistema fazia-se através da introdução de comandos escritos no teclado e da observação da resposta produzida no ecrã pelos programas executados.

Atualmente existem ambientes gráficos que correm sobre o *UNIX/Linux* e que permitem visualizar informação de texto e gráfica, e interagir por manipulação virtual de objetos gráficos recorrendo a um rato e ao teclado. É o caso do Sistema de Janelas **X**, ou simplesmente **X**², que está incluído em todas as distribuições usuais de Linux.

Apesar das novas formas de interação proporcionadas pelos ambientes gráficos, continua a ser possível e em certos casos preferível usar a interface de *linha de comandos* para muitas operações. No **X**, isto pode fazer-se usando um *emulador de terminal*, um programa que abre uma janela onde se podem introduzir comandos linha-a-linha e observar as respostas geradas tal como num terminal de texto à moda antiga.

Interfaces de texto e gráficas

Os sistemas *Linux* geralmente simulam um ambiente de trabalho formado por 6 interfaces de texto (consolas) e 2 interfaces gráficas. Na prática todas estas interfaces coexistem sobre os mesmos equipamentos de interface com os utilizadores: ecrã, teclado, rato, etc. Muito embora estas interfaces estejam sempre ativas, elas não estão sempre visíveis: só uma delas está visível em cada instante.

Na maior parte dos casos os sistemas *Linux* apresentam uma interface gráfica quando iniciam, ou após a terminação da sessão de um utilizador. A comutação entre interfaces faz-se através das seguintes combinações de teclas:

interface gráfica /consola → consola : Ctrl + Alt + F_n, onde F_n é uma das teclas F1, F2, ..., F6. O valor de n indica o número da consola que se pretende usar (F1 para a consola 1, etc.).

consola → interface gráfica : Alt + F7 (para a interface gráfica por omissão) ou Alt + F8 (para a interface gráfica secundária, normalmente inativa).

Após o arranque do *Linux*, cada consola apresenta uma interface muito simples de *login*, na qual são apresentadas algumas características do sistema (sistema operativo, nome da máquina, nome da interface (tty_n), e o que se pretende que o utilizador introduza: o *nome-de-utilizador* e a *senha*). Após um *login* bem sucedido, o utilizador pode continuar a trabalhar na linha de comandos que lhe é apresentada. Para terminar a sua sessão, o

²<http://www.x.org>

utilizador deverá fazer *logout* usando o comando **exit**. Após este comando, a consola voltará a apresentar a interface de *login* antes observada.

Exercício 1.1

Mude para uma consola de texto, faça *login* na mesma, execute o comando **whoami**, e execute em seguida o comando **exit**. Repita o processo de *login* e faça desta vez *logout* carregando na combinação de teclas Ctrl + D. Este conjunto de teclas é designado por **EOF** (*End-Of-File*).

Uma das consolas terá o ambiente gráfico ao qual pode voltar.

Interpretador de comandos

A maioria das distribuições *Linux* fornece uma Command Line Interface (CLI) através do programa **bash**.³ Como não poderia deixar de ser, existem alternativas, tais como: **ksh**, **tsh**, **zsh**, **dash** ou **fish**. Todos estes programas fornecem uma CLI ao utilizador, mas possuem funcionalidades ligeiramente distintas.

Quando se utiliza um ambiente gráfico existem programas denominados por *emuladores de terminal*, responsáveis por na realidade apresentarem uma janela de interacção com o utilizador, enviando teclas para a *Shell* e apresentando o conteúdo ao utilizador.

Existem vários emuladores de terminal, sendo que as distribuições fornecem sempre vários para escolha, tais como: **xterm**, **konsole**, **gnome-terminal** ou **lxterminal**.

Exercício 1.2

Usando o mecanismo de procura ou os menus do ambiente gráfico, procure um emulador de terminal e execute-o.

Procure um dos outros emuladores e execute-o.

Verifique que embora o texto apresentado seja semelhante, os emuladores apresentam aspectos ligeiramente diferentes e têm menus e funcionalidades de configuração diferentes. Descubra como pode mudar a cor de fundo e o tamanho de letra em cada um dos emuladores.

Execute o comando **echo \$SHELL** para verificar qual a *Shell* em utilização em cada terminal.

³O nome advém do facto de o programa **bash** ser uma versão melhorada do programa **Bourne shell**, desenvolvido por Steve Bourne.

Ao executar o emulador de terminal, pode reparar que o interface é bastante simples, sempre apresentada apenas uma pequena informação tal representado na Figura 1.2.

```
linux@linux:~$ █
```

Figura 1.2: *Prompt* apresentado pela **bash**.

O formato utilizado neste *Prompt* é o de **utilizador @ nome da maquina : directorio actual \$**. Ou seja, neste caso em particular, utilizador **labi**, máquina **labi**, directório actual **~**. Mais à frente, na Secção 1.4 iremos ver o que representa o **~**.

Se escrever qualquer coisa aleatória, exemplo **sdgt234rsfd**, seguido de **ENTER**, verá que a **bash** lhe indica uma situação de erro, tornando a pedir um comando válido.

```
linux@linux:~$ sdgt234rsfd
bash: sdgt234rsfd: comando não reconhecido
linux@linux:~$ █
```

Figura 1.3: Indicação de erro pela **bash**.

Como pode notar, por muitos comandos errados que introduza, a **bash** irá sempre pedir de novo um comando, sem que isso traga algum prejuízo para o sistema.

Observe também a resposta foi impressa imediatamente a seguir à linha do comando, de forma concisa, sem distrações nem grandes explicações. Este comportamento é usual em muitos comandos *UNIX* e é típico de um certo estilo defendido pelos criadores deste sistema.

Simples, mas eficaz.

Exercício 1.3

Execute o comando **date** e observe o resultado.

Exercício 1.4

Execute o comando **cal** e observe o resultado. Descubra em que dia da semana nasceu, passando o mês e o ano como *argumentos* ao comando **cal**, por exemplo: **cal jan 1981**.

Formato dos comandos

Os comandos em *UNIX* têm sempre a forma:

```
comando argumento1 argumento2 ...
```

onde **comando** é o nome do programa a executar e os argumentos são cadeias de caracteres, que podem ser incluídas ou não, de acordo com a sintaxe esperada por esse programa.

Os argumentos podem ainda modificar o comportamento base do programa, designando-se nesse caso por opções (ou, por vezes, por *flags* em inglês, por pretenderm sinalizar algo de especial). Tipicamente as opções são indicadas de duas formas:

-x , onde *x* representa uma letra, maiúscula ou minúscula, ou um algarismo.

--nome-da-opção , onde *nome-da-opção* é um bloco de texto, sem espaços em branco, com a designação da opção.

Exercício 1.5

Execute o comando **date -u** para ver a hora atual no fuso horário UTC e observe o resultado. Execute o comando **date --utc** para obter o mesmo resultado.

Na linha de comandos é possível recuperar um comando indicado anteriormente usando as teclas de direção \uparrow e \downarrow . É possível depois editá-lo (alterá-lo) para produzir um novo comando (com argumentos diferentes, por exemplo). Outra funcionalidade muito útil é a possibilidade de o sistema completar automaticamente comandos ou argumentos parcialmente escritos usando a tecla **Tab**.

Um interpretador de comandos também mantém uma lista numerada com todos os comandos executados durante a sua execução. Esta lista pode ser consultada através do comando **history**. O resultado produzido por este comando é uma listagem, numerada, dos comandos executados pela ordem cronológica da sua execução. Os interpretadores de comandos permitem usar os números usados nessa listagem para recuperar (e executar novamente) os respetivos comandos, através do comando **!n**, onde *n* é o número de ordem do comando que se pretende recuperar. A recuperação e execução rápida do comando

exatamente anterior pode ser feita com o comando `!!`.

Exercício 1.6

Execute o comando `history` e observe o resultado.

Exercício 1.7

Execute o comando `!!` e observe o resultado. E se executar `!2?`

Exercício 1.8

Recupere um comando anterior usando as teclas de direção, edite-o e execute-o.

Exercício 1.9

Obtenha o número de ordem de um comando anterior e re-execute-o usando o comando que permite a indexação de um comando anterior (`!numero`).

Uma linha pode ser editada de forma elementar ou sofisticada. A forma elementar consiste em deslocar o cursor ao longo da linha, para trás ou para a frente, usando as teclas de direção `←` e `→`, adicionar novo texto à linha e apagar texto da linha, atrás do cursor, com a tecla `DELETE`.

Ajuda dos comandos

O que um comando faz, bem como a sintaxe e significado dos seus argumentos são decididos pelos seus programadores. Um utilizador médio tem de aprender a sintaxe e semântica de uma dezena ou duas de comandos e algumas das suas opções mais usuais, mas ninguém consegue memorizar os milhares de comandos disponíveis. Para uniformizar um pouco a sintaxe dos comandos e assim facilitar a aprendizagem, os programas seguem algumas convenções básicas:

1. Muitos programas aceitam opções de funcionamento quer no formato longo (`date --utc`), quer no formato curto, de uma letra só (`date -u`).

2. Várias opções no formato curto podem geralmente ser amalgamadas, e.g., `ls -l -a` equivale a `ls -la`.
3. Quase todos os comandos aceitam uma opção `--help`, por exemplo `date --help`, que serve apenas para mostrar um breve texto de ajuda.
4. Existe uma base de dados com manuais de utilização para todos os comandos disponíveis, acessível através do comando `man` seguido do nome do comando que se pretende consultar (e.g., `man date`).

Exercício 1.10

1. Considerando o comando `date` que vimos anteriormente, verifique o resultado de `date -u` e `date -utc`.
2. Aceda à ajuda do comando `date` através dos métodos descritos.
3. Aplique este método a outros comandos tais como `echo`, `true`, `false`.

Exercício 1.11

Verifique a utilidade do comando `apropos`. Pode usar como guia o parâmetro `successfully` e correlacionar o seu resultado com o comando `man`.

1.4 Sistema de Ficheiros

Os ficheiros e directórios são organizados seguindo o princípio de uma árvore com uma raiz (`/`) e directórios por baixo dessa raiz. Isto é semelhante ao utilizado no sistema operativo *Windows*, com a grande diferença que, em quanto o *Windows* considera que cada dispositivo (ex, Disco Rígido, CD) é uma unidade distinta, em *Linux* tal como na maioria dos restantes sistemas operativos, os diversos dispositivos encontram-se mapeados em directórios da mesma raiz. A Figura 1.4 demonstra a estrutura do sistema de ficheiros se pode encontrar na máquina virtual fornecida, e típica de um qualquer *Linux*.

Embora o *Linux* não obrigue a existência de uma estrutura específica, tipicamente o mesmo modelo é sempre utilizado. A Tabela 1.1 descreve o propósito de alguns destes directórios.

Cada utilizador possui um diretório próprio nesta árvore, a partir do qual pode (e deve) criar e gerir toda a sua sub-árvore de diretórios e ficheiros: é o chamado *diretório do utilizador* ou *home directory*. Após a operação de *login* o sistema coloca-se nesse diretório.

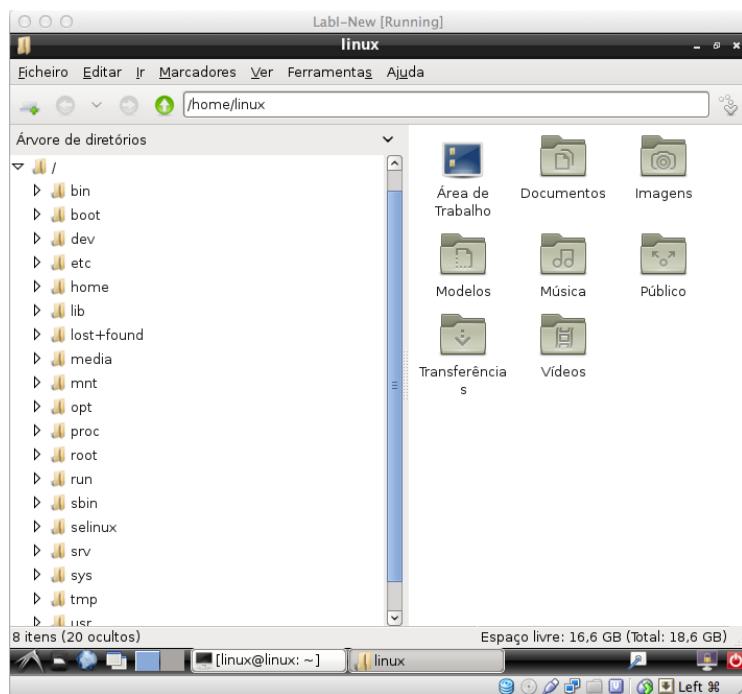


Figura 1.4: Estrutura do sistema de ficheiros *Linux*

Tabela 1.1: Principais directórios típicos do *Linux*

Directório	Descrição
/	Raiz do sistema de ficheiros
/bin	Executáveis essenciais do sistema.
/boot	Contem o <i>kernel</i> para iniciar o sistema.
/etc	Configurações.
/home	Áreas dos utilizadores.
/mnt	Pontos de montagem temporários.
/media	Pontos de montagem de unidades como os CDs.
/lib	Bibliotecas essenciais e módulos do <i>kernel</i> .
/usr	Bibliotecas e aplicações tipicamente usadas por utilizadores.
/sbin	Programas tipicamente utilizados pelo super-utilizador.
/tmp	Ficheiros temporários.
/var	Ficheiros frequentemente modificados (bases de dados, impressões).

Portanto neste momento deve ser esse o *diretório atual* (*current directory*). Tipicamente este directório é representado pelo caractere ~.

Para saber qual é o diretório atual execute o comando **pwd**⁴. Deve surgir um nome como indicado na Figura 1.5, que indica que está no diretório **labi** que é um subdiretório de **home** que é um subdiretório direto da raiz /.

Para listar o conteúdo do diretório atual execute o comando **ls**⁵. Deve ver uma lista dos ficheiros (e subdiretórios) contidos no seu diretório neste momento, tal como referido na Figura 1.5.

```
linux@linux:~$ pwd  
/home/linux  
linux@linux:~$ ls  
Área de Trabalho  Imagens  Música  Transferências  
Documentos        Modelos   Público  Vídeos  
linux@linux:~$ █
```

Figura 1.5: Resultado dos comandos **ls** e **pwd**

Dependendo da configuração do sistema, os nomes nesta listagem poderão aparecer com cores diferentes e/ou com uns caracteres especiais (/, @, *) no final, que servem para indicar o tipo de ficheiro mas de facto não fazem parte do seu nome.

(Num ambiente gráfico a mesma informação está disponível numa representação mais visual. Experimente, por exemplo, escolher *Árvores de directórios/labi* para ver o conteúdo do seu diretório pessoal.)

Ficheiros cujos nomes começam por “.” não são listados por omissão, são ficheiros *escondidos*, usados geralmente para guardar informações de configuração de diversos programas. Para listar todos os ficheiros de um diretório, incluindo os escondidos, deve executar a variante **ls -a**⁶

Por vezes é necessário listar alguns atributos dos ficheiros para além do nome. Pode fazê-lo executando as variantes **ls -l** ou **ls -la**, sendo que o resultado deverá ser semelhante ao apresentado na Figura 1.6.

Os principais atributos mostrados nestas listagens longas são:

Tipo de ficheiro identificado pelo primeiro carácter à esquerda, sendo **d** para diretório,
- para ficheiro normal, **l** para *soft link*, etc.

⁴Acrónimo de *print working directory*.

⁵Abreviatura de *list*.

⁶A opção **-a** indica que se devem listar todos (*all*) os elementos do diretório.

```

linux@linux:~$ ls -la
total 108
drwxr-xr-x 16 linux linux 4096 Set 26 19:03 .
drwxr-xr-x  3 root  root  4096 Set 21 18:56 ..
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Área de Trabalho
-rw-r--r--  1 linux linux  220 Set 21 18:56 .bash_logout
-rw-r--r--  1 linux linux 3637 Set 21 18:56 .bashrc
drwxrwxr-x  5 linux linux 4096 Set 26 15:17 .cache
drwx----- 18 linux linux 4096 Set 26 13:45 .config
drwx-----  3 linux linux 4096 Set 21 18:57 .dbus
-rw-r--r--  1 linux linux  23 Set 26 16:01 .dmrc
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Documentos
drwx-----  3 linux linux 4096 Set 26 16:01 .gconf
-rw-r--r--  1 linux linux 161 Set 26 13:01 .gtkrc-2.0
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Imagens
drwx-----  3 linux linux 4096 Set 21 18:57 .local
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Modelos
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Música
drwxr-xr-x  2 linux linux 4096 Set 21 23:54 .pip
-rw-r--r--  1 linux linux  675 Set 21 18:56 .profile
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Público
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Transferências
-rw-r----- 1 linux linux    5 Set 26 16:01 .vboxclient-clipboard.pid
-rw-r----- 1 linux linux    5 Set 26 16:01 .vboxclient-display.pid
-rw-r----- 1 linux linux    5 Set 26 16:01 .vboxclient-draganddrop.pid
-rw-r----- 1 linux linux    5 Set 26 16:01 .vboxclient-seamless.pid
drwxr-xr-x  2 linux linux 4096 Set 21 18:57 Vídeos
-rw-----  1 linux linux   50 Set 26 16:01 .Xauthority
-rw-r--r--  1 linux linux   14 Set 21 18:57 .xscreensaver
linux@linux:~$ █

```

Figura 1.6: Listagem completa dos ficheiros na área pessoal

Permissões representadas por 3 conjuntos de 3 caracteres. Indicam as permissões de leitura **r**, escrita **w** e execução/pesquisa **x** relativamente ao dono do ficheiro, aos outros elementos do mesmo grupo e aos restantes utilizadores da máquina.

Propriedade indica a que utilizador e a que grupo pertence o ficheiro.

Tamanho em número de bytes.

Data e hora da última modificação.

Nome do ficheiro.

Normalmente existe um *alias*⁷ **ll** equivalente ao comando **ls -l**.

Além do **ls** e variantes, existem outros comandos importantes para a observação e manipulação de diretórios, por exemplo:

⁷Um *alias* é um nome alternativo usado em representação de um determinado comando. São criados usando o comando interno **alias**.

cd — o diretório atual passa a ser o diretório do utilizador.

cd nome-do-dir — o diretório atual passa a ser o diretório **dir**.

mkdir nome-do-dir — cria um novo diretório chamado **dir**.

rmdir nome-do-dir — remove o diretório **dir**, desde que esteja vazio.

O argumento **dir** pode ser dado de uma forma absoluta ou relativa. Na forma absoluta, **dir** identifica o caminho (*path*) para o diretório pretendido a partir da raiz de todo o sistema de ficheiros; tem a forma **/subdir1/.../subdirN**. Na forma relativa, **dir** indica o caminho para o diretório pretendido a partir do diretório atual; tem a forma **subdir1/.../subdirN**.

Há dois nomes especiais para diretórios: “.” e “..” que representam respetivamente o diretório atual e o diretório pai, ou seja, o diretório ao qual o atual pertence.

Exercício 1.12

Execute os comandos seguintes e interprete os resultados:

```
cd /
pwd
cd /usr
cd ~
pwd
cd /usr/sbin
pwd
cd -
pwd
```

Exercício 1.13

Experimente utilizar o programa gráfico gestor de ficheiros para navegar pelos mesmos diretórios que no exercício anterior: **/**, **/usr**, **/usr/local/src**, etc.

Importante: Nos computadores das salas de aulas da UA, o subdiretório **arca** não é um diretório local do Computador Pessoal (PC) onde está a trabalhar; é na verdade a sua área privada de armazenamento no Arquivo Central de Dados (ARCA⁸), um servidor de

⁸<https://arcaweb.ua.pt>

ficheiros da Universidade de Aveiro. Esta área também é acessível a partir do ambiente Windows e através da Web. É neste diretório que deve gravar os ficheiros e diretórios que criar no decurso das aulas práticas. Os computadores das salas de aulas foram programados para apagarem o diretório de utilizador (e.g. `/homermt/a1245/`) sempre que são reiniciados. Só o conteúdo do subdiretório **arca** é salvaguardado. É portanto aí que deve colocar todo o seu trabalho.

Exercício 1.14

Experimente mudar o diretório atual para o seu subdiretório **arca**. Liste o seu conteúdo. Reconhece algum dos ficheiros?

Exercício 1.15

Crie, no diretório **arca**, um subdiretório chamado **labi** e, dentro desse, um diretório chamado **aula01**. Guarde neste diretório este guião.

Manipulação de ficheiros

O Linux dispõe de diversos comandos de manipulação de ficheiros. Eis alguns:

cat fic — imprime no dispositivo de saída *standard* (por omissão o ecrã) o conteúdo do ficheiro **fic**. O nome deste comando é uma abreviatura de *concatenate*, porque permite concatenar o conteúdo de vários ficheiros num só.

rm fic — remove (apaga) o ficheiro **fic**.

mv fic1 fic2 — muda o nome do ficheiro **fic1** para **fic2**.

mv fic dir — move o ficheiro **fic** para dentro do diretório **dir**.

cp fic1 fic2 — cria uma cópia do ficheiro **fic1** chamada **fic2**.

cp fic dir — cria uma cópia do ficheiro **fic** dentro do diretório **dir**.

head fic — mostra as primeiras linhas do ficheiro (de texto) **fic**.

tail fic — mostra as últimas linhas do ficheiro (de texto) **fic**.

more fic — imprime no dispositivo de saída padrão (por omissão o ecrã), página a página, o conteúdo do ficheiro **fic**.

less fic — comando similar ao **more**, mas que permite uma navegação mais sofisticada para trás e para diante nas linhas apresentadas.

grep padrão fic — seleciona as linhas do ficheiro (de texto) **fic** que satisfazem o critério de seleção **padrão**.

wc fic — conta o número de linhas, palavras e caracteres do ficheiro **fic**. O nome deste comando é um acrónimo de *word count*

sort fic — ordena as linhas do ficheiro **fic** de acordo com um critério (alfanumérica crescente, por omissão).

find dir -name fic — procura um ficheiro com o nome **fic** a partir do diretório **dir**.

Todos estes comandos podem ser invocados usando argumentos opcionais que configuram o seu modo de funcionamento.

Exercício 1.16

Utilizando a linha de comandos, crie um directório chamado **aula01** no seu Ambiente de Trabalho e copie o ficheiro **/etc/passwd** para este directório. Imprima o seu conteúdo no ecrã.

Experimente outros comandos da lista acima.

1.5 Edição de ficheiros de texto

Nos computadores do laboratório temos instaladas as aplicações **vim** e **gedit** para manipulação de ficheiros de texto na consola e num ambiente gráfico, respectivamente.

Um ficheiro de texto é um ficheiro constituído por octetos (bytes) que representam caracteres alfanuméricos, sinais de pontuação, espaços em branco e caracteres invisíveis de mudança de linha. Um dos editores de ficheiros de texto mais usados em sistemas UNIX é o **vim**, uma versão melhorada do **vi**. Apesar da sua antiguidade, continua a ser preferido por muitos utilizadores, particularmente programadores, pela sua eficiência, pelas suas funções avançadas e por correr numa consola de texto. Atualmente também existem versões gráficas, como o **gvim** e o **vim-qt**⁹ que é desenvolvido por membros da Universidade de Aveiro!

O **vim** é um editor com um modo de funcionamento único, porque permite usar o teclado completo como fonte de caracteres (para o texto que se pretende introduzir) e como fonte de comandos. Assim, se durante uma edição de um texto se carregar na tecla “a”, o resultado pode ser a introdução do carácter “a” no texto, no local onde se encontra o cursor, ou a execução do comando associado à tecla *a*, dependendo do modo de trabalho actual no editor vim.

⁹<https://bitbucket.org>equalsraf/vim-qt/wiki/Home>

O **vim** trabalha em dois modos, ditos de *comando* e de *inserção*. No modo de comando, todas as teclas representam comandos; no modo de inserção, todas as teclas representam algo que se pretende inserir no texto. A mudança entre estes dois modos faz-se com as seguintes teclas:

modo de comando → modo de inserção : através de uma de entre várias teclas que indicam a posição onde se pretende introduzir texto. Algumas das mais usadas:

- i** — inserir no local do cursor.
- o** — inserir uma linha abaixo da atual.
- O** — inserir uma linha acima da atual.

modo de inserção → modo de comando : através da tecla ESC (*escape*).

Para além destes dois modos, o **vi** possui ainda dois outros modos: editor (**ed**) e visual. O modo **ed** é acionado quando no modo de comando se carrega na tecla “:”. Este modo serve para executar comandos relacionados com a salvaguarda do conteúdo do ficheiro e com o abandono da edição:

- :q** — terminar a edição atual sem salvaguardar o conteúdo do ficheiro.
- :q!** — terminar a edição atual sem salvaguardar o conteúdo do ficheiro, ignorando avisos caso tenha sido alterado.
- :x** — terminar a edição atual com a salvaguarda do conteúdo do ficheiro.
- :w** — salvaguardar o conteúdo do ficheiro.
- :w fic** — salvaguardar o conteúdo editado no ficheiro **fic**.
- :e fic** — abre o ficheiro **fic** para edição.
- :r fic** — insere o conteúdo do ficheiro **fic** abaixo do cursor.

O modo visual é o modo por omissão, ao qual o **vim** volta após se ter executado um comando no modo **ed**. Na versão gráfica do **vim** (**gvim**) pode fazer as operações do modo **ed** recorrendo aos menus da sua interface gráfica.

Caso não se sinte confortável com o **vim** pode usar outros editores mais convencionais, como o **gedit**, **kate**, **leafpad** ou outros. Porém, o **vim** tem a vantagem de se poder usar quase da mesma forma em ambientes gráficos e em consolas.

Todos estes editores possuem a função de realce de sintaxe, muito útil para programadores. Ou seja, sempre que editar um texto numa linguagem de programação ou de representação

conhecida, o editor usa cores diferentes para realçar blocos sintáticos distintos, de modo a facilitar a sua leitura e análise.

Para programação, também existem ambientes integrados de desenvolvimento (IDE) que além de um editor também incluem ferramentas de compilação, execução e depuração. Um IDE simples e versátil é o **geany**.

Exercício 1.17

Edite um ficheiro e escreva os aspetos do sistema *Linux* que mais o surpreenderam, tanto positivamente como negativamente. Experimente fazê-lo com o **vim**, e exerçite as diferentes formas de entrada em modo de inserção, bem como alguns comandos do modo **ed**.

Procura de texto

O modo de procura de texto do **vim** é o mesmo que se usa em vários outros comandos, como o **man**, o **more**, o **less**, etc.

A pesquisa de um bloco de texto num ficheiro é feita em modo comando com a tecla **/** (pesquisa para diante do cursor) ou **?** (pesquisa para trás do cursor).¹⁰ Após a escrita de um destes caracteres deve-se escrever o texto a procurar e terminar com a tecla **Enter** (ou **Return**), após o que a aplicação desloca o cursor até ao texto encontrado. Para encontrar a ocorrência seguinte, usa-se o comando **n**. Para inverter o sentido e encontrar a ocorrência anterior, usa-se o comando **N**.

Exercício 1.18

Edite o ficheiro anterior com o **vim** e realize pesquisas de texto no mesmo. Note que para o fazer terá de estar em modo de comando.

Exercício 1.19

Apresente o conteúdo do ficheiro anterior com o comando **less** e realize pesquisas de texto no mesmo.

¹⁰Num teclado americano estes símbolos estão na mesma tecla física e num local muito conveniente.

Exercício 1.20

Execute o comando **man less** e realize pesquisas de texto do manual, tanto para diante como para trás.

Exercício 1.21

Leia a página de manual do comando **less** para descobrir como se consegue saltar para o início e para o fim do texto e experimente esses comandos. Tenha em consideração que a deslocação para um ponto do texto é referida através da expressão “Go to” no manual.

1.6 Processos

Em *Linux* existe uma máxima que diz que tudo é um ficheiro ou um processo. Até agora vimos como funcionam os ficheiros, mas ignorámos os processos.

Os processos não são nada mais do que os programas em execução na máquina. Todo o ambiente gráfico ou de texto que lhe é apresentado, assim como a *Shell* e muitos outros programas estão a executar simultaneamente. Um programa executado várias vezes dá origem a vários processos. No sistema, cada processo é identificado por um número chamado de Process IDentifier (PID).

Pode utilizar o comando **ps** para verificar os processos da sua sessão, ou combinar com as opções **-ax** para ver que processos estão activos, independentemente do utilizador. Se adicionar a opção **-u** irá igualmente ver detalhes dos processos como por exemplo a memória utilizada. O comando **ps Tu** irá mostrar os processos associados com o terminal e os seus detalhes. Como pode ver (Figura 1.7), é mostrado o utilizador, o PID, a quantidade de processador e memória utilizados, o seu estado, há quanto tempo foram iniciados e qual o seu nome.

```
linux@linux:~$ ps Tu
USER      PID %CPU %MEM      VSZ   RSS TTY      STAT START  TIME COMMAND
linux     1784  0.0  0.5    6372  2736 pts/0      Ss  07:08  0:00 /bin/bash
linux     4638  0.0  0.2    5276  1220 pts/0      R+  17:17  0:00 ps Tu
linux@linux:~$ █
```

Figura 1.7: Resultado da execução do comando **ps Tu** para mostrar os processos associados com a sessão actual.

Exercício 1.22

Compare o resultado de **ps**, **ps -a**, **ps -ax** e **ps -aux**.

Qual o processo que consome mais memória? Qual o processo que consome mais CPU? Quantos utilizadores têm processos activos?

Exercício 1.23

Repete o exercício anterior utilizando o comando **top**. Pode utilizar as teclas < e > para seleccionar qual a coluna escolhida para ordenar os processos. Use q para terminar.

Um aspecto importante dos processos é a sua gestão, nomeadamente a possibilidade de controlar o seu estado de execução e, eventualmente terminá-la de forma forçada.

Os comandos **kill** e **killall** permitem enviar sinais aos programas. A diferença entre estes comandos reside no facto de o primeiro (**kill**) enviar um sinal a um processo específico identificado pelos seu identificador:

```
kill [-sinal] identificador-do-processo
```

Enquanto o segundo envia um sinal a todos os processos com um determinado nome:

```
killall [-sinal] nome-do-processo
```

Se não se indicar o sinal, é enviado o sinal **SIGTERM**, que geralmente provoca a terminação do processo de forma relativamente segura. No entanto, em certas condições, um processo pode ficar num estado em que não reage ao sinal. Nesse caso, pode usar-se o sinal **SIGKILL** ou **9**, que não pode ser ignorado, mas corre-se maior risco de perder dados que não tenham sido gravados. Existem sinais com outras funcionalidades. Pode consultar uma lista com o comando **man 7 signal**.

Exercício 1.24

Inicie dois terminais. No primeiro inicie um processo **top**.

No segundo terminal, recorrendo aos comandos **ps** e **kill** ou **killall**, termine a execução do processo **top**.

1.7 Administração básica

(Pderá saltar esta secção numa primeira leitura.)

Com certeza já reparou que muitos diretórios e ficheiros do sistema pertencem a um utilizador chamado **root**, e só ele tem permissão de os modificar. Isto impede que um erro de outro utilizador possa destruir o sistema, mas também implica que só o **root**, também chamado de *super-utilizador* ou *administrador* do sistema, consegue fazer certas tarefas como instalar software nos diretórios do sistema ou registar novos utilizadores, por exemplo.

Nos sistemas modernos é usual permitir que um ou mais utilizadores do sistema possam usar o comando **sudo** para assumir temporariamente o papel de super-utilizador e assim realizar tarefas de administração. O primeiro utilizador criado numa instalação de *Linux* tem geralmente essa permissão.

Como é óbvio, nos computadores da sala de aula, um utilizador normal não tem essa permissão, pelo que não poderá fazer estas tarefas de administração. Por esta razão, os exercícios desta secção terão de ser feitos numa máquina diferente, onde tenha poderes de administrador.

Uma hipótese é usar uma *máquina virtual*: um computador simulado dentro de outro. Estudaremos máquinas virtuais noutra aula, mas por agora pode usar uma máquina virtual que preparamos para usar no computador da sala. Para isso:

1. extraia o conteúdo do arquivo **/usr/local/labi/labib-vb-setup.tbz2** para a sua pasta pessoal;
2. extraia o conteúdo do arquivo **/usr/local/labi/labib-lubuntu.vdi.tbz2** para a sua pasta pessoal (isto poderá demorar um bocado);
3. execute o programa VirtualBox;
4. na janela do VirtualBox, escolha Start.

Deve aparecer uma janela onde arranca o seu computador virtual. Faça login com o utilizador **labi** e password **labi**. Este utilizador tem permissões de administração e permite-lhe avançar para os exercícios das sub-secções seguintes.

Gestão de utilizadores

O sistema *Linux* assume uma gestão baseada em utilizadores e grupos. A utilização deste sistema permite que múltiplos utilizadores façam uso do sistema apenas após autenticação, respeitando a privacidade de cada utilizador, e evitando que um dado utilizador danifique os dados de um outro utilizador.

Este modelo foi generalizado de forma que além de utilizadores reais, como é o caso do utilizador **labi**, também há *utilizadores de sistema* que servem apenas para confinar aplicações a definições de segurança específicas. Por exemplo, um processo que serve páginas Web pertence a um certo utilizador de sistema para ter acesso aos ficheiros das páginas que serve, mas não aos ficheiros de qualquer outro utilizador.

Pode verificar qual o seu utilizador através do comando **whoami**, ou em mais detalhe através do comando **id**.¹¹ O resultado deverá ser o demonstrado na Figura 1.8. Experimente o mesmo comando no computador da sala (fora da máquina virtual).

```
linux@linux:~$ whoami  
linux
```

Figura 1.8: Resultado da execução do comando **whoami**

Exercício 1.25

Utilizando os comandos **mkdir** e **rmdir** verifique se possui permissões para escrever nos seguintes locais: **/home/labi**, **/etc**, **/tmp**, **/bin**.

Na máquina virtual, o utilizador **labi** consegue assumir temporariamente as funções de super-utilizador (**root**) através do comando **sudo**.

Na primeira vez que um utilizador usa o comando **sudo**, é pedida a sua palavra passe para confirmação. Execuções seguintes na mesma sessão “lebram-se” da validação durante alguns minutos e não a pedem de novo.

A Figura 1.9 demonstra a utilização do comando **sudo** nas suas duas formas mais comuns. Na primeira forma, (**sudo whoami**), o comando **whoami** é executado como super-utilizador,

¹¹O comando **id** irá mostrar o utilizador e todos os grupos aos quais ele pertence

mas o comando **whoami** seguinte é novamente executado pelo utilizador **labi**. A segunda forma, **sudo -s**, cria uma nova *Shell* no contexto do super-utilizador (repare no prompt). Todos os comandos inseridos serão executados com os privilégios de **root**, até que se termine esta shell com **exit** ou **Ctrl-D**.

```
linux@linux:~$ whoami  
linux  
linux@linux:~$ sudo whoami  
root  
linux@linux:~$ whoami  
linux  
linux@linux:~$ sudo -s  
root@linux:~# whoami  
root  
root@linux:~# exit  
exit  
linux@linux:~$ whoami  
linux  
linux@linux:~$ █
```

Figura 1.9: Resultado da execução do comando **sudo**, inspeccionando qual o utilizador em uso.

Exercício 1.26

Verifique se possui permissões para visualizar (**cat**) o ficheiro **/etc/shadow**. Se não possuir, visualize o seu conteúdo recorrendo ao comando **sudo**.

Consegue perceber para que serve este ficheiro? Pode recorrer ao comando **man**.

Na verdade, os utilizadores não são tratados internamente pelo sistema operativo com os nomes que temos utilizado (ex., **labi**). Na realidade tanto os utilizadores como os grupos existentes são mapeados para números. São esses números que se vê no resultado do comando **id**, tal como representado na Figura 1.10.

```
linux@linux:~$ id  
uid=1000(linux) gid=1000(linux) grupos=1000(linux),4(adm),24(cdrom),27(sudo),30(dip),4  
6(plugdev),108(lpadmin),124(sambashare)
```

Figura 1.10: Resultado da execução do comando **id**, inspeccionando qual o utilizador em uso (neste exemplo o utilizador tem o nome **linux**).

São de relevância dois valores apresentados:

uid — User Identifier, ou seja, o número que identifica o utilizador.

gid — Group Identifier, ou seja, o número do grupo principal do utilizador.

Estes valores são utilizados pelo sistema para controlar as permissões. Os nomes são utilizados para facilitar a gestão aos administradores (humanos).

Exercício 1.27

Verifique qual o **uid** e **gid** do utilizador **root**.

Exercício 1.28

Analise o resultado do comando **ls -la** sobre várias localizações, como por exemplo: **/etc**, **/tmp** e verifique a que utilizadores pertencem os ficheiros e quais as permissões.

Gestão de Aplicações

Uma característica importante das distribuições de *Linux* é o facto de utilizarem um sistema integrado para a pesquisa, instalação, actualização e remoção de aplicações. É um conceito semelhante (e anterior) ao das lojas de aplicações tipicamente disponíveis nos telemóveis actuais. As grandes diferenças é que as aplicações disponibilizadas por estes meios são geralmente de utilização livre e gratuita, e existe um conjunto de ferramentas para a sua gestão.

Importante! Todas as distribuições usam o seu sistema de gestão de pacotes, sendo que algumas partilham as ferramentas utilizadas. No entanto, não existe um método universal, comum a todas as distribuições, para a gestão de pacotes. Os comandos necessários para esta gestão podem variar dependendo da distribuição utilizada.

No caso das distribuições que derivam da distribuição *Debian*¹², a gestão de aplicações é realizada através da família de comandos **apt-**, ou através de aplicações mais intuitivas como o **synaptic**, o **aptitude**, ou mesmo o **Ubuntu Software Center**.

Através da linha de comandos, e recorrendo aos comandos **apt-***, é possível realizar uma gestão completa. De realçar que a gestão de aplicações é uma operação privilegiada. Portanto só disponível a utilizadores com permissões para o efeito, normalmente através de **sudo**.

Os principais comandos a utilizar são:

- **apt-get** — Permite actualizar, instalar e remover aplicações.

¹²Para ver como se organizam as distribuições, consultar <http://futurist.se/gldt/>.

- **apt-cache** — Permite procurar por aplicações.

Para a utilização deste sistema é necessário em primeiro lugar actualizar a lista de aplicações. Isto irá transferir a lista de aplicações nos servidores da distribuição para o computador local. Depois torna-se possível pesquisar sobre essa lista e assim seleccionar aplicações a instalar. Este processo é importante pois permite obter actualizações para as aplicações instaladas. Este processo de actualização deve ser repetido periodicamente e geralmente os sistemas já vêm configurados para o fazerem de forma automática.

Exercício 1.29

Utilizando o comando **apt-get update** actualize a base de dados de aplicações.
(Lembre-se que precisa de correr o comando dentro de um **sudo**.)
Utilizando o comando **apt-get upgrade** actualize as aplicações do sistema.

Os comandos mais usuais para gerir a instalação de uma nova aplicação, por exemplo a aplicação **cowsay**, são os seguintes (ver Figura 1.11):

1. **apt-cache search cowsay** — Procura pelo nome e indica pacotes relacionados.
2. **apt-get install cowsay** — Instala o pacote **cowsay**, que tem a aplicação.
3. **apt-get remove cowsay** — Para remover a aplicação, se já não a quisermos.

Exercício 1.30

Instale e verifique o funcionamento das aplicações **cowsay** e **fortune**.

1.8 Para aprofundar o tema

Exercício 1.31

Explore os restantes ficheiros e directórios nos directórios **/proc** e **/sys**.

Recorra ao comando **man** para obter informação sobre cada uma das entradas.

```

root@linux:~# apt-cache search cowsay
cowsay - configurable talking cow
xowsay - vaca falante gráfica e configurável
root@linux:~# apt-get install cowsay
A ler as listas de pacotes... Pronto
A construir árvore de dependências
A ler a informação de estado... Pronto
Pacotes sugeridos:
  filters
Serão instalados os seguintes NOVOS pacotes:
  cowsay
0 pacotes actualizados, 1 pacotes novos instalados, 0 a remover e 3 não actualizados.
É necessário obter 0 B/20,4 kB de arquivos.
Após esta operação, serão utilizados 90,1 kB adicionais de espaço em disco.
A seleccionar pacote anteriormente não seleccionado cowsay.
(A ler a base de dados ... 61522 ficheiros e directórios actualmente instalados.)
A descompactar cowsay (desde .../cowsay_3.03+dfsg1-4_all.deb) ...
A processar 'triggers' para man-db ...
A instalar cowsay (3.03+dfsg1-4) ...
root@linux:~# /usr/games/cowsay Labi
< Labi >
-----
 \ ^__^
  (oo)\_____
   (__)\       )\/\
    ||----w |
     ||     |

root@linux:~# █

```

Figura 1.11: Processo de instalação da aplicação **cowsay**

Exercício 1.32

Explore a utilização dos operadores “|” e “>”.
Por exemplo:

ls -la > ls.txt

ou

ls -la |wc

Máquinas Virtuais

Objetivos:

- Conceito de virtualização de hardware.
- Criação e configuração de máquinas virtuais.
- Instalação de um sistema operativo Linux.
- Clonagem de máquinas virtuais.

2.1 Introdução

As máquinas virtuais são hoje em dia um instrumento que permite usar a mesma máquina física para executar vários sistemas operativos, como se de várias máquinas se tratasse. É uma ferramenta muito importante para os sistemas atuais, sendo também relevante para o restante curso. A sua utilização torna possível criar ambientes virtuais isolados para trabalhos específicos ou disciplinas específicas sem modificar o sistema operativo que se utiliza no dia a dia.

Por exemplo, permite ter o ambiente *Linux* utilizado nas aulas de programação, sem modificar o sistema *Windows* que tipicamente vem pré-instalado no portátil. Nestas aulas em concreto, vai permitir que os alunos tenham uma máquina que podem controlar totalmente (instalar aplicações, configurar, etc.), algo que não é permitido nos computadores de utilização geral da universidade.

O objetivo deste tema é mostrar como se pode criar uma máquina virtual, como configurar o seu *hardware* e como instalar nela um sistema operativo.

2.2 Conceitos e terminologia

Uma *máquina virtual* é um computador simulado por *software* que corre num computador real. O *software* que permite simular máquinas virtuais é designado por *virtualizador* ou *software de virtualização* (*virtualization software*, em inglês). Existem vários virtualizadores, mas neste trabalho usaremos o virtualizador gratuito *VirtualBox*,¹³ distribuído pela Oracle e disponível para os principais sistemas operativos.

Usamos o termo *anfitrião* ou *hospedeiro* (*host*) quando nos referimos ao computador real que executa o virtualizador, ao seu hardware ou ao seu sistema operativo.

Usamos o termo *convidado* ou *hóspede* (*guest*) quando nos referimos ao computador virtual, ao seu hardware (virtual) ou ao seu sistema operativo.

2.3 Instalação de uma máquina virtual

Instalação do virtualizador

Antes de instalar uma máquina virtual temos de ter um virtualizador. Os computadores dos laboratórios já dispõem da versão 4.2 do *VirtualBox* instalado no ambiente Linux. Para usar o seu próprio computador terá de instalar este ou outro virtualizador, na versão adequada ao seu sistema operativo. Este guião apenas contemplará explicações para *VirtualBox*.

Instalador do sistema operativo

Além do virtualizador, também precisamos de ter o software de instalação do sistema operativo que pretendemos correr na máquina virtual. Poderá ser um CD de instalação de uma distribuição *Linux*, como referido no guião anterior. Na verdade, basta-nos ter a imagem do CD (o ficheiro ISO).

Neste guião, sugerimos que utilize o ficheiro Imagem de Arquivo de CD (ISO) com a imagem de um CD de arranque da distribuição *live* do *Slitaz*, que está disponível na página da disciplina¹⁴. Deverá descarregar e guardar o ficheiro antes de prosseguir para a instalação da máquina virtual.

A distribuição *Slitaz* é interessante para este trabalho porque:

1. O seu *live CD* tem uma dimensão reduzida (cerca de 45 Mebibytes (MiBs)).
2. O sistema base instalado num disco rígido não tem mais de 315 MiBs.

¹³<https://www.virtualbox.org/>

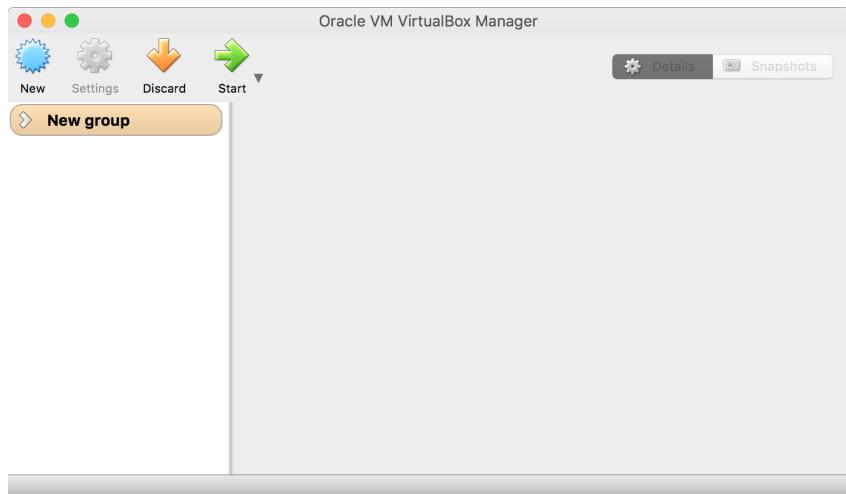
¹⁴<https://elearning.ua.pt/course/view.php?id=3470>

Estas dimensões permitem diminuir os custos de descarga da imagem do CD, de arranque do sistema *live* e de instalação do sistema numa máquina virtual.

No entanto, muito embora seja uma distribuição perfeitamente funcional do *Linux*, a *Slitaz* é pouco conhecida e não pertence a nenhuma das grandes famílias de distribuições *Linux* (Debian/Ubuntu, Red Hat, Slackware, Gentoo, Arch). Por isso, só a sugerimos para utilizações especializadas e esporádicas. Para uma utilização mais geral e frequente recomenda-se uma das distribuições no topo do ranking mantido pelo site <http://distrowatch.com>. Na página desta disciplina é fornecida uma imagem da distribuição *Debian* na sua versão 9.

Criação de uma máquina virtual

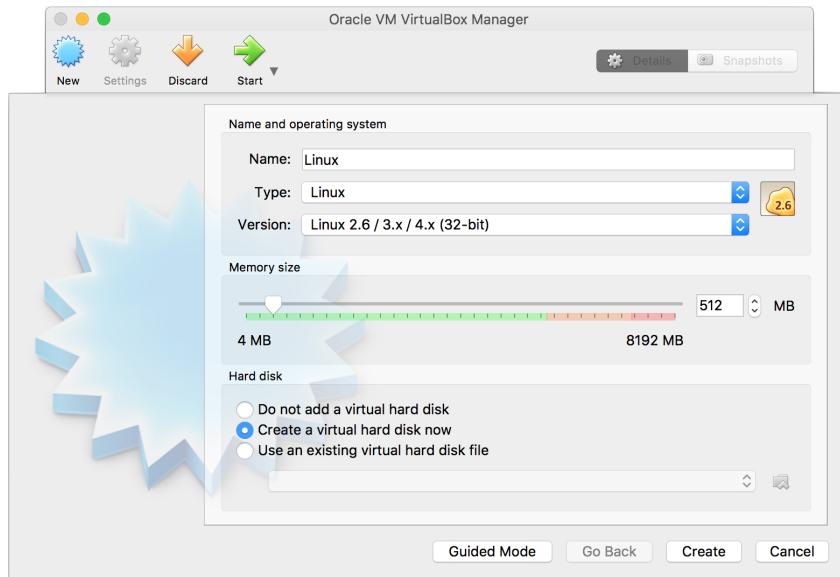
Os passos para criar uma máquina virtual vão ser seguidamente indicados, acompanhados de capturas de ecrãs exemplificativas. (Os ecrãs podem variar consoante a versão instalada.)



Inicie a execução do *VirtualBox*. Deverá surgir no ecrã uma janela como a indicada acima.

(Se desejar, antes de instalar qualquer máquina virtual, poderá alterar as definições do *VirtualBox*, por exemplo o diretório onde guarda os dados relativos às máquinas virtuais. Para tal, selecione **Ficheiro**, **Preferências**, **Geral**, **Pasta pré-definida das Máquinas**.)

Seleccione o botão **Novo** para indicar que deseja criar uma nova máquina virtual.



Escolha um nome para identificar a nova máquina virtual.

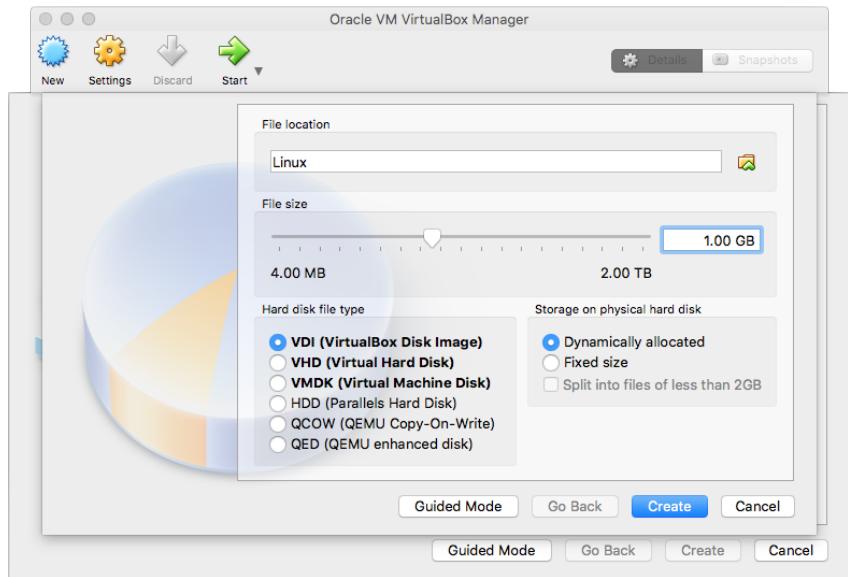
Quanto ao tipo de sistema operativo que pretende instalar, escolha *Linux*, versão 2.6/3.x/4.x 32 bits (versão do núcleo, ou *kernel*).

Esta informação ajuda o *VirtualBox* a configurar certos parâmetros da máquina virtual que poderão ser requeridos pelo sistema operativo convidado.

Indique a quantidade de memória Random Access Memory (RAM) de que disporá a máquina virtual. Não escolha mais do que 512 MiB, porque não será necessário e porque quanto mais escolher, menos memória sobrará para o sistema hospedeiro. Esta configuração poderá ser alterada mais tarde, não é irreversível.

Indique igualmente que pretende criar um disco rígido (virtual) na sua nova máquina.

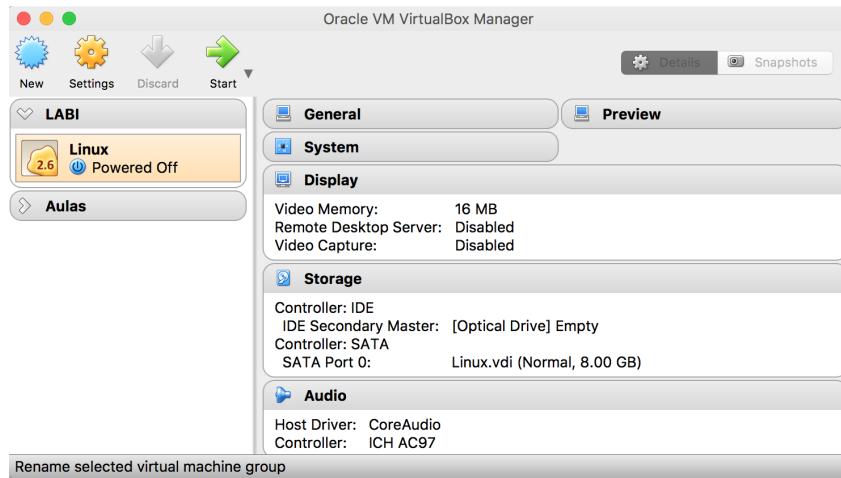
Mais tarde, instalaremos o sistema operativo neste disco.



Escolha um disco virtual do tipo VirtualBox Disk Image (VDI). Indique que pretende o disco virtual com tamanho dinâmico, para evitar que o disco virtual (um ficheiro no hospedeiro) ocupe o seu tamanho máximo à partida. Desta forma, ele terá apenas o tamanho suficiente para guardar o sistema de ficheiros da máquina virtual, sem nunca ultrapassar o máximo indicado.

Indique a localização do disco virtual (como foi dito atrás, será um ficheiro do sistema de ficheiros do sistema hospedeiro) e a sua dimensão. Nesta instalação indique apenas 1 Gibibyte (GiB). É normal os virtualizadores permitirem mais tarde aumentar o tamanho dos discos virtuais, mas tal não será necessário neste trabalho.

Neste momento já foi recolhida toda a informação necessária e podemos avançar com a criação da máquina virtual e do seu disco virtual ainda vazio.



Este é o aspeto do gestor de máquinas virtuais após a criação de uma máquina virtual.

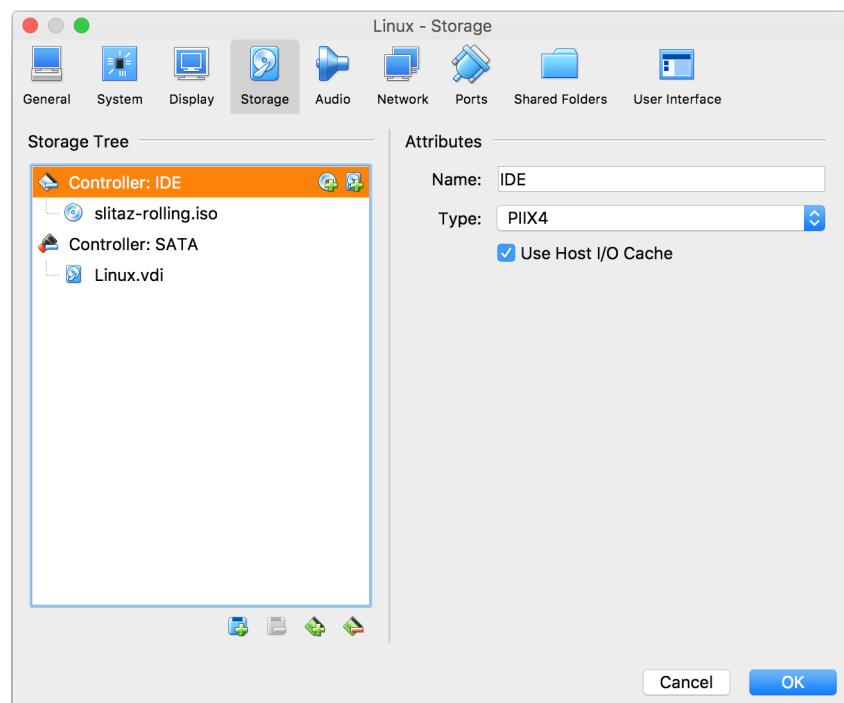
Na aba da esquerda é indicado o nome e tipo da máquina, bem como o seu estado (desligada).

A aba da direita mostra a configuração da máquina selecionada, i.e., o tipo e características do hardware virtual que inclui. Repare que tem dois dispositivos virtuais de armazenamento: um leitor de CD/DVD, que se encontra vazio; e o disco rígido que criámos anteriormente (ficheiro **Linux.vdi**).

2.4 Instalação do sistema operativo na máquina virtual

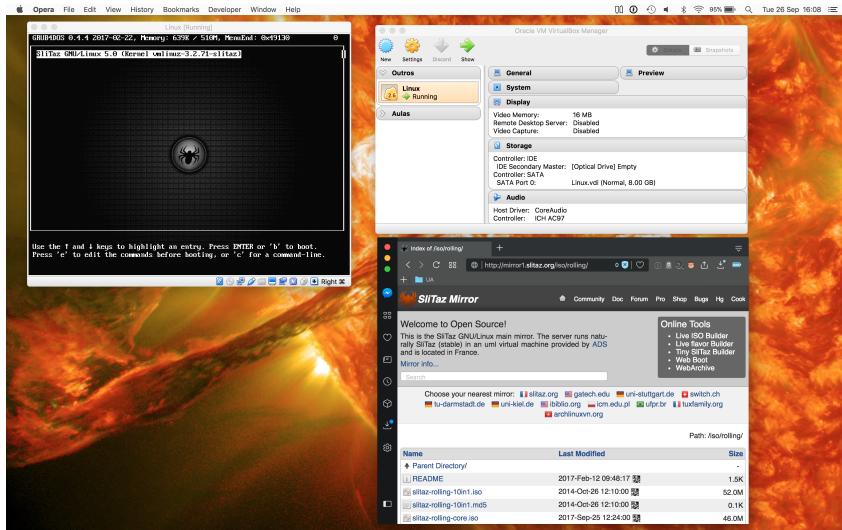
Neste momento temos uma máquina virtual criada, mas não tem qualquer software instalado. Precisamos de instalar um sistema operativo. Vamos fazê-lo recorrendo ao live CD que descarregámos anteriormente. Isto envolve: introduzir o CD no leitor, arrancar o sistema do live CD, correr o programa para instalar o sistema no disco, e reiniciar o novo sistema a partir do disco. Estas quatro operações são descritas nas secções seguintes.

Inserir um CD na máquina virtual



Agora precisamos de “introduzir” o CD no leitor, para podermos arrancar a máquina virtual, já que o disco rígido continua vazio. Para isso, selecione o botão **Definições**, seguido de **Armazenamento**, para gerir os dispositivos de armazenamento de dados.

Selecione o leitor de CD indicado como **Vazio**. À direita surgirão informações (atributos) sobre o dispositivo (nomeadamente, que estará associado a um controlador de IDE secundário). Selecione o ícone com um disco que surge à direita e escolha a opção **Escolher um ficheiro de CD/DVD virtual....** Aqui deverá selecionar o ficheiro **slitaz-rolling.iso** que descarregou na secção 2.3.



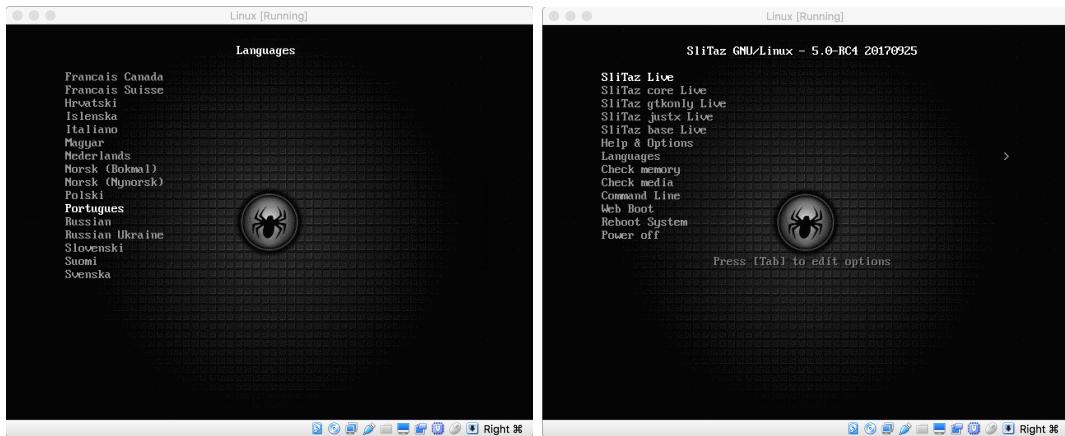
Neste momento, a configuração de instalação da máquina virtual está completa, o seu hardware está definido e no leitor de CD está acessível um CD virtual (a sua imagem ISO) que permitirá o arranque da distribuição *live* do *Slitaz*. Podemos terminar a alteração das definições e, regressando à janela principal, selecionar o botão **Iniciar** para “ligar” a máquina virtual. O resultado deverá o demonstrado na figura anterior, onde se observa o desktop, e várias aplicações. A máquina virtual, consiste apresenta-se como um computador completo para o sistema operativo do convidado, mas na realidade é uma aplicação no sistema do hospedeiro.

Arranque do live CD na máquina virtual

O arranque de uma máquina virtual é em tudo semelhante ao de uma máquina real. A máquina começa a executar uma BIOS (virtual) e, nesta fase, temos a possibilidade de selecionar o dispositivo de arranque (*boot device*) de onde vai ser arrancado o sistema. Neste caso, o dispositivo de arranque vai ser o CD.

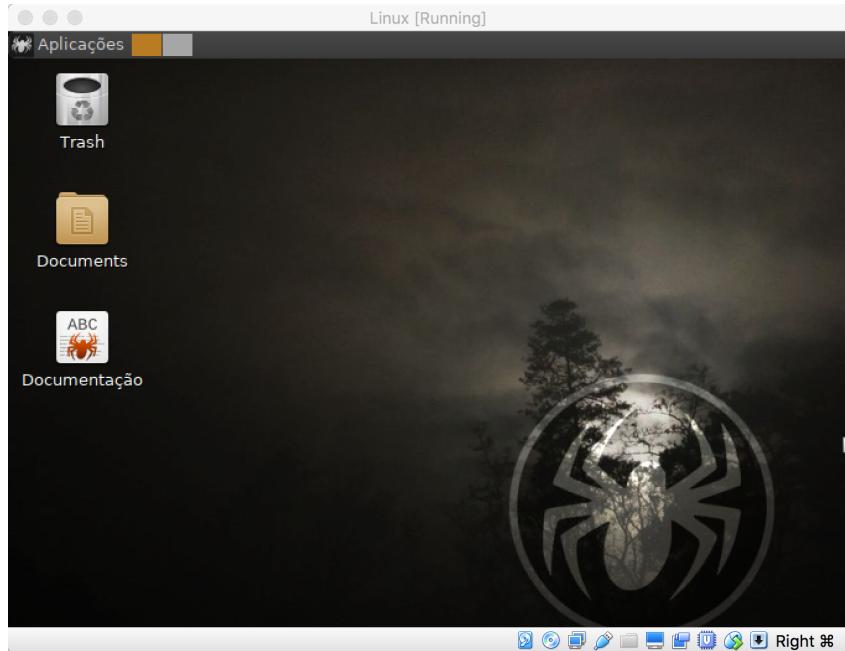
Os dispositivos de interface humana (teclado, rato) do sistema hospedeiro (*host*) vão ser partilhados com o sistema convidado (*guest*). Os dados desses dispositivos são encaminhados ora para o hospedeiro, ora para o convidado, em alternância, consoante a *focagem* escolhida. A focagem faz-se selecionando a janela da máquina virtual ou simplesmente deslocando o rato para cima dessa janela (quando a interface da máquina virtual possui um rato). Quando a focagem está ativa na máquina virtual, o ícone que possui uma seta para baixo, à direita na barra inferior da janela, apresenta a cor verde. Quando a focagem está inativa, o que se consegue fazer retirando o rato da janela ou carregando na tecla **Ctrl** da direita do teclado, a seta fica preta.

O arranque da distribuição *live* do *Slitaz* possui os seguintes passos:



Após o arranque é mostrado um menu com várias opções quanto à língua que deverá ser utilizada pelo sistema. Escolha a que mais lhe convier.

O passo seguinte consiste na escolha do sistema que efetivamente se quer executar a partir do CD. Vamos escolher a primeira opção (**Slitaz Live**). Caso nada seja feito, ao fim de alguns segundos esta opção é selecionada por omissão.



Feitas as duas configurações anteriores, o sistema progride sem qualquer intervenção humana até atingir a plena funcionalidade, altura em que apresenta o aspetto acime.

Repare que foi iniciada automaticamente uma sessão com um utilizador chamado **tux**.

Neste ponto pode explorar os menus da interface gráfica usando o apontador.

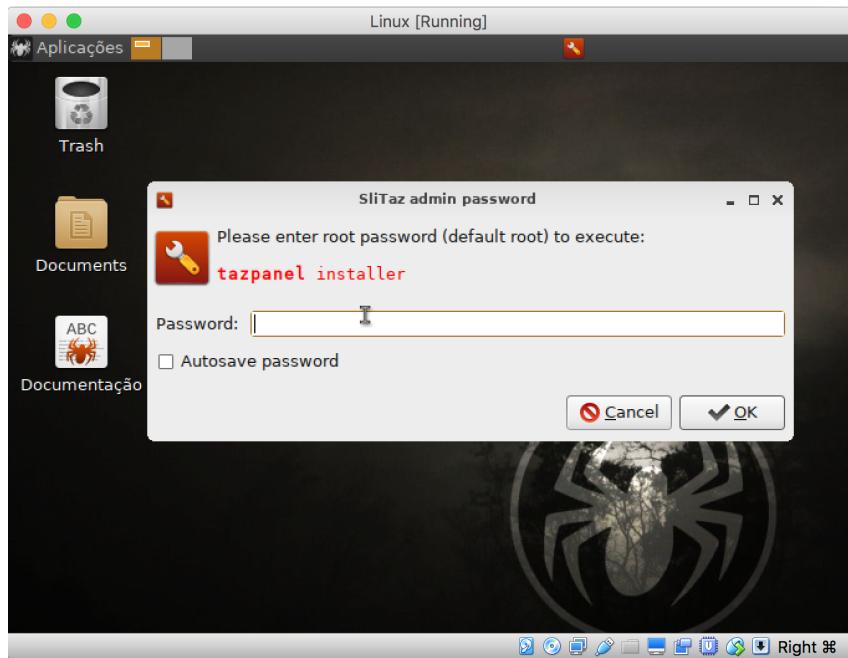
Instalação do sistema operativo no disco rígido virtual

Feito o arranque da distribuição *live*, vamos usar a mesma para criar uma instalação similar no disco rígido virtual, o que permitirá posteriormente arrancar o sistema desse dispositivo, dispensando o CD. A vantagem dessa opção é que posteriores alterações ao sistema irão ficar gravadas para utilização futura, enquanto que alterações ao sistema realizadas em execuções *live* perdem-se após desligar a máquina virtual.

Para fazer a instalação usa-se a aplicação **Instalação do Slitaz** do modo que seguidamente se descreve:

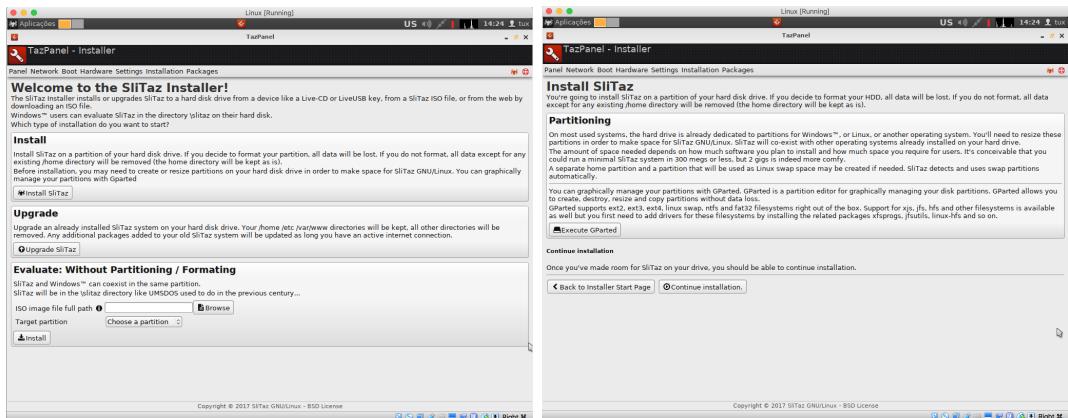


Selecione a aplicação **Instalador do Slitaz** ou **TazPanel** nos ícones que se encontram no canto superior esquerdo da interface (ícone da esquerda), Ferramentas do Sistema.



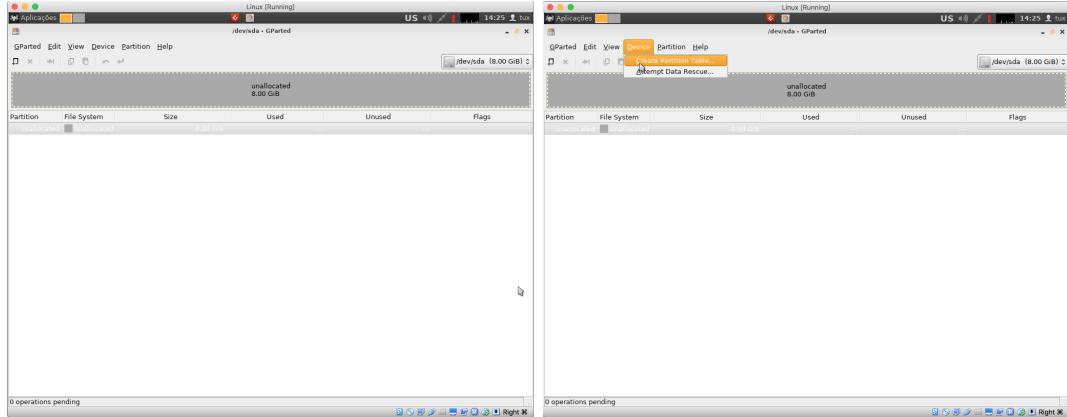
Esta operação está reservada para o administrador, que é designado em *Linux* por **root**. Use igualmente esse nome como senha no campo **password**.

(Ao contrário de outras distribuições, esta não usa o sistema **sudo** para promoção temporária de permissões.)



Neste passo o **TazPanel** mostra uma interface bastante explicativa, destinada a elucidar o utente sobre o que significa o particionamento de discos e que partições são usadas pelo *SliTaz*. Deslocando a janela para baixo aparece o botão **Execute Gparted**, que irá permitir criar as partições no disco virtual. Este passo é necessário pois um disco não

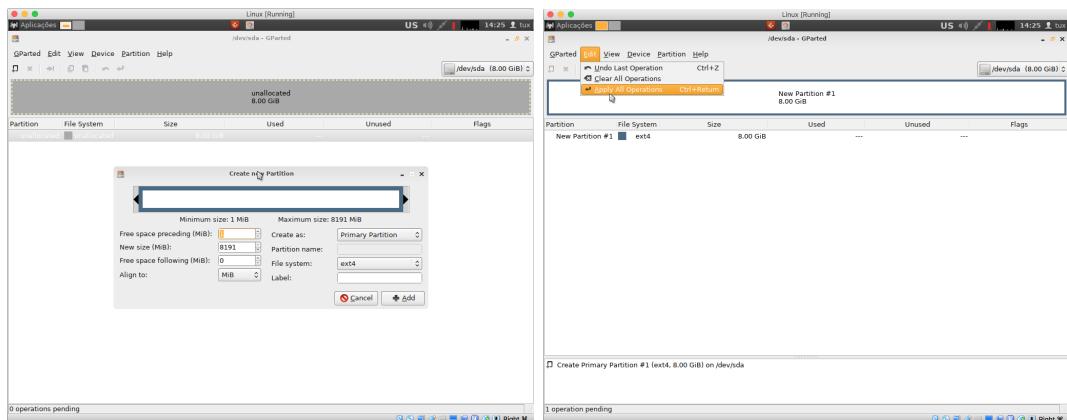
possui qualquer estrutura além de sectores. Uma partição permite criar uma zona que irá albergar um sistema de ficheiros.



A aplicação **GParted** mostra um disco não alocado (i.e., sem estar a uso). Para o usar é preciso primeiro criar uma tabela de partições no mesmo; só após isso se podem definir partições dentro desse disco. Esta tabela reside no início do disco e identifica que partições existem, onde iniciam, onde terminam e qual o sistema de ficheiros que é suposto lá existir.

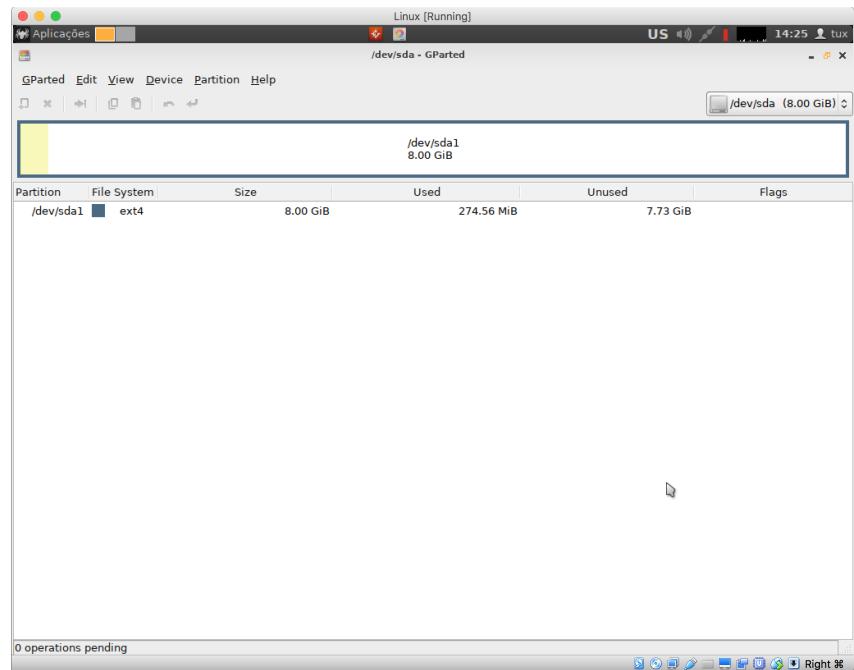
Escolha a opção **Device** da barra superior e a opção **Create Partition Table**.... A mensagem de aviso indica que irá apagar toda a informação presente no disco, mas tal não é um problema, porque o disco virtual está vazio. Também não existe qualquer problema para o anfitrião pois o disco é virtual e não real.

O passo seguinte consiste na criação de uma partição no disco virtual para aí instalar o sistema operativo. Para isso, selecione a área sombreada com o rato e selecione o ícone **New**.

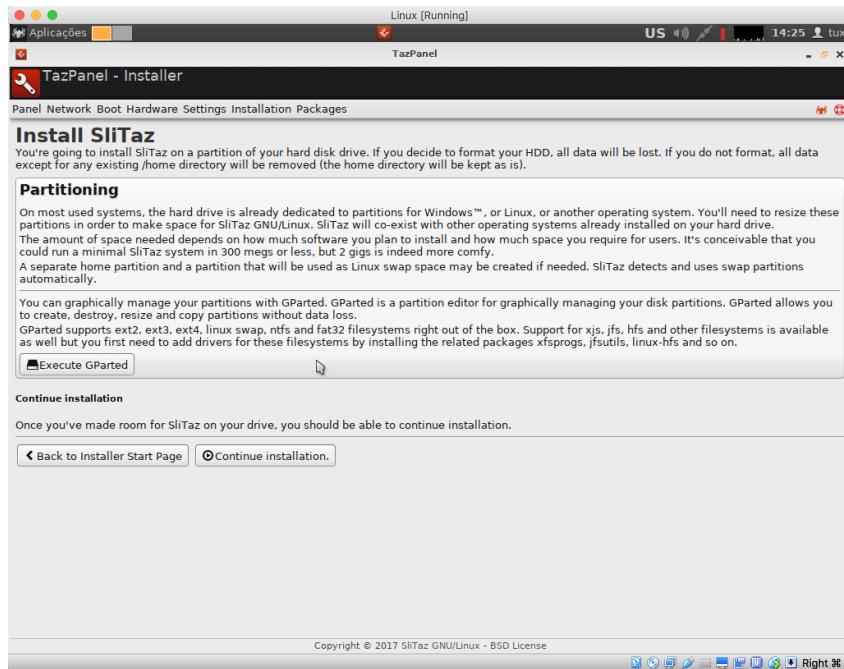


Na janela que surge, que indica a ocupação total do disco virtual por uma partição do tipo **ext4**, mantenha os valores indicados e selecione o botão **Add**.

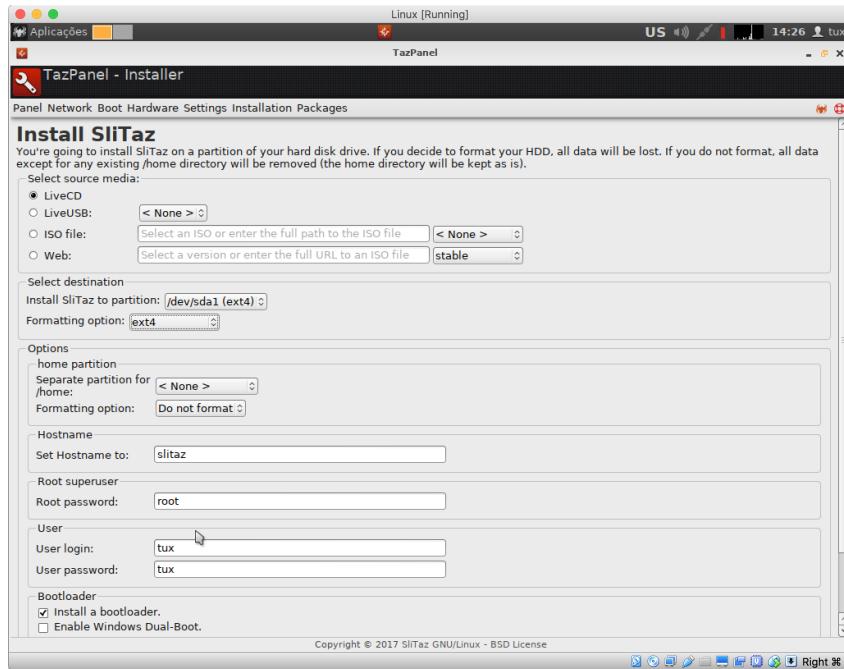
Após a definição da partição é preciso registá-la na tabela de partições. Para isto seleciona-se com o botão direito do rato a linha com a ordem de criação da partição que aparece na caixa inferior do **GParted** e escolhe-se a opção **Apply All Operations**. A mensagem de aviso avisa uma vez mais do facto de se perderem informações presentes no disco virtual, mas, como já se viu antes, o disco está vazio.



Uma vez criada a partição, o aspeto da aplicação **GParted** é o apresentado acima. Podemos terminar a aplicação selecionando o menu **GParted** da barra superior e escolhendo a opção **Quit**.



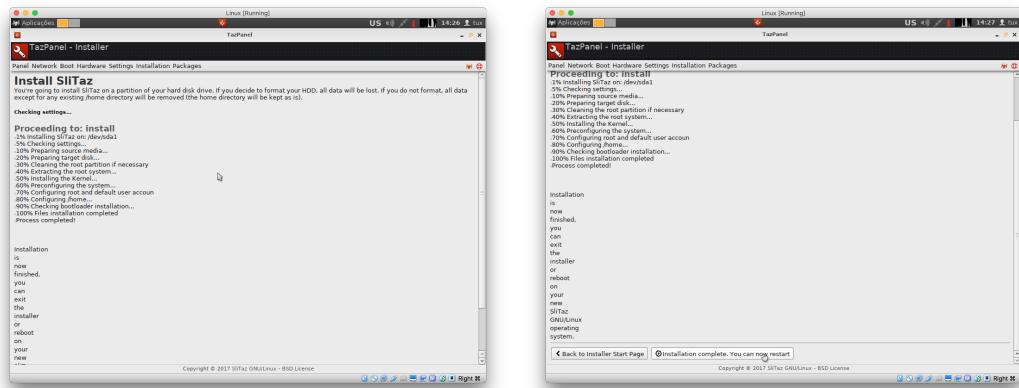
Vamos agora selecionar o botão **Continue installation** para instalar o sistema operativo na partição recém criada.



Neste passo a primeira ação fundamental a realizar consiste na indicação da partição onde o sistema operativo irá ser instalado, que irá ser a partição antes criada (`/dev/sda1`). Aqui **sda** representa o primeiro disco, e 1 representa a primeira partição.

O segundo passo fundamental a realizar consiste na especificação da senha e eventual alteração do nome do utilizador inicial da máquina (**tux**, por omissão). Não se esqueça de colocar exatamente a mesma senha na caixa de confirmação. Nas caixas acima também pode ser mudada a senha do utilizador **root**. Numa instalação a sério, é fortemente recomendado mudar estas senhas.

O último passo fundamental consiste na indicação da instalação do carregador de arranque **Grub**. Isto irá colocar código especial nos primeiros setores do disco. Este código é executado automaticamente pela **BIOS** do computador e permite depois iniciar o sistema operativo. O **Grub** é um sistema que realiza esta função, permitindo ainda escolher no início da máquina, qual o sistema operativo a utilizar.

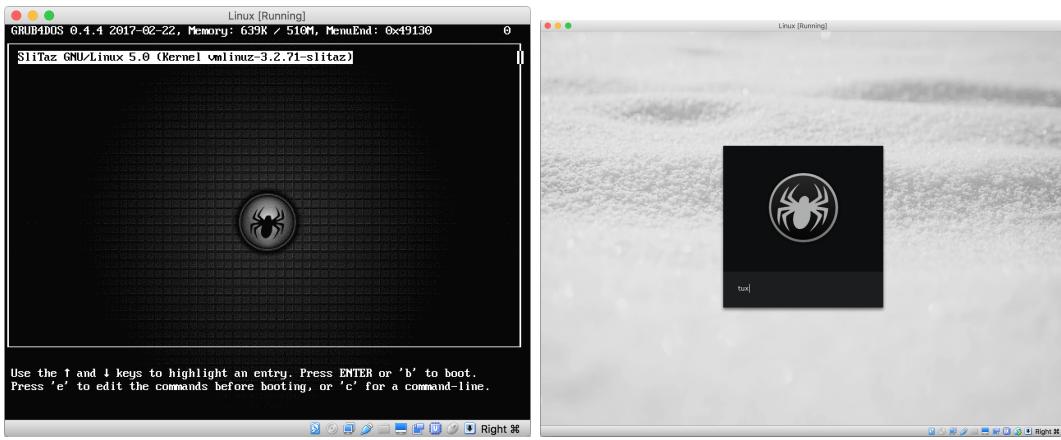


Uma vez feitas estas configurações pode-se prosseguir para a instalação do sistema operativo e do sistema de arranque **Grub** no disco rígido virtual.

Os passos realizados durante a instalação vão sendo mostrados à medida que ela ocorre, até ser indicada a terminação da instalação. O sistema está agora pronto para arrancar do disco rígido virtual.

Terminada a instalação, vamos selecionar o botão **Instalation complete. You can now restart (reboot)** para reiniciar o sistema. Entretanto o CD virtual foi “ejetado” da máquina virtual e podemos reiniciar o sistema.

Arranque do sistema instalado no disco virtual



Após o reinício do sistema, surge a interface do **Grub** apenas com uma opção: a do sistema que acabou de ser instalado. Se nada for feito ele arrancará por omissão ao fim de alguns segundos.

Durante este primeiro arranque poderá ser variada informação sobre o sistema e finalmente, aparecerá a interface gráfica de *login*. Se não realizou alterações aos utilizadores sugeridos, poderá utilizar o utilizador **tux** e a password **tux**.

