



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Exercícios Python de correção automática

Mónica Oliveira (43775)

Philipp Al Badavi (45138)

Orientador

Professor Doutor João Beleza

Setembro, 2020

Resumo

O presente relatório foi elaborado como parte integrante na avaliação da unidade curricular Projeto, lecionada no âmbito da Licenciatura em Engenharia Informática e Multimédia, e pretende demonstrar o trabalho realizado durante a mesma.

Inicialmente, a proposta do projeto visionou a produção em larga escala de exercícios de programação Python, onde o grupo criou enunciados e scripts que permitem a geração de várias versões dos mesmos exercícios. Estes exercícios têm como motivação a aprendizagem e o treino, dos alunos do curso, em aspetos relacionados com a linguagem de programação Python, para que durante o seu percurso na licenciatura se sintam mais à vontade nesta matéria.

Durante a criação dos exercícios Python foi possível compreender a sua complexidade, o que levou o grupo a propor/discutir novas ideias com o orientador e a alterar um pouco o rumo do trabalho. Levando assim, também à realização de uma aplicação que nos permite encontrar possíveis erros na geração de diferentes versões e criar exercícios de forma mais automática.

Ao longo do desenvolvimento deste relatório procuramos refletir sobre as aprendizagens realizadas e os objetivos alcançados, explicando e justificando o trabalho elaborado.

O presente relatório culmina com uma reflexão final acerca deste percurso ao longo da unidade curricular e a sua importância, apoiada numa prática de orientação.

Abstract

The following report was made as part of the evaluation from the discipline "Projeto", part of the degree in Engineering in Informatics and Multimedia, intending to explain the work we have done.

The main goal of the project was to mass-produce Python programming exercises, in which, the group created questions and scripts that allow generating various versions of the same exercise. These exercises focus on motivating, help to study and learning Python programming language to the students of this degree so that they feel comfortable with Python in later stages of the degree.

During the creation of the Python exercises, the group realised the true complexity of this final project, which lead to discuss with the project advisor new ideas, such as, the creation of an application that allows us to find errors in the exercises versions and create exercises more autonomously.

During the execution of the project, we worked on reflecting on what we were learning and the goals we were reaching, with the main goal of explaining the work we where doing. This final report ends with a reflection about the journey of the discipline and its importance, based on an orientation basis.

Agradecimentos

A elaboração do presente trabalho não seria possível sem o apoio de alguns intervenientes. E, por isso, não podemos deixar de agradecer a algumas pessoas que, direta ou indiretamente, nos ajudaram neste percurso.

Em primeiro lugar, gostaríamos de agradecer a disponibilidade que o Professor João Beleza sempre ofereceu durante a elaboração do presente trabalho. Obrigado pelo apoio e estímulo que nos proporcionou.

Um grande obrigado ao Professor Paulo Trigo por todo o apoio prestado durante a unidade curricular, por nos ter guiado e ajudado com todo o tipo de documentação necessária à realização deste trabalho, mas mais que isso, obrigado por nos fazer compreender que a saúde mental é algo necessário e jamais se deve deixar de lado, principalmente em tempos como estes.

Um obrigado aos amigos que nos motivaram e tornaram as instalações do Instituição Superior de Engenharia de Lisboa a nossa segunda casa.

E finalmente, gostaríamos de agradecer aos pais, que sempre estiveram aqui para nos garantir esta saúde mental, por nos fazerem sentir capazes e ajudarem em tudo o que podiam.

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Figuras	ix
1 Introdução	1
2 Trabalho Relacionado	5
3 Modelo Proposto	7
3.1 Requisitos	7
3.2 Fundamentos	7
3.3 Abordagem	8
3.3.1 Exercícios Python	8
3.3.2 Aplicação	10
4 Implementação do Modelo	13
4.1 Criação de Exercícios Python	13
4.2 Aplicação	14
5 Validação e Testes	19
5.1 Exercícios	19
5.2 Diferenças nas Versões	22
5.3 Geração e Visualização de versões	24

6 Conclusões e Trabalho Futuro	27
Bibliografia	29
Apêndice	31
.1 Exercícios Desenvolvidos	31

Lista de Figuras

4.1	Diagrama da Aplicação	14
4.2	Motor de Busca	15
4.3	Enunciado exercício	15
4.4	Gerar versões	16
4.5	Diferenças nas versões	16
4.6	Tokens alterados nas versões	17
4.7	Versões criadas	17
4.8	Versão exemplo	18
5.1	Alínea considerada correta	19
5.2	Função do enunciado	20
5.3	Alínea considerada correta	20
5.4	Indexação usada na validação alínea	20
5.5	Alínea considerada como correta	20
5.6	Esboço de código presente no ficheiro full_program.py	21
5.7	Código usadas nesta alínea para verificar o output	21
5.8	Alínea proveniente de uma versão de exercício já criado	22
5.9	Simulação do output da 5 alínea na aplicação	23
5.10	Enunciado visualizado no Adobe Acrobat Reade DC e que foi gerado correndo os scripts em sequência	24
5.11	Enunciado visualizado e gerado na aplicação	25

Capítulo 1

Introdução

Este projeto tem como objetivo a criação de exercícios de correção automática diretamente relacionados com a linguagem de programação Python. Estes exercícios foram construídos com o intuito de serem usados nos trabalhos de casa das unidades curriculares de Matemática Discreta e Programação (MDP) e Matemática para Computação Gráfica (MCG) pertencentes ao plano curricular da Licenciatura em Engenharia Informática e Multimédia.

Cada exercício apresenta um enunciado de introdução ao mesmo, seguido de um excerto de código como uma função ou uma classe e por último contem, em norma, cinco alíneas de verdadeiro e falso, podendo estas ser perguntas diretas ou necessitarem do desenvolvimento de código por parte do aluno para a sua resolução. É possível visualizar a lista de exercícios criados no apêndice, no final deste relatório.

A criação dos exercícios está dividida em três fases, sendo elas:

1. Criação do enunciado e alíneas em \LaTeX e do código Python necessário tanto para o enunciado como para a resolução das alíneas.
2. Produção de um script Python no qual são configuradas as alterações que podem ser feitas ao enunciado e que resultados são colocados em cada alínea em função dessas alterações. Este script usa uma biblioteca que permite a substituição de tokens tanto no documento LaTeX como no código Python, gerando assim uma nova versão do exercício. A este documento é atribuído o nome `make_transformations.py`.
3. Criar mais dois scripts, o `make_random_versions.py` no qual é identificado o número de versões a gerar e o `make_pdfs.py`, por norma, es-

tes dois últimos documentos podem ser reutilizados de exercício para exercício.

Após a criação destes documentos basta executarmos os scripts com o nome **make_random_versions.py** e **make_pdfs.py** (por esta ordem) e obtemos várias versões do exercício com as suas alíneas autocorrigidas. Cada alínea contém uma resposta verdadeira e uma resposta falsa, o programa escolhe de forma aleatória qual das duas usar. Estas versões são necessárias para proporcionar a cada aluno um TPC personalizado obrigando o mesmo a pensar por si.

Os exercícios dizem-se "de correção automática" pois para cada alínea existe um ficheiro de extensão **.solution** que contem a informação de que a mesma é verdadeira ou falsa, que mais tarde deve ser comparado com a resposta dada pelo aluno.

O sistema que posteriormente utiliza estes exercícios irá usar os ficheiros pdf, tanto do enunciado como das alíneas criados, para exibir as versões do exercício geradas, ou seja, para cada aluno irá existir uma versão da pergunta com os seus respetivos pdfs. Com várias perguntas é desenvolvido um questionário que será fornecido a cada aluno permitindo o seu preenchimento e submissão para posterior avaliação por parte do sistema. O \LaTeX é usado pois permite introduzir no enunciado o código criado nos ficheiros Python e dispõe do package *standalone* que possibilita que o conteúdo do pdf não ocupe toda uma página A4 e só o espaço necessário ao texto, fazendo com que o enunciado e alíneas possam estar na mesma página. Mas, mais importante, este permite a substituição de strings o que é indispensável para a geração de versões.

