TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2021-22
Primeiro Projeto
15 de Outubro de 2021

# Buggy Data Base (BDB)

Neste primeiro projeto de Fundamentos da Programação os alunos irão desenvolver as funções que permitam resolver cinco tarefas independentes para identificar e corrigir os problemas de uma base de dados (Buggy Data Base, BDB) que ficou corrompida por causas desconhecidas. A BDB contém a informação de autenticação dos utilizadores de um sistema e está a recusar erradamente o acesso de alguns dos utilizadores registados. As tarefas consistirão em: 1) Correção da documentação; 2) Descoberta do PIN da base de dados; 3) Verificação da coerência dos dados; 4) Desencriptação do conteúdo; e 5) Depuração de senhas.

## 1 Correção da documentação

### 1.1 Descrição do problema

O primeiro passo para começar a resolver os problemas da BDB é estudar a documentação fornecida. Infelizmente, a documentação também está corrompida, mas por sorte foram detetados alguns padrões de alterações no texto que podem ser revertidos. O primeiro padrão de alteração afeta as palavras do texto de forma individual: um *surto de letras* formado por pares de carateres da mesma letra minúscula/maiúscula foram inseridos nas palavras repetidamente. Para recuperar cada palavra, é preciso corrigir este surto. Para isso, quando um par minúscula/maiúscula da mesma letra se encontram adjacentes estas *reagem* desaparecendo. Por exemplo:

- Em `'aA'`, `'a'` e `'A'` reagem, deixando a cadeia de carateres vazia.
- Em `'aBbA'`, `'Bb'` autodestrói-se, deixando `'aA'` que, como acima, também reage não deixando nada.
- Em `'abAB'`, não há duas letras adjacentes compatíveis, portanto, nada acontece.
- Em `'aabAAB'`, embora `'aa'` e `'AA'` representem um par da mesma letra, as duas são minúsculas ou maiúsculas e, portanto, nada acontece.
- Em `'cCdatabasacCADde'`, produz-se a seguinte sequência de *reações* (indicadas entre parêntesis retos) até obter a palavra `'database'`:

  `'[cC]databasacCADde'` → `'databasa[cC]ADde'` → `'databas[aA]Dde'` → → `'databas[Dd]e'` → `'database'`

Após a correção do *surto de letras*, existe um segundo padrão de erros: aparecem palavras sem sentido no texto que correspondem a anagramas de palavras anteriores. A solução parece simples: para cada nova palavra, é preciso detetar e eliminar todos os anagramas (diferentes de si próprio) que se encontram no resto do texto.

## 1.2 Trabalho a realizar

O objetivo desta tarefa é escrever um programa em Python que permita corrigir a documentação da BDB conforme descrito anteriormente. Para isso, é necessário definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso seja necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos, para as outras assume-se que estão corretos.

### 1.2.1 corrigir_palavra: cad. carateres → cad. carateres (1 valor)

Esta função recebe uma cadeia de carateres que representa uma palavra (potencialmente modificada por um *surto de letras*) e devolve a cadeia de carateres que corresponde à aplicação da sequência de *reduções* conforme descrito para obter a palavra corrigida.

### 1.2.2 eh_anagrama: cad. carateres × cad. carateres → booleano (0,5 valores)

Esta função recebe duas cadeias de carateres correspondentes a duas palavras e devolve `True` se e só se uma é anagrama da outra, isto é, se as palavras são constituídas pelas mesmas letras, ignorando diferenças entre maiúsculas e minúsculas e a ordem entre carateres.

### 1.2.3 corrigir_doc: cad. carateres → cad. carateres (1,5 valores)

Esta função recebe uma cadeia de carateres que representa o texto com erros da documentação da BDB e devolve a cadeia de carateres filtrada com as palavras corrigidas e os anagramas retirados, ficando apenas a sua primeira ocorrência. Os anagramas são avaliados após a correção das palavras e apenas são retirados anagramas que correspondam a palavras diferentes (sequência de carateres diferentes das palavras anteriores ignorando diferenças de maiúsculas e minúsculas). Esta função deve verificar a validade do seu argumento, gerando um `ValueError` com a mensagem `'corrigir_doc: argumento invalido'` caso o seu argumento não seja válido. Para este fim, considere que as palavras apenas podem estar separadas por um único espaço, que o texto está formado por uma ou mais palavras, e que cada palavra é formada apenas por, pelo menos, uma letra (minúscula ou maiúscula).