Neste momento temos uma máquina virtual criada, com um sistema operativo instalado e completamente funcional!

Exercício 2.1

No início do guião, a quando da criação da máquina, foi referido que não era necessário alocar muita memória. Verifique isso executando o comando **free -m** na aplicação **Sakura Terminal**. A memória atualmente necessária é a que for designada como **used** menos a **cached** e a **buffers**.

A que conclusão chega?

2.5 Atualização e instalação de software

É normal que o sistema instalado a partir da versão *live* tenha algumas componentes de software desatualizadas e precise de componentes adicionais. Para resolver esses problemas, usaremos a ferramenta de gestão de pacotes **tazpkg** a partir do interpretador de comandos. Assim, depois de fazer *login*, inicie uma consola com o interpretador de comandos (ícone no canto superior esquerdo) e execute as operações seguintes.

Atualização do sistema operativo

Para atualizar o software já instalado, execute os comandos:

1. Mude o utilizador para **root** para poder administrar a máquina. A senha que o comando pedirá é a de **root** (será igualmente “**root**”, se não a mudou no processo de instalação).

```
su -
```

(Em *Ubuntu* seria equivalente fazer **sudo -s**.)

2. Carregue a lista de pacotes disponíveis para o seu sistema operativo:

```
tazpkg recharge
```

(Em *Ubuntu*, o comando equivalente seria **apt-get update**.)

3. Atualize os pacotes:

```
tazpkg upgrade
```

(Em *Ubuntu*, o comando equivalente seria **apt-get upgrade**.)

Instalação de software adicional

O sistema operativo que acabámos de instalar é minimalista, pelo que precisaremos de instalar algumas componentes adicionais.

1. Procure na lista de pacotes algum que possua uma versão mais avançada do editor **vi** (nomeadamente, o editor **vim**):

```
tazpkg search vim
```

(Em *Ubuntu*, o comando equivalente seria **apt-cache search vim**.)

2. Instale um dos pacotes listados (o **vim**, por exemplo):

```
tazpkg get-install vim
```

(Em *Ubuntu*, o comando equivalente seria **apt-get install vim**.)

Módulos especiais para o sistema operativo convidado

É normal os virtualizadores disponibilizarem módulos de software adicional para instalar nos sistemas convidados, a fim de melhorar o seu desempenho e acrescentar funcionalidades úteis. Uma dessas funcionalidades é a capacidade de usar diretórias do sistema hospedeiro a partir da máquina virtual. O resultado é que dentro sistema convidado irá aparecer um directório que corresponde a um outro directório existente no sistema hospedeiro. Ficheiros que estejam dentro deste directório serão partilhados.

A instalação dos módulos especiais para o convidado faz-se através do gestor de pacotes, se o repositório da distribuição os disponibilizar,¹⁵ ou através de um CD virtual disponibilizado pelo próprio virtualizador. No caso do *VirtualBox*, isso consegue-se da seguinte forma: numa máquina virtual ativa, selecionar a opção **Dispositivos** da barra superior da janela e escolher a opção **Instalar Adições de Convidado...**

Após o passo anterior, o CD virtual ficará acessível numa subdiretoria de **/media** após a sua montagem manual pelo utilizador, ou automática por ferramentas de navegação pelo sistema de ficheiros (procure no menu **Aplicações, Ferramentas de sistema, Gerenciador de arquivos** ou **Montagem de dispositivos**).

A instalação dos módulos adicionais segue então os seguintes passos, atuando sempre como **root**, como antes. Em primeiro lugar é preciso instalar algumas ferramentas e ficheiros de desenvolvimento que serão necessários para produzir os módulos especiais:

```
tazpkg get-install bzip2  
tazpkg get-install gcc  
tazpkg get-install make  
tazpkg get-install linux-module-headers
```

É possível que algumas destas aplicações já existam no sistema. Neste caso o pedido de instalação é ignorado.

Feito isto, monta-se o CD no sistema de ficheiros:

¹⁵Por exemplo, num sistema *Ubuntu* deve bastar executar o comando **apt-get install virtualbox-guest-x11**.

```
mount /media/cdrom
```

Mudando para a diretoria raiz do CD, executa-se a aplicação de instalação dos módulos especiais:

```
cd /media/cdrom  
sh ./VBoxLinuxAdditions.run
```

No final deste comando, que durante a sua execução pode apresentar alguns erros não graves, os módulos especiais foram criados e alguns estão já instalados (os seus nomes começam por **vbox**):

```
lsmod
```

Uma deles, o módulo **vboxsf**, é o que permite aceder a diretórias do sistema de ficheiros do hospedeiro. Poderá ter de reiniciar o sistema para que sejam aplicados todos os módulos compilados.

2.6 Partilha de ficheiros entre hospedeiro e convidado

Vamos agora definir uma partilha de uma diretoria do hospedeiro para uma máquina virtual e aceder a essa diretoria através do sistema convidado. Na janela da máquina virtual selecione a opção **Dispositivos** da barra superior da janela e escolha a opção **Pastas Partilhadas....**. Selecione o botão à direita que possui o sinal +, indique a localização da diretoria (pasta) do hospedeiro que quer partilhar e o nome que lhe quer dar para efeitos de partilha (por exemplo, **XPTO**). Finalmente, antes de terminar indique se quer apenas permitir ler da diretoria partilhada (para a proteger de escritas involuntárias ou maliciosas), e se quer tornar permanente a partilha (i.e., se a partilha continuará ativa em execuções futuras da máquina virtual).

Feita esta configuração, feche as janelas da mesma e volte ao interpretador de comandos da máquina virtual, onde deverá executar o seguinte comando:

```
mount -t vboxsf XPTO /mnt
```

Este comando indica que a diretoria exportada pelo hospedeiro com o nome **XPTO** deverá ser montada na diretoria **/mnt** através do módulo **vboxsf**. Após a execução deste comando poderá constatar que tem acesso à diretoria do hospedeiro mudando para a mesma e listando o seu conteúdo:

```
cd /mnt  
ls -la
```

Esta funcionalidade de partilha é muito útil para desenvolvimento de componentes que precisam de ser vistas por diversas máquinas. Por exemplo, podemos ter uma aplicação em rede, com um cliente a correr numa máquina e um servidor noutra, e ambos podem ser máquinas virtuais na mesma máquina hospedeira, e as aplicações cliente e servidor podem ser desenvolvidas no hospedeiro e exportadas através destes sistemas de ficheiros tanto para o cliente como para o servidor.

2.7 Duplicação de máquinas virtuais

A duplicação de máquinas virtuais é uma tarefa trivial, uma vez que não requer *hardware* adicional. Como as máquinas virtuais se executam sobre *hardware* virtual, e a “criação” deste último não tem limites, é possível duplicar as máquinas e, dessa forma, criar rapidamente um conjunto de máquinas homogéneas e com um sistema à partida igual (depois cada um pode evoluir separadamente).

Uma máquina virtual como a que criámos anteriormente é completamente descrita por dois ficheiros:

Ficheiro .vbox — Este ficheiro possui fundamentalmente uma descrição textual do *hardware* da máquina virtual.

Ficheiro .vdi — Este ficheiro é o disco virtual usado pela máquina virtual.

Para duplicar uma máquina virtual basta duplicar estes dois ficheiros e registar a nova máquina na lista de máquinas conhecida pelo gestor de máquinas virtuais do *VirtualBox*. Este processo manual pode ser transparentemente realizado através de própria interface

deste gestor, que permite *clonar* máquinas virtuais.

Exercício 2.2

Usando a interface do gestor de máquinas virtuais do *VirtualBox*, clone a máquina virtual anteriormente criada. A clonagem inclui várias perguntas. Tente perceber o que é perguntado e responder da forma que considerar mais acertada. Um aspeto interessante é se a clonagem é completa ou por ligação. Uma clonagem completa efetivamente duplica todos os recursos. Significa que, para o hospedeiro, a nova máquina virtual irá ocupar o mesmo espaço em disco do que a original. Também significa que as duas máquinas são independentes. A clonagem através de uma ligação cria uma máquina dependente da primeira. A vantagem é que apenas ocupa espaço em relação às alterações feitas no clone. Esta opção é muito útil se possuir uma máquina estável e desejar criar uma máquina temporária para testar algum aspetto.

Depois de clonar a máquina virtual observe o que de novo apareceu na diretoria onde se guardaram os dados da máquina virtual anterior.

2.8 Para aprofundar o tema

Exercício 2.3

Instale o software *VirtualBox* no seu computador pessoal e depois instale uma máquina virtual com o sistema Ubuntu, disponibilizado no site <http://www.ubuntu.com>. Poderá utilizar esta máquina virtual em várias disciplinas, nomeadamente Laboratórios de Informática e Programação 1.

Se tiver espaço disponível, não configure a máquina, crie um clone e utilize-o. A vantagem é que a máquina original ficará sempre disponível para clonar quando necessitar de criar uma nova máquina com Ubuntu. Assim, evita repetir o processo de instalação do sistema operativo.

Exercício 2.4

Porquê ter apenas uma máquina virtual? Pode clonar a mesma máquina múltiplas vezes e ter várias a executar simultaneamente. Qual o número máximo que consegue ter? Que recursos do seu computador limitam a criação de mais máquinas virtuais? Consegue ter máquinas virtuais dentro de máquinas virtuais?

Exercício 2.5

Num sistema Ubuntu, use o comando **adduser** para adicionar um outro utilizador ao sistema.

Tente aceder a uma sessão com esse utilizador. Verifique a que grupos pertence e se pode executar o comando **sudo**.

Pode depois apagar este utilizador através do comando **deluser**. O que aconteceu ao diretório pessoal desse utilizador?

Exercício 2.6

Utilizando o comando **apt-get** instale a aplicação **synaptic** e experimente-a. Pode agora, de uma forma gráfica, explorar todas as aplicações disponíveis para o sistema *Ubuntu*.

3

TEMA

Produção de documentos com **L^AT_EX**

Objetivos:

- Compilação de documentos L^AT_EX.
- Estruturação de documentos L^AT_EX.
- Referências bibliográficas com L^AT_EX e BibTeX.

3.1 Introdução

O L^AT_EX é uma ferramenta de apoio à produção de documentos complexos (particularmente de índole técnica). O seu paradigma de funcionamento é completamente diferente do de outras ferramentas mais populares atualmente, como o Word da Microsoft. Enquanto editores tradicionais, denominados What You See Is What You Get (WYSIWYG) focam-se na construção simultânea de todo o documento, incluindo aspecto gráfico e conteúdo, o L^AT_EX pressupõe que o autor se deve focar no conteúdo, categorizando apenas o texto produzido, sem se preocupar imediatamente com o aspeto. Este tipo de sistemas são denominados por What You See Is What You Mean (WYSIWYM).

Portanto, ao escrever um documento utilizando L^AT_EX, devemos focar-nos em adicionar conteúdo e em especificar o que queremos como título, secção, tabela, ou que imagens queremos incluir. Aspetos como o estilo, tipo de letra, localização das imagens, espaçamentos, paginação e mesmo hifenação ficam a cargo do L^AT_EX. O autor pode refinar estes pontos, mas à partida não terá de se preocupar com eles. (Para mais informação sobre como os modificar, pode consultar [1][2].)

O L^AT_EX é a ferramenta por excelência para a realização de relatórios de caráter técnico, publicações científicas e mesmo guiões para aulas (de que este é um exemplo). Neste guiaão iremos descrever os conceitos elementares para produzir um documento minimamente estruturado em L^AT_EX.

3.2 Ações de preparação

Esta aula irá utilizar programas que estão disponíveis nos computadores da sala. Se realizar este guiaão durante a aula, não é necessário qualquer ação adicional.

Se realizar este passo em casa, poderá instalar todas as ferramentas adicionais na imagem virtual fornecida através do comando:

```
apt-get install texmaker
```

Se quiser instalar o L^AT_EX no seu computador, e dependendo do sistema operativo que utilize, instale o TexLive ou o MikTex.

Se estiver a utilizar um sistema *Linux* como o Ubuntu, pode executar:

```
apt-get install texlive
```

3.3 Compilação de documentos L^AT_EX

Os documentos L^AT_EX são produzidos editando ficheiros de texto com um editor elementar, como o **vim**. Esta maneira de trabalhar facilita a edição colaborativa pois reduz problemas de compatibilidade entre sistemas. Qualquer editor de texto pode ser utilizado, o que permite que um documento seja facilmente editado. Este mesmo guião foi parcialmente realizado em sistemas *Windows*, *OS X* e *Linux*.

Os ficheiros L^AT_EX são então facilmente alterados por humanos, contendo o texto que deverá constar do documento final a produzir, juntamente com comandos relativas à sua formatação.

De notar que ao editar um documento através do sistema L^AT_EX, não estaremos a ver como o documento realmente irá ser apresentado. Para isso é necessário um passo de compilação. Esta compilação converte o documento L^AT_EX para um outro formato, sendo que o mais utilizado é o PDF. Pode considerar que o conceito é semelhante ao da compilação de um programa Java no formato **.java** para um ficheiro **.class**.

Existem editores especializados que permitem ter uma vista quase em tempo real do documento final. Na realidade, estes editores estão constantemente a compilar o documento, apresentando depois o resultado ao utilizador. Um exemplo é o **gummi**, outro o **texmaker**, mas existem ainda muitos outros¹⁶.

Para compilar um documento L^AT_EX é necessário executar o compilador. Num interpretador de linhas de comandos, como a **bash**, isso é feito colocando a seguir ao nome do compilador (por exemplo, **pdflatex**) o nome do ficheiro a compilar (a extensão **.tex** pode ser omitida). Por exemplo, pressupondo que o nosso ficheiro se chama **hello.tex**, poderemos compilar o ficheiro para PDF através do comando:

```
$ pdflatex hello.tex
```

O resultado deverá ser um conjunto de ficheiros auxiliares (que podem ser removidos), e um documento com extensão PDF. Na Secção 3.12 voltaremos a este assunto.

¹⁶Veja <http://tex.stackexchange.com/questions/339/latex-editors-ides>

Exercício 3.1

No sítio da disciplina existe um ficheiro chamado **hello.tex**. Obtenha este ficheiro. O conteúdo deste ficheiro ainda não é relevante. Compile este ficheiro e verifique que é produzido um ficheiro com extensão PDF.

Verifique que consegue visualizar o conteúdo do ficheiro L^AT_EX utilizando o comando **vim**.

Pode igualmente editar o ficheiro através da aplicação **texmaker**. A vantagem desta aplicação é já ter algumas funcionalidades que auxiliam a produção de documentos L^AT_EX.

Para visualizar o documento produzido, pode utilizar a aplicação **evince**.

3.4 Estrutura obrigatória de um documento

A estrutura obrigatória de um documento passa em primeiro lugar, e obrigatoriamente, pela definição do seu tipo de estrutura, pela definição do tipo de letra base (tipo e tamanho), pela definição da dimensão da página, pela definição de como cada página está organizada (uma coluna, duas colunas), etc.

Esta estrutura é definida pelo comando `\documentclass[options]{class}`. O seu parâmetro obrigatório é a classe de documento, a qual define o tipo base de documento. Eis alguns exemplos de classes: **book**, **report**, **article**. Os parâmetros opcionais podem ser (lista não exaustiva):

a4paper, **a5paper**, **etc.** : dimensão da folha.

10pt, **11pt**, **12pt** : dimensão do tamanho da letra.

onecolumn ou **twocolumn** : texto com uma ou duas colunas.

oneside ou **twoside** : só frente ou frente e verso.

Este é um exemplo para especificar um relatório em páginas a4, com letras de 11pt, a duas colunas e frente-e-verso:

```
\documentclass[a4paper,11pt,twocolumn,twoside]{report}
```

Após esta especificação, o conteúdo efetivo do documento deverá ser colocado entre dois comandos: `\begin{document}` e `\end{document}`.

```
\documentclass[a4paper]{article}

\begin{document}
% conteúdo do documento

\end{document}
```

Exercício 3.2

Copie o documento `hello.tex` para um novo ficheiro e produza o PDF respetivo. Coloque novas opções e/ou altere as que já existem. Observe o resultado. Coloque muito mais conteúdo, de forma a produzir múltiplas páginas, altere novamente as opções, e observe o resultado.

3.5 Carateres especiais do L^AT_EX

No ficheiro que já compilou pode verificar que, além do texto, existem diversas instruções, que denominámos por comandos. Estes usam carateres que normalmente não fazem parte do conteúdo de um texto de um documento, a saber: `\{`, `\}`, `\[`, `\]`, `\$`, `\%`, `\~`, `\#`, `_`, `\^`, e `\&`. Neste momento é prematuro introduzir o significado de todos eles, vamos apenas apresentar alguns já de seguida.

O caráter '\'

O caráter '`\`' é o símbolo base da definição de comandos, daí que normalmente está no seu início:

\o → ø

\oe → œ

\LaTeX → L^AT_EX

Para escrever o caráter '`\`' num texto (o que é raro) é preciso recorrer a um comando:

`\textbackslash` → \

Exercício 3.3

Edite o documento e adicione estes comandos depois do texto “Hello World!”. Não se esqueça de compilar o documento e verificar o resultado!

O caráter '\$'

O caráter '\$' é usado para sinalizar a entrada e saída do modo matemático. Por omissão o conteúdo fonte de um documento L^AT_EX é interpretado em modo de texto, entrando no modo matemático e saindo do mesmo com este caráter. O modo matemático será abordado mais adiante.

`text mode, $math mode$, text mode` → text mode, *mathmode*, text mode

Para escrever o caráter '\$' num texto é preciso recorrer a um comando:

`\$` → \$

Os carateres '{' e '}'

Estes carateres, chavetas, servem para dois fins: definir um contexto e definir um parâmetro de um comando.

Um contexto consiste num bloco de texto, entre as duas chavetas (a de abrir e a de fechar), que delimita o universo de abrangência de comandos de formatação expressos nesse mesmo bloco. Ou seja, uma formação imposta dentro de um bloco não tem efeito fora do bloco. Neste âmbito, as chavetas servem também para delimitar corretamente os comandos que não possuem parâmetros, nomeadamente para não interpretar o espaço que é usado após os mesmos como sendo um terminador.

`\o e` → øe (o espaço é interpretado como terminador de '\o')

`{\o} e` → ø e (o espaço é preservado como separador de '\o' e 'e')

Quando os comandos possuem parâmetros obrigatórios, estes são normalmente indicados entre chavetas (alguns só funcionam em modo matemático, delimitados por \$) :

`\textbf{a}` → a

`\vec{x}` → \vec{x}

`\sqrt{x}` → \sqrt{x}

`\textsc{texto em Small Caps}` → TEXTO EM SMALL CAPS

`\fbox{texto emoldurado}` → texto emoldurado

`\footnote{Nota de rodapé}` → ¹⁷

Quando os comandos possuem mais do que um parâmetro, cada um dos parâmetros é indicado entre chavetas:

`\frac{x}{y}` → $\frac{x}{y}$

Para escrever os caracteres '{' ou '}' num texto (o que é raro) é preciso recorrer a um comando:

`\{ \}` → { }

Exercício 3.4

Edite o documento e adicione alguns destes comandos junto do restante texto.

Pode igualmente experimentar outras formatações de texto como **negrito** (`\textbf{negrito}`), *itálico* (`\textit{itálico}`), ou como uma máquina de escrever (`\texttt{máquina de escrever}`).

¹⁷Nota de rodapé

Os carateres '[' e ']'

Este carateres normalmente são interpretados literalmente (i.e., sem qualquer significado especial) exceto quando delimitam opções de comandos:

`\sqrt[3]{x}` → $\sqrt[3]{x}$

O caráter '%'

O caráter '%' é usado para sinalizar o início de texto que não deverá ser interpretado (usualmente um comentário), até ao final da linha corrente.

`texto interpretado, %texto ignorado` → texto interpretado,

Para escrever o caráter '%' num texto é preciso recorrer a um comando:

`\%` → %

O caráter '˜'

O caráter '˜' é usado para representar um espaço entre dois elementos que nunca devem ficar separados em duas linhas consecutivas. Por exemplo, quando se escreve “*a arquitetura apresentada na figura 5*” não é correto que ocorra uma translineação (mudança de linha) entre “figura” e “5”, porque isso prejudica a leitura do texto. Para sinalizar esse facto, escreve-se:

`a arquitetura apresentada na figura~5` → a arquitetura apresentada na figura 5

Para realmente escrever o caráter '˜' num texto é preciso recorrer a um comando:

`\textasciitilde` → ~

3.6 Funcionalidades adicionais (linguísticas)

Normalmente nos documentos L^AT_EX usam-se funcionalidades para além das incluídas da especificação base do compilador. Essas funcionalidades são disponibilizadas em pacotes (*packages*), os quais são normalmente indicados no início de um documento através do comando `\usepackage[options]{packages}[version]`. É também comum usar um comando destes por cada pacote usado.

Um exemplo de pacote que lhe poderá ser útil é o que permite interpretar corretamente caracteres para além dos contemplados na tabela American Standard Code for Information Interchange (ASCII)[3]¹⁸, de que são exemplo todos os caracteres latinos acentuados e os caracteres de outros alfabetos que não o latino (grego, cirílico, árabe, etc.). Este pacote é o **inputenc**, que tem como opção a escolha do modo como os caracteres não ASCII estão codificados no documento fonte:

```
\usepackage[encoding name]{inputenc}
```

Este guião, por exemplo, foi escrito em L^AT_EX com codificação utf8¹⁹, o que implicou a inclusão deste pacote do seguinte modo:

```
\usepackage[utf8]{inputenc}
```

Exercício 3.5

Observe o resultado do PDF obtido quando se usa ou não o pacote (experimente comentar a linha da sua inclusão).

Outro pacote que é útil para português é o **babel**, que ensina o L^AT_EX a realizar corretamente a translineação de palavras:

```
\usepackage[language]{babel}
```

Caso um documento inclua textos de várias línguas, como português e inglês, tal é suportado pelo pacote:

```
\usepackage[portuguese,english]{babel}
```

Finalmente, outro pacote que é útil para português é o **fontenc**, que indica o tipo de codificação de caracteres do resultado da compilação do L^AT_EX. Aconselha-se o uso da codificação T1, porque a mesma permite lidar convenientemente com caracteres acentuados.

```
\usepackage[T1]{fontenc}
```

¹⁸Também pode consultar: <http://en.wikipedia.org/wiki/ascii>

¹⁹utf8 é uma codificação que pretende ser uniforme para todas as línguas.

Exercício 3.6

Crie um novo ficheiro contendo texto em português e em inglês. Cada parágrafo deverá ocupar mais do que uma linha de texto e deverá ter uma palavra translineada.

Verifique que o L^AT_EX procede à translineação automática do texto apresentado. No entanto esta poderá não ser a mais adequada à linguagem em causa.

O comando `\selectlanguage{language}` permite especificar qual a língua do texto que se lhe segue. Use-o.

3.7 Dimensão das letras

A dimensão base das letras é definida com o comando `\documentclass`, como vimos na Secção 3.4. Este tamanho é indicado com o comando `\normalsize`. relativamente a este tamanho o L^AT_EX possui outras dimensões, a saber:

Comando	Resultado para este guião
<code>\tiny</code>	texto de exemplo
<code>\scriptsize</code>	texto de exemplo
<code>\footnotesize</code>	texto de exemplo
<code>\small</code>	texto de exemplo
<code>\normalsize</code>	texto de exemplo
<code>\large</code>	texto de exemplo
<code>\Large</code>	texto de exemplo
<code>\LARGE</code>	texto de exemplo
<code>\huge</code>	texto de exemplo
<code>\Huge</code>	texto de exemplo

Estas indicações da dimensão do tipo de letra pode ser utilizando ao longo do texto, de forma a alterar palavras ou mesmo letras individuais. No entanto, não é comum de ser utilizado desta forma pois irá produzir um documento com pouca coerência.

Posso {\small mudar} de {\huge tamanho} de letra sempre que me apetecer. Essa \Large mudança permanece em efeito {\normalsize até} ao próximo comando de mudança de tamanho, caso exista, {\normalsize\space} ou até ao fim do bloco (delimitado por {\ e }) onde está inserida.



Posso mudar de tamanho de letra sempre que me apetecer. Essa mudança permanece em efeito até ao próximo comando de mudança de tamanho, caso exista, ou até ao fim do bloco (delimitado por { e }) onde está inserida.

Como pode reparar, quando se escreve um documento, não existe a necessidade de ajustar o tamanho das letras para valores específicos. Apenas se indica o que se pretende que seja maior ou menor, com base na referência do documento. Desta forma ele ficará sempre coerente.

3.8 Elementos estruturais de documentos

Título

Nos documentos com título o mesmo é criado com recurso aos seguintes comandos:

\title{título}: Define o texto do título. Se o mesmo tiver de ser dividido por várias linhas deverá ser usado o comando \\ para mudar de linha.

\author{autores}: Define os autores. Também neste caso os autores podem ser divididos por várias linhas usando o comando \\ para mudar de linha. Nesta parte podem ainda ser incluídos outros atributos relativos aos autores para além do seu nome, como afiliações, endereços, etc.

\date{data}: Define a data de criação do documento, que é apresentada no título. O comando \today fornece a data atual. Para não incluir data deve-se usar uma data vazia (ou seja, nada entre as chavetas).

\maketitle: Cria o título de acordo com o texto do mesmo, os autores e a data antes definidos.

Convém que estes comandos sejam colocados antes do início formal do documento, isto é, antes de \begin{document}, tal como acontece no ficheiro `hello.tex`.

Partes, capítulos e secções

Os documentos podem ser seccionados com os seguintes comandos, cujo nome é sugestivo:

\part{title}: inicia uma parte de documento (pode englobar vários capítulos);

`\chapter{title}`: inicia um capítulo (pode englobar vários seções);
`\section{title}`: inicia uma seção (pode englobar vários subseções);
`\subsection{title}`: inicia uma subseção (pode englobar vários sub-subseções);
`\subsubsection{title}`: inicia uma sub-subseção (pode englobar vários parágrafos);

Todos estes comandos de seccionamento têm obrigatoriamente um título, o qual terá a fonte e dimensão de letra apropriadas escolhidas automaticamente pelo L^AT_EX. Eles são igualmente numerados de forma automática e segundo estilos definidos pelo tipo de documento (numeração romana ou árabe, letras, etc.). Mais uma vez, ao utilizar L^AT_EX, o autor especifica qual a função do texto, deixando a composição gráfica a cargo do L^AT_EX.

Exercício 3.7

Experimente usar todos estes comandos de seccionamento num documento. Note que nem todos são possíveis em cada tipo de documento (v.g. com documentos do tipo `article` não existe `\chapter`).

Exercício 3.8

Experimente incluir o comando `\tableofcontents` logo no início do texto útil do documento. Observe o resultado após compilar o documento. Altere alguns dos títulos das seções, compile e observe o resultado. Poderá ter de repetir a compilação para a criação correta deste índice.

Alguns dos títulos criados por estes comandos, como os de `\part`, `\chapter` e `\tableofcontents`, incluem uma parte em inglês. Esta parte é automaticamente colocada pelo L^AT_EX de acordo com variáveis pré-definidas e com a língua definida, as quais podem ser alteradas no documento com os seguintes comandos:

```
\renewcommand{\partname}{Parte}
\renewcommand{\chaptername}{Tema}
\renewcommand{\contentsname}{Índice}
```

Não é possível alterar o nome das seções pois uma seção apenas inclui um número e um título. Pelo contrário, um capítulo inclui um texto (ex., Capítulo).

Se for utilizado o package `hyperref` é possível definir os nomes que são inseridos no texto a quando de referências para outras partes do documento (ex. Como explicado na Seção 1.1), utilizando o comando `\autoref`. Neste caso seria utilizado:

```
\usepackage{hyperref}

.....
\renewcommand{\sectionname}{Seção}
..

\section{Teste}
\label{sec:teste}

Como foi apresentado na \autoref{sec:teste}...
```

Exercício 3.9

Altere a ordem de declaração das línguas junto do pacote `babel`. Volte a compilar e verifique quais os títulos fornecidos.
Altere novamente o seu documento para incluir estas alterações logo no início do texto útil do documento. Observe o resultado após compilar o documento.

Listas de itens

O L^AT_EX inclui várias formas de listas de itens, tanto numeradas como não numeradas. Estas listas são automaticamente indentadas pelo L^AT_EX.

Uma lista não numerada é criada através do ambiente `itemize`:

```
Uma lista não numerada é:
\begin{itemize}
\item este é um item;
\item este é outro item;
\item etc.
\end{itemize}
Aqui retoma a indentação.
```



Uma lista não numerada é:

- este é um item;
- este é outro item;
- etc.

Aqui retoma a indentação.

onde cada item é iniciado através do comando `\item`. Em cada comando `\item` pode indicar um conteúdo alternativo que o L^AT_EX usa para iniciar o item (v.g. `\item[+]` usa o

símbolo '+' , ver abaixo):

```
Uma lista não numerada é:  
\begin{itemize}  
  \item[+] este é um item;  
  \item[+] este é outro item;  
  \item[+] etc.  
\end{itemize}
```

Aqui retoma a indentação.



Uma lista não numerada é:

- + este é um item;
- + este é outro item;
- + etc.

Aqui retoma a indentação.

Outra forma de criar uma lista não numerada é através do ambiente **description**, onde cada item é iniciado por texto (ou título) arbitrário em vez de um símbolo gráfico:

```
Uma lista não numerada é:  
\begin{description}  
  \item[UA] - Univ. de Aveiro;  
  \item[UC] - Univ. de Coimbra;  
  \item[UP] - Univ. do Porto.  
\end{description}
```

Aqui retoma a indentação.



Uma lista não numerada é:

- UA** - Univ. de Aveiro;
- UC** - Univ. de Coimbra;
- UP** - Univ. do Porto.

Aqui retoma a indentação.

onde cada item é iniciado através do comando `\item[title]`; o parâmetro opcional (que pode ser omitido) é escrito em negrito.

As listas numeradas são criadas através do ambiente `enumerate`:

```
Uma lista numerada é:  
\begin{enumerate}  
  \item este é um item;  
  \item este é outro item;  
  \item etc.  
\end{enumerate}  
Aqui retoma a indentação.
```



Uma lista numerada é:

1. este é um item;
2. este é outro item;
3. etc.

Aqui retoma a indentação.

onde cada item é iniciado através do comando `\item` e numerado automaticamente pelo L^AT_EX.

A numeração segue um formato por omissão (v.g. numeração árabe) a qual pode ser mudada em cada lista logo após a sua iniciação, como neste exemplo:

```
Uma lista numerada é:  
\renewcommand{\theenumi}{\Roman{enumi}}  
\begin{enumerate}  
  \item Este é um item;  
  \item Este é outro item;  
  \item etc.  
\end{enumerate}  
Aqui retoma a indentação.
```



Uma lista numerada é:

- I. Este é um item;
- II. Este é outro item;
- III. etc.

Aqui retoma a indentação.

onde cada item é numerado com numeração romana maiúscula; o termo `enumi` representa o contador usado na listagem. No exemplo abaixo mostra-se como se pode misturar vários tipos de numeração:

```
Múltiplas listas numeradas e encadeadas:  
\renewcommand{\theenumi}{\Roman{enumi}}  
\begin{enumerate}  
  \item Este é um item, com subitens:  
    \renewcommand{\theenumi}{\alpha{enumi}}  
    \begin{enumerate}  
      \item Este é um item;  
      \item Este é outro item;  
      \item etc.  
    \end{enumerate}  
  \item Este é outro item;  
  \item etc.  
\end{enumerate}  
Aqui retoma a indentação.
```



Múltiplas listas numeradas e encadeadas:

I. Este é um item, com subitens:

- a) Este é um item;
- b) Este é outro item;
- c) etc.

II. Este é outro item;

III. etc.

Aqui retoma a indentação.

No primeiro nível usou-se numeração Romana maiúscula; no segundo nível usaram-se letras minúsculas (começando em 'a'). **Nota:** a indentação usada neste último exemplo é irrelevante para o L^AT_EX, ela apenas facilita a compreensão do leitor.

Objetos flutuantes: figuras e tabelas

No L^AT_EX existe um conceito, denominado de objetos flutuantes (*floats*), que consiste na existência de objetos cujo lugar não precisa de ser fixo, porque pode ser referenciado de outra forma (v.g. através de um número de referência). Há dois tipos de objetos flutuantes: figuras e tabelas.

As figuras e as tabelas são em tudo idênticas, exceto na maneira como é apresentada a sua legenda. São objetos que podem conter qualquer conteúdo (texto, imagens, etc.) e que possuem pelo menos

uma legenda. As legendas são numeradas e o seu número pode ser usado noutras locais do texto para referir o objeto.

Uma figura é declarada do seguinte modo:

```
\begin{figure}[h]
\centerline{\fbox{Conteúdo da figura, pode ser texto, imagem, etc.}}
\caption{Legenda da figura}
\end{figure}
```

e o resultado é a figura abaixo:

Conteúdo da figura, pode ser texto, imagem, etc.

Figura 3.1: Legenda da figura

A opção 'h' indicada no início do ambiente **figure** indica que a figura deverá primordialmente ser colocada no lugar onde aparece em termos de fluxo de texto. Mas tal nem sempre é possível. O L^AT_EX possui heurísticas internas que determinam a melhor localização de um objeto flutuante de forma a que o documento fique mais equilibrado. O resultado é que os objectos flutuantes nem sempre irão ficar onde são declarados de forma a melhorar o aspetto final do documento.

Outras opções para além da 'h' são:

- t** - A figura é colocada no topo da página.
- b** - A figura é colocada no fundo da página.
- p** - A figura é colocada numa página isolada.

Várias destas opções podem ser usadas, indicando uma ordem de preferência para o L^AT_EX.

Tanto as figuras como as tabelas podem ser referidas através do seu número. Em L^AT_EX essa funcionalidade é obtida através dos comandos **\label** e **\ref**. Se utilizar o package *hyperref*, é possível também usar o comando **\autoref**. Estes comandos são descritos em maior pormenor na secção seguinte.

As tabelas seguem exatamente as mesmas regras que as figuras, simplesmente usa-se o ambiente **table** e a legenda é identificada de forma diferente. Assim, uma tabela é declarada do seguinte modo:

```
\begin{table}[htp]
\caption{Exemplo de uma tabela}
\centerline{Conteúdo de uma tabela}
\label{tabela-exemplo}
\end{table}%
```

e o resultado é a tabela abaixo:

Tabela 3.1: Exemplo de uma tabela

Conteúdo de uma tabela

Note-se que nesta tabela usou-se o comando `\label` para a referir neste texto pelo seu número (Tabela 3.1, obtido através da etiqueta `tabela-exemplo` no comando `\autoref{tabela-exemplo}`). Veja a secção seguinte para obter mais pormenores sobre este tipo de comandos. Também nesta tabela usou-se o comando `\caption` antes do conteúdo da tabela, o que causou que a legenda da tabela aparecesse no seu topo.

Exercício 3.10

Edito o seu documento e coloque-lhe várias figuras e tabelas, contendo uma ou mais figuras/tabelas por objeto flutuante. Experimente as diversas colocações e veja o que acontece com cada uma delas. Confirme também que os objetos flutuantes do mesmo tipo (figuras ou tabelas) nunca ficam por uma ordem diferente daquela em que são declaradas, muito embora possam ficar fora do local onde foram indicadas no documento fonte.

Referências a partes do texto

Qualquer identificador numérico usado num documento L^AT_EX, seja ele de parte, capítulo, secção, lista numerada, figura, tabela, e outros, pode ser usado no texto através dos comandos `\label`, `\ref` e `\autoref`.

O comando `\label` associa uma etiqueta a um número usado num dos elementos acima referidos (parte, capítulo, etc.). Para isso, o normal é colocar a etiqueta imediatamente após o comando que produz o número que se pretende, muito embora isso não seja estritamente necessário. Por exemplo, o início desta secção foi escrito com o seguinte conteúdo:

```
\subsection{Referências a partes do texto}
\label{refs.Section}
```

Qualquer identificador numérico usado num documento {\LaTeX}, seja ele de parte, capítulo, secção, lista numerada, figura, tabela, e outros, pode ser usado no texto através dos comandos ...

Assim, o comando `\ref{refs.Section}` produzirá o número 3.8. A vantagem deste método é que o texto pode ser livremente alterado, inclusive alterada a sua ordem, que os números utilizados para as imagens, figuras e as respetivas referências é mantido de forma ordenada. Também pode ser usado o método `\autoref{refs.Section}` que irá gerar uma descrição mais longa: Secção 3.8.

O uso destes comandos obriga normalmente a duas compilações de um documento para que os valores apresentados pelas referências estejam corretos. Na primeira compilação é obtido e guardado o número associado a cada etiqueta, na segunda compilação o número associado a cada etiqueta é usado onde quer que a etiqueta seja referida. Deste modo, é perfeitamente possível ter uma referência a uma etiqueta que aparece mais tarde no texto.

Mais uma vez, o autor não tem de se preocupar com a numeração. Apenas tem de criar identificadores únicos para as suas referências. Para fazer isto, por vezes usam-se esquemas como `fig.etiqueta_da_figura`, ou `table.etiqueta_da_tabela`, embora o L^AT_EX não imponha qualquer estilo de referência.

Em documentos grandes pode ser difícil ao autor lembrar-se dos nomes das etiquetas que usou. Durante a escrita (mas não na versão final!) do documento, a inclusão do comando `\usepackage{showlabels}` no preâmbulo do documento (antes de `\begin{document}`) faz com que sejam mostradas nas margens do documento os nomes das etiquetas que estão a ser usadas.

Exercício 3.11

Edite o seu documento e coloque etiquetas em todos os objetos numerados que o mesmo possui. Note que não pode usar etiquetas iguais para objetos diferentes. Uma vez feita a atribuição de etiquetas, use o comando `\ref` e ou `\autoref` para obter e apresentar o seu número. Não se esqueça que o comando `\autoref` necessita da package `hyperref`. Inclua depois o comando `\usepackage{showlabels}` no preâmbulo do documento e veja o que mudou.

Disposição de elementos em matriz

A disposição de elementos alinhados segundo uma matriz é muitas vezes útil, nomeadamente para criar tabelas (mas não só). Para esse fim deve ser usado o ambiente `tabular`, em modo texto, ou `array`, em modo matemático. Vamos aqui abordar apenas o primeiro.

Para facilitar a explicação de como se usa o ambiente `tabular` observe-se o seguinte conteúdo L^AT_EX:

```
\begin{tabular}{|l||c|r|} % Alinhamento: Esq, Centro, Dir
%
\hline
    & Temperatura & Humidade \\
Cidade & (\text{ordmasculine} C) & (perc.) \\ \hline\hline
Aveiro & {\large 10} & {\large 90} \\ \hline
Lisboa & {\tiny 13} & {\tiny 84} \\ \hline
Porto & \textbf{9} & \textbf{89} \\ \hline
%
\end{tabular}
```

que produz a seguinte matriz de elementos, ou tabela:

Cidade	Temperatura (°C)	Humidade (perc.)
Aveiro	10	90
Lisboa	13	84
Porto	9	89

O ambiente **tabular** produz uma caixa que, internamente, possui dados organizados de forma matricial. A matriz é especificada linha a linha, e a separação de linhas é indicada com o comando `\\"`. O separador de elementos na mesma linha é o carácter `'&'`, um dos caracteres especiais referidos na Secção 3.5. As linhas e colunas podem ser opcionalmente separadas por linhas horizontais e verticais, respetivamente. A presença das linhas verticais é indicada através do parâmetro de formatação de colunas, nomeadamente através do símbolo `'|'`; a presença das linhas horizontais é indicada através de comando `\hline`. Por omissão os elementos de cada linha são alinhados pela sua base (veja a linha relativa a Aveiro).

O ambiente **tabular** serve para produzir tabelas ou matrizes de elementos mas não está de modo nenhum associado ao objeto móvel tabela. Este último não passa de uma área móvel com uma determinada legenda, referenciável, enquanto que o primeiro serve para dispor elementos espacialmente segundo uma matriz. Para ilustrar esta separação, repare que na pág. 57 usámos um ambiente **tabular** para apresentar as dimensões de letra, e não usámos um objeto flutuante **table**.

Expressões e ambientes matemáticos

As expressões matemáticas são interpretadas num contexto diferente do do texto normal, tendo regras diferentes de gestão do resultado produzido. As expressões matemáticas podem ser colocadas em ambientes de linha delimitados pelos caracteres `'$'`, ou em ambientes matemáticos mais alargados, que podem ocupar um espaço vertical variável.

Uma forma de criar um ambiente matemático é com o comando `$$`, que inicia e termina um ambiente matemático numa nova linha:

a expressão `$$y = ax^2 + bx + c $$` é a forma geral da equação de 2º grau



a expressão

$$y = ax^2 + bx + c$$

é a forma geral da equação de 2º grau

Note-se a diferença quando se usam apenas os caracteres `$` isolados:

a expressão $y = ax^2 + bx + c$ é a forma geral da equação de 2º grau



a expressão $y = ax^2 + bx + c$ é a forma geral da equação de 2º grau

Outra forma de criar um ambiente matemático é recorrendo aos comandos `\[` e `\]`:

a expressão
`\[`
 $y = ax^2 + bx + c$
`\]`
é a forma geral da equação de 2º grau



a expressão

$$y = ax^2 + bx + c$$

é a forma geral da equação de 2º grau

Uma forma alternativa de criar um ambiente matemático com numeração é usando o ambiente `equation`:

a expressão `\ref{2ordem.eq}`
`\begin{equation}`
 $y = ax^2 + bx + c$ `\label{2ordem.eq}`
`\end{equation}`
é a forma geral da equação de 2º grau



a expressão 1

$$y = ax^2 + bx + c \quad (1)$$

é a forma geral da equação de 2º grau

Uma forma alternativa de criar um ambiente matemático multilinha com numeração é usando o ambiente `eqnarray`:

Perímetro e área do círculo:

```
\begin{eqnarray}
P = 2\pi r \\
A = \pi r^2
\end{eqnarray}
```



Perímetro e área do círculo:

$$P = 2\pi r \quad (2)$$

$$A = \pi r^2 \quad (3)$$

Dois dos carateres especiais do L^AT_EX já antes referidos mas ainda não explicados são o `'^'` e o `'_'`. Como se viu acima nas expressões matemáticas, o `'^'` serve para indicar um expoente, ou um elemento superior de um operador matemático, como um somatório ou um integral. Ao invés, o `'_'` serve para indicar um índice, ou um elemento inferior de um operador matemático.

Vejam-se os dois exemplos abaixo para clarificar o seu uso, bem como as duas maneiras como podem ser produzidas as mesmas expressões matemáticas consoante tal acontece num ambiente de linha (primeiro caso) ou num ambiente matemático (segundo caso):

```
$y = \sum_{i=0}^{i=n}{x_i}^{2n}$
$$y = \sum_{i=0}^{i=n}{x_i}^{2n}$$
```



$$y = \sum_{i=0}^{i=n} x_i^{2n}$$

$$y = \sum_{i=0}^{i=n} x_i^{2n}$$

3.9 Inclusão de figuras

É normal os documentos incluírem imagens, porém o L^AT_EX não possui nenhum mecanismo nativo de inclusão de imagens. Tal tem de ser feito através de *packages*.

Um dos *packages* mais usados para este efeito é o **graphicx**. Este *package* permite incluir imagens de vários tipos (bitmap, encapsulated postscript, etc.) e redimensioná-los. A inclusão de imagens pode ou não ser feita no contexto de um objeto flutuante (figura ou tabela).

O exemplo seguinte inclui uma imagem no texto a partir de um ficheiro **ua.pdf** sem usar um objeto móvel

```
Este
\includegraphics[height=24pt]{latex/ua}
é o novo logotipo da UA
```



e o resultado é: Este é o novo logotipo da UA

Também é possível utilizar a mesma imagem dentro de um objeto flutuante. Este é possivelmente o caso mais utilizado:

```
\begin{figure}[h]
\center % Centra as imagens
\includegraphics[height=24pt]{latex/ua}
\caption{Novo logotipo da Universidade de Aveiro}
\label{fig:ualogo.1}
\end{figure}
```

e o resultado é a Figura 3.2.



Figura 3.2: Novo logotipo da Universidade de Aveiro

Na inclusão de um ficheiro é usual indicar o seu nome sem extensão (nos casos acima o nome completo do ficheiro é `ua.pdf` e está localizado no diretório `latex`). Com efeito, podem existir vários ficheiros para a mesma imagem, cada um com o seu formato, e deste modo facilita-se a compilação do documento para formatos de saída diferentes.

O comando de inclusão de imagens possui várias opções, entre as quais as que permitem redimensionar ou de outra forma ajustar a imagem a incluir:

`height` - altura da imagem.

`width` - largura da imagem.

`scale` - fator de escala.

`angle` - ângulo de rotação.

Este comando possui muito mais opções, sendo estas as consideradas mais úteis. Para mais pormenores deve ser consultada a documentação do pacote²⁰. O exemplo seguinte, que resulta na Figura 3.3, mostra como utilizar a mesma imagem, processada de várias formas, com as opções indicadas.

²⁰<ftp://ftp.di.uminho.pt/pub/ctan/macros/latex/required/graphics/grfguide.pdf>

```

\begin{figure}[h]
\center % Centra as imagens
a) \includegraphics{latex/ua}
b) \includegraphics[height=3cm]{latex/ua}
c) \includegraphics[width=20mm]{latex/ua}
d) \includegraphics[scale=.5,angle=90]{latex/ua}
e) \includegraphics[height=5mm,width=3cm]{latex/ua}
\caption{Logotipo da Universidade de Aveiro: a) na dimensão real,  
b) com 3cm de altura, c) com 20mm de largura, d) com altura e largura  
reduzidas a  $\frac{1}{2}$  e simultaneamente rodado 90° e e) com uma modificação  
anamórfica da altura e da largura.}
\label{fig:ualogo.2}
\end{figure}

```

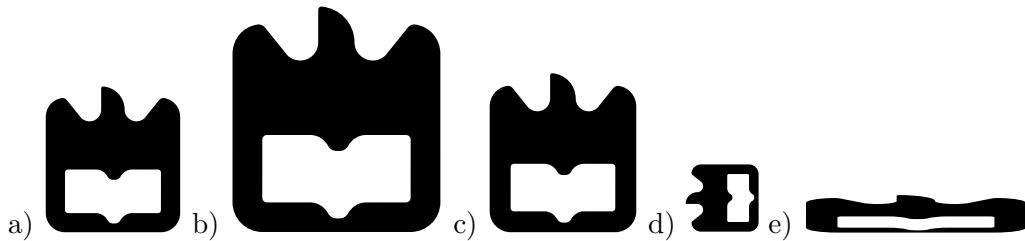


Figura 3.3: Logotipo da Universidade de Aveiro: a) na dimensão real, b) com 3cm de altura, c) com 20mm de largura, d) com altura e largura reduzidas a $\frac{1}{2}$ e simultaneamente rodado 90° e e) com uma modificação anamórfica da altura e da largura.

Exercício 3.12

Obtenha a partir da Internet várias imagens e guarde-as no formato PNG (Portable Network Graphics) ou PDF. Inclua estas figuras no seu documento, em vários locais (no texto, em objetos móveis) e redimensione-as. Observe o resultado no PDF resultante do documento.

3.10 Índices de conteúdos, de figuras e de tabelas

Por vezes é útil que os documentos incluam alguma parte um índice de conteúdos, um índice de figuras ou um índice de tabelas. O comando `\tableofcontents` provoca a inclusão, no local onde está colocado, de um índice de todas as partes, capítulos, seções, etc., usadas no documento. Os comandos `\listoffigures` e `\listoftables` incluem um índice das figuras e tabelas, respetivamente, no local onde os comandos são usados.

O uso destes comandos, à semelhança do que acontece com o comando de referênciação `\ref`, obriga normalmente a duas compilações de um documento para que os conteúdos apresentados pelas tabelas estejam coerentes com os conteúdos presentes no texto (títulos/legendas e respetivos números, páginas).

3.11 Referências bibliográficas

Na produção de documentos técnicos e científicos é fundamental que existam referências bibliográficas indicando trabalhos relacionados ou fontes onde se pode procurar mais informação sobre um assunto. Estas referências são agrupadas num capítulo/secção que normalmente se coloca no final do documento.

Há muitas maneiras de criar a bibliografia de um documento. Neste guia vamos explicar como a mesma se produz a partir de um ficheiro de referências guardadas num formato genérico.

Ficheiro de bibliografia, BibTeX

Um ficheiro de bibliografia é um ficheiro com a extensão `.bib` que possui regtos de documentos num formato independente do documento que os cita. Um regsto é criado por vários comandos com nomes distintos, mas que possuem a mesma estrutura base. A Tabela 3.2 apresenta alguns exemplos.

<code>@book</code>	livro
<code>@article</code>	artigo em revista
<code>@inproceedings</code>	artigo em ata de conferência (<i>proceedings</i>)
<code>@masterthesis</code>	dissertação de mestrado
<code>@phdthesis</code>	dissertação de doutoramento
<code>@techreport</code>	relatório técnico
<code>@manual</code>	manual
<code>@misc</code>	algo de diferente

Tabela 3.2: Comandos para criação de regtos bibliográficos.

Cada regsto é especificado através do conteúdo que se coloca entre as chavetas, normalmente dividido por linhas para facilitar a sua leitura e compreensão por quem o está a editar.

Na Figura 3.4 é apresentado o conteúdo de um ficheiro com regtos bibliográficos (neste caso, apenas 2). Cada regsto descreve uma publicação, neste caso apenas livros, e indica vários dos seus atributos: autor, título, número de páginas, editor, ano de publicação e os seus dois International Standard Book Numbers (ISBNs). O texto indicado imediatamente após a primeira chaveta de cada regsto, na primeira linha, é a sua etiqueta; esta etiqueta servirá para referenciar o regsto sempre que for citado.

A ordem dos atributos é irrelevante (trocámos a ordem de autor e título nos dois regtos propositadamente para demonstrar este facto). A indentação é igualmente irrelevante, apenas facilita a leitura pelo editor. Finalmente, o nome dos atributos pode usar minúsculas ou maiúsculas, como acontece no exemplo apresentado.

Para cada tipo de documento, livro, artigo, etc., existe um conjunto de atributos obrigatórios; existem também atributos opcionais. A lista de atributos é extensa e pode ser consultada em múltiplos documentos; recomenda-se uma consulta rápida na página do BibTeX na Wikipedia²¹.

²¹<http://en.wikipedia.org/wiki/BibTeX>

```

@book{Kopka.03,
  title = {Guide to LaTeX},
  author = {Helmut Kopka and Patrick W. Daly},
  pages = {624},
  publisher = {Addison-Wesley Professional},
  edition = {4th},
  year = {2003},
  note = {ISBN-10: 0321173856, ISBN-13: 978-0321173850}
}

@book{Lamport.94,
  AUTHOR = {Leslie Lamport},
  TITLE = {LaTeX: A Document Preparation System},
  PAGES = {288},
  PUBLISHER = {Addison-Wesley Professional},
  EDITION = {2nd},
  YEAR = {1994},
  NOTE = {ISBN-10: 0201529831, ISBN-13: 978-0201529838}
}

```

Figura 3.4: Exemplo de ficheiro de registo BibTeX contendo dois registo.

Note que a separação de nomes de autores no atributo **AUTHOR** é feita com a palavra **and** e não com vírgulas, como é normal. Tal é feito porque o nome de um autor pode ser colocado com o apelido primeiro, usando-se a vírgula para os separar:

```

AUTHOR = {Lamport, Leslie}
AUTHOR = {Kopka, Helmut and Daly, Patrick W.}

```

Uso de citações bibliográficas

Quando se pretende citar um documento noutra segue-se o seguinte procedimento:

1. Coloca-se num ficheiro de bibliografia (**myBib.bib**, por exemplo) os registo dos documentos a citar (pressupomos os registo da Figura 3.4).
2. No documento que faz a citação usa-se o comando `\cite{Lamport.94}` para criar uma citação do livro de L. Lamport nesse local. Esta ação pode ser feita inúmeras vezes ao longo do documento, sempre com o mesmo resultado prático.
3. No final do documento colocam-se os comandos para incluir (automaticamente) a bibliografia e indica-se quais são os ficheiros que deverão ser considerados como fontes de registo bibliográficos. No nosso caso colocaríamos os seguintes comandos:

```

\bibliographystyle{plain}
\bibliography{myBib}

```

O primeiro comando, `\bibliographystyle`, indica o estilo que deverá ser usado para formatar as citações e as referências bibliográficas (neste caso `plain`, ver secção 3.11). O segundo comando, `\bibliography`, cria um capítulo ou secção com a bibliografia, ao mesmo tempo que indica uma lista de nomes de ficheiros, separados por vírgulas, onde se podem obter os registos das citações realizadas ao longo do documento e que devem ser incluídos nessa bibliografia. Note que os registos bibliográficos presentes nos ficheiros `.bib` que não são citados no documento são omitidos.

Feito isto, a bibliografia no documento é produzida do seguinte modo:

1. Compila-se o ficheiro `.tex` (doc.tex, por exemplo) de forma a gerar o documento no formato final (PDF, por exemplo):

```
$ pdflatex doc
```

2. Compila-se o ficheiro `.aux` (`doc.aux`, neste exemplo) com o comando `bibtex` de forma a gerar um ficheiro de bibliografia (neste caso, `doc.bbl`):

```
$ bibtex doc
```

Neste processo é usado transparentemente o ficheiro `myBib.bib` para fornecer os registos de documentos citados em `doc.tex`. O formato dos conteúdos colocados no ficheiro `.bbl` resultante depende do estilo usado no comando `\bibliographystyle`.

3. Compila-se novamente o ficheiro `.tex` de forma a gerar o documento no formato final mas já com as referências bibliográficas corretamente formatadas e incluídas.

```
$ pdflatex doc
```

Estilos de bibliografia

Há inúmeros estilos de formatação da bibliografia e das referências colocadas ao longo do texto, a maior parte das quais fornecidas através de pacotes. O BibTeX possui apenas 4 estilos base:

plain: as referências são ordenadas alfabeticamente pelo último nome do primeiro autor e são identificadas por um número sequencial começando em 1.

unsrt: as referências são ordenadas pela ordem em que são pela primeira vez referenciadas no texto e são identificadas por um número sequencial começando em 1.

abbrv: similar ao **plain** mas as referências possuem os nomes e alguns outros campos abreviados.

alpha: similar ao **plain** mas as referências são identificadas por um acrónimo formado pelas primeiras letras do último nome de alguns dos primeiros autores e pelos dois algarismos menos significativos do ano de publicação.

Se considerarmos o seguinte trecho de um documento L^AT_EX:

O `\LaTeX`~\cite{Lamport.94} é um sistema criado por Leslie Lamport e há inúmeros manuais detalhados sobre o mesmo (v.g.~\cite{Kopka.03}).

este texto e a bibliografia produzida tendo em conta estes 4 estilos apresentados e os registos da Figura 3.4 estão patentes na Figura 3.5.

plain O L ^A T _E X [1] é um sistema criado por Leslie Lamport e há inúmeros manuais detalhados sobre o mesmo (v.g. [2]). Bibliografia	unsrt O L ^A T _E X [1] é um sistema criado por Leslie Lamport e há inúmeros manuais detalhados sobre o mesmo (v.g. [2]). Bibliografia
[1] Helmut Kopka and Patrick W. Daly. <i>Guide to LaTeX</i> . Addison-Wesley Professional, 4th edition, 2003. ISBN-10: 0321173856, ISBN-13: 978-0321173850. [2] Leslie Lamport. <i>LaTeX: A Document Preparation System</i> . Addison-Wesley Professional, 2nd edition, 1994. ISBN-10: 0201529831, ISBN-13: 978-0201529838.	[1] Leslie Lamport. <i>LaTeX: A Document Preparation System</i> . Addison-Wesley Professional, 2nd edition, 1994. ISBN-10: 0201529831, ISBN-13: 978-0201529838. [2] Helmut Kopka and Patrick W. Daly. <i>Guide to LaTeX</i> . Addison-Wesley Professional, 4th edition, 2003. ISBN-10: 0321173856, ISBN-13: 978-0321173850.

abrv O L ^A T _E X [1] é um sistema criado por Leslie Lamport e há inúmeros manuais detalhados sobre o mesmo (v.g. [2]). Bibliografia	alpha O L ^A T _E X [Lam94] é um sistema criado por Leslie Lamport e há inúmeros manuais detalhados sobre o mesmo (v.g. [KD03]). Bibliografia
[1] H. Kopka and P. W. Daly. <i>Guide to LaTeX</i> . Addison-Wesley Professional, 4th edition, 2003. ISBN-10: 0321173856, ISBN-13: 978-0321173850. [2] L. Lamport. <i>LaTeX: A Document Preparation System</i> . Addison-Wesley Professional, 2nd edition, 1994. ISBN-10: 0201529831, ISBN-13: 978-0201529838.	[KD03] Helmut Kopka and Patrick W. Daly. <i>Guide to LaTeX</i> . Addison-Wesley Professional, 4th edition, 2003. ISBN-10: 0321173856, ISBN-13: 978-0321173850. [Lam94] Leslie Lamport. <i>LaTeX: A Document Preparation System</i> . Addison-Wesley Professional, 2nd edition, 1994. ISBN-10: 0201529831, ISBN-13: 978-0201529838.

Figura 3.5: Exemplo do aspetto final da bibliografia consoante o estilo usado. A etiqueta de cada registo é exatamente igual ao que aparecerá ao longo do texto, no local onde for feita a respetiva referência.

Exercício 3.13

Edite um ficheiro com registos bibliográficos, coloque lá um de cada tipo com um número mínimo de atributos (autor, título e ano chegam, pode inventar). Inclua no seu documento L^AT_EX os comandos de inclusão de bibliografia e escolha um estilo. Inclua referência no seu texto ao registo bibliográfico que criou e gere as respetivas referências com o comando BibTeX. Observe as mensagens de erro (atributos em falta) reportados pelo BibTeX e atribua-os. Proceda até conseguir ter um documento que cite todas registas bibliográficos que criou. Altere o estilo de formatação das citações e observe o resultado.

3.12 Visão global da geração de documentos L^AT_EX

Para terminar este documento vamos dar uma visão global do modo como o L^AT_EX deve ser usado para produzir um documento complexo. Esta explicação será ilustrada pela Secção 3.12. Os ficheiros fonte estão a azul, o ficheiro alvo está a preto, os ficheiros verdes são ficheiros temporários, criados automaticamente, que são fundamentais para o processo de geração do documento alvo mas que podem em qualquer altura ser apagados. As setas a vermelho apontam para resultados de comandos realizados usando como parâmetro o ficheiro no início da seta. As setas tracejadas a azul representam inclusões de ficheiros por referência a partir de outros ficheiros, sendo indicado o comando responsável por essa inclusão.

O texto fonte de um documento pode estar repartido por vários documentos fonte, todos com a extensão `.tex`. Na Secção 3.12 são usados 4, um principal (`doc.tex`) e 3 outros incluídos por este usando o comando `\input`.

Para além deste ficheiros, o texto fonte de um documento existe também ao nível dos ficheiros de registos bibliográficos usados para retirar referências usadas no documento. Na Secção 3.12 são usados 3, os quais são associados ao documento através do comando `\bibliography`.

Resumindo, o documento possui 7 ficheiros fonte, 4 com conteúdos indiscriminados e 3 com registos BibTeX.

A compilação do documento faz-se com o comando `pdflatex` passando como parâmetro o nome base do ficheiro principal do documento (`doc.tex` ou `doc`, a extensão pode ser omitida). Daqui resulta um documento PDF (`doc.pdf`), que é o objetivo pretendido, e mais uma série de ficheiros que são um subproduto da compilação.

Alguns destes ficheiros são meramente informativos, como o `doc.log`, que possui um registo das atividades executadas pelo comando, ficheiros usados, erros identificados, etc.

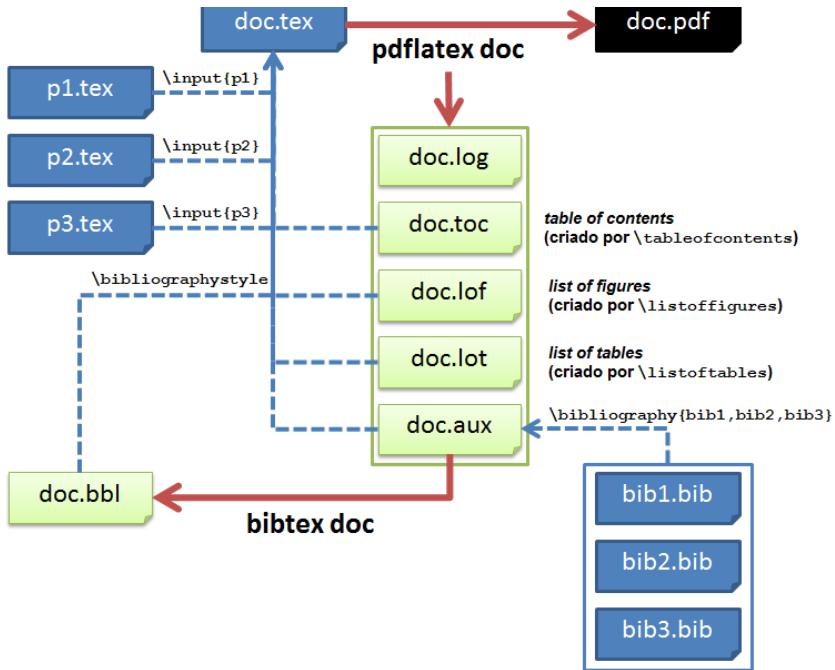


Figura 3.6: Ficheiros e comandos envolvidos na geração de um documento PDF a partir de fontes em L^AT_EX e BibTeX

Outros são ficheiros que deverão ser usados da próxima vez que o mesmo comando for executado e que possuem conteúdos relacionados com referências. É o caso do índice que fica guardado no ficheiro `doc.toc` para ser incluído na próxima compilação. O mesmo se passa com o índice de figuras (`doc.lof`) e o índice de tabelas (`doc.lot`).

Finalmente, no ficheiro `doc.aux` são registadas todas as atribuições de números a etiquetas, bem como o nome de ficheiros fonte de registo BibTeX. Este ficheiro é usado pelo `pdflatex` como uma fonte de informação auxiliar para obter identificadores associados a etiquetas. É ainda usado pelo comando `bibtex` para saber quais são os ficheiros de registo bibliográficos que devem ser considerados para o ficheiro `doc.tex`.

O comando `bibtex` gera um ficheiro de referências bibliográficas (`doc.bbl`), que contém todas as referências identificadas pelo `pdflatex` quando compilou `doc.tex`, e cujas etiquetas ficaram registadas em `doc.aux`. Estas referências serão incluídas na próxima compilação de `doc.tex`.

É claro, pelo fluxo de dados circular descrito e apresentado na Secção 3.12, que para se gerar um documento PDF a partir de fontes em L^AT_EX e BibTeX é necessário realizar ciclos de compilação com várias ferramentas até obter um resultado final completamente correto. O comando `pdflatex` dá uma ajuda neste sentido porque indica, após uma compilação, se é necessário ou não realizar mais alguma compilação, bem como se há etiquetas referenciadas mas desconhecidas. Estas indicações ficam igualmente registadas no ficheiro `doc.log`.

3.13 Para aprofundar

Exercício 3.14

O L^AT_EX possui muitas outras funcionalidades não exploradas aqui. Um bom exemplo é a capacidade de desenhar gráficos sem necessidade de recorrer à importação de ficheiros.

Explore a package *tikz* e verifique como pode construir imagens directamente em L^AT_EX.
Consulte o endereço <http://www.texample.net/tikz/examples>.

Redes de Comunicações

Objetivos:

- Conceito de endereço IP
- Máscaras
- Rotas
- Configuração de rede em Linux
- Serviços de Rede
- Acesso Remoto

4.1 Introdução

Os sistemas com capacidades de comunicação em rede possuem uma variedade de identificadores que possibilitam a troca de informação. Estes identificadores funcionam como a morada numa casa, quando se pretende trocar correspondência, ou o número de telefone quando se pretende falar com um amigo. Em ambos os casos existem identificadores que permitem que a informação chegue ao seu destino, e se identifique a origem.

Neste trabalho iremos explorar como estes identificadores estão relacionados e qual a sua utilidade para a comunicação na Internet. Utilizaremos máquinas virtuais para simular uma rede local e explorar comandos de monitorização e configuração de redes.

Importante: Neste guião, recorremos a diversos comandos e ficheiros UNIX. Alguns são mesmo específicos de certas distribuições de Linux, nomeadamente das derivadas da distribuição Debian. Também iremos manipular certas configurações e fazer diversas operações que requerem permissões de administrador. Por isso, recomenda-se que faça os exercícios numa máquina virtual criada propositadamente com um sistema operativo

Debian, Ubuntu ou um derivado desses. Para facilitar, fornecemos uma imagem comprimida de um disco virtual já preparado com um sistema Lubuntu 16.04 de 32bits.

Exercício 4.1

Se não tem já a máquina virtual da disciplina instalada, descarregue e descomprima o ficheiro do disco virtual fornecido.

Crie uma máquina virtual, indicando as opções:

Nome: `labi-debian` (ou outro qualquer).

Tipo de sistema operativo: Linux.

Versão de sistema operativo: Debian (64bit).

Memória: 512 MB.

Disco rígido: “Use an existing virtual hard drive file” e escolha a imagem que descarregou. Pode igualmente ativar a memória *cache* do controlador SATA de forma a acelerar o acesso ao disco.

A seguir, configure a máquina para ter duas interfaces de rede. A primeira do tipo *NAT* [4], que servirá para comunicação com a Internet; e a outra do tipo *Internal Network*, que servirá para comunicação com outras máquinas virtuais a correr no mesmo hospedeiro.

Os exercícios abaixo deverão ser feitos na máquina que acabou de criar, a não ser que indiquem explicitamente outro procedimento.

4.2 Configuração de rede de um computador

A configuração de rede de um computador envolve diversos componentes, podendo ser feita de forma mais ou menos automatizada. Neste guião será utilizada a forma manual, que é a tipicamente presente em servidores e outros dispositivos. Em várias distribuições é possível utilizar o serviço `NetworkManager` para configurar os mesmos parâmetros.

Os mais importantes são: endereços, interfaces, encaminhamento e serviço de resolução de nomes. De seguida iremos abordar a configuração de alguns parâmetros.

Exercício 4.2

Abra um terminal, execute o comando **ifconfig** e verifique:

1. Quantas interfaces de rede existem;
2. O endereço Internet Protocol v4 (IPv4)[5] de cada interface;
3. O(s) endereço(s) Internet Protocol v6 (IPv6)[6] de cada interface;
4. O endereço físico (Media Access Control (MAC)[7]) de cada interface;
5. A máscara de rede de cada interface;
6. Relacione o número de interfaces reportados no *Linux*, com o número reportado pelo *VirtualBox*.

(Pode experimentar o mesmo comando no sistema hospedeiro, se for Linux ou outro UNIX.)

Além de mostrar a configuração das interfaces, o comando **ifconfig** também pode ser usado (pelo super administrador) para definir ou alterar essa configuração. No entanto, é mais fácil e mais usual recorrer a programas gestores de interfaces de rede, que lêem as configurações guardadas em ficheiros do sistema e aplicam-nas da mesma forma que o **ifconfig**.

O gestor de interfaces de rede nativo em sistemas Debian é o conjunto de programas a que se chama coletivamente **ifupdown**. Este gestor mantém as configurações no ficheiro **/etc/network/interfaces**.²²

Este ficheiro permite especificar se as interfaces devem ser configuradas usando algum método dinâmico (ex, Dynamic Host Configuration Protocol (DHCP)[8]), ou através de uma configuração estática. Por exemplo, a configuração seguinte define que a interface **eth2** será configurada dinamicamente, enquanto a **eth3** será configurada estaticamente:

```
auto eth2
iface eth2 inet dhcp

auto eth3
```

²²Pode consultar **man 5 interfaces** para detalhes da sua sintaxe.

```
iface eth3 inet static
    address 192.168.0.1
    netmask 255.255.255.0
    gateway 192.168.0.254
```

Repare que no caso da interface **eth3**, é necessário fornecer todos os parâmetros essenciais à sua operação.

Exercício 4.3

Edite o ficheiro **/etc/network/interfaces** e especifique uma configuração dinâmica da primeira interface de rede (*NAT*) e uma configuração estática da segunda interface de rede (*Internal Network*). Nesta última, defina uma rede 192.168.56.0/24 e sem gateway.

Pode aplicar a configuração através do comando **ifup nome-da-interface**. O comando **ifup -a** aplica as configurações a todas as interfaces indicadas com **auto** no ficheiro **interfaces**. (Este comando é usado num dos scripts de arranque do sistema operativo.) Use o comando **ifconfig nome-da-interface** ou **ifconfig** para verificar o estado atual de uma ou de todas as interfaces. O comando **ifdown nome-da-interface** permite desactivar a configuração.

Pode necessitar desativar as interfaces através de **ifdown** antes que o comando **ifup** surja efeito.

(Necessita de permissões de administrador (usando **su**) para os comandos neste exercício.)

Também é comum haver outro gestor de interfaces de rede que corre no ambiente gráfico e que se configura através da interface gráfica. No caso do *Debian*, temos o gestor **Wicd**, que se pode configurar no menu principal, acedendo a *Internet->Wicd Network Manager*.

Exercício 4.4

Utilize a interface gráfica para aplicar uma configuração que define todas as interfaces com endereços obtidos dinamicamente (DHCP). Pode aceder à configuração através do menu *Iniciar->Internet->Wicd Network Manager*

4.3 Tabela de Endereços Físicos

Os dispositivos com capacidade de comunicação possuem endereços únicos que os identificam. O sistema operativo mantém uma tabela onde regista informação sobre as estações vizinhas conhecidas. Em *Linux* é possível a um administrador do sistema listar as entradas desta tabela executando o comando **arp -an**. A Figura 4.1 demonstra a tabela que pode encontrar.

```
root@linux:~# arp -an
? (10.0.2.2) at 52:54:00:12:35:02 [ether] at eth0 brd ff:ff:ff:ff:ff:ff
? (192.168.56.100) at 08:00:27:fe:45:4e [ether] at eth1 brd ff:ff:ff:ff:ff:ff
root@linux:~#
```

Figura 4.1: Resultado do comando **arp -an**.

Exercício 4.5

Execute o comando **arp -an**. Verifique se o endereço IPv4 de um computador ao seu lado está presente na tabela (terá de o perguntar ao colega que está nesse computador).

Repita o comando num computador físico da sala, ou no hospedeiro e compare resultado.

Exercício 4.6

De seguida, execute um **ping endereço-de-destino** para o endereço do PC ao seu lado e volte a observar o conteúdo da tabela. Este comando também funciona dentro da máquina virtual?

4.4 Tradução de nomes em endereços IP

Os nomes que utilizamos para aceder a conteúdos HyperText Transfer Protocol (HTTP)[9] não são os utilizados para as comunicações. Na realidade o endereço IPv4 (versão 4 ou versão 6) é que é utilizado a quando do estabelecimento de uma ligação. O Domain Name System (DNS)[10] é um serviço que permite traduzir nomes (ex. **www.ua.pt**) em endereços IPv4 e vice versa.

Exemplos de alguns nomes:

```
www.ua.pt
www.up.pt
www.sapo.pt
www.antena3.pt
www.fcporto.pt
www.scp.pt
www.sporting.pt
www.slbenfica.pt
www.google.com
www.google.pt
www.facebook.com
```

Exercício 4.7

A configuração de DNS de um sistema *Linux* encontra-se em **/etc/resolv.conf**. Visualize o conteúdo deste ficheiro e registe qual o servidor de DNS que está a utilizar.

Compare este valor com o presente no anfitrião da máquina virtual.

Exercício 4.8

Utilize o comando **host** (ex., **host www.ua.pt**) para obter os endereços associados a cada um dos nomes anteriormente listados (resolução direta). Seja curioso. Procure e registe endereços repetidos, múltiplos endereços ou outras situações que considere anómalas.

Exercício 4.9

Da mesma forma que é possível traduzir nomes em endereços, também é possível realizar a operação inversa. Utilizando o mesmo comando (ex., **host 193.136.92.123**), verifique qual a correspondência inversa (de endereço para nome). Procure identificar se a resolução direta e inversa produzem resultados compatíveis.

4.5 Conectividade e rotas

Até agora sabemos que é possível comunicar usando o endereço IPv4 do servidor. Também já abordámos o serviço que permite converter nomes em endereços IPv4. Resta saber como a informação atravessa a Internet. O segredo está no conceito de *rota de encaminhamento*. O comando **route** permite listar (e modificar) as rotas de encaminhamento.

Exercício 4.10

De forma a determinar as rotas existentes, execute **route -n**. Verifique qual a rota por omissão (*default*) do sistema. Que outras rotas tem?
(Para perceber a tabela, consulte **man 8 route**, secção OUTPUT.)

Existem dois comandos particularmente relevantes no domínio do diagnóstico do estado das redes e das suas rotas: **ping** e **traceroute**. O primeiro (**ping**) permite enviar um pacote especialmente construído que instrui o destinatário a responder. Pode ser utilizado para determinar a existência de conectividade e mesmo o atraso nas comunicações. O segundo comando (**traceroute**), mais complexo, permite identificar a rota utilizada para comunicar com o destino.

Exercício 4.11

Execute o comando **ping** para cada um dos destinos da tabela abaixo e registe o tempo médio de comunicação. Pode também verificar que algumas ligações apresentam ocasionalmente perdas de pacotes. Detecta uma correlação entre tempo, perdas e distância?

Enviando pacotes especialmente construídos e processando as notificações enviadas de volta por cada *Router*, é possível identificar os dispositivos numa rota. O programa **traceroute** implementa este mecanismo de sinalização. A Figura 4.2 mostra a rota que existe entre servidores na Universidade de Aveiro e os servidores de www.google.pt. Para cada uma das entradas é mostrado o nome, endereço IPv4 e o tempo médio de resposta.

Devido às regras de segurança aplicadas na Universidade de Aveiro, não é possível utilizar o programa **traceroute** dentro da rede de clientes da universidade (ex, **eduroam**).

Durante a aula recomenda-se utilizar um serviço Web que permite executar o comando

Nome	Localização
www.ua.pt	Aveiro, Portugal
www.ipp.pt	Porto, Portugal
www.utl.pt	Lisboa, Portugal
www.utad.pt	Vila Real, Portugal
www.uevora.pt	Évora, Portugal
www.uam.es	Madrid, Espanha
www.univ-paris8.fr	Paris, França
www.cmu.edu	Pittsburgh, EUA
www.bjut.edu.cn	Pequim, China
www.u-tokyo.ac.jp	Tóquio, Japão
www.adelaide.edu.au	Adelaide, Austrália
www.poea.gov.ph	Filipinas

```
traceroute to www.google.pt (173.194.45.23), 30 hops max, 60 byte packets
 1  193.137.173.209 (193.137.173.209)  0.389 ms  0.665 ms  0.746 ms
 2  10.0.34.1 (10.0.34.1)  0.609 ms  0.607 ms  0.713 ms
 3  Router2.Campanha.fccn.pt (193.136.4.26)  0.930 ms  0.965 ms  0.954 ms
 4  Router3.10GE.DWDM.Lisboa.fccn.pt (193.136.1.1)  5.621 ms  5.686 ms *
 5  ROUTER10.10GE.CR1.Lisboa.fccn.pt (193.137.0.8)  5.480 ms  5.474 ms  5.458 ms
 6  Google.RS15169.gigapix.pt (193.136.250.20)  5.611 ms  5.616 ms  5.617 ms
 7  209.85.254.70 (209.85.254.70)  6.435 ms  6.503 ms  6.494 ms
 8  lis01s06-in-f23.1e100.net (173.194.45.23)  5.645 ms  5.640 ms  5.653 ms
```

Figura 4.2: Resultado do comando **traceroute www.google.pt** executado a partir de um servidor na Universidade de Aveiro.

traceroute desde um servidor remoto em Portugal. Naturalmente, a origem dos pacotes não será Aveiro e o resultado será ligeiramente diferente ao que se obteria dentro da Universidade de Aveiro. Para aceder ao serviço, utilize o navegador que tem instalado e insira o endereço: <http://toolbox.3gnt.net/network/>.

Preencha o endereço de destino pretendido, selecione a ferramenta (**traceroute** ou **lft**) e inicie o teste. Ao fim de alguns segundos, aparece o resultado. Pode experimentar vários endereços através desta interface.

Se estiver a realizar este guia fora da Universidade de Aveiro, poderá utilizar o comando

`traceroute endereço` directamente a partir da linha de comandos da máquina virtual.

Exercício 4.12

Para cada um dos endereços anteriormente analisados, obtenha a rota desde o ponto de origem até ao destino. De seguida, analise a rota obtida, identifique e registe:

1. O número de routers na rota.
2. Alguns países por onde o tráfego foi encaminhado.
3. O Router com maior atraso.

4.6 Identificação da entidade responsável por uma máquina

Todos os equipamentos possuem uma entidade responsável e esta entidade tem de estar devidamente identificada perante os restantes utilizadores da Internet. Bases de dados disponíveis *online* como, por exemplo, <http://www.whois.sc> permitem consultar esta informação. Se estiver fora da Universidade de Aveiro, pode também consultar esta informação através do comando `whois`.

Exercício 4.13

Para cada um dos nomes, registe o nome do titular do registo.

Exercício 4.14

Considerando as rotas obtidas anteriormente, e utilizando o serviço de *Whois* registe qual a entidade responsável (Organization), pelo acesso à Internet de cada um dos destinos.

4.7 Transmissão de informação em redes: ping

Até agora tem-se referido que as redes atuais são orientadas à comunicação por pacotes, não tendo sido no entanto observados estes elementos de comunicação. Neste ponto iremos observar o que realmente acontece quando se executa o comando `ping www.google.pt`.

Para isso é necessário utilizar uma aplicação que permite escutar todo o tráfego enviado

para a rede: *Wireshark*.

De forma a capturar tráfego, execute o *Wireshark*, aceda às opções e defina que quer escutar pacotes na interface de rede *NAT*, que configurou no *VirtualBox*.

Responda às questões:

- Quais os endereços IPv4 e MAC envolvidos nas comunicações?
- Que protocolos são utilizados em cada comunicação?
- Consegue identificar o endereço do servidor de DNS?
- Com o comando **ping** consegue saber a que pedido corresponde uma resposta?

Exercício 4.15

Repete o comando **ping** para vários endereços. Consegue explicar o funcionamento do comando?

4.8 Transmissão de informação em redes: conteúdo HTTP

O protocolo HTTP é um protocolo de nível aplicacional, muito utilizado para a transferência de informação na Internet. Sempre que acede ao *Google* ou ao *Facebook* está a utilizar este protocolo. Visto ser um protocolo aplicacional, funciona em cima de um outro protocolo chamado Transmission Control Protocol (TCP)[11]. O protocolo TCP permite que vários serviços o utilizem, criando a noção de portas. Cada comunicação usa uma porta diferente e assim é possível comunicar. Isto será abordado nos parágrafos seguintes.

Para transferir a informação, o protocolo HTTP baseia-se no princípio do pedido (*request*) e resposta (*response*). Quando insere um Uniform Resource Locator (URL)[12] no browser, é enviado um pedido, ao qual o servior responde com o conteúdo pretendido.

O exemplo que se segue é um destes pedidos. Neste caso, requisita-se a página / ao servidor **www.google.pt**. Como pode verificar, o cliente identifica-se (**User-Agent**) e define que tipo de conteúdo aceita (**Accept**).

```
GET / HTTP/1.1
Host: www.google.pt
User-Agent: Mozilla/5.0
Accept: text/html
```

Exercício 4.16

O comando **telnet endereço-do-servidor porta** permite efetuar uma ligação TCP, sobre a qual se pode transmitir informação. Para verificar a simplicidade do protocolo HTTP, efectue uma ligação ao servidor indicado na porta 80 (ex, **telnet www.google.pt 80**) e envie o pedido mostrado acima. Deverá aparecer muito texto.

Utilize o *Wireshark* e verifique o que realmente acontece.

Exercício 4.17

Utilizando o *browser*, repita o processo para qualquer outro *site*, e analise o resultado com o *Wireshark*.

Consegue identificar os endereços IP e os protocolos utilizados?

Relativamente ao protocolo HTTP, consegue identificar a versão do protocolo, o cliente utilizado, o servidor e o caminho que compõem o URL pedido?

4.9 Acesso a servidores remotos

SSH

Como vimos, é possível realizar ligações a sistemas e trabalhar nestes, como se de um sistema local se tratasse. No passado, o método mais utilizado era o protocolo **telnet**, que ainda se pode encontrar em alguns equipamentos mais simples. Atualmente, um dos métodos mais utilizados é o **ssh**, acrônimo de Secure Shell.

O **ssh** possibilita aceder a uma *shell* num sistema remoto, sobre a qual é possível executar comandos. Ao contrário do **telnet**, todas as trocas de informação através do **ssh** são seguras. As comunicações são cifradas de forma que não poderão ser percebidas por um atacante que as intercepte. Existem algumas outras funcionalidades que levaram a que o **ssh** se tornasse mais popular:

- Suporta níveis de segurança configuráveis.

- Suporta a transferência de ficheiros.
- Suporta a criação de túneis de tráfego (como uma Virtual Private Network (VPN)[13]).
- Suporta a execução de aplicações gráficas remotas.

Para iniciar uma ligação **ssh** basta executar **ssh username@servidor** ou alternativamente, **ssh -l username servidor**.

No caso desta disciplina, o servidor mais utilizado será **xcoa.av.it.pt** enquanto o username terá o formato **labi-tXgY**.

Portanto, ao executar na consola: **ssh labi-tXgY@xcoa.av.it.pt**, o grupo Y da turma X estará a iniciar uma sessão remota no servidor **xcoa.av.it.pt**. Use o seu número de turma e grupo que pode ser fornecido pelo docente.

Como mecanismo de segurança, o **ssh** cria uma impressão digital dos servidores. Isto permite que os utilizadores tenham a certeza que se estão a ligar ao servidor certo.²³ No caso do servidor atual, e na primeira ligação, será apresentada a seguinte mensagem:

```
The authenticity of host 'xcoa.av.it.pt (193.136.92.147)' can't be established.
ECDSA key fingerprint is SHA256:Se2g3o+sVC1Y+zPOSNLBP/L5vCfIjo9W+08spExPXbg.
Are you sure you want to continue connecting (yes/no)?
```

Repare que a impressão digital (*fingerprint*) do servidor tem o valor **SHA256:Se2g3o+sVC1Y+zPOSNLBP/L5vCfIjo9W+08spExPXbg**.

Caso este valor seja apresentado, o utilizador sabe que está a ligar-se ao servidor correto. Caso seja diferente, deverá interromper imediatamente a tentativa de ligação. Num cenário real, um utilizador pode verificar o valor junto do administrador do sistema.

