Programação II

Ciências ULisboa

Enunciado do trabalho — Ano lectivo 2019-20

Grupos de **3** alunos — Cotação: 3 valores

Data-limite de entrega: 2 de Junho às 18h00m

Instruções para a entrega:

- Entregar apenas este ficheiro: **imagem.py**
  [com informação extra no nome do ficheiro, explicado abaixo]
- Os ficheiros devem estar identificados nas linhas iniciais com o número do grupo e com o número e nome de cada um dos elementos do grupo; essa identificação deve ser feita usando comentários com o símbolo **#**.
- Antes de entregar o ficheiro, mudar o seu nome para **imagem _XXX_fcXXXXX.py** onde **XXX** é o *número do grupo* e **fcXXXXX** é a referência ao *número do aluno que submete o trabalho no moodle*.
  - exemplo: **`imagem_009_fc90900.py`** se for o aluno 90900 a submeter o trabalho no moodle, e o seu grupo for o nº 9
- Fazer upload do ficheiro no moodle, quando a página de entrega estiver aberta.

Requisitos do programa:

Assegure-se de que o módulo **imagem** *(após editado por si)* não contém erros sintácticos, e de que os programas de teste **cliente_de_teste-Imagem** e **cliente_de_teste-Album** executam correctamente com o Python versão 3.7 ou superior no ambiente Windows.

Os programas serão avaliados pela correcção, mas também pela estrutura do código e pela formatação. Não deverá ultrapassar as 80 colunas por linha. Comente o seu código onde isso for útil.

Tema — combinar imagens e arrumá-las em álbuns

Neste trabalho concretiza-se um tipo de dados **Imagem** que guarda uma imagem, bem como informação diversa associada a essa imagem. É possível combinar (justapor, fundir) diferentes Imagem's, bem como processar Imagem's individuais, nomeadamente aplicando-lhes filtros concretizados na classe **Filtro**. As Imagem's são originalmente obtidas de ficheiros. Após processamento pela aplicação informática, podem ser visualizadas no ecrã ou guardadas de novo em ficheiro.

A partir de Imagem's processadas, pode-se construir álbuns, concretizados na classe **Album**. Opcionalmente, um Album pode construir e carregar as suas próprias Imagem's a partir de ficheiros.

A 4ª e última classe definida pelo programador neste trabalho é **AlbumPlus**, que *estende* Album. Esta é uma versão especializada de Album, que, entre outras novidades, facilita o carregamento de pastas com ficheiros de imagem e facilita a realização de pesquisas complexas sobre o álbum.

Estrutura — diagrama de classes

A estrutura do serviço fornecido está resumida no diagrama de **classes**. Este contém todas as classes desenvolvidas pelo programador e definidas no ficheiro **imagem.py**, com os seus atributos e métodos. Por simplicidade, omite algumas classes utilitárias da biblioteca do Python. É aconselhável consultar o ficheiro **diagrama_de_classes.pdf** frequentemente durante o desenvolvimento.

Comportamento — especificação e contratos

O comportamento dos **objectos** das classes desenvolvidas, que será explorado pelas aplicações cliente, é definido de diferentes modos que se complementam:

- leitura dos contratos dos métodos (*docstrings*);
- descrição dada no enunciado do trabalho (este documento);
- senso comum.

Note que, no ficheiro **imagem.py**, se omite os contratos para certos métodos triviais como *setters* e *getters*, cujo nome indica claramente a função.

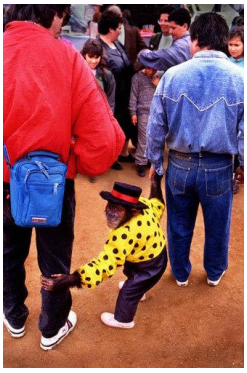
A confiança no programa desenvolvido aumentará se as classes passarem com sucesso os **testes** propostos no final, ou outros testes adequados, criados pelos alunos. O objectivo principal dos testes é ajudar a verificar se o comportamento desejado está bem concretizado no *software*.

