



Freshman Berries

Introdução 3

O que é um Raspberry Pi? 3

Usando o Raspberry 4

Sobrevivência no Terminal 4

Aceder Remotamente ao Raspberry Pi 7

Bases de Redes 8

Executar o cliente 9

Algumas Observações Sobre o Código 10

# Introdução

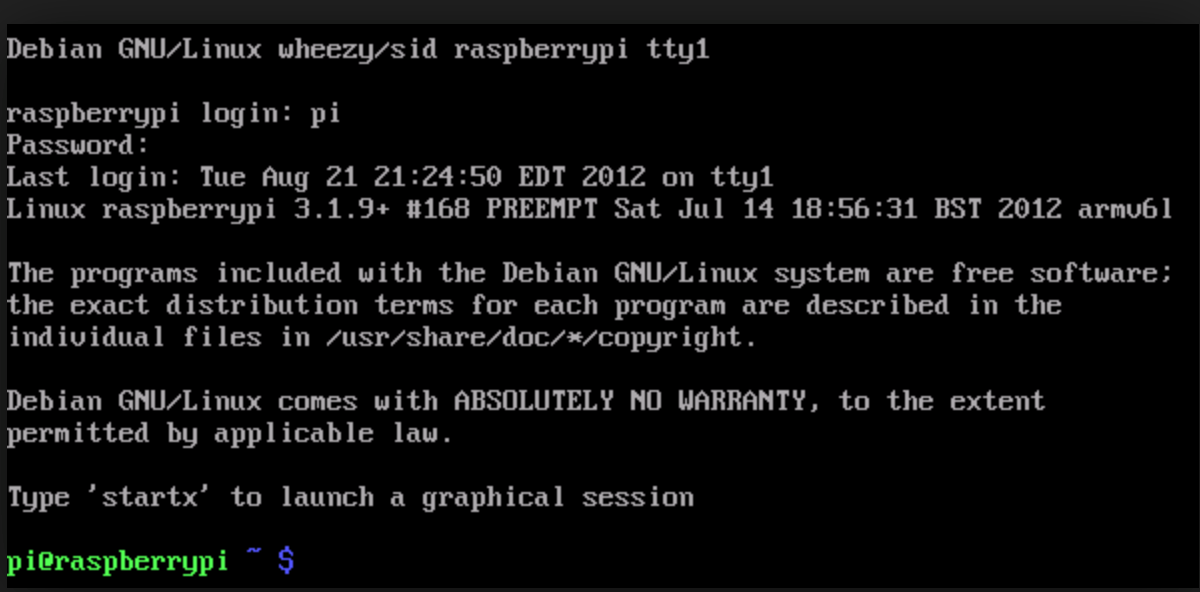
Este ano, o NEETI e o NAPE dão-te a oportunidade de explorares um Raspberry Pi, um computador do tamanho de um cartão de crédito que oferece variadas funcionalidades.

Existem incontáveis projetos e ideias com Raspberry Pi’s que poderás eventualmente implementar mas, de forma a introduzir esta tecnologia, propomos este guião para a construção de uma estrutura cliente-servidor, a qual será detalhada neste documento. O objetivo desta atividade é desenvolver o conhecimento relativo a redes de computadores, especificamente a forma como comunicam entre si, e programar a interação entre eles.

# O que é um Raspberry Pi?

Mas afinal, o que é um Raspberry Pi e por que razão é tão popular?  
O Raspberry Pi, como já referido, é um computador do tamanho de um cartão de crédito que é capaz de utilizar os periféricos tradicionais como um monitor, rato, teclado, entre outros. O aspeto algo minimalista do mesmo deve-se ao facto de que foi idealizado com o menor custo possível, de forma a permitir que pessoas de todas as idades possam explorar o mundo da computação e programação.

# Usando o Raspberry

Todos os computadores têm uma linha de comandos, neste caso denominada shell, que providencia uma interface para o sistema operativo. Esta matéria será explorada profundamente na cadeira de Sistemas Operativos no primeiro semestre do segundo ano mas, para já, o conceito fundamental consiste em saber que o Raspberry Pi, como qualquer computador, tem um sistema operativo e, de forma a interagir com o mesmo, utilizaremos a dita linha de comandos - shell - que tem este aspeto:

Ainda que algo intimidante inicialmente, é bastante eficiente e útil no modo como um programador interage com o sistema operativo, permitindo muito mais funcionalidades do que as apresentadas pela interface gráfica do mesmo. No ponto seguinte serão apresentados alguns dos comandos principais da *shell,* que utilizaremos neste projeto.