Após a criação de vários exercícios o grupo compreendeu a sua complexidade e deparou-se com alguns erros de execução do código, nomeadamente na substituição de tokens, como por exemplo uma letra ser substituída não só no lugar correto, mas também em texto que não era desejado. Por vezes, era fácil encontrar o erro, por outras, nem tanto. Como tal, foi discutida uma solução para este problema. Surgiu então a ideia de criar uma aplicação web a partir do Jupyter Notebook que conseguisse ajudar na correção destes erros.

Assim sendo foi proposta, discutida e aceite a ideia da criação de uma aplicação web que permitisse minimizar estes erros. Esta app foi criada a

partir do Jupyter Notebook, com a ajuda da biblioteca **ipywidgets**, esta permite gerar páginas web a partir de código Python, mantendo por detrás todo o HTML, CSS e JavaScript. Esta aplicação permite ao seu utilizador:

- Navegar entre as diretorias do seu computador, podendo navegar até aos exercícios.
- Abrir uma pasta que contenha um exercício Python e observar o seu enunciado e alíneas.
- Verificar que caracteres estão a ser substituídos aquando da geração de versões, mostrando assim todas as alterações produzidas no enunciado e alíneas facilitando a deteção de erros.
- Gerar várias versões do mesmo exercício a partir de um *click*, não sendo necessário o próprio utilizador executar vários scripts.
- Visualizar as versões criadas.

Capítulo 2

Trabalho Relacionado

Apesar de a criação dos nossos exercícios seguir uma estrutura específica, podemos considerar como trabalho relacionado a aplicação móvel SoloLearn. Esta não gera versões para os exercícios, mas alcança o mesmo propósito: ensinar matérias à distância sobre a linguagem de programação Python a quem resolver os exercícios propostos. E como o SoloLearn já existem outras aplicações e pequenos cursos online que se dedicam a este tipo de formação.

Sobre a nossa aplicação, esta é até à data o único trabalho conhecido pois está desenhada especificamente para a estrutura dos exercícios desenvolvidos, proporcionando ao utilizador várias facilidades perante a criação das novas versões de exercícios.

Capítulo 3

Modelo Proposto

3.1 Requisitos

São requisitos funcionais deste projeto:

1. Criação de exercícios relacionados com a linguagem de programação Python, que permitam gerar várias versões do mesmo exercício.
2. Possibilidade de observar todos os caracteres substituídos na geração de novas versões, para correção de erros.
3. Geração de novas versões de forma automática.
4. Visualização das versões geradas.

3.2 Fundamentos

O trabalho realizado irá contribuir para o aumento de uma desejada biblioteca de exercícios Python, que posteriormente poderá ser usada em algumas das unidades curriculares da licenciatura, seja como treino ou avaliação nas mesmas. As matérias presentes nestes exercícios seguem uma lógica dividida por temas que será explicada na próxima secção.

A nossa aplicação irá disponibilizar ao seu utilizado uma agradável interface, possibilitando o mesmo de usufruir de comandos essenciais de forma fácil e rápida, como também proporcionar uma visão global dos exercícios ajudando à sua compreensão. Esta aplicação servirá também para ajudar

o próximo "criador" nesta área, permitindo uma melhor detecção de erros e minimizando o esforço necessário no que toca à geração de versões.

3.3 Abordagem

3.3.1 Exercícios Python

Todos os exercícios produzidos têm como objetivo final a familiarização do aluno com as diferentes funcionalidades que a linguagem Python apresenta, fornecendo um treino dos temas que nos, alunos que passaram pelas unidades curriculares ligadas a esta área, achamos importantes. Os temas abordados nos exercícios são os seguintes:

1. If, else, ciclos for e while - Estes exercícios iniciais são bastante básicos, onde as alíneas podem, por exemplo, perguntar qual o número de prints produzido por um ciclo for, ou o output de uma função que contem vários ifs e elses. Foram feitos para introduzir o aluno na linguagem de programação e alguns deles necessitam de mais atenção e cuidado na compreensão do código apresentado.
2. Listas e List Comprehension - Ao longo da licenciatura as listas são um tema que nunca deixa de estar connosco, mesmo em outras linguagens de programação, deste modo é importante a compreensão das mesmas o mais cedo possível. Por isso, existiu um grande foco neste tema.
3. Matriz e Vetores - Houve uma tentativa de criar exercícios relacionados com o trabalho prático final da unidade curricular MCG, mas o grupo apercebeu-se de que os enunciados destes exercícios se tornavam redundantes em relação aos já existentes, então, após a criação de alguns exercícios optou-se por dar destaque aos temas que são abrangidos indiretamente por esses trabalhos.
4. Classes e funções mágicas - A programação orientada a objetos é uma área muito explorada ao longo das unidades curriculares do curso e por isso deve ser definitivamente o mais explorada possível. Dentro do tema das classes foram criados exercícios que introduzem a noção de variáveis "privadas" (apesar de estas não existirem oficialmente, em Python, é apresentado ao aluno um código que faz com que as mesmas

se tornem impossíveis de aceder da maneira habitual) nas classes de Python, mostrando assim algum aprofundamento. O tema das funções mágicas é essencial, os alunos acabam por usar estas funções ao longo dos trabalhos sem se aperceberem e não aprendendo que funcionalidades podem adicionar a uma classe a partir delas.

5. Exceções - Foi feita uma introdução ao tratamento de exceções a partir dos blocos "try catch", não foram criados exercícios inteiramente destinados a este tema, mas foi adicionado a outros, como por exemplo num exercício sobre classes onde no interior da classe existe uma função que contém estas exceções.
6. Variáveis globais e pointers - Estas matérias são bastante fáceis de compreender quando o estudo se foca nelas, são temas bastante importantes visto que se não forem bem compreendidos podem causar grandes perdas de tempo durante a programação. Um erro comum dos alunos, principalmente no caso dos pointers, é os mesmos guardarem, por exemplo, uma lista em X, efetuarem um pointer para esta lista como Y=X e após manipularem X esperarem que Y se mantenha inalterada. Os exercícios criados tentaram simular esta situação para que os alunos percebam o que esperar relativamente. Tanto o tema das variáveis globais como dos pointers, não engloba um exercício na sua totalidade, aparecendo por norma como um subtema.
7. Regex e lambda - Estes temas tal como o Exceptions, também foram adicionados aos exercícios já baseados noutros temas mais centrais, principalmente porque tanto o lambda como Regex são matérias mais avançadas, por isso é feita uma pequena introdução a esses temas demonstrando as suas funcionalidades.
8. Funções built-in do Python - Funções como len(), range(), abs(), sort(), min(), max(), etc. são essenciais para a programação em Python. Foram realizados exercícios de modo a que os alunos se familiarizem com estas funções e aprendam que algumas delas podem levar parâmetros adicionais. Um exemplo disso é o caso da função sort() que quando aplicada a uma lista de números os ordena de forma crescente, mas se colocarmos o atributo reverse a True os ordena de forma decrescente.

9. Numpy e Dicionários - A biblioteca Numpy e os dicionários são muito usados em cadeiras seguintes na licenciatura, por isso, foram abrangidas algumas das funcionalidades dos arrays numpy e das bibliotecas.

3.3.2 Aplicação

Como foi anteriormente explicado, a nossa aplicação web não se encontrava abrangida nos objetivos da proposta do projeto inicial, mas foi uma ideia discutida e estudada entre o grupo e o orientador, onde se tentou compreender e avaliar quais as maiores necessidades atuais e futuras. Estas necessidades passam pela observação do exercício como um todo, incluindo na mesma página o enunciado e as alíneas, a deteção de possíveis erros na geração de versões. Posto isto, o grupo comprometeu-se a realizar esta componente bastante interessante e útil.

A interface da nossa aplicação foi programada através do Jupyter Notebook usando as funcionalidades da biblioteca ipywidgets, tornando a mesma uma página web. Esta biblioteca foi-nos dada a conhecer pelo próprio orientador aquando da explicação do projeto dos TPC como um todo, outra das propostas seria a implementação de uma aplicação web que leva até aos alunos, através de um link, o seu próprio TPC. Esta aplicação usava os widgets desta biblioteca e o grupo rendeu-se à mesma quando se apercebeu que seria possível programar uma página web em Python.