### 1.3 Exemplo

```
>>> corrigir_palavra('abBAx')
'x'
>>> corrigir_palavra('cCdatabasacCADde')
'database'
>>> eh_anagrama('caso', 'SaCo')
True
>>> eh_anagrama('caso', 'casos')
False
>>> corrigir_doc('???')
ValueError: corrigir_doc: argumento invalido
>>> doc = 'BuAaXOoxiIKoOkggyrFfhHXxR duJjUTtaCcmMtaAGga \
eEMmtxXOjUuJQqQHhqoada JlLjbaoOsuUeYy cChgGvValLCwMmWBbclLsNn \
LyYlMmwmMrRrongTtoOkyYcCK daRfFKkLlhHrtZKqQkkvVKza'
>>> corrigir_doc(doc)
'Buggy data base has wrong data'
```

# 2 Descoberta do PIN

## 2.1 Descrição do problema

Ao fazer *double-click* para abrir o ficheiro que contém a BDB, abre-se uma janela com um painel de dígitos com o seguinte aspet:

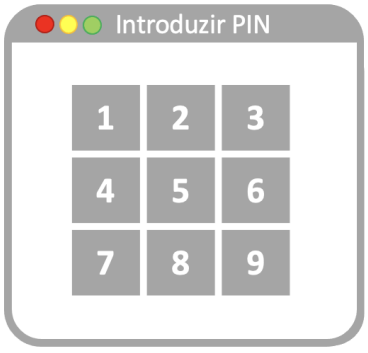


Figura 1: Painel com botões para a introdução do PIN da BDB.

Aparentemente, para aceder ao conteúdo da base de dados é necessário um PIN desconhecido. No entanto, entre a documentação fornecida, encontra-se o seguinte texto:

> *"Para melhorar a segurança, os códigos da base de dados deixaram de ser anotados. Em vez disso, memorize e siga o procedimento abaixo."*

A documentação continua explicando que cada botão a ser pressionado pode ser encontrado iniciando no botão anterior e movendo para botões adjacentes no teclado: `'C'` move para cima, `'B'` move para baixo, `'E'` move para a esquerda e `'D'` move para direita. Cada linha de instruções corresponde a um botão, iniciando no botão anterior (ou, para a primeira linha, o botão `'5'`); pressione qualquer botão em que estiver no final de cada linha. Se um movimento não levar a um botão ignore este movimento. Por exemplo, suponha que suas instruções estão codificadas num tuplo de strings como o seguinte:

```
('CEE', 'DDBBB', 'ECDBE', 'CCCCB')
```

Começando no `'5'`, mover para cima (`'2'`), para a esquerda (`'1'`) e para a esquerda (não podes e permaneces em `'1'`), então o primeiro botão é `'1'`. Começando com o botão anterior (`'1'`), mover para a direita duas vezes (`'3'`) e depois para baixo três vezes (parando em `'9'` após dois movimentos e ignorando o terceiro), terminando com `'9'`. Continuando a partir de `'9'`, mover para a esquerda, para cima, para a direita, para baixo e para a esquerda, terminando com `'8'`. Finalmente, mover para cima quatro vezes (parando em `'2'`), depois para baixo uma vez, terminando com `'5'`. Portanto, neste exemplo, o código da base de dados é 1985.

## 2.2 Trabalho a realizar

O objetivo desta tarefa é escrever um programa em Python que permita encontrar o PIN da BDB conforme descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso considere necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.

### 2.2.1 obter_posicao: cad. carateres × inteiro → inteiro (0,5 valores)

Esta função recebe uma cadeia de carateres contendo apenas um caráter que representa a direção de um único movimento (‘C’, ‘B’, ‘E’ ou ‘D’) e um inteiro representando a posição atual (1, 2, 3, 4, 5, 6, 7, 8 ou 9); e devolve o inteiro que corresponde à nova posição após do movimento.