É importante para o sucesso do trabalho reduzir a ambiguidade na descrição do comportamento desejado. Os resultados de testes fornecidos *também* podem ser úteis para isso ("testes como documentação").

Por último, a ambiguidade é minimizada dialogando com o *stakeholder* ou interessado, i.e., o utilizador cujos objectivos no uso do *software* devem ser satisfeitos. Para os efeitos deste trabalho, o regente da disciplina é o interessado de referência.

Expressões de Imagem's

Suponha que **chimpanze** e **casamento** são objectos Imagem que contêm as imagens abaixo.



Implementando os *métodos especiais* adequados na classe Imagem, é possível gerar novas imagens usando a sintaxe seguinte:

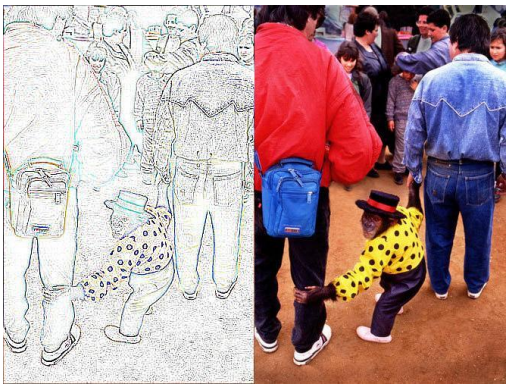
```
2 * (chimpanze + casamento) + ~chimpanze
```

(a operação unária de inversão **~** denota *obter o negativo da imagem*)

```
chimpanze & casamento
```

(mistura as imagens a 50% cada)

```
chimpanze & Filtro("CONTOUR") + chimpanze
```



(filtro de contorno aplicado ao chimpanzé)

Álbuns

Um objecto Album é um contentor de Imagem's, como referido na introdução ao tema.

A versão especializada AlbumPlus é a que oferece mais flexibilidade ao utilizador. Além dos *getters* e listagens triviais, permite:

- carregar imagens de muitas pastas, com apenas 1 instrução;
- fazer pesquisas aproximadas; por exemplo

  `umAlbumPlus.pesquisarImagens("ximpa", 3, etiquetas = ["animais"])`

  encontra e devolve as Imagem's que têm o nome (*excluindo sufixo*) **`"chimp"`**

  `['0001_chimp.jpg', '0002_chimp.jpg']` (números diferentes, mesmo nome)

  mas

  `umAlbumPlus.pesquisarImagens("ximpa", 2, etiquetas = ["animais"])`

  não encontra Imagem's respeitando o critério de pesquisa;

  no 1º caso o critério de pesquisa foi

  → procurar Imagem's com nome parecido com "ximpa", podendo diferir em até 3 caracteres, e contendo a etiqueta "animais" entre as suas etiquetas;

  no 2º caso a tolerância baixou para 2 caracteres de diferença, pelo que já não foram encontradas Imagem's.

Outro exemplo de uma pesquisa aproximada seria pedir todas Imagens de 2015 e 2016, quando se procura uma Imagem desse período mas não se lhe conhece a data/hora exacta. Se, além da data aproximada, se conhecer o nome aproximado da Imagem, ou alguma das suas etiquetas, a eficácia da pesquisa aumenta (pois diminui-se o número de falsos positivos).

Contador de Imagem's

O atributo de classe **Imagem.numeroDeImagens** é incrementado em 1 unidade de cada vez que um objecto Imagem é criado e devolvido por um método. A nova Imagem fica então com o seu atributo **numero** igual ao valor actualizado do contador da classe. Lembrando: uma Imagem pode ser criada por importação de ficheiro de imagem ou por manipulação interna. Todos os objectos Imagem, Album, e outros que interajam com a classe Imagem, têm acesso a este contador que tem portanto natureza global. As operações das várias classes estão desenhadas para que não sejam criadas Imagem's diferentes com o mesmo número, excepto em certas circunstâncias excepcionais (quais?...). Assim, as Imagem's são distinguidas individualmente por este número único, mesmo que tenham outros atributos repetidos. Da primeira vez que se executa num certo computador um programa que crie Imagem's, o contador é iniciado a zero. Cada sessão de execução de código cliente deve *terminar* com a invocação do método de classe **Imagem.guardarNumeroDeImagens()**, que guarda no ficheiro **numeroDeImagens.txt** o número total de Imagem's criadas até ao momento. Na próxima sessão, a classe **Imagem** começa por ler do ficheiro **numeroDeImagens.txt** o valor mais recente do contador **Imagem.numeroDeImagens**, e incrementa-o a partir daí. Se o ficheiro não for encontrado (por exemplo porque foi removido), o contador é reiniciado a zero.