As impressões digitais dos servidores previamente acedidos são armazenadas no ficheiro **~/.ssh/known_hosts**. Numa ligação posterior, o **ssh** confirma a impressão digital e não requer intervenção do utilizador. Caso a impressão digital guardada seja diferente da do servidor, é mostrada a mensagem que se segue:

```
oooooooooooooooooooooooooooooooooooooooooooo
©      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      ©
oooooooooooooooooooooooooooooooooooooooooooo
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

²³Note que é possível desviar as comunicações que passam na Internet, tal como um carteiro poderia desviar cartas se assim o entendesse.

```
Someone could be eavesdropping on you right now (man-in-the-middle attack)!  
It is also possible that a host key has just been changed.  
The fingerprint for the ECDSA key sent by the remote host is  
SHA256:Se2g3o+sVC1Y+zPOSNLBP/L5vCfIjo9W+08spExXbg.  
Please contact your system administrator.  
Add correct host key in /home/debian/.ssh/known_hosts to get rid of this message.  
Offending ECDSA key in /home/debian/.ssh/known_hosts:1  
ECDSA host key for xcoa.av.it.pt has changed and you have requested strict checking.  
Host key verification failed.
```

Depois da sessão se encontrar estabelecida, todos os comandos introduzidos são executados no servidor remoto, sendo o seu resultado enviado de volta para o cliente local. Pode verificar que utilizadores se encontram ligados executando o comando **who**.

Exercício 4.18

Remova o ficheiro `~/.ssh/known_hosts` do seu sistema (local). De seguida, efetue uma nova ligação ao servidor `xcoa.av.it.pt` e verifique que a impressão digital é a apresentada neste guião. Termine a ligação e volte a estabelecer-a. É apresentada alguma mensagem adicional?

Verifique agora o conteúdo do ficheiro `~/.ssh/known_hosts` e localize a entrada relacionada com o servidor `xcoa.av.it.pt`.

Transferência de ficheiros

O **ssh** não permite apenas executar comandos remotos, permite igualmente transferir ficheiros entre sistemas. É possível transferir ficheiros e diretórios de e para servidores remotos, ou entre servidores remotos.

Para transferir um ficheiro utilizando **ssh**, invoca-se o comando **scp** (*secure copy*). A sintaxe do comando **scp** é a seguinte:

```
scp origem destino
```

A **origem** e o **destino** podem ser simplesmente nomes de ficheiros e/ou diretórios locais e, nesse caso, o comportamento é idêntico ao do comando **cp**. Porém, se a **origem** e/ou o **destino** tiverem o formato **utilizador@servidor:ficheiro**, então indicam um determinado ficheiro (ou diretório) de um certo servidor, acedido pelo utilizador indicado. O exemplo seguinte copia um ficheiro `teste.txt`, que se encontra na área pessoal do utilizador atual, para a área pessoal do utilizador chamado `user` no servidor `xcoa.av.it.pt`. Num cenário real, terão de se utilizar utilizadores e caminho adequados.

```
scp ~/teste.txt user@xcoa.av.it.pt:/home/user
```

Para copiar o ficheiro de volta, desta vez para o diretório atual, poderia executar-se:

```
scp user@xcoa.av.it.pt:/home/user/teste.txt .
```

Também é possível copiar o ficheiro entre sistemas remotos:

```
scp user1@xcoa.av.it.pt:/home/user/teste.txt user2@xcoa.av.it.pt:
```

Exercício 4.19

Usando o *browser*, descarregue uma imagem para o seu computador, e utilizando o **scp**, copie-a para o servidor *xcoa.av.it.pt*.

Autenticação por chaves

Até agora a autenticação do protocolo **ssh** junto de servidores remotos tem sido efetuada através de um nome de utilizador e de uma senha. Quando se estabelecem ligações a servidores frequentemente, o facto de se introduzirem constantemente estes dados torna-se problemático. Além disso, a utilização deste par de elementos (utilizador e senha), não é a forma mais segura de autenticação. Entre outros problemas de segurança, o uso de senhas curtas ou baseadas em palavras de dicionário pode comprometer o sistema.

O **ssh** permite a utilização de um par de chaves que irá substituir a senha. Estas chaves são constituídas por duas partes (2 ficheiros). Uma é pública e deve ser colocada nos servidores a que pretendemos ligar, a outra nunca deve ser fornecida a terceiros e fica no computador local.²⁴

O primeiro passo a fazer é a criação das chaves para autenticação. A chave local deve estar cifrada com uma senha. Esta senha nunca é enviada para o servidor, serve apenas para decifrar o ficheiro da chave privada **id_rsa**. Isto consegue-se executando o comando **ssh-keygen** e seguindo as instruções fornecidas:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/debian/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/debian/.ssh/id_rsa.
```

²⁴O conceito de chaves públicas e privadas está fora do âmbito desta disciplina. Se tiver curiosidade pode consultar a página http://pt.wikipedia.org/wiki/Criptografia_de_chave_p%C3%BCblica

```
Your public key has been saved in /home/debian/.ssh/id_rsa.pub.  
The key fingerprint is:  
cd:78:2d:63:12:aa:02:89:22:de:89:4d:cd:3d:8e:73 labi@labi
```

A partir deste momento estarão criados dois ficheiros com as duas chaves:

```
/home/debian/.ssh/id_rsa.pub Chave PÚblica (a colocar no servidor).  
/home/debian/.ssh/id_rsa Chave Privada (a manter localmente).
```

A chave pública deve ser adicionada ao ficheiro `~/.ssh/authorized_keys` do servidor remoto para que o `ssh` passe a utilizar as chaves criadas em vez do método tradicional para autenticação. Pode acrescentar a chave usando comandos ou um editor no servidor, ou usando o comando `ssh-copy-id` a partir do sistema cliente.

Exercício 4.20

Crie um par de chaves no seu computador sem especificar uma palavra passe. Instale a chave pública na sua conta do servidor `xcoa.av.it.pt` e verifique o que acontece quanto volta a estabelecer uma sessão com o servidor.

Volte a criar e instalar um par de chaves mas especificando uma senha. Volte a estabelecer uma sessão com o servidor e verifique o que acontece.

Pode obter mais informação sobre o processo de autenticação se executar `ssh -v` em vez de `ssh`.

Reencaminhamento do protocolo X11

Também é possível executar aplicações gráficas através de `ssh`. Em *Linux*, a interface gráfica é um serviço que recebe pedidos das aplicações clientes (ex., desenhar um botão, apresentar uma imagem) e envia-lhes eventos (ex., tecla pressionada, nova posição do rato, etc). Como a comunicação dos pedidos e dos eventos é feita através de mensagens, usando o protocolo X11,²⁵ é perfeitamente possível que a aplicação e a interface gráfica se encontrem em sistemas distintos.

Utilizando `ssh`, podemos disponibilizar o servidor de X11 local como terminal gráfico para as aplicações remotas.²⁶ Para isto é necessário executar o `ssh` da seguinte forma:

```
ssh -X user@servidor
```

²⁵Para mais informação, consultar http://pt.wikipedia.org/wiki/X_Window_System.

²⁶Note que, nesta situação, o `ssh` é um *cliente* local do *servidor* remoto de aplicações, mas essas aplicações remotas são clientes do servidor de X11 que corre no sistema local.

Depois, poderá executar qualquer aplicação gráfica e não apenas comandos de texto nas sessões remotas. Se se pretender iniciar uma sessão sem suporte de reencaminhamento de X11, o `ssh` poderá ser executado com a opção `-x` (minúsculo).

Exercício 4.21

Efetue uma ligação ao servidor `xcoa.av.it.pt` com a opção `-x` e interprete o resultado do comando `midori`.

Volte a repetir a sessão mas desta vez utilizando a opção `-X`. Compare o resultado que obtém ao executar o comando `midori`.

Utilizando o `midori` (no servidor remoto) e o `firefox` (no computador local), aceda ao URL `http://labi2.aws.atnog.av.it.pt/ip/` e compare o resultado. Pode igualmente aceder a outras páginas como `http://my.ua.pt` ou `http://www.sapo.pt` e verificar que em ambos os casos existe conectividade à Internet.

4.10 Para aprofundar o tema

Exercício 4.22

Utilizando o wireshark, capture tráfego e identifique todos os pedidos efectuados. Pode utilizar o filtro `http.request` depois de capturar os pacotes, se quiser visualizar apenas os pedidos de HTTP.

Exercício 4.23

Execute o comando `traceroute` e, através da aplicação Wireshark, verifique que pacotes são enviados. Tente encontrar a função do campo TTL e como este é utilizado.

Exercício 4.24

Execute o comando `pftp glua.ua.pt` e utilize o utilizador `ftp` sem palavra passe. Capture o tráfego e verifique que dados consegue visualizar na captura.

Caso necessite de instalar este comando, pode fazê-lo através de: `apt install ftp`

Exercício 4.25

É comum nomear os sistemas com personagens e locais de livros, séries, filmes, sagas ou outras obras. Sabendo que os sistemas centrais da Universidade de Aveiro estão na rede **193.136.173.0/24** e recorrendo ao comando **host** ou **dig -x**, resolva vários endereços IPv4 e determine qual(ais) as obras que são utilizadas para nomear alguns sistemas da universidade.

Poderá ter de instalar o pacote **dnsutils**.

Colaboração em Projetos

Objetivos:

- Sistemas de colaboração em projetos.
- Introdução aos sistemas de controlo de versões.

5.1 Introdução

Quando diversas entidades colaboram num projeto comum é necessário que existam meios de coordenar as atividades. Tal não se faz através de plataformas genéricas como o *Facebook* ou o *Google+*. Por exemplo, é necessário saber se o projeto está atrasado ou não, quais são as próximas tarefas e quem está responsável por elas, ou quem realizou uma determinada tarefa. Também é necessário que existam meios de comunicação rápida entre todos, tal como uma *mailing-list*. Também pode existir necessidade de ter uma base de informação partilhada.

Esta é a razão pela qual utilizamos a plataforma <http://elearning.ua.pt>, pois possui ferramentas que facilitam o desenrolar das aulas e a colaboração entre docentes e alunos.

Quando se fala de projetos com caráter de desenvolvimento de aplicações (programação), o processo de colaboração necessita de meios ainda mais evoluídos. Isto porque os projetos de desenvolvimento têm uma complexidade acrescida ao nível da escrita do código, verificação das funcionalidades, identificação das alterações e gestão de problemas (vg. bugs²⁷). De igual forma, quando uma aplicação é desenvolvida por vários programadores, não se espera que trabalhe um de cada vez, ficando os outros à espera. Na Secção 5.2

²⁷Ver http://en.wikipedia.org/wiki/Software_bug

iremos abordar a gestão de processos de desenvolvimento através da plataforma **CodeUA**, enquanto que na Secção 5.3 iremos abordar a edição concorrencial de código.

5.2 A plataforma **CodeUA**

A plataforma **CodeUA**, que pode ser encontrada em <http://code.ua.pt>, é um sistema de gestão de projetos disponível para qualquer utilizador na Universidade de Aveiro. Foi implementado com o software livre Redmine²⁸ e inclui diversas ferramentas para facilitar a gestão de projetos, tais como: fóruns de discussão, Wikis, gestão e seguimento de tarefas, calendários, diagramas de *Gantt*²⁹ e integração com sistemas de controlo de versões. A plataforma pode ser utilizada para a gestão de qualquer tipo de projeto, mas é particularmente útil para projetos de desenvolvimento de aplicações, que mais podem beneficiar do sistema de controlo de versões de software coordenado com as ferramentas de gestão de tarefas, de funcionalidades e de pedidos de suporte. Embora o guião seja baseado na plataforma **CodeUA**, as funcionalidades descritas são análogas às de outras plataformas de gestão de projetos.

Os alunos podem usar a plataforma **CodeUA** para gerir projetos das suas disciplinas, os professores utilizam-na para criar e partilhar conteúdos das aulas, e todos a podem utilizar para divulgar trabalhos que realizem durante o seu contacto com a UA. A Figura 5.1 mostra a página inicial da plataforma **CodeUA**. Esta página inclui uma mensagem de boas vindas (lado esquerdo) e uma lista de projetos criados recentemente (lado direito). Na barra superior pode consultar a lista de projetos públicos, iniciar uma sessão ou consultar a ajuda.

Os projetos apresentados são apenas uma parte de todos os projetos. Se ainda não iniciou uma sessão, será apresentada a lista de projetos públicos. Para qualquer um destes projetos, pode verificar qual o seu propósito, quem são os seus membros, ou até contribuir para eles.

Exercício 5.1

Aceda à aplicação **CodeUA** no endereço <http://code.ua.pt> e consulte a lista de projetos públicos. Aceda a alguns e verifique a sua descrição e os seus membros.

Seja curioso! Alguns projetos poderão ser do seu interesse.

Um projeto em particular, chamado **CodeUA**, e disponível no endereço <http://code.ua.pt>.

²⁸Ver <http://www.redmine.org/>

²⁹Ver http://pt.wikipedia.org/wiki/Diagrama_de_Gantt



Figura 5.1: Página de entrada da plataforma **CodeUA**.

[pt/projects/codeua](#) serve para se trocar informação sobre a plataforma. Sempre que tiver dúvidas sobre o seu funcionamento, ou encontrar problemas, deve dirigir-se a este projeto e colocar a sua questão.

Todos os membros da *UA* podem possuir uma conta na plataforma **CodeUA**, bastando para isso identificarem-se perante o sistema. A partir desse momento passa a ser possível criar projetos e gerir o seu funcionamento.

Um aspeto importante em qualquer projeto é a gestão dos seus membros. No caso de projetos privados, apenas os seus membros terão acesso aos conteúdos do projeto. Para projetos públicos, o acesso que os membros possuem será diferenciado dos restantes (vg. convidados). Uma diferenciação normal é que os convidados apenas podem aceder a conteúdos existentes, enquanto os membros podem criar conteúdos.

De entre os membros também é possível diferenciar qual o seu papel no projeto. A plataforma **CodeUA** distingue entre vários papéis:

Manager: Gere o projeto e define como deve prosseguir.

Developer: Desenvolve conteúdos para o projeto. Num projeto de desenvolvimento de uma aplicação, estes serão os programadores.

Reporter: Relata questões ou problemas para o projeto, mas não tem como função o desenvolvimento de conteúdo. Num projeto de desenvolvimento de uma aplicação, estes serão pessoas que testam a aplicação ou que fornecem algum *feedback* sobre ela.

Exercício 5.2

Aceda à plataforma. Na lista de projetos, crie um projeto com o identificador labi2018-tXgY em que *X* representa o número da turma e *Y* representa o seu grupo. Como este projeto vai ser um projeto para gestão do grupo, defina-o como privado.

De entre os módulos, escolha apenas o módulo de *notícias*.

Pode adicionar uma descrição e definir um nome, que pode ser diferente do identificador.

Defina os membros do grupo como gestores (*Manager*) e o seu professor como *Reporter*.

Nas subsecções 5.2, 5.2 e 5.2 serão abordados alguns dos módulos mais relevantes (e geralmente utilizados) para a gestão de projetos. Existem no entanto outros que não irão ser abordados, mas que são interessantes para alguns casos. Experimente-os.

Comunicação no Projeto

A comunicação é vital nos projetos. Para isso a plataforma **CodeUA** possui o módulo de *notícias* que permite a divulgação de anúncios. Estes anúncios são enviados para todos os participantes do projeto e ficam disponíveis para consulta pelos membros.

Além disto, se o projeto for público a notícia fica disponível para todos os indivíduos que visitem o projeto. É extremamente útil para anunciar novas funcionalidades ou marcos no desenvolvimento do projeto.

As notícias não são uma simples mensagem e podem ser bastante mais ricas (ver Figura 5.2). São compostas por 3 partes: título, sumário e descrição. O título identifica claramente a notícia. O sumário é um texto curto, apresentado com a lista de notícias, que deve ajudar o leitor a decidir se a notícia lhe interessa ou não. O campo descrição permite escrever o corpo da notícia. Aqui é possível recorrer a alguma formatação na composição da notícia, bem como adicionar ficheiros e imagens. Repare na Figura 5.2 como se adiciona uma imagem. Em primeiro lugar é necessário adicionar o ficheiro e

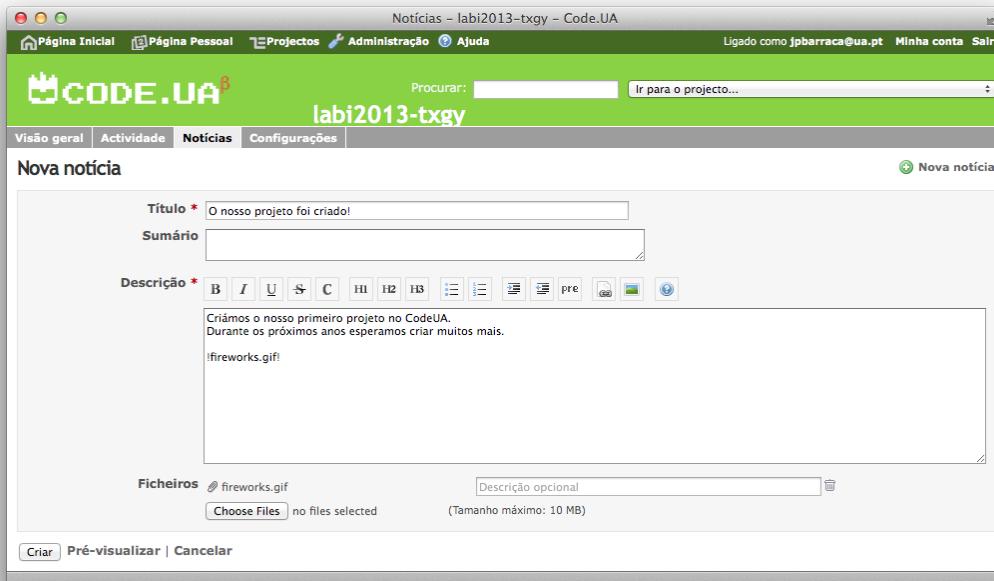


Figura 5.2: Página de introdução de uma notícia.

depois ele pode ser referido no texto entre carateres !!.

Exercício 5.3

Crie uma notícia no seu projeto anunciando a sua criação.

Dado o tema da notícia, adicione uma imagem para comemorar o evento. O resultado deverá ser semelhante ao apresentado na Figura 5.3.

Verifique se recebeu uma notificação na sua caixa de correio eletrónico da UA. Isto pode demorar alguns minutos.

Também relacionado com o módulo de notícias é o módulo de *fóruns*. Este módulo implementa um modelo em que se permite discutir assuntos de uma forma contextualizada. Isto é, é possível criar vários fóruns, um para cada sub-tema ou aspeto do projeto, onde os membros podem trocar ideias, tal como num fórum comum da Internet.

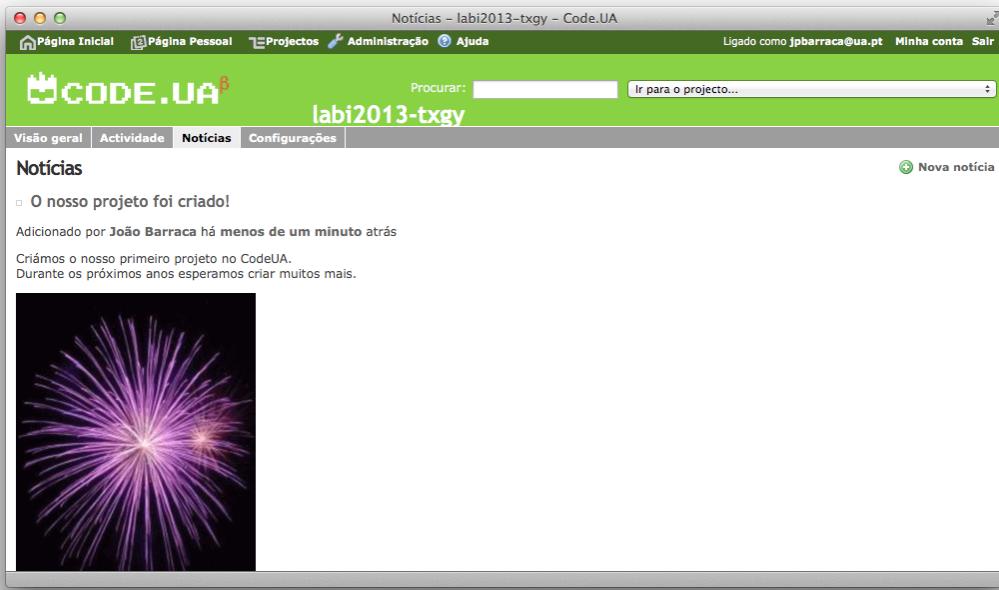


Figura 5.3: Página de apresentação das notícias.

Exercício 5.4

Crie um fórum para o primeiro trabalho de aprofundamento de conhecimentos, para discussão de aspetos deste primeiro trabalho. Experimente depois enviar uma mensagem para o fórum.

Verifique a sua caixa postal na UA.

Gestão de Tarefas

Em qualquer projeto colaborativo existem sempre tarefas. Estas podem ser de vários tipos e podem estar em diferentes estados. Podem existir tarefas que relatam problemas que existam na aplicação desenvolvida. Podem existir tarefas que identificam funcionalidades interessantes que se pretendem desenvolver. Outras podem referir-se a ações que há necessidade de realizar no futuro. Mesmo em pequenos projetos, o número de tarefas

Figura 5.4: Página de criação de uma nova tarefa.

pode ser bastante grande. Basta que para isso os membros do projeto formalizem o que é necessário realizar e os problemas encontrados na forma de tarefas pendentes.

As tarefas podem seguir um caminho desde que são criadas até que estão terminadas, necessitando de diversas interações dos participantes. Por exemplo, um relatório de um problema gera uma tarefa que necessita de discussão até que se compreenda o problema, se prepare uma solução e se verifique que a solução funciona. Tudo isto deverá estar devidamente documentado na plataforma.

Como pode constatar, a utilização desta metodologia, baseada em tarefas que são criadas e listadas numa plataforma comum, permite gerir com eficiência um projeto, sem que aspectos importantes sejam esquecidos.

A Figura 5.4 apresenta a interface para criação de uma nova tarefa. Note que é possível criar tarefas relativas a *Bugs* (problemas) e *Funcionalidades* (aspectos a acrescentar). Segue-se um campo de título que identifica a tarefa e uma descrição. Estes campos devem ser utilizados para descrever corretamente a tarefa.

Caso a tarefa seja o relato de um problema, deverá descrever-se o problema com detalhe de forma a auxiliar a resolução. Em alguns casos, ficheiros com capturas de ecrã ou registos de erros podem facilitar a análise.

Caso a tarefa corresponda a uma funcionalidade, devem descrever-se os detalhes da

funcionalidade de forma a que não exista qualquer dúvida em relação ao que falta fazer. Podem especificar-se outros aspetos como a prioridade da tarefa, o tempo estimado para a sua resolução ou mesmo se esta tarefa está relacionada com outra. Podem-se igualmente definir observadores, que são utilizadores que poderão estar interessados em seguir o desenvolvimento da tarefa.

Exercício 5.5

Crie uma tarefa indicando a necessidade de realizar os exercícios desta secção do guião. Qual o tipo desta tarefa?

Adicione todo o detalhe que ache importante.

Uma tarefa possui um percurso de vida bem definido, podendo ter um de dois estados principais: *aberto* ou *fechado*. O estado de fechado corresponde a uma tarefa que já concluiu o seu percurso e está tratada. O estado de aberto corresponde a uma tarefa que ainda se encontra ativa, podendo depois ter vários sub-estados. A Figura 5.5 mostra uma lista de tarefas abertas, indicando igualmente o sub-estado, neste caso *Novo*.

A plataforma **CodeUA** suporta vários sub-estados, a saber:

Novo: Estado definido para todas as tarefas recentemente criadas e que ainda não foram processadas para um outro sub-estado.

#	Tipo	Estado	Prioridade	Assunto	Atribuído a	Alterado	Data fim	% Completo
6549	Funcionalidade	Novo	Normal	Necessário realizar exercícios do guião		26/10/2013 20:21		

Figura 5.5: Página de listagem das tarefas

Em Curso: Tarefa que se encontra em processo de resolução. Isto é, a funcionalidade está a ser implementada ou o problema (*Bug*) está a ser resolvido.

Feedback: Após ter sido tomada uma ação sobre uma tarefa, o criador da tarefa tem de validar que o assunto foi abordado de forma correta. Até lá a tarefa fica a aguardar *Feedback*.

Recusado: A tarefa não se enquadra no projeto, não é válida, ou não foi especificada de forma correta, tendo sido recusada de qualquer outro processamento.

Resolvido: A tarefa foi resolvida e o seu criador verificou que tudo está de acordo com o reportado.

Fechado: O gestor, ou outro membro do projeto pode marcar a tarefa como inativa definindo-a como estando no sub-estado *Fechado*. Uma tarefa deve ir para o sub-estado de *Fechado* após ter sido recusada ou resolvida.

Exercício 5.6

Agora que se encontra a resolver os exercícios do guião, marque a tarefa como estando no estado *Em Curso* e defina quem é o responsável (Atribuído a).

Pode também adicionar uma estimativa de horas e indicar o quanto ela se encontra completa.

O módulo de *Tarefas* comunica com o módulo de *Diagramas de Gantt*. Este módulo é extremamente útil para planejar corretamente um projeto, permitindo avaliar se este se encontra dentro das expectativas temporais. O seu funcionamento baseia-se na lista de tarefas do projeto, e caso elas possuam datas de início e de fim, na representação destas num formato que permita identificar as diferentes fases do projeto. A Figura 5.6 apresenta o diagrama de *Gantt* relativo às tarefas de um hipotético primeiro trabalho de aprofundamento de conhecimentos. A linha vermelha representa o instante atual, sendo que cada barra horizontal representa a data de início e duração de uma tarefa.

Exercício 5.7

Defina várias tarefas e sub-tarefas, com datas de início e fim diferentes. Defina que algumas destas tarefas já se encontram em curso, tendo uma percentagem do seu trabalho já realizado. Verifique depois qual o diagrama de *Gantt* produzido.

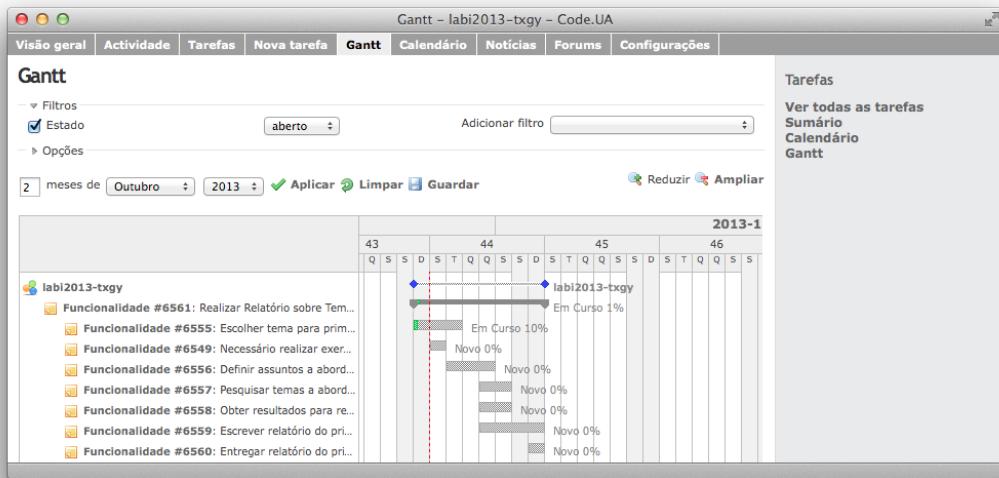


Figura 5.6: Diagrama de Gantt para o primeiro trabalho de aprofundamento de conhecimentos.

Partilha de informação

A plataforma **CodeUA** e, de uma forma geral, todas as plataformas de gestão de projetos, mesmo que não sejam orientadas a projetos de programação, possuem áreas para a partilha de ficheiros entre os seus membros ou entre os membros e o público em geral. Por exemplo, num projeto de elaboração de um relatório técnico, todos os artigos pesquisados, resultados obtidos e as diferentes partes do relatório irão estar alojadas na plataforma. Com isto tenta-se maximizar o paralelismo das ações, minimizando dependências entre membros. Isto é, nunca um projeto deve estar dependente das ações de um membro em específico ou pelo menos este tipo de eventos deve ser minimizado. Considerando que a lista de tarefas se encontra descrita, é natural presumir que diferentes membros irão focar-se em diferentes tarefas em paralelo, o que minimiza a duração do projeto.

A plataforma **CodeUA** em específico suporta 3 formas de alojar informação, cada uma adaptada a um fim diferente.

Documentos: Permite alojar documentos gerais do projeto, não relacionados com uma versão de desenvolvimento ou considerados estáticos. Uma notícia para a comunicação social é um exemplo de um documento para esta área.

Ficheiros: Permite alojar ficheiros do projeto em que se espera que existam versões diferentes destes ficheiros. Um exemplo será a publicação de versões diferentes de um programa que esteja a ser desenvolvido.

Repositórios: Permite alojar conteúdos de trabalho do projeto geridos por um sistema de controlo de versões.

Exercício 5.8

Ative os módulos de *Documentos* e de *Ficheiros*. Partilhe um ficheiro local em cada um destes módulos.

O módulo de *Repositórios* merece uma atenção particular porque permite associar ao projeto um repositório de ficheiros gerido por um *sistema de controlo de versões*. Estes sistemas são muito úteis, especialmente em projetos de software, porque mantêm um registo histórico da evolução do projeto. O **CodeUA** suporta dois sistemas alternativos de controlo de versões: Apache Subversion (SVN) ou Git.

5.3 Sistemas de controlo de versões

Um sistema de controlo de versões (em inglês: Version Control System (VCS)) permite gerir projetos de software de maneira a possibilitar a edição concorrente dos ficheiros por diferentes utilizadores ao longo do tempo, mantendo sempre um registo completo de todas as alterações feitas, de quem as fez e quando. Também permitem reverter alterações ou combinar versões. Estes sistemas também são conhecidos como sistemas de controlo de revisões (em inglês: Revision Control System (RCS) ou ainda gestores de configuração de software (em inglês: Software Configuration Management (SCM)), mas este termo é mais genérico. Há vários sistemas alternativos de controlo de versões usados atualmente. Neste guião usamos o sistema Git, por ser um dos mais poderosos.

Antes de se analisar o funcionamento desta ferramenta, existem alguns termos que é necessário aclarar, pois irão repetir-se ao longo deste guião. Na maioria dos casos os termos usados serão na língua inglesa, uma vez que têm uma correspondência direta com os comandos e operações disponíveis.

O processo de produção de software envolve a manipulação de muitos ficheiros por vários programadores, o que implica que exista alguma forma de coordenação das suas atividades para produzir algo coerente e de acordo com especificações inicialmente impostas. Este processo implica um ciclo de produção e de melhoramentos sucessivos que envolve **working trees** (árvores de trabalho) e repositórios de versões. As árvores de trabalho são as áreas que os programadores usam para desenvolver novas funcionalidades ou para corrigir erros e têm por base uma determinada versão do produto em que os programadores estão a trabalhar. Quando estas versões atingem um determinado ponto de maturidade, os programadores podem registá-las no repositório como (mais) uma versão do produto. À extração de uma versão do repositório para uma árvore de trabalho

dá-se o nome de **check out**. Já ao processo inverso, o de criar uma nova versão no repositório a partir de uma árvore de trabalho, dá-se o nome de **check in** ou **commit**.

Exercício 5.9

Aceda à página de desenvolvimento do kernel *Linux*, que se encontra em <http://git.kernel.org/cgit/> e verifique os múltiplos repositórios Git que contém. Verifique igualmente que cada uma possui um tema específico.

A partir de uma mesma versão no repositório podem criar-se linhas de evolução diferentes, às quais se dá o nome de *branches* (ramos). Dessa forma, a evolução das versões do software faz-se através de sucessivos *commits* no mesmo ramo e de ramificações a partir de algumas versões *committed*.

Exercício 5.10

Aceda à *Working Tree* do *Linux* dedicada ao controlo de temperatura, disponível em <http://git.kernel.org/cgit/linux/kernel/git/rzhang/linux.git/> e verifique que existem vários *branches*. Estes ramos contêm diferentes estados de desenvolvimento do código, sendo *master* o ramo atualmente ativo.

Feita esta explicação do paradigma base de atualização de software e de controlo de versões, podemos passar para uma descrição mais completa de alguns termos usados pelo Git.

Working tree (árvore de trabalho) - A árvore de trabalho é um diretório no sistema de ficheiros ao qual se encontra associado um repositório (tipicamente existe nesta diretoria um sub-diretório chamado `.git`). A árvore de trabalho inclui todos os ficheiros e sub-diretórios nela existentes. No fundo é onde o programador desenvolve o seu código e tem os seus ficheiros. Cada programador pode ter uma ou mais árvores de trabalho associadas a um mesmo repositório.

Repository (repositório) - Um repositório é uma coleção de *commits*, sendo que cada um destes é um registo do estado em que se encontrava uma dada árvore de trabalho numa data passada. De uma forma simplista pode-se considerar que um *commit* é uma alteração ao código ou versões. Os *commits* de um repositório podem ainda ser identificados como ramos, caso sejam o início de um ramo, ou possuir etiquetas (*tags*) de forma a serem facilmente identificados por um nome.

Check out (extração) - Quando se faz uma extração de uma versão (ou de um *commit*) de um repositório cria-se uma árvore de trabalho com todos os ficheiros e diretórios pertencentes a essa versão. O processo de extração regista também na árvore de trabalho o identificador do ramo ou *commit* do qual a árvore de trabalho actual descende. Esse identificador é genericamente designado por **HEAD**.

Commit (ou check in) - Um *commit* é uma cópia de uma árvore de trabalho realizada num dado momento. Para além disso, um *commit* é igualmente uma evolução de algo que existia anteriormente, que é o *commit* a partir do qual a árvore de trabalho foi criada (indicada por **HEAD**). O *commit* anterior torna-se pai do *commit* atual. É esta relação entre *commits* que dá por sua vez origem à noção de histórico de revisões (*revision history*).

Branch (ramo) - Um ramo é uma sequência de *commits* sucessivos que formam uma linha de desenvolvimento independente. No Git os ramos podem ser referenciados por um nome que funciona como sinónimo do último *commit* nesse ramo. A linha principal de desenvolvimento na maioria dos repositórios é feita num ramo chamado **master**. Embora seja um nome definido por omissão, não é de qualquer forma especial.

Index (índice) - O índice também é chamado de *staging area* (área de ensaio/testes), pois regista as alterações efetuadas na árvore de trabalho que serão incluídas no próximo *commit*. O fluxo mais comum de eventos envolvendo o índice é o representado na Figura 5.7. Após a criação de um repositório, cria-se uma árvore de trabalho com um *check out*, na qual são realizada todas as edições de ficheiros. Assim que o trabalho numa árvore de trabalho atinja uma meta (implementação de uma funcionalidade, correção de um erro, fim de um dia de trabalho, compilação com sucesso, etc.), as alterações na mesma são acrescentadas de forma sucessiva ao índice. Assim que o índice contiver todas as alterações que pretende salvaguardar no repositório, num *commit*, as mesmas são transmitidas a este.

Com base neste diagrama, as próximas secções irão descrever a importância de cada uma destas entidades na utilização do Git.

Repositório: Monitorização dos conteúdos de um diretório

Como referido anteriormente, a função do repositório resume-se a manter cópias congeladas dos conteúdos de um diretório tiradas em diversos momentos ao longo do tempo. A estas cópias chamam-se *snapshots* (instantâneos).

A estrutura de um repositório Git assemelha-se à estrutura de um sistema de ficheiros UNIX: um sistema de ficheiros começa num diretório raiz, que por sua vez possui outros tantos diretórios, muitos destes têm por sua vez nós folha, ou ficheiros, que contêm dados.

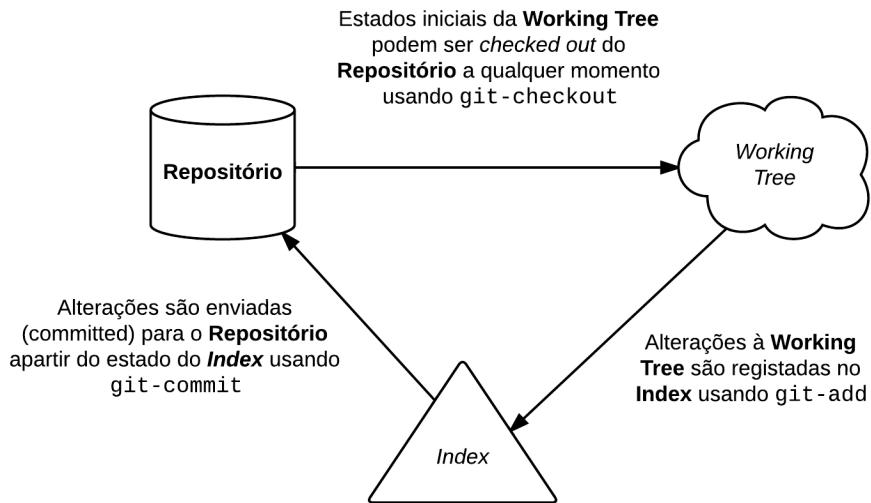


Figura 5.7: Fluxo de eventos no ciclo de vida de um projeto gerido por Git.

Internamente, o Git partilha uma estrutura em tudo similar, embora tenha uma ou outra diferença. Em primeiro, representa os conteúdos de um ficheiro em ***blobs***, que também são muito semelhantes a um diretório. O nome de um *blob*, também chamado o seu ***hash id***, é um número calculado pelo algoritmo Secure Hashing Algorithm (versão 1) (SHA-1) aplicado sobre o tamanho e conteúdos do próprio *blob*. O *hash id* parece um número arbitrário, mas tem duas propriedades interessantes: primeiro, certifica que o conteúdo do *blob* não foi alterado; e segundo, garante que *blobs* de conteúdo igual têm o mesmo nome, independentemente de onde apareçam.

A diferença entre um *blob* do Git e um ficheiro de um sistema de ficheiros é o facto do *blob* não haver relação qualquer com os conteúdos do ficheiro. Toda essa informação é armazenada na árvore que armazena o *blob*. Uma árvore pode conhecer os conteúdos de um *blob* como sendo o ficheiro ***foo*** criado em Agosto de 2004, ao passo que outra pode conhecer o mesmo ficheiro como sendo o ***bar*** criado cinco anos antes.

Introdução ao *blob*

Agora que tem um panorama geral sobre o funcionamento do Git, resta treinar com alguns exemplos. Como primeiro passo irá criar um repositório exemplo e mostrar como

o Git funciona.

Exercício 5.11

Dirija-se ao seu projeto no **CodeUA**, ative o módulo de repositórios e crie um repositório do tipo Git.

Copie o comando indicado no topo da página e execute-o numa linha de comando. Deverá ser algo como: `git clone https://code.ua.pt/git/labi2018-txgy`.

Este comando deve criar um novo diretório `labi2018-txgy/`, que é uma réplica local do repositório do projeto. É nesse diretório que irá criar e atualizar os ficheiros do projeto. Mude o seu diretório atual para lá (`cd labi2018-txgy`).

Cada membro do projeto deverá fazer a clonagem, agora ou mais tarde, para criar a sua cópia pessoal do repositório.

Exercício 5.12

Crie um diretório novo chamado `teste` e, dentro dele, crie um ficheiro chamado `saudacao` com o texto “Hello, world!”.

```
mkdir teste  
cd teste  
echo 'Hello, world!' > saudacao
```

A partir deste momento já pode usar o seguinte comando para conhecer o *hash id* que o Git irá usar para armazenar o texto introduzido.

```
git hash-object saudacao  
af5626b4a114abcb82d63db7c8082c3c4756e51b
```

No seu computador deverá obter exatamente o mesmo *hash id*. Muito embora esteja a usar um computador diferente daquele onde este guião foi escrito, os *hash ids* serão os mesmos porque os ficheiros têm o mesmo conteúdo. Esta propriedade permite que um repositório seja utilizado por programadores em vários computadores, mantendo a geração dos *blobs* consistente.

Agora podemos fazer um *commit*.

```
git add saudacao  
git commit -m "Adicionei a minha saudação"
```

Neste momento o nosso *blob* deverá fazer parte do sistema tal como se esperava, usando o *hash id* determinado anteriormente. Por conveniência o Git requer o menor número de dígitos necessários para identificar inequivocamente o *blob* no repositório. Tipicamente seis ou sete dígitos são suficientes:

```
git cat-file -t af5626b  
blob  
git cat-file blob af5626b  
Hello, world!
```

Ainda não conhecemos o *commit* que armazena o nosso ficheiro ou a sua *tree*, mas recorrendo exclusivamente ao *hash id* que resume o seu conteúdo, foi possível determinar que o ficheiro existe e consultar o mesmo. Ao longo de toda a vida do repositório, e independentemente do local no repositório onde o ficheiro estiver, este conteúdo manterá esta identificação.

Os *blobs* são armazenados em *trees*

Os conteúdos de um ficheiro são armazenados em *blobs*, mas estes têm poucas funções (Não têm nome, nem estrutura, servindo apenas como agregadores de conteudos). Para o Git poder representar a estrutura e o nome dos ficheiros, é necessário associar os *blobs* como nós folha de uma árvore (*tree*). É depois possível determinar que existe um *blob* na *tree* onde foi feito um *commit*.

Exercício 5.13

Liste os *blobs* armazenados na *tree HEAD*.

```
git ls-tree HEAD  
100644 blob af5626b4a114abcb82d63db7c8082c3c4756e51b saudacao
```

O primeiro *commit* acrescentou o ficheiro **saudacao** ao repositório. Este *commit* contém uma árvore Git, que por sua vez contém apenas uma folha: o *blob* do conteúdo de **saudacao**.

É também possível identificar a árvore, tal como fizemos com o *blob*:

```
git cat-file commit HEAD
tree 5ae5597b6ae0854f77d50dc8ec828eb0928b9fe2
author João Paulo Barraca <jpbarraca@ua.pt> 1382903138 +0000
committer João Paulo Barraca <jpbarraca@ua.pt> 1382903138 +0000
```

Adicionei a minha saudação

A *hash id* para cada *commit* é única no repositório, visto que contém o nome do autor e a data em que o *commit* foi realizado, mas a *hash id* da árvore deverá ser a mesma no exemplo deste guião e no seu sistema, contendo apenas o nome do *blob* (que é o mesmo).

Vamos verificar que se trata realmente do mesmo objecto *tree*:

```
git ls-tree 5ae5597
100644 blob af5626b4a114abcb82d63db7c8082c3c4756e51b    saudacao
```

O processo tem início quando se acrescenta um ficheiro ao índice. Por agora, vamos considerar que o índice é usado inicialmente para criar *blobs* a partir de ficheiros. Quando se acrescenta o ficheiro **saudacao** ocorre uma alteração no repositório. Ainda não é possível ver esta alteração através de um *commit*, mas é possível ver o que aconteceu.

Execute:

```
git log
```

Eis a prova de que o repositório contém um só *commit*, que contém uma referência para uma árvore que armazena um *blob*, *blob* este que armazena o conteúdo do ficheiro **saudacao**.

Commits

Um ramo numa *tree* não é, pois, mais do que um nome que referencia um *commit*. É possível examinar todos os *commits* no topo de um ramo usando o comando:

```
git branch -v
* master eb64bfd Adicionei a minha saudação
```

Este comando indica que a árvore **master** possui no seu topo um *commit* com *hash id* **eb64bfd**.

Neste exemplo podemos fazer o *reset* da **HEAD** da árvore de trabalho a um *commit* específico. Ou seja, colocar o nosso repositório no estado indicado pelo *commit* respetivo.

```
git reset --hard eb64bfd
```

A opção **-hard** serve para garantir que todas as alterações existentes na árvore de trabalho são removidas, quer tenham sido registadas para um *check in* ou não (mais será dito à frente).

Uma alternativa mais segura ao comando anterior seria:

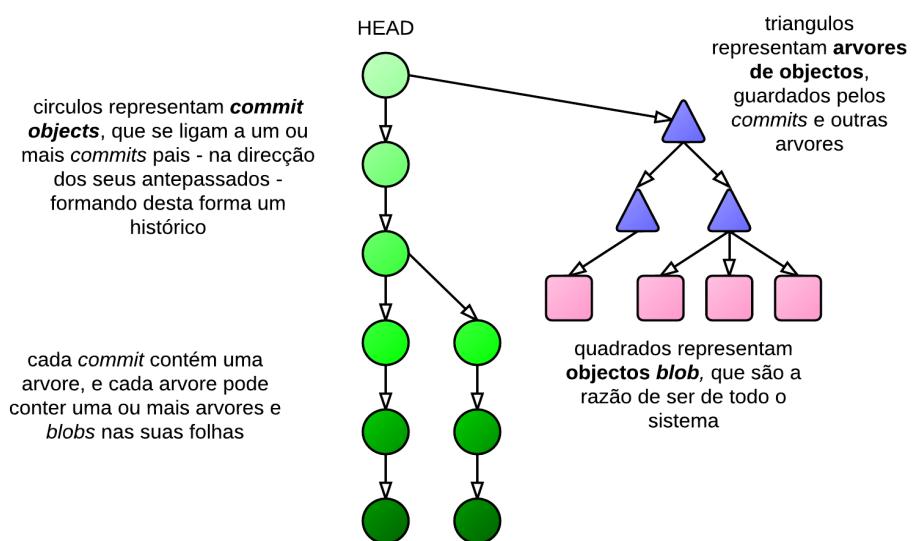
```
git checkout eb64bfd
```

A diferença é que as alterações aos ficheiros na *working tree* serão preservadas. Por exemplo, os ficheiros locais adicionais não são apagados.

Se usarmos da opção **-f** do comando **checkout**, o resultado será semelhante ao uso do **reset -hard**, excepto que **checkout** apenas altera a árvore de trabalho e **reset -hard** altera o **HEAD** do ramo atual para a referência da *tree* passada por argumento.

Um dos benefícios do Git, como sistema orientado a *commits*, é que é possível reescrever processos extremamente complexos usando um subconjunto de processos muito simples. Por exemplo, se um *commit* tiver múltiplos pais, trata-se de um *merge commit*: vários *commits* são unidos num único. Ou se um *commit* tiver múltiplos filhos, representa o antepassado (*ancestor*) de um ramo, etc. Para o Git estes conceitos não existem na realidade, são apenas “nomes” usados para descrever processos que existiam em sistemas de gestão de código anteriores. Para o Git, tudo é uma colecção de objectos de *commits*, sendo que cada um contém uma *tree* que referencia outras *trees* e *blobs* onde a informação está armazenada. Qualquer outra coisa é apenas uma questão de terminologia.

A próxima figura ajuda-nos a compreender melhor.



Nomes alternativos de *commits*

Já vimos que um mesmo *commit* podem ser referenciado por diversos nomes. Chamamos-lhes sinónimos ou nomes alternativos (em inglês *aliases*). Estes são os principais tipos de sinónimos e os seus casos de utilização:

eb64bfd... Um *commit* pode sempre ser referenciado usando o seu *hash id* completo de 40 dígitos hexadecimais. Normalmente isto acontece quando se recorre ao cortar&colar (*cut&paste*), já que existem normalmente outras formas mais práticas de referenciar um *commit*.

eb64bfd Apenas é necessário usar um número de dígitos suficientes para garantir que existe apenas um *commit* começado por esses mesmos dígitos no repositório. Na

maioria dos casos, entre 6 e 8 dígitos são suficientes.

HEAD O *commit* actual, o mais recente, pode ser sempre referido pelo nome **HEAD**. Quando se faz um novo *commit*, **HEAD** passa a apontar para esse. Se fizer *check out* de um determinado *commit* (em vez de um nome de ramo), então **HEAD** refere-se a esse *commit* e não ao de qualquer outro ramo. Este é um caso especial, chamado também de *detached head*.

branchname O nome de um ramo, por exemplo **master**, funciona como sinónimo do último *commit* feito nesse ramo. É portanto um sinónimo “móvel”, tal como **HEAD**.

tagname É possível associar explicitamente um nome alternativo a um qualquer *commit*, mesmo que não seja um novo ramo. Estes sinónimos chama-se *tags* (etiquetas) e ficam sempre associado ao mesmo *commit*, ao contrário dos nomes de ramos.

name[^] O pai de qualquer *commit* pode ser referenciado usando o acento circunflexo (^). Se um *commit* tiver múltiplos pais, refere-se ao primeiro.

name^{^^} Vários acentos circunflexos podem ser utilizados de forma sucessiva. Neste caso, está-se a referir ao pai do nosso pai (avô).

name^{^2} No caso de existirem múltiplos pais, pode-se referir a um pai em concreto através do seu número de ordem. No exemplo ao 2º pai.

name^{~10} Para aceder a um antepassado distante, pode-se recorrer ao til (~) para indicar quantas gerações subir na *tree*. É a mesma coisa que fazer name^{^^^^^^^^^}.

name:path Para referir um certo ficheiro dentro da *tree* de um determinado *commit*, pode-se especificar o nome do ficheiro após o carácter dois-pontos (:). Esta forma é extremamente útil em conjunto com o comando **show**, por exemplo para comparar versões anteriores de um ficheiro:

```
git diff HEAD^1:saudacao HEAD^2:saudacao
```

name^{^tree} É possível referenciar a *tree* de um *commit*, na vez do próprio *commit*.

name1..name2 Esta notação permite indicar uma gama de *commits*: todos os ocorridos após **name1** (sem o incluir) e até **name2**. (Mais rigorosamente, indica todos os antepassados de **name2** que não sejam antepassados de **name1**.) Se se omitir **name1** ou **name2**, **HEAD** é usado em seu lugar. É muito útil quando usado em conjunto com o comando **log** por forma a analisar o que ocorreu num determinado período de tempo.

name1...name2 O uso de três pontos é bastante diferente do uso anterior de dois pontos. Em comandos como **log**, refere-se a todos os *commits* antepassados de **name1** ou de **name2**, mas não de ambos.

master.. É equivalente a **master..HEAD**. Ou seja, inclui todos os *commits* desde que o ramo atual se desviou de master.

..master Também é uma equivalência, especialmente útil quando usada com o comando **fetch** para determinar que alterações ocorreram desde o último **rebase** ou **merge**.

--since="2 weeks ago" Refere-se a todos os *commits* desde uma data.

--until="1 week ago" Refere-se a todos *commits* até uma data.

--grep=pattern Refere-se a todos *commits* cuja a mensagem contém o padrão “pattern”.

--committer=pattern Refere-se a todos *commits* cujo autor (*committer*) contém no seu nome o padrão “pattern”.

--author=pattern Normalmente é o mesmo que *committer*, mas nalguns casos pode ser diferente (envio de *patches* por email).

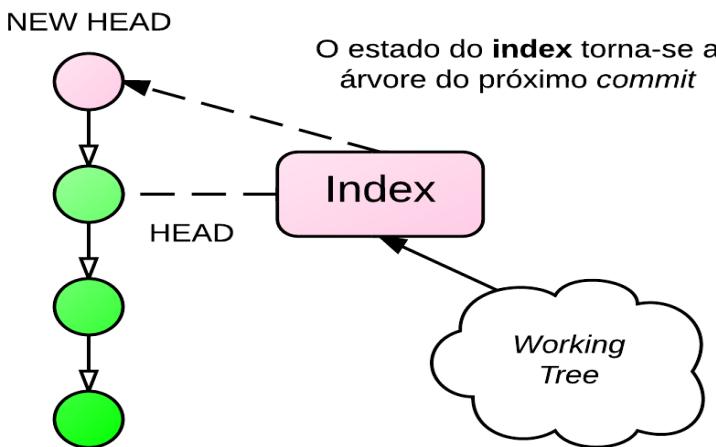
--no-merges Refere-se a todos os *commits* na gama que têm apenas um pai — desta forma ignora todos os *commits* de *merge*.

Índice: o intermediário

Entre os nossos ficheiros que contêm dados e estão armazenados na árvore de trabalho, e os *blobs* Git, que estão armazenados no repositório, fica a área de ensaio (*staging area*). A esta área também se chama índice, porque é realmente uma lista de apontadores para *trees* e *blobs* criados através do comando **add**. Estes novos objectos são depois compactados numa nova *tree* que será *committed* no repositório. Tal quer também dizer que caso se aplicarmos **reset** sobre o índice, todas as alterações não *committed* serão perdidas.

O índice é pois uma zona de preparação para o próximo *commit* e existe uma boa razão para existir: desta forma é possível ter mais controlo sobre as alterações a submeter. Por exemplo, podemos preparar um *commit* com vários ficheiros alterados, mas evitar incluir outro em que ainda estamos a trabalhar e que causaria problemas (como erros de compilação).

Também é possível usar o Git sem usar o índice, recorrendo à opção **-a** sempre que fazemos um *commit*. Desta maneira todas as alterações feitas são submetidas no mesmo *commit*.



Ficheiros, directórios e " hunks" (alterações individuais de um ficheiro) são adicionados ao **index** usando `git add` e `git add --patch`

Exercício 5.14

Altere o ficheiro criado, de forma a adicionar o nome do seu grupo.

De seguida utilize o comando `git add` para adicionar este ficheiro ao índice e depois `git commit` para criar um *commit*.

Use o comando `git status` para verificar que alterações existem na árvore de trabalho.

Utilização de um repositório Git

Nas secções anteriores foram descritas algumas funcionalidades internas do Git e diversos conceitos associados aos sistemas de gestão de versões. Nesta secção serão apresentados os comandos mais práticos e úteis de utilização no dia-a-dia.

Na raiz da árvore de trabalho do seu projeto, crie um ficheiro `README.md` com um pequeno texto. De seguida adicione o ficheiro `README.md` ao repositório.

```
echo "Este é o projeto labi-txgy" > README.md
git add README.md
git commit -m "Commit README inicial"
```

Agora altere o ficheiro **README.md** de modo a incluir uma descrição do que este repositório irá conter. Volte a registar o *commit* e a enviar para o servidor.

Sincronize com o repositório no servidor usando o comando:

```
git push --all origin
```

Isto evidencia um aspeto importante do Git: todas as alterações, mesmo que gerando novos *commits*, são efetuadas na cópia local do repositório. É o comando **git push** que força a sincronização do repositório remoto com o local. (A opção **-all origin** é necessária apenas no primeiro *push* pois ainda não existe nenhum **HEAD** no repositório do servidor.)

Uma vantagem de utilizar um repositório centralizado num servidor é que está sempre disponível para os vários programadores poderem submeter as suas alterações paralelamente. Porém o Git também pode ser usado de forma perfeitamente distribuída, ao contrário de outros VCS.

Exercício 5.15

Inicie sessão no servidor **xcoa.av.it.pt** utilizando Secure Shell (SSH).

Obtenha neste servidor uma cópia do repositório criado na plataforma **CodeUA**. Verifique que os conteúdos locais, os mostrados na página do repositório do **CodeUA** e os presentes no **xcoa.av.it.pt** são exactamente os mesmos.

Exercício 5.16

No computador local, faça novas alterações ao ficheiro **README.md**, por exemplo adicionando o email de contacto e número mecanográfico dos seus membros. Submeta e envie as alterações para o repositório remoto.

No servidor **xcoa.av.it.pt**, dentro do directório do projeto, utilize o comando **git pull** para actualizar o repositório local.

Como descrito anteriormente, o Git regista alterações, o que também inclui remoção de documentos. O comando **git rm** permite registar a remoção de ficheiros. No entanto, a remoção é apenas uma alteração ao estado do ficheiro e como qualquer alteração no Git, é sempre possível recuperar um estado anterior, ou seja, ir para o estado exacto de qualquer *commit*. A consequência é que uma vez que um ficheiro é adicionado a um

repositório, não é possível removê-lo definitivamente. Desta forma nunca existe perda de informação.

Considere a seguinte execução de comandos:

```
echo "teste" >> fich.txt
git add fich.txt
git commit -m "teste"
git rm fich.txt
git commit -m "teste"
```

O resultado deverão ser 2 *commits*. Um registando o ficheiro **fich.txt**, e outro sinalizando a sua remoção. O comando **git log** deverá demonstrar esta sequência.

Exercício 5.17

Adicione um ficheiro **test.txt** com um conteúdo arbitrário e efectue um *commit*, enviando de seguida o novo *commit* para o repositório remoto.

Verifique que pode obter este ficheiro na interface web da plataforma **CodeUA** e no servidor **xcoa.av.it.pt** (após um **git pull**).

Remova o ficheiro e envie as alterações para o servidor remoto. Verifique que uma nova atualização (**git pull**) na outra máquina irá remover o ficheiro adicionado.

O comando **git log** no repositório do servidor **xcoa.av.it.pt** deverá mostrar os passos dados.

Um ficheiro pode ser recuperado. Basta colocar a árvore de trabalho no local correto da tree. Uma maneira de realizar isto é executar um **git reset** para um *commit* anterior.

Exercício 5.18

Utilize o comando **git log** e localize o *commit* em que o ficheiro **test.txt** foi adicionado. Pode também utilizar o comando **git show <hashid>** para ver o conteúdo do *commit*, isto é, quais as modificações que o *commit* efetuou.

Utilize o comando **git reset** para reaver o ficheiro perdido.

Exercício 5.19

Altere o ficheiro `test.txt` para incluir mais texto.

Utilize o comando `git diff test.txt` para verificar a diferença entre o ficheiro existente na árvore de trabalho e o registado na `tree`.

Muitos mais comandos existem que não foram mencionados neste guião. No entanto, todos obedecem às regras descritas neste guião. Processos mais avançados de gestão de versões concorrentes ficam fora do âmbito deste guião.

5.4 Para aprofundar

Exercício 5.20

Utilize os métodos descritos para gerir a elaboração do próximo relatório. Pode definir todas as etapas de realização de um relatório técnico, definir o conteúdo das secções e definir quem está responsável por que parte. Quando todas as tarefas estiverem no estado *Resolvido*, o relatório deverá estar pronto a ser entregue.

Exercício 5.21

Explore os outros módulos da plataforma **CodeUA** e identifique qual a sua utilidade para a gestão de projetos.

Exercício 5.22

De uma forma geral, o Git (ou outro sistema semelhante) é extremamente útil a qualquer programador. Experimente criar um repositório para guardar os trabalhos que faz noutras disciplinas, por exemplo em Programação I.

Ferramentas de Automação

Objetivos:

- Automação em projetos de software
- GNU make, automake e autoconf

6.1 Automação em Projetos

Durante o desenvolvimento de um projeto de software, ou mesmo de um documento técnico em L^AT_EX, é frequente ter de realizar operações de forma repetitiva a cada nova versão. Um exemplo é a necessidade de compilar um programa sempre que é modificado, gerar documentação a partir de ficheiros .tex ou transferir novas versões de páginas HMTL para publicação num servidor web.

Considere o caso comum no desenvolvimento de uma aplicação simples em Java. De uma forma geral, e para evitar problemas, é necessário apagar todas as classes locais.

```
rm -f *.class
```

De seguida é necessária a compilação de todos os ficheiros de código fonte.

```
javac *.java
```

Se se pretender gerar documentação no formato *JavaDoc* será necessário:

- Criar um diretório para armazenar a documentação: `mkdir doc`

- Caso exista o directório, apagar o seu conteúdo: `rm -rf doc/*`
- Gerar a documentação: `javadoc -d doc classname`

Voltar a gerar a documentação exige a repetição do processo.

No caso de se utilizar L^AT_EX com bibliografia e acrónimos, torna-se necessário executar uma sequência tal como:

- Compilar o documento e criar a lista de índices, referências e acrónimos: `pdflatex doc.tex`
- Colocar as referências no texto e a lista no final: `bibtex doc`
- Compor o documento final com acrónimos e referências: `pdflatex doc.tex`
- Pode ser preciso repetir o passo pois podem ter sido introduzidas referências novas: `pdflatex doc.tex`

Estes passos têm de ser repetidos sempre que se altera algum ficheiro `.tex` ou a lista de referências ou a lista de acrónimos.

Outro problema frequente refere-se ao processo de distribuição de código fonte não compilado. Muitos programas não são autocontidos; dependem de bibliotecas, módulos ou classes externas às tradicionalmente incluídas na linguagem. Ora, como garantir que num dado sistema existem todas as dependências necessárias para compilar um certo programa? E se o programa requer um módulo M para o qual existem várias versões de fornecedores diferentes? Recusa-se a compilar se não encontrar o módulo M do fornecedor X, mesmo que o sistema já possua o módulo M do fornecedor Y?

Estas questões também se colocam com aplicações distribuídas na forma compilada, mas que usam bibliotecas partilhadas carregadas dinamicamente. É por essa razão que existem sistemas de gestão de pacotes como o `apt-get`, que já usou.³⁰

De forma a agilizar o processo de desenvolvimento, distribuição e adaptação a sistemas heterogéneos, existem ferramentas que auxiliam o programador ou escritor técnico, automatizando processos ou permitindo a deteção dos módulos existentes e a adaptação dinâmica.

Neste guião irão ser abordadas 2 funcionalidades principais:

- Automatização de tarefas.
- Deteção do ambiente para compilação e instalação.

³⁰O problema não se colocaria com aplicações monolíticas, mas seria um desperdício de recursos. Quando quer beber vinho, tem de comprar um pacote com a garrafa, um copo e um saca-rolhas?

6.2 GNU make

A ferramenta *make*[14] permite determinar que partes de um software necessitam de ser compiladas ou recompiladas após alguma alteração. É particularmente útil quando o programa é grande, porque permite reaproveitar ao máximo o trabalho que já se teve ao compilar outras partes não afetadas pela alteração. Repare que alguns programas podem demorar largos minutos ou mesmo horas a compilar todo o código fonte.

Embora o processo de compilação seja a principal aplicação do *make*, esta ferramenta pode ser aplicada a qualquer processo em que a produção de um ficheiro alvo depende de executar certas operações sobre outros ficheiros, que podem ser dados ou, por sua vez, também eles serem dependentes de outros.

Para o *make* saber quem depende de quem e o que tem de fazer para construir ou atualizar os ficheiros alvo, tem de ler essa informação de uma *makefile*. Uma *makefile* é um ficheiro de texto com uma sintaxe simples, que geralmente tem o nome **Makefile** e é colocado no diretório base de um projeto. Nesse caso, para compilar o projeto bastará invocar o comando **make** e assistir à magia da resolução de dependências.

Se não existir um ficheiro **Makefile**, o *make* procura outro chamado **makefile** e, se esse não existir, apresenta um erro.

Exercício 6.1

Crie um directório denominado **tema6** e desloque-se para lá.

Execute o comando **make** e verifique o seu resultado.

Crie um ficheiro vazio chamado **Makefile** e volte a executar **make**. Compare o resultado com a execução anterior.

Também é possível usar uma *makefile* com outro nome qualquer, executando **make -f nome-da-makefile**. Isto pode ser usado para ter *makefiles* diferentes para ambientes diferentes, *Linux* vs *OS X*, por exemplo.

Formato de uma *makefile*

Uma *makefile* é composta, essencialmente, por um conjunto de *regras*, com o formato e o significado que veremos na Secção 6.2. Além de regras, também pode conter:

- Definições de *variáveis*.

```
nome-da-variavel = valor
```

- Comentários, iniciados por um símbolo **#** e que se prolongam até ao fim da linha.

```
# comentario ignorado pelo make
```

- Diretivas de inclusão de outra makefile. Podem usarse para organizar as regras de forma modular.

```
include outra-makefile
```

Regras

As regras são a parte fundamental de uma makefile. Cada regra tem o seguinte formato:

```
alvo ... : pré-requisito ...
            receita-para-criar-alvo
            ...
            
```

O *alvo* é um ficheiro a ser gerado ou atualizado. Também se podem definir alvos falsos (*phony targets*), que se usam para invocar ações e que não vão corresponder a ficheiros. A seguir ao símbolo **:** vem uma lista de zero ou mais *pré-requisitos* ou dependências. Estes são os ficheiros de que o alvo depende. A *receita* é uma sequência de comandos de shell que permitirão gerar o alvo a partir dos pré-requisitos. Note que as linhas da receita têm de começar com um carácter TAB.

Quando o **make** é executado, procura a primeira regra da makefile e verifica-a: se algum pré-requisito é mais recente do que o alvo (ou algum dos alvos), então executa a receita. Porém, antes disso verifica se é preciso atualizar cada um dos pré-requisitos, procurando e verificando regras em que sejam alvos. Aplicado recursivamente, este processo garante que todos os alvos que dependam direta ou indiretamente de uma alteração serão atualizados.

Por exemplo, uma makefile para compilar um pequeno programa Java, poderia conter apenas uma regra:

```
Programa.class: Programa.java
                javac Programa.java
```

Esta makefile declara que o alvo **Programa.class** depende do pré-requisito **Programa.java** e que é possível criar (ou atualizar) o primeiro executando a receita **javac Programa.java**.

Assim, bastará correr `make` (ou `make Programa.class`) para produzir o programa executável. Correr o `make` de novo não vai re-executar a receita, porque o alvo é mais recente que o pré-requisito. Vai simplesmente produzir a mensagem abaixo.

```
make: 'Programa.class' is up to date.
```

Só depois introduzir alguma alteração ao ficheiro `Programa.java` é que o comando `make` voltará a ter efeito.

Escrever receitas de compilação é simples, no entanto têm de ser respeitadas regras e boas práticas.

- A receita tem de ser indentada com TAB. Convém usar um editor de texto que preserve esses caracteres.
- A receita pode estar vazia. Nesse caso, o `make` atualiza os pré-requisitos da regra e nada mais. (Isto é frequente em *phony targets*.)
- Se uma linha for demasiado longa, pode ser dividida utilizando o carácter \.
- Podem ser utilizadas variáveis nas receitas através da sintaxe `$(VARNAME)`.
- Se uma instrução da receita começar com o carácter @, é executada normalmente, mas não é apresentada pelo `make`. Por exemplo `@echo LABI` irá apresentar a mensagem `LABI`, mas a instrução em si não é apresentada.
- As receitas são executadas de forma sequencial. Se uma instrução falhar, a execução da receita é abortada. Pode evitar-se isso se a instrução for precedida de um símbolo -; mesmo que falhe, as restantes instruções serão tentadas.
- É possível executar receitas de que invocam outras receitas, mesmo que estas estejam noutras directórios.

O exemplo seguinte ilustra alguns dos aspetos descritos.

```
regra-vazia: Programa.class

Programa.class: Programa.java
    @echo \
        "Compilar ficheiro"
    -rm -f *.class      # avança, mesmo que falhe
    javac Programa.java
```

Alvos falsos

Uma regra pode ter um alvo que não corresponde a um ficheiro. Diz-se que é um *alvo falso* ou *phony target*. É meramente um nome que pode ser usado para invocar explicitamente uma certa receita. Como a receita não vai criar um ficheiro com esse nome, será executada sempre que o alvo for invocado. Para garantir que o aparecimento casual de um ficheiro com o mesmo nome do alvo não impede que esse alvo seja invocado, pode declarar-se explicitamente que se trata de um alvo falso usando a sintaxe:

```
.PHONY: alvo-falso
```

É muito usual os makefiles terem pelo menos dois alvos falsos, que se tornaram standards de facto:

all Executa um processo completo. Por exemplo a compilação de todos os ficheiros da aplicação e a geração do ficheiro executável final.

clean Limpa todos os ficheiros temporários ou auxiliares, deixando apenas ficheiros de código fonte.

Por exemplo, poderíamos reescrever o makefile como se segue.

```
.PHONY: all clean

all: Programa.class

Programa.class: Programa.java clean
    @echo "Compilar ficheiro"
    javac Programa.java

clean:
    rm -f *.class
```

Repare que o alvo **all** depende de todos executáveis do projeto (neste caso apenas um) e nem precisa de receita. Foi colocado como primeira regra, para ser o alvo por defeito.

O alvo `clean` limpa todos os ficheiros `.class` e neste caso também foi colocado como pré-requisito da regra anterior para forçar uma limpeza antes da compilação.

Exercício 6.2

Dentro do directório `tema6`, crie um sub-directório chamado `doc-latex`.

Neste directório coloque um ficheiro L^AT_EX muito simples chamado `doc.tex`.

Crie uma `Makefile` com alvos para compilar tudo e para limpar os ficheiros temporários.

Verifique que consegue invocar os alvos e confira os resultados.

Exercício 6.3

Altere a `Makefile` que criou anteriormente de forma a aproveitar o mecanismo de resolução de dependências. Neste caso, a compilação principal deverá depender de um alvo que cria os índices (`pdflatex nome.tex`) e de outro que compila a bibliografia (usando `bibtex`). Insira uma pequena bibliografia para testar.

Crie uma dependência na compilação para a limpeza dos ficheiros auxiliares e verifique que funciona.

Exercício 6.4

Crie uma regra chamada `publish` que envie (utilizando `scp`) o documento produzido para a sua área no servidor `xcoa.av.it.pt`.

Tenha em atenção que esta regra só deverá ser executada caso a compilação tenha sido previamente executada.

Podemos aproveitar a capacidade do `make` detetar a modificação de ficheiros para invocar

a compilação apenas se algum ficheiro com código fonte for alterado.

Exercício 6.5

Sabendo que o ficheiro `.bib` é convertido para um ficheiro `.bb1` depois de um ficheiro `.aux` ser criado, que o documento final depende do ficheiro `.tex`, dos índices (ex. `.toc`) e da bibliografia (`.bb1`), re-escreva as regras da makefile de forma a utilizarem o nome dos ficheiros e assim usufruir da detecção de alterações.

Verifique que necessita de alterar o ficheiro `.tex` para que a compilação seja repetida.

Uso de variáveis

É possível definir e usar variáveis numa makefile. Pode usar-se esta facilidade para evitar repetição de listas em múltiplas regras. O exemplo que se segue atribui uma lista de alvos à variável `objects`, usa o seu valor como lista de dependências do alvo `all` e como lista de ficheiros a remover na receita de `clean`.

```
objects = A.class B.class C.class

all: $(objects)

clean:
    rm -f $(objects)
#...
```

Também existem variáveis especiais cujo valor é atribuído automaticamente em cada regra avaliada pelo *make*. Chamam-se *variáveis automáticas* e as mais importantes são:

- `$@` representa o alvo da regra;
- `$<` representa o primeiro pré-requisito;
- `$^` representa a lista de todos os pré-requisitos;
- `$?` representa a lista dos pré-requisitos mais recentes que o alvo;
- `$*` representa a string que coincidiu com o padrão `%` numa regra implícita (ver abaixo).

Por exemplo, a regra seguinte cria um ficheiro `abc` por concatenação de três ficheiros `one`, `two`, `three`, mas evita repetir os nomes dos ficheiros.

```
abc: one two three  
      cat $^ > $@
```

Regras implícitas

É possível definir regras gerais, que se aplicam a ficheiros cujos nomes satisfaçam certos padrões. Chamam-se regras implícitas de padrão e são caracterizadas por conterem um símbolo `%` no alvo. Por exemplo, a regra

```
.class: %.java  
       javac $<           # $< representa o pré-requisito
```

especifica como se pode obter um ficheiro objeto de java (`.class`) por compilação do respetivo ficheiro fonte (`.java`). Com esta makefile, se invocarmos `make A.class` e existir um ficheiro `A.java`, o `make` irá ser executar o comando `javac A.java`.

Sub-Directórios

Frequentemente os código fonte de um programa encontra-se dividido por módulos, ou pacotes, colocados em directórios diferentes. Ora, para compilar todo o programa é necessário compilar todos os seus módulos pela ordem correta. O sistema `make` suporta este processo através de um mecanismo que permite invocar uma makefile a partir de outra makefile.

Exercício 6.6

No directório `tema6` crie um directório `src` e coloque algum código java que possua de outra disciplina.

Crie uma **Makefile** que permita compilar este código através do alvo *all*.

Não se esqueça de criar também o alvo *clean*.

Dada a estrutura criada, deverá ter um directório `tema6` com dois sub-directórios `doc` e `src`. Seria interessante que uma **Makefile** no directório pai pudesse invocar a compilação do código fonte e a geração da documentação. As linhas seguintes demonstram como tal pode ser feito. Neste caso, é necessário utilizar a palavra reservada `.PHONY` para garantir que os alvos são processados apesar de já existirem diretórios com esses nomes.

```
SUBDIRS = doc src

.PHONY: subdirs $(SUBDIRS)

subdirs: $(SUBDIRS)

$(SUBDIRS):
    $(MAKE) -C $@
```

Exercício 6.7

Adapte a estrutura demonstrada anteriormente de forma a que possa propagar a compilação e a limpeza (*clean*) para o código e para a documentação.

6.3 GNU Automake e Autoconf

O *automake* [15] é uma ferramenta que, baseando-se no *make* possibilita a gestão da compilação para programas mais complexos, e combinado com a ferramenta *autoconf* [16] facilita a distribuição dos programas na sua forma de código fonte. Em particular, possibilita que os ficheiros **Makefile** sejam construídos de forma dinâmica, adaptando-se às condições existentes (ex. localização de ficheiros ou programas) no sistema alvo. É ainda útil para que durante esta adaptação ao sistema alvo, seja possível verificar a existência de dependências sem os quais o programa não poderia ser compilado. Por exemplo, para os programas desenvolvidos nas disciplinas iniciais da *UA* é necessário que exista o *Java* na sua versão 1.6. Para os relatórios de Laboratórios de Informática é necessário que exista o *LATEX* com suporte para **pdflatex** e todos os packages declarados no documento.

De uma forma muito simples pode-se considerar que o sistema necessita de dois tipos de ficheiros: Templates para ficheiros **Makefile** com nome **Makefile.am** e uma configuração de configuração com nome **configure.ac**. O primeiro é utilizado para facilitar a escrita de ficheiros **Makefile** complexa e será abordado na Secção 6.3, enquanto o segundo é utilizado para validar a integridade do sistema e existência de todas as dependências e será abordado na Secção 6.3.

GNU Autoconf

O ficheiro **configure.ac** contém uma série de macros que indicam que validações executar e como agir face a esse resultado. Pode considerar que os macros irão efetuar validações no sistema (ex. existência de *Java*) ou determinar parâmetros operacionais (ex. localização do compilador **javac** e sua versão). Da perspectiva de um aluno isto é

extremamente importante pois permite enviar código (p.ex. para o docente) junto com a validação de que o ambiente de avaliação tem os componentes necessário à correta execução.

No início deste ficheiro, terá de ser declarado qual o nome do pacote de software, a sua versão e qual o ficheiro principal de código fonte. O exemplo seguinte demonstra o que seria esperado para um qualquer programa *Java* a realizar como parte do pacote *tema6*. Também é utilizada a macro **AC_PROG_GCJ** pois vamos utilizar uma aplicação *Java*.

```
AC_INIT([tema6], [0.1])
AM_INIT_AUTOMAKE
AC_CONFIG_SRCDIR([src/Foo.java])
AM_PROG_GCJ
```

Após a criação deste ficheiro é necessário inicializar o sistema *autoconf*. Para isso é necessário executar os commandos:

- **aclocal**: Cria uma base de dados local para utilização no projeto. Esta base de dados é construída com base nas macros que se encontram definidas no ficheiro **configure.ac**.
- **automake -a**: Processa os ficheiros **Makefile.am** e gera ficheiros adicionais necessários para a instalação.
- **autoconf**: Gera o script **configure**.

Exercício 6.8

Na raíz da sua área de trabalho crie uma pasta chamada **tema6-auto**. Lá dentro crie dois directórios: **src** e **doc**. No directório **src** coloque um qualquer programa em Java^a.

No directório **tema6-auto** crie um ficheiro chamado **configure.ac** com as macros apresentadas anteriormente.

Execute **aclocal**, **automake -a** e **autoconf**. Verifique que existe um ficheiro chamado **configure**.

Execute o ficheiro **configure**.

^aCaso não tenha um programa disponível, escreva um que imprima uma mensagem fixa tal como “Laboratórios de Informática”

Até agora o script **configure** apenas irá realizar algumas verificações muito básicas e incompletas para qualquer projeto. É necessário adicionar macros de forma a configurar o projeto para que este possa ser produzido no sistema. A sequência de macros a utilizar varia para cada projeto de acordo com as suas necessidades. Neste caso em concreto, será necessário realizar validações em relação à existência de compiladores Java na versão correta e do sistema de produção de documentos **LATEX**.

A macro **AC_CHECK_PROG** permite verificar a existência de programas específicos no sistema e segue a sintaxe seguinte:

```
AC_CHECK_PROG (variável, programa-a-verificar, se-encontrado,  
               se-não-encontrado, caminho, ignorar)
```

Em que:

- **variável**: o nome de uma variável onde se armazena o resultado do teste.
- **programa-a-verificar**: o nome do programa a verificar (ex. **javac**).
- **se-encontrado**: que valor deverá a variável ter caso o programa seja encontrado.
- **se-não-encontrado**: que valor deverá a variável ter caso o programa não seja encontrado (opcional).
- **caminho**: directórios onde procurar (opcional).

- **ignorar**: localizações que se ignoram. Serve para excluir versões antigas ou conhecidas por não funcionarem como necessário.

A variável pode depois ser utilizada numa condição para agir de forma adequada. Aplicado ao caso do **javac** podemos definir:

```
AC_CHECK_PROG(EXISTE_JAVAC,javac,yes)

if test "$EXISTE_JAVAC"; then
    AC_MSG_NOTICE([[Compilador de Java encontrado.]])
else
    AC_MSG_ERROR([[Compilador de Java em falta.]])
fi
```

Repare como a variável \$EXISTE_JAVAC foi utilizada numa condição para informar o utilizador que o compilador foi encontrado (AC_MSG_NOTICE), ou para abortar a configuração com uma mensagem (AC_MSG_ERROR) de erro. Caso o programa **javac** não exista, o resultado deverá ser o seguinte:

```
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking for javac... no
configure: error: Compilador de Java em falta.
```

Exercício 6.9

Utilizando a metodologia anterior escreva no seu ficheiro **configure.ac** todas as verificações que acha necessárias para construir devidamente a documentação e o código fonte Java.

Depois terá de voltar a executar os comandos **aclocal**, **autoconf** e **automake**.

Verifique a execução do novo **configure** gerado.

Além de se saber que existem as aplicações necessárias, também é necessário saber se as aplicações possuem a versão correta, ou são capazes de suportar as funcionalidades pretendidas. Até agora, verificou-se que existe L^AT_EX e Java mas não é sabido se o sistema consegue realmente compilar documentos, ou se a versão de Java é a correta. Determinar a versão de uma aplicação implica utilizar funcionalidades da **bash**, nomeadamente executar comandos e filtrar o seu resultado.

```
AC_MSG_CHECKING([Verificando versão do Java])
JAVA_VERSION=$(java -version 2>&1 |grep "java version" |cut -d '.' -f 2)
AC_MSG_RESULT($JAVA_VERSION)

if test $JAVA_VERSION -lt 6; then
    AC_MSG_ERROR([Necessário Java versão 6])
fi
```

No exemplo anterior o comando **grep** filtra as linhas da execução do comando **java -version** de forma a obter apenas a linha com informação de versão. O comando **cut -d '.' -f 2** separa esta linha pelo carácter “.” e devolve o segundo item. Repare que na comparação não é utilizado o operador “<” como está habituado a utilizar, mas sim o operador “-lt” que possui o mesmo significado: *less than*. Isto é uma particularidade da **bash** que utiliza “-lt” quando os operandos são inteiros e “==” quando os operadores são sequências de caráters (Strings).

Exercício 6.10

Execute o comando **java -version** e analise o que é impresso no ecrã e reconstrua o comando do exemplo.

Adicione ao seu **configure.ac** as verificações necessárias para verificar que possui Java versão 6 e **pdflatex** versão 2.4.

Também é possível verificar pela existência de bibliotecas específicas, ou no caso de se utilizar L^AT_EX a existência de packages. No entanto, para o caso de L^AT_EX ou Java não existem macros que realizem estas funções automaticamente, pelo que se torna necessário utilizar comandos externos.

Para verificar se existe um package de Java pode-se utilizar o comando **kpsewhich nome-da-package.sty** e observar o seu resultado. Para isto recorre-se à macro

`AC_MESSAGE_CHECKING` para indicar uma mensagem de um teste personalizado.

```
AC_MSG_CHECKING(Verificando existencia do package biblatex)
HAVE_BIBLATEX=$(kpsewhich biblatex.sty)
AC_MSG_RESULT($HAVE_BIBLATEX)
if test "x$HAVE_BIBLATEX" == "x"; then
    AC_MSG_ERROR(Falta package biblatex)
fi
```

Esta fórmula para escrever testes pode ser utilizada para qualquer validação que se pretenda. Por exemplo, para verificar se existem ficheiros em localizações específicas.

Exercício 6.11

Adicione testes ao seu `configure.ac` de forma a validar a existência das packages que utiliza.

Também pode ser importante validar se o sistema consegue realmente produzir um produto final pretendido. Neste caso será a documentação e a aplicação. Para isso é possível definir testes que efectivamente compilam pequenos programas de forma a validar o ambiente de compilação. Estes testes podem validar a existência de *packages* ou de *classes*. O exemplo seguinte valida se é possível compilar aplicações Java e se é possível utilizar a o método `exit` da classe *java.lang.System*.

```
# Preparar teste
AC_MSG_CHECKING(Verificando possibilidade de compilar programas Java)
cat <<__EOF__ >conftest.java [
import static java.lang.System.*;

public class conftest {
    public static void main(String[] args) {
        exit(0);
    }
}]
__EOF__

# Compilar
javac conftest.java

# Testar resultado
if test $? = 0; then
```

```
    AC_MSG_RESULT([ok])
else
    AC_MSG_ERROR([ERRO])
fi

# Limpar ficheiros produzidos.
rm -f conftest.java conftest.class
```

Exercício 6.12

Construa testes de forma a verificar se o sistema é capaz de gerar documentos com os packages necessários e compilar aplicações Java.

Por fim, caso todos os testes tenha completado corretamente, é necessário gerar os ficheiros **Makefile** para compilar o programa. Embora não tenha sido abordado neste guião, é possível adaptar os **Makefile** de acordo com o que foi sendo detetado na execução do script **configure**.

```
AC_CONFIG_FILES([
    Makefile
    src/Makefile
    doc/Makefile
])
AC_OUTPUT
```

Exercício 6.13

Adicione ao seu ficheiro **configure.ac** a indicação para se gerarem os ficheiros **Makefile** necessários.

Verifique que ele o tenta fazer. Tenha em atenção que ainda não deu indicações de como construir estes ficheiros.

GNU Automake

A ferramenta *automake* encaixa na *autoconf* permitindo definir ficheiros **Makefile** de uma forma mais simplificada e adaptável ao sistema alvo. Neste caso são utilizados ficheiros **Makefile.am**, que a ferramenta converte para ficheiros **Makefile.in**. O script **configure** irá utilizar estes ficheiros para gerar os ficheiros **Makefile** finais.