### 2.2.2 obter_digito: cad. carateres × inteiro → inteiro (0,5 valores)

Esta função recebe uma cadeia de carateres contendo uma sequência de um ou mais movimentos e um inteiro representando a posição inicial; e devolve o inteiro que corresponde ao dígito a marcar após finalizar todos os movimentos.

### 2.2.3 obter_pin: tuplo → tuplo (1 valor)

Esta função recebe um tuplo contendo entre 4 e 10 sequências de movimentos e devolve o tuplo de inteiros que contêm o pin codificado de acordo com o tuplo de movimentos. Esta função deve verificar a validade do seu argumento conforme descrito nesta secção

(isto é, tuplo de entre 4 e 10 sequências de movimentos, em que cada movimento é uma string com 1 ou mais carateres 'C', 'B', 'E' ou 'D'), gerando um `ValueError` com a mensagem `'obter_pin: argumento invalido'` caso o seu argumento não seja válido.

### 2.3 Exemplo

```
>>> obter_posicao('C', 5)
2
>>> obter_digito('CEE', 5)
1
>>> obter_pin(())
ValueError: obter_pin: argumento invalido
>>> t = ('CEE',  'DDBBB', 'ECDBE', 'CCCCB')
>>> obter_pin(t)
(1, 9, 8, 5)
```

## 3 Verificação de dados

### 3.1 Descrição do problema

Consegue-se aceder aos dados da BDB, mas claro, as entradas estão encriptadas e o processo de desencriptação é computacionalmente pesado. Por sorte, existe uma sequência de controlo para cada entrada que permite detetar entradas erradas e assim reduzir a quantidade de dados que é preciso desencriptar. A deteção de uma entrada errada não garante que a senha correspondente esteja errada (porque a alteração pode afetar outras informações do utilizador contidas na BDB), mas permite reduzir a lista de casos suspeitos.
Cada entrada da BDB é representada como um tuplo com 3 campos: uma cadeia de carateres *cifra* contendo uma ou mais palavras encriptadas separadas por traços; uma outra cadeia de carateres *checksum* contendo uma sequência de controlo (5 letras minúsculas entre parêntesis retos); e finalmente um tuplo com dois ou mais números inteiros positivos de segurança (ainda não usados nesta fase). As palavras que constituem a *cifra* apenas podem estar formadas por letras minúsculas (tamanho mínimo 1). Uma entrada da BDB está correta apenas se a sequência de controlo é formada pelas cinco letras mais comuns na sequência encriptada, por ordem inversa de ocorrências, com empates decididos por ordem alfabética. Por exemplo:

- `('aaaaa-bbb-zx-yz-xy', '[abxyz]', (950,300))` está correto porque as letras mais comuns são 'a' (5 ocorrências), 'b' (3 ocorrências) e um empate entre 'x', 'y' e 'z', que são listados em ordem alfabética.
- `('a-b-c-d-e-f-g', '[abcde]', (124,325,7))` está correto porque, embora as letras estejam todas empatadas (1 de cada), as cinco primeiras são listadas em ordem alfabética.
- `('entrada-muito-errada', '[abcde]', (50,404))` está errada.

## 3.2 Trabalho a realizar

O objetivo desta tarefa é escrever um programa em Python que permita validar cada uma das entradas da BDB conforme descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso considere necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.

### 3.2.1 eh_entrada: universal → booleano (1,5 valores)[1]

Esta função recebe um argumento de qualquer tipo e devolve `True` se e só se o seu argumento corresponde a uma entrada da BDB (potencialmente corrupta) conforme descrito, isto é, um tuplo com 3 campos: uma cifra, uma sequência de controlo e uma sequência de segurança.

### 3.2.2 validar_cifra: cad. carateres × cad. carateres → booleano (2 valores)

Esta função recebe uma cadeia de carateres contendo uma cifra e uma outra cadeia de carateres contendo uma sequência de controlo, e devolve `True` se e só se a sequência de controlo é coerente com a cifra conforme descrito.

