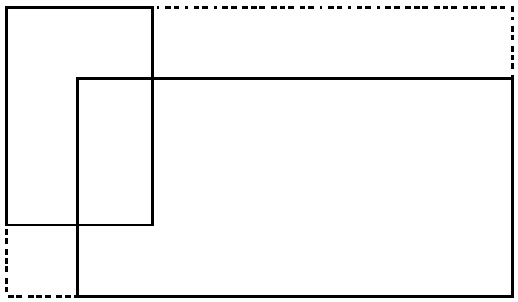
Folha de apoio à aula prática 10 (exercícios para praticar antes da aula).

10.1 Classe para retângulos. Implemente a classe `Rectangle` com as seguintes características:

- O contrutor tem como argumentos dois pontos extremos opostos do retângulo; por exemplo, deve permitir criar o retângulo com os pontos extremos (1,2), (5,2), (5,6), (1,6) com `r = Rectangle(5,6,1,2)`, e também com `r = Rectangle(1,2,5,6)`.
- A representação como *string* deve devolver, entre parêntesis retos, os pontos correspondentes aos cantos inferior esquerdo e superior direito, por essa ordem. Cada um desses pontos deve estar dentro de parêntesis curvos; para o exemplo anterior, `str(r)` deverá resultar na *string* `[(1,2),(5,6)]` (sem espaços e por essa ordem).
- O operador de adição deve resultar num objeto da classe `Rectangle` que inclui os dois retângulos operandos; por exemplo:

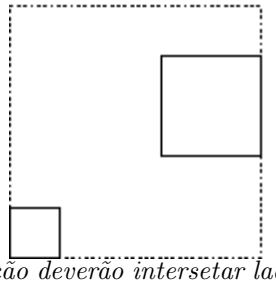
```
>>> r = Rectangle(3, 5, 1, 2)
>>> s = Rectangle(2, 4, 8, 1)
>>> print(r + s)
[(1,1),(8,5)]
```



10.2 Classe para quadrados. Implemente a classe `Square` com as seguintes características:

- O contrutor tem como argumentos `x, y, d`, em que `(x,y)` é o canto inferior esquerdo do quadrado e `d` é o tamanho do lado; por exemplo, deve permitir criar um quadrado com os pontos extremos (1,2), (5,2), (5,6), (1,6) com `s = Square(1,2,4)`.
- A representação como *string* deve devolver, entre parêntesis retos, os pontos correspondentes aos cantos inferior esquerdo e superior direito, por essa ordem. Cada um desses pontos deve estar dentro de parêntesis curvos; por exemplo, `str(s)` deverá resultar na *string* `[(1,2),(5,6)]` (sem espaços e por essa ordem).
- O operador de adição deve resultar num objeto da classe `Square` que inclui os dois quadrados operandos; por exemplo:

```
>>> s = Square(2, 1, 1)
>>> r = Square(5, 3, 2)
>>> print(r + s)
[(2,1),(7,6)]
```



Nota: O lado esquerdo e a base do quadrado resultante da adição deverão intersetar lados de quadrados operandos.

10.3 Classe para conjunto de palavras.
Implemente a classe `SetOfWords` para representar um conjunto de palavras, com as seguintes características:

- o construtor não tem argumentos (além de `self`)
- o método `add(self, word)` junta ao conjunto a palavra `word`
- o método `delete(self, word)` remove do conjunto a palavra `word`, caso esteja presente; caso contrário, não faz nada
- o método `size(self)` retorna o número de palavras contidas no conjunto
- não há distinção entre maiúsculas e minúsculas
- pode assumir que todas as palavras são compostas apenas por carateres alfabéticos ASCII (maiúsculos ou minúsculos).

Por exemplo:

```
>>> s = SetOfWords()
>>> s.add("Maria")
>>> s.add("Manel")
>>> s.add("MANEL")
>>> s.add("manel")
>>> print(s.size())
2
```

10.4 Seja $x = 10$ e $y = 3$. Verifique que cada uma das seguintes expressões dá origem a um erro de execução. Explore com pode tratar esse erro com a instrução `try`.

(a) `x / (y-3)`
(b) `x // (y-3)`
(c) `x % (y-3)`
(d) `log(y - x)`
(e) `sqrt(x / (y-5))`
(f) `sqrt(x / (y-3))`

10.5 Verifique no site `https://docs.python.org/3/library/exceptions.html` quais são as exceções prédefinidas para as operações que se seguem. Para cada uma delas, crie uma situação em que a exceção ocorra.

(a) operações matemáticas;
(b) indexação de listas;
(c) acesso a chaves de dicionários.

10.6 Implemente uma classe de dicionários para valores numéricos, em que o valor por omissão é dado no construtor. Se mais tarde se tentar aceder a chaves que não existem, é retornado esse valor. Segue-se um exemplo de utilização.

```
>>> d = MyDict(7)
>>> d[3]
7
>>> d[3] = 10
>>> d[3]
10
```

Sugestão: sobreponha os métodos `__getitem__(self, key)` e `__setitem__(self, key, value)`.
Nota: utilizar *herança* (de que não se falou nestas aulas) ajudaria a implementar uma classe mais funcional.

10.7 Implemente uma versão da função `factorial` que origina uma exceção `ValueError` se o argumento não for válido. Teste-a com inputs desses tipos, bem como com inputs válidos.

10.8 Movimento num reticulado. Implemente a classe `Grid` para representar um reticulado com o canto inferior esquerdo na origem, com as seguintes características:

- O construtor tem como argumentos a largura e altura do reticulado, e o ponto x, y onde inicialmente nos encontramos, dentro desse reticulado; permite fazer, por exemplo:

  ```
  >>> g = Grid(3,2,1,1)
  ```

  criando um objeto com largura 3, altura 2, e posição inicial no ponto (1,1).

- A representação como *string* deve devolver, entre parêntesis e sem espaços, o ponto onde nos encontramos; no exemplo anterior:

  ```
  >>> print(g)
  (1,1)
  ```

- Os métodos `up`, `down`, `left`, `right` permitem *"movimentarmo-nos"* no reticulado; por exemplo:

  ```
  >>> g.up()
  >>> print(g)
  (1,2)
  >>> g.left()
  >>> print(g)
  (0,2)
  ```

- Quando saímos do reticulado, deverá ser lançada uma excepção `OverflowError`, com a mensagem de erro:
  - `too far up` se ultrapassamos o topo do reticulado
  - `too far right` se ultrapassamos o lado direito do reticulado
  - `too far left` se ultrapassamos o lado esquerdo do reticulado
  - `too far down` se descemos abaixo da base do reticulado

Por exemplo:

```
>>> g = Grid(3,2,1,1)
>>> g.up()
>>> print(g)
(1,2)
>>> g.up()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 12, in up
OverflowError: too far up
```

10.9 Avaliador de funções. Implemente a função `evaluator(xs, f)` que, dada uma lista de valores numéricos `xs` e uma função `f`, cria uma lista com essa função avaliada para todos os valores de `xs`. Caso essa avaliação gere uma exceção `ZeroDivisionError`, o valor a colocar na lista deverá ser o objecto nulo de Python, `None`.

Por exemplo, se definirmos a seguinte função:

```
def f(x):
    return 1/x
```

deve-se obter o resultado seguinte:

```
>>> evaluator([0,1,2,3], f)
[None, 1.0, 0.5, 0.3333333333333333]
```

Na aula terá exercícios de avaliação contínua baseados nesta folha.