Tecnologias

Os alunos exercitam conceitos e técnicas aprendidas em Programação 1 e 2, incluindo a programação orientada a objectos, o uso de módulos, o processamento de ficheiros, indexação, pesquisas, entre outros.
Adicionalmente, deverão *investigar* novas funcionalidades, sejam as que já estão presentes na tecnologia *core* de objectos Python, sejam as oferecidas por módulos externos.

Os módulos que precisam de ser importados devem ser instalados nos PCs onde o trabalho é desenvolvido e testado.
Em ambiente Windows, a instalação típica será com pip install <modulo>.
Caso o pip não esteja no path, pode-se instalar com:

```
python -m pip install pathlib
python -m pip install jellyfish
python -m pip install pillow
```

Segue-se um resumo das novas funcionalidades e dos módulos mais relevantes.

Biblioteca Pillow

É uma evolução da biblioteca **PIL** (Python Imaging Library).

Por razões de compatibilidade de código legacy, os imports devem ser feitos de PIL, e não de pillow.

O módulo principal de Pillow/PIL é o **Image**, onde encontramos a classe **PIL.Image.Image** com as respectivas operações essenciais. Esta classe representa imagens, tipicamente importadas de ficheiros de imagem, mas que podem depois sofrer transformações diversas. Importamos também o módulo **PIL.ImageOps**, que contém operações adicionais (só usamos a *invert*). **PIL.ImageFIlter** permite "fabricar" filtros à medida, mas dele apenas usamos um conjunto de filtros pré-configurados. **PIL.ExifTags** é um módulo auxiliar na interpretação de informação **exif** que alguns ficheiros de imagem contêm. Da vasta informação exif, apenas retemos os dados de data/hora, caso existam no ficheiro.

A classe PIL.Image.Image é encapsulada na nossa classe **Imagem**. Cada uma das nossas Imagem's tem uma PIL.Image.Image dentro de si. Assim, pode invocar todas as operações nativas de PIL.Image.Image, e pode realizar outras não previstas explicitamente na biblioteca Pillow.

Por simplicidade neste trabalho, optamos por aproveitar apenas um subconjunto reduzido das operações de PIL.Image.Image.

Referências:

https://automatetheboringstuff.com/2e/chapter19/

https://pillow.readthedocs.io/en/latest/

https://pillow.readthedocs.io/en/latest/reference/Image.html

https://stackoverflow.com/questions/4764932/in-python-how-do-i-read-the-exif-data-for-an-image

Módulo pathlib

Oferece classes que representam *paths* em sistemas de ficheiros como objectos, de forma muito conveniente e independente do sistema operativo. Usamos a classe **Path**.

Referências:

https://medium.com/@ageitgey/python-3-quick-tip-the-easy-way-to-deal-with-file-paths-on-windows-mac-and-linux-11a072b58d5f

https://docs.python.org/3/library/pathlib.html

Módulo os

Permite o acesso do programa Python a funções do sistema operativo (onde se incluem algumas que também são tratadas pelo módulo pathlib). Neste trabalho, é usado para conhecimento dos alunos e para a tarefa de obter a data/hora de modificação de um ficheiro.

Referência: https://docs.python.org/3/library/os.html

Módulo datetime

Não é novidade, pois foi usado em Programação 1. Contém várias classes úteis para representar datas, horas e intervalos de tempo. Usamos a classe **datetime.datetime**.