### 3.2.3 filtrar_bdb: lista → lista (1 valor)

Esta função recebe uma lista contendo uma ou mais entradas da BDB e devolve apenas a lista contendo as entradas em que o `checksum` não é coerente com a cifra correspondente, na mesma ordem da lista original. Esta função deve verificar a validade do seu argumento, gerando um `ValueError` com a mensagem `'filtrar_bdb: argumento invalido'` caso o seu argumento não seja válido.

## 3.3 Exemplo

```
>>> eh_entrada(('a-b-c-d-e-f-g-h', '[abcd]', (950,300)))
False
>>> eh_entrada(('a-b-c-d-e-f-g-h-2', '[abcde]', (950,300)))
False
>>> eh_entrada(('a-b-c-d-e-f-g-h', '[xxxxx]', (950,300)))
True
>>> validar_cifra('a-b-c-d-e-f-g-h', '[xxxxx]')
False
>>> validar_cifra('a-b-c-d-e-f-g-h', '[abcde]')
True
>>> filtrar_bdb([])
ValueError: filtrar_bdb: argumento invalido
```

[1] **ATENÇÃO!!** Esta função é a mesma que a da Secção 4.2.1 e apenas precisa ser definida uma vez.

```
>>> bdb = [('aaaaa-bbb-zx-yz-xy', '[abxyz]', (950,300)),
           ('a-b-c-d-e-f-g-h', '[abcde]', (124,325,7)),
           ('entrada-muito-errada', '[abcde]', (50,404))]
>>> filtrar_bdb(bdb)
[('entrada-muito-errada', '[abcde]', (50,404))]
```

# 4 Desencriptação de dados

## 4.1 Descrição do problema

Com a BDB filtrada e todas as entradas erradas detetadas, está na hora de as desencriptar para tentar descobrir quais são os utilizadores com senhas corruptas.
A informação da BDB está encriptada por uma cifra de troca. Para desencriptar o conteúdo de cada entrada, primeiro é preciso determinar o número de segurança. O número de segurança corresponde à menor diferença de entre todos os números contidos na sequência de segurança (última posição do tuplo que representa cada entrada da BDB). A seguir, é preciso trocar cada letra avançando no alfabeto um número de vezes igual ao número de segurança de cada entrada mais um para as posições pares, e menos um para as posições ímpares do texto: com número de segurança igual a 2 a letra `'a'` torna-se `'d'` se estiver numa posição par da string ou `'b'` se estiver numa posição ímpar; com o mesmo número de segurança a letra `'z'` torna-se `'c'` se estiver numa posição par ou `'a'` se estiver numa posição ímpar. Os traços transformam-se em espaços. Para determinar a posição par ou ímpar de um caráter é considerada a cadeia de carateres completa com o primeiro caráter na posição par (índice 0). Por exemplo, para a seguinte entrada (com código de controlo errado):

```
('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', (2223,424,1316,99))
```

o número de segurança é 325 ($424 - 99$) e o conteúdo decifrado é `'esta cifra e quase inquebravel'`.

## 4.2 Trabalho a realizar

O objetivo desta tarefa é escrever um programa em Python que permita desencriptar as entradas da BDB conforme descrito anteriormente. Para isso, é necessário definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso sejam necessárias. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.

### 4.2.1 eh_entrada: universal → booleano (1,5 valores)[2]

Esta função recebe um argumento de qualquer tipo e devolve `True` se e só se o seu argumento corresponde a uma entrada da BDB (potencialmente corrupta) conforme

[2] **ATENÇÃO!!** Esta função é a mesma que a da Secção 3.2.1 e apenas precisa ser definida uma vez.

descrito, isto é, um tuplo com 3 campos: uma cifra, uma sequência de controlo e uma sequência de segurança.

#### 4.2.2 obter_num_seguranca: tuplo → inteiro (1 valores)

Esta função recebe um tuplo de números inteiros positivos e devolve o número de segurança conforme descrito, isto é, a menor diferença positiva entre qualquer par de números.