Pretende-se que a aplicação web desenvolvida disponibilize o seguinte cenário: o utilizador dispõe de um motor de busca otimizado para poder chegar à diretoria onde o(s) exercício(s) se encontra(m), quando entra numa diretoria em que a aplicação reconhece como uma pasta de um exercício é lhe aberta uma página diferente onde é possível observar o enunciado do mesmo junto com as suas alíneas. Nessa mesma página estarão vários botões de opções, que levarão até outras páginas, que permitem: observar os caracteres que serão substituídos no enunciado e código provenientes da geração versões; gerar o número de versões desejado para o exercício; visualizar as versões criadas. Quando o utilizador tiver terminado as ações pretendidas pode regressar ao motor de busca para abrir um novo exercício.

É possível reconhecer uma pasta como um exercício pois este é constituído por um sistema de ficheiros. São ficheiros obrigatórios: `program.py`, `full_program.py`, `make_transformations.py`, `make_random_versions.py`, `util_`

`make_random_versions.py`, `true_or_false_question.tex` e pelo menos uma alínea `answer_1_false.tex` e `answer_1_true.tex`.

São da autoria do criador do exercício os ficheiros:

`program.py`, contém o código Python que será apresentado no enunciado;

`full_program.py`, contém o código presente em `program.py` mais o necessário à resolução das alíneas;

`make_transformations.py`, onde se encontram descritas as alterações que serão feitas para a geração de versões, nomeadamente que tokens serão substituídos e pelo que;

`true_or_false_question.tex`, o enunciado do exercício;

`answer_1_false.tex` e **`answer_1_true.tex`**, o texto da alínea verdadeira e da alínea falsa.

Esta geração de versões está assente numa substituição de tokens porque este método nos garante uma dificuldade semelhante entre as versões do exercício.

Capítulo 4

Implementação do Modelo

4.1 Criação de Exercícios Python

Durante a criação dos exercícios e geração das sua versões acompanhou-se a seguinte sequência:

1. Escolha do tema do exercício.
2. Criação do código em Python no qual se baseia o exercício.
3. Criação do enunciado e das alíneas.
4. Criação do ficheiro **make_tansformations.py**, indicando todos os tokens que se pretendem mudar e para que valores.
5. Implementação de funções no script **make_tansformations.py** que possibilitam a geração das respostas corretas e incorretas para cada alínea.
6. Criação do script **make_random_versions.py** onde são indicados todos os ficheiros que devem ser alterados e que não devem, também é indicado o número de versões a gerar.
7. Geração de novas versões do exercício usando o script **make_random_versions.py** e após correr esse script executamos o **make_pdfs.py** que utiliza os ficheiros referentes a cada alínea e o ficheiro do enunciado que se encontram em formato **.tex** convertendo os em formato **.pdf**.

4.2 Aplicação

A classe responsável pela interface da nossa aplicação corre sobre o Jupyter Notebook com a ajuda da biblioteca *ipywidgets*, a qual demos o nome de ***App Interface***. Esta classe tem uma instância da classe ***Search Engine*** e uma instância da classe ***Manipulate Exercise***.

Search Engine como o próprio nome indica um o motor de busca, esta permite determinadas ações como navegar entre diretorias, obter o nome dos ficheiros e das pastas, correr scripts, etc. ***Manipulate Exercise*** implementa todos os processos ligados a manipulação de ficheiros, verificação de componentes necessárias a um exercício e obtenção das mesmas, como por exemplo as imagens png dos pdf gerados necessárias à apresentação do enunciado e alíneas do exercício na aplicação.

O script ***differenceFinder*** encarrega-se de retornar as diferenças entre versões. Este aproveita o script ***make_transformations.py*** com objetivo de compreender que tokens serão alterados e para que valores no documento escolhido, este pode ser tanto um script de Python como um texto em Latex, e por fim, usando a biblioteca de substituição de tokens realça estes novos valores dos tokens substituindo-os com *tags* de HTML para se diferenciarem no resultado apresentado na aplicação.

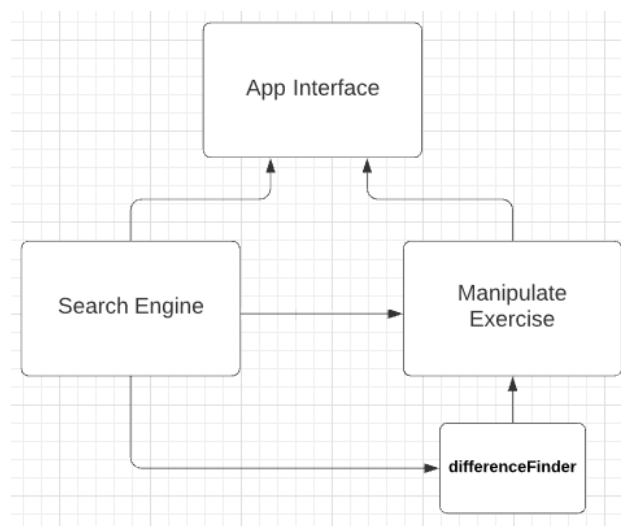


Figura 4.1: Diagrama da Aplicação

A aplicação implementa um motor de busca (figura 4.2) com a ajuda da biblioteca standart *os*, esta permite percorrer as diretorias do computador ajudando o utilizador a encontrar a pasta com o exercício Python.



Figura 4.2: Motor de Busca

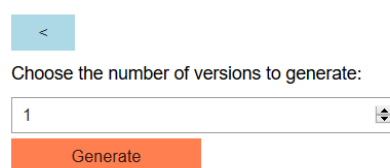
A página de exibição do exercício (figura 4.3) usa as imagens png geradas para mostrar o enunciado e as alíneas do mesmo. Caso não existiam a aplicação apresenta a funcionalidade de as gerar a partir dos documentos pdf.



Figura 4.3: Enunciado exercício

Para gerar versões (figura 4.4) a aplicação executa os scripts **make_random_versions.py** e **make_pdfs.py**.

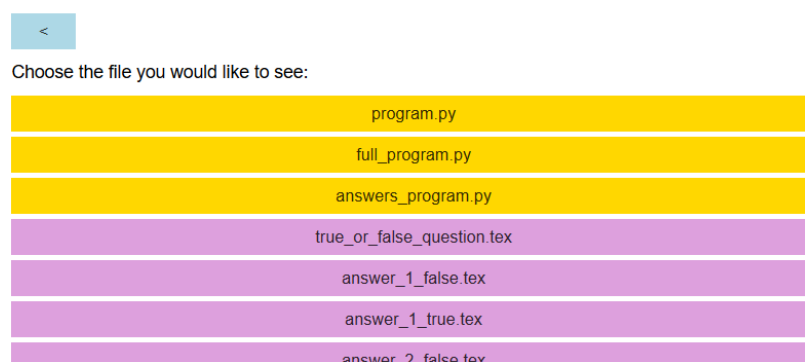
Nota: Em windows, para esta geração funcionar é necessário executar o Jupyter Notebook como administrador, pois sem ter os privilégios de administrador o mesmo não permite a criação dos documentos/ficheiros desejados.



A screenshot of a web interface for generating versions. It features a light blue button with a left-pointing arrow at the top left. Below it, the text "Choose the number of versions to generate:" is displayed. Underneath this text is a text input field containing the number "1". To the right of the input field is a small square button with a downward-pointing arrow. Below the input field is an orange button labeled "Generate".

Figura 4.4: Gerar versões

Existem vários ficheiros, tanto documentos produzidos em Latex como scripts criados em Python, que podem sofrer alterações com a geração de versões, então, a aplicação permite a visualização de todos eles (figura 4.5) com as diferenças entre versões a vermelho (figura 4.6).



A screenshot of a web interface showing a list of files. At the top left is a light blue button with a left-pointing arrow. Below it, the text "Choose the file you would like to see:" is displayed. Below this text is a list of seven files, each on a separate line. The first three files are highlighted in yellow: "program.py", "full_program.py", and "answers_program.py". The remaining four files are highlighted in purple: "true_or_false_question.tex", "answer_1_false.tex", "answer_1_true.tex", and "answer_2_false.tex".