Referência: https://docs.python.org/3/library/datetime.html

Módulo pickle

Permite fazer a *serialização* de um objecto, ou seja, representá-lo por uma sequência de *bytes*, e registar essa sequência num ficheiro. Guarda-se o estado completo do objecto, de modo a o mesmo poder ser recriado mais tarde, a partir do ficheiro onde ficou registado. A esta operação inversa chama-se *des-serialização*.

Usamos o pickle para poder guardar Album's de maneira persistente em ficheiro, recriando-os na actual execução da aplicação informática ou numa futura execução.

Referências:

https://docs.python.org/3/library/pickle.html

https://wiki.python.org/moin/UsingPickle

Biblioteca jellyfish; distância de Levenshtein

Esta biblioteca reúne vários métodos para fazer comparação de strings através de funções distância, as quais medem quão próximas duas strings estão uma da outra por um determinado critério. Usamos apenas a distância de Levenshtein. Esta corresponde ao número *mínimo* de trocas, inserções ou remoções de caracteres, necessário para passar de uma string à outra.

Referências:

https://pypi.org/project/jellyfish/

https://jellyfish.readthedocs.io/en/latest/comparison.html#levenshtein-distance

https://en.wikipedia.org/wiki/Levenshtein_distance

__rmul__, __rand__, __matmul__, __rmatmul__

Estes métodos especiais, e alguns dos outros métodos especiais usados neste trabalho, destinam-se originalmente a suportar operações com tipos "numéricos" (`+`, `*`, `&`, `~`) ou "matriciais" (`@`). Aqui, forçamos a definição dos métodos especiais de modo a poder usar esses operadores na nossa sintaxe de expressões com Imagem's, com significado o mais próximo possível do original. Os contratos dos métodos no ficheiro **imagem.py** explicam o objectivo de cada um. As imagens da secção **Expressões de Imagem's** neste enunciado ilustram vários exemplos de expressões.

Assinala-se o caso especial dos métodos **__r…__**. Estes são invocados quando o operando esquerdo não suporta a operação correspondente, e os operandos são de tipos diferentes. Por exemplo, com **__mul__** apenas fica suportada a operação **Imagem * int**. Para a operação **int * Imagem** não levantar uma excepção, tem de se definir a operação "reflectida" **__rmul__**. O mesmo se passa com **__and__**, que pode "misturar" **Imagem** com **Filtro**, ficando o Filtro à direita ou à esquerda do operador `&`; e também com **__matmul__**, que suporta a multiplicação de uma **Imagem** por um factor de escala (um **float**), à direita ou à esquerda (operador `@` neste trabalho).

Referência:

https://moodle.ciencias.ulisboa.pt/pluginfile.php/257819/course/section/26710/T16-classes_resumo_de_metodos_especiais.html (*não contém todos os métodos especiais*)

https://docs.python.org/3.8/reference/datamodel.html#specialnames

https://docs.python.org/3.8/reference/datamodel.html#emulating-numeric-types

Herança ou Composição?

Na maioria das situações em que se coloca a escolha entre herança e composição, a composição é uma melhor opção. Facilita o desenho, o depuramento e a manutenção do código. Promove o encapsulamento e a independência entre diferentes partes do *software*.

Aqui, tratava-se de reutilizar a funcionalidade da classe **PIL.Image.Image** e dos objectos filtro do módulo **PIL.ImageFIlter**. Fazê-lo usando o mecanismo de herança (*estendendo* essas classes) levantava dificuldades técnicas e não trazia vantagens significativas. Assim, **Imagem** e **Filtro** são *wrappers* para essas classes: contêm objectos dessas classes como atributos.

Poderíamos também ter definido a classe **Album** como estendendo um tipo colecção pré-existente; por exemplo, **dict** ou **list** do Python. Mais uma vez, não se encontrou vantagem em usar herança. Album tem um dicionário como seu atributo, pelo que também usa composição.

Mas há herança explícita neste trabalho pois a classe **Album** é estendida por **AlbumPlus**. A segunda herda as propriedades e operações da primeira, e acrescenta alguma funcionalidade.

Trabalho a realizar pelos alunos

Há 2 partes, mas apenas a 1ª é entregue e avaliada. A 2ª parte é para treino dos alunos.