Para o exemplo seguido é necessária a criação de um ficheiro **Makefile.am** na raíz do projeto indicando que sub-diretórios existem, seguido de ficheiros **Makefile.am** em cada directório. O conteúdo do ficheiro **Makefile.am** na raíz do projeto seria:

```
SUBDIRS = src doc
```

De realçar que a ferramenta *automake* também necessita que existam vários ficheiros de texto na raíz do projeto. Estes ficheiros não são necessários para os testes ou a compilação mas são obrigatórios de existir a quando da utilização desta ferramenta.

Exercício 6.14

Crie um ficheiro **Makefile.am** na raíz do seu projeto e outros em cada um dos directórios. Estes podem ser vazios.

Execute o comando **automake** e crie os ficheiros em falta. O conteúdo não é relevante. No final deverá obter vários ficheiros **Makefile.in** e o **automake** não deverá mostrar qualquer erro.

Verifique que o script **configure** gera ficheiros **Makefile** e inspecione o seu conteúdo.

Depois de existirem ficheiros **Makefile.am** o script **configure** irá gerar ficheiros **Makefile** com bastante informação e uma grande panóplia de alvos. Para além dos típicos *all* e *clean*, outros como *distclean* ou *install* são igualmente criados.

Exercício 6.15

Verifique que alvos são criados. Verifique por exemplo qual a utilidade do alvo *distclean*.

Exercício 6.16

Em cada um dos ficheiros **Makefile.am** adicione regras para os alvos *all* e *clean*, de forma a compilar e gerar documentação.

Na raíz do projeto execute **make**. Que observou?

Uma funcionalidade de um dos alvos é a de criar um pacote *.tar.gz* para distribuição imediata, contendo todo o programa e documentação³¹. Para isto é apenas necessário que se declarem os ficheiros a incluir neste pacote. Neste exemplo, para incluir o código fonte será necessário adicionar a seguinte informação ao ficheiro **src/Makefile.am**:

```
bin_PROGRAMS = foo
foo_SOURCES = Foo.java

%.class: %.java
    javac $*.java

foo : Foo.class

all: foo
```

Exercício 6.17

Corrija os seus ficheiros **Makefile.am** de forma a que ao executar **make dist** seja incluído todo o projeto num ficheiro *.tar.gz*.

Para incluir a documentação, outro directório ou ficheiro qualquer, é necessário que seja definida a variável **EXTRA_DIST** no ficheiro **Makefile.am** principal. Neste caso:

```
EXRA_DIST = doc/doc.tex
```

Exercício 6.18

Adicione o directório de documentação à lista de directórios a incluir no ficheiro *.tar.gz*.

Invoque **make dist** e verifique que funciona.

³¹Para verificar o conteúdo execute: **tar -ztf nome-do-ficheiro.tar.gz**

Conceitos elementares de HTML

Objetivos:

- O protocolo HTTP.
- Estrutura, marcas e atributos de um documento HTML.
- Publicação de documentos HTML para um servidor Web.

7.1 Protocolo HTTP

O HTTP(link) é um protocolo de troca de informação multimédia entre um fornecedor de conteúdos (vulgarmente designado por servidor Web) e um apresentador desses conteúdos (dito navegador). O HTTP foi inicialmente concebido para facilitar a divulgação e a consulta de informação científica multimédia, i.e., documentos contendo um ou mais tipos de informação (texto, imagens, filmes, etc.). Mais importante ainda, estes documentos possuíam hiperligações, que mais não são do que referências para locais onde se pode encontrar mais informação relacionada com o assunto associado à hiperligação. Ao grafo, ou teia, formada por todos os documentos interligados por hiperligações deu-se o nome de World Wide Web, vulgarmente abreviado simplesmente para Web.

Pedidos HTTP e URLs

O protocolo HTTP é na sua essência bastante elementar. Cada objeto (item de informação transferido em bloco) é referenciado por um Uniform Resource Identifier (URI)[17](link) e de cada vez que um navegador precisa de obter o conteúdo de um objeto pede-o a um servidor enviando o URI através do protocolo HTTP.

No âmbito do protocolo HTTP, um URI é designado por URL(link). Um URL indica, sem ambiguidades, onde está, na rede *Internet*, e num dado servidor *Web*, o objeto referido pelo URL. Um URL tem a forma `protocolo://máquina[:porta]/local` e a seguinte interpretação:

- O protocolo indica o modo como o objeto será transmitido entre o servidor e o navegador (*browser*). Exemplos dos protocolos mais usados são HTTP, File Transfer Protocol (FTP)[18](link), etc. Existem ainda algumas variações seguras destes, nomeadamente o HyperText Transfer Protocol Over TLS (HTTPS)[19](link).
- A máquina indica o nome do servidor que irá fornecer o objeto.
- A porta (um número) é um campo opcional que permite parametrizar a forma como o protocolo deve ser usado para comunicar com o servidor. Por omissão o número da porta é fixo por cada protocolo: 80 para HTTP, 21 para FTP, 443 para HTTPS, etc.
- O local indica um caminho, seguindo o paradigma *UNIX* — sucessão de nomes de directórios separados por barras para diante (/), terminadas por um nome de ficheiro — para o local onde está o objeto dentro do servidor. Este caminho pode ser absoluto ou relativo.

Usando como exemplo o URL `http://www.ua.pt/PageText.aspx?id=151`, o acesso a um objeto pelo navegador segue então os seguintes passos. Primeiro, o navegador estabelece uma ligação com o servidor indicado no URL (localizado na máquina com o nome `www.ua.pt`) tendo em conta o protocolo indicado (HTTP). Depois, usando o protocolo HTTP, pede ao servidor o objeto `PageText.aspx?id=151`, o qual lhe será enviado na íntegra.

Resposta do Servidor

A resposta criada por um servidor *Web* pode ser de vários tipos. O tipo é indicado nos meta-dados da resposta, em texto legível, num campo designado por **Content-Type**. Cabe à aplicação cliente (ex. um navegador) decidir o que fazer com o tipo da informação recebida. Os tipos mais vulgares são objetos do tipo `text/html` para apresentação de

páginas Web, `image/gif`, `image/jpeg` ou `image/png` para imagens, etc.

Exercício 7.1

Lance a aplicação **wireshark**, que permite observar o tráfego de rede. (Isto pode requerer permissões de superutilizador.) Inicie a captura de tráfego observável na interface de rede cablada. Na caixa de filtragem do tráfego escreva `http.content_type`, para apenas observar tráfego HTTP que inclua este atributo. Navegue para a página referida anteriormente e observe os tipos de objetos devolvidos.

Parâmetros num URL

Quando se faz um pedido de um objeto a um servidor Web, usando para esse efeito o URL do objeto pretendido, pode-se incluir no pedido um conjunto de parâmetros que de alguma forma condicionam o modo como o servidor irá responder. Por exemplo, pode-se incluir uma indicação da língua em que a resposta do servidor deverá ser fornecida.

Os parâmetros são codificados em blocos com o formato `nome=valor` que são acrescentados ao URL com o carácter `?` e separados entre si com o carácter `&`. Por exemplo o URL `http://www.ua.pt/ensino/PageText.aspx?id=13658&ref=ID0EFCA` usa dois parâmetros: `id` (com o valor `13658`) e `ref` (com o valor `ID0EFCA`). Estes valores vão condicionar a resposta do servidor.

Exercício 7.2

Copie o URL acima referido para um navegador (ou clique nele) e acceda ao objeto que o mesmo referencia. Altere ligeiramente os valores dos parâmetros, acceda novamente ao objeto, veja o efeito e tire conclusões.

7.2 Documentos HTML

Uma resposta do tipo `text/html` enviada por um servidor HTTP é um objeto formatado segundo as regras do padrão HyperText Markup Language (HTML)[20](link). Um objeto HTML é um documento legível por humanos (bem, em geral é...), escrito usando os caracteres que normalmente usamos em documentos de texto (letras, algarismos, sinais de pontuação, parêntesis, etc.). O HTML permite marcar conteúdos com atributos, dando-lhes desta forma um significado que deverá ser convenientemente interpretado pelo navegador, ao qual cabe a sua apresentação ao utilizador.

Abaixo está representado um extrato de um objeto HTML (página).

```
<html>
  <head>
    <title>Cabeçalho do documento HTML</title>
  </head>
  <body>
    Corpo do documento HTML
    <br/>
    Exemplos de texto em <b>negrito</b>, em <i>itálico</i>,
    em estilo <tt>máquina de escrever</tt>.<br/>
  </body>
</html>
```

Exercício 7.3

Crie um documento com a extensão `.html` com um editor de texto. Nesse documento coloque o conteúdo acima indicado. Abra o ficheiro criado com um navegador (clique com o rato no *icon* do ficheiro ou use o URL `file:///.../NomeDoFicheiro.html`, substituindo obviamente ... pelo caminho desde a raiz do sistemas de ficheiros até ao diretório onde o ficheiro pretendido se situa). Veja o resultado e interprete-o à luz do conteúdo do ficheiro. As letras com acentos aparecem corretamente?

Como o servidor não conhece a codificação de caracteres usada por omissão pelo navegador, é aconselhado indicar explicitamente a codificação usada no documento.

Exercício 7.4

Acrescente a linha

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

ao documento HTML logo a seguir a `<head>`. Refresque a página no navegador para ver o resultado desta alteração.

Elementos HTML e *tags*

Como se viu no exemplo anterior, um documento HTML é um documento de texto com marcas que dão indicações ao cliente de HTML (tipicamente, um navegador) de como deve interpretar os conteúdos. Estas *marcas* (*tags*) delimitam *elementos* que definem a estrutura do documento. Por exemplo, `negrito` é um elemento HTML. É delimitado por uma *marca de início* (``) e uma *marca de fim* (``), e tem um *conteúdo*

(o texto “negrito”). Este tipo de elemento indica que o seu conteúdo deve ser apresentado em negrito. Existem muitos tipos de elemento, sempre delimitados por marcas de início e de fim com a sintaxe `<nome>` e `</nome>`, respetivamente. No entanto, alguns tipos de elemento não têm conteúdo e são indicados por uma marca isolada. Por exemplo, o elemento que força uma mudança de linha é indicado pela marca `
`.³²

As marcas podem ser escritas indiscriminadamente com letras maiúsculas ou minúsculas. No entanto, é desaconselhada a utilização de um estilo misto (maiúsculas e minúsculas), dando-se preferência às minúsculas.

Os elementos HTML podem incluir outros elementos no seu conteúdo, formando assim uma estrutura hierárquica (em árvore). Na verdade, o elemento `` que vimos acima está contido num elemento `<body>`, que está contido no elemento `<html>`, que é o elemento raiz do documento HTML.

Os navegadores são bastante tolerantes e são capazes de interpretar HTML mal formatado, i.e. sem algumas marcas de terminação que deveriam existir. A omissão de marcas de terminação pode ou não ter efeitos na apresentação dos conteúdos; tal depende da marca concreta que foi omitida. Pode verificar se uma página não tem erros de formatação HTML através do serviço <https://validator.w3.org>.

Exercício 7.5

Edite o documento HTML anterior com um editor de texto. Experimente apagar uma das marcas de fim de elemento, grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões. Repita este exercício para todas as marcas de fim de elemento, repondo as que retirou antes de experimentar retirar outras.

Comentários

Os documentos HTML podem possuir comentários, que não são apresentados por um navegador. Os comentários são um bloco de texto iniciado por `<!--` e terminado por `-->`. Servem para o autor do documento assinalar algo digno de registo num determinado local, ou simplesmente remover (sem apagar) partes do conteúdo útil do documento. Os elementos contidos dentro de um bloco de comentário não são processados. Porém, os comentários são enviados para o navegador e podem ser consultados pelos utilizadores. Ou seja, ao contrário de documentos criados em Java ou L^AT_EX, em que os comentários são

³²Ver <http://www.w3schools.com/tags/default.asp> para consultar todas as marcas existentes.

omitidos do objeto final, em HTML os comentários fazem parte integrante do documento.

Exercício 7.6

Edite o documento HTML anterior. Experimente comentar uma parte do documento, grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões. Faça este exercício colocando os vários comentários, ou deslocando os comentários de sítio. Verifique o que acontece quando coloca comentários totalmente dentro de comentários (encaixados, *nested*) ou comentários parcialmente sobrepostos entre si (*overlapped*). Muitos navegadores permitem mostrar o texto original (não formatado) do documento HTML. Descubra como é que isso se faz nos navegadores que tem à sua disposição.

Atributos de Marcas

Algumas marcas permitem a sua parametrização.³³ Esta é feita indicando, após o nome da marca, o nome de um atributo e o valor que se lhe quer dar:

```
<marca atributo1="valor" atributo2="valor"></marca>
```

ou (o valor do atributo pode ser delimitado por plicas ou por aspas, ou, se só contiver letras ou só contiver números, nem precisa de ser delimitado)

```
<marca atributo1="valor" atributo2="valor"></marca>
```

Por exemplo, a marca **<p>** serve para delimitar parágrafos e estes podem ter um determinado alinhamento das suas margens: à esquerda (**left**), ao centro (**center**), à direita (**right**) ou a ambas (**justify**). Para indicar o alinhamento deverá ser usado o atributo **align**.

Exercício 7.7

Edite o documento HTML anterior. Acrescente vários parágrafos ao documento. Experimente usar a marca de delimitação de parágrafos para alterar o alinhamento de cada um deles. Grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões.

³³ver <http://www.w3schools.com/tags/default.asp> para visualizar a lista de atributos de cada marca.

Estruturação de documentos

Os documentos, ou páginas Web, normalmente possuem uma estrutura gráfica que facilita a sua leitura. Por outras palavras, não se limitam a ser uma sucessão contínua de texto.

Parágrafos e mudanças de linha

As mudanças de linha num documento HTML não provocam necessariamente uma mudança de linha no resultado apresentado pelo seu interpretador. A forma correta de separar parágrafos diferentes é através das marcas de início e fim dos mesmos, `<p>` e `</p>`, respetivamente. Alternativamente, pode-se usar a marca `
` para forçar uma mudança de linha (*line break*), mas o resultado visual é diferente porque não é criado um espaço extra entre parágrafos que facilita a sua leitura.

Exercício 7.8

Edite o documento HTML anterior. Experimente delimitar parágrafos com as marcas apropriadas, ou simplesmente forçar mudanças de linha. Grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões.

Capítulos e secções

Nos documentos é normal usar-se capítulos e secções para separar o texto em blocos coerentes. Em HTML esta separação pode ser feita através de cabeçalhos (*headings*). Os cabeçalhos colocam texto com uma dimensão diferente do texto ordinário e mudam também o tipo de letra para negrito. No entanto, não incluem qualquer numeração automática.

Os cabeçalhos indicam-se com as marcas `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>` ou `<h6>`, e são terminados com as marcas `</h1>`, ..., `</h6>`, respetivamente. Cada uma destas marcas produz um cabeçalho com tamanho de letra diferente, maior com `h1`, menor com `h6`.

Exercício 7.9

Edite o documento HTML anterior. Acrescente vários cabeçalhos de diferentes dimensões. Grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões.

Exercício 7.10

Utilize as marcas de cabeçalho para identificar cada um dos tópicos a tratar neste guião prático. Manter este ficheiro organizado será de extrema utilidade para as próximas aulas.

Listas de itens

Outra forma de estruturar um documento é através de listas de itens. As listas podem ser numeradas ou não, consoante isso seja relevante para a compreensão do texto. As listas são criadas com uma marca de início de lista e terminadas com uma marca de fim de lista. Estas marcas enquadram um bloco onde todos os conteúdos são colocados num elemento da lista. As marcas de início e fim de lista numerada são `` e `` (de *ordered list*); as marcas de início e fim de lista não numerada são `` e `` (de *unordered list*).

Dentro de uma lista cada um dos elementos (ou itens) é delimitado através das marcas `` e `` (de *list item*). A omissão da marca de fim de item não é normalmente um problema (tente explicar porquê).

```
<ul>
  <li> Este é o primeiro item da lista;</li>
  <li> Este é o segundo item da lista.</li>
</ul>
```

Exercício 7.11

Edito o documento HTML anterior. Acrescente várias listas numeradas e não numeradas ao documento, de diferentes dimensões (número de itens). Grave o ficheiro e recarregue-o com o navegador.

Exercício 7.12

Consulte a página http://www.w3schools.com/tags/tag_li.asp para saber que atributos deve usar para mudar a forma de numeração das listas numeradas. Experimente mudar as que criou.

Tabelas

A apresentação de dados em tabelas envolve alguma complexidade visual, mas os princípios são simples. Uma tabela é uma matriz formada por linhas e cada linha contém células.

Cada célula é denominada um dado da tabela (*table data*) e delimita-se através das marcas `<td>` e `</td>`. As células agrupam-se em linhas (*table rows*), as quais são delimitadas por marcas `<tr>` e `</tr>`. Finalmente, uma tabela constroi-se com uma sucessão de linhas delimitadas pelas marcas `<table>` e `</table>`. Por exemplo, o código abaixo cria a Tabela 7.1.

```
<table border=1>
  <tr><td> a </td><td> b </td></tr>
  <tr><td> c </td><td> d </td></tr>
</table>
```

a	b
c	d

Tabela 7.1: Exemplo de tabela.

Exercício 7.13

Edite o documento HTML anterior. Crie uma tabela com várias linhas e colunas. Grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões.

Frequentemente, as células da primeira linha e/ou da primeira coluna das tabelas possuem uma apresentação gráfica diferente das restantes. Em HTML isto pode ser indicado utilizando a marca `<th>` em vez da marca `<td>`.

O código HTML abaixo cria a Tabela 7.2.

```
<table border=1>
  <tr><th> c1 </th><th> c2 </th></tr>
  <tr><td> a </td><td> b </td></tr>
  <tr><td> c </td><td> d </td></tr>
</table>
```

c1	c2
a	b
c	d

Tabela 7.2: Exemplo de tabela com cabeçalho.

Exercício 7.14

Edito o documento HTML anterior. Crie uma tabela com várias linhas e colunas, assim como um cabeçalho. Grave o ficheiro e recarregue-o com o navegador. Veja o efeito e tire conclusões.

Se tivermos elementos da tabela que ocupem mais do que uma linha, ou mais do que uma coluna, isso é feito indicando nas células em causa o número de linhas (**rowspan**) ou de colunas (**colspan**) que ocupa. Por exemplo, a tabela 7.3 será criada pelo código HTML abaixo.

```
<table border=1>
<tr><th> c1 </th><th> c2 </th><th> c3 </th><th> c4 </th></tr>
<tr><td> a </td><td colspan=2> b </td><td rowspan=2> c </td></tr>
<tr><td> d </td><td> e </td><td> f </td></tr>
</table>
```

c1	c2	c3	c4
a	b		c
d	e	f	

Tabela 7.3: Exemplo de tabela com células que ocupam mais do que uma coluna ou mais do que uma linha.

Exercício 7.15

Edito o documento HTML anterior. Crie uma tabela com várias linhas e colunas. Inclua algumas células que ocupem mais do que uma linha, mais do que uma coluna, ou ambas. Grave o ficheiro e recarregue-o com o navegador.

Exercício 7.16

Consulte a página http://www.w3schools.com/tags/tag_table.asp para saber que atributos deve usar para mudar o alinhamento do texto nas células ou a cor do fundo da tabela ou de cada célula. Experimente mudar as células que criou usando esses atributos.

Hiper-referências

Uma hiper-referência ou hiper-ligação (*link*) é uma referência a outro objeto feita por um determinado conteúdo de um documento HTML. A forma como se indica uma hiper-referência é a seguinte:

```
<a href="URL">texto associado à hiper-referência</a>
```

Uma hiper-referência surge normalmente com uma coloração diferente e permite saltar de documento em documento quando selecionada (normalmente com o rato).

Exercício 7.17

Crie dois documentos HTML que se referenciem mutuamente usando hiper-referências. No URL das hiper-referências use o protocolo **file** (ex. **file:ficheiro**) ou, simplesmente, omita o protocolo e use o nome do ficheiro. Abra um dos documentos com o navegador e experimente mudar de um para outro usando as hiper-referências que criou.

De acordo com o modelo de interação definido para a página Web pode ser proveitoso definir o que acontece quando uma determinada hiper-referência é invocada. Por exemplo, recursos externos devem abrir numa nova janela, enquanto a navegação na mesma página deve reutilizar a janela atualmente utilizada. Isto é conseguido através do atributo **target**, no formato seguinte:

```
<a href="URL" target="destino">texto associado à hiper-referência</a>
```

O atributo **target** pode possuir vários valores, de entre os quais:

_blank: Abre a hiper-referência numa nova janela.

_self: Abre a hiper-referência na mesma janela (valor por definição).

_top: Abre a hiper-referência na mesma janela mesmo que a página atual seja um fragmento (**frame**) de uma outra página.

Exercício 7.18

Altere os documentos anteriormente criados de forma a que a navegação se processe na mesma página. Adicione referências para URLs externos, mas utilizando uma nova janela.

Imagens

Um documento HTML pode incluir referências a imagens que devem ser apresentadas juntamente com o texto. A forma como se indica a inclusão de uma imagem é a seguinte:

```
</img>
```

onde URL é a localização da imagem, i.e., o local onde a mesma pode ser obtida. Esta marca pode ser refinada com vários atributos, como a dimensão da imagem apresentada (atributos **width** e **height**).

Exercício 7.19

Edite um documento HTML usado anteriormente e altere-o para incluir a imagem <http://www.ua.pt/images/40anos.png>. Abra o documento com um navegador e observe o resultado.

Ao construir páginas Web deve-se ter em consideração a navegação em dispositivos com capacidade reduzida, como por exemplo os navegadores em modo de texto. Todo o conteúdo textual pode ser representado facilmente em qualquer navegador, mas o mesmo não é verdade para as imagens. Também é possível que a imagem não possa ser apresentada corretamente (ex., o ficheiro encontra-se em falta). Neste caso é possível definir um texto alternativo para cada imagem, que é sempre apresentado, através do atributo **alt**. A forma como se define o texto alternativo para uma imagem é:

```
</img>
```

Na realidade, segundo a norma HTML a utilização do atributo **alt** é obrigatória!

Exercício 7.20

Edite o documento anterior e adicione atributos **alt** a todas as imagens indicando a que se refere a imagem.

Adicione outras marcas ****, com atributo **alt**, para imagens não existentes, e verifique o resultado.

Divisões

Frequentemente é necessário organizar os conteúdos na página em blocos ou divisões. Até agora os elementos foram abordados de forma isolada, mas na realidade eles raramente são utilizados de forma isolada, pertencendo a uma dada divisão da página. A utilização de divisões é útil para posicionar o conteúdo de acordo com a sua função e assim melhorar a apresentação. A Figura 7.1 demonstra uma organização típica de uma página Web, que é muito utilizada na *Internet*. Um exemplo desta organização é a página do Sapo.pt. Outras páginas utilizam uma organização diferente, mas é sempre possível analisar uma dada página e identificar claramente que o conteúdo está organizado em divisões.

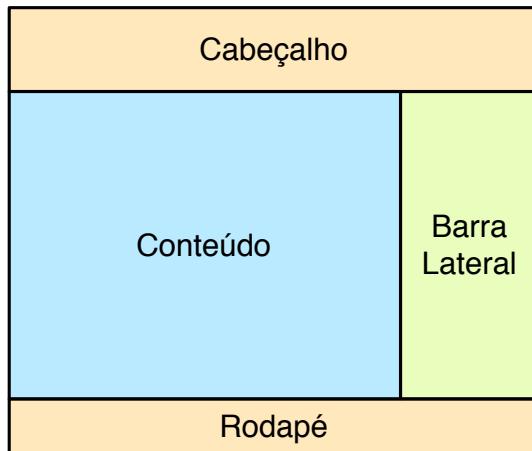


Figura 7.1: Organização típica de uma página Web.

Exercício 7.21

Analise as páginas seguintes e identifique qual a organização de divisões utilizada.

- <http://www.ua.pt>
- <http://code.ua.pt>
- <http://www.publico.pt>
- <https://news.ycombinator.com>

As divisões também podem ser utilizadas para posicionar corretamente conteúdo dentro de um bloco principal da página. É frequente fazer isto para ter um maior controlo sobre a disposição dos conteúdos ou para adaptar a página a dispositivos móveis.

Independentemente da função, as divisões são representadas em HTML da seguinte forma:

```
<div>
  conteúdo
</div>
```

A utilização de divisões, especialmente as divisões principais de uma página, implica a definição da largura para cada divisão. Uma divisão sem largura definida irá ocupar apenas o necessário para apresentar o seu conteúdo. Felizmente podem ser utilizadas medidas de largura percentuais, tais como 100%.

Exercício 7.22

Crie um documento HTML de forma que tenha um cabeçalho, corpo e rodapé. No cabeçalho coloque as 4 imagens que se encontram nos endereços http://banners.ua.pt/Banners/banner_7_52_X.jpg, em que X é um número entre 1 e 4.

Poderá ter de ajustar os atributos **width** e **height** para obter a organização pretendida. De forma a visualizar os limites das divisões, pode utilizar o atributo **style="border: 1px solid black;"**

Exercício 7.23

Verifique que pode modificar o estilo de um elemento, em particular uma divisão, através do atributo **style**. Cada valor deste atributo tem o formato **nome:valor;**, sendo que poderá utilizar nomes tal como **color**, **background-color** ou **font**.

Uma capacidade extremamente útil das divisões é que elas podem ser flutuantes, sendo posicionadas de forma controlada na página. Através desta funcionalidade é possível mostrar conteúdos de forma dinâmica, como é exemplo um *popup* (ver Figura 7.2). Isto consegue-se através da manipulação dos atributos **position**, **top** e **left**, da seguinte forma:

```
<div style="position: absolute; left: 100px; top: 150px;"></div>
```

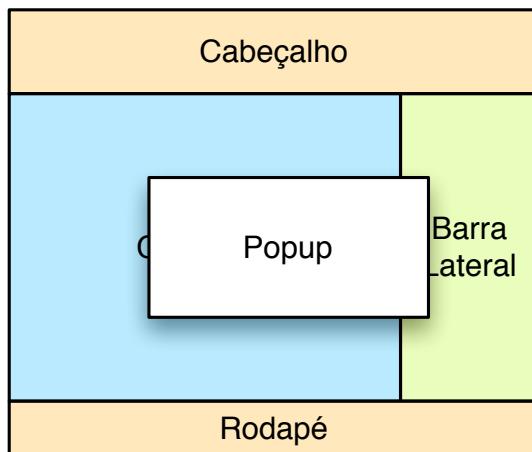


Figura 7.2: Utilização de uma divisão para implementar um *popup*.

Neste exemplo, a divisão terá posicionamento absoluto, e será colocada a **150px** do topo da página, e a **100px** da margem esquerda da página. Podem ser utilizados outros valores para o atributo **position**, nomeadamente:

absolute: Posição da divisão é determinada com base no canto superior esquerdo da página.

relative: Posição da divisão é determinada com base na localização esperada da divisão (onde é declarada).

static: O valor por omissão. A divisão é colocada onde declarada.

fixed: Posição da divisão é determinada com base no canto superior esquerdo da janela.
Neste caso, a divisão irá permanecer estática na janela.

inherit: Usado o valor do elemento pai.

Exercício 7.24

Na página que criou com divisões, adicione uma divisória com um posicionamento explícito e verifique o efeito dos vários valores do atributo **position**.

7.3 Utilização de um servidor HTTP

Neste último passo o objectivo será colocar ficheiros HTML com conteúdos estáticos criados neste guião num servidor HTTP.

Para este efeito, iremos usar o servidor **xcoa.av.it.pt**. Este servidor possui várias contas pessoais, com nomes do tipo **labi-tXgY**, onde X é o número de turma e Y o número de grupo (usar o número da bancada). Cada grupo pode iniciar uma sessão nesta máquina usando o nome de utilizador apropriado, tendo em conta a sua turma e grupo. Inicialmente, a senha é igual ao nome de utilizador. Para iniciar uma sessão deverá ser usado o comando:

ssh -l <user> xcoa.av.it.pt,
onde <user> é um nome do tipo **labi-tXgY**.

Recomenda-se que após a primeira ligação, a senha seja alterada através do comando **passwd.**

Uma vez criada esta sessão, os ficheiros HTML devem ser colocados numa diretoria **public_html** que deverá ser criada abaixo da diretoria pessoal (que pode ser referida, indiretamente, com o símbolo **~**), ou seja:

mkdir ~/public_html.

Após a colocação dos ficheiros HTML nesta diretoria, os seus conteúdos poderão ser acedidos usando o URL **http://xcoa.av.it.pt/~labi-tXgY**.

Para transferir os ficheiros, pode ser utilizado o comando **scp**, mas outro processo comum é sincronizar a diretoria com um repositório de código, tal como o disponível na plataforma

Code.UA.

Exercício 7.25

No seu computador, obtenha uma cópia do seu repositório na plataforma **Code.UA** através do comando `git clone endereço-do-repositorio`. No diretório criado, adicione um sub-directório chamado `html` e copie para aqui as páginas criadas. Adicione os ficheiros ao repositório e envie as alterações para o repositório remoto.

No servidor `xcoa.av.it.pt` dirija-se ao diretório `~/public_html` e obtenha o repositório do servidor **Code.UA**.

Verifique que pode aceder aos conteúdos que criou através do endereço
`http://xcoa.av.it.pt/~labi-tXgY/html/nome-da-pagina`.

7.4 Para aprofundar

Exercício 7.26

Crie uma página com exemplos de utilização de todas as marcas aqui estudadas. Pode aceder às opções possíveis para cada marca através do endereço `http://www.w3schools.com/tags/default.asp`.

Esta página será muito útil para as aulas futuras.

Exercício 7.27

Explore a utilização de outras marcas além das aqui estudadas. A lista completa encontra-se em `http://www.w3schools.com/tags/default.asp`.

Tenha em atenção que nem todos os navegadores irão suportar todas as marcas.

Exercício 7.28

Construa uma pequena página com todas as marcas que encontrar, de forma a identificar se um determinado navegador suporta a marca e qual o seu comportamento.

Conceitos elementares de CSS

Objetivos:

- Separação entre conteúdo e estilo.
- Sintaxe dos estilos CSS.
- Elementos básicos de CSS.
- Seletores, propriedades e valores.
- Introdução a *Twitter Bootstrap*.

8.1 Introdução

Foi visto no tema anterior que HTML implementa uma separação entre estrutura da página e conteúdo. Isto é realizado através da utilização de marcas que sinalizam qual o significado de um dado pedaço de conteúdo. Seguindo a mesma lógica, Cascading Style Sheets (CSS)[21] é uma linguagem que permite definir o estilo (aparência) de uma página de forma independente do seu conteúdo. Ou seja, a um dado conteúdo é atribuída uma estrutura usando HTML e uma aparência usando CSS. Com CSS, aplicar estilos diferentes para a mesma estrutura e conteúdo torna-se numa tarefa simples.

8.2 Sintaxe e aplicação de estilos

Embora se possa integrar num documento HTML, a definição de estilos por CSS tem uma sintaxe própria, distinta da utilizada pelo HTML. A sintaxe CSS baseia-se em *declarações* com o formato:

```
propriedade : valor;
```

que atribuem *valores* específicos a certas *propriedades*. As propriedades são depois aplicadas a elementos HTML de maneira a definir a sua apresentação. Em [22] estão listadas todas as propriedades que se podem aplicar a cada elemento de HTML e os valores válidos que podem assumir.

Por exemplo, no excerto abaixo, visto na aula anterior,

```

```

a declaração "width: 100px;" segue a sintaxe CSS. Atribui o valor **100px** à propriedade **width**.

Note que a atribuição de valor a uma propriedade é indicada por meio de um carácter **:** (e não **=**) e deve ser terminada por um símbolo **;**. Podem adicionar-se comentários através do par **/*** e ***/**.

Por vezes algumas propriedades aceitam valores em diferentes unidades, sendo obrigatória a definição da unidade a utilizar. Considere que quer definir a largura de uma imagem, utilizando **width: 100**. Ora, não é claro a que se refere o valor 100. Pode-se estar a falar de 100 centímetros, 100 pixels, 100%, etc. Com exceção do valor 0 (que não é ambíguo), é então necessário adicionar um sufixo que indique a unidade a utilizar.

Existem as seguintes unidades:

px: Normalmente é equivalente a 1 pixel no ecrã;

em: Medida de origem na tipografia representando a largura típica de um carácter m no tipo de letra atual;

pt: Um *Point* equivale a $\frac{1}{72}$ de uma polegada. É uma medida muito utilizada em tipografia e os tamanhos de letra tipicamente são definidos em **pt**. Este guião foi realizado usando o tamanho de 11pt para o texto;

%: Percentagem de uma outra medida como a largura da página ou a altura;

pc: Um Pica equivale a 12pt;

cm: Centímetro, equivalente a 10mm;

mm: Milímetro;

in: Polegada, equivalente a 25.4mm;

A utilização de cada medida depende do propósito. Se for pretendido que uma dada imagem tenha um tamanho proporcional ao tipo de letra, podemos usar **pt**. Se por outro lado for pretendido que uma imagem ocupe 50% da página, usa-se percentagem. Medidas como **cm**, **mm**, **in**, **pt** e **pc** são úteis pois permitem que se apresente conteúdo com um tamanho exato, independentemente da resolução utilizada.

Exercício 8.1

Crie um ficheiro HTML e inclua a mesma imagem 8 vezes. Em cada uma das inclusões, defina o atributo **style="width: XX;"** com valores e unidades diferentes.

Existem várias abordagens para incluir estilos num documento, nomeadamente: **em linha**, **no cabeçalho** e **inclusão externa**.

Estilos em linha

As abordagem em linha foi brevemente apresentada na aula anterior quando da utilização do atributo **style** de uma marca HTML. Considerando o exemplo da marca **<h1>**, é possível utilizar CSS de forma a tornar o texto vermelho, utilizando a seguinte combinação de HTML com CSS.

```
<!DOCTYPE html>
<html>
  <body>
    <h1 style="color: red;"> Este texto ficará vermelho </h1>
  </body>
</html>
```

O que resulta na apresentação da Figura 8.1.

Neste caso, a definição de CSS resume-se ao texto dentro do atributo **style**, o restante texto será HTML ou conteúdo. No caso das cores é possível serem representadas de 3 formas:

Nome da cor: algumas cores populares possuem nomes, tais como **red**, **lightred**, **green**, etc. A lista completa pode ser consultada no endereço http://www.w3schools.com/cssref/css_colornames.asp.



Figura 8.1: Simples cabeçalho `<h1>` com um estilo.

Código RGB em hexadecimal: Como as cores se podem ser representar pela combinação das 3 cores primárias, vermelho (Red), verde (Green) e azul (Blue), pode-se definir qualquer cor através do formato `#RRGGBB`. A cor vermelha será `#FF0000`, enquanto um laranja será `#FFA500`.

Código RGB em decimal: Semelhante ao anterior, mas usando números em decimal no formato `rgb(255, 165, 0)`.

Código RGB em percentagem: Semelhante ao anterior, mas usando percentagens, `rgb(100%, 65%, 0%)`;

Exercício 8.2

Crie um documento HTML com várias marcas `<p>`. A cor do texto deverá iniciar no azul e terminar no branco. Utilize o esquema de representação que preferir.

Deve-se mais uma vez salientar que os documentos HTML seguem o princípio da separação entre estrutura, conteúdo e estilo. A inclusão de estilos diretamente na marca é possível, mas desaconselhada!

Exercício 8.3

Crie um documento HTML com marcas `<h1>` a `<h6>` cada tipo de cabeçalho contendo diferentes estilos aplicados, através das propriedades `color` e `background-color`.

A página <http://www.w3schools.com/cssref/default.asp> fornece a lista com todas as propriedades existentes. Experimente outras.

Estilos no cabeçalho

Considerando o caso em que se pretende que a página tenha um aspeto coerente, é útil que se possa definir o estilo de uma forma mais geral. Por exemplo, definir que todas as marcas `<h1>` possuam o mesmo estilo, que é diferente do aspeto das marcas `<h2>`. Utilizando o método de definição de estilo em linha isto é possível, mas entediante, moroso e dado a erros. Uma abordagem mais aconselhada é a definição de estilos gerais no cabeçalho da página. Para isto recorre-se à marca de HTML `style` (não confundir com o atributo `style!`). Dentro da marca `style` define-se o estilo utilizando CSS. O exemplo seguinte demonstra como se pode definir um estilo a aplicar a todas as marcas `<h1>`. Este método é válido para qualquer marca, incluindo o `<body>`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Estilos no Cabeçalho </title>
    <style>
      h1 {
        color: red;
      }
      h2 {
        color: green;
      }
    </style>
  </head>

  <body>
    <h1> Este texto ficará vermelho </h1>
  </body>
</html>
```

O resultado será exactamente igual ao anterior, com a diferença que todos os elementos `<h1>` irão ver o estilo ser aplicado de forma automática.

Repare na sintaxe usada:

```
seletor { propriedade : valor ; [...] }
```

A isto chama-se uma *regra* de CSS. Começa com um (ou mais) *seletores*, que indicam a que elementos se deve aplicar. Depois inclui uma lista de declarações a aplicar, entre

chavetas. Os seletores podem indicar um tipo de elemento HTML, mas também há outras formas de selecionar elementos, como verá mais abaixo.

Exercício 8.4

Copie o ficheiro anterior para um novo e altere-o de forma a aplicar os estilos às marcas `<h1>` a `<h6>` através de um estilo genérico no cabeçalho.

Inclusão de estilos externos

A definição de estilos na forma de regras é a mais poderosa e tem a vantagem de poder ser reutilizada se for colocada num ficheiro externo. Isto aumenta ainda mais a separação entre estrutura, conteúdo e estilo, e tem vantagens para o tempo de carregamento das páginas.³⁴ Esta abordagem faz uso da marca `<link>`, no seguinte formato:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title> Estilos no Cabeçalho </title>
    <link rel="stylesheet" href="css/style.css"/>
  </head>

  <body>
    <h1> Este texto ficará vermelho </h1>
  </body>
</html>
```

Este exemplo presume a existência de um directório chamado `css`, contendo um ficheiro chamado `style.css` com o seguinte conteúdo:

```
h1 {
  color: red;
}

h2 {
  color: green;
}
```

³⁴Os ficheiros só são recarregados pelo browser quando são modificados. Se o estilo estiver num recurso externo, e não for modificado, o browser apenas o irá ler na primeira vez que accede ao site.

Este deve ser o método preferido sempre que se utilizam estilos em páginas. Por vezes, por questões de compatibilidade entre browsers, pode ser necessário recorrer aos outros métodos. No entanto este aspeto não será explorado neste tema.

Exercício 8.5

Copie o ficheiro anterior e altere-o de forma a que os estilos sejam obtidos de um ficheiro externo, não estando nem no cabeçalho, nem diretamente na marca.

8.3 Seletores, identificadores e classes

Aquando da definição de estilos no cabeçalho, foi demonstrado que é trivial definir o estilo de uma dada marca. Levantam-se no entanto problemas se se pretender definir o estilo de forma mais específica. Isto é, aplicar um estilo apenas a um dado elemento, ou a um conjunto de elementos. Como exemplo, considere o caso de uma tabela com séries de temperatura média por ano e mês.

```
<table>
  <tr><th>Nome</th><th>2009</th><th>2010</th><th>2011</th><th>2012</th></tr>
  <tr><td>Jan</td><td>14</td><td>13</td><td>11</td><td>9</td></tr>
  <tr><td>Fev</td><td>15</td><td>14</td><td>14</td><td>11</td></tr>
  <tr><td>Mar</td><td>17</td><td>17</td><td>15</td><td>16</td></tr>
  <tr><td>Abr</td><td>18</td><td>17</td><td>13</td><td>18</td></tr>
  <tr><td>Mai</td><td>19</td><td>18</td><td>16</td><td>21</td></tr>
  <tr><td>Jun</td><td>21</td><td>22</td><td>20</td><td>16</td></tr>
  <tr><td>Jul</td><td>23</td><td>21</td><td>22</td><td>18</td></tr>
  <tr><td>Ago</td><td>24</td><td>23</td><td>24</td><td>23</td></tr>
  <tr><td>Set</td><td>21</td><td>20</td><td>22</td><td>15</td></tr>
  <tr><td>Out</td><td>18</td><td>16</td><td>19</td><td>11</td></tr>
  <tr><td>Nov</td><td>15</td><td>14</td><td>14</td><td>10</td></tr>
  <tr><td>Dez</td><td>13</td><td>11</td><td>12</td><td>8</td></tr>
</table>
```

O resultado é o apresentado na Figura 8.2.

Utilizando os métodos anteriores, é possível definir uma borda para toda a tabela, ou mesmo para cada elemento da tabela.

```
table, td, th {
  border: 1px solid black;
}
```

The screenshot shows a Mozilla Firefox window displaying a table. The table has a header row with columns for 'Mês' and years '2009', '2010', '2011', and '2012'. Below the header, there are 12 rows, one for each month from January to December, with numerical values in each column. The data is as follows:

Mês	2009	2010	2011	2012
Jan	14	13	11	9
Fev	15	14	14	11
Mar	17	17	15	16
Abr	18	17	13	18
Mai	19	18	16	21
Jun	21	22	20	16
Jul	23	21	22	18
Ago	24	23	24	23
Set	21	20	22	15
Out	18	16	19	11
Nov	15	14	14	10
Dez	13	11	12	8

Figura 8.2: Exemplo de uma tabela com séries de temperatura de vários anos

Sem dúvida que a legibilidade da tabela é melhorada, mas poderia ser útil salientar visualmente os meses mais frios ou mais quentes de cada ano. Poderíamos conseguir isto alterando diretamente o atributo **style** de um elemento **<td>**, mas isso é desaconselhado.

O que precisamos é de formas de nos referirmos a um elemento HTML sem ser pelo seu *tipo*. Isto é conseguido através de 2 novos conceitos: o *identificador* e a *classe*.

O identificador (**id**) é um atributo opcional de qualquer marca HTML que permite identificar inequivocamente esse elemento. No exemplo atual, pode-se identificar o mês de dezembro de 2012 como o mais frio e colorir esta célula de forma diferente. Para isto é necessário, em primeiro lugar, atribuir um identificador à marca **<td>** respetiva, e depois definir o estilo para esse identificador.

```

<head>
  <style>
    table, td, th {
      border: 1px solid black;
    }

    #coldest {
      background-color: lightblue;
    }
  </style>
</head>

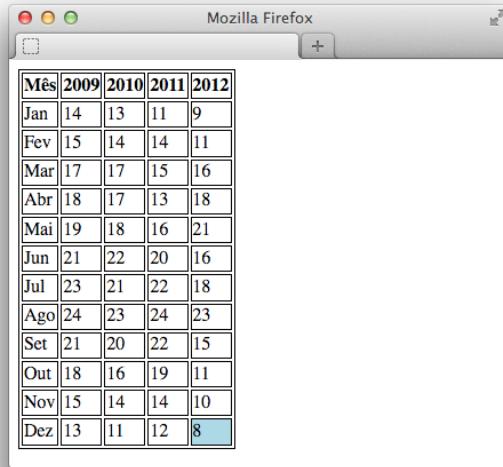
...

<tr><td>Dez</td><td>13</td><td>11</td><td>12</td><td id="coldest"> 8 </td></tr>

```

O resultado é o demonstrado na Figura 8.3.

Repare que o identificador de um elemento é selecionado na regra CSS através da utilização do carácter #.



A screenshot of a Mozilla Firefox window displaying a table. The table has a header row with columns for 'Mês' and years '2009', '2010', '2011', and '2012'. Below this are twelve rows, each representing a month from January to December. The last row, 'Dez', is highlighted with a light blue background, demonstrating the effect of the CSS rule `#coldest { background-color: lightblue; }`.

Mês	2009	2010	2011	2012
Jan	14	13	11	9
Fev	15	14	14	11
Mar	17	17	15	16
Abr	18	17	13	18
Mai	19	18	16	21
Jun	21	22	20	16
Jul	23	21	22	18
Ago	24	23	24	23
Set	21	20	22	15
Out	18	16	19	11
Nov	15	14	14	10
Dez	13	11	12	8

Figura 8.3: Exemplo de uma tabela com um elemento estilizado de forma individual.

Este tipo de aplicação de estilos individuais é muito útil se (entre outras coisas), num menu de navegação de uma página, quiser identificar qual a página atual onde o utilizador

se encontra. No entanto, normalmente utilizam-se classes e não identificadores.

Exercício 8.6

Crie uma página HTML em que, através do atributo **id** especifique o estilo individual de cada elemento. Utilize 2 ou mais elementos do mesmo tipo.

Uma outra alternativa para atribuir estilos é a definição de grupos de elementos, representados pelo atributo **class**. Um estilo aplicado a um grupo é aplicado a todos os elementos que pertençam a esse grupo.

```
<h1 class="cabeçalho-principal">Texto</h1>
```

Considerando o exemplo com a tabela, podem-se atribuir a classes distintas os meses mais frios e mais quentes de cada ano e atribuir um estilo a cada classe.

```
<head>
  <style>
    table, th, td {
      border: 1px solid black;
    }
    #coldest {
      background-color: lightblue;
    }
    .cold {
      background-color: cyan;
    }
    .hot {
      background-color: orange;
    }
  </style>
</head>

...
<tr><td>Jan</td><td>14</td><td>13</td><td class="cold">11</td><td class="cold">9</td></tr>
...
<tr>
  <td>Ago</td><td class="hot">24</td>
  <td class="hot">23</td>
  <td class="hot">24</td>
  <td class="hot">23</td>
```

```

</tr>
...
<tr>
<td>Dez</td>
<td class="cold">13</td>
<td class="cold">11</td>
<td>12</td>
<td id="coldest">8</td>
</tr>

```

O resultado é o demonstrado na Figura 8.4.

Neste caso, repare que a classe de um elemento é selecionada, na regra CSS, através da utilização do carácter . antes do nome da classe.

Mês	2009	2010	2011	2012
Jan	14	13	11	9
Fev	15	14	14	11
Mar	17	17	15	16
Abr	18	17	13	18
Mai	19	18	16	21
Jun	21	22	20	16
Jul	23	21	22	18
Ago	24	23	24	23
Set	21	20	22	15
Out	18	16	19	11
Nov	15	14	14	10
Dez	13	11	12	8

Figura 8.4: Exemplo de uma tabela com um elemento estilizado por classes.

Este tipo de selecção de elementos numa página é sem dúvida o mais utilizado pois permite

criar estilos que redefinem completamente um elemento, com base no seu propósito.

Exercício 8.7

Considere a marca `<input type="button" value="string"/>`. Crie uma página com 2 elementos desta marca, um com o valor “OK” e outro com o valor “Cancelar”. Utilizando classes (ex, “button-ok” e “button-cancel”) aplique estilos diferenciados.

Exercício 8.8

Complete a tabela fornecida aplicando estilos diferenciados às linhas ímpares e pares através de duas classes diferentes. Pode aplicar as classes à marca `<tr>`.

8.4 Margens, Bordas e Espaçamentos

Um estilo CSS opera segundo um modelo de caixas (*Box Model*), em que cada elemento da página ocupa uma *caixa* retangular. Nesta caixa existe um *conteúdo*, onde surge o texto ou imagens do elemento. Em torno do conteúdo pode haver um *espaçamento* (*padding*), depois uma *borda* e finalmente existe a *margem* entre caixas. O espaçoamento e a margem são transparentes. A borda pode ser visível. Este esquema está representado na Figura 8.5.

Uma consequência (entre outras) do modelo de caixas é que todos os elementos em HTML possuem um conjunto de propriedades que permitem alterar a maneira como se apresenta o espaço circundante. Para a margem e espaçoamento estas propriedades são:

margin: Define o tamanho da margem de um elemento no formato **topo direita baixo esquerda** (ex, "10px 10px 10px 10px"). As margens podem ser definidas individualmente utilizando **margin-top**, **margin-right**, **margin-bottom** e **margin-left**.

padding: Define a largura do espaçoamento de um elemento no formato **topo direita baixo esquerda**. O espaçoamento pode ser definido individualmente utilizando **padding-top**, **padding-right**, **padding-bottom** e **padding-left**.

As bordas são um pouco mais complexas pois pode ser necessário definir a sua espessura, o seu estilo e a sua cor, através das propriedades:

border-width: Define a espessura da borda, numa qualquer unidade.

border-style: Define o estilo da borda, podendo ser utilizados os tipos: **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge**, **inset**, **outset**.



Figura 8.5: Modelo de caixas em CSS.

border-color: Define a cor da borda, podendo ser utilizada qualquer cor.

Também é possível combinar as 3 características na propriedade abreviada **border**. Por exemplo, **border: 10px dashed green**. Existem também formas de definir cada aresta independentemente.

O exemplo seguinte define uma caixa do estilo **outset** para um elemento **<div>**.

```
<div style="border: 2px solid black;">
  <div style="border-style: outset; border-width: 10px; margin: 10px; padding: 10px">
    <div style="border: 2px dashed red;">
      Caixa outset
      </div>
    </div>
  </div>
```

O resultado deste pedaço de HTML está representado na Figura 8.6. Note que o segundo elemento **<div>** tem uma borda com o estilo **outset** e inclui tanto margem como

espaçamento.

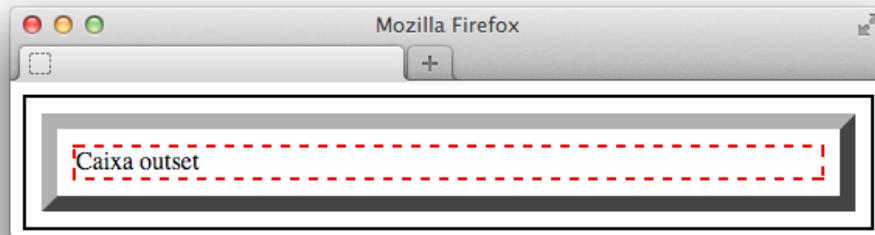


Figura 8.6: Modelo de caixas em CSS com padding.

Se ao pedaço de HTML anterior for retirada a definição de margem e de espaçoamento na segunda marca <div>, pode-se ver na Figura 8.7 que todos os elementos ficam em contacto uns com os outros.

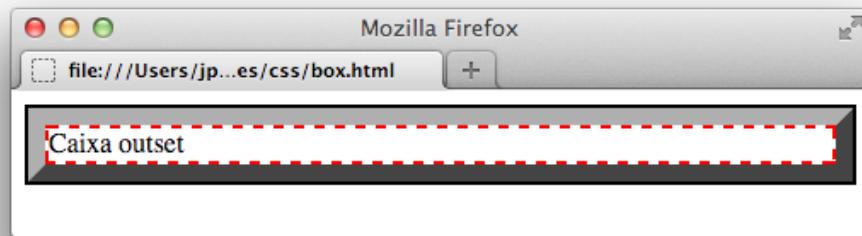


Figura 8.7: Modelo de caixas em CSS sem padding.

Exercício 8.9

Utilize as propriedades de margem, espaçoamento e borda para criar uma tabela visualmente agradável.

(Esta é uma alternativa poderosa ao atributo border do elemento table. De facto, usar CSS é a forma recomendada de formatar tabelas, atualmente.)

8.5 Texto

O texto numa página HTML compreende os cabeçalhos, corpo, hiper-ligações e outras marcas contendo caracteres. Em todos estes é possível definir diversas propriedades que permitem controlar de forma precisa a apresentação. Já foi abordado que a cor do texto pode ser modificada através da propriedade **color**, no entanto vários outros aspectos podem ser modificados.

Um dos aspectos é o tamanho do tipo de letra utilizado, que normalmente se expressa na unidade **pt**, através da propriedade **font-size**. Por omissão as páginas utilizam o valor de 12pt para o tamanho do texto normal. Cabeçalhos (**h1...h6**) irão utilizar múltiplos deste valor. Pode-se definir que se pretende utilizar o tamanho 11pt numa página e aplicar esta definição a todo o texto normal definindo a propriedade **font-size** para a marca **<body>**:

```
body {  
    font-size: 11pt;  
}
```

Exercício 8.10

Crie um documento HTML contendo vários cabeçalhos e texto. Altere a definição do tamanho do tipo de letra e verifique que elementos alteraram o seu tamanho. Utilize vários valores e veja como a página é alterada.

Outro aspecto é o tipo de letra a utilizar. Isto define como os caracteres são convertidos em símbolos para serem apresentados na página. Como em qualquer processador de texto, o tipo de letra utilizado pode ajudar a melhorar o aspeto de um dado documento.

Os tipos de letra estão classificados em *famílias* e existem algumas *famílias genéricas* que incluem várias famílias com características semelhantes. A propriedade **font-family** permite indicar a família pretendida. Como não sabemos que tipos de letra haverá no cliente, podemos indicar uma sequência de famílias a tentar, começando na mais específica e acabando na mais genérica. O browser escolhe um tipo de letra da primeira família ou, se não tiver, tenta um tipo da segunda família e assim sucessivamente.

Alguns tipos de letra são tão comuns que podemos admitir que existirão em todos os computadores. Em particular, espera-se que todos tenham tipos pertencentes às famílias genéricas. As famílias genéricas mais comuns são:

sans-serif: Inclui tipos de letra não serifadas tais como as famílias Arial ou Helvetica.

Têm um aspeto semelhante a este.

serif: Inclui tipos de letra serifados tais como Times New Roman.

Têm um aspeto semelhante a este.

monospace: Inclui tipos de letra com largura de símbolos uniforme.

Têm um aspeto semelhante a este.

O exemplo seguinte indica quatro famílias separadas pelo carácter , e termina com a família genérica **serif**. Se o browser não tiver nenhum tipo mais prioritário, acabará por escolher um desta última. Repare que quando o nome do tipo de letra possui um espaço, é necessário colocar o nome entre aspas.

```
body{  
    font-family: "Palatino Linotype", "Book Antiqua", Palatino, serif;  
}
```

Este processo de seleção pode ser observado na Figura 8.8. O sistema tem o tipo de letra *Garamond* mas não o tipo *XPTOFont*. Por isso utiliza o tipo *Garamond* para o primeiro parágrafo, e um outro tipo de letra serifado no segundo. Embora sejam mínimas, podem-se observar diferenças entre os textos dos dois parágrafos.

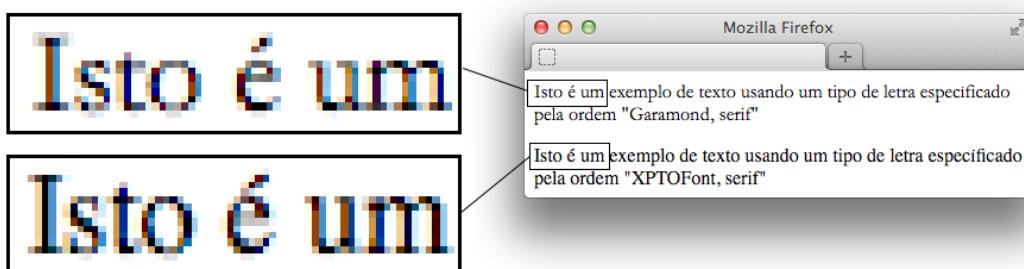


Figura 8.8: Exemplo de escolha de tipos de letra.

Exercício 8.11

Crie uma página onde defina tipos de letra diferentes para diferentes elementos. Por exemplo, é comum utilizarem-se tipos serifados para o corpo dos documentos e tipos não serifados para os cabeçalhos.

As versões mais recentes da especificação CSS permitem, com algumas limitações a nível dos dispositivos, ultrapassar a necessidade do cliente possuir os tipos de letra necessários para a correta visualização de uma página. Isto consegue-se graças à capacidade de incorporar recursos externos, como tipos de letra, na página.

O código seguinte produz o resultado apresentado na Figura 8.9, e demonstra como se podem utilizar tipos de letra externos. Apesar de interessante, esta funcionalidade requer que o navegador suporte a tecnologia e obriga a obter e incorporar na página recursos potencialmente de grandes dimensões.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css"
      href="http://fonts.googleapis.com/css?family=Tangerine"/>
    <style>
      body {
        font-family: 'Tangerine', serif;
        font-size: 48px;
      }
    </style>
  </head>
  <body>
    Usando tipos de letra externos!
  </body>
</html>
```



Figura 8.9: Exemplo de utilização de um tipo de letra externo.

Exercício 8.12

Consulte a página <http://www.google.com/fonts> que contém um vasto repositório de tipos de letra para utilização livre. Escolha várias fontes para utilização numa página local. Pode ver qual o impacto no tamanho final da página e obter o código para incorporação, se clicar no ícone com um quadrado e uma seta para a direita presente em cada tipo de letra.

Crie uma página incorporando 5 tipos de letra diferentes.

Atenção: Incorporar múltiplos tipos de letra na mesma página é uma má prática de design e só está a ser sugerida para efeitos exploratórios!

Além da família, também é possível especificar o *peso* do tipo de letra. Para isso usa-se a propriedade **font-weight** com os valores **normal**, **bold**, **bolder** ou **lighter**. Também é possível especificar um valor inteiro entre 100 e 900. O valor 400 equivale a texto normal e 700 a texto em negrito. A propriedade **font-style** permite ainda selecionar o *estilo*: **italic**, **oblique**, ou **normal**.

Se forem utilizados tipos de letra externos, terá de se instruir o browser para obter símbolos nas versões negrito e/ou itálico.

```
<link rel="stylesheet" type="text/css"
      href="http://fonts.googleapis.com/css?family=Tangerine:bold,italic"/>
```

Exercício 8.13

Adicione algum texto na página que está a construir, definindo pesos e estilos diferentes para alguns parágrafos.

Exercício 8.14

Consulte a página <http://www.w3schools.com/cssref/#text> e verifique como a utilização de atributos **text-decoration**, **text-transform** e **text-shadow** pode afetar o texto.

8.6 Pacote de estilos *Twitter Bootstrap*

Para construir páginas atraentes e funcionais é necessário conhecer com algum detalhe os aspetos apresentados anteriormente, e muitos outros que fazem parte da especificação CSS. No entanto, é frequente a utilização de pacotes de estilos, com vista à criação rápida de uma página HTML. Isso é particularmente relevante para a construção de pequenas páginas web que identifiquem uma equipa, um protótipo ou uma aplicação desenvolvida. A vantagem de utilizar estes pacotes é a facilidade com que se cria uma página com um design coerente e com garantias de funcionar em vários navegadores e dispositivos. A desvantagem é a menor diferenciação entre as diferentes páginas, pois irão partilhar grandes partes de um mesmo estilo.

Um dos pacotes mais utilizados é o *Twitter Bootstrap* [23], que permite a aplicação rápida de estilos pré-criados, sobretudo através da aplicação de classes de CSS. Para utilizar este pacote devem seguir-se algumas regras [24]. No entanto, um exemplo mínimo é bastante simples:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet"
      href="http://netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap.min.css">
    <link rel="stylesheet"
      href="http://netdna.bootstrapcdn.com/bootstrap/3.0.2/css/bootstrap-theme.min.css">
  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1> Exemplo de Twitter Bootstrap </h1>
      </div>
      Conteúdo.
    </div>
  </body>
</html>
```

Neste exemplo, realça-se a inclusão do recurso externo que define o estilo, e da definição de classes para alguns elementos. Neste caso, a marca `<div>` principal é da classe **container**, e existe uma outra marca `<div>` com a classe **jumbotron** que irá criar uma caixa de apresentação. O resultado está presente na Figura 8.10.

Exercício 8.15

Crie uma página semelhante ao exemplo fornecido utilizando *Twitter Bootstrap*.



Figura 8.10: Exemplo de utilização do Twitter Bootstrap.

As próximas secções abordam alguns aspectos do *Twitter Bootstrap*. Recomenda-se a consulta da documentação detalhando todos os estilos presentes [24].

Barra de navegação

É comum as páginas web conterem uma barra de navegação, frequentemente localizada no topo da página. Em *Twitter Bootstrap*, uma barra de navegação é uma marca `` que depois pode ser posicionada onde se pretenda. No exemplo seguinte vamos criar uma barra de navegação escura no topo da página. Repare como uma barra de navegação é precedida por 2 marcas `<div>`, que neste caso são utilizadas para definir o posicionamento da barra. Este caso é um bom exemplo de como uma lista não ordenada pode ser apresentada. A definição da classe `active` para um dos elementos da lista serve para indicar a página atual.

```
<div class="navbar navbar-inverse" role="navigation">
<div class="container">
<ul class="nav navbar-nav">
<li class="active"> <a href="#"> Link 1 </a> </li>
<li> <a href="#"> Link 2 </a> </li>
<li> <a href="#"> Link 3 </a> </li>
</ul>
</div>
</div>
```

Exercício 8.16

Adicione uma barra de navegação à página anteriormente criada. Esta barra deve ser incluída imediatamente após a marca <body>.

Botões

Um aspecto diferenciador dos estilos *Twitter Bootstrap* é a maneira como os botões de uma página são implementados. Normalmente recorre-se à marca <input>, mas usando *Twitter Bootstrap* os botões utilizam a marca (<button>) com um estilo particular. O objetivo é melhorar a apresentação e garantir a mesma apresentação em vários sistemas e navegadores. Os botões podem ter várias classes que definem a sua apresentação. No exemplo seguinte, o botão será do tipo normal (**btn-default**), tamanho grande (**btn-lg**).

```
<button type="button" class="btn btn-lg btn-default"> Default </button>
```

No *Twitter Bootstrap* o aspeto dos botões depende da sua funcionalidade, identificada por uma das classes: **btn-default**, **btn-primary**, **btn-success**, **btn-info**, **btn-warning**, **btn-danger** e **btn-link**. O aspeto de cada um destas classes deverá ser o apresentado na Figura 8.11.



Figura 8.11: Botões em *Twitter Bootstrap*.

Exercício 8.17

Adicione vários botões à sua página de forma a exemplificar cada um dos estilos possíveis.

Da mesma forma, os botões podem ter vários tamanhos, representados pelas classes **btn-lg**, **btn-sm** e **btn-xs**. Se não for indicado o tamanho, o botão será considerado de tamanho normal. Além disso, é possível definir o estado do botão como sendo **active** ou **disabled**.

Exercício 8.18

Acrescente cópias dos botões que criou com cada um dos tamanhos possíveis.

Experimente criar também botões com a classe **disabled** e **active**.

Painéis

O Twitter Bootstrap define que conteúdos podem ser apresentados em painéis de forma a realçar um dado conteúdo. Podem-se considerar estes painéis como elementos informativos de destaque numa página. São compostos por um cabeçalho e um corpo, sendo que o cabeçalho contém um título e o corpo contém o conteúdo do painel. Tudo isto é implementado através de marcas **<div>** e segue os princípios de estilo também presentes nos botões.

O exemplo seguinte cria um painel do tipo **danger**:

```
<div class="panel panel-danger">
  <div class="panel-heading">
    Atenção!
  </div>
  <div class="panel-body">
    Mensagem de aviso.
  </div>
</div>
```

Que resulta no apresentado na Figura 8.12.



Figura 8.12: Painel em Twitter Bootstrap.

Exercício 8.19

Crie vários painéis, utilizando os tipos `panel-danger`, `panel-default`, `panel-primary`, `panel-warning` e `panel-success`.

Exercício 8.20

De forma a colocar os painéis lado a lado numa mesma linha, coloque o conjunto de painéis que criou dentro de uma sequência de marcas.

```
<div class="row">
  <div class="col-sm-4">
    Conteúdo do painel.
  </div>
  ...
</div>
```

8.7 Para aprofundar

Exercício 8.21

Explore as propriedades de CSS presentes no endereço <http://www.w3schools.com/css/default.asp>. Tente criar uma página que exemplifique as diversas propriedades, tanto na forma isolada como combinadas entre si.

Depois pode avaliar o seu conhecimento accedendo a um mini-teste presente em <http://www.w3schools.com/quiztest/quiztest.asp?qtest=CSS>.

Exercício 8.22

Aceda ao endereço <http://getbootstrap.com/examples/theme/> e replique no seu computador a página apresentada.

Exercício 8.23

Aceda ao endereço <http://getbootstrap.com/css/> e experimente cada um dos elementos funcionais de uma página, nomeadamente: estrutura, tabelas, formulários, botões e imagens.

Conceitos elementares de JavaScript

Objetivos:

- Sintaxe JavaScript
- Interacção com o DOM
- Temporizadores
- Eventos

9.1 JavaScript

JavaScript (JS)[25] é uma linguagem interpretada, muito utilizada em páginas Web, mas não só. O facto de ser interpretada significa que não é necessário um passo explícito de compilação para produção de um objeto executável, como acontece com as linguagens Java ou C. A vantagem da interpretação é que permite um desenvolvimento mais rápido por executar diretamente o código escrito pelo programador. A desvantagem é que muitos erros só são detectados quando o fluxo de execução atinge a linha onde o erro está presente. Além disso, código interpretado é bastante mais lento do que código compilado. Na verdade, os browsers atuais fazem compilação Just-in-Time (JIT), que consiste em processar o código JS progressivamente e compilá-lo à medida que é necessário. Assim não é necessário repetir o passo de tradução em execuções posteriores do código, o que melhora o desempenho.

A linguagem JavaScript pode ser incluída em documentos HTML e tem a capacidade de aceder a elementos da página. Isto é, através de JS é possível interagir com a página,

alterando o seu conteúdo e apresentação de forma dinâmica. Neste guião serão explorados aspectos genéricos de JS, como a sua sintaxe, e aspectos específicos de interação com elementos HTML.

9.2 Inclusão numa página

A inclusão de JS numa página Web é análoga à inclusão de estilos CSS. Ou seja, através de elementos específicos na página, é possível incluir o código JS diretamente ou obter o código de uma localização externa.

O exemplo que se segue demonstra o processo de inclusão direta na página, através da marca `<script>`.

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      /* Código JavaScript aqui */
    </script>
  </head>

  <body>
    ...
  </body>
</html>
```

A alternativa é obter o código de um recurso externo, que por motivos de modularidade, reutilização e desempenho é o método recomendado.

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="labi.js"> </script>
  </head>

  <body>
    ...
  </body>
</html>
```

O conteúdo do ficheiro `labi.js` será código JS, seguindo a sintaxe definida na Secção 9.3. Tal como no caso dos estilos, é comum colocar os recursos de JS num diretório diferente

da página. Nomes comuns para esse diretório são **scripts** ou **js**.

Exercício 9.1

Construa uma pequena página, utilizando o exemplo apresentado anteriormente. Verifique que o navegador tenta obter o ficheiro **labi.js**.

A linguagem JS é bastante poderosa e o facto de executar em qualquer navegador permite desenvolver aplicações que podem ser distribuídas de forma muito eficaz. No entanto, note que qualquer código JS é sempre enviado ao cliente na sua forma textual, podendo ser facilmente copiado.

9.3 Sintaxe

A sintaxe da linguagem JS é inspirada na linguagem **C** e algo semelhante à linguagem **Java**. Este guião não irá explorar com detalhe todos os aspetos de sintaxe, ou todas as propriedades da linguagem, mas irá possibilitar uma utilização básica da mesma.

A sintaxe básica da linguagem JS é baseada em instruções, que são organizadas por linhas. Cada linha corresponde a uma instrução, podendo estas instruções ser terminadas com o carácter **;**. A utilização deste carácter é facultativo mas muito recomendado. JS é **case-sensitive**, o que significa que se deve ter cuidado na escrita.

O exemplo seguinte declara uma variável **x**, atribui-lhe um valor e apresenta o resultado na consola do navegador. Utilizando o exemplo anterior, este código estaria dentro do ficheiro **labi.js**, mas também poderia ser declarado numa marca **<script>**.

```
/* Comentário */
var s = "3";
var x;
x = 3;
console.log(x);
```

Como deve identificar, existe uma chamada a uma função **console.log**, com o argumento **x**. Esta sintaxe é em tudo semelhante ao **Java**. A declaração de variáveis por sua vez é diferente. Isto deve-se ao facto de JS ser uma linguagem com tipos dinâmicos, não sendo necessário declarar explicitamente qual o tipo da variável. Portanto, todas as variáveis

são declaradas da mesma forma. É o conteúdo que determina como ela será utilizada.

Exercício 9.2

Seguindo o exemplo anterior, replique os dois exemplos anteriores no seu computador. Aceda à consola do navegador^a e verifique o valor impresso. Experimente com outros valores.

Para voltar a executar o código JS basta atualizar a página do navegador, o que tipicamente se consegue através da tecla **F5**, ou da combinação **CMD + R** no caso do sistema OS X.

^aNo firefox, pode ativar a consola no menu Tools/Web Developer/Web Console.

Exercício 9.3

Experimente substituir a chamada **console.log** por **document.write** e **alert**, de forma a verificar como o JS pode apresentar mensagens aos utilizadores.

Podem ser aplicados operadores aritméticos às variáveis, tais como a adição (+), ou a subtração (-). No entanto o significado desta operação irá variar com o tipo de variável (que depende do seu conteúdo atual). Um bom exemplo é o operador +, que no caso de números irá calcular a soma, mas no caso de sequências de caracteres irá concatená-las.

O exemplo seguinte demonstra a aplicação do operador +:

```
var s = "3";
var x = 3;
console.log(s+1);
console.log(x+1);
```

Em que o resultado deverá ser:

Exercício 9.4

Replique o exemplo anterior no seu computador. Aceda à consola do navegador e verifique o valor impresso. Experimente com outros valores, números reais e sequências de caracteres.

Exercício 9.5

Verifique o que acontece quando troca o tipo de uma variável. Isto é, considerando que **s** é uma *String*, define que passa a ser um inteiro (**s=3;**), realize operações aritméticas e mostre o seu conteúdo.

Quando uma operação aritmética não é válida, a linguagem JS faz uso do termo **NaN** que significa *Not a Number*. Isto pode ser facilmente obtido se se subtrair um inteiro a uma *String*.

Exercício 9.6

Verifique o resultado do seguinte pedaço de código. Poderá ser útil relembrar uma certa banda sonora.

```
for(var i=0;i<16;i++){
    document.write("uma-string" - 2);
    document.write("<br />");
}
document.write("Batman");
```

Funções

De forma a melhor organizar o código, e evitar a replicação desnecessária, é possível organizar um programa em funções. Estes elementos são constituídos por um nome, uma lista de argumentos e um corpo. Tal como a declaração das variáveis é indicada pela palavra reservada **var**, a declaração de funções faz uso da palavra reservada **function**, tal como descrito no exemplo seguinte:

```
function nome_da_funcao(arg1, arg2, arg3){
    /* ...Conteúdo... */
}
```

Comparando com a linguagem *Java*, verifica-se que não é necessário declarar qual o tipo

de retorno da função, nem os tipos dos parâmetros.

Um exemplo simples, de uma função que realiza a soma de dois números, pode ser declarada e invocada da seguinte forma:

```
function soma(x,y){  
    return x+y;  
}  
  
var resultado = soma(3,4);  
console.log(resultado);
```

Exercício 9.7

Construa um programa em JS com quatro funções, uma para cada operação aritmética elementar. Invoque as funções criadas e apresente o resultado.

Condições

A execução condicional segue uma sintaxe semelhante ao Java e é implementada através das palavras reservadas **if**, **else**, no seguinte formato:

```
if ( /*condição*/ ) {  
    /* Instruções no caso verdadeiro */  
} else {  
    /* Instruções no caso falso */  
}
```

As chavetas podem ser omitidas caso apenas exista uma instrução a executar. Na condição poderá utilizar operadores de comparação tais como **<**, **>**, **>=**, **==**, **!=**, etc. Isto é em tudo semelhante ao Java. No entanto, existe uma diferença fundamental, que advém do facto de os tipos das variáveis serem dinâmicos. Considere o seguinte excerto:

```
var a = "3";  
var b = 3;  
  
if (a == b)  
    alert("Iguais");  
else  
    alert("Diferentes");
```

Se executar este excerto, irá verificar que em JS o operador igual (`==`) permite comparar tipos diferentes, convertendo os seus valores. No entanto, por vezes pretende-se efetuar uma comparação do valor e do tipo. Para isso, existe o operador `===` e a sua negação, o operador `!==`. Na linguagem JS diz-se que estes comparadores verificam se o valor é igual e o tipo idêntico. No caso anterior, `a` é igual a `b` mas as variáveis não são idênticas.

Exercício 9.8

Repete o exemplo anterior e compare o resultado utilizando os operadores `==` e `===`.

É possível ainda utilizar a operação `switch`, com uma sintaxe semelhante ao Java:

```
var a ="abc";
switch(a) {
  case "abc": alert("string abc"); break;
  case 3: alert("inteiro 3"); break;
  default: alert("outro");
}
```

Exercício 9.9

Verifique como pode utilizar a operação `switch` misturando tipos diferentes.

Ciclos

Para implementar ciclos, a linguagem JS suporta as mesmas instruções que Java: `while`, `do-while`, `for`.

```
do {
  /*instruções*/
} while (/*condição*/);

while (/*condição*/) {
  /*instruções*/
}

for (/*início*/ ; /*condição*/ ; /*incremento*/) {
  /*instruções*/
}
```

Exercício 9.10

Pratique a utilização de ciclos em JS implementando um caso para cada um dos exemplos apresentados.

9.4 Interação com o DOM

O grande potencial da linguagem JS quando é executada no navegador é a possibilidade de aceder a qualquer elemento HTML, sendo possível manipular em tempo real qualquer aspeto. Isto é, através de JS é possível alterar o conteúdo da página, estilos e marcas após a página ter sido carregada no navegador. A característica que possibilita esta interação é chamada de Document Object Model. Tal como o nome indica, o Document Object Model (DOM)[26] cria um modelo de objetos da página HTML. Estes objetos podem depois ser manipulados na linguagem JS. Os objetos são entidades que possuem propriedades e métodos. É possível consultar ou alterar o valor das propriedades de um objeto (ex, o atributo `href` de uma marca `<a>`) e invocar métodos para produzir ações.

No DOM, os objetos estão organizados num estrutura hierárquica com pais e filhos que reflete a hierarquia de elementos do documento HTML (ver Figura 9.1).

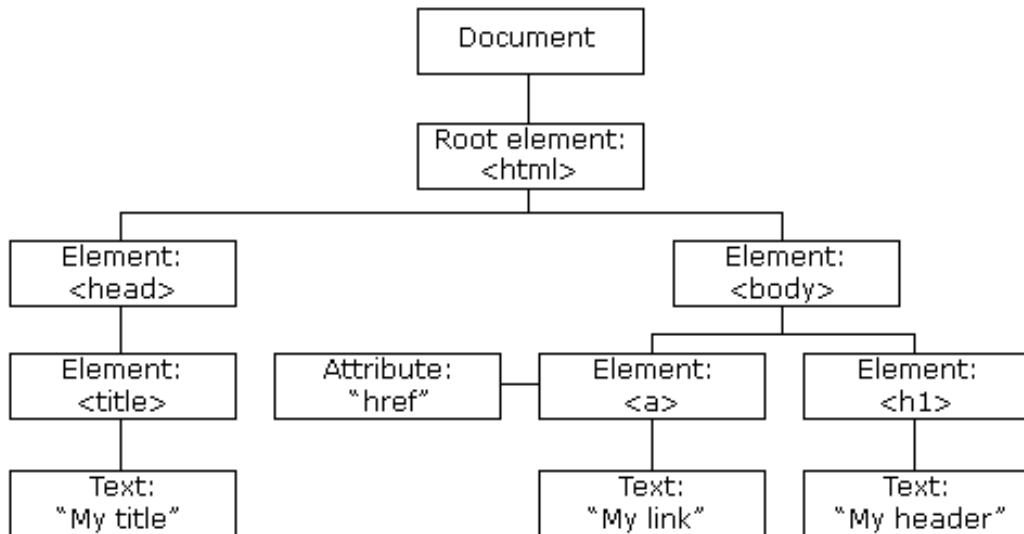


Figura 9.1: Estrutura hierárquica do DOM

Embora existam vários métodos para o fazer, esta secção irá focar-se na utilização do

atributo **id** das marcas HTML. Considere o seguinte pedaço de HTML:

```
<body>
  <input id="op1" value="2"/>
  <input id="op2" value="3"/>
  <input id="res" value="" />

  <script type="text/javascript" src="dom.js"></script>
</body>
```

Repare que a marca **<script>** é incluída depois de todos os outros elementos. Isto é necessário para que os elementos HTML já existam na DOM quando o código JS é executado. O conteúdo do ficheiro **dom.js** será o seguinte:

```
var x = document.getElementById( "op1" );
var y = document.getElementById( "op2" );
console.log( parseFloat(x.value) );
console.log( parseFloat(y.value) );
```

Note a utilização de dois métodos novos:

document.getElementById: Procura por um elemento descendente do objeto (**document**) que tenha o atributo **id** especificado no parâmetro (ex, "op1").

parseFloat: Converte uma *String* (ex, **x.value**) num valor real (*float*).

Note ainda que se acede à propriedade **value** de cada um dos objetos devolvidos. No caso de **x**, o valor será 2, enquanto o que no caso de **y** o valor será 3. Esta propriedade é de escrita e leitura, o que significa que se pode facilmente alterar o texto apresentado num dado campo **<input>** apenas modificando a propriedade **value**.

Exercício 9.11

Implemente o exemplo anterior, completando-o de forma a escrever no elemento **<input id="res" ...>** o resultado da adição dos 2 valores.

Caso se procure um elemento inexistente, o valor devolvido pelo método **getElementById** será **null**, o que pode ser verificado usando uma condição:

```
var x = document.getElementById("nao-existe");
if(x == null)
    alert("Elemento não encontrado");
else
    alert(x.value);
```

Eventos

Até agora o código JS tem sido executado de forma automática quando a página é carregada. Na realidade só o código que se encontre fora de funções é que é automaticamente executado. Este pode depois invocar as diversas funções disponíveis. Ora, por vezes este não é o comportamento desejado, podendo o programador querer que nenhum código seja executado automaticamente, mas apenas após certos eventos. Repare que no caso anterior foi necessário mudar a inclusão do código JS para o final da página. A abordagem mais correta seria a de programar um evento, indicando que o código deve ser chamado após a página ser carregada completamente.

O exemplo seguinte melhora o código anterior, através da utilização do evento `window.onload`. Repare que é necessário colocar o código dentro de uma função. Neste caso a função tem o nome `calcular`.

```
function calcular(){
    var x = document.getElementById( "op1" );
    var y = document.getElementById( "op2" );
    console.log( parseFloat(x.value) );
    console.log( parseFloat(y.value) );
}

window.onload = calcular;
```

Exercício 9.12

Melhore o exercício 11 de forma a que o código da calculadora apenas seja executado após a construção completa da página.

Os eventos também se podem referir a ações do utilizador, nomeadamente mover o apontador, pressionar teclas ou simplesmente a modificação de algum elemento HTML. No caso de uma calculadora será útil incluir um botão que calcule o valor ou outro que indique a operação. Isto realiza-se através da inclusão de propriedades diretamente nas marcas HTML.³⁵

³⁵Para uma lista dos eventos, consulte http://www.w3schools.com/tags/ref_eventattributes.asp.

Considere o seguinte excerto de HTML:

```
<html>
  <head>
    <script type="text/javascript" src="calculadora.js"></script>
  </head>
  <body>
    <input id="op1" value="2" />
    <span id="op-view">+</span>
    <input id="op2" value="3" />
    <input id="res" value="" /><br/>

    <button onclick="calcular()">Calcular</button>
  </body>
</html>
```

Repare como a marca **button** possui um atributo **onclick** que está definido para "**calcular()**". Isto significa que quando o utilizador clicar com o apontador em cima do botão, essa função será executada.

Exercício 9.13

Complete o excerto anterior implementando a função **calcular()**. Verifique o funcionamento da página quando pressiona o botão.

Podemos generalizar este exemplo de forma a que se possa especificar a operação a executar através de campos de seleção:

```
<html>
  <head>
    <script type="text/javascript" src="calculadora.js"></script>
  </head>
  <body>
    <input id="op1" value="2" />
    <span id="op-view">+</span>
    <input id="op2" value="3" />
    <input id="res" value="" /><br/>

    <select onchange="operacao()">
      <option value="+"> Soma </option>
      <option value="-"> Subtração </option>
    </select>

    <button onclick="calcular()">Calcular</button>
```

```
</body>  
</html>
```

Neste caso, a marca `<select>` invocará a função `operacao()`.³⁶ A função simplesmente irá definir uma variável global com a operação a realizar:

```
var op = "+"; //Deverá estar no topo do ficheiro.  
  
function operacao() {  
    var elemento = event.target;  
    var elementoSeleccionado = elemento.options[elemento.selectedIndex];  
    op = elementoSeleccionado.value;  
    console.log("Operação: "+op);  
}
```

A utilização de `event.target` é útil pois permite indicar à função qual o elemento que invocou o evento. Neste caso permite aceder imediatamente ao elemento onde se clicou, para saber qual a operação a realizar, evitando usar `document.getElementById()`. No entanto, este elemento não está disponível em todos os navegadores!

Exercício 9.14

Integre o código anterior numa página funcional. E verifique o funcionamento da mesma. Terá de alterar a função `calcular()` de forma a aplicar uma operação diferente, de acordo com o valor da variável `op`.

Exercício 9.15

Adicione suporte para mais operações, tais como a multiplicação, divisão, ou resto da divisão inteira.

Exercício 9.16

Altere a propriedade `innerHTML` do elemento `id="op-view"` de forma a que a página apresente sempre a operação correta.

³⁶Ver http://www.w3schools.com/jsref/dom_obj_select.asp.

Como pode ver em http://www.w3schools.com/tags/ref_eventattributes.asp, os eventos existentes são inúmeros, podendo inclusive reagir à posição do rato.

Considere que modifica o código anterior de forma a adicionar o evento **onmouseover** e a propriedade **id** à marca **<button>**:

```
...
<button id="btn" onclick="calcular()" onmouseover="mover('btn')">Calcular</button>
...
```

Pode-se agora implementar a função **mover()**, que será ativada sempre que o apontador se encontre em cima do botão. Por exemplo, a função seguinte move um elemento especificado por parâmetro para uma posição aleatória dentro dos limites da janela:

```
function mover(elemento){
    var e = document.getElementById(elemento);

    e.style.position = "absolute";
    e.style.top = (Math.random() * window.innerHeight)+"px";
    e.style.left = (Math.random() * window.innerWidth)+"px";
}
```

Exercício 9.17

Componha o exemplo anterior e verifique o que acontece quando o apontador passa por cima do botão.

Exercício 9.18

Generalize o último exemplo de forma a que todos os elementos se tornem móveis.

9.5 Temporizadores

A linguagem JS tem a possibilidade de atrasar a execução das funções. Isto é útil para implementar animações e controlar a sua duração. Por exemplo, considere o seguinte código que faz um elemento diminuir de altura até desaparecer:

```
function diminuirVertical(elemento) {
    var altura = parseInt(elemento.style.height,10);
```

```
for( ;altura> 0; altura--){
    elemento.style.height = altura+"px";
}
}
```

Neste exemplo não existe maneira de controlar o tempo de execução, ou seja, o tempo que o elemento demora a desaparecer. Na realidade o efeito irá executar rapidamente, não sendo sequer visível qualquer animação.