# Sobrevivência no Terminal

Nesta secção do guião vou enunciar os comandos básicos do terminal para que seja possível navegar no sistema de ficheiros do raspberry pi. São os que seguem:

## pwd

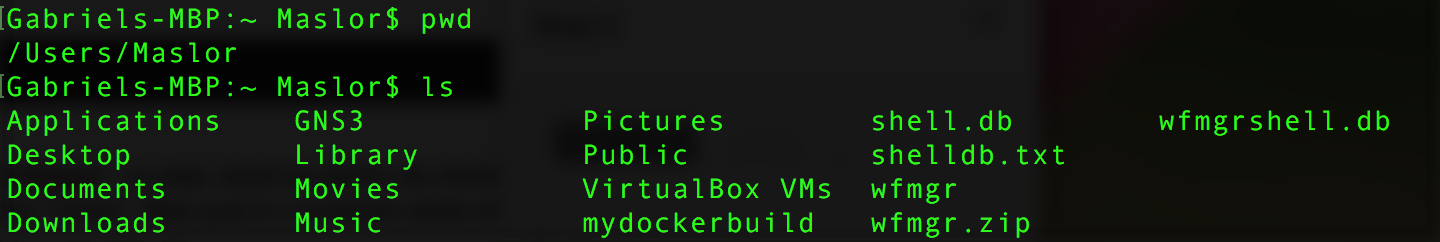
return working directory name

O comando *pwd* devolve o pathname para o diretório onde estamos localizados via terminal. Por exemplo:

Nota que estamos localizados em “/Users/Maslor”, ou seja, estamos dentro do diretório “Maslor” que, por sua vez, fica dentro do do diretório “Users”. Mas qual é o conteúdo deste diretório? Para responder a esta questão, segue o próximo comando - *ls*.

## ls

list directory contents

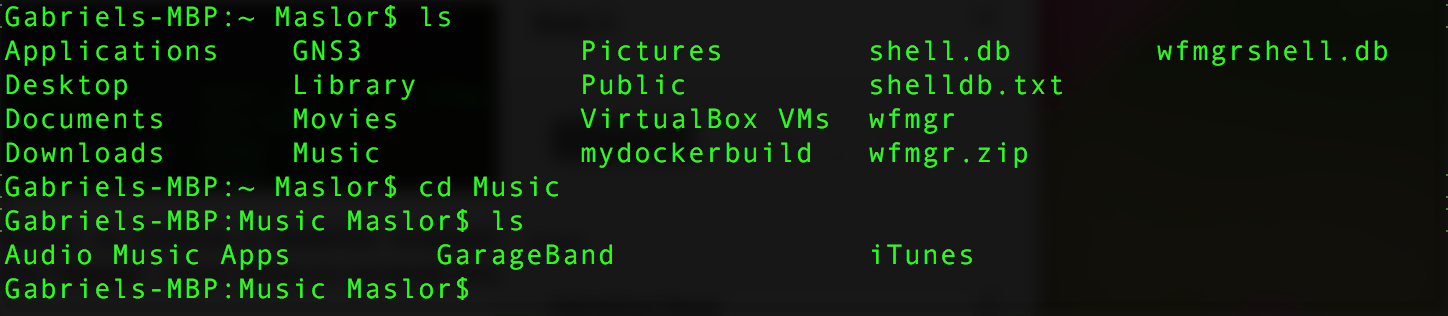
O comando *ls* lista o conteúdo do diretório atual. Seguindo o contexto do exemplo anterior, vemos que:

Repara que através do comando *ls* obtivemos os ficheiros e diretórios localizados dentro do diretório onde estamos localizados no terminal (“Maslor”, como vimos no comando anterior).