Parte 1

A primeira parte consiste em definir ("implementar") os seguintes métodos no ficheiro **imagem.py**:

- da classe **Imagem**:
  - __mul__
  - __invert__
  - __rand__
  - __str__
  - __matmul__
- da classe **Album**:
  - dump
  - __getitem__
  - __delitem__
  - carregarImagemPreProcessada
  - getImagens
- da classe **AlbumPlus**:
  - pesquisarImagens *(método parcialmente definido; completar a parte em falta)*

Não é preciso re-implementar os métodos que são fornecidos aos alunos.

Aliás, ***não é mesmo permitido alterar o código que é fornecido e que está finalizado***.

Pode definir outros métodos e funções auxiliares caso considere útil.

Todos os métodos a definir devem estar de acordo com a respectiva documentação, ou seja, ***é preciso respeitar os contratos.***

O código-fonte dos métodos pedidos deve ser inserido no ficheiro **imagem.py** onde actualmente estão as linhas da forma

```
pass # código a completar pelos alunos.
```

Testes da Parte 1

Os testes são opcionais, mas seria insensato não os fazer.

Se os métodos estiverem correctamente definidos, a execução dos módulos de teste **cliente_de_teste-Imagem** e **cliente_de_teste-Album** fornecidos aos alunos deve

- completar-se sem levantar qualquer excepção não tratada;
- enviar para o output standard texto *quase* igual ao que está registado nos ficheiros **output_de_teste-Imagem.txt** e **output_de_teste-Album.txt**, respectivamente;

a única diferença no output será nas *timestamps* de criação de algumas Imagem's geradas, e de criação de Album's;

- criar na pasta de trabalho o ficheiro **chimp_comFiltro_CONTOUR+chimp.JPEG**;
- criar na **pastaOut** o ficheiro **novoNome.PNG**;
- criar na pasta de trabalho o ficheiro *pickle* **guardaMeuAlbum.album**.

Para efeitos de depuramento (*debugging*), os alunos podem descomentar as instruções **`show()`** no cliente. Verão então aparecer no ecrã as imagens criadas pelo programa.

Avaliação do output

Nas fases finais do trabalho, pode-se guardar em ficheiros de texto o output obtido e verificar se o mesmo já é igual ao pretendido, usando o comando **`FC`** (**File Compare**) numa janela de comandos do Windows.

Para guardar o output do trabalho dos alunos:

```
C:\...\work>python cliente_de_teste-Imagem.py > resultImagem.txt
C:\...\work>python cliente_de_teste-Album.py > resultAlbum.txt
```

A sintaxe do comando **`FC`** é aqui ilustrada com ficheiros de texto simplificados. Neste caso, **`output.txt`** e **`output_2.txt`** são iguais, mas **`output_3.txt`** difere numa linha.

```
C:\...\pasta_auxiliar>FC output.txt output_2.txt
Comparing files output.txt and OUTPUT_2.TXT
FC: no differences encountered

C:\...\pasta_auxiliar >FC output.txt output_3.txt
Comparing files output.txt and OUTPUT_3.TXT
***** output.txt
Um texto assim.
Um texto assim assim.

***** OUTPUT_3.TXT
Um texto assim.
Um texto assim assado.

*****
```

Nota sobre o formato do output

Uma vez que é examinado o *output* de um programa que depende de código escrito pelos alunos (apenas na parte 1), determina-se que

> *a resolução apresentada pelos alunos deve obedecer aos exemplos dados nos ficheiros* ***output_de_teste-Imagem.txt*** *e* ***output_de_teste-Album.txt,*** *respectivamente; exceptua-se as timestamps de criação de Imagem's geradas, e de criação de Album's.*

Parte 2

Na segunda parte os alunos criam uma pequena aplicação que

- tem uma interacção *user-friendly* com o utilizador;
- incluindo um menu de opções;
- explora a funcionalidade das classes;
- antes de a aplicação terminar a sua execução, invoca (automaticamente, sem a intervenção do utilizador) o método **Imagem.guardarNumeroDeImagens()**, de modo a manter a contagem de Imagem's actualizada.