Exercício 9.19

Implemente o exercício anterior e verifique qual o resultado quando aplicado ao evento **onclick** de uma imagem.

A alternativa que a linguagem JS fornece é a utilização de temporizadores com uma resolução de 1 milissegundo. É assim possível ativar funções de forma periódica, sendo igualmente possível controlar o intervalo entre execuções. No caso de animações é possível controlar a duração da animação. As funções relevantes são:

setInterval("função", intervalo): Define que a função indicada no parâmetro deve ser invocada a cada intervalo de tempo. O intervalo de tempo é expresso em milissegundos. A função devolve um objeto para que seja possível cancelar o temporizador;

clearInterval(variável): Apaga o temporizador passado no argumento;

setTimeout("função", atraso): Define que a função indicada deve executar depois do atraso especificado, em milissegundos. Neste caso a função é executada apenas uma vez.

No exemplo seguinte, altera-se a altura de um elemento (tal como uma imagem), por 10px de cada vez e o processo é executado a cada 10ms. Neste caso a função além de reduzir a imagem, deteta através da variável global **temp** que é a primeira execução e programa o temporizador. No final cancela-o.

```
var temp = null;

function diminuirVertical(elemento){
    if(temp == null){
        var elemento = event.target;
        temp = setInterval("diminuirVertical("+elemento.id+)",10);
    }
}
```

```
}

var altura = parseInt(elemento.style.height) - 10 ;
elemento.style.height = altura+"px";

if(altura == 0){
    window.clearInterval(temporizador);
    temp = null;
}
}
```

Através da função **setTimeout** também é possível executar o mesmo processo, sendo que neste caso o programa fica mais compacto. O princípio de funcionamento é ligeiramente diferente. Neste caso, a cada execução, se a altura for superior a 0, a função programa uma nova execução de si própria para um tempo futuro.

```
function diminuirVertical(elemento){
    if(elemento == null) //Primeira execução
        elemento = event.target;

    var altura = parseInt(elemento.style.height) - 10 ;
    elemento.style.height = altura+"px";

    if(altura > 0){
        setTimeout("diminuirVertical("+elemento.id+)",10);
    }
}
```

Exercício 9.20

Implemente diversas animações utilizando estes métodos. Pode recorrer aos seguintes atributos de estilo:

- **opacity**: Valor real entre 0 e 1 que traduz a opacidade do elemento;
- **width**: Valor inteiro (em px) que representa a largura do elemento;
- **top**: Valor inteiro (em px) que representa a deslocação a partir do topo;
- **left**: Valor inteiro (em px) que representa a deslocação a partir da esquerda;
- **display**: Define a visibilidade do elemento. Ver http://www.w3schools.com/cssref/pr_class_display.asp.

9.6 Para aprofundar

Exercício 9.21

Controlando o atributo `display` do estilo de um elemento `<div>` implemente um *popup* ativado por um clique num botão. Este *popup* deverá ter outro botão que o faz desaparecer.

Exercício 9.22

Implemente um relógio usando JS. Pode obter a data atual utilizando o seguinte conjunto de instruções:

```
var hoje = new Date();
var horas = hoje.getHours();
var minutos = hoje.getMinutes();
var segundos = hoje.getSeconds();
```

Exercício 9.23

Verifique a página <http://getbootstrap.com/javascript> que descreve as funcionalidades de animações, através de JS que o *Twitter Bootstrap* fornece. Experimente criar *popups* (classe `modal`) e outros elementos dinâmicos.

Exercício 9.24

Construa uma pequena página com um botão que utilize *Twitter Bootstrap*. Adicione um evento ao botão de maneira que, quando o apontador clicar nele uma primeira vez, ele adquira a classe `active`. E quando clicar novamente, seja retirada a classe.

10

TEMA

Integração de componentes em páginas Web

Objetivos:

- Componentes de *Twitter Bootstrap*.
- Gráficos usando *Highcharts JS*.
- Mapas e elementos georeferenciados.
- Sistemas de gestão de comentários.

10.1 Páginas Web

As páginas Web atuais necessitam de possuir uma estrutura e estilos muito sofisticados. Cada vez mais a interação faz-se através de interfaces Web e menos através de aplicações que os utilizadores necessitem de instalar no seu computador. Bons exemplos desta tendência são aplicações como o Google Mail [27], Google Docs [28], ou o Microsoft Office 365 [29].

Como vimos na aula dedicada ao tema de páginas Web, existem componentes que permitem acelerar o desenvolvimento, fornecendo, de uma maneira simplificada, estrutura e estilo agradável. Estes sistemas permitem ainda melhorar a interação através da inclusão de componentes "pré-feitos" que aumentam a atratividade e funcionalidade de uma página. Isto reforça a ideia de que o desenvolvimento de um sistema deve favorecer componentes já existentes, disponibilizados por outras entidades, não devendo o programador pretender desenvolver toda a programação e estilo. Só em casos muito específicos se pode tomar esta atitude, à custa de um maior esforço de desenvolvimento e de manutenção do código desenvolvido.

As páginas Web são um dos domínios onde mais se utilizam recursos externos, sendo o resultado visível na quantidade e complexidade das páginas disponíveis na Internet. Este guião irá abordar alguns componentes que podem ser utilizados para enriquecer uma página Web, com foco em componentes dinâmicos para a representação de gráficos, mapas e interação com visitantes.

Para tal, irá fazer-se uso de uma página construída usando o Twitter Bootstrap, sendo que serão adicionados componentes de forma a enriquecer as suas funcionalidades.

Relembra-se que uma página mínima, fazendo uso de Twitter Bootstrap pode ser criada fazendo inclusão da página de estilos correta. De forma a suportar um modelo de interação mais rico, poderá também ser adicionado um conjunto de scripts.

O exemplo que se segue resulta numa página muito minimalista com Twitter Bootstrap mas que poderá servir de base para o restante guião:

```
<!DOCTYPE html>
<html lang="pt">
    <head>
        <meta charset="utf-8">
        <title>LABI</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- Bootstrap minified CSS -->
        <link rel="stylesheet"
            href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
        <!-- Optional theme -->
        <link rel="stylesheet"
            href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap-theme.min.css">
        <!-- jQuery library -->
        <script
            src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
        <!-- Bootstrap JavaScript -->
        <script
            src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"></script>
    </head>

    <body>
        <div class="container">
            (Conteúdo da página.)
        </div>
    </body>
</html>
```

De realçar que ao contrário do exemplo fornecido numa aula anterior, neste caso a página inclui componentes de CSS e de JS. Existe igualmente um elemento **<div>** com a classe

`container`, no qual se deverá incluir o conteúdo da página.

Exercício 10.1

Obtenha uma cópia do seu repositório na plataforma **CodeUA** e, lá dentro, crie um diretório com o nome **aula11**. Entre no diretório recém criado e crie uma página com o conteúdo do exemplo anterior. Verifique o funcionamento correto desta página.

A qualquer instante poderá adicionar a página ao repositório, o que deverá fazer no máximo no final da aula.

10.2 Componentes

Uma página Web é composta por várias secções, definindo a sua estrutura, sendo que em cada secção existirão componentes variados tais como menus, sub-menus, painéis, imagens, entre outros. A linguagem HTML fornece já um grande conjunto de ferramentas que possibilitam a criação de páginas ricas, no entanto por vezes é insuficiente aplicar as marcas de forma isolada, sendo necessário conjugar a marca (HTML), um estilo (CSS) e ações (JS).

Para uma lista completa dos componentes disponíveis para *Twitter Bootstrap*, consultar <http://getbootstrap.com/components>. Neste guião será abordado um número reduzido de componentes.

Menus e Sub-Menus

Os menus e sub-menus são vitais para a navegação numa página. Tipicamente estes são implementados através do recurso a marcas `` e ``. Ou seja, um menu é tratado como uma lista de items. O estilo aplicado a essa lista é que lhe atribui o aspeto típico de um menu.

Usando *Twitter Bootstrap*, o menu principal de uma página chama-se barra de navegação e é incluída dentro de elementos `<nav>` e `<div>`.

```
<nav class="navbar navbar-default">
  <div class="navbar-header">
    <a class="navbar-brand" href="#">LABI</a>
  </div>

  <div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
      <li class="active"><a href="#">TópicoA</a></li>
      <li><a href="#">TópicoB</a></li>
```

```
</ul>
<ul class="nav navbar-nav navbar-right">
    <li class="active"><a href="#">Default</a></li>
</ul>
</div>
</nav>
```

O elemento `<nav>` inicial possui classes (`navbar` e `navbar-default`) que servem para definir o posicionamento e o aspetto (borda, sombra).³⁷ Este elemento inclui marcas que indicam o conteúdo, respetivamente: uma `<div>` com classe `navbar-header` indicando o nome da página; outra `<div>` com classes para definirem estilo; duas marcas `` para a construção do menu em si. Cada elemento (``) da lista será uma opção do menu.

Exercício 10.2

Adicione uma barra de navegação à sua página. A barra deve ser incluída no início do elemento com classe `container`.

Para obter sub-menus é necessário utilizar a classe `dropdown` num item do menu. Os elementos do sub-menu são implementados através de uma nova lista.

```
<li class="dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown">
        TópicoC<b class="caret"></b></a>
    <ul class="dropdown-menu">
        <li><a href="#">Opção1</a></li>
        <li class="divider"></li>
        <li><a href="#">Opção2</a></li>
    </ul>
</li>
```

Exercício 10.3

Adicione o exemplo acima à sua página.

O resultado deverá ser o apresentado na Figura 10.1.

³⁷A marca `<nav>` foi introduzida no HTML5. Em versões anteriores poderá usar-se uma marca `<div class="..." role="navigation">`.

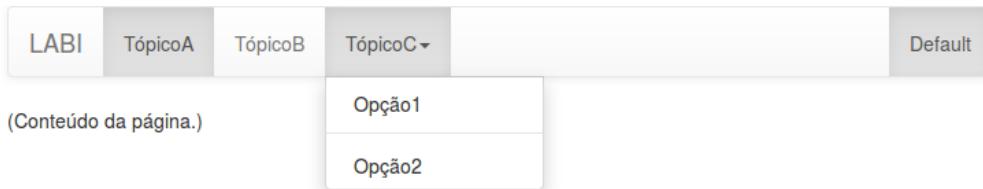


Figura 10.1: Barra de navegação

De forma a navegar na página, é possível criar ligações internas. Estas ligações funcionam através do atributo `id` de cada elemento. Por exemplo, se existir um elemento com `id=topicoA`, uma marca `<a>` pode enviar o utilizador para esse elemento tendo `#topicoA` no seu atributo `href`. O exemplo seguinte demonstra este caso.

```
<a href="#topicoA"> link para o Topico A </a>
...
<h2 id="topicoA">Este é o Tópico A</h2>
```

Exercício 10.4

Na sua página, adicione marcas `<h2>` para cada um dos tópicos deste guião. Crie um número igual de ligações na barra de navegação, possibilitando ir rapidamente para cada tópico a partir desta barra.

Execute os restantes exercícios por baixo do tópico correspondente.

Popups

Os *Popups* são bastante úteis para apresentar mensagens importantes aos utilizadores. Tipicamente sobrepõem-se à página e necessitam de uma ação explícita para desaparecerem, tal como clicar num botão. Usando *Twitter Bootstrap*, os *Popups* são implementados com recurso à classe `modal` aplicado à marca `<div>`.

Tal como demonstrado no exemplo que se segue, existe uma marca `<div>` inicial da classe `modal` que depois possui diversas outras marcas `<div>` com cada uma das áreas do elemento. Pode identificar um cabeçalho (`modal-header`), um corpo (`modal-body`) e um rodapé (`modal-footer`). Note também a existência de duas marcas `button` correspondendo ao botão com o símbolo `X`, que tipicamente existe no topo de uma janela, e ao botão com o texto *Fechar*.

```
<div id="oMeuPopup" class="modal fade">
<div class="modal-dialog">
<div class="modal-content">
    <div class="modal-header">
        <button type="button" class="close"
            data-dismiss="modal" aria-hidden="true">&times;</button>
        <h4 class="modal-title">Título</h4>
    </div>
    <div class="modal-body">
        <p>Conteúdo</p>
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Fechar</button>
    </div>
</div>
</div>
```

Existem vários outros atributos, tais como **data-dismiss**, ou **aria-hidden** que servem para adicionar funcionalidades. O resultado poderá ser o apresentado na Figura 10.2.

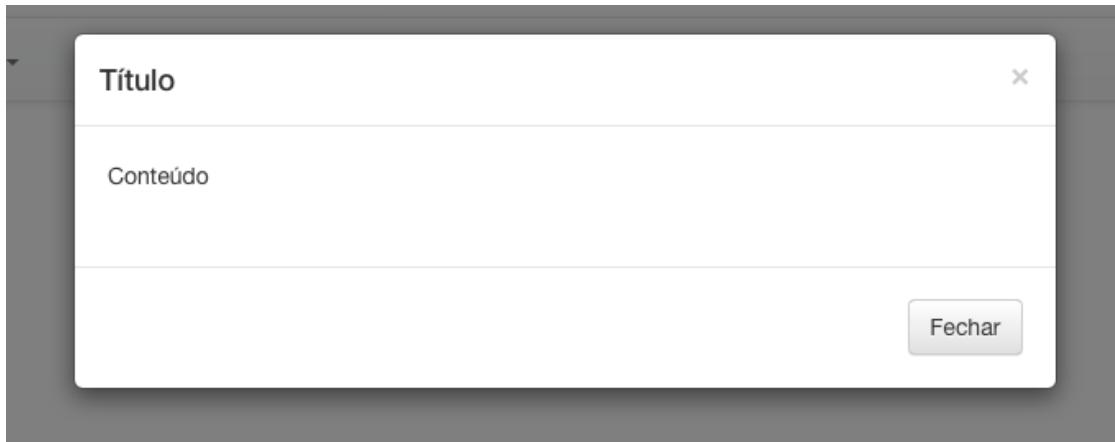


Figura 10.2: *Popup*

Exercício 10.5

Adicione o exemplo acima. Repare que nada é apresentado. No entanto pode ver no código fonte que isto se deve apenas a um atributo do estilo.

Os *Popups* não são sempre apresentados apesar do seu conteúdo permanecer na página. Uma maneira simples de ativar o *Popup* será através de um botão, usando o atributo **data-target**. O valor deste atributo terá de corresponder ao valor do atributo **id** do *Popup* a apresentar.

```
<button class="btn btn-primary" data-toggle="modal" data-target="#oMeuPopup">  
    Lançar popup  
</button>
```

Exercício 10.6

Implemente um exemplo com um botão que ativa um *Popup* e adicione-o à sua página. Verifique que o *Popup* já é apresentado.

10.3 Gráficos

O HTML5 suporta gráficos através dos elementos **<canvas>** e **<svg>**. No entanto, há recursos externos de JS que facilitam muito a criação de gráficos complexos. É o caso da biblioteca *Highcharts JS* que iremos introduzir nesta secção.

Para adicionar um gráfico com esta biblioteca é preciso:

1. Incluir, na marca **head**, os recursos de JS que permitem desenhar gráficos. No caso do *Highcharts JS* é necessária uma linha:

```
<script type="text/javascript"  
src="http://code.highcharts.com/highcharts.js"></script>
```

2. Incluir na página um elemento que irá conter o gráfico.

```
<div id="grafico-linhas" style="width: 400px; height: 300px;"></div>
```

Neste caso, o gráfico terá 400px por 300px de dimensão. O atributo **id** permitirá identificar o elemento nos scripts.

3. Incluir na página um script:

```
<script type="text/javascript" src="grafico-linhas.js"></script>
```

cujo conteúdo (JS) deve configurar o gráfico e os dados:

```

function desenhaGrafico() {
    $("#grafico-linhas").highcharts({
        title: {
            text: "Média de temperaturas",
        },
        xAxis: {
            categories: ["Jan", "Fev", "Mar", "Abr", "Mai", "Jun",
                "Jul", "Ago", "Set", "Out", "Nov", "Dez"]
        },
        series: [{
            name: "Lisboa",
            data: [7.0, 6.9, 9.5, 14.5, 18.2, 21.5, 25.2, 26.5, 23.3, 18.3, 13.9, 9.6]
        }]
    });
}

```

4. Invocar a função de JS que desenha o gráfico.

```

<button class="btn btn-primary" onclick="desenhaGrafico()">
    Grafico de Linhas
</button>

```

Ao compor este exemplo, o resultado deverá ser semelhante ao apresentado na Figura 10.3.

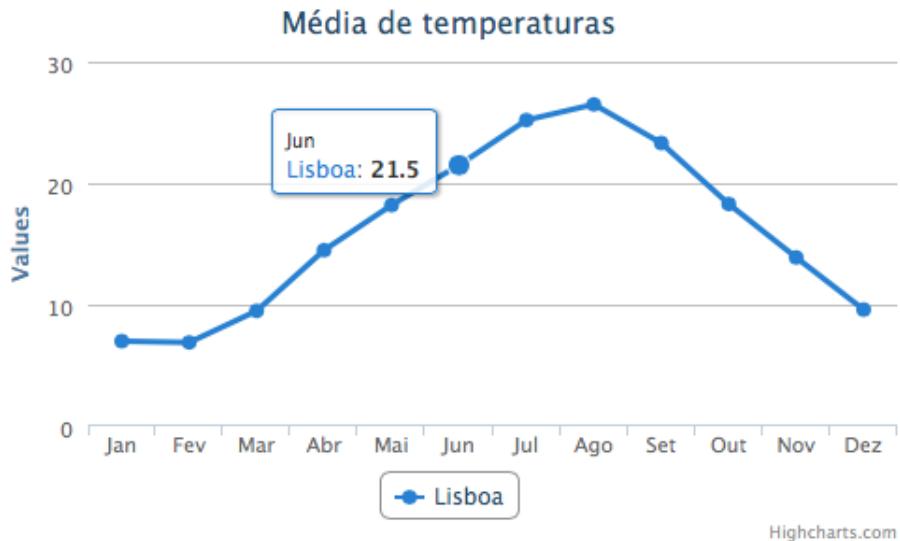


Figura 10.3: Gráfico de linhas

Exercício 10.7

Adicione um gráfico de linhas com as médias de temperatura do sítio onde cresceu.
Pode consultar esta informação no endereço <http://www.weatherbase.com>.

Repare que no exemplo anterior apenas é mostrada a série de temperaturas relativas a Lisboa. No entanto é possível ter várias séries simultaneamente. Para isso basta que o campo **series** possua o seguinte formato:

```
series: [{
    name: "nome-da-serie-1",
    data: [... valores ...]
}, {
    name: "nome-da-serie-2",
    data: [... valores ...]
}]
```

Exercício 10.8

Adicione uma segunda linha com as temperaturas de Aveiro de forma a comparar os dois locais.

A biblioteca *Highcharts JS* possibilita muitos outros tipos de gráficos, tais como: **pie**, **column**, **scatter**, **bar**, **area** [30]. Frequentemente, modificar a aparência do gráfico apenas requer alterar o seu tipo, definido da seguinte forma:

```
$("#grafico-linhas").highcharts({
    chart: {
        type: "column"
    },
    title: {
        text: "Média de temperaturas",
    },
    ...
})
```

Exercício 10.9

Faça cópias do ficheiro com a definição do gráfico e altere-as para testar os vários tipos de gráficos possíveis.

Inclua todos os ficheiros na sua página. Adicione um botão e um elemento para desenhar cada gráfico.

10.4 Mapas

Outro componente muito comum em páginas Web são os mapas ou outros elementos que apresentam informação geo-referenciada. Os mais populares são provavelmente o *Google Maps* [31] e o *OpenStreetMap* [32].

A utilização destes elementos é análoga à dos gráficos, visto que é necessária a inclusão de um recurso externo, a existência de um elemento onde apresentar o mapa e algum código JS para personalizar o conteúdo.

Apesar de ser possível utilizar serviços como o *Google Maps* diretamente, por vezes é mais vantajoso utilizar bibliotecas que facilitem a integração e utilização. Uma das vantagens é a possibilidade de utilizar mapas de diversos fornecedores sem modificações relevantes na programação. Neste guião recomendamos a utilização da biblioteca *Leaflet JS* [33].

Para utilizar *Leaflet JS* é necessário efetuar os seguintes passos:

1. Incluir a definição de estilos e de código do *Leaflet JS*.

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.0.2/dist/leaflet.css">
<script src="https://unpkg.com/leaflet@1.0.2/dist/leaflet.js"></script>
```

2. Criar um elemento para o mapa e incluir um atributo **id** para o identificar no código JS.

```
<div id="oMeuMapa" style="width: 500px; height: 400px"></div>
```

3. Finalmente é necessário incluir código JS que crie o mapa.

```
<script type="text/javascript" src="map.js"></script>
```

Sendo que o conteúdo do ficheiro **map.js** poderá ser o seguinte:

```
var map = new L.Map("oMeuMapa", {center: [40.633258, -8.659097], zoom: 15});
```

```

var osmUrl="http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png";
var osmAttrib="Map data OpenStreetMap contributors";
var osm = new L.TileLayer(osmUrl, {attribution: osmAttrib});

map.addLayer(osm);

```

Neste caso o valor 40.633258 refere-se à latitude, o valor -8.659097 refere-se à longitude e o valor 15 é o nível de *zoom*.

O resultado deverá ser semelhante ao apresentado na Figura 10.4.



Figura 10.4: Mapa utilizando *Leaflet JS*

Exercício 10.10

Adicione um mapa à sua página. Este mapa deverá mostrar a cidade de Aveiro e a cidade onde cresceu.

É possível associar funções a eventos que ocorram no mapa criado. O evento **click**

é muito útil porque fornece as coordenadas de um ponto selecionado no mapa.³⁸ Por exemplo, para mostrar as coordenadas atuais poderíamos incluir um elemento `` na página e acrescentar a instrução

```
map.on("click", mostraCoordenadas);
```

bem como a definição da função

```
function mostraCoordenadas(e){  
    var s = document.getElementById("coordenadas");  
    s.innerHTML = "Latitude, Longitude = "+e.latlng.lat+", "+e.latlng.lng;  
}
```

ao código JavaScript.

Uma das utilidades dos mapas é a apresentação de pontos (ou marcadores), indicando a localização de um ponto de interesse. Esta funcionalidade é muito útil para localizar eventos, locais de interesse ou a morada de empresas e serviços.

Adicionar um ponto a um mapa requer que se defina as coordenadas do ponto e de seguida que se adicione este ponto ao mapa. No exemplo seguinte é criado um array com vários pontos, que são depois adicionados ao mapa.

```
var pontos = [  
    L.marker([40.633258, -8.659097]),  
    L.marker([40.642729, -8.747899])  
];  
  
for(i in pontos) {  
    pontos[i].addTo(map);  
}
```

Também pode ser adicionado um pequeno texto que será apresentado quando o apontador clicar em cima do ponto. Para isto, é utilizado o método `bindPopup(msg)` em que o argumento `msg` pode ser qualquer texto ou mesmo código HTML.

```
...  
L.marker([40.633258, -8.659097]).bindPopup("LABI@DETI")  
...
```

³⁸Para a lista completa de eventos, consultar <http://leafletjs.com/reference-1.0.0.html#map-event>.

Por vezes é útil ajustar a vista de forma a contemplar todos os pontos adicionados. Aqui é necessário criar um grupo com todos os pontos e depois invocar um método que ajusta a vista de forma automática.

```
...
var grupo = new L.featureGroup(pontos);
map.fitBounds(grupo.getBounds());
```

Exercício 10.11

Adicione marcadores que identifiquem vários locais de interesse para si (morada em Aveiro, morada permanente, onde estuda, etc). Não se esqueça de adicionar textos que descrevam os locais e de ajustar o mapa aos pontos.

Frequentemente é desejável apresentar imagens diferentes dependendo da natureza do local assinalado. Isto é simples pois os elementos **marker** podem ser criados com uma lista de opções, onde se inclui o ícone.³⁹

```
var iconeUA = L.icon({ iconUrl: "http://xcoa.av.it.pt/ua.png" });
...
L.marker([40.633258, -8.659097], {icon: iconeUA}).bindPopup("LABI@DETI")
...
```

Exercício 10.12

Adicione imagens personalizadas aos seus ícones. Na página <http://mapicons.nicolasmollet.com> encontra muitos e variados ícones. Terá de escolher os ícones e colocá-los junto com a sua página.

Além de marcadores também é possível adicionar polígonos que sinalizam uma área alargada e não um ponto individual. A metodologia é semelhante à criação de marcadores, mas os polígonos são criados através de um array de pontos. A Reitoria de Universidade de Aveiro pode ser indicada através de um polígono, da seguinte forma:

```
var reitoria = L.polygon(
  [ [40.63102, -8.65793], [40.63149, -8.65731],
```

³⁹Para uma lista completa de opções, consultar <http://leafletjs.com/reference-1.0.0.html#marker>.

```
[40.63126, -8.65699], [40.63078, -8.65759] ],  
{ color: "red" } );  
reitoria.addTo(map);
```

Exercício 10.13

Adicione um polígono que delimite o DETI.

Ao polígono que criou, use o método `bindPopup(msg)` para mostrar informação do departamento quando se clicar no polígono.

Exercício 10.14

Em vez de utilizar o tipo `polygon`, se utilizar o `polyline` o resultado será uma linha e não um polígono. Utilize estes dois tipos de objetos para indicar onde mora e qual o trajeto para a Universidade de Aveiro.

10.5 Comentários

Um aspeto interessante de se adicionar a um blog ou outra página com visitantes é a capacidade de se incluir comentários. Existem várias maneiras de obter esta funcionalidade, sendo que a mais simples é a adição de um componente externo em JS. Um exemplo é o serviço Disqus [34], que apenas necessita da inclusão de código HTML e JS na página em questão.

Exercício 10.15

De forma a adicionar comentários à sua página, aceda a <http://disqus.com> e identifique-se com o utilizador `alunolabi` e a palavra passe `alunolabi`.

Seguidamente, aceda a <http://disqus.com/admin/universalcode/> de forma a obter o código a incluir na página. O valor da variável `disqus_shortname` deverá ser `labi`.

Nota: Por favor não altere os dados da conta! Este serviço é partilhado por todos os

alunos.

Exercício 10.16

Envie a página que acabou de criar para o servidor **xcoa.av.it.pt**, no diretório **~/public_html**. Pode utilizar o comando **scp** ou o repositório **GIT** onde tem esta página.

Pode aceder à página através do endereço <http://xcoa.av.it.pt/~labi-tXgY>.

Verifique que é possível adicionar comentários.

Exercício 10.17

Aceda novamente ao endereço <http://disqus.com> e verifique as funcionalidades que lhe são apresentadas. Verifique que, por exemplo, ao adicionar comentários também pode ter informação sobre os seus visitantes e moderar os comentários.

10.6 Para aprofundar

Exercício 10.18

Aceda à página do *Twitter Bootstrap* e implemente uma página local com todos os componentes extra que são apresentados.

Exercício 10.19

Aceda à página do *Fuel UX* (<http://exacttarget.github.io/fuelux>) e implemente outros componentes que permitem enriquecer páginas Web.

Programação em Python

Objetivos:

- A linguagem Python
- Variáveis
- Condições, Ciclos e Funções
- Listas e Dicionários

11.1 Linguagem Python

A linguagem *Python* [35] é uma das linguagens de uso geral mais populares e tem uma extensa documentação [36]. É considerada uma linguagem de alto nível por abstrair os conceitos fundamentais do computador, focando-se no desenvolvimento rápido de algoritmos, na fácil compreensão do código produzido e na sua estrutura. O programador foca-se na expressão do algoritmo sem necessidade de compreender aspectos de baixo nível.

Os programas escritos em *Python* são tipicamente bastante compactos (considerando o número de linhas), especialmente em comparação com outras linguagens como *Java* ou *C*. Estas características tornam a linguagem *Python* ideal para a prototipagem rápida de sistemas, para o desenvolvimento de sistemas complexos e, através dos módulos que possui, como ferramenta para cálculo científico.

Python é uma linguagem que suporta múltiplos paradigmas (ou estilos) de programação. Enquanto *Java* requer obrigatoriamente a utilização de classes, uma característica da programação orientada a objetos pura, a linguagem *Python* suporta esse paradigma,

mas não o impõe, permitindo outros como a programação funcional ou a programação imperativa procedural.

Visto ser uma linguagem interpretada, não são necessários os passos de compilação e ligação a bibliotecas, como noutras linguagens. Isto possibilita que o mesmo código seja facilmente executado em múltiplos sistemas operativos e que partes de um sistema sejam alterados dinamicamente durante a execução de uma aplicação.⁴⁰

Em Laboratórios de Informática, *Python* será a linguagem principal para a exploração de vários conceitos do domínio da informática. As secções seguintes são o primeiro passo nessa direção, introduzindo *Python* para a resolução de problemas de programação comuns.

11.2 Características Básicas

Python é uma linguagem com um conjunto de características particulares, que a distingue de outras linguagens vulgarmente utilizadas:

Uso geral: Pode ser utilizada para o desenvolvimento de qualquer tipo de aplicações ou serviços, não sendo uma linguagem para um nicho específico.

Interpretada: Os programas são processados à medida que é necessário executar cada pedaço de código. Não é necessário compilar o programa antes de ser utilizado.

Alto nível: Não são expostos detalhes da plataforma de computação, como registos, ponteiros, ou endereços de memória. O programador foca-se na implementação de um algoritmo e não nos detalhes do *hardware*.

Tipos dinâmicos: As variáveis não têm um tipo fixo e por isso não precisam de ser pré-declaradas. O tipo da variável está sempre associado ao valor que tem armazenado e pode modificar-se dinamicamente, bastando para isso atribuir-lhe um valor de tipo diferente.

Tipagem forte: Em todas as operações, os tipos dos operandos são verificados. Não há conversões implícitas entre tipos de dados que não sejam naturalmente compatíveis. Isto contrasta com a tipagem fraca do Javascript, por exemplo.

Gestão automática de memória: A alocação e libertação de memória dinâmica é gerida de forma automática, não sendo este detalhe exposto ao programador.

Os programas *Python* são ficheiros de texto, tipicamente com a extensão `.py`, que são executados através de um interpretador de *Python*. O exemplo seguinte demonstra o conteúdo de um programa *Python* (`hello.py`) que imprime um texto curto no ecrã:

⁴⁰A alteração de um ficheiro não implica qualquer recompilação.

```
print("Gosto de LabI")
```

Uma forma alternativa de o fazer seria através do módulo `sys`:

```
import sys
sys.stdout.write("Gosto de LabI\n")
```

Note que neste caso é necessário especificar de forma explícita o final de linha através do carácter "`\n`". No entanto permite maior controlo sobre a escrita para a consola.

Em qualquer dos casos, o programa executa-se através do comando:

```
python3 hello.py
```

O interpretador de *Python* executa programas em ficheiros, mas também pode ser usado interativamente. Quando se invoca o interpretador sem argumentos, ele processa as instruções que vierem do dispositivo de entrada (o terminal). Neste modo de funcionamento aparece um *prompt* com o formato `>>>` e o utilizador pode introduzir instruções ou expressões de *Python*, que são executadas linha a linha e produzem resultados imediatos. Desta forma, o utilizador interage com uma interface de linha de comandos que interpreta *Python*, o que é muito útil para explorar a linguagem e testar ideias ou resolver pequenos problemas. Para terminar o interpretador pode ser utilizada a sequência **CTRL+D** (em Windows usa-se **CTRL-Z**), que sinaliza o fim do “ficheiro” de entrada de dados. O exemplo abaixo demonstra uma breve interação com o interpretador de *Python*.

```
user@host:~$ python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Gosto de LABI")
Gosto de LABI
>>> ^D
```

Em modo interativo, o interpretador tem um comportamento ligeiramente diferente do modo *offline*. Em particular, quando se avalia uma expressão em modo interativo, o seu resultado é apresentado imediatamente. Assim, pode usar-se o interpretador como uma calculadora avançada.

```
>>> 3+5
8
>>> "cara" + "pau"
'carapau'
>>> print("cara" + "pau")
carapau
```

Note como a representação interna de um valor pode diferir do seu valor impresso.

Exercício 11.1

Faça um programa com as 3 linhas do exemplo anterior e execute-o. Repare que os resultados das duas primeiras linhas não são impressos, apenas a terceira linha produz um resultado visível. Esse é o comportamento esperado em modo não interativo.

Um ponto importante é que existem duas versões em utilização da linguagem: 2.x e 3.x. Estas duas versões são maioritariamente compatíveis entre si, apenas com algumas diferenças relevantes para estes laboratórios. Os exercícios aqui propostos são apresentados utilizando a versão 3.x, mas podem ser facilmente convertidos para a versão 2.x.

As diferenças relevantes são:

```
#Impressão de valores
#Python 2
print "Hello World"

# Python 3
print("Hello World")

#Entrada de valores
#Python 2
x = raw_input("Insira X")

#Python 3
x = input("Insira X")
```

Estrutura de um programa

Um programa em *Python* é composto por uma zona inicial onde são declaradas as importações de módulos, a que se seguem as definições de variáveis, de funções e/ou de classes, com a implementação do algoritmo necessário. Um aspeto importante é que não existem delimitadores de blocos explícitos como na linguagem *Java*. Em vez disso,

cada instrução ocupa uma linha e é a indentação dessa linha que determina a que bloco pertence a instrução. O código a seguir demonstra esta característica da linguagem.

```
1 # encoding=codificação
2
3 # Zona de importação de módulos
4 import modulo1
5 import modulo2
6
7 # Funções
8 def funcao(a):
9     print(a)
10    if a:
11        print("SIM")
12    return False
13
14 # Outras instruções
15 funcao(True)
16 print("d")
```

A linha 8 do código define uma função (isto será abordado na Secção 11.2), e a linha 9 pertence a essa função apenas porque está mais indentada. A linha 11 possui uma indentação ainda maior, indicando que pertence a um sub-bloco, neste caso de uma instrução condicional. Por sua vez a instrução na linha 12 já não pertence à instrução condicional.

Note ainda que as linhas 8 a 12 definem uma função, mas esta só é executada quando invocada, na linha 15. Também não existe qualquer indicador de fim de instrução, sendo que esta corresponde a toda uma linha.

Sequências iniciadas pelo carácter `#` são comentários, que se estendem até ao fim de linha. Em *Python* não existem comentários de múltiplas linhas, pelo que cada linha de comentário tem de começar com o carácter `#`.

Em *Python 3* também é facilitado o processamento de texto com caracteres acentuados, suportanto *unicode* de forma nativa. Em *Python 2* é importante definir explicitamente a codificação dos ficheiros (ver primeiro comentário do exemplo anterior).

Valores, variáveis e tipos

As variáveis são declaradas no momento em que lhes é atribuído um dado valor. Em *Python* todas as variáveis são referências, não existindo a distinção entre variáveis primitivas e não primitivas que existe na linguagem *Java*. O exemplo seguinte define duas variáveis `a` e `b` e imprime a soma de ambas.

```
a = 3
b = 5.0
print(a + b)
```

De notar que o valor de **a** é do tipo inteiro (**int**) e o de **b** é real (**float**), que são dois (sub)tipos numéricos compatíveis em operações aritméticas. A adição de inteiros produz um inteiro, mas uma adição com um **float** produz um **float**, como seria de esperar. Pelo contrário, o operador de divisão em Python 2, tal como em Java ou C, tem um comportamento inesperado para um leigo, como se pode verificar no exemplo seguinte,

```
a = 3
b = 5.0
print a / b
b = 5
print a / b
```

que produz o resultado abaixo.

```
0.6
0
```

A primeira divisão, entre um inteiro e um real produziu o quociente real, mas na segunda, entre dois inteiros, o resultado é o quociente da divisão inteira. Este comportamento ambíguo do operador **/** é propício a erros de programação e por isso recomenda-se que se use sempre o operador **//** quando se pretende a divisão inteira. Na versão 3 do Python a ambiguidade foi mesmo eliminada: o operador **/** dá sempre o quociente real, mesmo quando usado entre inteiros. O exemplo anterior também demonstra a variação dinâmica do tipo da variável **b**.

Exercício 11.2

Crie um ficheiro *Python* e verifique o resultado de adicionar, subtrair, multiplicar, dividir (**/** e **//**) e achar o resto da divisão inteira (operador **%**) de diferentes valores. Use valores positivos, negativos e nulos, inteiros e reais.

Se puder, execute o programa com o Python 2 e com o Python 3.

Em *Python*, os operadores de quociente (**//**) e resto (**%**) de divisão inteira têm resultados diferentes dos equivalentes em Java ou C quando o dividendo é negativo. Nenhuma

das linguagens está errada. Tratam-se apenas de interpretações diferentes, mas ambas matematicamente coerentes, da operação de divisão inteira. Na opinião dos autores, a interpretação adotada no Python tem vantagens na maioria das aplicações práticas.

Strings

Um dos tipos mais utilizados para além dos tipos numéricos é talvez o tipo *String*. Em *Python* ele define uma sequência com zero ou mais caracteres. Não existe um tipo específico para caracteres isolados como o **char** do *Java*. Um caráter é tratado como um caso especial de uma *String* com comprimento um. Isto vai de encontro a um dos princípios da linguagem *Python* que diz *Special cases aren't special enough to break the rules*.

Valores do tipo string indicam-se entre aspas ("string") ou entre plicas ('string').

```
# encoding=utf-8

a = "Laboratórios"
b = " de "
c = "Informática"
print(len(c)) # qual o resultado?
print(a+b+c) # concatenação de strings
```

O exemplo seguinte demonstra como se pode aceder a partes de uma string. Note que a sintaxe é semelhante ao modo como são acedidos os arrays em *Java*, mas permitindo também selecionar subsequências.

```
# encoding=utf-8

a = "Laboratórios"
b = " de "
c = "Informática"

print(a[0:3]+"-"+c[0]+" "+str(2018)) # Imprime Lab-I 2018
```

Exercício 11.3

Reita os exemplos anteriores com outras variáveis do tipo *String*. Experimente operações matemáticas como a adição e multiplicação com inteiros (sem utilizar **str**).

Existem várias funções auxiliares que permitem manipular variáveis do tipo *String*. Nominalmente, há funções para a determinar o comprimento da string, para converter para maiúsculas ou minúsculas, para retirar espaços, etc. Para uma lista completa, consultar <http://docs.python.org/3/library/stdtypes.html#string-methods>.

```
# encoding=utf-8

a = "    Laboratórios de Informática 2018 "

print(len(a))          # Comprimento de uma string
print(a.lower())        # Converte para minúsculas
print(a.upper())        # Converte para maiúsculas
print(a.title())        # Converte para título
print(a.find("t"))      # Primeira posição de "t"
print(a.isalpha())       # Verifica se só tem letras
print(a.isdigit())       # Verifica se é um número
print(a.islower())       # Verifica se tem só minúsculas
print(a.strip())         # Remove espaços nos extremos
print(a.split(" "))     # Divide por espaços
```

Exercício 11.4

Implemente um pequeno programa que experimente as funções apresentadas e outras que encontre na documentação da linguagem Python.

Não existe uma função **printf**, mas é possível criar uma string com uma dada formatação e imprimi-la. Para isso pode usar-se o operador de formatação %, que insere valores em locais assinalados numa string de especificação de formato. A sintaxe da especificação de formato é semelhante à usada em C e Java. O exemplo seguinte produz a linha “Benfica 4, Porto 0” a partir de uma formatação de *Strings*.

```
equipa1 = "Benfica"
equipa2 = "Porto"
golos1 = 4
golos2 = 0
print("%s %d, %s %d" % (equipa1, golos1, equipa2, golos2))
```

Exercício 11.5

Implemente um exemplo semelhante ao anterior, mas referente a cursos da Universidade.

Por vezes é necessário converter *Strings* em valores numéricos e vice versa. A conversão de valores numéricos para *String* é conseguida através da função **str()**, enquanto que a conversão inversa é conseguida através das funções **float()** ou **int()**. O exemplo seguinte converte valores de e para *String*, imprimindo-os de seguida.

```
a = 3
sa = str(3)
b = int(sa)
c = float(sa) * 1.2
print("%d, %s, %d, %.2f" % (a, sa, b, c))
```

Listas

Em *Python*, as listas são as estruturas de dados nativas com maiores semelhanças aos *arrays* usados noutras linguagens, mas são bastante mais versáteis. Estas estruturas são compostas por uma sequência de valores, que podem ser acedidos através de um índice. O primeiro valor corresponde ao índice 0 e o final ao índice **len(array)-1**. O exemplo seguinte demonstra a definição de uma lista com os cursos do DETI e a sua impressão de várias formas. A primeira forma imprime toda a lista, a segunda forma imprime apenas o primeiro elemento, enquanto a terceira forma imprime todos os valores entre o primeiro e o terceiro (exclusive).

```
l = ['MIECT', 'LEI', 'MEI', 'MSI', 'MIEET']
print(l)
print(l[0])
print(l[0:2])
```

Uma vantagem das listas é o facto de a sua dimensão ser dinâmica, sendo possível adicionar mais elementos a uma lista através dos métodos **append** e **extend**. O método **append** acrescenta um novo elemento ao fim da lista, enquanto o método **extend** permite estender uma lista com outra. A lista criada no exemplo anterior poderia ser refeita através destes métodos da seguinte forma:⁴¹

```
l = []
l1 = []
l2 = []

l1.append('MIECT')
l1.append('LEI')
```

⁴¹Embora estes métodos sejam úteis, não existe qualquer vantagem nesta implementação, sendo preferido o método anterior.

```
11.append('MEI')
12.append('MSI')
12.append('MIEET')

l.extend(11)
l.extend(12)

print(len(l))
```

Exercício 11.6

Crie um programa que declare uma lista e imprima o seu conteúdo. Imprima partes, toda a lista e estenda o seu conteúdo.

Exercício 11.7

Verifique qual o resultado de aplicar a função `sorted` a uma dada lista.

Uma lista importante é a que é utilizada para fornecer ao programa os argumentos na linha de comandos. Pode ser acedida através da variável `sys.argv` definida no módulo `sys`. Esta lista contém todos os argumentos passados ao programa, incluindo o próprio nome do programa. O programa seguinte imprima os valores dos argumentos que lhe são passados. Experimente executá-lo com e sem argumentos.

```
import sys
print(sys.argv)
```

Exercício 11.8

Utilizando a lista `sys.argv`, implemente um programa que calcule a soma do primeiro e segundo argumento.

Dicionários

Os dicionários são semelhantes às listas, no sentido em que estabelecem uma associação entre uma chave (índice no caso da lista) e um valor. No entanto, os dicionários permitem que se possa definir tanto a chave como o valor. Os dicionários são vantajosos em determinadas aplicações pois permitem ter uma estrutura em que o acesso é efetuado através de uma chave com significado para o programador. A sintaxe básica para criar

um dicionário é a seguinte:

```
nome = {'chave1': valor1, 'chave2': valor2, ... }
```

Para aceder a um elemento usa-se a forma `nome['chave']`.

```
nome = {'chave1': 0} # Cria um dicionário com uma chave  
nome['chave1'] = 1 # Redefinição do valor  
nome['chave2'] = 2 # Definição de um novo par <chave, valor>  
  
print(nome['chave1'])  
print(nome['chave2'])
```

O exemplo seguinte mostra a criação de uma lista de alunos, a impressão do nome do primeiro elemento, seguida da impressão de toda a lista. Cada aluno é um dicionário com nome e número mecanográfico.

```
l = []  
  
l.append( {'nome': "Catarina", 'mec': 4534} )  
l.append( {'nome': "Pedro", 'mec': 1234} )  
l.append( {'nome': "Joana", 'mec': 5354} )  
l.append( {'nome': "Miguel", 'mec': 6543} )  
  
print(l[0]['nome'])  
print(l)
```

Exercício 11.9

Crie um dicionário que permita armazenar as pontuações de um jogo de futebol entre duas equipas.

Entrada de dados da consola

Em Python, a leitura de dados do teclado pode ser efetuada através da função `raw_input("mensagem")`. Esta função imprime a mensagem passada como parâmetro (um prompt) e espera pela introdução de uma linha de texto. Essa linha é devolvida como string e poderá depois ser convertida para um valor inteiro, real ou de outro tipo através das funções `int()`, `float()` ou outra.

No exemplo seguinte é implementada uma calculadora simples que multiplica 2 valores inseridos pelo teclado.

```
# encoding=utf-8

valor1 = float(input("Primeiro Valor: "))
valor2 = float(input("Segundo Valor: "))

print("Resultado: %f * %f = %f" % ( valor1, valor2, valor1*valor2))
```

Exercício 11.10

Implemente um programa que repita o texto inserido mas convertendo todos os caracteres para maiúsculas.

InSTRUÇÕES CONDICIONAIS

As instruções condicionais permitem executar blocos de código alternativos dependendo do valor lógico de uma ou mais condições. Em Python estas instruções são indicadas pelas palavras chave **if**, **else** e **elif**, e têm um modo de funcionamento semelhante ao de outras linguagens de programação, como pode constatar no exemplo abaixo. A diferença mais significativa é meramente sintática: decorre do uso de indentação para definir se um conjunto de instruções pertence a um bloco ou não.

```
if cond:
    instruções a executar em caso positivo
else:
    instruções a executar em caso negativo
```

A condição **cond** é geralmente uma expressão booleana que pode resultar de uma comparação, de um valor de uma variável ou de uma combinação de múltiplas expressões com operadores lógicos como **and**, **or**, ou **not**. O exemplo seguinte imprime “Sim” se um valor for divisível por 2 ou por 3:

```
a = ... #Valor
if a % 3 == 0 or a % 2 == 0:
    print("Sim")
else:
    print("Não")
```

Exercício 11.11

Implemente um pequeno programa que calcule se um dado ano é bissexto ou não.

Um aspeto algo peculiar das condições em *Python* é que não têm de ser estritamente do tipo `bool` (o tipo de dados booleanos). Por exemplo, um valor numérico usado como condição é considerado verdadeiro sse for diferente de zero; uma string ou outro tipo de sequência é considerado verdadeiro sse tiver um tamanho superior a 0; o valor `None` é sempre considerado falso. O exemplo seguinte demonstra como seria possível imprimir a palavra “Par” caso um dado valor seja divisível por dois.

```
a = 3 # Ou outro valor
if not (a % 2):
    print("Par")
else:
    print("Impar")
```

Exercício 11.12

Use esta funcionalidade para determinar se uma *String* é vazia, sem utilizar a função `len()`.

O mérito desta característica da linguagem é, no mínimo, questionável. A vantagem de poupar dois ou três caracteres a expressar algumas condições não parece compensar o que se perde em termos de legibilidade e comprehensibilidade e simplicidade.

Instruções de repetição

As instruções de repetição (ou ciclos) permitem executar blocos de instruções repetidamente. Existem duas palavras chave reservadas na linguagem para implementar ciclos: `for` e `while`. noutras linguagens é comum estas duas instruções serem usadas sem grande diferenciação. Isto **não** é verdade na linguagem *Python*. Isto resulta da aplicação da regra “*There should be one—and preferably only one—obvious way to do it*”, impedindo que existam múltiplas instruções com a mesma funcionalidade.

- **for** - Percorre os elementos de uma sequência, repetindo um conjunto de instruções por cada um deles. Por exemplo, pode iterar sobre os caracteres de uma *String*, ou sobre todos os valores entre 1 e 10.

- **while** - Executa repetidamente um conjunto de instruções enquanto uma condição for verdadeira.

O exemplo seguinte demonstra um ciclo **for**. Note que não há qualquer operação aritmética explícita para incrementar valores (ex, **i+=1**). Usou-se a função **range**, que gera uma sequência de valores em progressão aritmética $[0, 1, \dots, 9]$, e o ciclo repete a instrução **print(i)** após atribuir cada um desses valores à variável **i**.

```
for i in range(0, 10):
    print(i)
```

A execução da função `range` pode ser analisada em detalhe se se executar o exemplo seguinte. O resultado deverá ser uma lista de valores entre 0 e 9.

```
print(range(0, 10))
```

Exercício 11.13

Utilizando um ciclo `for`, implemente um programa que imprima uma sequência de Fibonacci. Uma sequência de Fibonacci é composta por números em que cada número é composto pela soma dos 2 anteriores. Uma sequência típica de 10 elementos será 1,1,2,3,5,8,13,21,34,55.

Aplicando um ciclo `for` a uma sequência do tipo *String* irá iterar sobre os seus caracteres. O exemplo seguinte imprime cada um dos caracteres de um texto.

```
a = "Laboratórios de Informática"
for i in a:
    print(i)
```

Exercício 11.14

Implemente um ciclo `for` que determine quantos dígitos existem numa frase. Pode recorrer à função `isdigit()` de uma *String*.

Exercício 11.15

Implemente um ciclo `for` que permita contar quantas palavras existem numa frase. Recorra à função `split()` para criar uma lista com a frase dividida pelos espaços.

Exercício 11.16

Implemente um ciclo `for` que permita criar uma *String* que seja o inverso de outra ("abcde" -> "edcba").

O ciclo **while** difere do ciclo **for** pois não percorre um conjunto de valores. Em vez disso, repete instruções enquanto uma condição for verdadeira. O exemplo abaixo usa um ciclo **while** para apresentar os valores entre 0 e 9. Neste caso é necessário inicializar e incrementar explicitamente a variável **i** de forma a esta variar o seu valor.

```
i = 0
while i < 10:
    print(i)
    i = i + 1
```

Exercício 11.17

Inverta uma string, mas agora utilizando um ciclo **while**.

Exercício 11.18

A utilização do ciclo **while** é mais adequada a cálculos com o índice, ao invés de iterar por um conjunto. Desta forma, é mais adequada a utilização deste ciclo em cálculos como o factorial de um número.

Use o ciclo **while** para calcular o factorial de um valor.

Funções

A utilização de funções permite reutilizar instruções, sem que exista duplicação de blocos de código, assim como isolar instruções que desempenhem operações específicas. Esta característica é parte integrante do conceito de modularidade, essencial para o desenvolvimento de aplicações com mais de umas dezenas de linhas.

Na linguagem *Python* as funções podem possuir parâmetros, tal como podem devolver valores por retorno. No entanto, não existe lugar à definição de tipos de valores. A função presente no exemplo seguinte permite calcular o número de valores pares entre **a** e **b - 1**. Para definição da função utiliza-se a palavra chave **def**, seguida do nome da função e dos seus parâmetros.

```
def pares(a, b):
    c = 0
    i = a
    while i < b:
        if i % 2 == 0:
            c = c + 1
        i = i + 1
    return c
```

Neste caso, a função é declarada mas não é realmente utilizada no programa. Para isso é necessário que seja invocada e só são automaticamente executadas instruções que não possuam indentação. De forma a executar esta função para os valores 1 e 10, poderia ser adicionada a linha `print(pares(1, 10))` ao final do ficheiro. O ficheiro resultante seria o apresentado de seguida.

```
def pares(a, b):
    ...
print(pares(1, 10))
```

Exercício 11.19

Crie um programa que contenha uma função que calculo o número de múltiplos de 3 entre dois valores.

11.3 Ficheiros

Python permite aceder a ficheiros em modos de escrita e leitura, através das funções `open()`, `read()`, `readline()` e `write()`.

Em primeiro lugar é necessário criar uma representação do ficheiro que se pretende aceder. O exemplo seguinte abre o ficheiro “texto.txt” em modo de leitura.

```
f = open("texto.txt", "r")
```

Para se abrir um ficheiro noutras modos de acesso, seria necessário utilizar os especificadores:

- “`r`” - Modo de leitura. Não é possível escrever.

- "w" - Modo de escrita. Se o ficheiro não existir, é criado. Se existir, é truncado.
- "a" - Modo de adição. Escritas são adicionadas ao final do ficheiro.
- "r+" - Modo de leitura e escrita. Se o ficheiro não existir, dá erro.

A partir deste momento a variável `f` terá uma representação do ficheiro e permite acesso ao mesmo. Para se efetuar uma leitura é necessário usar o método `read(tamanho)`. O parâmetro tamanho indica quantos bytes se devem ler. Se o valor for menor ou igual que 0, ou omitido, todo o ficheiro é lido para memória. Uma alternativa é utilizar o método `readline()` que obtém apenas uma linha.

A leitura de um ficheiro completo para a memória deve ser usada com parcimónia. Pode justificar-se para ficheiros curtos ou quando se tenha de fazer acessos aleatórios ao conteúdo. Em situações de processamento sequencial, deve optar-se pela leitura incremental por blocos ou linhas.

No final da leitura ou escrita, deve-se sempre fechar o ficheiro que se abriu. Um exemplo completo que imprime para o ecrã o conteúdo de um ficheiro, lendo uma linha de cada vez seria:

```
f = open("texto.txt", "r")

while True:
    linha = f.readline()
    if linha == '':
        break
    print(linha)

f.close()
```

O ciclo `for` permite iterar sobre um grande número de estruturas de dados incluindo ficheiros. Neste caso, o ciclo `for` itera por ficheiros linha a linha. Assim o exemplo anterior poderia ser reescrito da seguinte forma:

```
f = open("texto.txt", "r")

for linha in f:
    print(linha)

f.close()
```

Exercício 11.20

Implemente um programa que imprima o número de caracteres, palavras e linhas de um ficheiro de texto.

Exercício 11.21

Implemente um programa que imprima o conteúdo de um ficheiro, invertendo cada palavra.

As funcionalidades que permitem verificar se um ficheiro existe, se é um ficheiro ou um diretório, etc..., estão disponíveis através do módulo `os.path`. Usando programação defensiva, a verificação de existência de um dado ficheiro pode ser implementada através das seguintes linhas:

```
import os.path
import sys

fname = "Ficheiro.txt"
if not os.path.exists(fname):
    sys.exit("Não existe")

if os.path.isdir(fname):
    sys.exit("É diretório")

if not os.path.isfile(fname):
    sys.exit("Não é ficheiro")

f = open(fname, "r")
```

Exercício 11.22

Melhore os 2 exercícios anteriores de forma a verificar se o ficheiro realmente existe e se pode ser acedido.

11.4 Para aprofundar o tema

Exercício 11.23

Escreva um programa que determine a nota na época normal de um aluno de Laboratórios de Informática e que indique se o aluno está aprovado ou reprovado. Para esse fim o programa deve pedir as notas necessárias (MT1, MT2, MT3, MT4, AP1, AP2, P1 e P2) e calcular a nota final.

Exercício 11.24

Escreva um programa que indique se um número (inteiro positivo) é primo.

Exercício 11.25

Na terra do Alberto Alexandre (localmente conhecido por Auexande Aubeto), o dialecto local é semelhante ao português com duas exceções:

- Não dizem os Rs
- Trocam os Ls por Us

Implemente um tradutor de português para o dialecto do Alberto. Por exemplo “lar doce lar” deve ser traduzido para “ua doce ua”. A tradução deve ser feita linha a linha, até que surja uma linha vazia.

Para este exercício, considere a função `replace` (ver <http://docs.python.org/2/library/string.html#string.replace>).

Exercício 11.26

Escreva um programa que leia uma lista de números e imprima a sua soma e a sua média. O fim da lista é indicado pela leitura do número zero, que não deve ser considerado parte da lista. (Note que se a lista for vazia, a soma será zero, mas a média não pode ser calculada.)

Exercício 11.27

Escreva um programa que implemente o jogo “Adivinha o número!”.

Neste jogo, o programa deve escolher um número aleatório no intervalo $[0; 100]$,^a dando depois a possibilidade de o utilizador ir tentando descobrir o número escolhido. Para cada tentativa, o programa deve indicar se o número escolhido é maior, menor ou igual à tentativa feita. O jogo termina quando o número correcto for indicado, sendo a pontuação do jogador o número de tentativas feito (portanto o valor 1 será a pontuação máxima).

```
aimport random  
random.randint(0,100)
```

12

TEMA

Criptografia em Python

Objetivos:

- Módulos e bibliotecas Python para lidar com criptografia
- Cálculo de sínteses de ficheiros
- Cálculo de sínteses de senhas
- Cifra de ficheiros com criptografia simétrica e assimétrica

12.1 Introdução

A criptografia é um conceito antigo de transformação de conteúdos, que até há cerca de duas décadas era sobretudo usado em ambientes onde a segurança é um elemento fundamental (ambientes militares, serviços de informações, etc.). Hoje em dia, em virtude da massificação do uso da informática e da Internet para os mais variados fins, a segurança criptográfica faz parte do dia-a-dia do cidadão comum, mesmo que disso ele não se aperceba (por exemplo, quando usa comunicações seguras usando HTTPS).

O objetivo desta aula é o de mostrar como se conseguem realizar as transformações criptográficas mais comuns usando módulos Python.

Para uma referência mais profunda sobre a segurança e as suas aplicações recomenda-se a consulta da bibliografia existe[37][38]

12.2 Funções de síntese

As funções de síntese (*digest*) não são funções de cifra convencionais, como as que veremos nas demais secções, mas são funções que usam princípios relacionados com a criptografia

no seu modelo de operação. Para além disso, são muitas vezes usadas em conjugação com funções de cifra para as completar de alguma forma (por exemplo, para calcular chaves de cifra de dimensão fixa a partir de senhas de dimensão arbitrária).

O objetivo de uma função de síntese é o de calcular um valor de dimensão fixa (em número de bits) a partir de conteúdos constituídos por conjunto arbitrários de bits. Normalmente diz-se que estas funções permitem criar *impressões digitais informáticas* (*digital fingerprints*) de conteúdos, porque é muito difícil (por requisito) encontrar dois conteúdos que tenham a mesma síntese, assim como é difícil encontrar dois humanos com as mesmas impressões digitais. Também é comum designar o valor calculado por estas funções como *soma de controlo (checksum)*, muito embora existam inúmeras funções de cálculo de somas de controlo que não possuem as propriedades (criptográficas) das funções de síntese.⁴²

As funções de síntese mais usadas são a MD5, a SHA-1, SHA-256 e SHA-512, mas há muitas mais. Estas funções usam-se de forma similar, mas produzem resultados de dimensão diferente (128, 160, 256 e 512 bits, respetivamente).

A forma usual de aplicar uma função de síntese num programa consiste sem seguir os 4 passos seguintes:

1. Iniciar o seu contexto interno;
2. Adicionar dados para serem processados pela função;
3. Repetir o passo anterior até que tenham acabado todos os dados do conteúdo a processar;
4. Calcular a síntese resultante de todos os dados fornecidos à função.

Em Python o módulo **hashlib**⁴³ possui as funções de síntese mais usuais. O exemplo seguinte mostra a utilização da função MD5 para calcular a síntese de uma frase longa dividida em duas partes.

```
$ python3
>>> import hashlib
>>>
>>> h = hashlib.md5()                                # Iniciar contexto
>>> h.update("A long sentence ".encode('utf-8'))    # Adicionar dados
>>> h.update("broken in two halves".encode('utf-8')) # Adicionar mais dados
>>> print(h.hexdigest())                            # Calcular síntese
f2b8308b3e84e032f5d7e6dee84e647a
```

⁴²Por outras palavras, as funções de síntese podem ser consideradas como funções de cálculo de somas de controlo, enquanto o inverso não é verdade em geral.

⁴³Disponível em <https://docs.python.org/3/library/hashlib.html>

O resultado apresentado por este programa é uma sequência de 32 algarismos hexadecimais. Cada um desses algarismos representa 4 bits ($0 \rightarrow 0000$, $1 \rightarrow 0001$, $2 \rightarrow 0010$, \dots , $8 \rightarrow 1000$, $9 \rightarrow 1001$, $a \rightarrow 1010$, \dots , $d \rightarrow 1101$, $e \rightarrow 1110$, $f \rightarrow 1111$). Logo, o resultado possui 32×4 bits, ou seja, 128 bits.

O resultado apresentado seria o mesmo se a frase tivesse sido fornecida apenas de uma vez e não dividida em duas metades:

```
$ python3
>>> import hashlib
>>>
>>> h = hashlib.md5()
>>> h.update("A long sentence broken in two halves".encode('utf-8'))
>>> print(h.hexdigest())
f2b8308b3e84e032f5d7e6dee84e647a
```

Porém, qualquer pequena alteração do texto processado pela função de síntese muda o resultado de forma radical:

```
$ python3
>>> import hashlib
>>>
>>> h = hashlib.md5()
>>> h.update("A long sentence".encode('utf-8'))           # trailing space removed!
>>> h.update("broken in two halves".encode('utf-8'))
>>> print(h.hexdigest())
1d0b93b21eb945593abab4b1a04456d6
```

Exercício 12.1

Faça um programa que calcule e apresente a síntese de ficheiros usando a função de síntese SHA-1 (cujo nome, no módulo `hashlib`, é `sha1`). Os nomes dos ficheiros deverão ser passados como argumentos ao programa (acessíveis através da variável `sys.argv`). Confirme os resultados apresentados pelo seu programa confrontando-os com os apresentados pelo comando `shasum` disponível na linha de comando UNIX.

Exercício 12.2

Reescreva o programa anterior para calcular a síntese de cada ficheiro usando blocos 512 octetos de cada vez. Use, para esse fim, a função `read` para ler octetos dos ficheiros^a:

```
f = open(name, "rb")
buffer = f.read(512)

# len(buffer) == 0 --> End-of-file reached
# len(buffer) > 0 --> buffer has len(buffer) bytes

while len(buffer) > 0:
    ...
    buffer = f.read(512)
```

Compare os resultados obtidos neste exercício com os do exercício anterior (não deverão mudar!).

^aTorna-se útil abrir o ficheiro em modo de leitura binária usando "rb"

12.3 Biblioteca pycrypto

A biblioteca (*toolkit*) **pycrypto**⁴⁴ possui uma coleção muito interessante de funcionalidades relacionadas com criptografia, incluindo funções de síntese. Daqui em diante iremos usar esta biblioteca.

Instalação

O interpretador de Python tipicamente é acompanhado por duas ferramentas que auxiliam a instalação de módulos adicionais, necessários aos programas. Estes programas são respetivamente o `easy_install` e `pip`. Para esta aula será necessário instalar a biblioteca **pycrypto**, que fornece mecanismos para a utilização de métodos criptográficos.

Para instalar a biblioteca, no caso de se utilizar um sistema pessoal com Ubuntu ou Debian, será necessário executar:

```
sudo apt install python3-crypto
```

ou em alternativa, usando um método mais demorado

⁴⁴<https://www.dlitz.net/software/pycrypto/>

```
sudo apt-get install build-essential python3-dev python3-pip
```

seguido de:

```
pip3 install --user pycrypto
```

Repare na utilização da opção `--user`, que indica que as bibliotecas adicionais deverão ser instaladas apenas para o utilizador atual (e não para todo o sistema). Isto é importante pois os utilizadores comuns não possuem permissões para instalar bibliotecas no sistema.

Pode-se confirmar a existência da biblioteca executando os seguintes comandos:

```
$ python3
>>> from Crypto.Hash import MD5
>>>
>>> h = MD5.new()
>>> h.update( "A long sentence ".encode('utf-8') )
>>> h.update( "broken in two halves".encode('utf-8') )
>>> print(h.hexdigest())
f2b8308b3e84e032f5d7e6dee84e647a
```

Utilização

O programa antes indicado que usava MD5 agora escrever-se-á assim para usar a função MD5 da biblioteca `pycrypto`:

```
$ python3
>>> from Crypto.Hash import MD5
>>>
>>> h = MD5.new()
>>> h.update( "A long sentence ".encode('utf-8') )
>>> h.update( "broken in two halves".encode('utf-8') )
>>> print(h.hexdigest())
f2b8308b3e84e032f5d7e6dee84e647a
```

Exercício 12.3

Altere o programa que antes desenvolveu com a função SHA-1 do módulo `hashlib` para usar a função SHA-256 da biblioteca `pycrypto`. Confirme os resultados confrontando-os com os produzidos pelo comando `sha256sum`.

12.4 Cifras simétricas

As cifras simétricas são funções de transformação (reversível) de conteúdos que usam duas chaves iguais na cifra e na decifra. Ou seja, se se cifrar um conteúdo original T com a função de cifra E e a chave K , produzindo o criptograma C , poder-se-á recuperar T a partir de C com a função de decifra D e a mesma chave K .

As cifras simétricas subdividem-se em duas grandes famílias: as contínuas (*stream*) e as por blocos.

Cifras contínuas (*stream cyphers*)

As cifras contínuas produzem um criptograma C por mistura de um conteúdo original T com uma chave contínua (*keystream*) KS . A decifra consiste em retirar do criptograma a componente KS que lhe foi misturada usando uma função inversa da de mistura. Por simplicidade, a função de mistura e a sua inversa são exatamente a mesma: a adição módulo 2 de bits, vulgarmente designada por **XOR** (de *eXclusive OR*, cujo símbolo matemático é \oplus).

Se A e B forem bits, que podem tomar os valores 0 e 1, a operação \oplus é a seguinte:

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

É fácil de ver que sempre que um operando é 1, o resultado é o inverso do outro operando; pelo contrário, sempre que um operando é 0, o resultado é o valor exato do outro operando. Daqui resulta a seguinte propriedade, para qualquer valor de X :

$$A \oplus X \oplus X = A$$

Logo, é fácil de mostrar que a cifra e decifra se podem fazer usando a operação **XOR** e a mesma chave contínua KS :

$$\begin{aligned} C &= T \oplus KS \\ T &= C \oplus KS = T \oplus KS \oplus KS = T \end{aligned}$$

A diferença entre as várias cifras contínuas está na forma como é produzido o valor de KS . A grande maioria das cifras contínuas usa um gerador pseudoaleatório, controlado por uma chave de dimensão fixa, para gerar KS (tanto para a operação de cifra como para a de decifra).

As cifras contínuas são muito usadas em comunicações rádio envolvendo equipamentos móveis, porque existem geradores muito simples de realizar em hardware e de baixo consumo. É o caso dos geradores A5 (usado nas comunicações GSM) e RC4 (usado inicialmente nas comunicações WiFi). Neste trabalho iremos usar esta última. O seu nome, porém, é diferente: ARC4 (de *Alleged ARC4*). A razão de ser desta diferença de nomes vem do facto do algoritmo RC4 ainda ser oficialmente secreto, sendo o ARC4 uma sua versão que alegadamente (e comprovadamente) é compatível com o RC4 (e que, por isso, é obviamente igual).

A forma usual de usar uma função de cifra/decifra num programa consiste sem seguir os 3 passos seguintes:

1. Iniciar o seu contexto interno, normalmente indicando uma chave;
2. Adicionar dados para serem processados pela função e recolher o resultado da sua operação;
3. Repetir o passo anterior até que tenham acabado todos os dados do conteúdo a processar.

Note-se que com uma cifra contínua o processamento dos dados (na cifra ou na decifra) é por omissão feito sequencialmente, não se podendo cifrar ou decifrar zonas de dados por uma ordem arbitrária. Há cifras contínuas que permitem essa liberdade, mas a sua utilização é diferente. Neste guião não vamos explorar essa faceta.

Exercício 12.4

Faça um programa em Python (`cifraComRC4.py`) que use a cifra RC4 (ou ARC4, na biblioteca `pycrypto`). O programa deverá receber como argumentos o nome de um ficheiro a cifrar e uma chave textual.

O RC4 suporta chaves com uma dimensão entre 40 e 2048 bits (5 a 256 octetos), pelo que deverão ser tomados cuidados para adaptar a chave fornecida pelo utilizador a algo que seja aceitável. Sugere-se a seguinte política: se a chave tiver menos do que 5 octetos (letras), deverá ser usada uma síntese da mesma (calculada, por exemplo, com SHA-1). Se tiver mais dos que 256 octetos, deverão ser usados apenas os primeiros 256. Caso contrário, deverão ser usados exatamente os octetos fornecidos.

O programa deverá escrever o criptograma para o `stdout` (por omissão, a consola), o qual poderá ser redirigido para um ficheiro usando os mecanismos do interpretador de comandos:

```
python3 cifraComRC4.py ficheiro chave > criptograma
```

Em Python a escrita para o `stdout` pode ser feita da seguinte maneira:

```
import os
from Crypto.Cipher import ARC4

cipher = ARC4.new("chave")
cryptogram = cipher.encrypt("Text")
os.write(1, cryptogram)

decipher = ARC4.new("chave")
decrypted = decipher.decrypt(cryptogram)
print( decrypted.decode('utf-8') )

# se o conteúdo for binário, utiliza-se:
# os.write(1, decrypted)
```

Exercício 12.5

Cifre vários ficheiros executando várias vezes o seu programa e verifique se o comprimento dos ficheiros resultantes, contendo os criptogramas, têm a mesma dimensão dos originais (têm de ter!).

Exercício 12.6

Verifique que o programa está a funcionar corretamente decifrando o criptograma usando novamente a mesma chave:

```
python3 decifraComRC4.py criptograma chave > textoDecifrado
```

Após executar este comando deverá surgir, listado no ecrã, o conteúdo do ficheiro original (**ficheiro**).

Exercício 12.7

Execute o comando anterior usando uma chave diferente e veja o resultado. Explique o sucedido.

Cifras por blocos

As cifras por blocos consideram que os dados a transformar (e o resultado da sua transformação) são constituídos por blocos contíguos de dimensão constante; esta dimensão é imposta pela cifra. O modo mais simples de usar uma cifra (ou decifra) por blocos, denominado EBC (*Electronic Code Book*) consiste em realizar os seguintes passos:

1. Iniciar o seu contexto interno, normalmente indicando uma chave;
2. Selecionar o primeiro bloco dos dados a transformar para serem processados pela função e recolher o bloco resultante da sua operação;
3. Repetir o passo anterior para os blocos seguintes até que tenham acabado todos os dados do conteúdo a processar.

Este processo implica que a dimensão total dos dados a transformar seja múltipla da dimensão do bloco (diz-se que estão alinhados ao bloco). Porém, é natural que nem sempre assim aconteça, o que implica que se tenha de forçar esse alinhamento acrescentando dados extra, denominados excipiente (*padding*). Estes dados extra são acrescentados na cifra e removidos na decifra. Há várias maneiras de lidar com os excipientes, mas independentemente do método usado, é preciso indicar ao decifrador a sua presença e dimensão. Uma forma padrão de o fazer, denominada PKCS #7 [39], consiste em fazer o seguinte:

- Acrescentar sempre excipiente, mesmo quando à partida não é necessário (por os dados a cifrar já estarem alinhados);

- Cada octeto do alinhamento tem um valor igual ao comprimento desse alinhamento.

Neste exercício vamos usar a cifra por blocos que é atualmente o padrão, denominada AES (*Advanced Encryption Standard*), que foi a vencedora de um concurso de cifras que terminou pouco depois do início do atual milénio. Esta cifra processa blocos de 128 bits (16 octetos) usando para o efeito chaves de 128, 192 ou 256 bits (16, 24 ou 32 octetos). Quando se usa a biblioteca **pycrypto**, a dimensão do bloco de uma cifra por blocos pode ser obtida através da variável **blocksize** de um objeto de cifra:

```
$ python3
>>> from Crypto.Cipher import AES
>>>
>>> key = '1234567890abcdef'      # Must provide a valid key (with 16, 24 or 32 bytes)
>>> cipher = AES.new( key )
>>> print(cipher.block_size)      # Prints the number of bytes in each block
16
>>> print(cipher.mode)           # Prints the cipher mode (1 for ECB)
1
```

Para se cifrar ou decifrar dados devem-se usar os métodos **encrypt** ou **decrypt**, respetivamente, do objeto de cifra:

```
$ python3
>>> from Crypto.Cipher import AES
>>>
>>> key = '1234567890abcdef'
>>> cipher = AES.new( key )
>>> x = cipher.encrypt( "texto para cifrar" )
>>> print(cipher.decrypt( x ))
b'texto para cifrar'
```

Exercício 12.8

Faça um programa em Python (`cifraComAES.py`) que use a cifra AES. O programa deverá receber como argumentos o nome de um ficheiro a cifrar e uma chave textual.

O AES suporta chaves com uma dimensão exata de 16, 24 ou 32 octetos, pelo que deverão ser tomados cuidados para adaptar a chave fornecida pelo utilizador a algo que seja aceitável. Sugere-se a seguinte política: se a chave tiver menos do que 16 octetos (letras), deverá ser usada uma síntese da mesma (calculada, por exemplo, com SHA-1), de cujo resultado serão usados apenas os 16 primeiros octetos. Caso contrário, deverão ser usados apenas os primeiros 16 octetos da senha fornecida.

O programa deverá escrever o criptograma para o `stdout` (por omissão, a consola), o qual poderá ser redirigido para um ficheiro usando os mecanismos do interpretador de comandos

Exercício 12.9

Faça o programa correspondente de decifra (`decifraComAES.py`). Note que o programa será fundamentalmente igual ao de cifra, mas deverá ter os seguintes cuidados:

- Como os criptogramas estão necessariamente alinhados, não deverá aceitar fazer a decifra de ficheiros que não tenham um comprimento alinhado à dimensão do bloco de cifra. A dimensão de um ficheiro pode ser obtida com a função `os.path.getsize(nome_do_ficheiro)`.
- Não se esqueça de retirar (não escrever) o excipiente colocado durante a cifra.
Não se esqueça de que, se usou o método de colocação de excipiente descrito, existe sempre excipiente no último bloco do ficheiro cifrado!

12.5 Cifras Assimétricas

As cifras assimétricas são cifras que usam **duas chaves**, uma para cifrar e outra para decifrar (designadas por par de chaves). Uma destas chaves designa-se por privada e a outra por pública. A privada só é conhecida por uma entidade, que é dona do respetivo par de chaves; a pública pode ser universalmente conhecida. O conhecimento da chave pública não permite a dedução da correspondente chave privada. Estas cifras também são por vezes designadas por *cifras de chave pública*.

As cifras assimétricas, ou de chave pública, são historicamente muito recentes. Enquanto as cifras simétricas são tão antigas quanto a própria escrita, as assimétricas só existem desde meados de década de 1970. Neste exercício iremos usar a primeira cifra assimétrica

que foi publicada, denominada RSA, que é atualmente a mais usada.

RSA

Como se disse, as cifras assimétricas usam pares de chaves, uma privada e outra pública. O RSA permite cifrar com a pública e decifrar com a privada (para garantir confidencialidade) ou o inverso, cifrar com a privada e decifrar com a pública (para garantir autenticidade, o conceito que está na base das assinaturas digitais).

O RSA opera através da realização de operações matemáticas com números inteiros de grande dimensão (centenas ou milhares de bits). As operações são a exponenciação e o resto da divisão por um número inteiro. À combinação destas duas operações dá-se o nome de exponenciação modular.

Um par de chaves RSA possui 3 elementos:

- Um módulo, n , comum às componentes privada e pública;
- Um expoente, d , pertencente à componente privada;
- Um expoente, e , pertencente à componente pública.

Assim, a chave privada é formada pelo par de valores (d, n) , enquanto a chave pública é formada pelo par de valores (e, n) . As operações de transformação de dados usando estas chaves são as seguintes:

$$\begin{array}{ll} C = T^e \mod n & \text{Cifra com a chave pública (confidencialidade)} \\ T = C^d \mod n & \text{Decifra com a chave privada} \end{array}$$

$$\begin{array}{ll} C = T^d \mod n & \text{Cifra com a chave privada (autenticidade)} \\ T = C^e \mod n & \text{Decifra com a chave pública} \end{array}$$

onde a expressão $x \mod n$ representa o resto da divisão inteira de x por n (em Python seria calculado com a expressão `x % n`).

Ao contrário das cifras anteriores, as cifras assimétricas não usam chaves indicadas por uma pessoa, nem uma pessoa é capaz de memorizar um par de chaves. Os pares de chaves são gerados por programas, usando para o efeito geradores aleatórios de bits, e as chaves geradas por esses programas têm de ser guardadas algures (por exemplo, em ficheiros) para poderem ser usadas mais tarde.

```

$ python3
>>> from Crypto.PublicKey import RSA
>>>
>>> keypair = RSA.generate( 1024 )
>>> fout = open( "keypair.pem", "wb" )
>>> kp = keypair.exportKey( "PEM", "senha" )
>>> fout.write(kp)
>>> fout.close()
...
>>> fin = open( "keypair.pem", "rb" )
>>> keypair = RSA.importKey( fin.read(), "senha" )
>>> fin.close()

```

O exemplo acima mostra como se gera um par de chaves RSA com um módulo de 1024 bits e se guarda o mesmo num ficheiro (**keypair.pem**) codificando o seu conteúdo em PEM (um formato textual) e protegendo a chave privada da observação de terceiros através da cifra com a senha **senha**. O conteúdo do ficheiro terá alguma semelhança com o seguinte:

```

-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,A8751314737B2A42

5JbCg5KVTxYUfheJBVS11G7VkJFf90M0+Q/1FgKRluV5DJWRfw9IzYgy1HBX8VL8
JdmqHo9HRmxdvLUZWSg3nn0UpVaRjiRzFjC+rxtHEWto9o8izmxBhozeaN1MWGEe
Kmt6B2+rrm/KFSrDonuz3r+GitfzGRM2nT+09D1aYQQsS+L7ZZBk/nZ5hLB082S
SWTEQZtrUEQho8o872GVANeA9M2A5urjdPKdav0Dw/CS8a/pD7s96cDEwFOV2Kyw
eG+TMeDDSG7SR+uzQGuEWpzuxciRmwxmlB8C4pxdWArhqu+/2feTt1uDH+PeBQS1
J9/WreLJVwNleo/ZXKhVbp+zL9+IT08b72NmzK1FsF0zk6heh9GIEFJNL6oGVRn8
eD2b1bs6KWVp0XoZrdMdLPDmbjbfaAn+RY15Rvgn0m2Jfs8g+iD1UeLhha05VTIq
3fYOL+QU3scaRUMvkyNbbVZ9/6Tj7GNgWZRMFn5oiKhEtkH9Hsa6esSmAoLmYUol
vYhs1eRk1LnhfiMlg8itfEaX1RKTi5BNne7GxgW63picPf9ryeKnITifpJS+G19+j
3uYOMqCohJqJwk+smYT/QSua7hRXjHLqwZAubhQ4iAVis1WMBUVOfTybF4K6Hub1
jQa/Mgf9hh1jynxohORUvAv+OkuNSICkQcyB2LGf4iKcHbj/5lf5xhpc1NoZVYCw
fARDGXFFHFnURBJ12XjKv8TIDuRFPeA6UxaKhEeD8QDWHc6GGQ7qApMpj8f60Dbk
cPfLAheh9yx7i+xU9rAfHeyu43rHaF4m3XonstPsNxtQR0p586SAQ==
-----END RSA PRIVATE KEY-----

```

Exercício 12.10

Faça um programa em Python que gere um par de chaves e que o guarde num ficheiro. O programa deverá ter como parâmetros o nome do ficheiro para o par de chaves, a chave de cifra e o número de bits da chave.

A cifra RSA não é usada diretamente tal como indicado anteriormente nas expressões matemáticas. Com efeito, muito embora se usem as operações de exponenciação modular referidas, os valores que as mesmas processam na operações de cifra são pré-processados para obter algumas funcionalidades adicionais (controlo de erros e randomização). Há duas formas fundamentais de fazer esse pré-processamento, designadas por PKCS #1 v1.5 e PKCS #1 OAEP (*Optimal Asymmetric Encryption Padding*). Neste guião vamos considerar apenas esta última pois é a mais correta de ser utilizada em novas implementações.