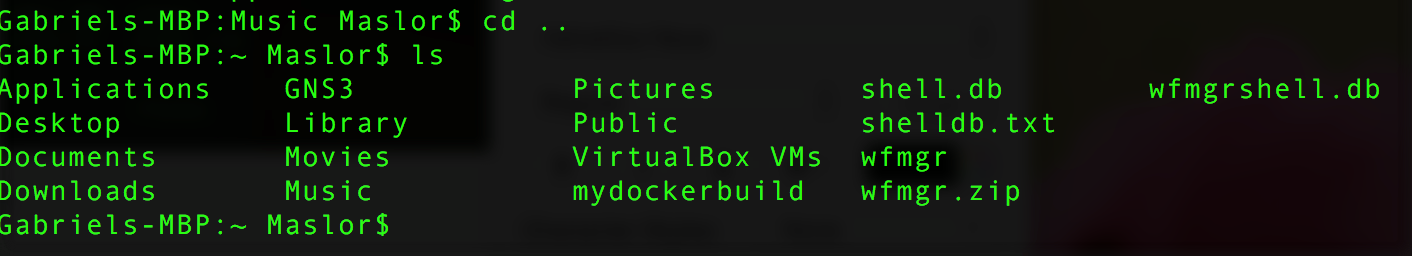
## cd

change directory

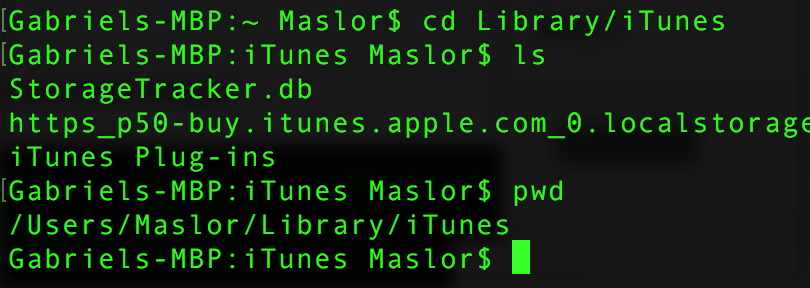
O comando *cd* permite mudar de diretório no terminal. Pode-se mudar de diretório de três maneiras principais:

* + Usando *cd* seguido do nome do diretório para onde navegar:

Neste caso mudámos de diretório para o diretório “Music”, repara que aplicando o *ls*, o conteúdo apresentado é o localizado neste diretório.

* + Usando *cd* seguido de dois pontos ( ‘..’ ):

“*cd ..*” é o comando para mudar de diretório para o diretório anterior. Ao usá-lo quando localizados no diretório “Music”, onde estamos localizados, retornamos ao diretório anterior - “Maslor”.

* + Usando *cd* seguido do pathname de um diretório (não necessariamente localizado no diretório onde estamos localizados):

Neste caso, navegámos diretamente para o diretório “iTunes”, escrevendo o seu pathname (que poderia ter sido obtido através de um *pwd* no mesmo) a seguir ao comando *cd.*

## mkdir

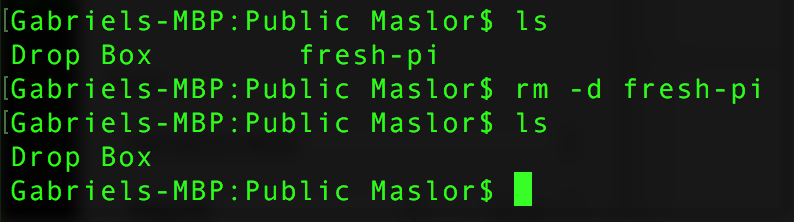
make directory

O comando *mkdir* cria um subdiretório no diretório atual. De forma simples, cria uma pasta no diretório onde nos encontramos. Por exemplo:

Aplicando os comandos anteriores, navegámos até ao diretório “Public” a partir do diretório “Maslor” e criámos a pasta “fresh-pi” no mesmo. Nota a diferença entre o output do comando *ls* antes e depois do *mkdir.*

## rm

remove entry

Este comando permite remover uma entrada, seja ela um ficheiro ou um diretório. Neste exemplo, vamos remover o diretório criado no comando anterior:

