



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 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, pretendendo espelhar o trabalho realizado durante a mesma e a sua reflexão.

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 defendendo o trabalho elaborado.

O presente trabalho 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 project 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	11
4.0.1 Criação de Exercícios Python	11
4.0.2 Aplicação	12
5 Validação e Testes	17
5.1 Exercícios	17
5.2 Diferenças nas Versões	20
5.3 Geração e Visualização de versões	22

6 Conclusões e Trabalho Futuro	25
Bibliografia	27

Lista de Figuras

4.1	Diagrama da Aplicação	12
4.2	Motor de Busca	13
4.3	Enunciado exercício	13
4.4	Gerar versões	14
4.5	Diferenças nas versões	14
4.6	Tokens alterados nas versões	15
4.7	Versões criadas	15
4.8	Versão exemplo	16
5.1	Alínea considerada correta	17
5.2	Função do enunciado	17
5.3	Alínea considerada correta	18
5.4	Indexação usada na validação alínea	18
5.5	Alínea considerada como correta	18
5.6	Esboço de código presente no ficheiro full_program.py	19
5.7	Código usadas nesta alínea para verificar o output	19
5.8	5 alínea proveniente de uma versão de exercício já criado	20
5.9	Simulação do output da 5 alínea na aplicação	21
5.10	Enunciado visualizado no Adobe Acrobat Reader DC e que foi gerado correndo os scripts em sequência	22
5.11	Enunciado visualizado e gerado na aplicação	23

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 são inseridas, 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. 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 posteriormente deve ser comparado com a resposta dada pelo aluno.

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, por ser desenhada especificamente para a estrutura dos exercícios criados, proporcionando ao utilizador várias facilidades na criação de novos exercícios e manipulação dos mesmos.

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 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 é apresentada 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 contem 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.

Capítulo 4

Implementação do Modelo

4.0.1 Criação de Exercícios Python

Durante a criação dos exercícios e geração das suas 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_transformations.py**, indicando todos os tokens que se pretendem mudar e para que valores.
5. Implementação de funções no script **make_transformations.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**.

4.0.2 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 trata do motor de busca da aplicação e *Manipulate Exercise* implementa todos os processos ligados á manipulação de ficheiros dos exercício, também ele contem uma instância de *Search Engine* para poder navegar dentro das diretorias do exercício e executar ficheiros Python.

Os scripts ***differenceFinder*** e ***string_formatter*** encarregam-se de re-tornar as diferenças entre versões.

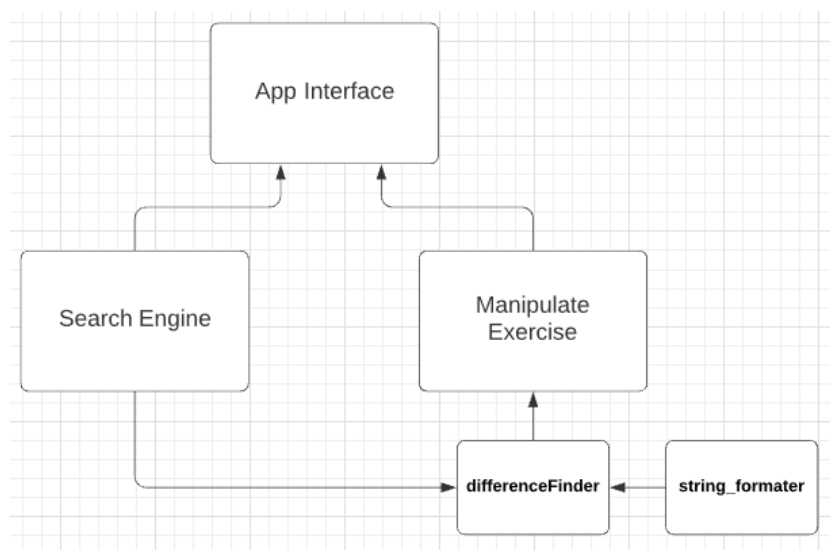


Figura 4.1: Diagrama da Aplicação

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

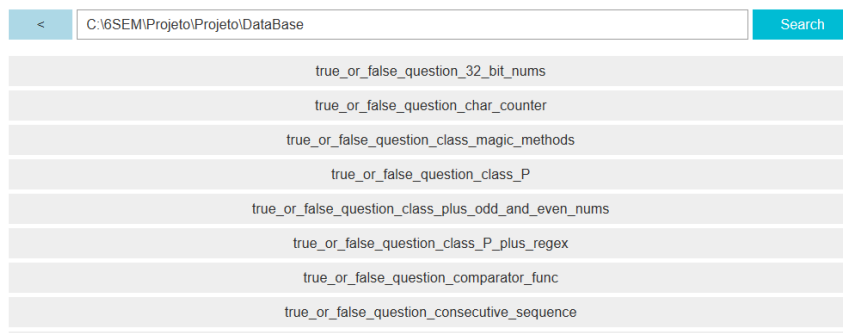


Figura 4.2: Motor de Busca

A página de exibição do exercício (figura 4.3) usa as imagens png geradas para mostrar o enunciado e as alíneas do mesmo.