#### 4.2.3 decifrar_texto: cad. carateres × inteiro → cad. carateres (1 valores)

Esta função recebe uma cadeia de carateres contendo uma cifra e um número de segurança, e devolve o texto decifrado conforme descrito.

#### 4.2.4 decifrar_bdb: lista → lista (1 valores)

Esta função recebe uma lista contendo uma ou mais entradas da BDB e devolve uma lista de igual tamanho, contendo o texto das entradas decifradas na mesma ordem. Esta função deve verificar a validade do seu argumento, gerando um `ValueError` com a mensagem `'decifrar_bdb: argumento invalido'` caso o seu argumento não seja válido.

### 4.3 Exemplo

```
>>> eh_entrada(('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', \
               (2223,424,1316,99)))
True
>>> obter_num_seguranca((2223,424,1316,99))
325
>>> decifrar_texto('qgfo-qutdo-s-egoes-wzegsnfmjqz', 325)
'esta cifra e quase inquebravel'
>>> decifrar_bdb([('nothing')])
ValueError: decifrar_bdb: argumento invalido
>>> bdb = [('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]',
            (2223,424,1316,99)), ('lctlgukvzwy-ji-xxwmzgugkgw',
            '[abxyz]', (2388, 367, 5999)), ('nyccjoj-vfrex-ncalml',
            '[xxxxx]', (50, 404))]
>>> decifrar_bdb(bdb)
['esta cifra e quase inquebravel', 'fundamentos da programacao',
 'entrada muito errada']
```

# 5 Depuração de senhas

## 5.1 Descrição do problema

Após de detetar as entradas da BDB erradas e desencriptar o conteúdo com o algoritmo desenvolvido, é possível obter uma lista de dicionários com o nome de utilizador (chave `'name'`), a senha potencialmente corrompida (chave `'pass'`)) e a regra *individual* de quando essa senha foi definida (chave `'rule'`)). O resto de informações de utilizador contidas na BDB (também potencialmente corrompidas) não são relevantes. Por exemplo:

```
[ {'name':'john.doe', 'pass':'aabcde', 'rule':{'vals':(1,3), 'char':'a'}},
  {'name':'jane.doe', 'pass':'cdefgh', 'rule':{'vals':(1,3), 'char':'b'}},
  {'name':'jack.doe', 'pass':'cccccc', 'rule':{'vals':(2,9), 'char':'c'}} ]
```

De acordo com a documentação fornecida, existem dois conjuntos de regras que devem ser cumpridas na definição de senhas: as regras *gerais* e as regras *individuais*.
As regras *gerais* são aplicadas a todas as senhas e são as seguintes:

- As senhas devem conter pelo menos três vogais minúsculas (`'aeiou'`),
- As senhas devem conter pelo menos um caráter que apareça duas vezes consecutivas.

As regras *individuais* são codificadas pelo valor da chave `'rule'` de cada entrada como dicionários com as chaves `'vals'` e `'char'`. O valor de `'vals'` é um tuplo de dois inteiros positivos correspondentes ao menor (primeira posição) e o maior (segunda posição) número de vezes que uma determinada letra minúscula (valor da chave `'char'`) deve aparecer para que a senha seja válida. Por exemplo, a regra `{'vals': (1,3), 'char':'a'}` significa que a senha deve conter a letra `'a'` pelo menos 1 vez e no máximo 3 vezes. O valor que representa o máximo número de vezes que aparece a letra é sempre maior ou igual que o mínimo.
No exemplo acima, apenas a primeira senha é válida. A senha do meio, `'cdefgh'` não é válida, porque não cumpre nenhuma das regras gerais (não contém 3 vogais, nem duas letras consecutivas iguais), nem a regra individual (não contém instâncias de `'b'`, e precisa de pelo menos 1). A terceira senha cumpre a regra individual (contém 6 `'c'`), mas apenas uma das regras gerais (não contém 3 vogais). A primeira senha é válida: contém 2 `'a'` (dentro dos limites da sua regra individual), 3 vogais (2 `'a'` e 1 `'e'`) e uma sequência de dois carateres iguais (`'aa'`).