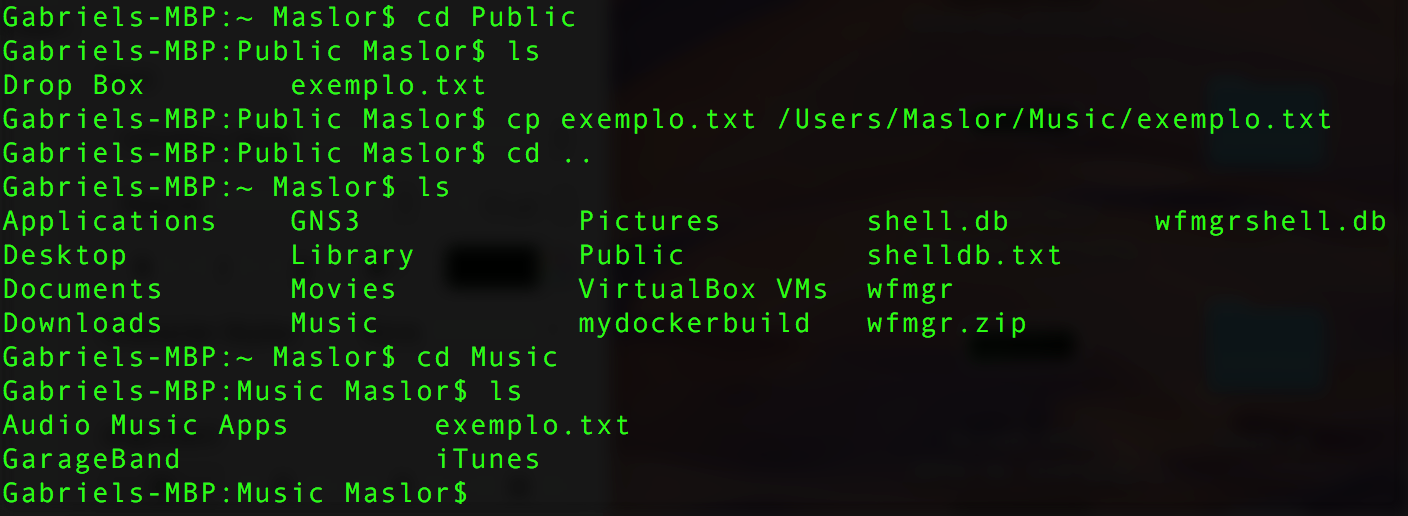
Neste caso acrescentei “-d” para indicar que se trata de um diretório, caso contrário o terminal indicava um erro visto estar à espera de um ficheiro, por defeito.

## cp

copy file

Este comando permite copiar ficheiros de um diretório para outro. A sua sintaxe simplificada é a seguinte:

**cp** *source\_file* *target\_file*

Na figura seguinte será demonstrado o seu funcionamento:

Neste exemplo, copiei o ficheiro “exemplo.txt” para o diretório “Music”, como podes verificar.

# Aceder Remotamente ao Raspberry Pi

De forma a conseguir aceder remotamente ao raspberry pi, vamos utilizar um protocolo de rede criptográfico, o Secure Shell, mais conhecido simplesmente por *SSH*.

Sem entrar em muito detalhe, o *SSH* oferece um canal seguro de comunicação sobre uma arquitetura servidor-cliente insegura, o que torna este protocolo muito usado. É através do *SSH* que vamos aceder remotamente ao Raspberry Pi utilizando um computador pessoal. Para isso, bastam os seguintes passos:

1. Conectar o Raspberry Pi ao computador utilizando um cabo ethernet.
2. Abrir o terminal no computador e introduzir o seguinte comando:

ssh pi@neetiproj.tagus.ist.utl.pt

Esta instrução permite-nos aceder ao raspberry pi com o IP correspondente ao *hostname* [neetiproj.tagus.ist.utl.pt](http://neetiproj.tagus.ist.utl.pt).

3. Fazer login com as credenciais do Raspberry Pi.

A partir daqui, estás a trabalhar no terminal do Raspberry Pi, como pretendido.

# Bases de Redes

Nas disciplinas de *Introdução a Redes de Computadores* e *Arquitetura de Redes* são explorados os fundamentos que permitem entender o funcionamento das redes de computadores. Neste projeto, vamos apenas tratar alguns conceitos inseridos nesta matéria, de uma forma muito superficial.

Nesta secção, serão enunciados estes conceitos.

## Endereço IP

IP address

O endereço IP de um determinado computador é um número que identifica cada computador numa rede. Uma parte do endereço IP remonta à rede onde o computador se encontra, enquanto os bits de menor peso identificam cada computador na referida rede.

Por exemplo, na secção “Aceder Remotamente ao Raspberry Pi”, o comando que estabelecia o *SSH*, referia o IP “192.168.1.3“. De acordo com a forma como configurámos a rede, esta contém todos os endereços de 192.168.1.0 a 192.168.1.255 . Portanto, o nosso Raspberry Pi no caso que referimos tinha o IP 192.168.1.3, ou seja, atribuímos-lhe o número 3, mas se adicionássemos outro Raspberry à rede, poderíamos atribuir-lhe um IP como 192.168.1.5 ou 192.168.1.100, por exemplo. No entanto, um Raspberry Pi cujo IP fosse 192.194.1.3, por exemplo, há não faz parte da rede que estamos a usar.