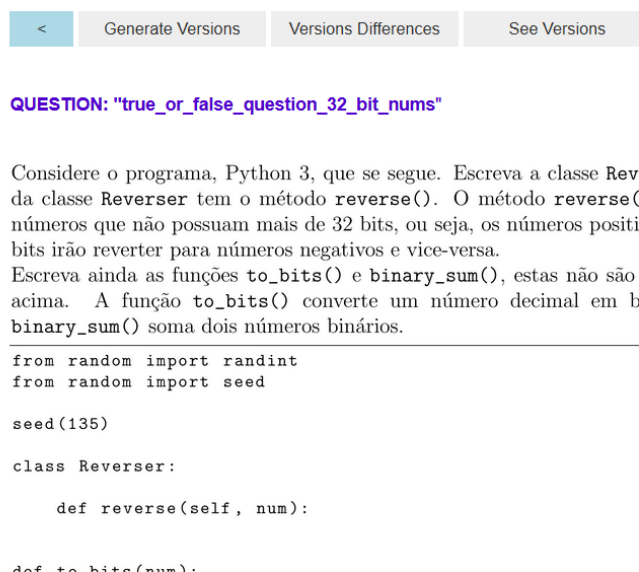
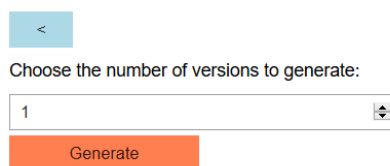


Figura 4.3: Enunciado exercício

Para gerar versões (figura 4.4) a aplicação executa os scripts "make_random_versions.py" e "make_pdfs.py" a partir do cmd.

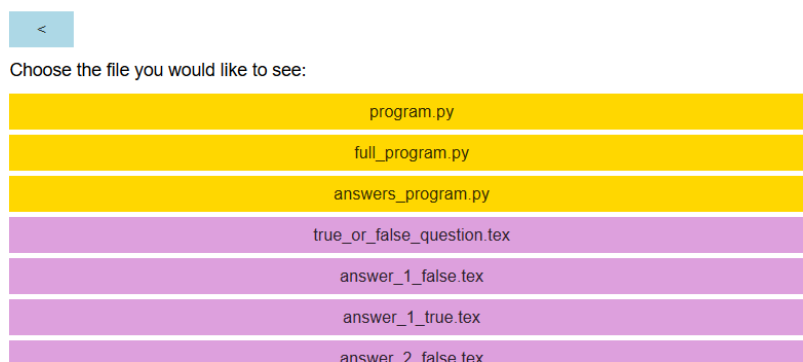
Nota: Em windows, para esta geração funcionar é necessário executar o Jupyter Notebook como administrador.



A light blue button with a left-pointing chevron (<) is positioned above the text "Choose the number of versions to generate:". Below this text is a text input field containing the number "1" and a small downward-pointing arrow icon on its right side. Directly beneath the input field is an orange button labeled "Generate".

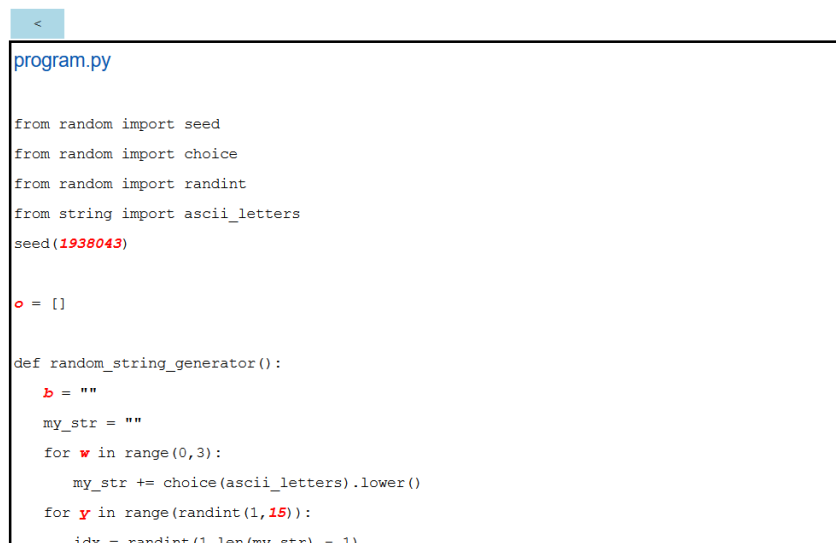
Figura 4.4: Gerar versões

Existem vários scripts 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 light blue button with a left-pointing chevron (<) is positioned above the text "Choose the file you would like to see:". Below this text is a list of file names, each on a separate horizontal bar. The first three bars are yellow and contain the text "program.py", "full_program.py", and "answers_program.py" respectively. The remaining four bars are purple and contain the text "true_or_false_question.tex", "answer_1_false.tex", "answer_1_true.tex", and "answer_2_false.tex" respectively.

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.

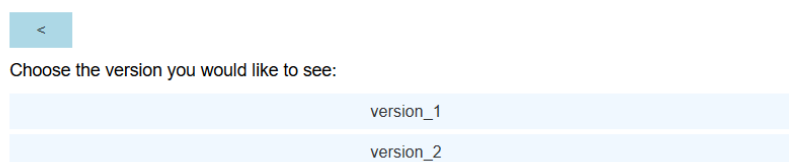


Figura 4.7: Versões criadas

**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. 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  
    return g
```

Figura 5.2: Função do enunciado

2. **Validação usando indexação**, existem alíneas que pedem, por exemplo, qual o valor da lista no índice X, 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 já necessitam alguma 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.

|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: 5 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{}}\textit{verb+o+[\textit{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/>