Figura 4.5: Diferenças nas versões



```
program.py

from random import seed
from random import choice
from random import randint
from string import ascii_letters
seed(1938043)

o = []

def random_string_generator():
    b = ""
    my_str = ""
    for w in range(0,3):
        my_str += choice(ascii_letters).lower()
    for y in range(randint(1,15)):
        idx = randint(1,len(my_str) - 1)
```

Figura 4.6: Tokens alterados nas versões

Para visualizarmos versões já geradas a aplicação usa as imagens png (figura 4.7 e 4.8) que se encontram na pasta de cada versão.

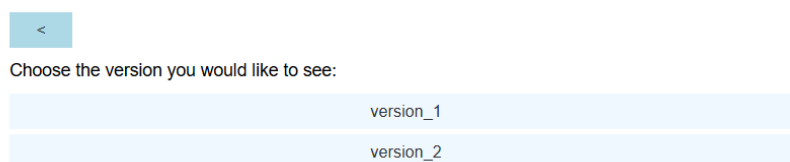


Figura 4.7: Versões criadas

**QUESTION: "true_or_false_question_class_plus_odd_and_even_nums", version 1**

Considere o programa, Python 3, que se segue. A classe 1 dispõe do método `fixed_num(self,nums)`, crie o método `fixed_num(self,nums)`, que devolve uma lista com os números iguais ao seu índice na lista que se encontram, se nenhum verificar esta condição é retornada uma lista vazia.

```
from random import randint
from random import seed

seed(1712110)

f = 35
class 1:
    global f
    f = 36
    def print_v1(self):
        if self == True:
            print("function_print_v1")
        , ..
```

Figura 4.8: Versão exemplo

Capítulo 5

Validação e Testes

5.1 Exercícios

Ao ser gerada uma nova versão do exercício são criadas para cada alínea uma opção verdadeira e uma opção falsa, para podermos validar se essas opções estão corretas é necessário usar o ficheiro **full_program.py**, este contém todo o código apresentado no enunciado e código essencial para a resolução das alíneas. Este script também é importante visto que durante a geração das alíneas, este em forma executável é utilizado para aceder as variáveis existentes no script e usá-las para gerar resultados das alíneas. A validação destas alíneas pode ser feita das seguintes formas:

1. **Validação direta**, existem alíneas que são fáceis de validar não sendo necessário o ficheiro anteriormente mencionado. Temos como exemplo as figuras abaixo, onde não é necessário recorrer ao ficheiro `full_program.py` para a sua validação, pois a função mencionada na alínea terá sempre o mesmo resultado: 1 print.

A função ***print_cicle***(10919) produz 1 print.

Figura 5.1: Alínea considerada correta

```
def print_cicle(m):
    for g in range(m):
        print(g)
    return g
```

Figura 5.2: Função do enunciado

2. **Validação usando indexação**, existem alíneas que pedem, por exemplo, qual o valor da lista num certo índice, para esta validação já é necessário recorrer ao ficheiro **full_program.py** onde se indexa a lista corretamente para verificar se os resultados coincidem, como é possível observar nas figuras abaixo.

[O elemento da lista **x**, no índice 14397, é 1220.]

Figura 5.3: Alínea considerada correta

```
>>> x[14397]
1220
```

Figura 5.4: Indexação usada na validação alínea

3. **Validação usando funções**, para alíneas que necessitam de mais programação relativamente à anterior, existe sempre uma função que as valida no ficheiro **full_program.py**. Os alunos não dispõem destas funções pois elas fazem parte da resolução da alínea. Apresentamos abaixo um exemplo deste tipo de validação onde é possível observar que para validar a alínea da figura 5.5, é o método **_truediv_** que pertence a classe **Y**(figura 5.6), que produz o output apresentado na figura 5.7, onde é possível averiguar que são produzidos exatamente 4 prints.

O output desta função **m[4420] / m[13851]** produz 4 prints.

Figura 5.5: Alínea considerada como correta


```
m = []

class Y:
    __i = 7

    def __init__(self, num = 0):
        self.num = num

    def __truediv__(self, othernum):
        try:
            print(self.num, "/", othernum.num)
            result = self.num / othernum.num
            print(result)
            assert result % 2 == 0
        except AssertionError:
            print("result odd")
        except ZeroDivisionError:
            print("ZeroDivisionError")
        finally:
            print("Done!")

    def __repr__(self):
        return str(self.num)

for b in range(19446):
    m.append(Y(randint(-33, 41)))
```

Figura 5.6: Esboço de código presente no ficheiro full_program.py

```
>>> m[4420]
-24
>>> m[13851]
31
>>> m[4420] / m[13851]
-24 / 31
-0.7741935483870968
result odd
Done!
```

Figura 5.7: Código usadas nesta alínea para verificar o output

5.2 Diferenças nas Versões

Quando é gerada uma versão do exercício, na pasta que contem essa versão, é criado um ficheiro txt, **seed.txt**, que contem o valor do seed que foi utilizado na geração da versão, esse ficheiro depois pode ser utilizado pela aplicação para gerar o mesmo output, caso não exista ainda nenhuma versão feita. A aplicação gera um seed, guardando esse ficheiro na raiz para futuro uso, quando aparecerem versões a aplicação vai sempre por definição optar pelo seed da primeira versão.

Para validação do correto funcionamento desta app, foi usada a comparação de exercícios já criados, se o output da aplicação for igual ao exercício já criado e forem destacadas as partes do texto que mudam então a aplicação esta a funcionar corretamente. Nas figuras seguintes é possível observar essa comparação.

```
\documentclass[12pt,varwidth=16cm,border=1pt]{standalone}

\input{latex_packages.tex}

\begin{document}

O output da função \textbf{\textit{factorial{}}}\verb+o+[\verb+75+]\textbf{\textit{}}}, é 3628800.

\questiontrue

\end{document}
```

Figura 5.8: Alínea proveniente de uma versão de exercício já criado

```
\documentclass[12pt,varwidth=16cm,border=1pt]{standalone}
```

```
\input{latex_packages.tex}
```

```
\begin{document}
```

O output da função `\textbf{\textit{factorial{}}}`***verb+o+[verb+75+]***`\textbf{\textit{}}}`, é **3628800**.

```
\questiontrue
```

```
\end{document}
```

Figura 5.9: Simulação do output da 5 alínea na aplicação

5.3 Geração e Visualização de versões

Para validarmos esta funcionalidade geramos versões de duas formas distintas, uma delas correndo os scripts em sequencia através do cmd, como era feito anteriormente à criação da aplicação, e a outra pela funcionalidade da aplicação que permite gerar estas versões. É possível observar nas figuras 5.10 e 5.11 que os outputs foram iguais, validando assim a geração através da app.

Considere o programa, Python 3, que se segue. Ignore a variável `seed` e a função `pseudo_random_integer`, que se destinam exclusivamente à geração de números pseudo-aleatórios.

```
seed = 1871146
def pseudo_random_integer(min_int, max_int):
    global seed
    seed = (16807*seed) % 2147483647
    return int(min_int + (max_int - min_int) * seed / 2147483646)

d = []
for p in range(19053):
    d.append(pseudo_random_integer(939, 1939))
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

Figura 5.10: Enunciado visualizado no Adobe Acrobat Reader DC e que foi gerado correndo os scripts em sequência

Considere o programa, Python 3, que se segue. Ignore a variável `seed` e a função `pseudo_random_integer`, que se destinam exclusivamente à geração de números pseudo-aleatórios.

```
seed = 1871146
def pseudo_random_integer(min_int, max_int):
    global seed
    seed = (16807*seed) % 2147483647
    return int(min_int + (max_int - min_int) * seed / 2147483646)

d = []
for p in range(19053):
    d.append(pseudo_random_integer(939, 1939))
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

Figura 5.11: Enunciado visualizado e gerado na aplicação

Capítulo 6

Conclusões e Trabalho Futuro

Com a finalização deste trabalho o grupo criou cerca de vinte e dois exercícios sobre a linguagem de programação Python, os quais abrangem temas dados nas unidades curriculares de Matemática Discreta e Programação (MDP) e Matemática para Computação Gráfica (MCG), pertencentes ao plano curricular da Licenciatura em Engenharia Informática e Multimédia. Estes temas englobam matérias introdutórias, que nos servem como bases de programação ao longo da licenciatura. Foram criados em enunciados em Latex e código Python e permitem gerar várias versões do mesmo exercício com o intuito de personalizar o TPC destinado a cada aluno, obrigando o mesmo a raciocinar por si.