## Porta

Port

Uma porta é um ponto lógico, no caso das redes, que está sempre associada a um endereço IP e a um protocolo de comunicação como o *TCP* e o *UDP,* completando o endereço de origem ou destino de uma sessão de comunicações.

Números de porta específicos são frequentemente usados para identificar serviços específicos, como, por exemplo, o porto 8080, associado ao protocolo *HTTP (*Hypertext Transfer Protocol).

## TCP vs UDP

Transmission Control Protocol vs User Datagram Protocol

Ambos são protocolos de transporte mas apresentam características diferentes no seu comportamento, especialmente relativamente à sua rapidez e fiabilidade.

De uma forma muito abreviada e simplificada, o *TCP* é um protocolo que estabelece uma ligação com um destinatário e se certifica de que este recebe todas as mensagens enviadas, reenviando as mesmas caso isso não aconteça. O *UDP,* por outro lado, envia mensagens a um destinatário sem garantir que este recebeu as mensagens corretamente. Apesar de, aparentemente, o *UDP* ser muito pior do que o *TCP* depois de ler este parágrafo, a verdade é que, como não aguarda confirmações do destinatário, o *UDP* é mais rápido. Em situações em que o programador não se importa que eventualmente se percam algumas mensagens pelo caminho (porque são enviadas várias iguais e existe redundância, por exemplo) o *UDP* é vantajoso.

Neste projeto vamos utilizar o *TCP*.

## Modelo Cliente-Servidor

Client-server model

Este tipo de arquitetura consistem em duas ou mais entidades computacionais, sendo que uma delas é o servidor e as outras clientes. Na cadeira de *Sistemas Distribuídos* serão tratados modelos com mais do que um servidor mas neste projeto existe apenas um.

A função do servidor é receber os pedidos dos clientes e processá-los, devolvendo os resultados dos mesmos. No caso deste projeto, o servidor contém a estrutura principal da aplicação (toda a gestão das mensagens enviadas/recebidas é feita aqui) e cada cliente que se conecta tem apenas uma interface que lhe permite fazer pedidos e enviar informação ao servidor.

## Sockets

Os sockets serão devidamente tratados na cadeira de *Sistemas Operativos.* Basicamente, um socket é uma ligação que se estabelece entre um par de programas a correr na rede. O socket tem agregado o número do porto para que a camada TCP consiga identificar para que aplicação os dados estão a ser enviados. No caso deste projeto, é criado um socket entre cada cliente e o servidor.

# Executar o cliente

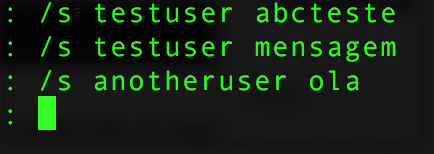
## Pré-requisitos:

* Python3 instalado (esta aplicação foi programada em compatibilidade com a versão 3 do python, pode ter problemas a executar na versão 2).
* (***opcional***) IDE para python como, por exemplo, o WingIDE (já usado em *Fundamentos de Programação*)

Usando um IDE, basta abrir o ficheiro python e executá-lo. Deverá aparecer uma linha como esta no *shell* à espera que se introduza um input:

Os dois comandos base que esta aplicação aceita são os seguintes:

## /s user message

Quando este comando é usado, a mensagem é enviada para o user selecionado. A figura abaixo mostra um exemplo.

*No exemplo acima, foram enviadas duas mensagens para o ‘testuser’ e uma para o ‘anotheruser’.*