```
$ python3
>>> from Crypto.PublicKey import RSA
>>> from Crypto.Cipher import PKCS1_OAEP
>>>
>>> f = open( "keypair.pem", "r" )
>>> keypair = RSA.importKey( f.read(), "senha" )
>>> cipher = PKCS1_OAEP.new( keypair )
>>> # Encryption w/ public key
>>> x = cipher.encrypt( "The quick brown fox jumps over the lazy dog".encode('utf-8') )
>>> # Decryption with private key
>>> print(cipher.decrypt( x ).decode('utf-8'))
The quick brown fox jumps over the lazy dog
```

Exercício 12.11

Altere os programas que usam AES para usarem também o RSA. O objetivo genérico é fazer a cifra do ficheiro com uma chave pública, recorrendo à privada para fazer a sua decifra. Na prática, vai-se recorrer à chamada cifra mista que tem um resultado semelhante mas um custo muito menor em termos de desempenho. O processo é o seguinte:

- Gera-se uma chave simétrica aleatória para cifrar os dados do ficheiro;
- Cifra-se a chave simétrica com a chave pública do destinatário e acrescenta-se o resultado ao ficheiro cifrado.

Para gerar a chave simétrica aleatória use a função `os.urandom()` com um parâmetro que indique o número de octetos aleatórios desejados. Use a senha fornecida pelo utilizador para decifrar a chave privada do par de chaves RSA.

12.6 Para Explorar

Exercício 12.12

Avalie a performance das diferentes cifras e sínteses que foram exploradas. Para isto, use o módulo `time` para medir quantas operações são efetuadas em 10 segundos.

Exercício 12.13

Sabe que pode enviar mensagens cifradas mesmo sobre canais não seguros, como um chat? Experimente cifrar mensagens de texto e, antes da impressão, converter o criptograma para Base64 usando o módulo `base64`. O resultado final será um texto com caracteres compatíveis com aplicações de comunicação por mensagens. Antes de as decifrar terá de reverter a codificação de Base64.

Consegue comunicar no canal de LABI com outro colega com quem tenha partilhado uma chave?

Testes e Depuração

Objetivos:

- Test Driven Development
- Testes Unitários
- Testes Funcionais
- Depuração

13.1 Introdução

A validação do software é vital para que as equipas consigam realizar entregas dos componentes que desenvolvem, com alguma garantia do funcionamento do código em causa. Ao realizar testes de forma continuada e logo desde o início do desenvolvimento, é possível construir aplicações mais robustas e previsíveis. A existência de testes sistemáticos permite aferir o progresso e detetar regressões no desenvolvimento de uma aplicação. Quando se detetam problemas, é necessário encontrar a razão da anomalia, o que normalmente se faz através de depuração interativa e revisão do código desenvolvido. Este guiaão irá abordar ambos os temas, na perspectiva de pequenas equipas, ou programadores isolados, que pretendem desenvolver aplicações funcionais.

13.2 Desenvolvimento guiado por testes

A metodologia Test Driven Development (TDD) tem ganho adeptos nos últimos anos. Adota uma abordagem alternativa perante o desenvolvimento de aplicações, dando um ênfase especial à definição e execução de testes ao software. Considera-se que não é possível determinar qual o estado de uma aplicação se ela não for testada em todos

os seus componentes. Sem testes sistemáticos é possível que alguns problemas passem despercebidos e venham a provocar problemas no futuro.

Segundo a TDD, o desenvolvimento de uma nova funcionalidade deve começar pela definição de testes à funcionalidade desejada, ainda antes de existir qualquer código. Claro que inicialmente todos os testes irão falhar, pois as funcionalidades não estão implementadas. A este estado dá-se o nome de *RED* (vermelho). À medida que se implementa cada funcionalidade e o teste respetivo é satisfeito, diz-se que estado da funcionalidade passa para *GREEN* (verde). De forma a tornar a aplicação consistente e evitar que se torne uma colagem de funcionalidades, depois de cada teste deverá ser realizada uma análise do que foi produzido e harmonização da solução, a fase *REFACTORING*. Este processo está representado na Figura 13.1. Quando todos os testes forem bem sucedidos, considera-se que a aplicação possui a funcionalidade desejada, para todos os casos de utilização previstos.

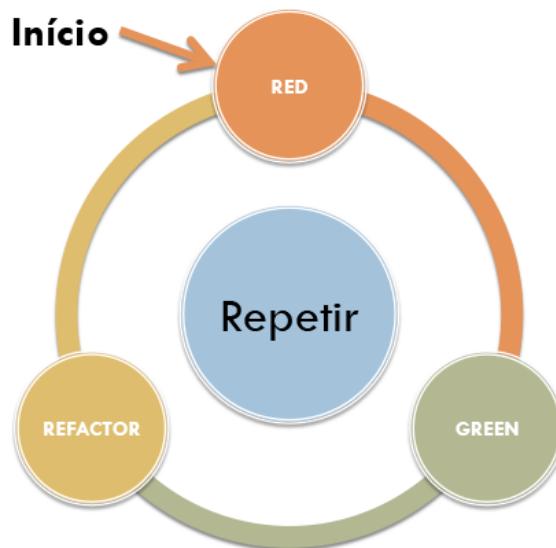


Figura 13.1: Fluxo de processos na metodologia TDD.

Esta metodologia é completamente independente da linguagem de programação, e pode implementar-se sem qualquer ferramenta específica. No entanto, na prática é comum utilizar ferramentas como *Jenkins*,⁴⁵ que de forma pontual executa testes a aplicações, permitindo acompanhar de forma detalhada qual o estado de desenvolvimento da aplicação. Permite igualmente que os programadores evitem um erro muito comum: começar a implementar código da primeira solução, ignorando todos os outros casos que o algoritmo também tem de considerar. Permite igualmente que se pense no problema de forma objetiva e em como será implementado, de uma forma mais distante e considerando o que

⁴⁵<http://jenkins-ci.org/>

seria a arquitetura ótima da solução. Começar imediatamente a programar implica que a solução final irá apenas realizar parte do que é pedido, os módulos irão comportar-se da maneira que dá mais jeito ao programador enquanto desenvolve, e nunca se terá uma caracterização completa da solução implementada.

É importante realçar que existem vários tipos de testes aos quais se pode sujeitar uma solução. Este guia irá focar-se no teste individual dos seus componentes (unidades) e no teste das funcionalidades necessárias da aplicação. Aplicações mais complexas irão necessitar de muitos outros testes.

13.3 Testes Unitários

Os testes unitários são testes aplicados pelos programadores às unidades que compõem os programas que desenvolvem. Uma unidade é um pequeno trecho de código que pode ser testado de forma independente, tal como: parte de uma função, uma função ou uma pequena classe. Podem e devem ser aplicados diversos testes a uma unidade, de forma a cobrir todos os casos de interesse, normais e excepcionais. Se um dado caso não for coberto por um dos testes, esse caso é considerado indeterminado, não se podendo presumir que a solução cobre a situação de forma correta.

Foram criadas diversas ferramentas, adequadas a cada linguagem, que permitem criar testes e automatizar a sua validação. Em *Python* pode-se encontrar o módulo **unittest** ou a ferramenta **py.test**, que iremos usar de seguida. Em *Java* a classe **JUnit** apresenta funcionalidades semelhantes. Muitas outras plataformas permitem realizar o mesmo. Encontra uma lista no endereço http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks.

O programa **py.test** permite a execução e validação de testes em programas *Python*, de

forma simplificada.

Exercício 13.1

Verifique se o comando `py.test` está disponível no seu computador, executando:

```
$ py.test-3
```

Se aparecer uma mensagem parecida com esta:

```
platform linux -- Python 3.5.3, pytest-3.4.1, py-1.5.2, pluggy-0.6.0
rootdir: /home/debian/Desktop, inifile:
collected 0 items

===== no tests ran in 0.00 seconds =====
```

então já está instalado.

Se aparecer uma mensagem de erro, terá de instalar o pacote `pytest`. Para isso é necessário realizar **uma** das seguintes operações.

Se tiver permissões para a instalação de pacotes, num sistema *Ubuntu* ou *Debian*, pode executar como administrador:

```
$ apt install python3-pytest
```

Se não tiver permissões, pode instalar a aplicação na área pessoal usando um instalador de pacotes `python`:

```
$ pip3 install --user pytest      # instala o pacote
$ export PATH=~/local/bin:$PATH   # indica à shell onde encontrar o comando
```

Isto irá instalar a aplicação em `~/local/bin/py.test`.

Vejamos um exemplo de utilização da ferramenta `py.test` para o desenvolvimento de uma função segundo a metodologia TDD.

Considere que pretendemos criar uma função chamada `fibonacci`, que determine os `n` primeiros valores da sequência de Fibonacci. Antes de iniciar a implementação devem especificar-se os testes, e estes devem cobrir tanto os casos considerados normais, como

os casos especiais. Decidimos que a função irá aceitar um argumento e irá devolver depois uma lista com a sequência, em que o tamanho da lista deverá ser igual ao valor especificado no argumento.

Começamos por enumiciar os seguintes testes:

- Para valores de n inferiores a 0 a função deverá devolver uma lista vazia.
- Para n igual a 0 a função deverá devolver [0].
- Para n igual a 1 a função deverá devolver [0, 1].
- Para n igual a 2 a função deverá devolver [0, 1, 1].
- Para n igual a 5 a função deverá devolver [0, 1, 1, 2, 3, 5].
- Para qualquer n a função deverá devolver uma lista com $n + 1$ elementos.

Após a definição destes testes podemos iniciar o desenvolvimento, tratando cada um dos testes de forma isolada. Isto envolve implementar o código, validar o teste e re-arranjar o código de forma a continuar consistente. Neste caso, o primeiro teste requer que a função tenha a seguinte estrutura:

```
def fibonacci(n):
    res = []
    if n < 0:
        return res
```

Como é óbvio, esta implementação não pode ser considerada correta pois caso `n` seja superior ou igual a 0 a função não irá devolver nada. No entanto, do ponto de vista do primeiro teste ele será concretizado com sucesso. A implementação pode então continuar, procurando satisfazer os testes seguintes, **um de cada vez**, até que todos sejam bem sucedidos.

A execução dos testes pode ser feita de forma bastante simples, recorrendo a sequências de instruções que validam o valor devolvido pelas funções. O primeiro teste poderia ser implementado fazendo:

```
def teste1():
    if fibonacci(0) == [0] and fibonacci(-1) == []:
        print("Teste OK")
    else:
        print("Teste Falhou")
```

Esta função irá imprimir **Teste OK** ou **Teste Falhou** dependendo do valor devolvido pela função. No entanto, isto não é suficiente para, de uma forma sistemática, executar testes e avaliar a situação do desenvolvimento da aplicação. Por isso recomenda-se a utilização da ferramenta **pytest**.

Para usar esta ferramenta, os testes têm de ser codificados em funções com nomes começados por **test_**. Normalmente essas funções são definidas em ficheiros auxiliares com nomes começados por **test_** e com extensão **.py**. Por exemplo, o ficheiro **test_fibonacci.py** poderia começar por definir uma função para o primeiro teste que enunciámos:

```
# test_fibonacci.py
import pytest
from fibonacci import fibonacci

def test_inferior_1():
    print("Testa comportamento com n < 1")
    assert fibonacci(0) == []
    assert fibonacci(-1) == []
```

De notar que é necessário importar a função **fibonacci**, que deverá estar num ficheiro chamado **fibonacci.py**. A palavra reservada **assert** tem o significado de *afirmar* e usa-se para verificar *asserções* ou seja, condições que se presume serem sempre verdadeiras. Num programa normal, se a condição avaliada num **assert** for falsa, é gerada uma exceção que interrompe o programa com uma mensagem de erro informativa. Pelo contrário, quando executada pelo **py.test**, uma asserção falsa é reportada como uma falha do teste, mas a verificação avança para os testes seguintes.

Exercício 13.2

Implemente os testes que definimos acima no ficheiro **test_fibonacci.py** e verifique que executam através da ferramenta **py.test**. Depois de todos os testes estarem implementados, implemente progressivamente o código num ficheiro **fibonacci.py** de forma a cumprir cada um dos testes.

Exercício 13.3

Defina testes unitários para seis funções que realizem as operações aritméticas de soma, subtração, multiplicação, divisão, resto da divisão inteira e raiz quadrada sobre valores reais.

Após a implementação dos testes, implemente as funções de forma a cumprirem os testes desenvolvidos. Tenha em consideração os valores aceitáveis (o domínio) para cada uma das operações.

Tenha igualmente em atenção que os valores reais possuem limite de precisão. Pode verificar isto se utilizar a linguagem Python para somar 1.3 com 1.6. Poderá ter de utilizar a função `round` para limitar a precisão e evitar estes erros de aproximação.

13.4 Testes Funcionais

Os testes unitários dedicam-se à verificação de componentes do software tão pequenos quanto possível. Mas uma aplicação é formada por muitos componentes que interagem de forma complexa e a correção dos componentes isolados não basta para garantir a correção da aplicação. Para verificar a funcionalidade de uma aplicação é necessário considerar aspectos de execução e de interface com o utilizador, tomando a aplicação como um todo. Esse é objetivo dos testes funcionais.

Os testes funcionais podem ser encarados da mesma forma que os unitários, mas focam-se sobre aspectos mais amplos da aplicação. Eles devem ser desenvolvidos de forma única para cada aplicação, visto que cada aplicação possui funcionalidades distintas. No caso dos exemplos tratados anteriormente, faz sentido avaliar os resultados debitados pelo programa (o seu *output*), quando se lhe fornece certos dados (o seu *input*).

A ferramenta `py.test` também pode ser utilizada para testes funcionais, pois é possível executar uma aplicação, capturar o que escreve para o ecrã e comparar o resultado obtido com o esperado. A chave para isto é o módulo `subprocess` e a classe `Popen`. Usando estas classes é possível executar uma aplicação da seguinte forma:

```
from subprocess import Popen
from subprocess import PIPE

proc = Popen("comando a executar", stdout=PIPE, shell=True)

return_code = proc.wait()
output = proc.stdout.read().decode('utf-8')
```

A linha `proc.stdout.read()` obtém tudo o que a aplicação escreve para o ecrã. Para se iterar sobre cada linha em tempo real seria possível fazer:

```
...  
for line in iter(proc.stdout.readline, b''):  
    print( line.decode('utf-8'), end='')
```

Exercício 13.4

Implemente um programa que execute o comando "ls -la "+sys.argv[1] e imprima o seu resultado, mas descartando qualquer linha que contenha um termo fornecido no segundo argumento ao programa (`sys.argv[2]`). Pode verificar se uma linha contém um termo usando `if termo in line:`.

O exemplo anterior pode ser combinado de forma a ser parte integrante de um teste. Pode-se comparar a impressão da execução de um comando e o seu código de retorno, bastando para isso que o código pertença a uma função com nome iniciado por `test_`.

Exercício 13.5

Defina um conjunto de testes **funcionais** para as aplicações consideradas anteriormente (Fibonacci e Calculadora). Considere a introdução de argumentos inválidos (*Strings*), a total ausência de argumentos, ou a introdução de valores inválidos para operações específicas. Em cada caso a aplicação deverá devolver mensagens de erro específicas.

Implemente os testes usando o formato necessário pela ferramenta `py.test`.

Não implemente qualquer funcionalidade na aplicação que vá de acordo aos testes.

Exercício 13.6

Ordene os testes e, um de cada vez, implemente as funcionalidades necessárias para a aplicação os passar com sucesso.

13.5 Depuração

Um programa pode apresentar uma miríade de erros que impedem o seu funcionamento correto. No contexto da linguagem *Python* podemos encontrar:

Erros de sintaxe: Encontrados pelo interpretador de *Python* quando converte o código fonte para instruções. Estes erros são detetados imediatamente após a tentativa de execução de uma aplicação, resultando numa mensagem de **SyntaxError: invalid syntax**. Parêntesis ou outro carácter em falta, indentação incorreta, ou erros na escrita das palavras reservadas levam a que seja produzido este erro.

Erros de execução: Encontrados pelo interpretador quando uma situação excepcional é encontrada durante a execução. São situações como uma divisão por zero, ou uma operação entre tipos incompatíveis, que ocorrem devido ao fluxo de execução particular, não sendo possível prever esta situação quando o ficheiro é carregado.

Erros semânticos: O programa executa sem detetar qualquer erro, mas o resultado não é o esperado. Os testes funcionais e unitários podem detetar estes erros.

O primeiro tipo de erros é detetado facilmente pelo interpretador, que os localiza e identifica. Os erros de execução também são detetados pelo interpretador, mas não é detetada a sua causa. O terceiro tipo de erros apenas pode ser detetado com testes, mas mais uma vez, apenas se saberá que o erro existe dentro de uma unidade ou de uma funcionalidade, não se sabendo em concreto qual o problema.

Para estes casos existem mecanismos nas linguagens e ferramentas que permitem executar de forma interativa uma aplicação, sendo possível inspecionar cada ponto da execução. Estas ferramentas, denominadas por depuradores (*debuggers*), executam os programas fornecendo ao programador muito mais controlo sobre a sua execução. São exemplos o módulo **pdb**, o **pydbgr** ou o **ipdb**, todos eles disponíveis para instalação com **pip** ou **easy_install**. Alguns Integrated Development Environment (IDE) já possuem esta funcionalidade incluída, como é o caso do Eclipse⁴⁶, do Microsoft Visual Studio⁴⁷ ou do PyCharm⁴⁸, entre outros.

Considere o seguinte programa que, de uma forma simples (e incorreta) tenta verificar se um dado número é primo:

```
import sys

def main(argv):
    num = int(argv[1])
    for x in range( num//2 ):
        if num % x != 0:
            print("False")
    print("True")
```

⁴⁶<https://eclipse.org/>

⁴⁷<http://www.visualstudio.com/>

⁴⁸<https://www.jetbrains.com/pycharm/>

```
main(sys.argv)
```

Este programa não apresenta nenhum erro sintático. No entanto apresenta um erro de execução e um semântico. Neste caso é simples de detetar por revisão do código, mas poderia não ser, servindo mesmo assim como exemplo para utilização de um depurador.

Para instalar o depurador **ipdb** poderá usar o gestor de pacotes da sua distribuição Linux, no caso da VM da disciplina (Debian) execute:

```
$ apt install python3-ipdb
```

Alternativamente, poderá usar o gestor de pacotes do Python3, executando:

```
$ pip3 install --user ipdb
```

Depois, pode-se iniciar o depurador para um dado programa executando:

```
$ python3 -m ipdb prime.py 10
```

Se tiver problemas a instalar este depurador, pode utilizar o depurador integrado **pdb**.

De notar que os argumentos, neste caso 10, são passados para o programa tal como se não estivesse a ser depurado.

A partir deste momento o programa é carregado e está pronto a ser depurado.

```
> prime.py(1)<module>()
----> 1 import sys
      2
      3 def main(argv):
ipdb>
```

O depurador indica a instrução que está pronta para ser executada, apresenta um prompt e aguarda por comandos.

Alguns comandos bastante úteis que se podem utilizar são os seguintes:

continue: Continua a execução;

run: Volta a executar o programa;

break <nome-da-funcao>: Define que a execução deve ser parada na função indicada;

break <numero-da-linha>: Define que a execução deve ser parada na linha de código indicada;

print(<nome-da-variavel>): Mostra o valor de uma variável;

list: Lista o código fonte;

next: Executa a próxima instrução;

step: Executa a próxima instrução.

Se a instrução chamar uma função, a execução interativa entra dentro da função, de forma a que a próxima instrução a executar interactivamente seja a primeira instrução da função.

Neste caso interessa definir um *breakpoint* na linha 4 e depois executar cada instrução passo a passo até encontrar um erro.

O resultado seria:

```
> prime.py(1)<module>()
----> 1 import sys
      2
      3 def main(argv):
          ipdb> break 4
          Breakpoint 1 at prime.py:4
          ipdb> continue
> prime.py(4)main()
      3 def main(argv):
      4     num = int(argv[1])
      5     for x in range( num//2 ):
          ipdb> n
> prime.py(5)main()
      1     4     num = int(argv[1])
      2     ----> 5     for x in range( num//2 ):
      3         6         if num % x != 0:
          ipdb> n
> prime.py(6)main()
      5     4     for x in range( num//2 ):
      6         5         if num % x != 0:
      7             6             print("False")
```

```
ipdb> n
ZeroDivisionError: integer division or modulo by zero
> prime.py(6)main()
  5      for x in range( num//2 ):
----> 6          if num % x != 0:
    7              print("False")

ipdb> print(num)
10
ipdb> print(x)
0
ipdb>
```

Após a ocorrência do erro, pode-se verificar em que linha aconteceu e inspecionar o valor das variáveis atuais. Neste caso, verifica-se que a variável `x` é igual a 0, o que gera uma divisão por 0.

Sem o depurador seria possível obter informação do erro, mas não poderíamos inspecionar a memória no momento do erro. Um aspecto interessante do depurador é que ele permite inspecionar a memória de uma aplicação quando são encontrados problemas, mesmo que não tenhamos definido um *breakpoint* previamente.

Exercício 13.7

Volte a executar o programa no depurador e não defina nenhum *breakpoint* nem execute o programa de forma interativa. Simplesmente forneça a instrução de `continue`. Repare que o programa é interrompido na ocorrência de um erro.

Verifique que pode inspecionar o estado de todas as variáveis e pode mesmo alterar o seu conteúdo através de sintaxe *Python* (p. ex. `x = 1`).

Exercício 13.8

Corrija o erro em causa e voltando a utilizar o depurador, verifique a correta execução do programa. Caso encontre um problema, corrija-o e volte a iniciar uma sessão de depuração.

Exercício 13.9

Crie e implemente testes funcionais de forma a validar o programa acima referido. Ele deverá aceitar um argumento inteiro superior a 0 na linha de comandos e determinar se é primo ou não. O resultado é apresentado através da impressão das palavras **False** ou **True** e definição do código de execução: 1 para primo, 0 para não primo.

13.6 Para Aprofundar

Exercício 13.10

Construa testes unitários para os programas do segundo guião de Programação 2. Utilize para isso a linguagem Java. Em Java existe a classe *JUnit* que pode facilitar a sua especificação e que possui funcionalidades semelhantes à ferramenta **py.test**. Pode encontrar exemplos de como utilizar a class *Junit* no endereço:
<https://github.com/junit-team/junit/wiki/Getting-started>

Exercício 13.11

Construa testes funcionais para os programas do segundo guião de Programação 2 e valide a sua execução. Utilize para isso a linguagem Java ou Python.

Comunicação entre aplicações

Objetivos:

- Comunicação entre aplicações
- Sockets UDP
- Sockets TCP
- Acesso Assíncrono

14.1 Introdução

As aplicações informáticas realizam trabalho sobre dados que lhes são fornecidos, o que tem sido feito através de argumentos, introdução de informação pelo teclado, ou de ficheiros. No entanto também é possível e extremamente útil as aplicações trocarem informação diretamente entre si. Um exemplo muito comum é a consulta de páginas na Internet. Segundo a indicação do utilizador, um navegador Web irá consultar os diversos servidores de forma a obter páginas. Tanto o navegador, como o(s) servidores Web são aplicações que possuem a capacidade de trocarem informação, o que é feito através de mensagens que são transmitidas através de uma rede. A construção e transmissão das mensagens foi já abordado anteriormente. Este guião aborda a geração, envio e recepção das mensagens pelas aplicações.

14.2 Conceitos de comunicação

Modelo Cliente-Servidor

Nas comunicações entre aplicações é usual distinguir-se dois tipos de intervenientes: o cliente e o servidor. O cliente é aquele que inicia a comunicação e faz um pedido, enquanto o servidor é aquele que aceita um pedido e realiza uma ação, podendo emitir uma resposta. Por exemplo, o navegador Web é um cliente que emite pedidos enquanto os servidores Web são aqueles que aceitam pedidos dos clientes, fornecendo depois as páginas como resposta. A Figura 14.1 representa uma simples interacção entre vários clientes e um servidor.

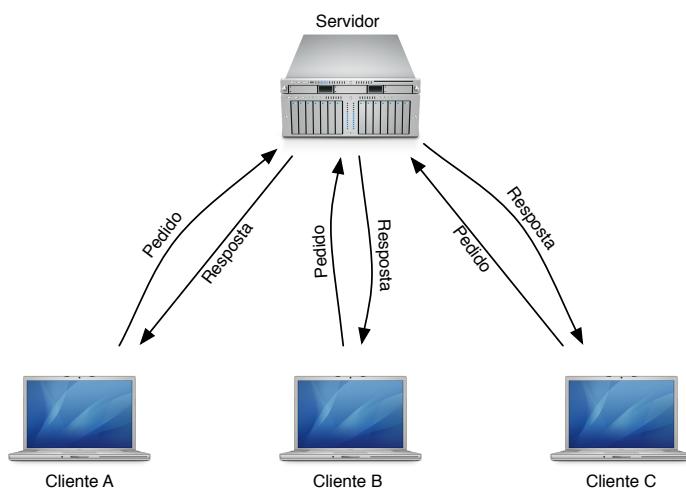


Figura 14.1: Modelo Cliente-Servidor

Esta separação de funções é importante porque implica que as aplicações do cliente e do servidor são distintas: o fluxo lógico (algoritmo) do cliente é diferente do fluxo lógico do servidor. Considerando o mesmo exemplo, os servidores Web têm de fornecer páginas e outros ficheiros, enquanto os clientes têm de processar HTML, JS, CSS, construir a página e interagir com o cliente. Outra característica deste modelo é que vários clientes se podem ligar a um único servidor.

Existem aplicações, como as utilizadas nas redes Peer to Peer (P2P) que são construídas de forma a funcionarem ao mesmo tempo como cliente e servidor. Isto não vai contra o dito anteriormente relativamente a existirem dois tipos de aplicações, pois para suportar esta modalidade, estas aplicações têm de possuir código para ambas as funções. Quando um aplicação P2P comunica com outra, durante esta transação essa age como servidor e a outra como cliente.

Sockets

As aplicações na consola podem interagir com o utilizador através de três dispositivos básicos: **stdin**, **stdout** e **stderr**. Estes dispositivos têm sido extensivamente utilizados quando se lê texto do teclado ou se escreve para a consola. As aplicações também podem criar representações internas de ficheiros e diretórios, que depois utilizam para ler, escrever, criar ou apagar estes elementos. Para permitir a comunicação entre processos, ou Inter Process Communication (IPC), existem outros mecanismos como o Socket. O termo inglês *Socket* designa uma tomada onde se pode ligar uma ficha, tal como uma tomada de corrente eléctrica ou uma tomada de rede ethernet. No âmbito do software, um *Socket* é um mecanismo pelo qual as aplicações podem criar tomadas virtuais onde outras aplicações se podem “ligar”. Uma aplicação pode criar quantos *Sockets* quiser, tal como uma casa pode ter quantas tomadas forem necessárias. A Figura 14.2 demonstra uma aplicação com vários pontos de comunicação, sendo que alguns são *Sockets*.

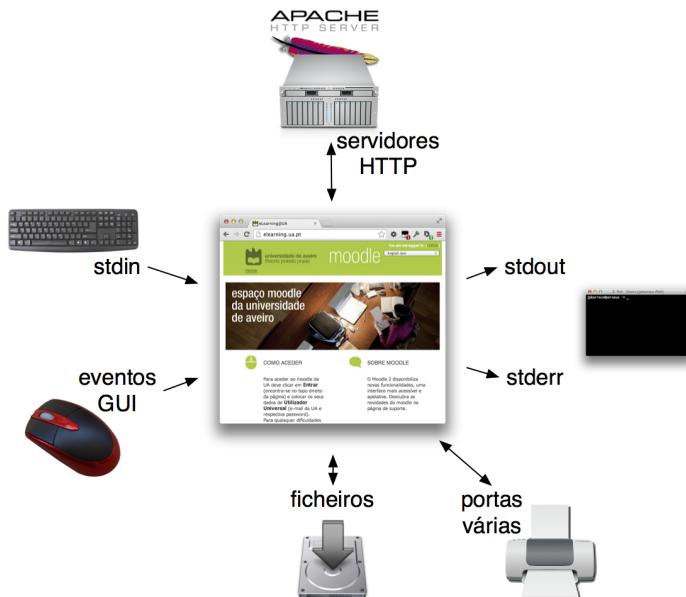


Figura 14.2: Vários pontos de comunicação com um navegador Web.

Tal como uma tomada possui um dada especificação, quanto à sua forma e contactos, também um *Socket* possui algumas características distintivas, das quais se destacam a *família* e o *tipo*. Um *Socket* de uma dada família e tipo só aceita ligações com essas características e não com outras. Existem várias famílias de *Socket*. As principais são:

AF_UNIX: Indica um *Socket* para ser utilizado em comunicações entre aplicações

locais (na mesma máquina).

AF_INET: Indica um *Socket* para ser utilizado sobre endereços IPv4. Pode ser utilizado para comunicações entre aplicações locais ou remotas.

AF_INET6: Indica um *Socket* para ser utilizado sobre endereços IPv6. Pode ser utilizado para comunicações entre aplicações locais ou remotas.

Neste caso, um *Socket* da família **AF_INET** pode ser utilizado para trocar mensagens com aplicações que estejam em sistemas com um IPv4 disponível, mas não permite trocar mensagens com aplicações que estejam em sistemas que só possuam IPv6.

Além da família, o tipo do *Socket* irá indicar como as mensagens devem ser encapsuladas antes de serem enviadas. Há dois tipos mais relevantes:

SOCK_DGRAM: As comunicações deverão utilizar o protocolo User Datagram Protocol (UDP)[40]. Com este tipo de *Socket*, é possível que as mensagens se percam ou cheguem fora de ordem.

SOCK_STREAM: As comunicações deverão utilizar o protocolo TCP. Com este tipo de *Socket*, em caso de perda, o protocolo TCP irá retransmitir as mensagens. Este também garante que a ordem de envio é mantida à chegada.

14.3 Sockets não orientados à ligação (UDP)

Um tipo de *Sockets* permite o envio de mensagens entre aplicações sem que estas estabeleçam uma ligação explícita persistente. Entre muitos outros cenários, são indispensáveis para envio de mensagens de broadcast, para sistemas com baixas capacidades computacionais, ou para comunicações com restrições de tempo real. Este tipo de *Sockets* são normalmente utilizados nas redes IP TeleVision (IPTV) que temos em casa, nas aplicações Voice Over IP (VoIP) como o *Skype*, ou no processo de atribuição de endereços dinâmicos (DHCP). Este tipo de *Socket* utiliza o protocolo UDP para o transporte das mensagens.

Um *Socket*, seja ele orientado à ligação ou não, não realiza comunicações de forma imediata só porque existe. Comunicar com um *Socket* requer uma sequência de acções, nomeadamente:

Criação: Um *Socket* é criado, definindo-se a sua família e tipo. Utiliza-se para isso a instrução **socket**.

Nomeação: É necessário dar um nome ao *Socket*, usando-se a primitiva **bind**. O nome permite identificar **unicamente** um dado *Socket* num dado sistema que pode ter

inúmeros Sockets de várias aplicações. As aplicações trocam informação a partir de e para um Socket em particular. O nome é composto por um caminho, normalmente um endereço IPv4 e um porto (ou porta).

Enviar Informação: A aplicação emissora usa a acção **send** para enviar informação para o Socket. O sistema de destino armazena essa informação numa memória associada ao Socket, temporariamente.

Receber Informação: A aplicação recetora usa a acção **receive** para recolher a informação recebida e armazenada pelo Socket.

Fechar: Tal como um ficheiro, o Socket deve ser fechado quando a comunicação termina.

A Figura 14.3 representa uma utilização destas primitivas numa comunicação entre uma aplicação cliente e uma aplicação servidora. Note que não há nenhuma diferença formal entre um Socket no servidor ou no cliente, mas a lógica das aplicações é diferenciada. Uma espera por pedidos, a outra efetua pedidos.

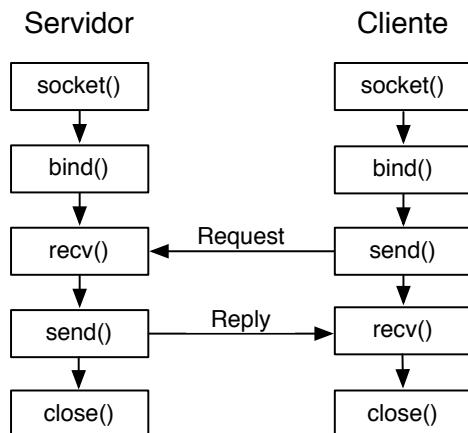


Figura 14.3: Sequência de primitivas utilizadas num Socket UDP.

Em Python é possível utilizar Sockets através da inclusão do módulo **socket**. Assim, um Socket pode ser criado e nomeado através das seguintes instruções:

```
import socket

def main():
    udp_s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    udp_s.bind( ("127.0.0.1", 1234) )

main()
```

Neste exemplo o *Socket* é criado de forma a comunicar na porta **1234** para aplicações no próprio computador (**127.0.0.1**). Este endereço determina em que interface de rede o *Socket* vai comunicar, sendo que o valor especial **0.0.0.0** indica que irá comunicar através de todos os interfaces. Se o valor da porta for igual a 0, o sistema operativo irá escolher uma porta aleatória.

Atenção: Num dado sistema, não é possível existirem dois *Sockets* com o mesmo nome (endereço e porta)!

Para a troca de informação, é agora necessário receber e enviar informação. O exemplo seguinte espera por uma mensagem e responde enviando a mesma mensagem em maiúsculas. De notar que se utiliza o método **recvfrom** que possui como parâmetro o número máximo de octetos a receber e devolve 2 valores: uma string com os dados recebidos e o endereço do socket que os enviou.

```
...
def main():
    ...
    while 1:
        b_data, addr = udp_s.recvfrom(4096)
        udp_s.sendto(b_data.upper(), addr)
    udp_s.close()
...
```

Exercício 14.1

Utilizando os exemplos anteriores, implemente um servidor de mensagens UDP.
Pode testar o servidor utilizando o comando **nc -u localhost 1234**.

O cliente para comunicar com esta aplicação seria programado de maneira muito semelhante. Apenas teria as instruções de envio e recepção por ordem inversa: primeiro envia e depois recebe a resposta. O trecho seguinte demonstra um cliente que lê uma frase do teclado, envia-a para o servidor, espera uma resposta e imprime-a.

```
...
def main():
    ...
    udp_s.bind(("127.0.0.1", 0))
    server_addr = ("127.0.0.1", 1234)
    while 1:
        str_data = input("<-:")
        b_data = str_data.encode("utf-8")
        udp_s.sendto(b_data, server_addr)
        # ---
        b_data, addr = udp_s.recvfrom(4096)
        str_data = b_data.decode("utf-8")
        print("->: %s \n" % str_data)

    udp_s.close()
...

```

Exercício 14.2

Utilizando os exemplos fornecidos implemente um cliente que permita a troca de mensagens. Tenha em consideração que, por residirem no mesmo sistema, o cliente não pode criar um *Socket* na mesma porta que o servidor.

Exercício 14.3

Coordene com o grupo ao seu lado de forma a executarem um servidor com um *Socket* no endereço 0.0.0.0. Deverá conseguir enviar mensagens para este servidor se o seu cliente se ligar ao endereço do computador onde reside o servidor.

14.4 Sockets orientados à ligação (TCP)

Além das primitivas já vistas, um *Socket* do tipo **SOCK_STREAM** necessita de mais três. Isto deve-se ao facto dos *Sockets* deste tipo serem orientados à ligação, existindo a necessidade de se estabelecer uma ligação (ou sessão) entre as duas aplicações antes da transmissão de informação.

Aceitar Ligações: Um *Socket* do servidor irá ser definido como aceitando ligações de novos clientes, realizando-se esta ação através da instrução **listen**.

Estabelecer ligação: O cliente necessita de se ligar ao servidor antes de se poder trocar informação, usando-se para isso a instrução **connect**.

Aceitar Clientes: O servidor, após receber o pedido de ligação, pode aceitá-lo através da instrução `accept`.

Estas primitivas são utilizadas da forma descrita na Figura 14.4. Como pode ser visto, neste tipo de Sockets, **existe** uma declaração formal de que Socket é servidor ou cliente e a lógica das aplicações é também diferenciada. Uma aplicação espera por pedidos, outra efetua pedidos. As primitivas adicionais, necessárias ao estabelecimento da sessão, estão realçadas na figura.

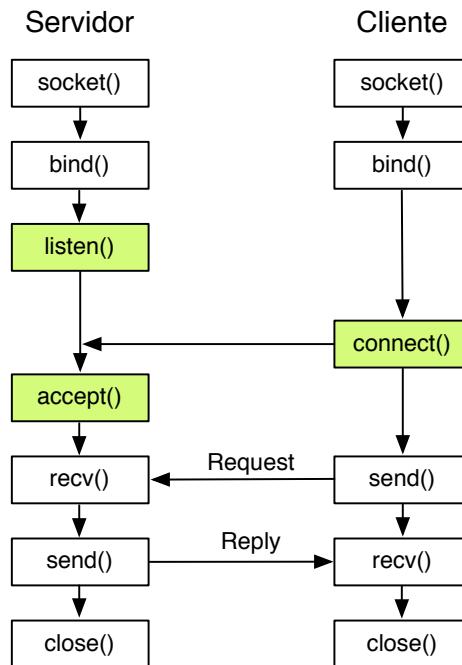


Figura 14.4: Sequência de primitivas utilizadas num Socket TCP.

Utilizando *Python* é possível utilizar Sockets orientados à ligação de uma forma semelhante aos anteriores, mas agora considerando a necessidade de estabelecimento da ligação antes da troca de informação. Assim, um Socket TCP pode ser criado e nomeado através das seguintes instruções:

```
# encoding=utf-8
import socket

def main():
    tcp_s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_s.bind( ("127.0.0.1", 1234) )

main()
```

Para o estabelecimento da sessão é agora necessário que o servidor defina o *Socket* como aceitando ligações e que espere por novos clientes, aceitando-os depois. Note que quando se aceita uma ligação de um novo cliente é criado um novo **Socket**, que é utilizado para identificar a ligação entre o servidor e o cliente.

```
...
def main():
    ...
    # máximo de 1 cliente à espera de
    # aceitação
    tcp_s.listen(1)

    # esperar por novos clientes
    client_s, client_addr = tcp_s.accept()

    while 1:
        b_data = client_s.recv(4096)
        client_s.send(b_data.upper())

    client_s.close()
    tcp_s.close()
...
```

De notar que neste caso utiliza-se o *Socket* chamado **client_s** para todas as futuras comunicações. Também se utilizam os métodos **recv** em vez de **recvfrom** e **send** em vez de **sendto**. Isto porque não é necessário saber de onde chegou ou para onde vai a informação: toda a informação que seja lida ou escrita no **client_s** é trocada entre o servidor e o cliente específicos.

Exercício 14.4

Implemente um servidor TCP utilizando o exemplo fornecido. Pode testar o servidor utilizando o comando **telnet localhost 1234**, ou o comando **nc localhost 1234**.

O cliente será bastante semelhante ao caso dos Sockets não orientados à ligação, mas neste caso, é necessário estabelecer previamente a sessão.

```
...
def main():
    ...
    # Ligar ao servidor
    tcp_s.connect( "127.0.0.1", 1234 )
    while 1:
        str_data = input("Mensagem: ")
        b_data = str_data.encode("utf-8")
        tcp_s.send(b_data)
        # ---
        b_data = tcp_s.recv(4096)
        str_data = b_data.decode("utf-8")
        print(str_data)

    tcp_s.close()
...

```

Tal como acontece com a implementação do servidor, a troca de informação é feita através dos métodos `recv` e `send`, não existindo necessidade de especificar sempre qual o destino das mensagens, ou de obter informação relativa à sua origem.

Exercício 14.5

Implemente um cliente que utilize Sockets orientados à ligação. Mais uma vez, tenha em atenção que os clientes devem utilizar portas aleatórias de forma a não colidirem com serviços existentes.

14.5 Servidor de Mensagens Instantâneas

De uma forma simples é possível implementar um servidor que actue como uma ferramenta de *chat*, permitindo a troca de mensagens entre utilizadores. Este exemplo é útil pois permite demonstrar com é possível interagir com múltiplos clientes e, no caso do cliente, ler de forma alternada de duas fontes de informação. De notar que o servidor e o cliente podem ser implementados utilizando qualquer um dos tipos de Socket abordados. De forma a simplificar a explicação, o exemplo irá focar-se no caso de comunicações através de UDP deixando-se a implementação com TCP para exercício.

A implementação do servidor é simples, bastando que possua uma lista de Sockets conhecidos e envie as mensagens recebidas de um Socket para todos.

```
...
def main():
    ...
    # Lista de sockets conhecidos
    addr_list = []

    while 1:
        b_data, addr = udp_s.recvfrom(4096)
        print(b_data.decode("utf-8"))

        # Adicionar o nome do socket à lista de sockets conhecidos
        if not addr in addr_list:
            addr_list.append(addr)

        # Enviar a mensagem para todos
        for dst_addr in addr_list:
            udp_s.sendto(b_data.upper(), dst_addr)
```

Exercício 14.6

Implemente um servidor como o indicado no exemplo. Este deverá funcionar com os clientes UDP criados anteriormente. No entanto o cliente não irá suportar a nova lógica, pelo que não serão apresentadas as mensagens de todos os clientes.

Um cliente de *chat* também necessitaria de pequenas alterações de forma a permanentemente ouvir novos dados do teclado e do *Socket*. O teclado permite ao utilizador enviar mensagens, o *Socket* permite receber as mensagens dos outros clientes. Isto é possível através da utilização da instrução **select**, que basicamente permite ficar à escuta de informação de múltiplas fontes, indicando depois qual das fontes possui informação para ser consumida.

O método **select** aceita três parâmetros: a lista de *Sockets* (ou outros dispositivos) onde se espera por dados, a lista de *Sockets* onde recentemente foram escritos dados e se espera que estes sejam transmitidos e a lista de *Sockets* onde se querem receber notificações de exceções (p.ex, *Socket* fechado). Irá devolver igualmente três listas com os *Sockets* que tiveram os eventos respetivos. Aqui apenas nos interessa a primeira das listas, a que indica os *Sockets* com informação pronta a ser consumida.

```
import select
...
def main():
    ...
    while 1:
        rsocks = select.select([udp_s, sys.stdin, ], [], [])[0]

        for sock in rsocks:
            if sock == udp_s:
                # Informação recebida no socket
                b_data, addr = udp_s.recvfrom(4096)
                sys.stdout.write("%s\n" % b_data.decode("utf-8"))
            elif sock == sys.stdin:
                # Informação recebida do teclado
                str_data = sys.stdin.readline()
                udp_s.sendto(str_data.encode("utf-8"), server_addr)
        ...

```

Exercício 14.7

Implemente um cliente de *chat* de forma a que possa dialogar com os outros utilizadores ligados ao mesmo servidor.

Exercício 14.8

Implemente o mesmo cliente e servidor mas utilizando TCP. Neste caso tenha em consideração que o servidor irá possuir um *Socket* por cliente, armazenando estes *Sockets* na lista de clientes (e não o endereço e porta utilizados pelo cliente).

14.6 Para Aprofundar

Exercício 14.9

Implemente um programa que dado um endereço HTTP como primeiro argumento, e um nome de ficheiro como segundo argumento, crie uma ligação TCP e envie os cabeçalhos HTTP de forma a obter o recurso (ficheiro) indicado. A resposta deverá ser escrita para o ficheiro indicado no segundo argumento. (Este programa é um cliente HTTP simples.)

Exercício 14.10

Implemente um programa que escute por mensagens no *Socket TCP* no formato:

```
GET /dir/fname
```

O programa deve responder com o conteúdo do ficheiro que se encontra no local indicado. Considere que os ficheiros pedidos encontram-se em diretórios e sub-diretórios de um diretório raiz específico. A título de exemplo, se o diretório raiz for **/tmp** e for pedido o ficheiro **/labi/aula/a.txt**, o programa deve fornecer o conteúdo do ficheiro **/tmp/labi/aula/a.txt**. (Este programa é um servidor HTTP simples.)

Documentos

Objetivos:

- Documentos
- Comma Separated Values
- JavaScript Object Notation
- Extensible Markup Language

15.1 Introdução

A informação é frequentemente transferida entre máquinas, ou fornecida a aplicações para processamento, e estas atividades são vitais para o modo como os sistemas atuais funcionam e como nós utilizamos os recursos informáticos. Existe no entanto a necessidade das aplicações entenderem o formato utilizado para a codificação da informação. Para isso foram criados vários formatos, de acordo com as necessidades de cada caso de utilização. Num tema anterior foi abordado o formato HTML que permite a representação de documentos Web. Também foi abordado o formato L^AT_EX que permite a edição e geração de documentos principalmente de carácter técnico. Certamente também já utilizou ferramentas que operam sobre ficheiros **xls**, **docx**, ou **odt**.

Este tema irá focar-se sobre a manipulação de documentos em três formatos muito comuns, usados para a troca de informação entre aplicações e dispositivos (p.ex, através de *sockets*). Todos utilizam representações textuais para a codificação da informação, o que tem a vantagem de permitir que os dados sejam interpretados ou até gerados por humanos.

15.2 CSV

O formato Comma Separated Values (CSV)[41] é muito comum para a troca de informação, em especial quando se trata de séries temporais de valores medidos em sensores ou equipamentos laboratoriais, por exemplo. Teve a sua origem nos primórdios da computação, sendo um formato que exige muito pouca capacidade de processamento. O seu nome deriva do uso de vírgulas para separar os diversos campos. Ele é estruturado em linhas de texto, terminadas por uma indicação de nova linha ("\\n"), e que cada linha contém um ou mais valores relacionados entre si. Por outras palavras, cada linha representa um registo de dados composto por vários campos de informação. O formato pode possuir um cabeçalho, indicando qual o nome de cada coluna e frequentemente todas as linhas possuem um número igual de colunas.

O exemplo seguinte contém valores da temperatura medida dentro de um frigorífico num período de alguns minutos.

```
id,time,timestamp,value
1,15/03/2014 18:07:24,1394903244.0,2.3
1,15/03/2014 18:08:24,1394903304.0,1.8
1,15/03/2014 18:09:24,1394903364.0,1.2
1,15/03/2014 18:10:24,1394903424.0,1.6
1,15/03/2014 18:11:24,1394903484.0,2.1
1,15/03/2014 18:12:24,1394903544.0,2.5
1,15/03/2014 18:13:24,1394903604.0,2.9
1,15/03/2014 18:14:24,1394903664.0,3.3
1,15/03/2014 18:15:24,1394903724.0,3.0
1,15/03/2014 18:16:24,1394903784.0,2.8
1,15/03/2014 18:17:24,1394903844.0,2.4
```

Cada linha inclui um identificador do dispositivo que reportou os dados, uma data e hora em formato textual, um tempo em formato absoluto,⁴⁹ e finalmente o valor da temperatura. Este exemplo também demonstra que nem sempre a vírgula é um bom separador. Na língua Portuguesa utiliza-se a vírgula como separador decimal, pelo que qualquer número com parte decimal iria resultar numa linha com mais uma coluna. Visto que o formato CSV não possui qualquer indicação de língua ou tabela de caracteres, a escolha do separador deve ser feita com cuidado. Uma solução passa por delimitar com aspas todos os campos que potencialmente possam ter o carácter separador no seu interior. Utilizam-se também variações do formato CSV, tal com o formato Tabulation Separated Value (TSV). Neste último, o carácter de tabulação é utilizado para separar os diversos campos. É muito comum encontrarem-se ficheiros denominados CSV em que são utilizados os caracteres ;, : ou |. Deste modo, deve-se considerar o formato CSV

⁴⁹Este formato é muito comum e descreve o número de segundos desde 1 de Janeiro de 1970

como uma família genérica de formatos e não apenas aqueles que utilizam vírgulas para separar valores.

Em *Python* e de uma forma geral na maioria das linguagens, o processamento deste tipo de ficheiros é simples e compatível com ferramentas existentes em qualquer sistema operativo. Por este motivo, o formato não morreu, encontrando ainda grande utilidade. Os ficheiros podem ser abertos usando o módulo **csv**, sendo que o módulo permite a iteração pelas linhas do ficheiro. Cada linha é apresentada como uma lista contendo um valor (da linha) em cada elemento da lista.

O exemplo seguinte abre um ficheiro CSV e imprime os seus valores.

```
import csv
import sys

def main(argv):
    fich_csv = open(argv[1], "r")
    csv_reader = csv.reader(fich_csv, delimiter=',')
    for row in csv_reader:
        print(row)

main(sys.argv)
```

Quando aplicado aos dados de temperatura o resultado deverá ser parecido com o seguinte.

```
['id', 'time', 'timestamp', 'value']
['1', '15/03/2014 18:07:24', '1394903244.0', '2.3']
['1', '15/03/2014 18:08:24', '1394903304.0', '1.8']
...
['1', '15/03/2014 18:17:24', '1394903844.0', '2.4']
```

Desta forma, é possível aceder aos valores através de índices da lista, como por exemplo **row[0]**. Tenha em consideração que é possível especificar o delimitador a utilizar. De notar ainda que o cabeçalho do ficheiro é tratado como uma qualquer linha. No entanto, se o ficheiro for interpretado através do método **csv.DictReader**, o resultado será um dicionário que utiliza o cabeçalho para chave dos valores. Com este método o resultado será:

```
{'timestamp': '1394903244.0', 'id': '1', 'value': '2.3', 'time': '15/03/2014 18:07:24'}
{'timestamp': '1394903304.0', 'id': '1', 'value': '1.8', 'time': '15/03/2014 18:08:24'}
...
{'timestamp': '1394903844.0', 'id': '1', 'value': '2.4', 'time': '15/03/2014 18:17:24'}
```

Exercício 15.1

Implemente um programa que leia os dados fornecidos e calcule o valor máximo, mínimo e médio da temperatura.

A criação de ficheiros CSV pode ser feita escrevendo os valores através da instrução `print`, mas o processo pode ser facilitado utilizando as estruturas e métodos adequados. Nomeadamente é possível criar uma lista com os valores e usar o módulo `csv` para a construção do ficheiro `csv`.

O exemplo seguinte cria um ficheiro chamado `rand.csv` com duas séries de valores: um valor incremental e um aleatório. De notar que o programa cria uma lista chamada `data` e depois essa lista é escrita para um ficheiro. O ficheiro também terá um cabeçalho com o nome das duas colunas.

```
import csv
import random

def main():
    fout = open('rand.csv', 'w')
    writer = csv.DictWriter(fout, fieldnames=['time', 'value'])

    writer.writeheader()

    for i in range(1,10):
        writer.writerow({'time': i, 'value' : random.randint(1,100)})

    fout.close()

main()
```

Exercício 15.2

Replique o exercício anterior e verifique o ficheiro criado. Modifique o delimitador especificando-o no construtor do objecto `DictWriter`.

A documentação desta classe pode ser consultada em <http://docs.python.org/3/library/csv.html#csv.DictWriter>.

Exercício 15.3

Considere o módulo `time` que permite acesso à hora atual e o módulo `psutil` que permite aceder a estatísticas do sistema,^a em particular os seguintes métodos:

`time.time()`: Devolve o número de segundos desde o início da época (1 Janeiro 1970).

`psutil.cpu_percent(interval=x)`: Verifica qual a utilização do processador durante o intervalo especificado.

`psutil.net_io_counters()`: Verifica quantos pacotes/octetos foram enviados/recebidos por cada interface de rede. O resultado é um dicionário de tuplos. Por exemplo, o número de octetos enviados pela interface pode ser obtido acedendo à primeira posição, accedendo-se como a uma lista.

Construa um programa que registe o tempo em segundos, o número de octetos enviados e recebidos e a percentagem de ocupação do processador. O programa deve executar durante 60s, capturando valores a cada segundo. Execute o programa implementado, navegue por algumas páginas e verifique o resultado.

^aPode ser necessário instalar os módulos usando `pip3 install nome-do-modulo`.

15.3 JSON

Um outro formato bastante comum, especialmente no âmbito de aplicações Web é o formato JavaScript Object Notation (JSON)[42]. Sendo mais rico do que o formato CSV, mas mais compacto e menos rígido do que o formato XML (ver abaixo), é o formato utilizado em grande parte das transações de informação entre aplicações Web. A razão para isto reside no facto de ser um formato que é nativo para a linguagem *JavaScript* e muito bem suportado em muitas outras, como a linguagem *Python*.

O formato JSON é constituído pela descrição textual de pares chave-valor, sendo que cada valor pode ser uma *String*, um número, um *Array*, um valor booleano ou um outro objeto. Um exemplo de um documento JSON seria:

```
{  
    'time' : 1394984189,  
    'name' : 'cpu',  
    'value': 12  
}
```

Este documento é composto por um objeto com três chaves, duas possuindo um valor

inteiro e outra possuindo um valor *String*. Repare como a estrutura é semelhante à de um dicionário na linguagem *Python*. De facto é possível converter qualquer estrutura de dicionário ou lista para o formato JSON e vice-versa, o que é extremamente útil para transmitir informação sobre *Sockets*.

O documento anterior pode ser gerado na linguagem *Python* através da criação e posterior conversão de um dicionário. O exemplo seguinte demonstra como uma lista de valores poderia ser convertida para o formato JSON.

```
import json

def main():
    data = [
        {'time': 1394984189, 'name': 'cpu', 'value': 12},
        {'time': 1394984190, 'name': 'cpu', 'value': 19}
    ]
    print(json.dumps(data, indent=4))

main()
```

Exercício 15.4

Verifique o resultado do exemplo anterior e altere-o de forma à variável **data** conter várias listas e dicionários dentro da lista principal.

A leitura de documentos do tipo JSON pode ser realizada através do método **json.loads**, que recebe uma *String* em formato JSON e devolve uma lista ou dicionário.

Exercício 15.5

Altere o Exercício 3 de forma a que o seu resultado seja um ficheiro no formato JSON com o seguinte conteúdo:

```
{
    'stats' : [
        {'time': timestamp, 'cpu': value, 'network': value},
    ]
}
```

15.4 XML

Outro formato bastante popular nos sistemas informáticos é o Extensible Markup Language (XML). Tal como o nome indica (*ML = Markup Language*), o formato XML é uma linguagem em si, o que significa que é bastante mais completo do que o formato CSV. O formato XML partilha muitas características com o HTML, mas tem em âmbito de aplicação muito mais variado e uma sintaxe mais rígida e uniforme.

Um documento XML é um texto que inclui *marcas* e *conteúdo*. As marcas seguem o padrão `<texto-da-marca>`, enquanto o conteúdo encontra-se entre marcas. As marcas usam-se para organizar o conteúdo do documento como um conjunto de *elementos* estruturais. Cada elemento é indicado por uma *marca de início*, algum *conteúdo* e termina com uma *marca de fim* correspondente. O conteúdo de um elemento pode incluir outros elementos, formando uma estrutura hierárquica. As marcas de início e fim têm o formato `<tipo-de-marca>` e `</tipo-de-marca>`, repetivamente. Também há elementos vazios, sem conteúdo, indicados por um marca com o formato `<tipo-vazio/>`. Além do tipo, o texto da marca também pode incluir atributos. O exemplo abaixo mostra um elemento XML de tipo **foo** cujo conteúdo inclui três elementos vazios de tipo **bar**. Todos os elementos têm atributos definidos.

```
<foo attrib1='x' attrib2='y'>
  <bar attrib1='z1' />
  <bar attrib1='z2' />
  <bar attrib1='z3' />
</foo>
```

Repare que este formato é em tudo semelhante ao do formato HTML, mas no XML as marcas utilizadas não estão restringidas ao necessário para construir páginas. Pelo contrário, podem codificar uma grande variedade de conteúdos. Por exemplo, os formatos **docx** e **odf** utilizados pelas aplicações *Microsoft Office* e *LibreOffice* são baseados em XML. É um formato muito popular para codificar informação de documentos, para ficheiros de configuração e mesmo para transmitir séries de dados.

Um ficheiro contendo XML deve ser iniciado por um cabeçalho que indica algumas características do ficheiro nomeadamente: a codificação utilizada e por vezes o *Schema*. A codificação é importante para identificar corretamente os caracteres utilizados, enquanto o *Schema* define que marcas podem ser encontradas no documento, o seu tipo e como se relacionam entre si. Um *Schema* é uma forma poderosa de definir a sintaxe específica de certo tipo de documento baseado em XML. É geralmente definido num documento separado, que também pode estar em formato XML.

Os documentos XML possuem uma estrutura hierárquica, o que significa que existe um elemento raiz e vários sub-elementos. Relembre o caso do HTML em que o elemento

`<html>` é a raíz de qualquer documento deste tipo.

Leitura de Ficheiros XML

De uma forma ad-hoc, o formato XML é vulgarmente utilizado para armazenar configurações, ou para construir documentos que sejam interoperáveis entre várias plataformas, particularmente quando a informação a armazenar ou a enviar tem uma estrutura complexa.

Considere o exemplo seguinte, que poderia ser um ficheiro de configuração para o programa do Exercício 3. Em particular, este ficheiro define o intervalo de atualização, o formato de saída e quais os valores que o programa deve monitorizar.

```
<?xml version="1.0" encoding="utf-8"?>
<conf>
    <interval>1</interval>
    <output>
        <csv active="true" separator="," />
        <xml active="false" />
    </output>
    <monitor>
        <cpu active="true" />
        <network active="true" />
        <ram active="false" />
    </monitor>
</conf>
```

Para ler este ficheiro no programa podemos usar funções fornecidas no módulo `lxml.etree`, que permitem aceder ao conteúdo do ficheiro na forma de uma árvore. O código seguinte processa o ficheiro `conf.xml` e dá acesso aos seus elementos. Cada elemento da árvore possui um nome de marca (`tag`), atributos (`attrib`) e conteúdo (`text`).

```
from lxml import etree

def main():
    xml = etree.parse('conf.xml')
    root = xml.getroot()
    print(root.tag)
    for child in root:
        print(child.tag, child.attrib, child.text)

main()
```

O exemplo deverá imprimir a raiz do documento (**conf**) e todos os elementos contidos dentro da raíz, também chamados os elementos filhos.

Exercício 15.6

Altere o exemplo anterior de forma a imprimir todos os atributos e valores presentes no ficheiro **conf.xml**. Note que cada elemento filho poderá conter também outros filhos, que é preciso mostrar recursivamente.

Também é possível procurar entradas num ficheiro XML de forma a obterem-se rapidamente os valores pretendidos. Neste caso, isto seria interessante para que o programa pudesse obter os valores das configurações que irão condicionar a sua operação. Para isto utiliza-se o método **findall**, como exemplificado no excerto seguinte.

```
...
monitor_cpu = root.findall('./monitor/cpu')
monitor_ram = root.findall('./monitor/ram')

print(monitor_cpu[0].attrib['active'])
print(monitor_ram[0].attrib['active'])
```

O método **findall** devolve uma lista com todos os elementos encontrados. Se não for encontrado nenhum elemento, devolve uma lista vazia. De seguida é possível aceder ao atributo **active** de forma a saber se se deve monitorizar a ocupação de CPU e a utilização de RAM.

Exercício 15.7

Altere o programa criado no Exercício 3 de forma a que ele tenha em consideração o ficheiro **conf.xml**. Para obter a informação da memória, use o método **psutil.virtual_memory()**, que indica a memória disponível no segundo elemento do tuplo devolvido (ver <https://github.com/giampaolo/psutil>).

Escrita de valores

Os documentos XML, sendo baseados num formato textual, são facilmente editáveis por humanos. No entanto, estes documentos são também utilizados para a comunicação entre sistemas, pois a codificação textual também uniformiza o processamento do documento

numa multiplicidade de sistemas, evitando complicações decorrentes de detalhes de codificações binárias como a *endianness* da representação.

Voltando ao exemplo anterior, gostaríamos que o programa pudesse escrever o seu resultado como um ficheiro XML em vez de CSV. Esta preferência poderá ser indicada através do ficheiro de configuração **conf.xml**. O processo de escrita de XML inclui 3 fases: 1) criação da raiz do documento e elementos principais, 2) adição de valores ao documento, 3) escrita do documento para um ficheiro de texto.

O exemplo seguinte cria uma raiz para utilizar num documento XML, cria depois um sub-elemento, adiciona-o à raiz e escreve o resultado para o ecrã.

```
from lxml import etree
import time

def main():
    root = etree.Element("stats")

    for i in range(1,10):
        value = etree.SubElement(root, 'value')
        value.set('time', str(int(time.time())))
        value.text = str(i)
        time.sleep(1)

    print(etree.tostring(root, xml_declaration=True, encoding="utf-8", pretty_print=True).decode('utf-8'))
main()
```

Neste caso a raiz terá um sub-elemento chamado **value** com um atributo chamado **time** que contém a hora atual. Depois de impresso, o resultado será o seguinte:

```
<?xml version='1.0' encoding='UTF-8'?>
<stats>
  <value time="1394984189"> 0 </value>
  ...
</stats>
```

Exercício 15.8

Implemente o exemplo anterior e verifique o resultado obtido.

Comparando com o formato CSV, pode-se verificar que o XML é muito menos compacto, ocupando muito mais espaço de armazenamento para conter a mesma informação. No

entanto, a informação está mais estruturada e claramente identificada.

Se fosse necessário adicionar mais elementos ao elemento **value**, seguir-se-ia a mesma metodologia de criar um sub-elemento, tal como demonstrado no exemplo seguinte.

```
...
    value = etree.SubElement(root, 'value')
    value.set('time', str(int(time.time())))

    cpu = etree.SubElement(value, 'cpu')
    cpu.text = 10
...

```

Exercício 15.9

Considere o Exercício 3 e escreva o resultado para um ficheiro XML no formato indicado de seguida.

```
<stats>
    <value time="timestamp-atual">
        <cpu> valor </cpu>
        <ram> valor <ram>
        <network> valor </network>
    </value>
</stats>
```

Validação de documentos

Um aspecto importante no processamento de documentos, independentemente do formato utilizado, é a validação da sua estrutura e conteúdo. No caso de XML, um erro pode ocorrer devido a caracteres inválidos ou em falta (ex, falta de um carácter */*), mas também pode ser devido à colocação de marcas numa posição incorreta, porque alguns elementos só admitem conter elementos de certos tipos. Outra situação importante é a validação das marcas utilizadas no documento. Por exemplo, em HTML, a marca **h1** existe mas a marca **hh1** não faz qualquer sentido, sendo assim considerada como errada.

O exemplo seguinte considera uma pequena *Playlist* de música no formato XML Shareable Playlist Format (XSPF). Este formato, baseado em XML, é utilizado para armazenar e partilhar *Playlists* com músicas.

```
<?xml version="1.0" encoding="utf-8"?>
<playlist version="1" xmlns="http://xspf.org/ns/0/">
```

```

<title>My playlist</title>
<creator>Jane Doe</creator>
<annotation>My favorite songs</annotation>
<info>http://example.com/myplaylists</info>

<trackList>
  <track>
    <location>http://example.com/my.mp3</location>
    <title>My Way</title>
    <creator>Frank Sinatra</creator>
    <annotation>This is my theme song.</annotation>
    <album>Frank Sinatra's Greatest Hits</album>
    <trackNum>3</trackNum>
    <duration>19200</duration>
  </track>
</trackList>
</playlist>

```

Neste exemplo, a primeira linha identifica o ficheiro como sendo XML usando uma codificação **UTF-8**. A segunda linha identifica qual o elemento raíz do documento. Tendo em consideração um dado documento e o seu *Schema* é possível validar conteúdo e estrutura, detectando erros no documento. Pode reparar que com a excepção da marca **title**, nenhuma outra está presente num documento HTML, sendo que a estrutura é bastante semelhante.

O exemplo seguinte demonstra como é possível validar a *Playlist* anterior. O ficheiro **playlist.xspf** contém o exemplo demonstrado anteriormente, enquanto o ficheiro **xspf-draft8.rng** pode ser encontrado no endereço <http://www.xspf.org/schema/> e contém o *Schema* relativo ao formato XSPF.

```

from lxml import etree

def main():
    doc = etree.parse('playlist.xspf')

    schema = etree.parse('xspf-draft8.rng')
    validator = etree.RelaxNG(schema)

    print(validator.validate(doc))
    print(validator.error_log.last_error)

main()

```

Neste exemplo, a primeira impressão irá apresentar o resultado da validação, enquanto a segunda impressão irá apresentar o erro encontrado (se algum). Ao acto de se ler um formato estruturado dá-se o nome de *Parsing* e como pode deduzir, este método de

validação permite que aplicação (ex, o *LibreOffice*, ou um navegador) verifique se um dado documento foi construído corretamente ou possui erros.

Exercício 15.10

Replique o exemplo anterior e verifique se o documento apresentado é XML válido para uma *Playlist*.

Introduza uma qualquer pequena modificação ao ficheiro e volte a validar o documento.

Exercício 15.11

Altere o programa anterior de forma a escrever o resultado em XML.

15.5 Para Aprofundar

Exercício 15.12

Considere o ficheiro disponível em <http://api.openweathermap.org/data/2.5/weather?q=NOME-DA-CIDADE&mode=xml>, contendo os dados meteorológicos observados na cidade indicada. Obtenha-o para o seu computador e construa um programa que aceite um argumento com o nome da cidade e imprima os dados observados.

Exercício 15.13

Considere o ficheiro disponível em <http://services.web.ua.pt/parques/parques>, contendo os dados relativos à capacidade dos parques da Universidade de Aveiro. Implemente um programa que apresente a disponibilidade do parque mais próximo do utilizador. Considere que a localização é fornecida como argumento, no formato latitude e longitude. Para calcular a distância entre duas coordenadas, recorra à fórmula da distância de Haversine.

Exercício 15.14

Considere um outro serviço disponível no endereço <http://api.web.ua.pt> e apresente os dados fornecidos. Neste site pode encontrar serviços que fornecem informação tal como as ementas ou as senhas da secretaria.

Pode deixar um programa a monitorizar com frequência um dado serviço. Por exemplo, será que a ementa depende do estado do tempo? E qual a hora da manhã a partir da qual os parques enchem?

Aplicações e Serviços Web

Objetivos:

- Servidores Web
- Serviços Web

16.1 Introdução

A World Wide Web (WWW) ou *Web* como é hoje popularmente conhecida, teve a sua géneze em 1990 no Conseil Européen pour la Recherche Nucléaire (CERN) pelas mãos de Tim Berners-Lee. Inicialmente, a *Web* pretendia ser um sistema de hiper-texto que permitisse aos cientistas seguir rapidamente as referências num documento, evitando o processo tedioso de apontar e pesquisar referências. A *Web* pretendia na altura ser um repositório de informação estruturado em torno de um grafo (daí a *Web*), em que o utilizador pudesse seguir qualquer percurso entre os diversos documentos interligados pelas suas referências.

A *Web* assenta em 3 tecnologias, já tratadas nesta disciplina:

- Um sistema global de identificadores únicos/uniformes (URL, URI).
- Uma linguagem de representação de informação (HTML).
- Um protocolo de comunicação cliente/servidor (HTTP).

Com base nestas tecnologias, podemos não só transferir ficheiros estáticos entre um servidor e um cliente equipado com um *Web browser*, como também podemos construir

documentos de forma dinâmica a partir de dados recebidos ou disponíveis no servidor num determinado momento. Um bom exemplo deste último caso são os serviços meteorológicos que processam dados de observação e produzem documentos JSON e XML com as observações e previsões, para consumo por humanos ou outras máquinas. Este guia irá guiar o desenvolvimento de uma aplicação Web dinâmica com base na linguagem de programação *Python* e no formato de documentos JSON.

16.2 Servidores Web

Common Gateway Interface

Um servidor Web é uma aplicação de software que permite a comunicação entre dois equipamentos através do protocolo HTTP. A função inicial de um servidor Web era a de fornecer documentos armazenados em disco em formato HTML a um cliente remoto equipado com um Web browser. Esta simples tarefa desde cedo demonstrou-se demasiado restritiva, uma vez que frequentemente era necessário condicionar os dados nos documentos HTML a vários fatores, tais como: a identidade do utilizador, a sua localização, a sua língua nativa, etc.

É desta forma que surge o conceito de Common Gateway Interface (CGI). A CGI permite ao servidor interagir com um programa externo capaz de produzir dinamicamente conteúdos de qualquer formato. O standard CGI define um conjunto de parâmetros que são passados do servidor Web para a aplicação externa (denominada script CGI), assim como o formato que essa mesma aplicação deve de obedecer por forma ao servidor Web re-interpretar o seu *output* antes de enviar ao Web browser do cliente.

É possível criar programas CGI em qualquer linguagem, inclusive uma linguagem de

scripting como a *Bash*.

Exercício 16.1

No servidor `xcoa.av.it.pt`, no diretório `public_html`, crie um novo diretório com o nome `cgi-bin`. Dentro desse diretório crie um novo ficheiro `test.sh` com o seguinte conteúdo:

```
#!/bin/bash
echo "Content-type: text/plain"
echo ""
echo "Hello World"
```

Dê permissões de execução ao ficheiro (`chmod +x test.sh`).

Execute-o na linha de comando (`./test.sh`).

Agora no seu navegador Web, acceda ao ficheiro que acabou de criar.

Altere o ficheiro para mostrar outras *Strings*.

Do exercício anterior é importante reter a necessidade do programa imprimir um cabeçalho com informação do tipo de ficheiro que será criado dinamicamente. Através da interface CGI é possível não só criar ficheiros de texto (*plain*, HTML, JS, etc) como também ficheiros binários (imagens, vídeos, etc).

Exercício 16.2

Altere o ficheiro anterior adicionando o comando `env`.

```
#!/bin/bash
echo Content-type: text/plain
echo ""
echo "Hello World"
env
```

Aceda ao ficheiro no seu navegador Web.

O resultado deste exercício mostra as *variáveis de ambiente* que o servidor Web envia para o programa externo através da interface CGI.⁵⁰

⁵⁰ As variáveis de ambiente (*environment variables*) são um mecanismo providenciado pelo sistema operativo para disponibilizar informação aos programas, para além do mecanismo de passagem de

Servidores Aplicacionais

Na secção anterior abordámos a interface CGI que se popularizou nos finais do século passado como a ferramenta para desenvolver conteúdos dinâmicos para a Web. Servidores como o *Apache* e o *Microsoft IIS* permitem a execução de programas externos usando a interface CGI e ainda hoje diversos sites Web fazem uso desta tecnologia. No entanto, o crescente dinamismo da Web e a necessidade de criação de aplicações Web que requerem múltiplas interações com o utilizador tornam a interface CGI extremamente ineficiente e até mesmo insegura (já que os programas correm com as mesmas permissões do servidor Web).

Servidores aplicacionais como o *Glassfish*, *JBoss*, *.NET* permitem ao programador ultrapassar muitas destas dificuldades ao incorporarem em si próprios código desenvolvido por programadores externos. Não estamos mais na situação de o servidor executar um programa externo, mas na de o próprio programa incluir o servidor Web.

Neste capítulo, vamos abordar um servidor aplicacional específico para *Python*. O *CherryPy* é um servidor aplicacional maduro usado tanto para pequenas aplicações como para grandes (ex.: *Hulu*, *Netflix*). O *CherryPy* pode ser usado sozinho (*stand-alone*) ou através de um servidor Web tradicional via interfaces Web Server Gateway Interface (WSGI). Nesta disciplina, vamos usar o *CherryPy* apenas como servidor *stand-alone*.

Para instalar o *CherryPy* pode recorrer ao gestor de pacotes da sua distribuição Linux ou ao **pip**.

No ubuntu pode executar:

```
sudo apt-get install python-cherrypy3
```

Em alternativa pode executar:

```
sudo pip install CherryPy
```

ou:

```
sudo pip install --upgrade CherryPy
```

É altamente aconselhado que se instale o *CherryPy* via o comando **pip pois a versão é mais recente.**

argumentos.

O *CherryPy* é composto por 8 módulos:

CherryPy.engine Controla início e o fim dos processos assim como o processamento de eventos.

CherryPy.server Configura e controla a WSGI ou servidor HTTP.

CherryPy.tools Conjunto de ferramentas ortogonais para processamento de um pedido HTTP.

CherryPy.dispatch Conjunto de *dispatchers* que permitem controlar o encaminhamento de pedidos para os *handlers*.

CherryPy.config Determina o comportamento da aplicação.

CherryPy.tree A árvore de objetos percorrida pela maioria dos *dispatchers*.

CherryPy.request O objeto que representa o pedido HTTP.

CherryPy.response O objeto que representa a resposta HTTP.

Comecemos por criar uma aplicação semelhante ao *script CGI* anterior.

Exercício 16.3

Crie no seu próprio computador o seguinte ficheiro.

```
import cherrypy

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        return "Hello World!"

cherrypy.tree.mount(HelloWorld(), "/")
cherrypy.server.start()
```

Se possuir a última versão do *CherryPy* as duas linhas finais do ficheiro anterior podem ser substituídas por:

```
cherrypy.quickstart(HelloWorld())
```

Sendo que neste caso a aplicação é lançada automaticamente quando existirem alterações ao ficheiro e permite que seja terminada usando **CTRL-C**.

No seu Web browser aceda à aplicação usando o endereço <http://localhost:8080/>.

Revendo cada linha do exercício anterior, começamos por identificar a importação do módulo *CherryPy*. De seguida temos a declaração de uma classe *HelloWorld*. Esta classe é composta por um método chamado *index* que devolve uma *String*. O *decorador* `@cherrypy.expose` determina que o método *index* deverá ser exposto ao cliente Web. Por fim, o módulo *CherryPy* cria um objeto da classe *HelloWorld* e inicia um servidor com ele.

Quando um cliente Web acede ao servidor aplicacional *CherryPy*, este procura por um objeto e método que possa atender ao pedido do cliente. Neste exemplo básico, existe apenas um objeto e método que irá servir ao cliente a *String* "Hello World".

O *CherryPy* disponibiliza através do `CherryPy.request.headers` as variáveis enviadas pelo cliente ao servidor.

Exercício 16.4

Altere o programa anterior para mostrar o nome do servidor ao qual o cliente fez um pedido HTTP

```
...  
host = cherrypy.request.headers["Host"]  
return "You have successfully reached " + host
```

Qualquer objeto associado ao objeto raiz é acessível através do sistema interno de mapeamento *URL*-para-objeto. No entanto, tal não significa que um objeto esteja exposto na Web. É necessário que o objeto seja exposto explicitamente como visto

anteriormente.

Exercício 16.5

Crie um novo programa com o seguinte conteúdo

```
import cherrypy

class Node(object):
    @cherrypy.expose
    def index(self):
        return "Eu sou um objecto Folha"

class Root(object):
    def __init__(self):
        self.node = Node()

    @cherrypy.expose
    def index(self):
        return "Eu sou o objecto Raiz"

    @cherrypy.expose
    def page(self):
        return "Eu sou um método da Raiz"

if __name__ == "__main__":
    cherrypy.tree.mount(Root(), "/")
    cherrypy.server.start()
```

Aceda a cada um dos recursos a partir do seu navegador Web (`/page`, `/node/`).

Exercício 16.6

Acrescente agora uma nova classe `HTMLDocument` que devolva o conteúdo de um ficheiro HTML lido do disco.

Mais uma vez, é importante reter alguns aspetos do exercício anterior. O método `index` serve os conteúdos na raiz do URL `(/)` e cada método tem que ser exposto individualmente.

Formulário HTML

O protocolo HTTP define dois métodos principais para a troca de informação entre cliente e servidor: os métodos `GET` e `POST`. O método `GET` já foi extensivamente usado

nos capítulos e secções anteriores, e permite ao cliente Web solicitar um documento que resida no servidor Web. Por sua vez o método **POST** permite enviar informação do cliente Web para o servidor Web. É geralmente usado para enviar ao servidor um ficheiro ou um formulário HTML preenchido.

Exercício 16.7

Crie uma página HTML com o código para formulário seguinte:

```
<form action="actions/doLogin" method="post">
<p>Username</p>
<input type="text" name="username" value="" size="15" maxlength="40"/>
<p>Password</p>
<input type="password" name="password" value="" size="10" maxlength="40"/>
<p><input type="submit" value="Login"/></p>
<p><input type="reset" value="Clear"/></p>
</form>
```

Não esquecer de completar a página com o código HTML apropriado.
Crie um novo método na sua aplicação:

```
@cherrypy.expose
def form(self):
    cherrypy.response.headers["Content-Type"] = "text/html"
    return open("formulario.html","r").read()
```

No exercício anterior permitimos ao nosso servidor aplicacional servir uma página HTML com o conteúdo de um formulário usando o método **serve_file**. No entanto, a submissão

do formulário de *login* necessita ainda da implementação de mais um método.

Exercício 16.8

Crie um novo objeto (*actions*) e método na sua aplicação. Não se esqueça de associar o novo objeto à Raiz.

```
class Actions(object):
    @cherrypy.expose
    def doLogin(self, username=None, password=None):
        return "TODO: verificar as credenciais do utilizador " + username
```

Abra o formulário através do endereço `http://localhost:8080/form/`, preencha-o e submeta.

Importa referir que os argumentos *username* e *password* chegam até à nossa aplicação Web através de um mapeamento direto do nome das variáveis do formulário HTML para os argumentos do nosso método `doLogin` (também mapeado diretamente).

16.3 Serviços Web

Na secção anterior vimos como um cliente Web pode interagir com uma aplicação Web alojada no servidor. Nesta secção vamos ver como duas aplicações podem interagir entre si através do protocolo HTTP.

O primeiro desafio que se coloca é como escrever uma aplicação Python capaz de aceder a uma página Web via o protocolo HTTP. Para tal vamos fazer uso da biblioteca `urllib2`, cuja documentação completa encontra-se disponível em <http://docs.python.org/2/library/urllib2.html>.

A biblioteca `urllib2` permite-nos aceder a uma página Web de forma muito semelhante à que utilizamos em Python para aceder a um ficheiro.

```
import urllib2

f = urllib2.urlopen("http://www.python.org")
```

Exercício 16.9

Faça um pedido **GET** ao endereço `http://www.ua.pt`.

A sua aplicação deverá ler por completo o conteúdo da página da Universidade de Aveiro.

O uso directo do método `urlopen` permite-nos obter o conteúdo de um recurso HTTP através do método **GET**. No entanto, se pretendermos enviar algum conteúdo para uma aplicação Web, é necessário usar o método **POST** como vimos anteriormente.

O método **POST** possibilita o envio de informação codificada no corpo do pedido **POST**. A codificação dos dados segue um de dois *standards* definidos pelo World Wide Web Consortium (W3C), o `application/x-www-form-urlencoded` e o `multipart/form-data`. O primeiro formato é o usado por omissão e permite o envio de informação trivial como variáveis não muito extensas. O segundo é apropriado para o envio de variáveis mais extensas assim como de ficheiros.

O *Python* possui na biblioteca `urllib` o método `urlencode` que permite converter um dicionário *Python* numa *String* codificada em `application/x-www-form-urlencoded`.

```
import urllib

data = urllib.urlencode({"nome": "Ana", "idade": 20})
```

Munidos dessa *String* codificada podemos construir um objecto `Request` que é usado como argumento de `urlopen`.

```
req = urllib2.Request(url)
req.add_data(data)

f = urllib2.urlopen(req)
```

Exercício 16.10

Fazendo uso da aplicação Web desenvolvida anteriormente implemente uma aplicação capaz de fazer login.

Os exercícios anteriores demonstraram como criar uma aplicação Web capaz de interagir com um cliente (*Web browser*), mas a sua utilidade pode ser transposta para a comunicação

entre duas aplicações.

Exercício 16.11

O Google dispõe de uma Application Programming Interface (API) que permite converter um endereço em coordenadas (latitude e longitude). Neste exercício deverá usar a API do google com base no seguinte código para encontrar as coordenadas de qualquer cidade passada ao seu programa através de um argumento de linha de comando.

```
serviceurl = "http://maps.googleapis.com/maps/api/geocode/json?"  
  
url = serviceurl + urllib.urlencode({"sensor":"false", "address": address})  
f = urllib.urlopen(url)
```

16.4 Para Aprofundar

Exercício 16.12

Recupere o exercício de aprofundamento do guião anterior.

Construa uma aplicação Web que acceda ao ficheiro disponível em http://www.ipma.pt/resources/www/internal.user/pw_hh_pt.xml, contendo os dados meteorológicos observados nas principais cidades portuguesas.

A sua aplicação deverá receber o nome da cidade por método POST, pelo que deve construir um formulário com a lista de cidades possíveis usando uma dropdown.

À submissão do formulário deverá seguir-se a impressão dos dados da cidade indicada no formulário.

Bases de Dados

Objetivos:

- Bases de dados relacionais
- SQL
- Acesso programático a bases de dados

17.1 Bases de Dados

As bases de dados são um componente fundamental em muitas aplicações, pois agem como repositórios onde é possível armazenar e consultar informação. É certo que isto pode ser feito utilizando ficheiros comuns, por exemplo é perfeitamente possível armazenar uma listagem de clientes num ficheiro de texto que depois é lido pela aplicação. No entanto,

- se a quantidade de dados é grande;
- se se pretende consultar os dados de forma não sequencial;
- se se pretende pesquisar ou ordenar dados segundo múltiplos critérios;
- se se pretende cruzar dados de várias naturezas;
- se ocasionalmente se acrescentam dados;
- se várias das anteriores são verdade;

então uma base de dados é uma solução muito mais poderosa e eficiente do que um ficheiro tradicional. As bases de dados, em particular as relacionais, possuem características de

manipulação de informação muito interessantes, que facilitam a sua utilização face a outros métodos.

Considere uma lista de contactos de clientes em que é necessário armazenar o nome, endereço de correio electrónico e contacto telefónico. Um ficheiro simples poderia armazenar esta informação, por exemplo utilizando o formato CSV.

João, Manuel, Fonseca, jmf@gmail.com, 912345654
Pedro, Albuquerque, Silva, pedro23@gmail.com, 932454349
Maria, Carreira, Dinis, mariadi@ua.pt, 234958673
Catarina, Alexandra, Rodrigo, calexro@sapo.pt, 963343386

Se precisarmos de procurar um contacto de um cliente específico, temos de percorrer o ficheiro até o encontrar. Se precisarmos de alterar os dados de algum cliente, temos de reescrever o resto do ficheiro. Considerando que uma base de dados de clientes (ou outra) pode conter milhares ou milhões de entradas, usar um simples ficheiro de texto rapidamente se torna um problema.

Uma *base de dados relacional* organiza os dados na forma de *tabelas*. Cada tabela contém várias linhas, cada uma contendo um *registo* (ou *tuplo*) de dados de uma entidade de um certo tipo, e cada linha contém várias colunas, correspondendo aos diferentes *atributos* (ou *campos*) dessa entidade. Por exemplo, os dados de contactos descritos acima poderiam ser guardados numa base de dados relacional na forma de uma tabela com 4 registos com 5 atributos cada, como representado abaixo.

Table: contacts

firstname (TEXT)	middlename (TEXT)	lastname (TEXT)	email (TEXT)	phone (TEXT)
João	Manuel	Fonseca	jmf@gmail.com	912345654
Pedro	Albuquerque	Silva	pedro23@gmail.com	932454349
Maria	Carreira	Dinis	mariadi@ua.pt	234958673
Catarina	Alexandra	Rodrigo	calexro@sapo.pt	963343386

Na base de dados, a tabela é caracterizada por um nome e pelo identificador e tipo de dados de cada atributo.

Os *sistemas de gestão de bases de dados*(SGBD) são programas que permitem fazer pesquisas e alterações de registos numa base de dados, segundo critérios complexos, de forma versátil e eficiente. Existem diversos sistemas de gestão de bases de dados relacionais, mas todos suportam interação com programas clientes através de uma linguagem comum. Em particular, todos estes sistemas suportam a Structured Query Language (SQL), que permite efectuar operações muito detalhadas sobre a estrutura da

informação ou os seus dados. Nos próximos exercícios veremos como a linguagem SQL pode ser utilizada para manipular bases de dados relacionais.

Para criar uma base de dados vamos usar o **sqlite3**, que é um SGBD simples que permite criar bases de dados num sistema de ficheiros tradicional, usando o formato *SQLite*. Esta base de dados é muito usada e encontra-se por exemplo nas plataformas Android e iOS. Em alternativa, poderíamos utilizar um servidor de bases de dados, como se faz tipicamente em ambientes de produção, mas do ponto de vista de utilização as diferenças são mínimas.

Para criar uma base de dados é necessário executar a ferramenta **sqlite3** com a seguinte sintaxe:⁵¹

```
sqlite3 database.db
```

Deve aparecer novo *Prompt* que nos permite invocar comandos sobre a base de dados. Um exemplo de um comando é a criação de uma tabela, que se faz através da linguagem SQL:

```
CREATE TABLE tablename(
    field1 TYPE1,
    field2 TYPE2
);
```

Em que **tablename** representa o nome da tabela, **field1** representa o nome da primeira coluna e **TYPE1** o seu tipo de dados. São suportados vários tipos de dados. Os mais relevantes para este trabalho são:

TEXT: Texto.

INTEGER: Números inteiros.

REAL: Números reais.

BLOB: Um qualquer conteúdo (ex, uma imagem).

⁵¹Esta ferramenta deve existir nos repositórios da maioria dos sistemas.

Exercício 17.1

Crie uma tabela chamada **contacts** contendo os campos identificados anteriormente. Para facilitar a criação da base de dados, crie os comandos num ficheiro de texto e depois copie e cole o texto para o *Prompt* da ferramenta.

Pode verificar o conteúdo da tabela se executar o comando **.tables**. Pode sair desta ferramenta executando o comando **.quit**. (Estes comandos são específicos do **sqlite**. Não são SQL.)

Depois de criada a tabela, é necessário inserir dados. Para isso, usa-se o comando **INSERT INTO**.

```
INSERT INTO tablename  
VALUES (  
    "value1", "value2"  
)
```

A directiva **VALUES** indica os valores a colocar nas colunas existentes. Valores do tipo **TEXT** devem possuir aspas. É possível inserir informação em apenas alguns campos, bastando para isso que se especifiquem quais os campos de destino da informação.