Foi desenvolvida uma aplicação web que facilita tanto a geração de versões, como a correção de erros destes exercícios, programada em Jupyter Notebook com a ajuda da biblioteca ipywidgets.

Esperamos que em trabalhos futuros continuem a aumentar esta biblioteca exercícios Python necessária para um maior desenvolvimento das unidades curriculares acima referidas e que a nossa aplicação sirva de suporte, tornando o processo mais fácil do que antes.

Bibliografia

[Python 3.7.9, 2020] Python3.7.9 (2020). Python programming language. <http://docs.python.org/py3k/>.

[Jupyter, 2020] Jupyter Notebook (2020). Jupyter Notebook documentation. <https://jupyter-notebook.readthedocs.io/en/stable/jupyter-notebook.readthedocs.io/en/stable/>.

[Pynative, 2020] Pynative (2020). Pynative Python Exercises. <https://pynative.com/python-exercises-with-solutions/>.

[SoloLearn, 2020] SoloLearn (2020). Aplicação de Andriod/iOS SoloLearn.

[Jupyter, 2020] Jupyter Widgets (2020). ipywidgets documentation. <https://ipywidgets.readthedocs.io/en/latest/>

[Python 3.7.9, 2020] Python (2020). Python Package Index. <https://pypi.org/project/pdf2image/>

Apêndice

.1 Exercícios Desenvolvidos

Considere o programa, Python 3, que se segue. Implemente a função `take_out_repetitions`, que devolve uma lista sem repetições e a função `merge`, que recebe uma lista de intervalos, onde cada intervalo é um tuplo (início, fim), e retorna uma lista com intervalos fundidos, caso isso seja possível.

```

from random import randint
from random import seed

seed(1241846)

b = []
f = []
k = []
o = []

def take_out_repetitions(nums):

def merge(nums):

dim = 19863
for t in range(dim):
    b.append([randint(405,868),randint(868,1736)])

f = take_out_repetitions(b)
k = merge(f)
o = merge(b)

```

Para testar o funcionamento das funções execute o seguinte código.

```

>>> merge([(1, 405), (5, 8), (4, 10), (20, 25)])
[(1, 405), (4, 10), (20, 25)]
>>> take_out_repetitions([(5, 8), (1, 405), (5, 8),
                          (4, 10), (1, 405), (20, 25)])
[(5, 8), (1, 405), (4, 10), (20, 25)]

```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Caso seja necessário use a função `round` com 2 casas decimais.

Indique se é verdadeiro ou falso.

O elemento da lista **b**, no índice 1, é 5.

A lista **o**, é 3972.6 vezes menor que a lista **b**.

O maior intervalo da lista **k**, tem o tamanho de 1323

O maior intervalo da lista **k**, é [406, 1730].

O elemento da lista **b**, no índice 2800, é [468, 1659].

Considere o programa, Python 3, que se segue. Escreva a classe **Reverser**. Os objetos da classe **Reverser** tem o método **reverse()**. O método **reverse()** reverte todos os números que não possuam mais de 32 bits, ou seja, os números positivo com mais de 32 bits irão reverter para números negativos e vice-versa.

Escreva ainda as funções **to_bits()** e **binary_sum()**, estas não são métodos da classe acima. A função **to_bits()** converte um número decimal em binário e a função **binary_sum()** soma dois números binários.

```
from random import randint
from random import seed

seed(1501676)

class Reverser:

    def reverse(self, num):

def to_bits(num):

def binary_sum(num1, num2):

d = []
g = []
v = []

dim = 19566
for h in range(dim):
    d.append(randint(-2**31, 2**31))

q = Reverser()
for h in range(dim):
    g.append(q.reverse(d[h]))

for h in range(dim):
    v.append(q.reverse(g[h]))
```

Considere o código de testes, à classe **Reverser** e funções **to_bits()** e **binary_sum()**, que se segue.

```
>>> q.reverse("-110101101101001011111110011000")
"-000110011111110100101101101011"
>>> q.reverse("101011011001101010010111101100")
"001101111010010101100110110101"
>>> q.reverse(728147436)
634741827
>>> to_bits(0)
"0"
>>> to_bits(728147436)
"101011011001101010010111101100"
>>> binary_sum("101011011001101010010111101100"
               ,"10101001100001010111001111110")
"1000000100101110101010001101010"
>>> binary_sum("101011011001101010010111101100"
               ,"-1100001000110110110010110000100")
"-110101101101001011111110011000"
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes

O elemento da lista **d**, no índice 11349, é 703105724.

O elemento da lista **g**, no índice 17201, é -754169488.

A soma do número de índices em que a lista **v** e a lista **d** diferem são 10127.

O número de elementos da lista **d**, com 32 ou mais bits, é 8171.

A soma binária do elemento da lista **d** e da lista **g**, no índice 1246, tem como resultado inverso 0000000010001000011100010100001.

Considere o programa, Python 3, que se segue. A função `random_string_generator` gera uma string de dimensão aleatória constituída por 3 tipos caracteres e a função `message_optimizer` comprime esta string da seguinte forma: sempre que existam caracteres iguais seguidos são substituídos pela identificação desse carácter seguido do número de ocorrências do mesmo, como por exemplo, a string "aaaaadaaaafdddfdd" será comprimida para "a5da3fd3f2d".

```
from random import seed
from random import choice
from random import randint
from string import ascii_letters
seed(1938043)

o = []

def random_string_generator():
    b = ""
    my_str = ""
    for w in range(0,3):
        my_str += choice(ascii_letters).lower()
    for y in range(randint(1,15)):
        idx = randint(1,len(my_str) - 1)
        b += choice(my_str)
    return b

def message_optimizer(my_str):
    char_to_evaluate = my_str[0]
    finalMessage = ""
    counter = 1
    char_occurrence = 1
    if len(my_str) == 1:
        return my_str
    for char in my_str[1:]:
        counter += 1
        if char_to_evaluate == char:
            char_occurrence += 1
        if char_to_evaluate != char:
            finalMessage += char_to_evaluate
            finalMessage += str(char_occurrence) if char_occurrence > 1 else char_to_evaluate
            char_to_evaluate = char
            char_occurrence = 1
        if counter == len(my_str):
            finalMessage += char_to_evaluate
            finalMessage += str(char_occurrence) if char_occurrence > 1 else char_to_evaluate
    return finalMessage

for w in range(19049):
    b = random_string_generator()
    r = message_optimizer(b)
    o.append(r)
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

O número de vezes que a letra **q** da lista **o** aparece no índice **15460** é **5**.

O primeiro elemento da lista **o**, com o **maior** tamanho, é **v2akv2a3kvkvkv**.

O elemento da lista **o**, no índice **14261**, é **g2ugul2g2**.

A letra com o maior número de ocorrência na lista **o** é **q**, com um valor de **12**.

O tamanho do elemento da lista **o**, no índice **5661**, é **15**.

Considere o programa, Python 3, que se segue. Este exercício é baseado no tema "métodos mágicos" e é aconselhada a procura de informação sobre o mesmo antes da resolução das seguintes alíneas.

```
from random import randint
from random import choice
from random import seed

seed(1749806)

class M:

    def __init__(self, lista = []):
        self.lista = lista

    def __getitem__(self, index):
        decision = choice((-1, 1))
        idx = (index + decision)

    def __setitem__(self, index ,value):
        self.lista[index] = value

    def __repr__(self):
        str_lista = ""
        for u in self.lista:
            str_lista += ", " + str(u)
        return '[' + str_lista[2:] + ']'

    def __len__(self):
        return len(self.lista) - 1

j = []

for f in range(19760):
    value = randint(1, 220)
    j.append(value)

b = M(j)
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

Com a função `__setitem__(self, index, value)` a classe `M` suporta `item assignment`.

O output da função `b.__getitem__(19471)` e função `M(j)[19471]` é igual.

Não é possível **alterar** o elemento da classe `M` da seguinte forma: `b[index] = value`.

A função ***print***(`b`), faz print de lista de strings.

A função ***len***(`b`), retorna 19760.

Considere o programa, Python -27, que se segue. Este exercício consiste na introdução ao tratamento de exceções. Tome em atenção que os elementos da classe B podem ser divididos usando o método `_truediv_` e o símbolo de divisão `"/"`.