## 5.2 Trabalho a realizar

O objetivo desta tarefa é escrever um programa em Python que permita encontrar os utilizadores com senhas que não cumprem as regras descrita atrás. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso considere necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos.

### 5.2.1 eh_utilizador: universal → booleano (1 valor)

Esta função recebe um argumento de qualquer tipo e devolve `True` se e só se o seu argumento corresponde a um dicionário contendo a informação de utilizador relevante da BDB conforme descrito, isto é, nome, senha e regra individual. Considere para este efeito que nomes e senhas devem ter tamanho mínimo 1 e podem conter qualquer caráter.

### 5.2.2 eh_senha_valida: cad. carateres × dicionário → booleano (1,5 valores)

Esta função recebe uma cadeia de carateres correspondente a uma senha e um dicionário contendo a regra individual de criação da senha, e devolve `True` se e só se a senha cumpre com todas as regras de definição (gerais e individual) conforme descrito.

### 5.2.3 filtrar_senhas: lista → lista (1 valor)

Esta função recebe uma lista contendo um ou mais dicionários correspondentes às entradas da BDB como descritas anteriormente, e devolve a lista ordenada alfabeticamente com os nomes dos utilizadores com senhas erradas. Esta função deve verificar a validade do seu argumento, gerando um `ValueError` com a mensagem `'filtrar_senhas: argumento invalido'` caso o seu argumento não seja válido.

## 5.3 Exemplo

```
>>> eh_utilizador({'name':'john.doe', 'pass':'aabcde',
                   'rule':{'vals': (1,3), 'char':'a'}})
True
>>> eh_utilizador({'name':'john.doe', 'pass':'aabcde',
                   'rule':{'vals': 1, 'char':'a'}})
False
>>> eh_senha_valida('aabcde', {'vals': (1,3), 'char':'a'})
True
>>> eh_senha_valida('cdefgh', {'vals': (1,3), 'char':'b'})
False
>>> filtrar_senhas([])
ValueError: filtrar_senhas: argumento invalido
>>> bdb = [ {'name':'john.doe', 'pass':'aabcde', 'rule':{'vals':(1,3),
             'char':'a'}}, {'name':'jane.doe', 'pass':'cdefgh',
             'rule':{'vals':(1,3), 'char':'b'}}, {'name':'jack.doe',
             'pass':'cccccc', 'rule':{'vals':(2,9), 'char':'c'}} ]
>>> filtrar_senhas(bdb)
['jack.doe', 'jane.doe']
```

# 6 Condições de Realização e Prazos

- A entrega do 1º projeto será efetuada exclusivamente por via eletrónica. Deverá submeter o seu projeto através do sistema Mooshak, até às **17:00 do dia 5 de Novembro de 2021**. Depois desta hora, não serão aceites projetos sob pretexto algum.

- Deverá submeter um único ficheiro com extensão *.py* contendo todo o código do seu projeto.

- O sistema de submissão assume que o ficheiro está codificado em UTF-8. Alguns editores podem utilizar uma codificação diferente, ou a utilização de alguns carateres mais estranhos (nomeadamente nos comentários) não representáveis em UTF-8 pode levar a outra codificação. Se todos os testes falharem, pode ser um problema da codificação usada. Nesse caso, deverá especificar qual é a codificação do ficheiro na primeira linha deste[3].

- Submissões que não corram nenhum dos testes automáticos por causa de pequenos erros de sintaxe ou de codificação, poderão ser corrigidos pelo corpo docente, incorrendo numa penalização de três valores.

- Não é permitida a utilização de qualquer módulo ou função não disponível built-in no Python 3.

- Pode, ou não, haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

- Lembre-se que no Técnico, a fraude académica é levada muito a sério e que a cópia numa prova (projetos incluídos) leva à reprovação na disciplina e eventualmente a um processo disciplinar. Os projetos serão submetidos a um sistema automático de deteção de cópias[4], o corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

# 7 Submissão

A submissão e avaliação da execução do projeto de FP é feita utilizando o sistema Mooshak[5]. Para obter as necessárias credenciais de acesso e poder usar o sistema deverá:

- Obter a senha para acesso ao sistema, seguindo as instruções na página: `http://acm.tecnico.ulisboa.pt/~fpshak/cgi-bin/getpass-fp21`. A senha ser-lhe-á enviada para o email que tem configurado no Fenix. A senha pode não chegar de imediato, aguarde.