```
INSERT INTO tablename(field1)  
VALUES (  
    "value1"  
)
```

Exercício 17.2

Construa os comandos necessários para inserir os dados de todos os clientes. Mais uma vez, crie os comandos num ficheiro de texto e depois copie-os para o **sqlite3**.

De forma consultar os dados inseridos em tabelas utiliza-se o comando **SELECT**. Com este comando é possível discriminar de forma muito detalhada que informação deve ser obtida e em que formato. Na sua forma básica o comando **SELECT** pode obter toda a informação de uma tabela:

```
SELECT * FROM contacts;
```

O que deverá produzir o seguinte resultado:

```
João|Manuel|Fonseca|jmf@gmail.com|912345654
Pedro|Albuquerque|Silva|pedro23@gmail.com|932454349
Maria|Carreira|Dinis|mariadi@ua.pt|234958673
Catarina|Alexandra|Rodrigo|calexro@sapo.pt|963343386
```

Também é possível obter apenas algumas colunas, e/ou apenas algumas linhas. Por exemplo, poderemos obter apenas o email e número de telefone dos contactos com nome Pedro:

```
SELECT email, phone FROM contacts WHERE firstname="Pedro";
```

Exercício 17.3

Construa vários comandos que permitam obter informação específica sobre os utilizadores e teste-os.

Exercício 17.4

Adicione a directiva **ORDER BY columnname ASC** aos seus comandos e compare o resultado. Em vez de **ASC**, também pode utilizar **DESC**

Por vezes é necessário actualizar informação, nomeadamente mudar o valor de células específicas ou mesmo apagar linhas inteiras. O comando **UPDATE** permite actualizar uma ou mais células. Por exemplo, podemos mudar o número de telefone do João Fonseca através do comando:

```
UPDATE contacts SET phone = 912345653 WHERE email="jmf@gmail.com";
```

Tem de se ter em consideração que o comando **UPDATE** altera todas as linhas, excepto se for especificada uma regra que restrinja o número de linhas a considerar. Neste caso isto é feito usando **WHERE email="jmf@gmail.com"**. Sem esta especificação o comando

UPDATE iria colocar todas as linhas com o mesmo número de telefone.

Exercício 17.5

Construa um comando que altere o último nome do utilizador com telefone 963343386 para "Sousa".

Finalmente, é possível apagar linhas inteiras utilizando o comando **DELETE**. Deve-se ter o mesmo cuidado que com o comando **UPDATE** no sentido em que o comando **DELETE** pode apagar uma ou mais linhas. O exemplo seguinte apaga uma linha específica da tabela:

```
DELETE FROM contacts WHERE phone = 912345653;
```

Modelo Relacional

Em geral, uma aplicação terá várias tabelas, cada uma com um tipo de dados. Por exemplo, à nossa base de dados podemos acrescentar uma tabela das empresas clientes, às quais pertencem os clientes anteriormente listados.

Table: companies

name	address	vatnumber
MaxiPlano	Aveiro	123123123123
Luís Manuel & filhos	Águeda	54534343435
ProDesign	Porto	54534343435

Exercício 17.6

Crie uma tabela chamada **companies** e insira a informação anterior.

Levanta-se agora a questão de como indicar que uma dada pessoa pertence a uma empresa. Uma solução passaria por armazenar o nome da empresa junto de cada contacto. E a morada e número fiscal da empresa? Também se acrescentam à tabela de contactos? Mas isso implicaria replicar informação, visto que várias pessoas pertencem à mesma empresa. Isso é um problema porque caso uma empresa mudasse a morada, teríamos de pesquisar todos os contactos e alterar esse atributo, ou senão ficaríamos com informações incoerentes!

A solução para esta questão passa por estabelecer uma *relação* entre as tabelas da base de dados.⁵² Para criar relações entre dados, cada registo de uma tabela deve ter uma *chave*, que pode depois ser utilizada noutra tabela para o referir. Uma chave é um qualquer atributo que permita identificar univocamente o registo. As chaves podem ser de qualquer tipo, mas são tipicamente números inteiros, que até podem ser atribuídos de forma automática quando o registo é criado.

Para estabelecer a relação no nosso exemplo, a tabela das empresas passaria a ser:

Table: companies

id	name	address	vatnumber
1	MaxiPlano	Aveiro	123123123123
2	Luis Manuel & filhos	Águeda	54534343435
3	ProDesign	Porto	54534343435

Enquanto que a tabela de contactos ficaria:

Table: contacts

id	firstname	middlename	lastname	email	phone	comp
1	João	Manuel	Fonseca	jmf@gmail.com	912345654	3
2	Pedro	Albuquerque	Silva	pedro23@gmail.com	932454349	2
3	Maria	Carreira	Dinis	mariadi@ua.pt	234958673	1
4	Catarina	Alexandra	Rodrigo	calexro@sapo.pt	963343386	1

Em ambas as tabelas, a coluna **id** é uma chave: serve para identificar de forma única cada registo. Na tabela **contacts**, o campo **comp** contém a chave da empresa a que pertence cada pessoa. Neste caso o João pertence à empresa ProDesign, enquanto a Maria e a Catarina pertencem ambas à empresa MaxiPlano.

Assim, a informação entre contactos e empresas encontra-se relacionada. A relação é estabelecida pelo campo **comp** da tabela contactos. A este tipo de campo chama-se uma *chave estrangeira* por conter valores que são chaves (ditas *chaves primárias*) noutra tabela.

Os comandos **SELECT** podem ser construídos de forma a permitir pesquisar informação relacionada distribuída por diversas tabelas. Por exemplo, poderíamos listar todos os contactos da empresa MaxiPlano com o seguinte comando:

```
SELECT contacts.*
  FROM contacts,companies
 WHERE contacts.comp = companies.id
   AND companies.name = "MaxiPlano"
```

⁵²É esta característica que está na origem do termo Base de Dados Relacional.

Analizando o comando verifica-se que ele lista todos os campos dos regtos da tabela **contacts** que possuam no atributo **comp** dessa tabela um valor igual ao do atributo **id** da tabela **companies** sempre que **name** nessa mesma tabela seja igual a MaxiPlano.

Exercício 17.7

Volte a criar uma tabela de empresas com uma nova coluna chamada **id**. Especifique o tipo como sendo **INTEGER PRIMARY KEY AUTOINCREMENT**. Isto irá criar uma coluna que armazena um contador inteiro, único e incrementado automaticamente quando novas linhas são inseridas. Recomenda-se que se crie uma base de dados diferente da anterior.

Exercício 17.8

Aplique o mesmo processo à tabela de contactos mas neste caso adicione também uma nova coluna no final chamada **comp**. Esta coluna serve para indicar a que empresa pertence um dado contacto.

17.2 Acesso Programático

Na secção anterior vimos os comandos básicos que permitem criar, aceder e modificar uma base de dados relacional. Nesta secção veremos como fazer o mesmo dentro de um programa.

Como em geral as bases de dados podem residir num sistema servidor, distinto do sistema que executa o programa cliente, o procedimento a seguir envolve três fases:

1. estabelecer a ligação à base de dados;
2. fazer as consultas e/ou alterações aos dados;
3. terminar a ligação.

Estas fases também são seguidas mesmo no caso da base de dados residir num ficheiro local.

Em *Python* o acesso a bases de dados do tipo *sqlite* é fornecido pelo módulo **sqlite3**, que tem de ser importado. Para instalar o módulo, pode recorrer ao gestor de pacotes do seu sistema, ou ao comando **pip** de acordo com as permissões que possuir.

```
apt-get install python-sqlite
```

ou:

```
pip install --user pysqlite
```

O exemplo seguinte mostra o esqueleto de um programa que abre um ficheiro fornecido no primeiro argumento (p.ex, **database.db**).

```
import sqlite3 as sql
import sys

def main(argv):
    db = sql.connect(argv[1])      # estabelecer ligação à BD
    ...
    # realizar operações sobre a BD
    db.close()                   # terminar ligação

main(sys.argv)
```

O método **connect** estabelece a ligação à base de dados e devolve um objeto que representa essa ligação. É através desse objeto que serão realizadas as operações na base de dados e deve-se depois terminar a ligação. Se o argumento do método **connect** for igual a "**:memory:**", a base de dados é criada em memória, sendo apagada no final do programa. Isto é útil caso de pretenda armazenar informação apenas durante a execução do programa, não sendo ela válida numa futura execução.

Exercício 17.9

Construa um programa como indicado anteriormente e verifique que pode aceder à base de dados que criou anteriormente.

Tal como anteriormente, as operações sobre a base de dados são codificadas através de comandos SQL. As respostas podem depois ser obtidas sob o formato de linhas da tabela. O exemplo seguinte demonstra como é possível obter todos os registo da tabela de contactos.

```
import sqlite3 as sql
import sys

def main(argv):
```

```
db = sql.connect(argv[1])

result = db.execute("SELECT * FROM contacts")
rows = result.fetchall()
for row in rows:
    print row

db.close()

main(sys.argv)
```

Neste exemplo o ciclo percorre os registo devolvidos pelo comando **SELECT** e, em cada iteração, a variável **row** recebe um tuplo com os atributos de um desses registo. Também é possível obter os resultados um de cada vez, o que por vezes é obrigatório caso o seu tamanho seja muito grande.⁵³ Para isso podemos utilizar o método **fetchone()**:

```
...
result = db.execute("SELECT * FROM contacts")
while True:
    row = result.fetchone()
    if not row:
        break
    print row
...
```

Ou pode-se simplesmente tratar o resultado como um iterador:

```
...
result = db.execute("SELECT * FROM contacts")
for row in result:
    print row
...
```

⁵³As bases de dados possuem sempre um limite (por exemplo, 4MB) para o tamanho de cada resultado.

Exercício 17.10

Considere os métodos anteriores e implemente um programa que imprima os nomes próprios de todos os contactos e indique quantos são, como no exemplo abaixo.

```
Catarina  
João  
Maria  
Pedro  
4 contactos
```

Frequentemente é necessário realizar pesquisas com argumentos variáveis. Por exemplo, podemos querer pesquisar todos os contactos cujo email pertença a um domínio indicado pelo utilizador. Assim, se o utilizador introduzir `%@gmail.com`, serão encontrados todos os contactos com email naquele servidor.

Uma solução naïf seria construir uma *String* com a estrutura do comando pretendido:⁵⁴

```
...  
domain = raw_input("Domínio de email? ")  
result = db.execute("SELECT * FROM contacts WHERE email LIKE '%s'" % domain)  
# ATENÇÃO: ESTA SOLUÇÃO DEVE SER EVITADA!  
...
```

Esta abordagem, embora frequente, **deve ser absolutamente evitada!** Caso contrário, correm-se sérios riscos de segurança. Por exemplo, se o utilizador introduzir

```
%@gmail.com'; DELETE FROM contacts --
```

o comando SQL construído seria

```
SELECT * FROM contacts WHERE email LIKE '%@gmail.com'; DELETE FROM contacts --'
```

o que inclui duas instruções: **SELECT** e **DELETE**. Efectivamente o resultado é que todos os dados seriam apagados.

A solução correcta é a seguinte.

⁵⁴O operador **LIKE** permite pesquisar valores com um certo padrão, que pode incluir partes desconhecidas.

```
...
domain = raw_input("Domínio de email? ")
result = db.execute("SELECT * FROM contacts WHERE email LIKE ?", (domain,) )
# ESTA É A SOLUÇÃO CORRETA!
...
```

Note que se utiliza um ponto de interrogação para indicar explicitamente onde deve ser colocado o parâmetro variável e o seu valor é passado separadamente. Desta forma é o próprio módulo que constrói internamente o comando SQL e assim é impossível injetar comandos adicionais. Esta abordagem tem o nome de *Prepared Statements* e deve ser seguida independentemente da linguagem de programação.

Exercício 17.11

Construa um programa que permita pesquisar contactos por qualquer um dos nomes. Para isto pode utilizar o operador **OR** para conjugar diferentes condições de pesquisa. Por exemplo: **...WHERE firstname LIKE ? OR middlename LIKE ?**

Exercício 17.12

Construa um programa que aceite um argumento, usando-o para localizar uma pessoa através das partes do seu nome, imprimindo a que empresa pertence.

17.3 Para Aprofundar

Exercício 17.13

Obtenha a base de dados Chinook acedendo ao endereço <http://chinookdatabase.codeplex.com/downloads/get/557773> e crie um programa que permita pesquisar por albums de música. O esquema da base de dados pode ser encontrado em http://chinookdatabase.codeplex.com/wikipage?title=Chinook_Schema.

Exercício 17.14

Obtenha a base de dados Chinook e crie uma aplicação que demonstre quais os 10 clientes que efectuaram o maior volume de compras.

Exercício 17.15

Relembre um exercício anterior em que se capturava a taxa de ocupação de CPU.
Replique o exercício, mas registando os valores numa base de dados relacional.

Exercício 17.16

Implemente um programa que, tendo em consideração a base de dados criada no exercício anterior, imprima o valor médio de ocupação de processador entre duas datas.

Representação de Som

Objetivos:

- Representação de informação sonora
- Operações sobre som

18.1 Princípios de acústica

No mundo físico o som é transmitido através de ondas sonoras que são flutuações contínuas da pressão do ar ao longo do tempo e do espaço. Se a frequência de flutuação for alta, resulta um som agudo. Se a frequência for baixa, resulta um som grave. A amplitude da flutuação está relacionada com a *força* do som. Assim, um som muito fraco como um sussurro tem uma amplitude muito baixa, enquanto um som forte como um motor tem uma amplitude mais alta. A figura 18.1 mostra as flutuações de pressão de um som ao longo de um intervalo de tempo num certo ponto do espaço. O som representado é mais fraco e agudo no início, mais forte e grave no fim.

Um tom dito puro corresponde a uma variação de pressão que é uma função sinusoidal do tempo: $\Delta p(t) = A \sin(2\pi ft)$. Aqui, $\Delta p(t)$ representa a variação de pressão do ar no instante t , A é a amplitude da variação e f é a frequência da variação, indicada em ciclos por unidade de tempo.⁵⁵ A maioria dos sons, mesmo quando emitidos por instrumentos musicais, têm formas de onda mais complexas, mas que se podem sempre considerar como combinações de tons puros (sinusóides) de diferentes frequências e amplitudes (e diferentes desfasamentos). A combinação de diferentes frequências, com diferentes

⁵⁵A unidade de frequência é o Hertz (Hz) e corresponde a um ciclo por segundo: $1\text{Hz} = 1\text{s}^{-1}$.

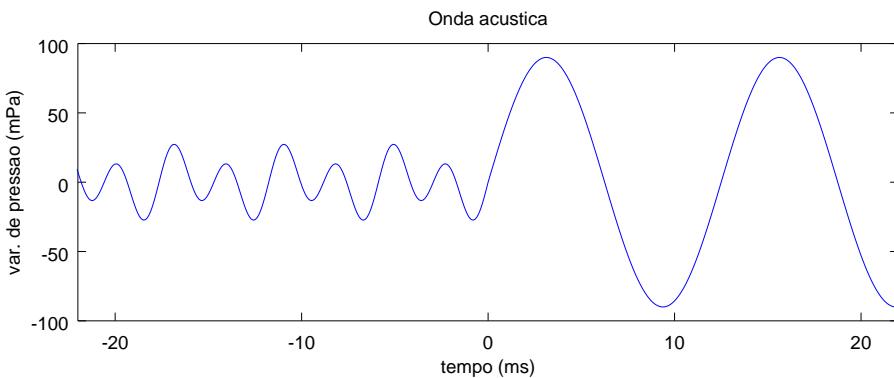


Figura 18.1: Forma de onda de um som medida ao longo do tempo num ponto do espaço.

amplitudes para cada frequência, produz toda a multíitude de sons que somos capazes de reconhecer.

A Figura 18.2 demonstra as frequências emitidas por sete notas diferentes de um piano. O eixo horizontal representa o tempo, enquanto o vertical representa a frequência das componentes sinusoidais do som. Uma cor quente (vermelho/branco) num ponto da figura indica uma componente forte (amplitude alta) nessa frequência e instante de tempo. Cores frias (azul, cinza) indicam componentes fracas ou mesmo ausentes. Este tipo de representação chama-se um espetrograma. Como se pode ver, cada nota contém múltiplas componentes de frequências bem definidas (linhas horizontais), com amplitude suavemente decrescente ao longo da duração da nota. Isto é mais evidente na última nota, mais aguda, em que as componentes aparecem mais afastadas entre si, a primeira com $f \approx 800\text{Hz}$ e as seguintes em frequências múltiplas dessa. Também se percebe, no início de cada nota, uma maior intensidade, mas mais espalhada por diversas frequências (linhas verticais), o que é característico de sons mais curtos e explosivos, que neste caso correspondem a componentes transitórias do som causadas pela ação de percussão dos martelos nas cordas.

Exercício 18.1

Utilize o programa *Audacity* e analise o ficheiro **piano-c5-c6.wav** que foi fornecido pelos docentes. Em particular, experimente as diferentes formas de visualização do sinal, comutando entre forma de onda e espectrograma, por exemplo. Pode aceder a esta função se pressionar a seta que se encontra à direita do nome do ficheiro, do lado esquerdo da aplicação. Também pode selecionar um trecho curto de uma das notas e usar a função de *Analyze->Plot Spectrum* para ver o espetro desse segmento. Os picos no espetro indicam as frequências mais fortes presentes nesse trecho.

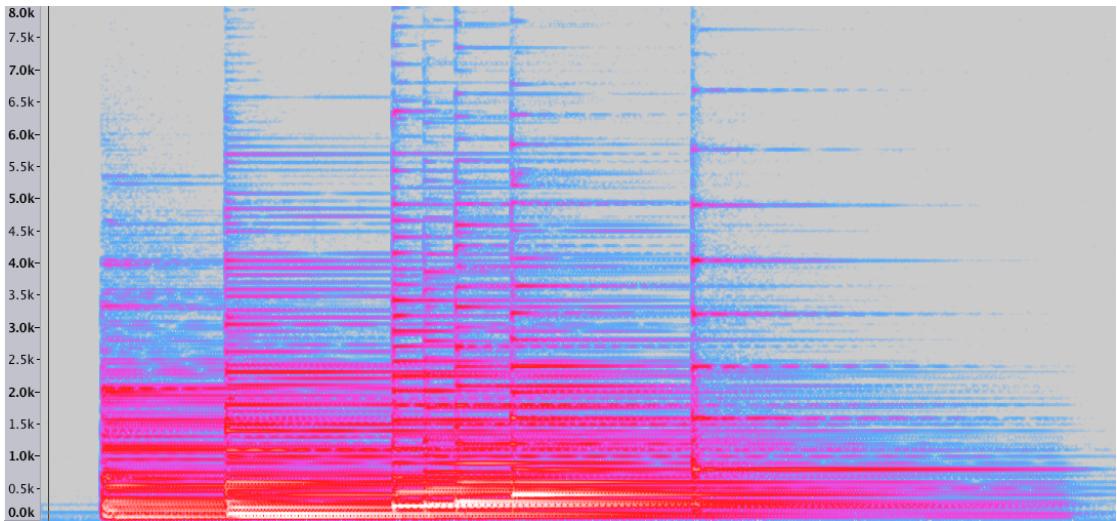


Figura 18.2: Várias notas de um piano capturadas num espetrograma.

18.2 Representação de informação sonora

Um microfone converte as variações de pressão em variações de tensão (ou de intensidade) de uma corrente elétrica. A tensão do sinal elétrico varia continuamente em função do tempo, de forma análoga à variação de pressão do sinal acústico. Por isso diz-se que é um *sinal analógico*.⁵⁶ Sistemas eletrónicos analógicos (formados por resistências, condensadores, transístores e outros componentes) permitem amplificar, processar e até armazenar sinais analógicos diretamente. Os rádios AM ou FM, as televisões antigas (não TDT), os gravadores de fita magnética de áudio (cassetes) ou vídeo (VHS), os giradiscos de vinil são exemplos de sistemas de transmissão, processamento e armazenamento puramente analógicos. O problema destes sistemas é que em cada passo de processamento, os sinais vão-se degradando com o acumular de pequenas distorções, interferências e ruído eletrónico.

Os sistemas digitais resolvem esse problema, mas para os usar, é preciso transformar os sinais analógicos em sinais digitais. Para isso, recorre-se a um conversor analógico-digital, ou Analog to Digital Converter (ADC), que é um dispositivo que faz duas operações:

1. tira amostras instantâneas do sinal a intervalos regulares (amostragem) e
2. mede a tensão de cada amostra, e converte-a num número binário com um número fixo de dígitos (quantização).

⁵⁶Deveria talvez designar-se *sinal análogo*, mas *sinal analógico* está legitimado pelo uso.

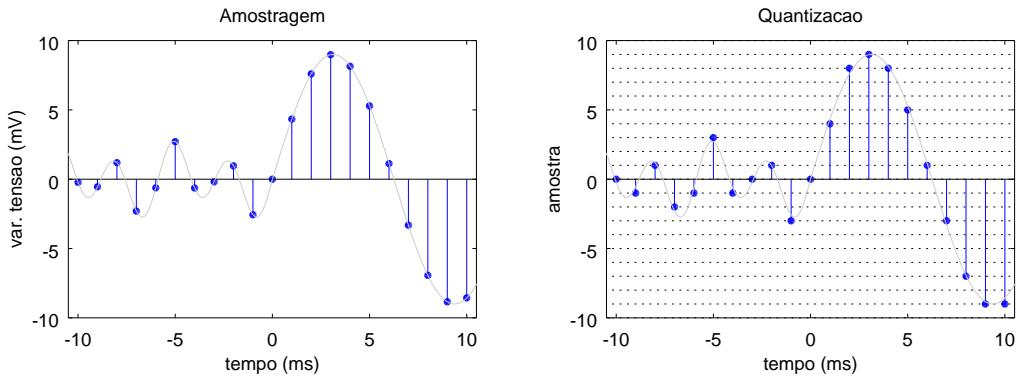


Figura 18.3: Conversão analógico-digital: amostragem e quantização. O sinal analógico original é representado em cinzento. As linhas verticais azuis representam os instantes de amostragem. A quantização aproxima as amplitudes por valores de um conjunto finito.

A figura 18.3 mostra estas duas operações aplicadas a um trecho do sinal analógico correspondente ao som da figura 18.1.

O sinal analógico que é uma função real, contínua no tempo, é assim convertido numa sequência de números inteiros. Este *sinal digital* resultante é portanto descontínuo no tempo e descontínuo em amplitude. Não consegue por isso representar a infinidade de detalhe que existe numa onda que varia continuamente ao longo do tempo, mas tem a vantagem de se poder armazenar, processar e transmitir virtualmente sem acumulação de degradação.

O número de amostras que uma ADC tira por segundo é chamada a sua *frequência de amostragem* e o número de bits usado em cada amostra é a sua *resolução*. As ADCs usadas atualmente para sinais áudio utilizam resoluções típicas de 8, 16 ou 24 bits e frequências de amostragem típicas de 8000, 11025, 22050, 44100, 48000 ou 96000Hz. Quanto maiores forem este valores, mais precisa será a representação digital feita do som original. No entanto, também será necessário possuir um sistema mais veloz e mais espaço de armazenamento para a informação. Os valores ideais dependem das limitações da sensibilidade auditiva e da aplicação.

De acordo com teorema da amostragem de Nyquist, a frequência de amostragem deve ser, no mínimo, o dobro da máxima frequência do sinal registado para permitir a sua reconstrução exata. Como o ouvido humano detecta frequências até perto dos 20Khz, será necessária uma frequência de amostragem superior a 40000Hz para registar devidamente todas as frequências audíveis. Não é portanto surpresa que a frequência de amostragem típica para registo musical (CD, MP3) seja de 44100Hz (ou 48000Hz). Quando se pretende registar voz, como as componentes mais importantes da voz humana estão compreendidas entre 100 e 3000Hz, basta uma frequência de amostragem de 8000Hz, que

é um valor popular nas comunicações móveis.

Para reproduzir um som a partir de um sinal digital, é preciso fazer o processo inverso: usar um Digital to Analog Converter (DAC) para converter a sequência de números num sinal elétrico analógico; amplificar esse sinal e convertê-lo em ondas de pressão acústica através de um altifalante.

Armazenamento de som

Num computador o som é armazenado através da sequência de valores medidos nos instantes de amostragem. Um ficheiro sonoro pode registar uma sequência obtida de um único microfone ou pode ter várias sequências obtidas em simultâneo de vários microfones. Chama-se canal a cada sequência registada em simultâneo no mesmo ficheiro. São vulgares os ficheiros *monofónicos* (ou *Mono*), com um canal apenas, e os ficheiros estereofónicos (*Stereo*), com dois canais (esquerdo/direito), mas é possível ter ficheiros com 7 ou mais canais como é o caso dos ficheiros para os sistemas *Surround*.

A informação em si pode estar armazenada sob a forma de texto, mas tal só é comum em aplicações científicas. De resto espera-se que esteja armazenada numa forma binária, eventualmente comprimida. A compressão destina-se a reduzir os requisitos para o armazenamento de informação. Há métodos de compressão chamados *Lossless* (sem perdas), que permitem a recuperação exata da sequência de valores originais, e métodos de compressão *Lossy* (com perdas), que introduzem distorção nos sinais, mas fazem-no descartando ou alterando componentes menos perceptíveis pelo sistema auditivo humano. Esta é a abordagem seguida quando se cria um ficheiro *MP3*, por exemplo. Portanto, os sons guardados num formato *Lossy* não são iguais ao original, mas usando taxas de compressão razoáveis, soam-nos igual ao original. Durante este guião o foco serão os ficheiros não comprimidos como é o caso do formato WAVEform audio file format (WAVE) pois facilitam a manipulação da informação contida.

Os ficheiros WAVE são constituídos por um pequeno cabeçalho onde é possível indicar alguma meta-informação, sendo este cabeçalho seguido de blocos com a informação sonora, geralmente no formato Linear Pulse Code Modulation (LPCM). A cada bloco dá-se o nome de *Frame*, e contém os valores registados para cada canal num certo instante de amostragem. Cada um dos valores numa *Frame* é uma amostra (*Sample*) de um dos canais e é codificada num número de bytes suficiente para a resolução usada. Por exemplo, considerando um ficheiro com dois canais, com resolução de 16 bits e frequência de amostragem de 44100Hz, cada segundo de som teria 44100 frames com 2 samples de 2 bytes cada, ou seja, um ritmo de 176400 octetos por segundo. Nestas condições, uma música de 5 minutos gera um ficheiro com um pouco mais de 50MB. Este formato está apresentado na Figura 18.4.

Em *Python* é possível inspecionar os metadados dos ficheiros WAVE e mesmo obter a informação sonora. Também é possível criar novos ficheiros, o que será efectuado na

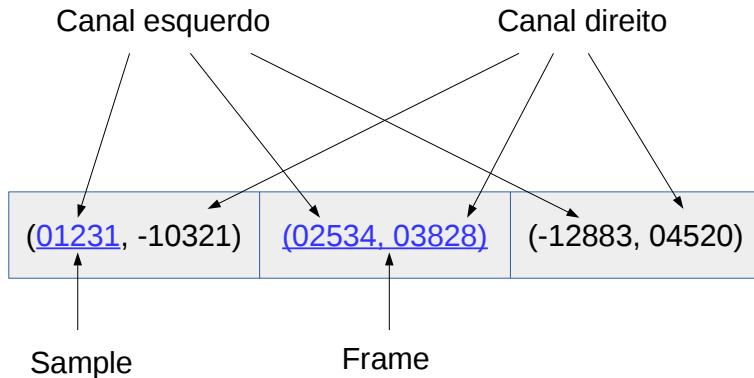


Figura 18.4: Estrutura dum ficheiro WAVE.

Secção 18.2. Para isto é necessário utilizar o módulo `wave`⁵⁷ que pode ser instalado das formas usuais em *Python*. O exemplo seguinte demonstra como pode ser aberto um ficheiro WAVE e obtida informação do seu cabeçalho.

```
import wave
import sys

def main(argv):
    wf = wave.open(argv[1], "rb")
    print wf.getnchannels()
    ...
    wf.close()

main(sys.argv)
```

Exercício 18.2

Implemente um programa que obtenha a frequência de um ficheiro, o tamanho de cada *Sample*, o número de canais e o número de *Frames* de som contidas no ficheiro.

Também é possível obter os dados sonoros e reproduzir o som directamente de forma programática. O módulo que permite isto possui o nome de **PyAudio** podendo ser instalado através de `pip` ou pelo gestor de pacotes da distribuição. O exemplo seguinte cria um `player` que pode ser utilizado para reproduzir um ficheiro WAVE.

⁵⁷ver <https://docs.python.org/2/library/wave.html>

```

import pyaudio
player = pyaudio.PyAudio()

...

stream = player.open(format = player.get_format_from_width(sample_width),
                      channels = nchannels,
                      rate = frame_rate,
                      output = True)

while True:
    data = wf.readframes(1024)
    if not data:
        break

    stream.write(data)

stream.close()
player.terminate()

```

Exercício 18.3

Melhore o programa anterior de forma a que apresente informação de um dado ficheiro e o reproduza. Experimente modificar a variável `frame_rate` para um qualquer outro valor e volte a reproduzir o ficheiro.

Geração de tons

Além da leitura de ficheiros WAVE, ou a sua construção através da leitura do microfone, também é possível a criação de sons através da sintetização das diversas frequências de uma forma matemática. Isto porque sendo um som composto por uma onda a oscilar numa frequência específica e com uma determinada amplitude, esta onda pode ser recriada através da função `sin` (seno) multiplicada por um factor de amplitude.

Para gerar um tom com frequência f é necessário criar uma sequência de valores através da expressão:

$$v(i) = amplitude * \sin\left(\frac{2 * \pi * freq * i}{rate}\right) \quad (4)$$

em que $v(i)$ representa a amostra no instante i , $freq$ a frequência desejada e $rate$ a frequência de amostragem do som (p.ex 44100Hz).

Aplicando a fórmula à linguagem *Python*, podem ser gerados ficheiros WAVE com tons puros da seguinte forma:

```

from struct import pack
from math import sin, pi
import wave
import sys

def main(argv):
    rate=44100
    wv = wave.open(argv[1], "w")
    wv.setparams((1, 2, rate, 0, "NONE", "not compressed"))

    amplitude = 10000
    data = []
    freq = 440
    duration = 1 # Em segundos
    for i in range(0, rate * duration):
        data.append(amplitude*sin(2*pi*freq*i/rate))

    # Gerar (pack) a informação no formato correto (16bits)
    wvData = []
    for v in data:
        wvData += pack("h", int(v))

    wv.writeframes(bytarray(wvData))
    wv.close()

main(sys.argv)

```

Exercício 18.4

Implemente o exemplo anterior e gere ficheiros com vários tons. Analise os ficheiros criados através da aplicação Audacity.

Podemos criar sons compostos somando tons de múltiplas frequências gerados em simultâneo. Por exemplo, para criar um som com duas componentes, uma a 440Hz e outra a 880Hz, podemos fazer:

```

...
freq_a = 440
freq_b = 880
for i in range(0, rate):
    data.append(
        amplitude*sin(2*math.pi*freq_a*i/rate) +
        amplitude*sin(2*math.pi*freq_b*i/rate)
    )
...

```

Exercício 18.5

Crie um novo programa baseado no anterior que gere um som composto por dois tons. Analise o resultado na aplicação *Audacity*.

A simplicidade deste método foi explorada em muitos sistemas, sendo que um dos mais famosos é o sistema Dual-Tone Multi-Frequency signaling (DTMF). Este sistema é utilizado para enviar algarismos e outros 6 símbolos através de ligações analógicas. Cada símbolo é codificado como um par de tons com certas frequências e enviado para o receptor. O receptor separa e deteta o par de frequências para descodificar o símbolo. A tabela de codificação é a seguinte:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

A figura 18.5 mostra o espectrograma de um número codificado no sistema DTMF.

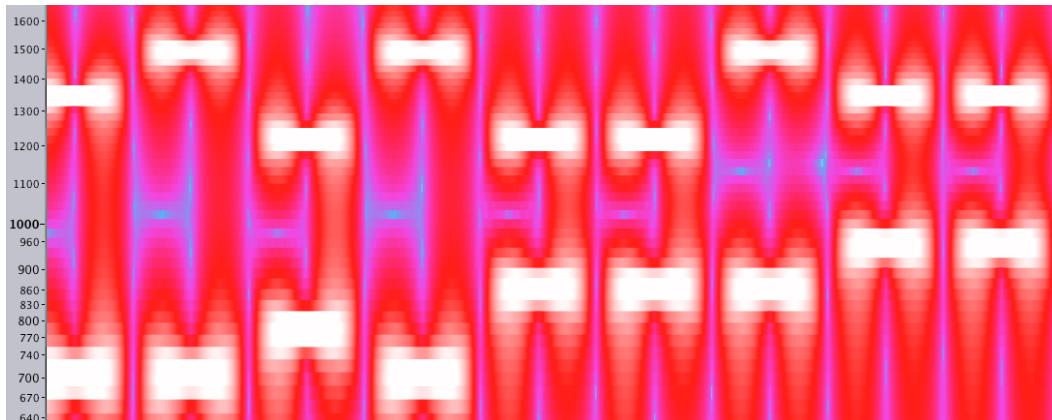


Figura 18.5: Número de telefone codificado em DTMF

Exercício 18.6

Identifique qual o número de telefone representado na Figura 18.5.

Em Python uma maneira simples de implementar esta tabela seria através de um dicionário, em que cada símbolo possui uma lista com as frequências a utilizar.

```
...  
tones = {\  
    "1": [697, 1029], \  
    "2": [697, 1336], \  
    ...}  
...
```

Isto pode depois ser utilizado para gerar sons DTMF, com duração de 40ms, seguidos de uma pausa de outros 40ms, tal como um telefone ou telemóvel actual fazem. O exemplo seguinte demonstra a estrutura básica de um programa para codificar qualquer número em DTMF.

```
...  
tones = {...}  
number = "" # número a codificar  
for n in number:  
    # Códigos DTMF  
    for i in range(0, int(rate*0.040)):  
        data.append(  
            # Valores dos tons  
        )  
    # Pausa (silêncio)  
    for i in range(0, int(rate*0.040)):  
        data.append(  
            # Silêncio  
        )  
...
```

Exercício 18.7

Crie um programa que leia um número do teclado e gere um ficheiro com os códigos DTMF respectivos. Analise o resultado na aplicação Audacity.

18.3 Operações sobre som

São várias as operações que podem ser efectuadas sobre os ficheiros de som, além claro de os reproduzir. Nomeadamente é possível aplicar transformações, normalmente denominados de efeitos, que alterem as características sonoras da informação. Muitas das transformações são complexas, necessitando de conceitos mais complexos sobre o proces-

samento de sinal. Alguns são bastante triviais ou relativamente simples de implementar, em particular os que operam sobre a informação numa perspectiva puramente temporal.

O seguinte trecho de código *Python* mostra uma forma bastante geral de aplicar uma qualquer transformação a um ficheiro WAVE, devolvendo o resultado noutro ficheiro do mesmo tipo. As secções seguintes deverão fazer uso deste programa, ou de um com estrutura semelhante para testar os efeitos desenvolvidos.

```
import wave
import struct
import sys
from struct import pack
import math

def copy(data):
    output = []
    for index,value in enumerate(data):
        output.append(value)
    return output

def main(argv):
    stream = wave.open(argv[1], "rb")

    sample_rate = stream.getframerate()
    num_frames = stream.getnframes()

    raw_data = stream.readframes( num_frames )
    stream.close()

    data = struct.unpack("%dh" % num_frames, raw_data) # "B" para ficheiros 8bits

    # Aplica efeito sobre data, para output_data
    i = 2
    output_data = []
    while i < len(argv):
        if argv[i] == "copy":
            output_data = copy(data)
        elif argv[i] == "foo":
            param = int(argv[i+1])
            output_data = foo(data, param)
            i += 1
        elif... #Outros filtros

        i += 1

    wvData = []
    for v in output_data:
        wvData += pack("h", int(v))
```

```

stream = wave.open("out-"+argv[1], "wb")
stream.setnchannels(1)
stream.setsampwidth(2)
stream.setframerate(sample_rate)
stream.setnframes(len(wvData))
stream.writeframes(bytarray(wvData))
stream.close()

if len(sys.argv) < 3:
    print("Usage: %s wave-file filter1 <params> filter2 <params> ..." % sys.argv[0])
else:
    main(sys.argv)

```

Exercício 18.8

Crie um programa com o código anterior e verifique que consegue processar um ficheiro WAVE, criando uma cópia semelhante ao original. Use para isso um filtro chamado `copy`. invoque o programa criado com a sintaxe: `python process.py file.wav copy`.

Exercício 18.9

Adicione um filtro ao código anterior que devolva `reversed(data)` e verifique que consegue processar um ficheiro WAVE, criando uma cópia invertida do original. Use para isso um filtro chamado `reverse`. invoque o programa criado com a sintaxe: `python process.py file.wav reverse`.

Controlo de Volume

O volume a que o som é reproduzido depende essencialmente da amplitude do sinal, o que no caso de um ficheiro WAVE é representado pelo valor em absoluto de cada impulso. Para controlar o volume basta multiplicar todos os valores de amplitude por um factor multiplicativo. Se este factor for 0.5 o volume deverá ser diminuído em metade. Se for 2.0 o volume deverá ser multiplicado por 2.

Exercício 18.10

Implemente um filtro chamado `volume` que aceite um factor multiplicativo como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav volume 0.5`. Verifique o resultado obtido através da aplicação Audacity.

Normalização

A normalização de um ficheiro diz respeito a controlar o volume de forma a que o valor máximo encontrado corresponda ao valor máximo possível. No caso de um ficheiro de 16bits, um ficheiro normalizado para o valor máximo deverá conter pelo menos um valor igual a -32768 ou a igual a 32767.

Este filtro necessita de dois passos de processamento. O primeiro determina qual o valor absoluto máximo dos valores. Daqui pode-se calcular um factor multiplicativo. O segundo passo aplica o factor multiplicativo tal como no caso do filtro de volume.

Exercício 18.11

Implemente um filtro chamado **normalize**. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav normalize`. Verifique o resultado obtido através da aplicação *Audacity*.

Fade In-Out

Um filtro de *Fade* aplica um envelope progressivo à amplitude do som de forma a que o seu volume aumente ou diminua de forma contínua. A Figura 18.6 demonstra um som a que foi aplicado um *Fade In* e *Fade Out*.

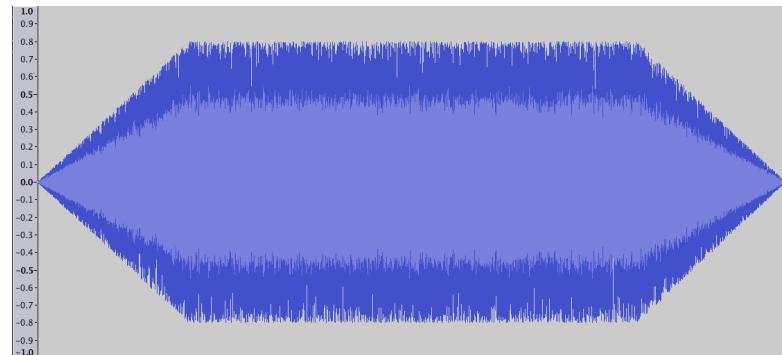


Figura 18.6: Exemplo de *Fade In* e *Fade Out*

A aplicação deste filtro é em tudo semelhante ao controlo de amplitude, com a diferença que o valor de amplitude é variável e só aplicado num intervalo temporal. Para ambos os casos é importante determinar onde iniciar e terminar a aplicação do efeito, que deve ter em consideração a frequência de amostragem do som. Também se deve ter em consideração qual o declive do factor multiplicativo a aplicar. Desta forma, o valor final do sinal será $vf_i = vo_i * index * step$. Em que vf_i representa o valor final i , vo_i o valor

original i , $index$ o número do *Sample* e $step$ o tamanho de cada incremento ao longo do processo de *Fade*.

O código Python seguinte demonstra o início de uma função aplicando este efeito:

```
def fade-in(data, sample_rate, duration):
    time_start = 0
    time_stop = duration * sample_rate
    step = 1.0 / (sample_rate * duration)
    for index, value in enumerate(data):
        ...
```

Exercício 18.12

Implemente um filtro chamado **fade-in** que aceite uma duração em segundos como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav fade-in 2`. Verifique o resultado obtido através da aplicação *Audacity*.

Exercício 18.13

Implemente um filtro chamado **fade-out** que aceite uma duração em segundos como parâmetro. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav fade-out 2`. Verifique o resultado obtido através da aplicação *Audacity*.

Pode inclusive aplicar os dois efeitos usando: `python process.py file.wav fade-in 2 fade-out 2`

Máscaras

A aplicação de máscaras serve para omitir parte do conteúdo do som. É frequentemente utilizado para remover partes sensíveis, tal como palavras menos cuidadas. A operação deste filtro resume-se à substituição dos valores sonoros por outros no intervalo pretendido. Estes novos valores (vo_i) podem ser o resultado de:

- uma sinusóide com uma frequência específica (um tom): $vo_i = amplitude * \sin(\frac{2 * \pi * freq * i}{rate})$
- silêncio: $vo_i = 0$
- valores aleatórios: $vo_i = random.randint(-32768, 32767)$

Este filtro pode ter como parâmetro o tipo de máscara a aplicar, o instante de tempo inicial e o instante de tempo final para aplicação do efeito. Tal como no caso do filtro anterior é necessário calcular em que *Sample* iniciar a máscara e em que *Sample* terminar a máscara.

Exercício 18.14

Implemente um filtro chamado **mask** que aceite como parâmetro um tipo de máscara com os valores **silence**, **noise**, **tone**, um instante de início e uma duração em segundos. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav mask silence 2 2`. Verifique o resultado obtido através da aplicação Audacity.

Modulação

A modulação tem como princípio multiplicar um dado som por um outro tom. Considerando $vf(i)$ o valor final, $vo(i)$ o valor original, $freq$ uma frequência de modulação e $rate$ uma frequência de amostragem, o filtro pode ser concretizado com:

$$vf(i) = vo(i) * \sin\left(\frac{2 * \text{math.pi} * freq * i}{rate}\right) \quad (5)$$

Uma variante deste efeito, mas considerando um tom variável, é o *Wah Wah* aplicado a guitarras e voz. O resultado de usar um tom fixo é um som com aspecto metálico, como se tivesse sido emitido por um robot num filme de televisão. Se a frequência for muito baixa o resultado é apenas uma variação cíclica no volume do som original.

Exercício 18.15

Implemente um filtro chamado **modulate** que aceite como parâmetro uma frequência em Hertz. Deve conseguir invocar o programa desenvolvido através da sintaxe: `python process.py file.wav modulate 3000`. Verifique o resultado obtido através da aplicação Audacity.

Atraso

O atraso é normalmente chamado de *Reverb* ou *Echo* e consiste na repetição de um sinal emitido num instante t_i , para um instante $t_i + x$ mas com uma amplitude ligeiramente inferior. Devidamente aplicado este efeito confere profundidade ao som, simulando as ondas refletidas naturalmente quando um som é reproduzido numa sala.

Considerando uma lista **output** que irá conter o resultado final, e uma lista **data** que contém o som original, pode-se construir este filtro através da seguinte estrutura:

```
output = [0] * len(data)+tdelay
...
for index,value in enumerate(data):
    output[index] += value
    output[index+tdelay] += value * amount
```

Neste caso o valor **tdelay** consiste no número de *Samples* que se pretende atrasar o som (consideram-se valores na ordem de 0.5-1.5 segundos), e **amount** consiste na força do efeito. Considera-se um valor inferior a 1. Um aspeto interessante deste efeito é que ele pode ser aplicado de forma recursiva, sendo que em cada nova iteração o efeito deve ser aplicado mais tarde e ter menos força.

Considerando que o efeito se encontra implementado numa função chamada **delay**, pode-se implementar este princípio da seguinte forma:

```
def delay(data, sample_rate, amount, delay):
    if amount < 0.05:
        return data
    ...
    ...

#Repetir com 80% da força e com 20% mais de atraso.
return delay(output, sample_rate, amount * 0.8, delay * 1.2)
```

Exercício 18.16

Implemente um filtro chamado **delay** que aceite como parâmetro um atraso segundos e uma força. Deve conseguir invocar o programa desenvolvido através da sintaxe: **python process.py file.wav delay 0.8 0.6**. Verifique o resultado obtido através da aplicação *Audacity*.

Exercício 18.17

Implemente a versão recursiva deste filtro e avalie a complexidade do efeito resultante, em comparação com a versão mais simples.

Esteganografia

Os ficheiros de som, em particular no formato WAVE também permitem a aplicação de técnicas de esteganografia. Em particular é simples codificar mensagens através da manipulação do último bit de cada *Sample*. Pode-se considerar que para codificar um valor 0, o valor do último bit deverá ser 0, sendo 1 para codificar um valor 1. Quando aplicado com a linguagem *Python*, é possível codificar um texto em *Samples* sonoros através da seguinte estrutura:

```
def steg_add(data, message):
    bitstream = "" # Irá conter uma string. ex: "011101010"
    for c in message:
        bitstream += format(ord(c),2)

    output = []
    encoded_bit = 0
    for index, value in enumerate(data):
        ....
```

A descodificação é muito semelhante sendo que um dado carácter **c** (neste caso o primeiro) pode ser obtido de uma *String* com os bits fazendo: **c = chr(int(decoded_bits[0:8],2))**.

Normalmente é útil adicionar um marcador para sinalizar a existência e/ou final de uma mensagem, separadores ou mesmo códigos de detecção de erros.

Exercício 18.18

Implemente um filtro chamado **steg_add** que aceite como parâmetro uma mensagem. Deve conseguir invocar o programa desenvolvido através da sintaxe: **python process.py file.wav steg_add 'mensagem de teste'**. Verifique o resultado obtido através da aplicação Audacity.

Exercício 18.19

Implemente um filtro chamado **steg_get**. Deve conseguir invocar o programa desenvolvido através da sintaxe: **python process.py file.wav steg_get** e este deverá imprimir a mensagem previamente escondida.

18.4 Para aprofundar

Exercício 18.20

Considere as notas musicais podem ser reconstruídas a partir de uma nota inicial segundo a seguinte fórmula: $freq_n = 2^{n/12} * 440$. Relembre que as notas são: La, La#, Si, Do, Do#, Re, Re#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si, Do..., correspondendo estas notas a valores de n entre 1 e 16.

Implemente um programa que leia uma sequência de notas e as reproduza (p.ex: 1 1 8 8 10 10 8)

Exercício 18.21

Melhore o programa anterior de forma a considerar duração das notas e pausas entre as notas. Pode fazê-lo através de um carácter como o '-' para indicar a duração e 0 para indicar uma pausa. Uma nota Do# com um terço da duração ficaria 2-3 (Do-um_terço) e uma pausa com um quarto da duração normal de uma nota ficaria 0-4 (Pausa-um_quarto).

Exercício 18.22

Desenvolva outros efeitos a aplicar a informação sonora, nomeadamente:

- Expansor: Sons menores com uma amplitude menor que x são aumentados para o máximo de amplitude. Os restantes mantêm-se inalterados.
- Compressor: Possui dois intervalos x_{max} e x_{min} . Amplitudes superiores a x_{max} são iguais a x_{max} , enquanto amplitudes inferiores a x_{min} são iguais a x_{min} .
- Limitador: Limita a amplitude a um valor. Ou seja, se o valor for superior a x , este passa a x .

Aplicações Móveis Web

Objetivos:

- Conceitos sobre aplicações móveis
- Conceitos sobre HTML para ambientes móveis
- Interacção com câmara e GPS
- Comunicação com serviços externos

19.1 Introdução

Os ambientes móveis, nomeadamente os focados em *smartphones* e *tablets*, possuem métodos próprios para o desenvolvimento de aplicações, sobre a forma de Software Development Kits (SDKs). Esses ambientes possibilitam aceder às capacidades do dispositivo, nomeadamente às APIs para acesso à câmara fotográfica, captura de som, localização, ou outros sensores. As aplicações desenvolvidas podem igualmente trocar informação com servidores, permitindo a implementação de aplicações ricas de troca de mensagens, troca de imagens ou mesmo jogos.

Uma desvantagem desta abordagem é que as três principais plataformas actualmente em utilização (*Windows Phone*, *Android*, *iOS*) apresentam APIs completamente diferentes. Além disso, todas estas plataformas exigem que as aplicações sejam desenvolvidas em linguagens diferentes. A plataforma *Android* utiliza *Java*, a plataforma *Windows Phone* utiliza *C#* e a plataforma *iOS* utiliza *ObjectiveC*. Desenvolver uma aplicação para as três plataformas obriga a triplicar parte do trabalho, o que levanta vários problemas para o planeamento e manutenção do código produzido, assim como para o planeamento do modelo de interacção.

Felizmente, os navegadores Web presentes nas várias plataformas possuem capacidades bastante avançadas de processamento. Assim, em vez de uma aplicação nativa, podemos muitas vezes criar uma aplicação Web, que corre igualmente em qualquer plataforma. A grande vantagem é que a aplicação pode ser desenvolvida uma única vez, usando tecnologias standard como *JavaScript* e *HTML5*. Ainda para mais, bibliotecas como o *Twitter Bootstrap* foram desenhadas para permitirem uma visualização correta das páginas Web tanto em dispositivos móveis como em computadores com ecrãs maiores. A página do serviço *Facebook*, que se ajusta automaticamente ao dispositivo cliente, é um bom exemplo da capacidade de escalabilidade das páginas atuais.

É claro que as aplicações web também têm algumas desvantagens: a velocidade de execução pode ser inferior, o aspeto da aplicação não será igual ao das aplicações nativas, a execução depende da existência de uma ligação de dados e pode não ser possível aceder a todos os recursos disponíveis nativamente.

Para o desenvolvimento deste tipo aplicações é comum utilizarem-se sistemas como o *PhoneJS* ou *PhoneGap*. No entanto, a utilização destes sistemas requer conhecimentos mais avançados. Por isso, este guião foca-se na utilização das funcionalidades através de uma variante da biblioteca *Twitter Bootstrap* denominada de *Ratchet* que pode ser obtida em <http://goratchet.com/>. A documentação desta biblioteca pode ser obtida em <http://goratchet.com/components/>. Deve consultar esta documentação antes da execução deste guião.

19.2 Uma aplicação simples

As aplicações não são mais do que páginas Web em que a lógica é implementada em *JavaScript* e um conjunto de páginas de estilos criam a ilusão de se tratar de uma aplicação real. Antes de iniciar este ponto, aceda à documentação da biblioteca *Ratchet* e verifique que componentes estão disponíveis.

O desenvolvimento inicia-se obtendo a biblioteca *Ratchet* e colocando os ficheiros necessários nos directórios corretos (directório **dist**). Tem também de existir um ficheiro **index.html** que representará a aplicação.

O exemplo seguinte resulta numa aplicação apenas com uma barra de título com o texto **LABI**.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>LABI</title>
<meta name="viewport"
      content="initial-scale=1, maximum-scale=1, user-scalable=no, minimal-ui">
```

```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style" content="black">
<link rel="stylesheet" href="css/ratchet.css">
<link rel="stylesheet" href="css/ratchet-theme-ios.css">
<link rel="stylesheet" href="css/app.css">
<script src="js/jquery-2.2.3.min.js"></script>
<script src="js/ratchet.js"></script>
</head>
<body>
<header class="bar bar-nav">
<h1 class="title">LABI</h1>
</header>
<div class="bar bar-footer">
</div>
<div class="content">
<ul class="table-view">
<li class="table-view-cell">Hello World</li>
</ul>
</div>
</body>
</html>
```

Exercício 19.1

Crie um directório `hello/` e coloque nele um ficheiro `index.html` com o conteúdo acima. Obtenha a biblioteca *Ratchet* e copie o conteúdo do directório `dist/` para dentro do directório `hello/`. Verifique o resultado com o seu browser.

Para verificar a aplicação num telemóvel, pode enviar a pasta completa para o servidor `xcoa.av.it.pt` e depois aceder à página Web a partir do telemóvel. Ou então verifique a versão dos professores em `http://xcoa.av.it.pt/labi/labi2016/ratchet/hello/`.

Em alternativa ao `xcoa`, se estiver numa rede local sem firewalls, poderá activar um servidor HTTP no seu computador para servir o conteúdo do diretório atual e poder aceder noutra dispositivo. Pode fazer isto executando `python -m SimpleHTTPServer`. (Na rede da UA, devido às firewalls instaladas, isto não resultará.)

Exercício 19.2

Modifique a referência do tema para `ratchet-theme-android.css` e verifique que a aplicação muda para um aspeto semelhante ao sistema android.

Exercício 19.3

Adicione um componente adicional à aplicação desenvolvida.

19.3 Localização

Frequentemente os dispositivos móveis possuem um equipamento de Global Positioning System (GPS) que permite determinar a localização com bastante exactidão. No entanto, é possível determinar a localização de outras formas, por exemplo através das redes sem fios disponíveis. Os diversos meios de determinação de localização foram assim englobados numa mesma API que permite obter a localização independente do método utilizado. Para salvaguardar a privacidade, o acesso da aplicação à localização requer uma autorização explícita pelo utilizador.

O código JavaScript abaixo demonstra como obter a localização.

```
function showPosition(position){  
    $("#location").html(position.coords.latitude+" "+position.coords.longitude);  
}  
  
$(document).ready(function() {  
    navigator.geolocation.getCurrentPosition(showPosition);  
});
```

O método `navigator.geolocation.getCurrentPosition` pede acesso à localização e regista a função `showPosition` como *callback* para ser chamada quando o resultado estiver disponível.

Na aplicação cliente será necessário incluir este código *JavaScript* e um elemento HTML com o identificador certo para apresentar o resultado.

```
<div class="content">  
    <div id="location"></div>  
</div>
```

Recomenda-se a utilização da biblioteca *jQuery* que tem de ser incluída no directório `js` e importada para a página (no `<head>`).

Exercício 19.4

Altere a sua aplicação de forma a mostrar a localização do utilizador. Pode testar localmente ou enviar a página para o servidor `xcoa.av.it.pt`.

Exercício 19.5

Utilizando *LeafletJS* adicione um mapa centrado na posição atual do dispositivo.

19.4 Comunicação com serviços externos

A comunicação com serviços externos é importante pois permite que uma aplicação possa trocar informação com serviços, ou mesmo com outros utilizadores. Aplicações de *chat*, de visualização do estado do tempo, ou de partilha de outra informação (ex, imagens), podem ser implementadas rapidamente. Acima de tudo, este tipo de aplicações é útil para demonstrar que as aplicações Web, mesmo em ambientes móveis, podem ser dinâmicas e interagir com outros serviços externos.

Um exemplo simples é o de obter a data e hora de um servidor remoto. Para isto serão necessários os seguintes componentes:

- Um botão para iniciar o processo (pode ser substituído por um *timer* caso se pretenda actualizar a informação de forma periódica.)
- Código *JavaScript* que obtenha a informação do servidor remoto.
- Uma aplicação que implemente o serviço pedido.
- Um elemento HTML para armazenar o resultado.

Os dois elementos HTML são adicionados no elemento de classe **content**.

```
...
<div class="content">
    <button id="refresh" class="btn btn-block btn-primary">Refresh</button>
    <div id="clock" style="width:100%; text-align:center;"></div>
</div>
```

Enquanto que o código *JavaScript* é adicionado num ficheiro que tipicamente se denomina **app.js**. Neste caso, o código espera que seja enviado um elemento JSON com dois atributos: **date** e **time**. O pedido é activado quando o elemento com identificador **refresh** receber um evento de **click**.

```
function refresh(){
    $.get("/time", function(response){
        var text=<h2>"+response.date+"</h2><br /><h2>"+response.time+"</h2>";
        $("#clock").html(text);
```

```

    });
}

$( document ).ready(function() {
    $("#refresh").on("click",refresh);
});

```

Do lado do serviço, é necessário implementar um método que devolva a data e hora, como o seguinte.

```

import cherrypy
import time

class HelloWorld(object):
    @cherrypy.expose
    def index(self):
        # Método serve_file tb poderia ser utilizado
        f = open("index.html")
        data = f.read()
        f.close()
        return data

    @cherrypy.expose
    def time(self):
        cherrypy.response.headers["Content-Type"] = "application/json"
        return time.strftime('{"date":"%d-%m-%Y", "time":"%H:%M:%S"}')

cherrypy.server.socket_port = 8080
cherrypy.server.socket_host = "0.0.0.0"
cherrypy.tree.mount(HelloWorld(),"/","app.config")
cherrypy.engine.start()
cherrypy.engine.block()

```

Note que alguns ficheiros são estáticos, enquanto outros são gerados dinamicamente. Os ficheiros estáticos encontram-se nos directórios **css** e **js**. Desta forma, é necessário criar um ficheiro chamado **app.config** com o seguinte conteúdo:

```

[/]
tools.staticdir.root = "/home/utilizador/directorio-do-servico"

[/css]
tools.staticdir.on: True
tools.staticdir.dir: "css"

[/js]
tools.staticdir.on: True
tools.staticdir.dir: "js"

```

Exercício 19.6

Construa um exemplo que demonstre a comunicação entre uma aplicação Web e um serviço, através de JSON.

Exercício 19.7

O exemplo anterior pode ser modificado de forma a devolver algo mais útil, tal como a distância para o estádio do SL Benfica (38.752667, -9.184711).

Usando a função seguinte componha um exemplo que envie a localização actual do cliente para o servidor, que responderá devolvendo a distância para o estádio.

```
from math import radians, cos, sin, asin, sqrt
...
def distance(lat, lon):
    lat1 = 38.752667
    lon1 = -9.184711

    lon, lat, lon1, lat1 = map(radians, [lon, lat, lon1, lat1]) # Graus -> rads

    dlon = lon - lon1
    dlat = lat - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat) * sin(dlon/2)**2 # Haversine
    c = 2 * asin(sqrt(a))

    km = 6367 * c # 6367=raio da terra

    cherrypy.response.headers["Content-Type"] = "application/json"
    return json.dumps({"distance": km})
...
```

19.5 Acesso a imagens

Através de APIs específicas é igualmente possível aceder às imagens armazenadas num dispositivo móvel ou mesmo activar a câmara e obter uma imagem. Estas podem depois ser processadas localmente ou enviadas para servidores. Este método recorre a um elemento **<input>**, especificando que se pretende aceder a fotografias.

```
...
<div class="content">
    <input type="file" accept="image/*"></input>
</div>
...
```

O resultado será que o dispositivo irá pedir ao utilizador que selecione o método de entrada, podendo este ser a câmara ou os documentos já existentes. Se se especificar o atributo `capture="camera"`, a câmara será activada imediatamente para que possa capturar uma imagem.

Exercício 19.8

Integre o exemplo anterior numa aplicação e verifique o que acontece quando activa o input criado.

Exercício 19.9

Adicione as classes `btn`, `btn-primary`, `btn-block` e `button-input`. Considere depois o seguinte excerto de CSS.

```
.button{
    text-align: center;
    color: #007aff;
    width: 137px;
}
.button::before{
    color: white;
    content: 'Usar Imagem';
    padding-right: 40px;
    margin-top: -20px;
    padding-left: 10px;
}
```

Componha a aplicação e teste.

Depois de estar selecionada a imagem, seria interessante poder visualizá-la. Isto pode ser feito se for activada uma função depois da fotografia ter sido seleccionada e existir um elemento onde a apresentar. O elemento neste caso será um `<canvas>`. Este elemento HTML é semelhante a um ``, com a diferença que é possível desenhar para ele em tempo real, enquanto que um elemento `` apresenta uma imagem estática. O novo HTML seria:

```
...
<input type="file" accept="image/*" onchange="updatePhoto(event);"></input>
<canvas id="photo" width="530" height="400">
...

```

O código *JavaScript* necessário corresponde à função `updatePhoto()` e irá aplicar a imagem capturada ao elemento `<canvas>`:

```
function updatePhoto(event){
    var reader = new FileReader();
    reader.onload = function(event){
        //Criar uma imagem
        var img = new Image();
        img.onload = function(){
            //Colocar a imagem no ecrã
            canvas = document.getElementById("photo");
            ctx = canvas.getContext("2d");
            ctx.drawImage(img,0,0,img.width,img.height,0,0,530, 400);
        }
        img.src = event.target.result;
    }

    //Obter o ficheiro
    reader.readAsDataURL(event.target.files[0]);
    sendFile(event.target.files[0]);
}
```

Exercício 19.10

Componha uma aplicação que replique o exemplo anterior.

Eventualmente a imagem pode ser enviada para um servidor remoto. Será necessário criar código *JavaScript* para construir um pedido de envio (`POST`):

```
function sendFile(file) {
    var data = new FormData();
    data.append("myFile", file);
    var xhr = new XMLHttpRequest();
    xhr.open("POST", "upload");
    xhr.upload.addEventListener("progress", updateProgress, false);
    xhr.send(data);
}
```

```

function updateProgress(evt){
    if(evt.loaded == evt.total)
        alert("OK!");
}

function updatePhoto(evt){
    ...

    sendFile(image[0]);

    //Libertar recursos da imagem seleccionada
    windowURL.revokeObjectURL(picURL);
}

```

Do lado do servidor será depois necessário receber o ficheiro. Os ficheiros enviados são fornecidos ao serviço e é necessário copiar a informação para um local mais permanente. Isto porque os dados serão apagados assim que o método **upload** terminar. O código de exemplo será o seguinte:

```

@cherrypy.expose
def upload(self, myFile):
    fo = open(os.getcwd() + '/uploads/' + myFile.filename, 'wb')
    while True:
        data = myFile.file.read(8192)
        if not data:
            break
        fo.write(data)
    fo.close()

```

Exercício 19.11

Crie um conjunto de aplicações que permita o envio de imagens para o servidor.

Exercício 19.12

Altere o exemplo de forma a enviar outros ficheiros além de imagens.

Representação de Imagem

Objetivos:

- Representação de cor nos diferentes espaços
- Efeitos básicos sobre imagens
- Marcas de água

20.1 Introdução

As imagens como fotografias, desenhos ou gráficos são deveras importantes para as aplicações computacionais actuais. Neste guião analisamos as formas de representação e processamento digital de imagens.

Para a realização deste guião será necessário instalar a biblioteca **Pillow**. Num computador com Debian/Ubuntu pode instalá-la através do comando:

```
sudo apt-get install python-imaging
```

Em alternativa, pode instalar o pacote através do comando **pip**, mas antes terá de instalar várias dependências. Assim, em *Ubuntu* deverá correr:

```
sudo apt-get install libtiff4-dev libjpeg8-dev zlib1g-dev \
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.5-dev tk8.5-dev python-tk
pip install Pillow
```

Em *OS X* deverá correr:

```
brew install libtiff libjpeg webp little-cms2  
pip install Pillow
```

Por razões pedagógicas, neste guião serão sugeridos processos que podem ser sub-ótimos. A biblioteca **Pillow** possui transformações otimizadas para muitos dos processos aqui tratados, devendo código de produção fazer uso desses métodos. A documentação da biblioteca pode ser encontrada no endereço <http://pillow.readthedocs.org/en/latest/reference>.

20.2 Imagens

Num computador, as imagens são representadas na forma de uma matriz composta de pequenas células organizadas em linhas e colunas. A cada célula da matriz dá-se o nome de píxel e corresponde ao menor elemento da imagem que pode ter cor distinta dos restantes. A *geometria da imagem* é definida pela largura e altura, medida em número de píxeis (ver Figura 20.1).

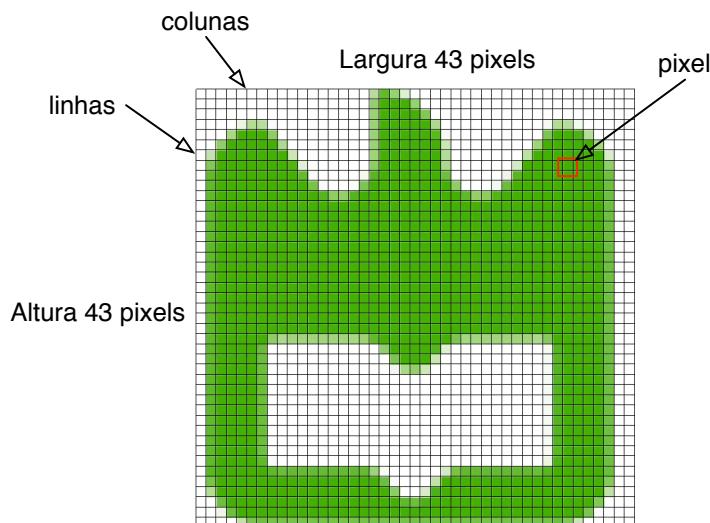


Figura 20.1: Composição de uma imagem digital.

Além da informação para representação da imagem, a maioria dos formatos de representação adiciona dados extra aos ficheiros, que são utilizados para indicar como deve o ficheiro ser processado ou que codificação é utilizada. Alguns outros, como o formato Joint

Photographic Experts Group (JPEG), podem incluir dados num formato denominado Exchangeable image file format (Exif), que indicam entre outros aspectos que programa produziu a imagem ou qual a máquina fotográfica utilizada e os parâmetros da lente (no caso de uma fotografia). Também pode ser incluída uma versão miniatura da imagem para pré-visualização no navegador de ficheiros do sistema.

Num programa em *Python* é possível inspeccionar esta informação através da biblioteca **Pillow**, como se demonstra no exemplo abaixo.

```
from PIL import Image
from PIL import ExifTags
import sys

def main(fname):
    im = Image.open(fname)

    width, height = im.size

    print "Largura: %dpx" % width
    print "Altura: %dpx" % height
    print "Formato: %s" % im.format

    tags = im._getexif()

    for k,v in tags.items():
        print str(ExifTags.TAGS[k])+" : "+str(v)

main(sys.argv[1])
```

Exercício 20.1

Utilize o programa anterior para analisar os ficheiros de imagem fornecidos.

O tamanho em píxeis de uma imagem apresentada num monitor pode não ser igual ao tamanho da imagem armazenada. Por exemplo, quando se utiliza HTML, o tamanho de apresentação de uma imagem pode ser especificado através de atributos que se aplicam ao elemento ``, independentemente do tamanho original da imagem. Este redimensionamento obriga o computador a gerar uma nova imagem mais pequena ou maior que a original. Note que este processo é destrutivo e não cria informação. Ou seja, ao reduzir a dimensão de uma imagem, não se está a reduzir a dimensão de cada píxel, mas antes a substituir vários por um só, perdendo informação dos outros. Ao aumentar a dimensão de uma imagem, criam-se novos píxeis, mas os seus valores (cores) são dependentes apenas dos píxeis originais; a definição da imagem não melhora, fica

apenas mais “esbatida”.

Podem-se criar imagens com diversos tamanhos usando o código seguinte.

```
...
def main(fname):
    im = Image.open(fname)
    width, height = im.size

    for s in [0.2, 8]:
        dimension = ( int(width*s), int(height*s) )
        new_im = im.resize( dimension, Image.NEAREST)
        new_im.save(fname+"-%.2f.jpg" % s)

...

```

Exercício 20.2

Implemente o código anterior e experimente modificar a dimensão de alguns ficheiros.

Exercício 20.3

O método utilizado para modificar as imagens é um dos mais simples (**Nearest Neighbour**), Existem vários outros, tais como **BILINEAR**, **BICUBIC** ou **ANTIALIAS**. Uns são mais adaptados à redução, outros à ampliação. Teste-os e compare visualmente o resultado.

20.3 Formatos de ficheiros

Existem vários formatos para a representação de imagens, sendo que cada um é optimizado para um tipo particular de informação. O formato mais utilizado para representar fotografias é sem dúvida o formato JPEG. Já os conteúdos gráficos na Web utilizam sobretudo o formato Portable Network Graphics (PNG). Outro formato popular é o Tagged Image File Format (TIFF), especialmente nos meios criativos, ou o BitMaP image file (BMP) em sistemas mais antigos. Existem razões objetivas para preferir um formato em detrimento de outro. As diferentes formas de representação e de compressão de dados dos diferentes formatos são mais ou menos adequados consoante o tipo de imagem (se é um desenho ou uma fotografia, por exemplo).

No caso concreto da compressão, os formatos BMP, PNG e TIFF não aplicam compressão ou aplicam uma compressão sem perda de informação. Por outro lado, o formato JPEG aplica algoritmos de compressão com perdas para conseguir reduzir substancialmente

o tamanho dos ficheiros. Como os algoritmos estão otimizados para imagens naturais como fotografias, os artefactos introduzidos tornam-se mais evidentes quando aplicados a imagens artificiais como desenhos com áreas de cores sólidas e alto contraste.

O formato JPEG possui uma escala de compressão que varia em 0 e 100, sendo que quanto mais baixo for o valor, maior compressão e maiores perdas serão observadas. A Figura 20.2 demonstra o resultado de utilizar uma compressão de 30% para uma imagem com cores sólidas. Como se pode notar, a imagem foi bastante adulterada e torna-se evidente que o algoritmo processa as imagens por blocos.

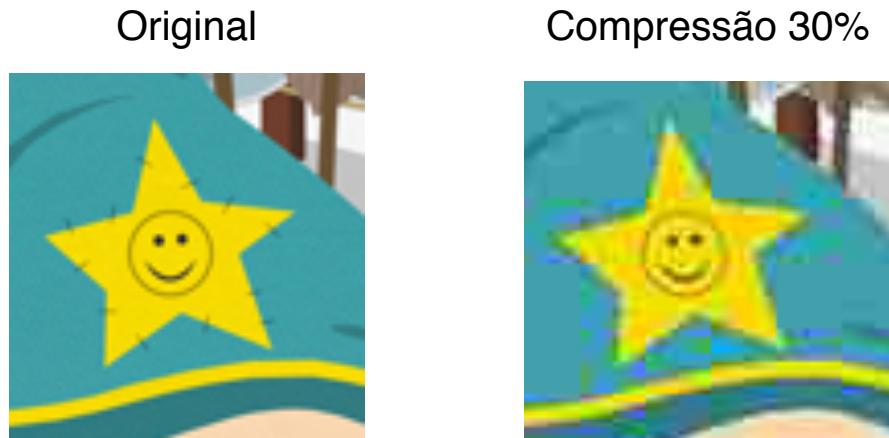


Figura 20.2: Compressão com JPG.

Exercício 20.4

Determine o tamanho dos blocos utilizados para comprimir a imagem anterior. Recomenda-se que simplesmente aumente o zoom do visualizador e conte os píxeis. (É conveniente desativar qualquer opção de suavização nas preferências do visualizador.) A imagem possui uma geometria de 90px por 88px.

O exemplo que se segue comprime o mesmo ficheiro com 11 valores de qualidade distintos e pode ser utilizado para verificar este aspeto.

```
from PIL import Image
import sys
```

```

def main(fname):
    im = Image.open(fname)
    for i in [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]:
        im.save(fname + "-test-%i.jpg" % i, quality=i)

main(sys.argv[1])

```

Exercício 20.5

Utilize o exemplo anterior para criar versões comprimidas do ficheiro `southpark.png` e do ficheiro `vasos.jpg`. Determine para ambos os casos a partir de que valor de qualidade os erros adicionados perturbam a imagem. Tenha em especial atenção as áreas de alto contraste, tais como linhas.

Exercício 20.6

Escolha um dos ficheiros fornecidos em formato JPEG e crie uma versão em cada um dos formatos: PNG, TIFF, BMP. Compare o tamanho do ficheiro criado.

20.4 Representação de cor

Existem várias formas para representar cores numa imagem. A escolha do formato mais apropriado depende do tipo de imagem que se possui e do objectivo da informação. O método mais comum e já visto nas aulas anteriores (ex, HTML) é o formato **RGB**. Neste formato, cada cor é representada por 3 componentes de cores primárias: vermelho (R), verde (G), e azul (B). A Figura 20.3 apresenta alguns valores **RGB** do logótipo da Universidade de Aveiro.

As cores **RGB** são denominadas por cores primárias aditivas. Ao se somarem estas cores é possível reconstruir todas as outras. Em contrapartida, o modelo **CMYK** (Cyan, Magenta, Yellow, Black), constituído pelas cores primárias subtrativas também é capaz de formar todas as cores, mas quando elas são subtraídas (absorvidas). Num ecrã os píxeis emitem luz, portanto as cores somam a sua intensidade e é utilizado o modelo **RGB**. Numa impressora as tintas depositadas num papel absorvem luz e por isso é frequentemente utilizado o modelo **CMYK**.

Existem vários modos de representação de cor que podem ser comumente utilizados:

- **BW ou 1:** 1 canal com apenas 1 bit por píxel, que pode representar uma de duas cores. Utilizado para representar imagens a preto e branco.

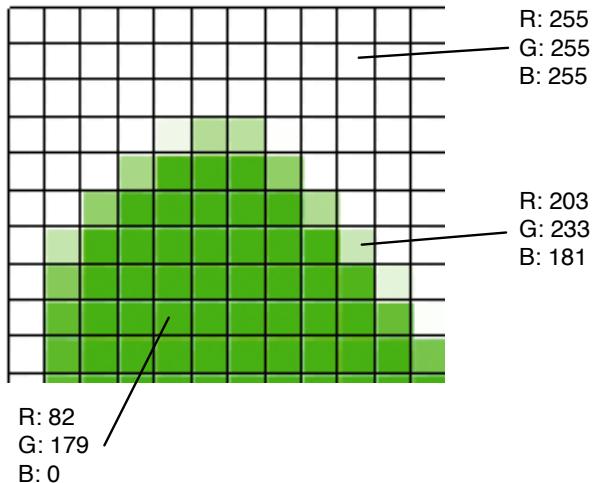


Figura 20.3: Valores RGB do logótipo da UA

- **RGB:** 3 canais, representando as intensidades de vermelho, verde e azul. Muito utilizado para a representação de cores em monitores.
- **RGBA:** O mesmo que **RGB** com um canal adicional (canal alfa) para representar a transparência. Suportado por alguns formatos como o PNG.
- **CMYK:** 4 canais, utilizando ciano/cião, magenta, amarelo e preto. Muito utilizado para a representação de cores para impressão.
- **L:** Apenas um canal representando a luminância (intensidade de luz). Utilizado para representar imagens em tons de cinza.
- **P:** Paleta de cores específica. Cada píxel da imagem tem um número que é usado como índice para uma tabela de cores (uma *paleta*), que contém a cor desejada num dos outros formatos (tipicamente RGB). Utilizada para formatos como o Graphics Interchange Format (GIF).
- **YCbCr:** 3 canais representando a luminância (Y), crominância azul (Cb), crominância vermelha (Cr). A luminância pode ser obtida por uma média ponderada dos valores RGB, enquanto os canais Cb/Cr são determinados pela diferença entre os valores de vermelho/azul e a luminância, $Cb = B - Y$ e $Cr = R - Y$. O sistema é utilizado para fotografias (JPEG) e em vídeo.

Exercício 20.7

Verifique o atributo `mode` de uma imagem para cada um dos ficheiros fornecidos.

É possível converter as imagens entre modos de representação, o que pode trazer vantagens do ponto de vista de representação e processamento. Para casos de utilização dos modos de cor, consulte a secção 20.5.

Repare que na Figura 20.3 os valores apresentados se referem a uma escala com $2^8 = 256$ níveis para cada cor, sendo que o total de cores representáveis será igual a $2^{3 \times 8}$. O número pode parecer grande mas na realidade não é adequado para todos os fins. As máquinas fotográficas atuais produzem imagens *RAW* com resoluções de 14 a 16 bits por componente, que depois são convertidas para o formato de 8 bits. No processo perde-se informação, pelo que alguns formatos como o TIFF permitem guardar 8 bits, 16 bits e 32 bits por componente. Outros como o Flexible Image Transport System (FITS) permite um número indeterminado de bits por píxel.

Não sendo possível analisar em concreto e de forma fácil o resultado de se terem imagem de 16bits, pode-se fazer o exemplo contrário: restringir o número de bits de uma imagem e observar como isso altera as cores percebidas pelo olho humano. O exemplo seguinte anula (coloca a 0) os 4 bits menos significativos de cada componente, reduzindo o ficheiro para 4 bits de resolução efetiva.

```
...
def main(fname):
    im = Image.open(fname)

    width, height = im.size

    for x in xrange(width):
        for y in xrange(height):
            p = im.getpixel( (x,y) )
            r = p[0] & 0b11110000
            g = p[1] & 0b11110000
            b = p[2] & 0b11110000
            im.putpixel( (x,y), (r,g,b) )

    im.save(fname+"-4bits.jpg")
...
```

Exercício 20.8

Replique o exemplo anterior e experimente anular quantidades diferentes de bits.

20.5 Efeitos sobre imagens

São várias as manipulações que podem ser aplicadas a imagens de forma a alterar o seu aspeto. Podem focar-se na manipulação das cores ou mesmo na alteração da geometria da imagem. As sub-secções seguintes demonstram como algumas manipulações podem ser conseguidas. Os programas foram escritos como forma de explicar os métodos da forma mais simples. Privilegiou-se a legibilidade e compreensibilidade, não a eficiência. A biblioteca **Pillow** possui métodos otimizados para muitas destas operações, que deverão ser utilizados em situações reais.

Recomenda-se que se implemente cada efeito como uma função que aceite como parâmetro uma imagem e devolva novamente uma imagem. Desta forma torna-se possível encadear efeitos.

Troca de Cores

A troca de cores das imagens é um efeito simples que resulta da troca dos canais de uma imagem. Tipicamente aplicado no modo **RGB** pois permite um controlo mais direto sobre a imagem. O exemplo seguinte troca o canal verde pelo vermelho, sendo o resultado o demonstrado na Figura 20.4

```
...
    new_im = Image.new(im.mode, im.size)

    for x in range(width):
        for y in range(height):
            p = im.getpixel( (x,y) )
            r = p[1]
            g = p[0]
            b = p[2]
            new_im.putpixel((x,y), (r, g, b) )
...

```

Exercício 20.9

Construa uma função que troque os canais de cores de uma imagem.



Figura 20.4: Figura original em RGB e com os canais R e G trocados.

Exercício 20.10

Construa uma função que crie uma imagem negativa. Esta imagem consiste na substituição de cada valor v por $255 - v$.

Tons de Cinza

Converter para tons de cinza implica remover toda a informação cromática, deixando apenas a intensidade. No modo **YCbCr** isto pode ser feito de forma imediata mantendo apenas o primeiro canal, mas não é o método mais poderoso. A biblioteca **Pillow** converte igualmente de forma automática qualquer imagem para tons de cinza, especificando o modo **L**.

O exemplo seguinte converte uma imagem para o modo **L** e guarda-a.

```
...
def main(fname):
    im = Image.open(fname)
    new_im = im.convert("L")
    new_im.save(fname + "-L.jpg")
...

```

O resultado é o demonstrado na Figura 20.5.

Converter uma imagem para escala de cinzas implica processar as suas cores e aplicar uma fórmula que converta **RGB** (ou outro modo) em **L**. No caso anterior a fórmula utilizada é $L = \frac{299}{1000}R + \frac{587}{1000}G + \frac{114}{1000}B$, mas podem ser utilizados outras expressões, tal como utilizar apenas uma cor, ou combinar as cores de forma diferente. Frequentemente existem vantagens em aplicar um processamento diferente. O exemplo seguinte aplica a fórmula descrita, mas de uma forma manual.



Figura 20.5: Figura original em RGB e em tons de cinza (L).

```
...
def effect_gray(im):
    width, height = im.size
    new_im = Image.new("L", im.size)

    for x in xrange(width):
        for y in xrange(height):
            p = im.getpixel( (x,y) )
            l = int(p[0]*0.299 + p[1]*0.587 + p[2]*0.144)
            new_im.putpixel( (x,y), (l) )

    return new_im
...
```

Exercício 20.11

Replique o exemplo anterior mas combinando as cores de forma diferente. Pode, por exemplo, considerar apenas uma ou duas cores, combinadas ou utilizadas sem qualquer processamento. Verifique o resultado final.

Controlo de Intensidade

O controlo de intensidade resulta na alteração dos valores de intensidade da imagem. Valores mais altos resultam numa imagem mais clara, valores mais baixos resultam numa imagem mais escura. Esta operação é bastante simplificada quando aplicada no modo **YCbCr**, pois o primeiro canal (Y) é o único que possui informação de intensidade.

A Figura 20.6 demonstra o resultado de manipular a intensidade de uma imagem.

A aplicação deste efeito requer assim uma conversão de modo de representação e a



Figura 20.6: Figura com intensidade aumentada ($f=1.5$) ou diminuída ($f=0.5$).

multiplicação do canal Y por um factor. Se o factor for superior a 1 a imagem ficará com uma intensidade superior, se o valor for inferior a 1 ela aparecerá mais escura. É necessário ter em atenção que os valores resultantes têm de ser inteiros e nunca podem ultrapassar o mínimo ou o máximo da escala.

```
def effect_intensity(im, f):
    new_im = im.convert("YCbCr")
    width, height = im.size

    for x in xrange(width):
        for y in xrange(height):
            pixel = new_im.getpixel( (x,y) )
            py = min(255, int(pixel[0] * f))      # [0] is the Y channel
            new_img.putpixel( (x,y), (py, pixel[1], pixel[2]) )
    ...
```

Exercício 20.12

Construa uma função que multiplique a intensidade de um ficheiro por um factor.

Controlo de Gama

A gama de uma imagem diz respeito a quanto linear é a representação da intensidade dos seus valores. Tipicamente as imagens necessitam de ser corrigidas de forma a que a intensidade seja adaptada ao meio de reprodução. Nos Cathode Ray Tubes (CRTs), já raramente utilizados, é necessário aplicar curvas de compensação, pois a sua intensidade de reprodução não é linear. A Figura 20.7 apresenta este problema. Os CRTs possuem uma gamma típica de 2.2, o que significa que reproduzem as cores segundo uma linha

exponencial. Uma correção típica irá distorcer as cores com o valor $\frac{1}{2.2}$ de forma que a imagem realmente percebida pelos utilizadores seja apresentada de forma linear.

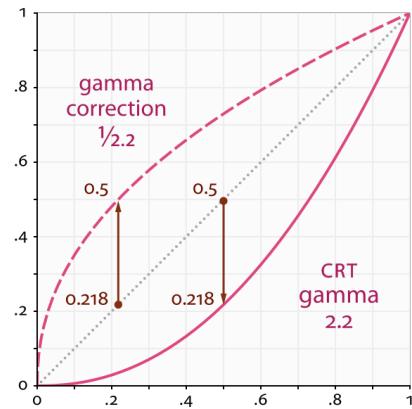


Figura 20.7: Correção de gama para um CRT (Fonte Wikimedia Foundation)



Figura 20.8: Figura com gama de 2.2 ou de 0.5.

Para além da compensação de não-linearidades dos dispositivos, este tipo de operações também se usa para melhorar imagens obtidas em condições de iluminação deficiente, por exemplo. A Figura 20.8 apresenta duas imagens com compensações de gama distintas.

Este ajuste pode ser obtido aplicando-se uma fórmula $py = y^g * f$ no modo **YCbCr**, onde y é o valor do canal **Y**, g é o factor de correção gamma e f é um factor de normalização da intensidade de imagem. O fator de normalização é dado pela fórmula $f = \frac{m}{m^g}$, onde

m representa o valor máximo de intensidade (frequentemente 255).