```

from random import randint
from random import seed

seed(1918616)

o = []

class B:
    __q = 7

    def __init__(self, num = 0):
        self.num = num

    def __truediv__(self, othernum):
        try:
            print(self.num,"/", othernum.num)
            result = self.num / othernum.num
            print(result)
            assert result % 2 == 0
        except AssertionError:
            print("result_odd")
        except ZeroDivisionError:
            print("ZeroDivisionError")
        finally:
            print("Done!")

def factorial(num):
    if isinstance(num, B):
        num = num.num
    if num == 1: return 1
    else: return num * factorial(num - 1)

def print_cicle(o):
    for n in range(o):
        print(n)
    return
return n

for e in range(19684):
    o.append(B(randint(-27, 1380)))

g = B()

```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

A função ***print_cicle(7650)*** produz 7650 prints.

Executar o código: `o[19192] / o[19009]`, produz 3 prints.

É possível aceder ao atributo `__q` da classe B da seguinte forma: `B.__q`.

O método `g.num` retorna 0.

O output da função ***factorial(o[75])*** é 3628801.

Considere o programa, Python 3, que se segue. Este exercício consiste na introdução às expressões regulares. As funções `regular_match()` e `regular_search()` retornam o número de elementos da lista fornecida, com o padrão especificado. Já a função `regular_subtitution()` retorna a lista manipulada.

```

from random import randint
from random import seed
import re

seed(1918616)

o = []
e = []

class Q:
    __n = 7

    def __init__(self, num = 0):
        self.num = num

    def get_var(self):
        return self.__n

def power(num, exp):
    if isinstance(num, Q):
        num = num.num
    if exp == 1: return num
    else: return num * power(num, exp - 1)

def regular_search(pattern_val, the_list):
    return sum(list(map(lambda x: 1 if re.search(r" + pattern_val, x)
        else 0, the_list)))

def regular_subtitution(pattern_val, substitute_val, the_list):
    return list(map(lambda x: re.sub(r" + pattern_val, substitute_val, x)

def regular_match(pattern_val, the_list):
    return sum(list(map(lambda x: 1 if re.match(r" + pattern_val, x)
        else 0, the_list)))

for l in range(19974):
    value = randint(1, 1482)
    o.append(Q(value))
    e.append(str(value))

b = Q()
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

O retorno da função ***regular_search***(6, e) é 6.

Após executar a função ***regular_substitution***(9, 5, e), a quantidade de elementos com o número 5 aumenta 1.92 vezes.

As funções `Q().get_var()` e `b._Q__n` produzem um output igual.

O retorno da função ***regular_match***(2{0-1}, e) é 0.

O output da função ***power***(o[62], o[715]) é 63.

Considere o programa, Python 3, que se segue. São apresentados 3 arrays, os primeiros 2 são constituídos por números aleatórios e o último pelo resultado de utilização da função `comparator()`. Implementar a função `comparator()`, está recebe como parâmetros dois valores e compara os mesmos. Se o primeiro valor for maior que o segundo retorna "1", se for menor retorna "-1" e se for igual retorna "0".

```

from random import seed
from random import randint
seed(1599060)

y = []
c = []
l = []
for t in range(19223):
    c.append(randint(0,226))

for n in range(19223):
    l.append(randint(0,226))

for h in range(19223):
    y.append(comparator(l[h], c[h]))

```

Acrescente a este programa o código necessário que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

O elemento da lista `y`, no índice 10659, é 1.

A soma dos elementos 0 da lista `y` é 77.

O elemento da lista `y`, no índice 263, é -2.

O elemento da lista `y`, no índice 1823, é -1.

A soma da lista `y` é -297.

Considere o programa, Python 3, que se segue. Implemente a função `longest_consecutive_sequence`, que retorna uma lista com a maior sequência de números consecutivos de uma dada lista. Implemente a função `consecutive_sequences`, que retorna uma lista com todas as sequências de números consecutivos da uma dada lista. Implemente a função `filtered_consecutive_sequences`, que filtra a lista anterior removendo as sequências repetidas, E finalmente, implemente a função `distinct_numbers_in_sequence`, que retorna, sem repetições, os números da lista anterior.

```
from random import randint
from random import seed

seed(1487572)

q = []

def longest_consecutive_sequence(nums):

def consecutive_sequences(nums):

def filtered_consecutive_sequences(nums):

def distinct_numbers_in_sequence(nums):


for p in range(2233):
    q.append(randint(733,7233))

j = longest_consecutive(q)
k = consecutive_sequences(q)
h = filtered_consecutive_sequences(k)
m = distinct_numbers_in_sequence(h)
```

Para testar o funcionamento das funções execute o seguinte código.

```
>>> teste = [100, 4, 200, 1, 1, 3, 2, 10, 11, 12, 13]
>>> print(longest_consecutive_sequence(teste))
[1, 2, 3, 4]
>>> print(consecutive_sequences(teste))
[[1, 2, 3, 4], [1, 2, 3, 4], [3, 4], [2, 3, 4],
 , [10, 11, 12, 13], [11, 12, 13], [12, 13]]
>>> xx = consecutive_sequences(teste)
>>> print(filtered_consecutive_sequences(xx))
[[1, 2, 3, 4], [2, 3, 4], [3, 4], [10, 11, 12, 13],
 , [11, 12, 13], [12, 13]]
>>> yy = filtered_consecutive_sequences(xx)
```

O resultado da função ***longest_consecutive_sequence***(**q**), é [6273, 6274, 6275, 6276, 6277, 6278, 6279].

O maior número de repetições do mesmo valor é 7.

Eliminar as sequências que se repetem na lista **k**, reduz o tamanho da mesma 0.95 vezes. Use a função **round()** com duas casas decimais.

Eliminar os valores que se repetem na lista **h**, reduz o tamanho da mesma 1.51 vezes. Use a função **round()** com duas casas decimais.

O elemento da lista **m**, no índice 619, é 1618.

Considere o programa, Python 3, que se segue. Este exercício é centrado nos temas listas e dicionários. Implemente as funções `get_max_key()` e `get_min_key()`, estas retornam a *key* que contem o valor máximo e mínimo do dicionário, respectivamente.

```
import random
from random import seed
from random import choice
from random import randint
from string import ascii_letters

seed(1885716)

z = []
for i in range(19000):
    z.append(choice((randint(0,410), choice(ascii_letters))))

def str_int_splitter(List):
    return [j for j in List if isinstance(j, str)],
    [j for j in List if isinstance(j, int)]

def create_dict(str_z, int_z):
    l = {}
    size_str_a = len(str_z)
    size_int_a = len(int_z)
    for i in range(size_str_a):
        if i < size_str_a:
            l[str_z[i]] = int_z[i % size_int_a]
        else:
            l[str_z[i]] = ''
    return l

def get_max_key():

def get_min_key():

class Dictionary:

    def __init__(self, dictionary):
        self.l = dictionary

    def __getitem__(self, var):
        if isinstance(var, str):
            return [value for key, value in self.l.items() if key
if isinstance(var, int):
            return [key for key, value in self.l.items() if value
else: return "None"]
```

Na lista `z`, existem mais letras que números.

O objeto `f`, tem tamanho menor que a lista `int_z`.

O output da função `f[102]`, é `r`.

O objeto `f` possui o valor de 77, no índice `b`. Adicione mais um valor ao mesmo índice indexando-o da seguinte forma - `f[b] = 225`, após esta manipulação o índice indicado passa a armazenar o valor 225.

A *key* do dicionário `l` que armazena o menor valor é `a`.