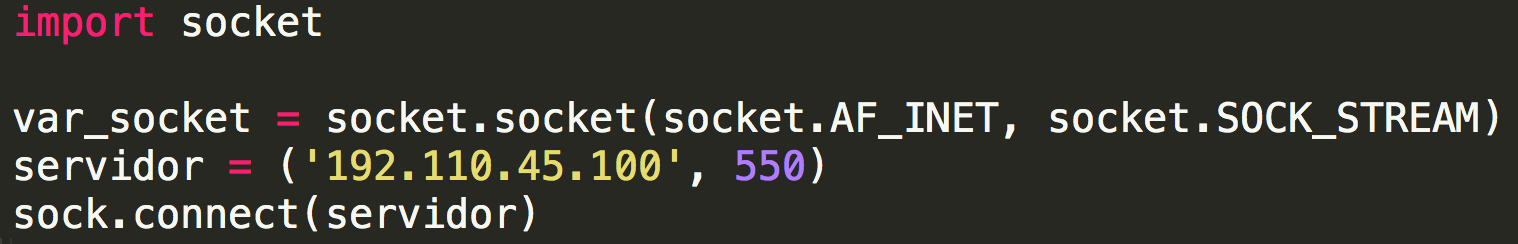
## /r user

Este comando permite ler/listar as mensagens recebidas pelo utilizador selecionado. No seguimento do exemplo anterior, a figura abaixo demonstra o funcionamento deste comando.

# Algumas Observações Sobre o Código

No código utilizado tanto no servidor como no cliente, é importado o módulo *socket* que oferece uma interface para operar sockets.

Para criar um socket, é necessário criar uma instância do mesmo e guardá-la numa variável:

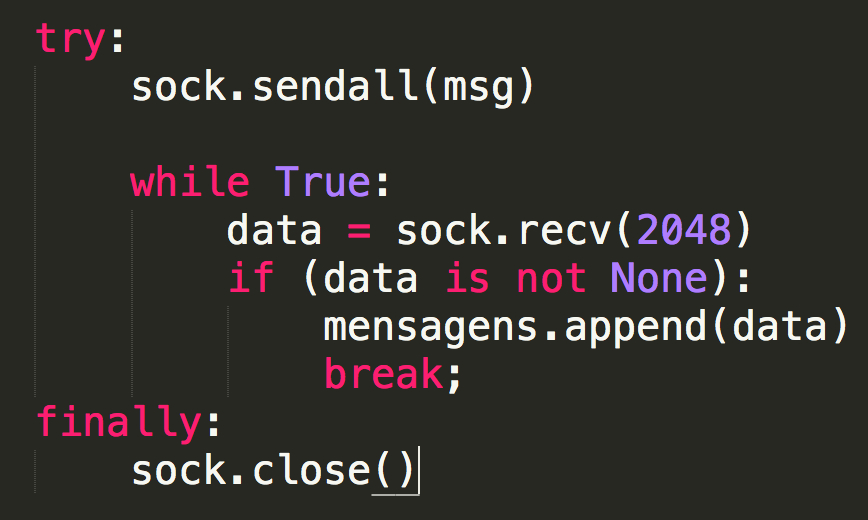


Sempre que desejamos utilizar uma função do módulo *socket* basta escrever *socket.funcao -* em que método é o nome do método em questão. Neste caso são usados as funções *socket* - que cria o socket em si - e *connect -* que liga o socket a um servidor.

Importa também notar que, no caso da função *socket* os argumentos são uma constante que representa família protocolar do socket (neste exemplo *AF\_INET*) e o tipo de socket (neste caso *SOCK\_STREAM*).

A função *connect* tem como argumentos o par (*host, port*) sendo que os valores apresentados são meramente exemplificativos.

Observemos agora o seguinte excerto simplificado de código em python:



É vital referir que a keyword *finally* em python é **sempre** executada em python.

A função *sendall(string)* do módulo *socket* que importámos no exemplo anterior envia toda a mensagem *msg.* A diferença entre a função *sendall(string)* e a *send(string)* é que a *sendall* continua a enviar os dados da *string* até terminar de enviar a totalidade dos dados **ou** ocorrer um erro.

A função *append* faz parte de um grupo de funções relativas a manipulação de cadeias de caracteres - strings - e esta, como o nome indica, anexa a string que é recebida como argumento (neste caso *data*) à variável sobre a qual é chamada esta função (neste caso *mensagens*).

## Em relação ao código do servidor…

O servidor desta atividade consiste em três funções, para além da função *main( ):*

* *adicionaMensagem(utilizador, mensagem)*

Esta função é a responsável por anexar mensagens à lista de mensagens de um determinado utilizador, usando o método *append(string)* referido anteriormente.

* *login(utilizador)*

Esta função cria um utilizador e atribui-lhe uma lista de mensagens na qual, através do método *adicionaMensagem* falado acima, são guardadas as mensagens enviadas para este utilizador.

* *leMensagens(utilizador)*

Este método apresenta no terminal todas as mensagens enviadas ao utilizador selecionado até ao momento **e que ainda não foram lidas**.A partir do momento em que são apresentadas são eliminadas e deixa de ser possível lê-las novamente.