Exercício 20.13

Construa uma função que aplique uma correção de gama a uma imagem.

Controlo de Saturação

A saturação de uma imagem refere-se à intensidade das cores apresentadas. Uma imagem muito saturada terá cores vivas enquanto uma imagem pouco saturada terá tons muito suaves. Uma imagem em tons de cinza não possui qualquer saturação. A Figura 20.9 demonstra duas imagens com saturações bastante distintas.



Figura 20.9: Figura com saturação aumentada por 1.5 ou reduzida para 0.5.

O controlo de saturação também faz uso do modo **YCbCr**, alterando neste caso os canais **Cb** e **Cr**. Estes canais codificam a saturação como a diferença em relação ao seu valor central, 128. Portanto, quando mais distante de 128 for o valor maior será a saturação, enquanto que quanto mais próximo de 128 for o valor, menor será a saturação. Um método comum de controlo de saturação é o apresentado no exemplo que se segue, onde se considera que **p** é um dado píxel da imagem.

```
...
py = p[0]
pb = min(255,int((p[1] - 128) * f) + 128)
pr = min(255,int((p[2] - 128) * f) + 128)
...

```

Exercício 20.14

Construa uma função que aplique uma transformação de saturação na imagem.

Sépia

Um efeito artístico que também se baseia na manipulação de cores é o efeito Sépia, frequentemente utilizado para emular fotografias antigas. Este efeito é simplesmente uma manipulação dos valores **RGB** de cada píxel de acordo com a seguinte fórmula:

$$\begin{aligned} R' &= 0.189R + 0.769G + 0.393B \\ G' &= 0.168R + 0.686G + 0.349B \\ B' &= 0.131R + 0.534G + 0.272B \end{aligned}$$

onde R , G e B são os valores da imagem original e R' , G' e B' são os novos valores.

Este efeito pode servir de base para muitos outros, bastando apenas a manipulação dos coeficientes aplicados em cada multiplicação. O resultado será tal como representado na imagem da esquerda da Figura 20.10. A imagem da direita é semelhante a um efeito chamado de *Lomography* e é obtido pela troca de R' por B' na fórmula anterior.



Figura 20.10: Figura convertida para tons Sépia ou Lomo.

Exercício 20.15

Construa duas funções que implementem os efeitos descritos.

Detecção de Bordas

A detecção de bordas é bastante importante para tarefas como o reconhecimento de texto, mas pode ter outras utilizações no domínio do reconhecimento de padrões. O

funcionamento básico deste processo é o de detectar alterações significativas entre dois píxeis e adicionar um traço. O resultado pode ser uma imagem a duas cores (preto e branco), onde o preto indica as zonas de alto contraste, ou uma imagem semelhante à original mas onde é sobreposta informação. Existem diversos algoritmos de deteção de bordas, que funcionam em diferentes modos de representação de cor e aplicam pesquisas mais ou menos exaustivas.

Um algoritmo simples para este problema consiste em detectar variações através do cálculo da diferença entre píxeis adjacentes. Caso algum vizinho apresente uma diferença superior a um limiar pré-definido, estamos perante uma borda. O resultado será o demonstrado na Figura 20.11.

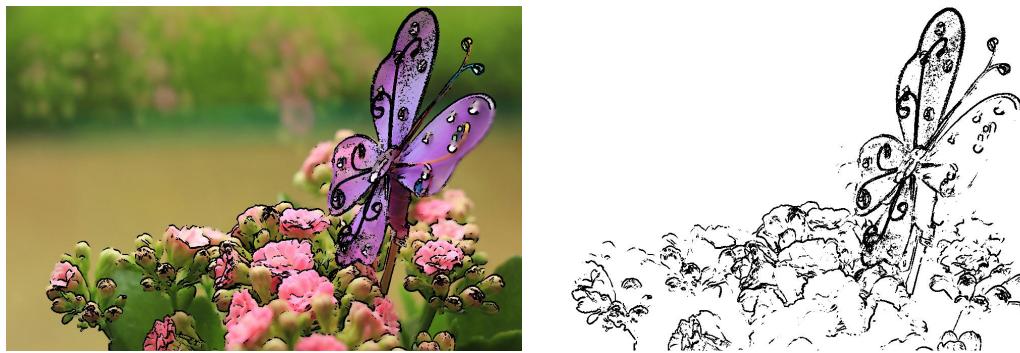


Figura 20.11: Bordas detectadas numa imagem.

A função seguinte implementa este algoritmo. Para cada píxel **p** que esteja na imagem e não seja um limite da imagem, consultam-se todos vizinhos (acima, abaixo, à esquerda e à direita). Caso a diferença para o píxel atual seja superior a **diff**, é devolvido um píxel preto. Caso contrário é devolvido um píxel original ou branco, dependendo da opção especificada em **bw**.

```
def is_edge(im, x,y, diff, bw):
    #Obter o pixel
    p = im.getpixel( (x , y) )
    width, height = im.size

    if x < width-1 and y < height-1 and x > 0 and y > 0:

        #Vizinhos acima e abaixo
        for vx in xrange(-1,1):
            for vy in [-1, 1]:
                px = im.getpixel( (x + vx, y + vy) )
                if abs(p[0]- px[0]) > diff:
                    return (0,128,128)
```

```

#Vizinhos da esquerda e direita
for vx in [-1, 1]:
    px = im.getpixel( (x + vx, y) )
    if abs(p[0]- px[0]) > diff:
        return (0,128,128)

if bw :
    return (255,128,128)
else:
    return p

```

Exercício 20.16

Implemente uma função que calcule as bordas de uma imagem e teste-a para várias imagens fornecidas.

Vignette

O efeito de Vignette é na realidade um defeito das lentes fotográficas em que as bordas das imagens ficam mais escuras que o seu centro. Diferentes lentes apresentarão diferentes níveis de Vignette, sendo típico de equipamento de baixa qualidade, ou mais antigo. Nas lentes modernas este efeito normalmente existe mas é pouco pronunciado. No entanto, o efeito pode ser aplicado a imagens já obtidas, sendo muito comum em algumas comunidades artísticas. A Figura 20.12 apresenta uma imagem sofrendo de Vignette e outra com um Vignette deslocado de forma a realçar o elemento decorativo.

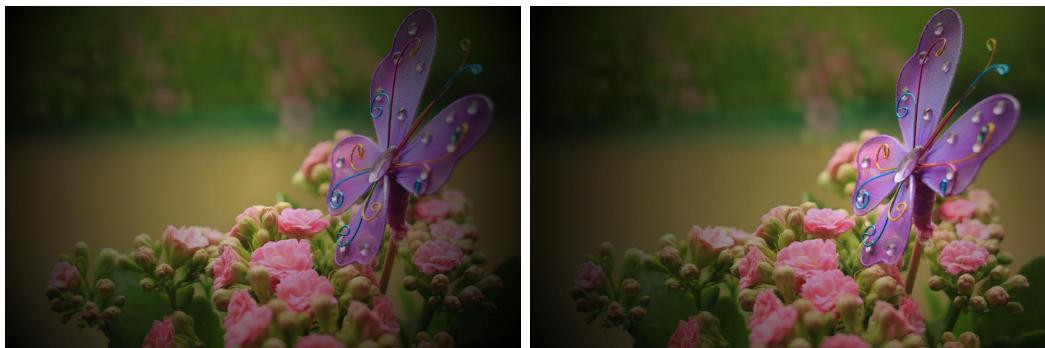


Figura 20.12: Vignette comum (esquerda) e deslocado para a direita (direita)

A implementação deste algoritmo implica que se determine um ponto de referência (normalmente o centro da imagem) e se calcule um factor de atenuação de intensidade (escurecimento) que é tanto maior quanto mais distante um píxel estiver da referência. A intensidade de todos os píxeis é multiplicada por este factor.

A distância entre dois pontos pode ser calculada através da fórmula da distância Euclidiana,

$$distance = \sqrt{(x - x_0)^2 + (y - y_0)^2}. \quad (6)$$

O factor de atenuação irá assim variar entre 0 (nenhum escurecimento) para o ponto de referência (x_0, y_0) e 1 (100%) para os limites da imagem.

```
def get_factor(x, y, xref, yref):
    distance = math.sqrt( pow(x-xref,2) + pow(y-yref,2) )
    distance_to_edge = math.sqrt( pow(xref,2) + pow(yref,2) )

    return 1-(distance/distance_to_edge) #Percentagem
```

Exercício 20.17

Implemente um efeito que aplique *Vignette* a uma imagem.

20.6 Marcação de imagens

Marca de Água

Uma marca de água é uma imagem que é sobreposta a outra imagem, normalmente utilizada para questões de direitos de autor. Para sobrepor uma imagem é necessário somar cada um dos píxeis das 2 imagens, depois de multiplicadas por um factor **f**. Este factor irá indicar o grau de transparência da marca de água. No exemplo que se segue quanto maior for **f**, menos transparente será a marca de água. Os valores **start_x** e **start_y** indicam a posição onde se deve iniciar a colocação da imagem.

```
...
#p1 é um pixel da imagem original
#p2 é um pixel da marca de água
p1 = im1.getpixel( (x+start_x, y+start_y) )
p2 = im2.getpixel( (x,y) )
if(p2[3] == 0):
    continue

r = int(p1[0]*(1-f)+p2[0]*f)
g = int(p1[1]*(1-f)+p2[1]*f)
b = int(p1[2]*(1-f)+p2[2]*f)
...
```

A Figura 20.13 mostra o resultado de uma operação destas, neste caso com **f=0.8**.



Figura 20.13: Fotografia da UA com o símbolo em marca de água.

Exercício 20.18

Construa um programa que, aceitando duas imagens como argumento e um factor de transparência, adicione a segunda à primeira.

Uma alternativa para adicionar uma marca de água quase imperceptível é manipular os bits individuais da imagem de forma a codificar o bit mais significativo da marca de água no bit menos significativo da imagem a marcar. Ou seja, manipular o bit que introduz menos erro na imagem a marcar, codificando o bit da marca de água que possui maior valor. Isto é uma forma de esteganografia: a marca de água é escondida na imagem original de forma imperceptível. Em *Python* o processo é conseguido alterando o exercício anterior para que a transformação aplicada a cada canal seja a seguinte.

```
...
r = (p1[0] & 0b11111110) | (p2[0] >> 7)
b = ...
...
```

A recuperação da marca de água efectua-se processando toda a imagem e promovendo o bit menos significativo a mais significativo, o que se consegue com uma operação de *shift* à esquerda de 7 bits. Tipicamente este bit irá conter ruído, mas nos sítios onde foi codificada a marca de água, será possível identificá-la.

```
...
r = (p1[0] << 7) & 255
...
```

A Figura 20.14 mostra o resultado da imagem com a marca de água e a versão recuperada.

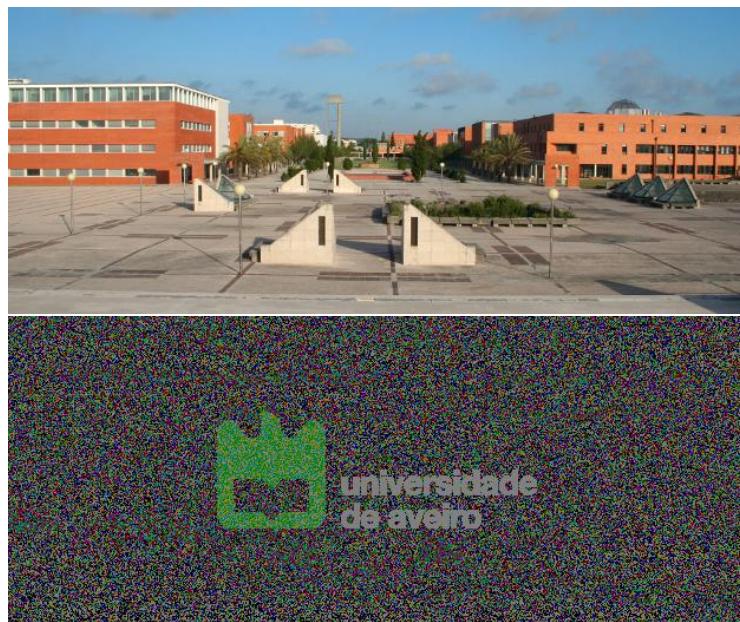


Figura 20.14: Imagem com marca de água usando técnicas de estanografia e marca de água recuperada.

Exercício 20.19

Implemente um programa semelhante ao anterior mas que aplique a marca de água usando técnicas de esteganografia. Experimente processar todos os canais ou apenas alguns. Consegue notar diferenças na imagem original?

Adição de texto

Uma outra forma de marcação de imagens é a sobreposição de textos sobre a imagem. Neste caso, o texto é construído directamente para a imagem através de métodos fornecidos pela biblioteca **Pillow**. O processo implica a selecção de um ficheiro de tipo de letra, um texto, um tamanho de letra, uma cor e a definição de uma posição para colocar o texto.

O exemplo seguinte acrescenta uma mensagem de texto a uma imagem **im**, neste caso a palavra **LabI** com tamanho 40, escrita a branco, na posição $x = 20$, $y = 20$.

```
from PIL import ImageDraw  
...  
im = Image.open('image.jpg')
```

```
draw = ImageDraw.Draw(im)
font = ImageFont.truetype("caminho-para-um-ficheiro.ttf", 40)

draw.text( (20, 20) , "LabI" , (255,255,255) , font=font)
```

De notar que é necessário especificar onde se encontram os tipos de letra. Esta localização irá depender de cada sistema. Nos sistemas *Linux*, podem ser encontrados no directório **/usr/share/fonts/truetype**.

Exercício 20.20

Implemente um programa que permita adicionar mensagens de texto a imagens.

21

TEMA

Microcontroladores

Objetivos:

- Programar uma plataforma com um microcontrolador.
- Configurar e utilizar portos I/O multifunções.
- Utilizar contadores de tempo e ADCs.
- Usar a linguagem C e operações bit-a-bit.

21.1 Introdução

Os microcontroladores são sistemas computacionais completos integrados num único chip. Incluem unidade de processamento, memória (RAM, Read Only Memory (ROM) e flash) e dispositivos de entrada/saída configuráveis (portos I/O digitais, ADC, interfaces de comunicação, etc.). Geralmente funcionam a velocidades muito inferiores e têm muito menos memória do que um computador pessoal, não têm unidades de processamento gráfico nem de vírgula flutuante, mas gastam muito menos energia e custam muito menos. Usam-se tipicamente como *sistemas embutidos* para controlar eletrodomésticos, periféricos de computador (teclados, discos, impressoras) e muitos outros sistemas. Ao contrário de um computador de uso geral, que permite correr múltiplos programas sobre um sistema operativo, um microcontrolador tipicamente corre apenas um programa, com um fim específico.

O Microchip PIC32MX795F512H é um microcontrolador relativamente poderoso, que usaremos neste trabalho. As suas características mais relevantes são as seguintes:⁵⁸

⁵⁸Especificações completas em <http://goo.gl/4LkFps>.



Figura 21.1: O Microchip PIC32MX795F512H e o Intel i5, aproximadamente à mesma escala.
(As imagens pertencem aos seus fabricantes.)

- CPU de 32 bits MIPS M4K com relógio até 80Mhz (40MHz no DETPIC32).
- 128KB de RAM.
- 512KB de Flash.
- Interface USB 2.0.
- Interface Ethernet 10/100.
- 2 Interfaces Controller Area Network (CAN).
- ADC de 16 canais com 10 bits de resolução a 1 Msps.⁵⁹
- RTC interno.

Por contraste, um computador pessoal atual pode incluir um CPU Intel i5 de 64 bits a 3 GHz, ter 4 ou 8 GiB de RAM e um disco rígido com 1 TB de capacidade. É cerca de cem vezes mais rápido e tem capacidades de memória muitos milhares de vezes superiores.

A Figura 22.1 mostra um microcontrolador Microchip PIC32MX795F512H e um processador Intel i5 lado-a-lado. O PIC32 mede $10 \times 10\text{mm}$, enquanto o Intel i5 mede $37.5 \times 37.5\text{mm}$, ou seja, é cerca de 14 vezes maior e não é um sistema completo.

Atualmente existem sistemas integrados num chip (System on a chip (SoC)) mais poderosos, com capacidades gráficas e de processamento avançadas, que se usam em telemóveis, televisões inteligentes, consolas de jogos, etc. Tipicamente correm sistemas operativos completos e múltiplas aplicações. Não são considerados microcontroladores, mas dada

⁵⁹ Mega samples per second ou milhões de amostras por segundo.

a constante miniaturização e redução de custo, começam a ser uma alternativa aos microcontroladores em certas aplicações. As plataformas Raspberry Pi e BeagleBone usam processadores deste tipo.

Placas de desenvolvimento para microcontroladores

Para desenvolver software para microcontroladores usam-se geralmente placas de desenvolvimento que têm o microcontrolador e algum hardware extra que permite comunicar com o microcontrolador através de uma interface USB ou outra. Essas placas de desenvolvimento são fornecidas pelos próprios fabricantes de microcontroladores ou por terceiros. Um exemplo muito popular é a plataforma Arduino, disponível com diversos microcontroladores da Atmel.

A placa de desenvolvimento DETPIC32 (ver Figura 21.2) foi desenhada em 2010 no DETI e tem sido usada em diversas disciplinas e projetos do departamento. Contém um microcontrolador Microchip PIC32MX795F512H, uma interface USB para comunicação e alimentação e dois conectores de 28 pinos que dão acesso a dezenas de portos multifunções (GPIO) do microcontrolador.

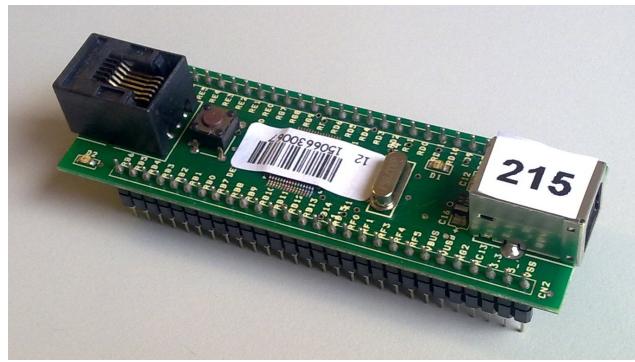


Figura 21.2: Placa de desenvolvimento DETPIC32.

Materiais e método

Este guiaõ foi preparado para ser executado num laboratório equipado com:

- Um computador equipado com cabo USB standard, com sistema operativo Linux e com o software de desenvolvimento instalado (ver abaixo).
- Um osciloscópio.
- Opcionalmente, um voltímetro.

A cada grupo é fornecido um *kit* de exploração contendo:

- Uma placa branca *breadboard*.
- Uma placa de desenvolvimento DETPIC32.
- Um *DIP switch*.
- Um potenciômetro.
- Duas resistências de $10\ k\Omega$ e duas de $470\ \Omega$.⁶⁰
- Vários fios de ligação (8 ou mais).
- Opcionalmente, poderão ser fornecidos componentes adicionais (LEDs, botões, receptores de IR).

Os componentes já vêm montados na placa branca da forma que se vê na Figura 21.3.

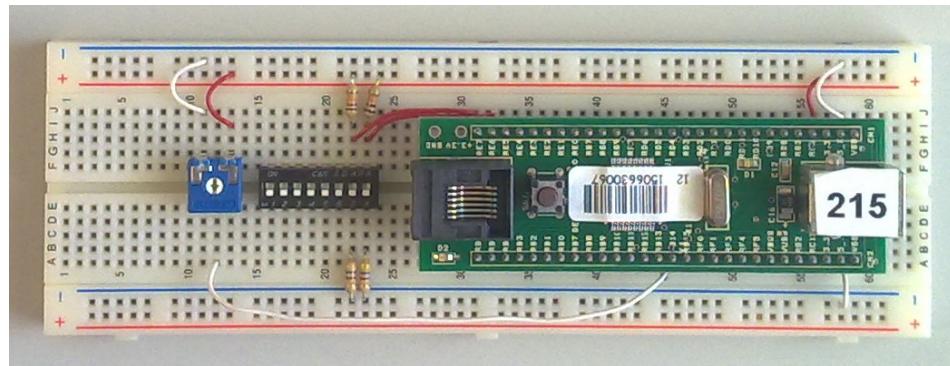


Figura 21.3: Kit fornecido para exploração de microcontroladores.

Exercício 21.1

Verifique se o kit que recebeu está montado de acordo com o da Figura 21.3.

Se detetar alguma falha, avise o professor e corrija.

Os primeiros exercícios propostos podem ser realizados com o computador apenas. Os seguintes requerem a placa DETPIC32 e depois os diversos componentes fornecidos, progressivamente.

⁶⁰Códigos de cores: $10\ k\Omega$ = III, castanho, preto, laranja, dourado; $470\ \Omega$ = III, amarelo, violeta, castanho, dourado.

21.2 A linguagem de programação C

Dadas as limitações de recursos, os microcontroladores não correm interpretadores nem máquinas virtuais. Por isso os seus programas são compostos de código-máquina puro: instruções diretas para a unidade de processamento, codificadas na memória. Para criar esses programas usa-se geralmente

- uma linguagem de baixo nível, i.e. uma linguagem *assembly*, que é traduzida diretamente para código-máquina por um programa chamado de *assembler*; ou
- uma linguagem de alto nível que possa ser compilada para código-máquina dessa arquitetura.

Em qualquer dos casos, a tradução ou compilação é feita num computador externo e apenas o código máquina resultante é transferido e executado no microcontrolador.

As linguagens assembly dão acesso a todas as “nuances” do hardware, mas são específicas para cada arquitetura (não são portáteis), e não têm estruturas de programação abstratas (como funções ou ciclos estruturados), pelo que dificultam o processo de desenvolvimento.

A linguagem *C* é uma linguagem de alto nível, com variáveis, funções, instruções condicionais e de repetição, mas que também disponibiliza operações de “baixo nível” como aceder a posições de memória diretamente. Além disso, existem compiladores de *C* para muitas arquiteturas, o que facilita a portabilidade dos programas. Por estas razões, *C* é uma linguagem muito usada para a programação de microcontroladores.

Este guião irá abordar apenas alguns aspectos desta linguagem, focando-se no mínimo necessário para desenvolver programas para a plataforma DETPIC32.

Do ponto de vista sintático, a linguagem *C* é bastante semelhante à linguagem Java: as instruções condicionais, as instruções de repetição, as declarações de variáveis e de funções são idênticas. Por outro lado, não há classes em *C* e há algumas diferenças nos tipos de dados básicos.

O programa **first.c** (ver abaixo) demonstra o uso de variáveis simples, instruções de entrada/saída e o uso de expressões. Podem identificar-se comentários, a inclusão de uma biblioteca e a declaração de uma função.

```
// A simple program in C, demonstrating arithmetic and bitwise operators.
#include <stdio.h>

int main() {
    printf("A simple program in C.\n");
    unsigned int x, y; // C allows unsigned variables!
    printf("x? ");
    // print a prompt
```

```

scanf("%u", &x); // read an unsigned integer into x
printf("y? ");
scanf("%u", &y);
printf(" x - y = %u\n", x-y); // difference, in decimal
printf(" x = %08X\n", x); // x in hexadecimal (8 digits)
printf(" y = %08X\n", y); // y in hexadecimal (8 digits)
printf(" NOT y = %08X\n", ~y); // bitwise NOT
printf("x AND y = %08X\n", x&y); // bitwise AND
printf(" x OR y = %08X\n", x|y); // bitwise OR
printf("x XOR y = %08X\n", x^y); // bitwise XOR
unsigned int n = y & 0b011111; // extract only 5 lower bits (0 to 31)
printf("x << %2u = %08X\n", n, x<<n); // bitwise shift left
printf("x >> %2u = %08X\n", n, x>>n); // bitwise shift right
return 0;
}

```

Exercício 21.2

Crie um diretório novo e extraia o conteúdo do arquivo **guide-22-skel.zip** lá dentro. Abra e analise o ficheiro **first.c**.

O programa tem vários aspectos interessantes.

- A linguagem C tem o tipo `int` idêntico ao do Java, mas também tem uma versão de inteiro sem sinal, `unsigned int`, que só suporta valores não negativos.
- A função `printf` é bastante semelhante à disponível em Java.
- A função `scanf` permite ler valores do dispositivo de entrada (teclado), tal como um objeto `Scanner` do Java, mas tem uma forma de utilização muito diferente.
- Existe uma série de operadores que fazem manipulações bit-a-bit de valores inteiros e que são muito úteis para a programação de microcontroladores. Estes operadores são análogos aos disponíveis na linguagem Java.

Exercício 21.3

Compile o programa com o comando `gcc -o first first.c`.

Se não tiver erros, execute-o com o comando `./first`. Corra várias vezes, com valores diferentes e tente perceber o funcionamento dos operadores usados.

21.3 Programação de microcontroladores

Até aqui compilámos programas e executámos os ficheiros resultantes no mesmo computador ou noutra com a mesma arquitetura e sistema operativo. Mas também é possível compilar um programa para produzir um ficheiro executável que possa correr noutra processador de arquitetura diferente. É isso que teremos de fazer para compilar programas para um microcontrolador. A esse processo chama-se *compilação cruzada (cross compilation)* e requer um conjunto de ferramentas e bibliotecas especializadas para a arquitetura alvo.

Teste da placa e das ferramentas DETPIC32

As ferramentas de desenvolvimento de microcontroladores PIC32, numa versão simplificada para uso nas aulas do DETI, já estão instaladas nos computadores das salas de aula. (Essas ferramentas também estão disponíveis em <http://sweet.ua.pt/jla/> com instruções para instalação noutras computadoras em Linux.)

Exercício 21.4

Verifique se as ferramentas de desenvolvimento do PIC32 estão instaladas. Devem estar na pasta `/opt/pic32mx/`.

Ligue o DETPIC32 ao computador através do cabo USB. O LED D2 deverá iluminar-se.

Execute o comando `ldpic32 -i`. Quando solicitado, pressione o botão de RESET do DETPIC32.

Deverá obter um resultado semelhante ao seguinte.

```
DETPIC32 Serial Programmer, V0.4
Universidade de Aveiro, DETI
J.L.Azevedo, 2011, 2012
Report bugs to jla@ua.pt

System info:
- BL Version      : DETPIC32 SBL V0.3
- Device ID       : 14 (32MX795F512H)
- Core frequency  : 40 MHz
- Peripheral freq. : 20 MHz
- Flash size       : 524288 Bytes (512 kB)
- RAM size         : 131072 Bytes (128 kB)
- User program size: 5532 Bytes (1.1%)
```

Este resultado mostra informações sobre o hardware e firmware (o *boot loader*) da placa DETPIC32 e permite comprovar que está a funcionar e a comunicar corretamente.

Processo de desenvolvimento

O processo de criação de um programa no microcontrolador implica:

- Editar o programa fonte no computador, usando um qualquer editor de texto.
- Compilar o programa fonte e converter o ficheiro objeto para um formato apropriado, usando o comando **pcompile**.
- Transferir o ficheiro e programar o microcontrolador, com o comando **ldpic32 -w**.
- Executar o programa no microcontrolador e interagir com ele através da porta USB, usando o comando **pterm**. Isso é muito útil para depuração.

O ficheiro **termIO.c** é um exemplo de um programa para a placa DETPIC32. Abra-o num editor de texto e analise-o.

```
// jmr@ua.pt 2016
#include "detpic32.h"      // Library of system calls for DETPIC32
// See: http://sweet.ua.pt/jla/Sw/DETPIC32-system_calls.pdf

int main() {
    printStr("Example using terminal I/O syscalls in DETPIC32.");
    printStr("\nx? ");           // print a string
    unsigned int x = readInt(10); // read unsigned integer in base 10
    printStr("\ny? ");
    unsigned int y = readInt(10);

    // print difference in base 10:
    printStr("\nx-y = "); printInt(x-y, 10);

    // print x, y and (x XOR y) in binary (all 32 bits):
    printStr("\n  x = "); printInt(x, 2 | 32<<16);
    printStr("\n  y = "); printInt(y, 2 | 32<<16);
    printStr("\nx^y = "); printInt(x^y, 2 | 32<<16);
    return 0;
}
```

Repare que as instruções de introdução e impressão de dados são mais primitivas que no programa que correu no computador. A biblioteca **detpic32.h** define apenas operações muito simples, denominadas funções de sistema (*system calls*), para gastar um mínimo

de recursos do microcontrolador. Pode consultar a tabela de syscalls⁶¹ ou o ficheiro **detpic32.h** para conhecer as operações disponíveis.

Exercício 21.5

Execute o comando `pcompile termIO.c` para compilar o programa para o PIC32. Verifique que foram gerados novos ficheiros com o código compilado.

Execute `ldpic32 -w termIO.hex` e pressione o botão de RESET na placa para iniciar a transferência e programar o PIC32. Se não houver erros deve ver o LED D1 a piscar rapidamente.

Execute `pterm` para poder interagir com a placa.

Pressione RESET novamente para executar o programa que transferiu.

Deverá ver um resultado idêntico ao seguinte.

```
DETPIC32 terminal emulator, V0.2 (February, 2012)
...
DETPIC32, Bootloader V0.3
Universidade de Aveiro, DETI
J.L.Azevedo, 2011
```

```
Example using terminal I/O syscalls in DETPIC32.
x?
```

Parabéns! Está a correr o seu primeiro programa no DETPIC32.

Exercício 21.6

Responda ao programa, introduzindo dois valores inteiros, por exemplo 30 e 121. Verifique e interprete os resultados apresentados.

Pode pressionar RESET para voltar a executar o programa.

Quando quiser, introduza **Ctrl-C** para terminar o `pterm`.

⁶¹http://sweet.ua.pt/jla/Sw/DETPIC32-system_calls.pdf

21.4 Periféricos e portos de entrada/saída

Os microcontroladores têm portos que permitem comunicar com circuitos eletrónicos externos. Podem ser configurados como portos de *saída* (*output*) para emitir informação para o exterior, ou como portos de *entrada* (*input*) para obter informação do exterior. Por serem configuráveis para diversas funções, são denominados *general purpose input/output (GPIO) ports*.

Na placa DETPIC32 existe um LED que está ligado através de uma resistência ao porto RD0 do microcontrolador. (Pode consultar o esquema do circuito disponibilizado na página de LabI.) Se configurarmos esse porto como saída, então ele passa a funcionar como uma fonte de tensão (com baixa impedância) que podemos ligar ou desligar através de operações no software: se colocarmos um 1 no bit correspondente, o porto fornece 3.3 V; se colocarmos um 0, o porto fica a 0 V. Quando o porto está “alto”, flui uma corrente através da resistência e do LED, que se ilumina. Quando o porto está “baixo”, não cria diferença de potencial entre os terminais da resistência e portanto não passa qualquer corrente e o LED não emite luz.

O programa `toggleLED.c` demonstra como se configura e como se controla um porto de saída. Analise-o. Repare que:

- O programa principal começa por fazer algumas configurações e depois entra num ciclo infinito. Isto é vulgar em microcontroladores: como são usados tipicamente para um único propósito, só têm de executar sempre a mesma tarefa, repetidamente.
- O registo **TRISDbits** permite configurar a direção (I/O) dos portos RD0 a RD11. (Existem registos análogos para os portos RBx, RCx, etc.)
- Estes registos têm campos de 1 bit correspondentes a cada porto: **TRISDbits.TRISD0** a **TRISDbits.TRISD11**.
- Também se pode aceder ou manipular todos estes bits em grupo através da variável inteira **TRISD** (ou do campo **TRISDbits.w**).
- Inicialmente (ou após um *reset*) todos os portos estão configurados como entradas. É mais seguro assim.
- ATENÇÃO: Só se deve configurar como saídas os portos estritamente necessários. Configurar como saída um porto ligado a uma fonte de tensão (como um dispositivo de entrada) pode destruir esse porto!
- O registo **LATDbits** tem campos de 1 bit (.LATD0 a .LATD11) que permitem alterar o estado de cada porto RDx que esteja configurado como saída. Estes campos também permitem ler o estado atualmente especificado para os portos. (Existem registos análogos para os portos RBx, RCx, etc.)

- Também se pode manipular todos os portos do grupo RD através da variável inteira **LATD** (ou do campo **LATDbits.w**).
- Alterar o estado de (escrever em) portos que estejam configurados como entradas não tem qualquer efeito.

Exercício 21.7

Compile o programa **toggleLED.c** e transfira-o para a placa. Execute o **pterm** e inicie o programa pressionando RESET.

Verifique que o bit 0 do registo de configuração **TRISDbits** está a zero enquanto os 11 bits seguintes têm valor 1, o que indica que RD0 foi configurado como saída enquanto que RD1 a RD11 se mantêm como entradas.

Verifique que RD0 = 0 e que o LED D1 está apagado. Pressione uma tecla. O que acontece ao valor de RD0 e ao LED? Repita algumas vezes.

A placa DETPIC32 disponibiliza a maioria dos portos do PIC32 através dos conetores CN1 e CN2. Identifique os pinos Vss (correspondente à “massa” ou seja a 0 V), 3.3 (também designado Vdd, correspondente à tensão de alimentação do microcontrolador) e RD0. Verifique como estão ligados à placa branca.

Exercício 21.8

Use o osciloscópio (ou um voltímetro) para medir a tensão no pino Vss. Depois meça a tensão no pino RD0 e observe o estado do LED.

Sugere-se que fixe o terminal negativo da ponta de prova à pata de uma das resistências ligadas ao carril azul (-) da placa branca. Com o terminal positivo da ponta de prova deve agarrar a extremidade de um fio e espertar a outra extremidade no orifício a medir. (Pode retirar o fio comprido ligado ao potenciómetro e usá-lo para as medidas. Só será necessário repô-lo para os últimos exercícios.)

Pressione uma tecla e volte a medir RD0.

21.5 Contadores e tempo

É usual os microcontroladores terem registos contadores que são incrementados regularmente de forma a contabilizarem a passagem do tempo. Estes dispositivos são chamados de *timers*. No PIC32, um desses timers tem 32 bits e é incrementado a cada dois ciclos de relógio do CPU. Na placa DETPIC32 o relógio do sistema está configurado a 40 Mhz, pelo que o *core timer* é atualizado 20 milhões de vezes por segundo. É possível ler o valor

desse contador com a *syscall* `readCoreTimer` e repô-lo a zero com `resetCoreTimer`.

O programa `blinkLED.c` usa o *core timer* para fazer piscar o LED a intervalos regulares. Repare que:

- Com exceção da mensagem incial, o programa não comunica com o terminal.
- Na rotina de configuração, o timer é iniciado a zero.
- O símbolo `PBCLK` está definido no ficheiro `detpic32.h` e corresponde à frequência de atualização do *core timer*. Ou seja, num segundo, o timer é incrementado `PBCLK` vezes. Por essa razão definimos novos símbolos (também chamados de *macros*) para representarem o número de ciclos num segundo, num milisegundo e num microsegundo. Esses símbolos facilitam a compreensão dos valores temporais escritos no programa.
- No ciclo principal, a inversão do valor RD0 está escrita numa forma mais compacta, usando a variável `LATD` e a atribuição `^=` que aplica um XOR bit-a-bit ao valor da variável.
- A função `wait` serve para esperar um determinado tempo. Está feita à custa de um ciclo que monitoriza constantemente o valor do timer. A isto chama-se *busy-waiting* porque o processador está continuamente a verificar se atingiu a condição esperada.⁶² Outra solução seria adormecer o processador para poupar energia e programar o timer para gerar uma *interrupção* para o acordar no momento certo. Seria uma solução muito mais eficiente, mas envolve conceitos avançados que serão estudados noutras disciplinas.
- Quando termina, a função `wait` regista o valor atual do timer numa variável global para servir de referência para a próxima espera. Deste modo a função consegue compensar quaisquer flutuações de tempo das restantes instruções do ciclo principal, resultando num ritmo bastante rigoroso.

Exercício 21.9

Compile o programa `blinkLED.c` e transfira-o para a placa. Inicie o programa pressionando RESET.

Verifique que o LED pisca regularmente. Estime o período cronometrando 10 ou mais ciclos ON-OFF completos.

⁶²Como crianças num viagem: *Já chegámos? Já chegámos? ...*

Exercício 21.10

Altere o programa para, no início, pedir ao utilizador o período desejado (em *ms*). Compile, transfira e teste o programa com vários períodos.

Exercício 21.11

Corra o programa e especifique um período de 1 *ms*. Com o osciloscópio, meça o período do sinal RD0. Se não for o esperado, corrija o programa.

Exercício 21.12

Especifique um período de 10 *ms*. Consegue ver o LED a piscar? Será problema do LED ou da sua vista? Desligue da placa ambos os terminais da ponta de prova. Segure bem na placa e faça um movimento oscilatório suave, como se estivesse a dizer adeus. Consegue ver um traço intermitente?

À percentagem de tempo que um sinal binário periódico está ativo chama-se *duty cycle*. Neste programa, o LED pisca com um *duty cycle* de 50%.

Exercício 21.13

Modifique o programa de forma que o LED esteja aceso só 10% do tempo em cada ciclo, matendo o período original. Guarde o novo programa com o nome **blinkLED1.c**. Verifique o resultado.

Exercício 21.14

Altere o programa para pedir também o *duty cycle* ao utilizador (em %). Guarde o novo programa com o nome **blinkLED2.c**. Transfira e teste o programa. Indique um período de 10 *ms* e *duty cycle* de 10 %. Oscile a placa para ver o resultado.

Exercício 21.15

Volte a colocar a ponta de prova para medir RD0. Selecione um período de 1 ms e experimente vários *duty cycles*.

De cada vez, observe o brilho do LED e confira a forma de onda.

21.6 Entradas digitais

Se configurarmos os portos GPIO como entradas (que é o modo inicial), então eles ficam num estado de alta impedância (muitos $M\Omega$) e por isso praticamente não transferem corrente de ou para o circuito. Isso significa que o circuito externo pode aplicar uma tensão alta (perto de 3.3 V) ou baixa (perto de 0 V) e esse sinal elétrico externo será traduzido respetivamente num bit a 1 ou a 0 num certo registo do microcontrolador.

Na placa branca fornecida está montado um DIP switch. Dois dos seus interruptores estão ligados aos portos RE6 e RE7 do PIC32. Cada interruptor está montado em série com duas resistências: uma de $470\ \Omega$ ligada à massa e outra de $10\ k\Omega$ ligada a Vdd. Quando o interruptor está aberto (OFF), a resistência de $10\ k\Omega$ “puxa” a tensão no porto até Vdd. Quando está fechado (ON), passa corrente nas resistências e a tensão cai para $V = \frac{470}{10470}Vdd \approx 0.15\ V$. Apesar de não ser nulo, este valor é suficientemente baixo para ser considerado um 0 pela lógica digital do microcontrolador. A resistência de $470\ \Omega$ foi incluída apenas como medida de segurança: mesmo que o porto seja configurado por engano como saída, a corrente debitada será no máximo $I = Vdd/470 \approx 7\ mA$, que está perfeitamente dentro dos limites.

Exercício 21.16

Coloque os interruptores na posição OFF (interruptor aberto). Com o osciloscópio (ou um voltímetro) meça a tensão nos terminais dos interruptores e registe.

Comute para a posição ON (interruptor fechado). Repita as medições.

Quando se fecha um interruptor (OFF → ON), a tensão à entrada do porto deve variar de um valor alto (Vdd) para um valor baixo (perto de 0). Deste modo, podemos usar o DIP switch como dispositivo de entrada de dados para o microcontrolador.

O programa `readDigital.c` demonstra como se configuram e como se utilizam portos de entrada digitais. Repare que:

- O operador OR da instrução `TRISE = TRISE | 0xC0`; força os bits 6 e 7 da variável a ficar com o valor 1, mas não modifica o estado dos outros bits. Desta forma, garantimos que os portos RE6 e RE7 ficam configurados como entradas sem estragar

configurações anteriores dos outros portos REx. (Neste caso era desnecessário.)

- O registo **PORTEbits** tem campos de 1 bit (.RE0 a .RE7) que permitem ler o estado (binário) de cada porto REx. (Existem registos análogos para os portos RBx, RCx, etc.)
- Também se pode ler simultaneamente todos os portos do mesmo grupo, usando a variável inteira correspondente (**PORTE**).
- Na expressão `(PORTE & 0xC0) >> 6`, o operador AND isola os bits 6 e 7, anulando os restantes, e o *shift* desloca-os para as posições menos significativas. O valor resultante será 0 = 00₂, 1 = 01₂, 2 = 10₂ ou 3 = 11₂.
- Esse valor é usado para selecionar uma palavra na tabela **messages**.

Exercício 21.17

Compile, transfira e teste o programa **readDigital.c**. O valor lido dos dois portos é impresso continuamente no terminal e o LED deve mostrar o resultado da disjunção exclusiva (XOR) dos dois valores lógicos.

Experimente comutar os interruptores e verifique o resultado.

Exercício 21.18

Altere o programa **blinkLED.c** do exercício 14 para que o período seja escolhido usando o DIP switch. Guarde o novo programa com o nome **blinkLED3.c**.

O *duty cycle* deve ser lido apenas no arranque e depois o programa deve selecionar um período de 1, 10, 100 ou 1000 ms consoante o estado dos interruptores. Sempre que mudar os interruptores, o período deve mudar também.

21.7 Entradas analógicas

Na placa branca está montado um *potenciómetro* com as extremidades ligadas a $Vdd = 3.3\text{ V}$ e $Vss = 0\text{ V}$. Um potenciómetro tem uma pista resistiva e um contacto que pode deslizar ao longo dessa pista. Quando o contacto está a meio da pista, há 50% da resistência total em direção a cada uma das extremidades, e no terminal intermédio surge uma tensão $V = 0.5Vdd$. Quando se desloca o contacto em direção a Vss , a tensão

baixa, quando se desloca em direção a Vdd , a tensão sobe.

Exercício 21.19

Com o osciloscópio (ou um voltímetro) meça a tensão no terminal intermédio do potenciômetro.

Com uma ferramenta apropriada, varie a posição do contacto. Verifique o valor medido. Qual o valor mínimo? E o máximo?

O PIC32 inclui um conversor analógico-digital com 10 bits de resolução ligado a um multiplexer analógico de 16 entradas. Estas 16 entradas analógicas (AN0 a AN15) coincidem fisicamente com os portos RB0 a RB15. Qualquer destes portos pode ser configurado como entrada analógica ou como entrada digital, que é a configuração inicial normal. Podemos usar um desses portos para medir a tensão no potenciômetro e assim saber a sua posição.

A configuração de entradas analógicas é muito versátil, permitindo vários modos de funcionamento, múltiplos canais, etc. Em consequência, é preciso ajustar um grande número de parâmetros distribuídos por diversos campos de vários registos. Para evitar os detalhes complicados, neste guião vamos usar apenas um modo muito simples para adquirir assincronamente amostras de um único canal analógico.

O programa `readAnalog.c` mostra como se configura e se lê valores de uma entrada analógica. Repare que:

- A rotina `setup` invoca a função `adc_init`, que configura a ADC para ler um único porto ANx.
- A rotina `adc_sample` é invocada no ciclo principal para iniciar uma conversão e esperar pelo resultado.
- O programa imprime continuamente o valor lido. Se detetar uma tecla, interrompe a listagem até ser pressionada a tecla ENTER.

Exercício 21.20

Compile o programa `readAnalog.c`, transfira para a placa e execute-o.

Experimente variar o potenciômetro entre os dois limites. Qual o valor mínimo reportado? E o máximo?

Exercício 21.21

Altere o programa `blinkLED3.c` do exercício 18 para que o *duty cycle* seja escolhido usando o potenciômetro. Guarde o novo programa com o nome `blinkLED4.c`.

Deixa de ser necessário ler dados no arranque do programa. Deve simplesmente amostrar o valor do potenciômetro no ciclo principal e ajustar o *duty cycle* em função do valor lido.

Teste o programa, experimentando variações no potenciômetro e alterações do DIP switch.

Agradecimentos

Agradecemos ao Sr. António Pereira pela preparação dos kits; ao Prof. Tomás Oliveira e Silva pelas sugestões e ajuda na depuração de erros de configuração; e ao Prof. José Luís Azevedo pela preparação e disponibilização de várias placas DETPIC32 para esta aula, pelas sugestões e pela revisão crítica do texto.

Micro-controladores e a plataforma MicroRato

Objetivos:

- Conhecer o modelo de Programação do MicroRato
- Conhecer as incertezas associadas a sensores analógicos
- Algoritmos para evitar obstáculos
- Algoritmos para seguir linhas

22.1 Introdução

O MicroRato é um sistema robótico desenvolvido na Universidade de Aveiro, tendo o mesmo nome da competição de robótica que o Departamento de Electrónica, Telecomunicações e Informática criou em 1995. O objetivo do concurso é o de atingir o centro de um labirinto, recorrendo apenas às capacidades sensoriais e de processamento presentes nos pequenos robots. A Figura 22.1 representa parte desse labirinto, exatamente no final da prova.

As plataformas robóticas MicroRato são compostas por um conjunto de sensores e atuadores, podendo ou não ter um sensor de farol. Estes sensores são descritos nas secções seguintes.

Todo o processamento é realizado por um micro-controlador PIC32MX795F512H⁶³. De notar que não se trata de um microprocessador de uso genérico como os encontrados

⁶³Especificações em <http://goo.gl/4LkFps>

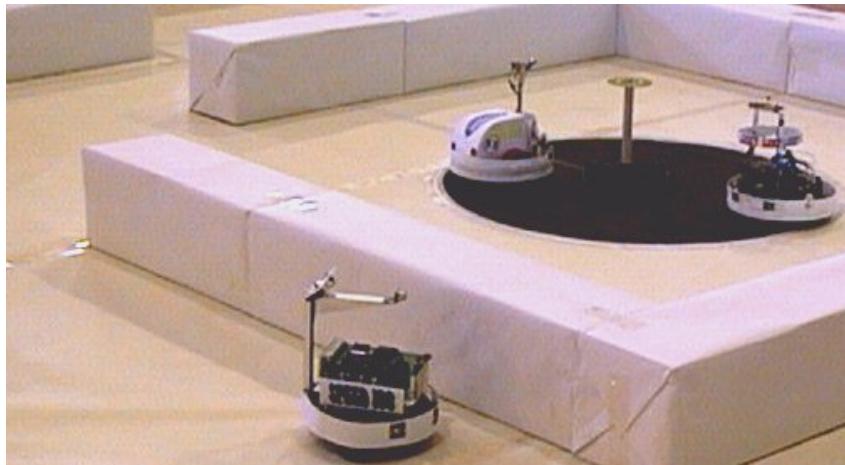


Figura 22.1: Final de uma prova do Micro Rato

nos computadores portáteis. Tanto os micro-controladores como os microprocessadores processam instruções que implementam um dado algoritmo que o programador criou e são muito semelhantes. No entanto existem diferenças muito importantes entre eles.

Os microprocessadores são compostos por um ou mais núcleos e eventualmente alguma memória *cache* interna ou um controlador de memória. Em casos específicos podem incluir um processador gráfico. Estes processadores não possuem qualquer RAM ou ROM, mas são integrados em sistemas onde o designer tem a responsabilidade de desenhar uma placa mãe que permita acesso a estes recursos. O resultado é um computador comum como os dos laboratórios ou um computador portátil.

Um micro-controlador também possui um processador, mas possui igualmente RAM, ROM e mesmo outros sistemas eletrónicos, tudo no mesmo circuito integrado. Um exemplo de um sistemas eletrónicos frequentemente integrados nos micro-controladores são memórias flash ou Static Random Access Memory (SRAM), ADC, interfaces Serial Peripheral Interface (SPI), CAN ou Inter-Integrated Circuit (I²C). Nada disto existe num microprocessador comum. Além disto, os micro-controladores são desenhados para tarefas mais simples, baseadas em entradas e saídas de dados. Por exemplo para processar dados de teclados, controlar máquinas de lavar ou outros eletrodomésticos. Estes sistemas necessitam de uma quantidade de recursos muito pequena e tudo pode ser incluído no mesmo circuito integrado. Um exemplo de um sistema muito popular construído com micro-controladores é o Arduino, que utiliza vários micro-controladores, sendo o mais famoso o ATMEGA328P a 8 MHz ou 16 MHz.

Este micro-controlador possui características específicas que estão relacionadas com os números de definem o seu modelo. Por isso o modelo é complexo, ao contrário dos microprocessadores comercializados como os Intel i5. A Figura 22.1 apresenta ambos.

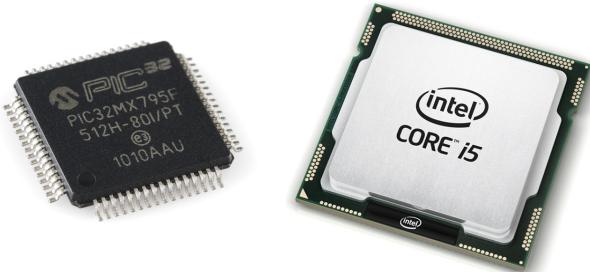


Figura 22.2: O Microchip PIC32MX795F512H e o Intel i5. Imagens pertencem aos seus fabricantes.

De notar que o Intel i5 mede 37.5 mm, enquanto o PIC32 mede 10 mm de lado, sendo portanto 7.5 vezes mais pequeno.

Neste caso as características relevantes deste micro-controlador são as seguintes:

- CPU 32bits MIPS M4K a 80Mhz
- Interface USB 2.0
- Interface Ethernet 10/100
- 2 Interfaces CAN
- 512KB de Flash
- 128KB de RAM
- ADC de 16 canais com 10 bits de resolução a 1MSps⁶⁴
- RTC Interno

Uma diferença bastante grande é a velocidade do CPU (80 MHz), a quantidade de RAM (128 KiB) e a quantidade de memória flash (512 KiB). Num computador portátil o CPU funciona a vários GHz, existem vários GiB de RAM e temos discos rígidos com vários TiB de capacidade.

Por este motivo o modo de funcionamento e desenvolvimento para estes sistemas é bastante diferente do encontrado nos microprocessadores.

⁶⁴*Mega Samples Per Second* ou Milhões de Leituras por Segundo.

22.2 Linguagem C

No caso destes micro-controladores, a linguagem *Python* é deveras desadequada por o sistema não tem capacidade de armazenamento ou processamento compatíveis. Pode experimentar criar no seu computador um programa em *Python* muito simples (um *Hello World*), podendo ver que a quantidade de RAM necessária para a execução deste programa será bastante superior à disponível pelo micro-controlador.

Por estes motivos os micro-controladores são tipicamente programados usando linguagens de mais baixo nível como as linguagens Assembler e *C*. Nesta aula vamos utilizar *C* visto que a linguagem Assembler é deveras mais complexa para os problemas que tentaremos resolver.

A linguagem *C* não é interpretada como a *Python* e os programas resultantes não são executados dentro de uma máquina virtual, como a linguagem *Java*. Este guia não irá abordar os detalhes da linguagem, focando-se no mínimo necessário para implementar um programa para a plataforma MicroRato.

Do ponto de vista sintático a linguagem *C* é bastante semelhante à linguagem *Java*, sendo que os ciclos, condições e declarações de variáveis ou funções são bastante semelhantes. O código seguinte implementa o típico *Hello World*, podendo identificar-se comentários, declaração de uma função com argumentos, a inclusão de bibliotecas, assim como impressão de texto para a consola.

```
/*
 * Ficheiro hello.c
 */
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char* argv[]){
    printf("Hello World!\n");

    return 0;
}
```

No exemplo o carácter * é um conceito novo, representando um ponteiro. Ou seja variáveis que em vez de possuírem um conteúdo, possuem o endereço desse conteúdo. No caso em questão a variável `argv` é um `Array` para ponteiros de `char`. Isto acontece porque na linguagem *C* não existe o tipo `String`, sendo que um texto é representado por um tipo `char *`. Portanto, `argv` contém um `Array` de `Strings` que são os argumentos passados

ao programa.

Exercício 22.1

Estas instruções podem ser convertidas num programa e depois executadas através dos comandos `gcc -o hello hello.c` e depois `./hello`.

Teste a compilação e execução deste pequeno programa.

A sintaxe dos ciclos `for`, `do` ou `while`, condições como `if` é exatamente igual à linguagem Java, tal como apresentado no pequeno programa seguinte. Este programa tem uma variável inteira `i` que toma valores de 0 a 9, sendo imprimido o valor do fatorial para cada um dos valores. O objetivo é demonstrar algumas estruturas básicas da linguagem C.

```
#include <stdlib.h>
#include <stdio.h>

int factorial(n){
    if(n == 0)
        return 1;

    return n*factorial(n-1);
}

int main(int argc, char* argv[]){
    int i;
    for(i=0; i < 10; i++){
        printf("Factorial %d = %d\n", i, factorial(i));
    }
}
```

Exercício 22.2

Crie um ficheiro chamado `factorial.c`, compile-o com o compilador `gcc` e verifique a sua execução.

Os programas criados até agora executam no seu computador mas, usando princípios muito semelhantes, podem ser construídos programas que são executados pelo MicroRato. Neste caso será necessário utilizar um compilador que produza código com as instruções. Este compilador encontra-se na página da disciplina, devendo ser obtido e descompactado

para o diretório `/opt/pic32mx` de forma a que existam diretórios `/opt/pic32mx/bin`, `/opt/pic32mx/lib`, `/opt/pic32mx/include` e por aí fora.

O processo de criação de um programa implica a compilação, conversão do programa para um formato capaz de ser transferido e depois a programação do micro-controlador. A compilação faz-se com o programa `pic32-gcc`, a conversão para o formato hexadecimal para transferência faz-se com o programa `pic32-bin2hex`. Finalmente a transferência faz-se com o programa `ldpic32`. Todos estes programas acompanham o compilador. Após a execução, o comando `pterm` criar um terminal de texto para interagir com o programa. A título de exemplo, realizando um `printf` irá apresentar esse resultado no terminal criado com o `pterm`.

Exercício 22.3

Verifique se o compilador existe no seu sistema e caso não exista obtenha-o da página da disciplina. Nesta mesma página existe um pacote contendo um programa base de exemplo muito simples. O programa é acompanhado de uma `Makefile` que permite compilar o exemplo com o compilador adequado ao MicroRato executando `make hello.hex`.

Exercício 22.4

Verifique que além de compilar, também consegue executar o programa criado no MicroRato. Para isto utilize o comando `make run` e de seguida ligue o MicroRato. Isto irá copiar o código desenvolvido para o micro-controlador. Se o processo falhar, volte a tentar.

O programa irá esperar que pressione o botão de inicio, correspondendo este ao botão preto da plataforma.

Se aparecer a frase "Hello World from MicroRato", parabéns pois conseguiu desenvolver uma aplicação para um micro-controlador.

22.3 Sensores Analógicos e Digitais

A plataforma MicroRato está equipada com vários sensores, sendo que alguns são digitais, enquanto outros são analógicos. Neste contexto entende-se que os sensores digitais fornecem informação já processada com valores discretos, enquanto os sensores analógicos apenas produzem variações de tensão que têm de ser convertidas para valores concretos⁶⁵. O processo de conversão implica adaptação dos valores a intervalos de tensão compatíveis

⁶⁵Embora não seja aplicável a esta placa, também poderiam fornecer variações de corrente, frequência, capacidade, etc...

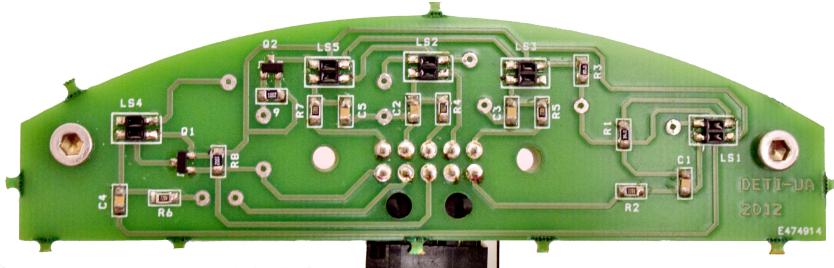


Figura 22.3: Sensores digitais de chão.

com o micro-controlador, sendo que uma ADC irá depois converter os diferentes níveis de tensão em valores inteiros.

No caso da plataforma MicroRato existem sensores digitais que identificam se o utilizador pressionou um dos dois botões, assim como sensores que detetam se o chão é claro ou escuro, fornecendo um valor de 1 caso seja escuro e de 0 caso seja claro. Atenção que detetam claro e escuro e não branco e preto. O sensor emite pulsos de luz e tem um detector de nível de luminosidade. Dependendo se o valor lido é superior ou inferior a um valor pré-configurado, ele irá considerar claro ou escuro. Estes sensores, que são 5, estão localizados à frente em baixo, tal como apresentado na Figura 22.3. É necessário especificar qual a amplificação a aplicar ao detector, o que faz com o que o nível de decisão seja aumentado ou diminuído.

Todos os sensores, mesmo os digitais, possuem ruído. Isto é, além de 0 e 1 os sensores digitais podem oscilar rapidamente entre 0 e 1 devido a ruído. Um caso típico é o de pressionar um botão: até que o contacto seja realmente estável, existem momentos de alguns micro-segundos onde o valor irá oscilar entre 0 e 1. O exemplo seguinte irá imprimir para o terminal o tempo atual e o estado dos botões da plataforma, mas só quando for detetada uma diferença. Um programa deste tipo pode ser utilizado para detetar o ruído de pressionar um botão.

```

#include "mr32.h"
#include <stdlib.h>

int main(void)
{
    initPIC32();
    closedLoopControl( false );
    setVel2(0, 0); //Stop Engines

    int start, oldStart = 2;
    resetCoreTimer();

```

```

while(1)
{
    start = startButton();
    if(start != oldStart)
        printf("%10d %d\n",readCoreTimer()/1000, start);
    oldStart = start;
}
return 0;
}

```

Exercício 22.5

Implemente o exercício anterior e verifique quantas alterações são apresentadas quando se pressiona uma vez o botão *start* (botão preto).

Os sensores de brilho do chão normalmente são tratados com as designações CENTER, NEAR LEFT, NEAR RIGHT, FAR LEFT e FAR RIGHT. Ao se realizar uma leitura, são lidos 5 bits, com valores de 0 ou 1 e que são interpretados de acordo com as designações anteriores. Para aceder aos sensores deve-se invocar a função `readLineSensors(int gain)`, indicando o ganho (amplificação) a aplicar aos sensores, sendo devolvido um inteiro em que cada bit corresponde ao estado de um sensor. O exemplo seguinte demonstra como se podem ser os sensores. Neste exemplo é chamada a função `waitTick40ms()` para introduzir uma cadência de 40 ms na execução de cada ciclo, evitando igualmente algum do ruído dos sensores.

```

#include "mr32.h"
#include <stdlib.h>

#define GROUND_CENTER_BLACK(x) ( (x & 0b000000100) != 0)

int main(void)
{
    initPIC32();
    closedLoopControl( false );
    setVel2(0, 0); //Stop Engines

    while(1)
    {
        printf("Press start to continue\n");
        while(!startButton());
        do
        {
            waitTick40ms();
            int ground = readLineSensors(0);

```

```

        printf("%0x CENTER=%d\n", ground, GROUND_CENTER_BLACK(ground));
    } while(!stopButton());
}
return 0;
}

```

Exercício 22.6

Crie um ficheiro chamado `ground.c` e altere o ficheiro `Makefile` para que compile esse ficheiro (veja a primeira linha). Programe a plataforma e verifique o resultado colocando o MicroRato em sítios onde o chão seja preto ou branco.

Exercício 22.7

O programa anterior possui uma macro que devolve 1 ou 0 dependendo se o sensor de chão central está a ver preto ou branco. Implemente um conjunto de 4 outras macros para os restantes sensores de chão. Cada macro irá corresponder a um teste sobre os restantes 4 bits devolvidos pela função `readLineSensors()`.

Existem igualmente sensores analógicos que são utilizados para medir a distância entre a plataforma e obstáculos. O seu método de funcionamento consiste na emissão de um impulso de luz infra-vermelha, medindo-se o ângulo do sinal refletido. Estes sensores fornecem um valor que, diretamente não tem unidades, mas pode ser convertido para valores de distância através de cálculos simples. De realçar que os sensores analógicos raramente devolver valores precisos, existindo sempre um erro associado aos valores. Isto resulta que se a plataforma estiver parada, com um objeto a uma distância fixa, os valores irão variar naturalmente. Estes sensores também possuem um limite de distância, que corresponde a uma distância entre os 25 cm e os 30 cm. A Figura 22.3 apresenta estes sensores instalados na plataforma MicroRato.

A leitura dos sensores analógicos é efetuada através da função `readAnalogSensors()`, sendo que uma estrutura chamada `analogSensors` é preenchida com os valores lidos. São lidos valores para a distância diretamente em frente, a 45° à direita e a 45° à esquerda. O programa seguinte imprime para o terminal o valor medido pelo sensor.

```

...
while(1)
{
    printf("Press start to continue\n");
    while(!startButton());
    enableObstSens();
}

```

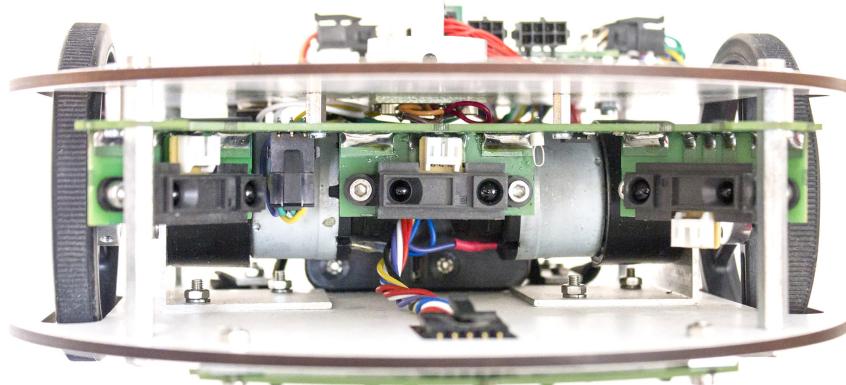


Figura 22.4: Sensores de distância por medição de ângulo da reflexão.

```
do
{
    waitTick40ms();
    readAnalogSensors();
    printf("%d\n", analogSensors.obstSensFront);
} while(!stopButton());
}
return 0;
...
```

Exercício 22.8

Implemente o exercício anterior e programe o MicroRato com o programa resultante. Verifique que valores são devolvidos para várias distâncias e crie uma função `int sensorDistance(int x)` que transforme o valor medido numa distância em centímetros.

A estrutura `analogSensors` possui o seguinte formato:

```
struct
{
    int obstSensRight;
    int obstSensFront;
    int obstSensLeft;
    int an6;
    int an7;
    int batteryVoltage;
};
```

Exercício 22.9

Volte a implementar o exercício anterior de forma a reportar a distância detetada nos 3 sensores. Pode reutilizar a mesma função ou ter de implementar funções independentes (com fórmulas diferentes) para cada sensor de distância.

22.4 Atuadores

Além de ter capacidade de observar o meio à sua volta, a plataforma MicroRato também possui a capacidade de atuar, isto é, de gerar ações. Dois tipos de ações são possíveis: controlar os Light Emitting Diode (LED) que a plataforma possui no topo, ou controlar os motores que estão diretamente ligados a cada roda. As Figuras 22.4 demonstra a localização dos LED e dos motores da plataforma.

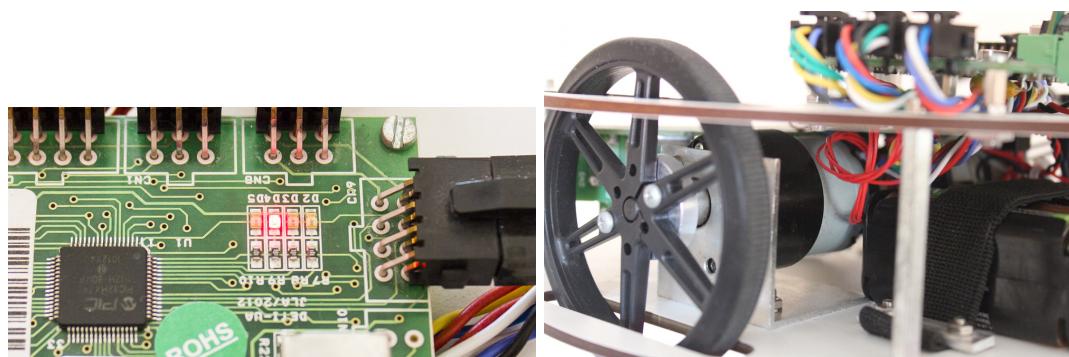


Figura 22.5: LEDs e motores

Os LED são frequentemente utilizados para indicar aos utilizadores da plataforma qual o estado interno de algumas variáveis ou processos. Por exemplo, podem ser utilizados para indicar visualmente se está a ser detetado algum objeto próximo, ou qual a distância ao objeto.

O controlo dos LED é feito através da função `led(int nr, int value)`, sendo que `nr` corresponde ao número do LED, possuindo valores entre 0 e 4. O argumento `value` corresponde ao valor que o LED deve ter, sendo admitidos dois valores: 0 (desligado) e 1 (ligado). O exemplo seguinte demonstra como se podem ativar sequencialmente todos os LED da plataforma.

```

...
while(!startButton());
int i = 0;
do
{
    waitTick40ms();
    led(i, 0);
    i = (i+1) % 4;
    led(i, 1);
    delay(10000);

} while(!stopButton());
...

```

Exercício 22.10

Complete o programa anterior e verifique que os LEDs são ativados de forma sequencial. Implemente depois uma variação que ativa os LEDs de acordo com a distância entre a plataforma e um objeto à sua frente. Uma forma simples é considerar que cada LED corresponde a um intervalo de 5 cm. Se todos os LEDs se encontrarem ativos deverá existir um objeto muito próximo. Se nenhum estiver ativo, o objeto está a mais de 20 cm.

Os motores são sem dúvida os principais atuadores da plataforma, permitindo que ela se mova no labirinto de uma prova. Existem várias versões da plataforma, sendo que a que usamos possui motores Direct Current (DC). Este tipo de motores não permite movimentos precisos pois baseia-se apenas no controlo da corrente útil que lhes é aplicada, através do método Pulse Width Modulation (PWM). Variações no fabrico dos motores ou rodas e peso da estrutura resultam em movimentos com menor previsão. Por outras palavras, não é possível dizer aos motores para moverem a plataforma durante 5 cm, apenas dizer que se movam com potência X , sendo que o programador terá de controlar o tempo durante o qual esta ordem está aplicada. Outras versões da plataforma possuem sensores denominados de *encoders* que contam quantos graus as rodas realmente rodam. Para definir a velocidade dos motores (indirectamente indicando a potência), é utilizada a função `setVel2(int velLeft, int velRight)`. Os argumentos da função indicam qual a percentagem de potência a aplicar, podendo variar entre -100 e 100. Valores negativos indicam movimento para trás e valores positivos indicam movimento para a frente. De notar que só valores absolutos superiores a 40-60 implicam movimento, variando este valor de motor para motor. Mesmo na mesma plataforma, o mesmo valor aplicado aos dois motores pode resultar em velocidades diferentes para cada motor. O excerto seguinte roda a plataforma durante 1 segundo numa direção, invertendo depois a direção de rotação.

```
...
while(!startButton());
do
{
    setVel2(60, -60);
    delay(10000);
    setVel2(-60, 60);
    delay(10000);

} while(!stopButton());
...
```

Exercício 22.11

Simplifique o programa fornecido como exemplo mas varie o argumento da função `delay(int t)`. Esta função cria um período de espera em múltiplos de 100 µs.

Exercício 22.12

Utilizando valores de velocidade apenas suficientes para a plataforma se movimentar, determine quais os valores a aplicar a cada motor de forma a garantir que o MicroRato anda em frente a direito.

22.5 Tarefas

Através da leitura dos sensores analógicos, dos sensores digitais e da ativação dos motores é possível realizar tarefas bastante interessantes. Através de um detetor que indique a direção, o que não se encontra presente nas plataformas que usamos, é possível percorrer labirintos, determinando a direção correta e atingindo o centro. Seguem-se diversas tarefas simples que permitem utilizar os sensores e atuadores da plataforma MicroRato de forma combinada.

Evitar obstáculos

Um dos objetivos de qualquer plataforma móvel é o de evitar obstáculos. Isto consegue-se através da conjugação de leituras contínuas dos sensores de distância, com a ativação dos motores de forma coordenada para evitar qualquer colisão. O algoritmo base considera que o MicroRato deve andar em frente, caso detete um objeto à esquerda, deve rodar para a direita. Se for detetado um objeto à direita, deve rodar para a esquerda. Caso o objeto se encontre imediatamente em frente podem ser tomadas várias ações. O MicroRato pode rodar sempre numa direção, pode rodar numa direção aleatória, pode rodar na

direção inversa da última ordem. Estas ações podem ser isoladas ou combinadas com uma deslocação para trás. O excerto seguinte, após detetar um objeto a menos de 5 cm, roda para a esquerda durante aproximadamente 1 segundo.

```
...
while(!startButton());
do
{
    waitTick40ms();
    readAnalogSensors();
    if(distance(analogSensors.obstSensFront) < 5){
        for(int i = 0;i<25; i++) {
            waitTick40ms()
            setVel(-50, 50)
        }
    }
}
while(!stopButton());
...

```

Exercício 22.13

Adapte o programa apresentado de forma que o, quando for encontrado um obstáculo em frente, o MicroRato rode aproximadamente 90°.

Exercício 22.14

Implemente um programa que permita ao MicroRato seguir objetos. Isto é, manter-se próximo de objetos detetados.

Exercício 22.15

Adapte o programa apresentado de forma que o, quando for encontrado um obstáculo em frente, o MicroRato reduza a sua velocidade até que estabeleça contacto. Depois deverá empurrar lentamente o objeto.

Exercício 22.16

Implemente um programa que permita ao MicroRato evitar obstáculos à direita, esquerda ou à frente. Compare a efetividade das diferentes estratégias possíveis para evitar colisões.

Seguir Linhas

Uma das funções permitidas pelos sensores de brilho é o de orientar o MicroRato com base no brilho do chão. Por exemplo, de forma a evitar zonas que estejam rodeadas por uma linha preta como uma pista, ou seguir linhas. Este último caso é muito utilizado em sistemas de automação industrial onde autómatos seguem linhas pelas instalações fabris para transportar cargas através de um percurso. Neste caso o sensor apenas distingue chão claro e escuro, enquanto sensores mais evoluídos podem distinguir várias cores, permitindo seguir apenas linhas de uma determinada cor.

O processo de seguir ou evitar uma cor é bastante simples, baseando-se na leitura contínua do brilho do chão. Se o objetivo foi seguir uma linha, deve-se manter o sensor central sempre sobre a linha preta. Se isto deixar de acontecer ou se os sensores laterais detetarem a linha, é necessário realizar uma alteração na rota, rodando a plataforma. Se o objetivo for o de evitar linhas pretas, caso um dos sensores detete a linha, a rota tem de ser corrigida, rodando a plataforma na direção oposta.

O excerto seguinte pode ser utilizado para movimentar o robot até que seja encontrado uma linha, parando de seguida.

```
...
while(!startButton());
do
{
    waitTick40ms();
    int ground = readLineSensors(0);
    if(ground != 0)
        setVel2(0, 0);
    else
        setVel2(50, 50);
} while(!stopButton());
...
```

Exercício 22.17

Implemente um programa com o exemplo anterior de forma a que a plataforma MicroRato pare quando encontrar uma linha preta na mesa.

Exercício 22.18

Implemente um programa de forma a que a plataforma evite linhas pretas.

Exercício 22.19

Implemente um programa de forma a que a plataforma siga linhas pretas. Quando encontrar o final da linha, deverá dar meia volta até voltar a detetar a linha.

Exercício 22.20

Implemente um programa que faça o MicroRato capaz de lidar com cruzamentos. Quando atinge um cruzamento, deve sempre virar à direita. Se chegar ao final da linha deve dar meia volta. Um cruzamento deteta-se facilmente pois deverá ser uma das únicas situações onde tanto o sensor central como os laterais estarão sobre uma linha preta.

Evitar obstáculos na linha

Os sensores de brilho e de distância podem ser combinados de forma a implementar algoritmos mais complexos, por exemplo o de seguir uma linha onde existem obstáculos. O algoritmo base é o de seguir uma linha. Quando for encontrado um objeto, o MicroRato deverá contornar o objeto até que a linha volte a ser encontrada. Quando isto acontece, retoma-se o algoritmo de seguir linha.

Exercício 22.21

Implemente um algoritmo que permite ao MicroRato contornar objetos na linha. Utilize uma carteira, caderno ou outro objeto.

Música com o MicroRato

Os sensores de distância também podem ser utilizados para outros fins, tal como o de gerar música, imitando de forma grosseira parte do funcionamento de um instrumento chamado Teremim⁶⁶. Para isto considere que pode gerar frequências diferentes dependendo da distância a que coloca as suas mãos. O excerto seguinte demonstra a parte principal de um programa *Python* que gera notas com duração de 200 ms e com uma frequência lida da porta de série.

```
import serial
import math
import pyaudio
```

⁶⁶<http://pt.wikipedia.org/wiki/Teremim>

```

RATE = 44100
BASE_DURATION = 0.2

stream = player.open(format = player.get_format_from_width(2),
    channels = 1,
    rate = RATE,
    output = True)

sp = serial.Serial('/dev/ttyUSB0', 115200, timeout=1)

while True:
    freq = int(sp.readline())
    data = []
    for i in xrange(0, int(RATE*BASE_DURATION)):
        data.append(amplitude * sin(math.pi*i*freq/RATE))

    wvData = ""
    for v in data:
        wvData += pack('h',v)

    stream.write(data)
...

```

Exercício 22.22

Implemente um programa para executar na plataforma MicroRato de forma devolver uma frequência com base na distância da mão à plataforma. Complete o script de forma a tornar o sistema funcional.

Exercício 22.23

Melhore o sistema anterior de forma a eliminar ruído das leituras lendo vários valores e calculando a média. Implemente igualmente suporte para duas mão, uma controlando a amplitude e outra controlando a frequência. Terá de fazer uso dos vários sensores de distância da plataforma, ou em alternativa, dos sensores de brilho.

Glossário

ADC	Analog to Digital Converter
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AES	Advanced Encryption Standard, cifra simétrica por blocos
BMP	BitMaP image file
CAN	Controller Area Network
CD	Compact Disk
CERN	Conseil Européen pour la Recherche Nucléaire
CGI	Common Gateway Interface
CLI	Command Line Interface
CPU	Central Processing Unit
CRT	Cathode Ray Tube
CSS	Cascading Style Sheets
CSV	Comma Separated Values
DAC	Digital to Analog Converter
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DOM	Document Object Model

RCS	Revision Control System
DTMF	Dual-Tone Multi-Frequency signaling
DVD	Digital Versatile Disk
Exif	Exchangeable image file format
FITS	Flexible Image Transport System
FTP	File Transfer Protocol
GiB	Gibibyte
GIF	Graphics Interchange Format
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Over TLS
I²C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IPC	Inter Process Communication
IPTV	IP TeleVision
ISBN	International Standard Book Number
IPv4	Internet Protocol v4
IPv6	Internet Protocol v6
ISO	Imagen de Arquivo de CD
JPEG	Joint Photographic Experts Group
JS	JavaScript
JSON	JavaScript Object Notation
LED	Light Emitting Diode
LPCM	Linear Pulse Code Modulation

MAC	Media Access Control
MiB	Mebibyte
MD5	Message Digest 5
OAEP	Optimal Asymmetric Encryption Padding
P2P	Peer to Peer
PDF	Portable Document Format
PID	Process IDentifier
PC	Computador Pessoal
PDF	Portable Document Format
PNG	Portable Network Graphics
PWM	Pulse Width Modulation
RAM	Random Access Memory
ROM	Read Only Memory
RSA	Cifra assimétrica, acrônimo dos nomes dos criadores (Rivest, Shamir, Adleman)
RTC	Real Time Clock
SATA	Serial ATA
SCM	Software Configuration Management
SDK	Software Development Kit
SHA-1	Secure Hashing Algorithm (versão 1)
SHA-256	Secure Hashing Algorithm (versão 2 com resultado de 256 bits)
SHA-512	Secure Hashing Algorithm (versão 2 com resultado de 512 bits)
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random Access Memory
SSH	Secure Shell

SVN	Apache Subversion
SoC	System on a chip
TIFF	Tagged Image File Format
TCP	Transmission Control Protocol
TDD	Test Driven Development
TSV	Tabulation Separated Value
UDP	User Datagram Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
USB	Universal Serial Bus
VCS	Version Control System
VDI	VirtualBox Disk Image
VoIP	Voice Over IP
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WAVE	WAVEform audio file format
WWW	World Wide Web
WSGI	Web Server Gateway Interface
WYSIWYM	What You See Is What You Mean
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language
XSPF	XML Shareable Playlist Format

Referências

- [1] Wikimedia, *Latex - Wikibooks, Open books for an open world*, <http://en.wikibooks.org/wiki/LaTeX>, [Online; acedido em 12 de Março de 2018], 2013.
- [2] Stack Exchange, *TeX - LaTeX Stack Exchange*, <http://tex.stackexchange.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [3] V. Cerf, *ASCII format for network interchange*, RFC 20, Internet Engineering Task Force, out. de 1969.
- [4] K. Egevang e P. Francis, *The IP Network Address Translator (NAT)*, RFC 1631 (Informational), Obsoleted by RFC 3022, Internet Engineering Task Force, mai. de 1994.
- [5] J. Postel, *Internet Protocol*, RFC 791 (Standard), Updated by RFC 1349, Internet Engineering Task Force, set. de 1981.
- [6] S. Deering e R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460 (Draft Standard), Updated by RFCs 5095, 5722, 5871, Internet Engineering Task Force, dez. de 1998.
- [7] C. Hornig, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*, RFC 894 (Standard), Internet Engineering Task Force, abr. de 1984.
- [8] R. Droms, *Dynamic Host Configuration Protocol*, RFC 2131 (Draft Standard), Updated by RFCs 3396, 4361, 5494, Internet Engineering Task Force, mar. de 1997.
- [9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach e T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616 (Draft Standard), Updated by RFCs 2817, 5785, 6266, Internet Engineering Task Force, jun. de 1999.
- [10] P. Mockapetris, *Domain names - implementation and specification*, RFC 1035 (Standard), Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, Internet Engineering Task Force, nov. de 1987.
- [11] J. Postel, *Transmission Control Protocol*, RFC 793 (Standard), Updated by RFCs 1122, 3168, 6093, Internet Engineering Task Force, set. de 1981.
- [12] M. Mealling e R. Denenberg, *Report from the Joint W3C/IETF URI Planning Interest Group: Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations*, RFC 3305 (Informational), Internet Engineering Task Force, ago. de 2002.
- [13] L. Andersson e T. Madsen, *Provider Provisioned Virtual Private Network (VPN) Terminology*, RFC 4026 (Informational), Internet Engineering Task Force, mar. de 2005.

- [14] GNU Project (FSF), *GNU Make*, <http://www.gnu.org/software/make/>, [Online; acedido em 12 de Março de 2018], 2013.
- [15] ———, *GNU Automake*, <http://www.gnu.org/software/automake/>, [Online; acedido em 12 de Março de 2018], 2013.
- [16] ———, *GNU Autoconf*, <http://www.gnu.org/software/autoconf/>, [Online; acedido em 12 de Março de 2018], 2013.
- [17] T. Berners-Lee, R. Fielding e L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986 (Standard), Internet Engineering Task Force, jan. de 2005.
- [18] J. Postel e J. Reynolds, *File Transfer Protocol*, RFC 959 (Standard), Updated by RFCs 2228, 2640, 2773, 3659, 5797, Internet Engineering Task Force, out. de 1985.
- [19] E. Rescorla, *HTTP Over TLS*, RFC 2818 (Informational), Updated by RFC 5785, Internet Engineering Task Force, mai. de 2000.
- [20] W3C. (1999). HTML 4.01 Specification, URL: <http://www.w3.org/TR/1999/REC-html401-19991224/>.
- [21] ———, (2001). Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, URL: <http://www.w3.org/TR/2011/REC-CSS2-20110607/>.
- [22] W3Schools, *CSS Reference*, <http://www.w3schools.com/cssref/>, [Online; acedido em 12 de Março de 2018], 2013.
- [23] Twitter, *Twitter Bootstrap*, <http://getbootstrap.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [24] ———, *Twitter Bootstrap CSS*, <http://getbootstrap.com/css/>, [Online; acedido em 12 de Março de 2018], 2013.
- [25] ECMA International, *Standard ECMA-262 – ECMAScript Language Specification*, Padrão, dez. de 1999. URL: <http://www.ecma-international.org/publications/standards/Ecma-262.htm>.
- [26] W3Schools, *JavaScript and HTML DOM Reference*, <http://www.w3schools.com/cssref/>, [Online; acedido em 12 de Março de 2018], 2013.
- [27] Google, *Google Mail*, <http://www.gmail.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [28] ———, *Google Documents*, <http://doc.google.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [29] M. Corporation, *Office - Office.com*, <http://office.microsoft.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [30] H. JS, *Hicharts Demos*, <http://www.highcharts.com/demo/>, [Online; acedido em 12 de Março de 2018], 2013.
- [31] Google, *Google Maps*, <http://maps.google.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [32] OpenStreetMap, *OpenStreetMap*, <http://www.openstreetmap.com>, [Online; acedido em 12 de Março de 2018], 2013.

- [33] V. Agafonkin, *LeafLet - a JavaScript library for mobile-friendly maps*, <http://leafletjs.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [34] D. Inc., *Disqus - The Web's Community of Communities*, <http://www.disqus.com>, [Online; acedido em 12 de Março de 2018], 2013.
- [35] Python Software Foundation, *Python Programming Language*, <http://www.python.org/>, [Online; acedido em 12 de Março de 2018], 2014.
- [36] ———, *Python Documentation*, <http://www.python.org/doc>, [Online; acedido em 12 de Março de 2018], 2014.
- [37] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*, 2nd. New York: Wiley, 1996, ISBN: 0-471-12845-7.
- [38] A. Zúquete, *Segurança em Redes Informáticas: 4ª Edição*. FCA - Editora de Informática, Lda., 2013, ISBN: 978-972-722-767-9.
- [39] R. Housley, *Cryptographic Message Syntax (CMS)*, RFC 5652 (Standard), Internet Engineering Task Force, set. de 2009.
- [40] J. Postel, *User Datagram Protocol*, RFC 768 (Standard), Internet Engineering Task Force, ago. de 1980.
- [41] Y. Shafranovich, *Common Format and MIME Type for Comma-Separated Values (CSV) Files*, RFC 4180 (Informational), Internet Engineering Task Force, out. de 2005.
- [42] E. T. Bray, *The JavaScript Object Notation (JSON) Data Interchange Format*, RFC 7159, Internet Engineering Task Force, mar. de 2014.