Considere o programa, Python 3, que se segue. Implemente a função `most_frequent()`, que devolve o número mais frequente de uma dada lista, a função `least_frequent()`, que devolve o número menos frequente de uma dada lista. Aconselha-se a exploração das funções Python `min()` e `max()`. Implemente a função `rotate_list()`, que roda os elementos de uma lista em torno do índice recebido no parâmetro k e finalmente a função `int_to_roman`, que transforma um número decimal em numeração romana, para esta função assuma que o output de um número negativo é igual ao do número positivo.

```
from random import randint
from random import seed

seed(1767673)

def int_to_roman(num):

def most_frequent(List):

def least_frequent(List):

def rotate_list(nums, u):

b = []
for c in range(19612):
    b.append(randint(37, 569))
```

Execute o seguinte código para testar o funcionamento das funções criadas.

```
>>> most_frequent([2,3,4,5,7,2,3])
2
>>> least_frequent([2,3,4,5,7,2,3])
4
>>> rotate_list([2,3,4,5,7,2,3],3)
[5, 7, 2, 3, 2, 3, 4]
>>> int_to_roman(653)
'DCLIII'
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

O elemento da lista **b**, no índice 11460, é 217.

O número mais frequente na lista **b** é 166.

O número menos frequente na lista **b** é 313.

O número da lista **b**, no índice 15161 em numeração romana, é CCCLXVI.

O output da função `rotate_list` para 5 elementos a começar no índice 11161 da lista **b**, com `u` igual a 4 é [307, 522, 325, 117, 208]

Considere o programa, Python 3, que se segue. Na função apresentada, a cada iteração do ciclo *for* é adicionada á lista **a** um número aleatório, dentro da gama apresentada.

```
from random import randint
from random import seed
seed(1623537)
a = []
def func1():
    global a
    x = range(19294);
    for j in x:
        a.append(randint(718,1552))

func1()
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

O elemento da lista **a**, no índice 786, é 1205.

O elemento da lista **a**, no índice 16663, é 1290.

O elemento da lista **a**, no índice 15474, é 1154.

O elemento da lista **a**, no índice 15567, é 1096.

O elemento da lista **a**, no índice 3237, é 1220.

Considere o programa, Python 3, que se segue. A lista **n** é constituída por pixels RGB, simulando uma imagem. Para responder às questões seguintes acrescente ao código do enunciado código que permita guardar em **o** a conversão da imagem original para tons de cinzento e guardar em **k** a imagens em tons de cinzento aplicado um filtro de **Threshold** com um limiar de 135. Para converter a imagem em tons de cinzento use a equação $0.2989 * R + 0.5870 * G + 0.1140 * B$, onde R, G e B são *red*, *green* e *blue*, arredonde o resultado a duas casas decimais usando a função built-in **round**

```
import numpy as np
from random import seed
from random import choice
from random import randint
from string import ascii_letters
seed(1338344)

n = []
o = []
k = []

for s in range(19054):
    pixel = [randint(0,255), randint(0,255), randint(0,255)]
    n.append(pixel)
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

Número de pixels BLUE maiores que 83 na imagem é 12731.

O elemento da lista **o**, no índice 16, é 60.17.

Número de pixels **brancos** após o filtro, é 8534.

Após todas as manipulações efetuadas a dimensão da imagem reduziu 4.0 vezes.

O elemento da lista **n**, no índice 18753, é [133, 245, 85].

Considere o programa, Python 3, que se segue. São apresentadas 3 listas, a lista **c** contém 9 números gerados aleatoriamente, a lista **n** contém 19091 números gerados aleatoriamente e a lista **d** contém o resultado da comparação das listas **c** e **n**, se o valor das duas listas, no mesmo índice, for igual é adicionado nesta nova lista o valor **True**, caso contrário, será adicionado **False**.

```
from random import seed
from random import randint
seed(1303994)

c = []
n = []
d = []
for v in range(9):
    n.append(randint(0,9))

for a in range(19091):
    c.append(randint(0,9))

for m in range(19091):
    if c[m] == n[(m + 1)%len(n)]:
        d.append(True)
    if c[m] != n[(m + 1)%len(n)]:
        d.append(False)
```

Acrescente a este programa o código necessário que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.
Indique se é verdadeiro ou falso.

O elemento da lista **c**, no índice 17743, é 2.

O elemento da lista **d**, no índice 757, é **True**.

O elemento da lista **c**, no índice 6196, é 6.

O elemento da lista **d**, no índice 1320, é **False**.

O elemento da lista **c**, no índice 4581, é 0.

Considere o programa, Python 3, que se segue. A classe M dispõe do método `__init__()` que inicia um objeto da classe do tipo matriz e o método `__repr__()` que faz print da matriz. Crie o método `row_times_column()`, que multiplica uma linha por uma coluna e vice-versa, recebe como parâmetros o índice da linha e coluna em questão, o método `matrix_cross_product()`, que devolve o produto cruzado de uma matriz, o método `std()`, que calcula o desvio padrão da matriz e finalmente o método `mult_by_itself()` que multiplica a matriz por si mesma.

```

from random import randint
from random import seed

seed(1838467)

l = []

class M:
    def __init__(self , numero_linhas , numero_colunas ):
        self.numero_linhas = numero_linhas
        self.numero_colunas = numero_colunas
        self.linhas = []
        for l in range(numero_linhas):
            linha = []
            for c in range(numero_colunas):
                linha.append(randint(0, 148))
            self.linhas.append(linha)

    def __repr__(self):

        resultado = "Matriz(" +str(self.numero_linhas) + ","
+ str(self.numero_colunas) + ")\n"
        for l in range(self.numero_linhas):
            for c in range(self.numero_colunas):
                resultado += str(self.linhas[l][c]) + " "
            resultado += "\n"
        return resultado

    def row_times_column(self , cl_1 , v1 , other_matrix , cl_2 , v2):

    def matrix_cross_product(self):

    def std(self):

    def mult_by_itself(self):

for b in range(19488):
    l.append(M(3,3))

```

O resultado da função `row_times_column(coluna, 2, 1[3958], linha, 2)` para matriz da lista 1, no índice 119, é 13631.

A primeira linha da matriz da lista 1, no índice 119, tem como valores: [31, 132, 26].

O produto cruzado da matriz da lista 1, no índice 737, é 4395.

O resultado da função ***mult_by_itself()*** para matriz da lista 1, no índice 849, é

Matriz(3,3)

23331	15833	8584
15833	18469	3597
8584	3597	3922

.

O desvio padrão da matriz da lista 1, no índice 1959, é 38.46.

Considere o programa, Python 3, que se segue. A função `random_string_generator` gera uma string de tamanho aleatório constituída por 3 caracteres, implemente a função `message_optimizer` que permite otimizar a mensagem, que torna uma mensagem com caracteres repetidos numa mensagem mais facilmente compreensível. Exemplo do que essa função tem de ser possível fazer, uma mensagem "aaaaadaaaafdddfdd" tem de tornar em "a5da3fd3f2d".

```

from random import seed
from random import choice
from random import randint
from string import ascii_letters
seed(1142344)

t = []

def random_string_generator():
    w = ""
    my_str = ""
    for e in range(1,3):
        my_str += choice(ascii_letters).lower()
    for f in range(randint(1,15)):
        idx = randint(1,len(my_str) - 1)
        w += choice(my_str)
    return w

for e in range(19782):
    p = random_string_generator()
    m = message_optimizer(p)
    t.append(m)

```

Acrescente a este programa o código necessário que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.
 Indique se é verdadeiro ou falso.

O elemento da lista \mathfrak{t} , no índice 12632, é $\mathfrak{f}\mathfrak{m}\mathfrak{f}4\mathfrak{m}7$.

O elemento da lista \mathfrak{t} , no índice 6613, é $\mathfrak{z}\mathfrak{m}3\mathfrak{z}\mathfrak{m}\mathfrak{z}$.

O elemento da lista \mathfrak{t} , no índice 7952, é $\mathfrak{h}2$.

O elemento da lista \mathfrak{t} , no índice 3722, é $\mathfrak{e}3$.

O elemento da lista \mathfrak{t} , no índice 1100, é $\mathfrak{t}2\mathfrak{j}\mathfrak{t}2\mathfrak{j}\mathfrak{t}\mathfrak{j}2$.

Considere o programa, Python 3, que se segue. Neste exercício é apresentada a classe `numpy` e algumas facilidades que esta apresenta perante a geração e manipulação de arrays n-dimensionais. Implemente a função `join_arrays(dimension, arr_1, arr_2)`, esta recebe uma dimensão e 2 arrays, e retorna a junção dos dois, caso a dimensão for diferente da suportada, é devolvido um array aleatório, caso seja necessário esta função deve de igualar o tamanho dos arrays recebido, preenchendo o array de menor dimensão com "1's".