[3] `https://www.python.org/dev/peps/pep-0263/`

[4] `https://theory.stanford.edu/~aiken/moss`

[5] A versão de Python utilizada nos testes automáticos é Python 3.7.3.

- Após ter recebido a sua senha por email, deve efetuar o login no sistema através da página: `http://acm.tecnico.ulisboa.pt/~fpshak/`. Preencha os campos com a informação fornecida no email.

- Utilize o botão *"Browse..."*, selecione o ficheiro com extensão *.py* contendo todo o código do seu projeto. O seu ficheiro *.py* deve conter a implementação das funções pedidas no enunciado. De seguida clique no botão *"Submit"* para efetuar a submissão.
  Aguarde (20-30 seg) para que o sistema processe a sua submissão!!!

- Quando a submissão tiver sido processada, poderá visualizar na tabela o resultado correspondente. Receberá no seu email um relatório de execução com os detalhes da avaliação automática do seu projeto podendo ver o número de testes passados/falhados.

- Para sair do sistema utilize o botão *"Logout"*.

Submeta o seu projeto atempadamente, dado que as restrições seguintes podem não lhe permitir fazê-lo no último momento:

- Só poderá efetuar uma nova submissão 5 minutos depois da submissão anterior.

- O sistema só permite 10 submissões em simultâneo pelo que uma submissão poderá ser recusada se este limite for excedido [6].

- Não pode ter submissões duplicadas, ou seja, o sistema pode recusar uma submissão caso seja igual a uma das anteriores.

- Será considerada para avaliação a **última** submissão (mesmo que tenha pontuação inferior a submissões anteriores). Deverá, portanto, verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada. Não há excepções!

- Cada aluno tem direito a **15 submissões sem penalização** no Mooshak. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.

# 8 Classificação

A nota do projeto será baseada nos seguintes aspetos:

[6] Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns alunos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

1. **Avaliação automática (80%).** A avaliação da correta execução será feita através do sistema Mooshak. O tempo de execução de cada teste está limitado, bem como a memória utilizada.
   Existem 132 casos de teste configurados no sistema: 28 testes públicos (disponibilizados na página da disciplina) valendo 0 pontos cada e 104 testes privados (não disponibilizados). Como a avaliação automática vale 80% (equivalente a 16 valores) da nota, uma submissão obtém a nota máxima de 1600 pontos.
   O facto de um projeto completar com sucesso os testes públicos fornecidos não implica que esse projeto esteja totalmente correto, pois estes não são exaustivos. É da responsabilidade de cada aluno garantir que o código produzido está de acordo com a especificação do enunciado, para completar com sucesso os testes privados.

2. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura[7]. Em particular, serão consideradas as seguintes componentes:

   - Boas práticas (1,5 valores): serão considerados entre outros a clareza do código, a integração de conhecimento adquirido durante a UC e a criatividade das soluções propostas.
   - Comentários (1 valor): deverão incluir a assinatura das funções definidas, comentários para o utilizador (*docstring*) e comentários para o programador.
   - Tamanho de funções, duplicação de código e abstração procedimental (1 valor).
   - Escolha de nomes (0,5 valores).

# 9 Recomendações e aspetos a evitar

As seguintes recomendações e aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

- Leia todo o enunciado, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.

- No processo de desenvolvimento do projeto, comece por implementar as várias funções dentro de cada tarefa pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina. Como as tarefas são independentes umas das outras (com a excepção da função comum das Secções 3.2.1 e 4.2.1), estas podem ser resolvidas por qualquer ordem e com independência umas das outras.

- Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.

[7]Podem encontrar sugestões de boas práticas em `https://gist.github.com/ruimaranhao/4e18cbe3dad6f68040c32ed6709090a3`

- Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá sentir a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).
- Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
- Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.
- A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
- Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.