```

import numpy as np
from random import choice
from random import shuffle
from numpy.random import seed
from numpy.random import randint
from string import ascii_letters

seed(1616638)

m = []
num_range = 19372
for k in range(num_range):
    m.append(randint(5, 191))

p = np.array([])
for k in range(19372):
    p = np.append(p, randint(5, 191))
z = np.array([])
for k in range(300):
    z = np.append(z, choice(ascii_letters))
g = np.array([])
for k in range(300):
    g = np.append(g, randint(5, 191))

def join_arrays(dimension, arr_1, arr_2):
    if dimension == '1d':
        #TO DO
    if dimension == '2d':
        #TO DO
    return np.hstack((np.ones(1), np.zeros(1), np.ones(1)))

e = join_arrays("2d", p, z)
o = join_arrays("2d", z, g)

```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

A função `array()` representa uma forma incorreta de criação de um numpy array vazio.

No numpy array `m`, não é possível ter esta dimensão `[11, 1761]`.

A execução da função `np.vstack((p, g))`, vai originar um numpy array de 2 dimensões.

O shape do numpy array `e` é `(2,)`.

O output da função `o[:, :]` é array constituído somente por números.

Considere o programa, Python 3, que se segue. A função `random_string_generator` gera uma string com caracteres aleatórios de tamanho aleatório e a função `random_number_generator` gera um número aleatório. Em cada iteração do último ciclo *for* presente no enunciado, é gerada uma string, caso o tamanho dessa string seja menor que 15 caracteres essa string é substituída por um número.

```
from random import seed
from random import choice
from random import randint
from string import ascii_letters
seed(1367269)

r = []

def random_string_generator():
    s = ""
    for i in range(randint(1,30)):
        s += choice(ascii_letters)
    return s

def random_number_generator():
    g = ""
    for o in range(randint(1,30)):
        g += str(randint(0,9))
    return g

for l in range(18856):
    k = random_string_generator()
    if len(k) < 15:
        k = random_number_generator()
    r.append(k)
```

Acrescente a este programa o código necessário que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas. Indique se é verdadeiro ou falso.

A função `rfunc1` e a função `r.func2` produzem o output igual.

O elemento da lista `r`, no índice `8191`, é `iyhHKtQzoftAWJYJKeAZbvcbx`.

O elemento da lista `d`, no índice `12847` após ter percorrido o segundo ciclo de `for`, é uma string.

O elemento da lista `r`, no índice `3509`, é `18336248684715708207`.

O elemento da lista `r`, no índice `7347`, é um numero.

Considere o programa, Python 3, que se segue. Implemente a classe `Vector3D`, esta possui o método `mult`, capaz de multiplicar dois vetores, o método `add`, capaz de adicionar dois vetores, o método `div`, capaz de realizar a divisão entre dois vetores e o método `sub`, capaz de subtrair dois vetores.

```
from random import seed
from random import randint
seed(1478846)

def act(vector1, vector2, val):
    if val == 1:
        return vector1.mult(vector2)
    elif val == 2:
        return vector1.add(vector2)
    elif val == 3:
        return vector1.div(vector2)
    elif val == 4:
        return vector1.sub(vector2)

g = []
for v in range(19512):
    v1 = Vector3D(randint(1,255), randint(1,255), randint(1,255))
    v2 = Vector3D(randint(1,255), randint(1,255), randint(1,255))
    g.append(act(v1, v2, randint(1,4)))
```

Acrescente a este programa o código necessário que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.
Indique se é verdadeiro ou falso.

O elemento da lista `g`, no índice 975, é (70, -105, -13).

O elemento da lista `g`, no índice 12259, é (350, 173, 349).

O elemento da lista `g`, no índice 10422, é (495, 10486, 6751).

O elemento da lista `g`, no índice 16426, é (1980, 21210, 1912).

O elemento da lista `g`, no índice 1438, é (2320, 297, 19305).

Considere o programa, Python 3, que se segue. Este exercício consiste na introdução aos ciclos *for* e *while*.

```
from random import randint
from random import seed

seed(1916894)

n = 19553
y = []
def while_cicle(m):
    n = m
    while(True):
        if n == 0: break
        print("function: while_cicle")
        n -= 1

def for_cicle(b):
    decision1 = randint(0, round(b / 2))
    decision2 = decision1 + 2
    decision3 = randint(decision2, b)
    values = []
    for idx in range(b):
        if idx < decision1:
            continue
        if idx == decision2:
            pass
        if idx >= decision3:
            break
        values.append(idx)
    return values

def print_indexes(num):
    for idx in range(num):
        print(idx)

y = for_cicle(257)
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

Dentro da função `while_cicle(21)`, a função `print("function: while_sicle")` é executada 21 vezes.

Os 3 `ultimos` valores da lista `y`, retornados pela função `for_cicle(b)`, são `[51, 52, 53]`.

Dentro da função `print_indexes(num)`, o `ultimo` valor que é apresentado na Python Shell pela função `print(idx)`, é 57.

A função `while_cicle(m)` não é um exemplo de um ciclo "do while".

Após executar a função `while_cicle(m)` o valor da variável `n`, é 19553.

Considere o programa, Python 3, que se segue. Implemente a função `most_frequent_num(nums, max_num)`, que retorna o número mais frequente da lista que tenha o valor inferior ao valor máximo fornecido, caso não tenha encontrado nenhum número, é retornado o valor "None", considere também que o valor 0 nesta questão não é considerado nem par nem ímpar.

```
import random
from random import randint
from random import seed

seed(1918616)

o = []
for e in range(19824):
    o.append(randint(0,357))

g = 0

def while_1():
    while g < 5:
        print(g + 1)

def while_2():
    while g < 5:
        if g == 4:
            g = 0
        print(g)
        g += 1

def while_3():
    global g
    while g < 5:
        print(g)
        g += 1
```

```
def most_frequent_num(nums, max_num):
```

Exemplo de output da função `most_frequent_num(nums, max_num)`:

```
>>> o = [11, 8, 15, 9, 0, 14, 9, 0, 4, 11, 8, 11, 11]
>>> most_frequent_num(o, 11)
8
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

A função `while_2()`, não é executada sem qualquer problema.

A função `while_2()`, não é um exemplo de um **loop** sem fim.

O número mais frequente da lista `o`, sabendo que o valor máximo tem de ser igual a 299, é 214.

Na lista `o`, existem 9916 números **ímpares**.

A função que consegue calcular os números **par** da lista `o`, pode ser:
`len([x for x in nums if x % 2 == 0 and x == 0])`.

Considere o programa, Python 3, que se segue. A função `random_string_generator()` gera uma string com caracteres aleatórios, o segundo ciclo *for* do código apresentado adiciona em cada iteração um valor inteiro a lista `o` e uma string a lista `l`, o terceiro ciclo *for* a cada iteração multiplica o elemento da lista `o` por 3 .

```
from random import seed
from random import choice
from random import randint
from string import ascii_letters

seed(1120117)

o = []
l = []
def random_string_generator():
    v = ""
    for b in range(randint(1,15)):
        v += choice(ascii_letters)
    return v

for t in range(18354):
    o.append(randint(0, 104))
    l.append(random_string_generator())

k = o

for t in range(55062):
    o[t%len(o)] = o[t%len(o)] * 3
```

Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas.

Indique se é verdadeiro ou falso.

A função `o[:-1:].sort()` e a função `o.sort(reverse=True)` produzem o output igual.

A função `l.sort(key=len, reverse=True)` organiza os elementos da lista `l` de forma crescente com base no tamanho da string.

O elemento da lista `o`, no índice 26, é 756.

A lista `o`, após o segundo ciclo de *for*, aumenta o valor de cada elemento 27 vezes.

Os 3 últimos elementos da lista `o`, podem ser apresentados da seguinte maneira `o[-3::]`.