▷ **10.1** Defina uma função

```
mediana(a: int, b: int, c: int) -> int
```

que determina a *mediana* de 3 valores inteiros a, b, c, isto é, o valor no meio quando os colocamos por ordem crescente. Exemplos:

```
>>> mediana(3, 1, 2)
2
>>> mediana(1, 3, 3)
3
```

Escreva mais alguns exemplos e teste-os usando o *doctest*.

▷ **10.2** No Totoloto sorteiam-se seis números distintos de 1 a 49 (mais um "número da sorte" que vamos ignorar neste exercício). Escreva uma função `acertos(chave, aposta)` para contabilizar os números certos numa aposta; a chave e a aposta são listas de inteiros por ordem arbitária, mas que não contêm valores repetidos. Exemplo:

```
>>> acertos([25, 49, 27, 3, 17, 33], [33, 9, 19, 49, 7, 40])
2
```

Use `asserts` para exprimir as pré-condições da função: a chave e a aposta são listas sem valores repetidos.

10.3 Escreva uma função `bagdiff(xs,ys)` que calcula a *diferença* de duas listas: removemos da lista xs os elementos de ys psela ordem indicada e na multiplicidade correspondente. Exemplos:

```
>>> bagdiff([3, 2, 1, 2, 3, 3], [2, 2, 3])
[1, 3, 3]
>>> bagdiff([3, 3, 2], [1, 2, 3])
[3]
```

Começe a sua definição por escrever mais alguns casos de teste e verifique a sua implementação usando o módulo *doctest*.

10.4 Escreva uma função `remadj(xs)` para remover de uma lista elementos adjacentes repetidos, isto é, tais que `xs[i] == xs[i+1]`. O resultado da função deve ser uma nova lista; a lista original não deve ser modificada. Exemplos:

```
>>> remadj([2,2,3,1,4,4])
[2, 3, 1, 4]
>>> remadj(['a','b','b','b','a'])
['a', 'b', 'a']
```

Note que, ao contrário dos exemplos apresentados na aula teórica, esta função não deve eliminar *todos* os repetidos. Começe a sua definição por escrever mais alguns casos de teste. Teste a sua implementação usando o módulo *doctest*.

10.5 Escreva uma função

```
transposta(A: List[List[int]]) -> List[List[int]]
```

que transforme uma matriz A de dimensões $n \times m$ na sua *transposta* A^T com dimensões $m \times n$, i.e., ${A^T}_{ij} = A_{ji}$ para todos i, j. Começe a sua definição por escrever alguns casos de teste, como por exemplo:

```
>>> transposta([[1, 2],[3, 4]])
[[1, 3], [2, 4]]
```

10.6 Uma matriz quadrada A está em forma triangular superior se todos os coeficientes abaixo da diagonal são zero, isto é, se $A_{ij} = 0$ para todos i, j tais que $j < i$. Contudo, por causa da posibilidade de erros de arrondamento, vamos considerar como *zero* os números inferiores em valor absoluto a uma "tolerância" ϵ; a condição será então $|A_{ij}| < \epsilon$ para todos i, j tais que $j < i$.
Escreva a definição duma função

```
triang_sup(A: List[List[float]], eps: float) -> bool
```

que verifica a condição acima. O resultado deve ser um valor lógico (`True`/`False`).

10.7 Pretende-se escrever um programa que decomponha uma quantia usando o *menor número possível de moedas* do sistema Euro (2€, 1€, 50c, 20c, 10c, 5c, 2c, 1c). Por exemplo, podemos decompor 3.45€ usando 5 moedas: 2€ + 1€ + 20c + 20c + 5c; este é o menor número de moedas para esta quantia.
A ideia do algoritmo de decomposição é: começe por escolher a *maior moeda* cujo valor é *menor ou igual* ao total; retiramos o valor seu valor ao total, e repetimos o processo para o restante até chegar a zero.
Implemente este algoritmo como uma função

```
decompor(quantia: int) -> List[int]
```

cujo resultado é a lista de moedas. Para evitar erros de arrendamento, deve representar a quantia e os valores de moedas usando inteiros de cêntimos:

```
>>> decompor(345)
[200, 100, 20, 20, 5]
```

Escreva mais alguns casos de teste e use o *doctest* para testar a sua solução.

10.8 O *Sudoku* é um *puzzle* lógico que consiste em preencher posições vazias numa grelha 9×9 com números de 1 a 9 tal que cada linha, cada coluna e cada quadrado 3×3 contenha todos os números (1 a 9). Pretende-se que escreva uma função que verifica se uma grelha é uma solução:

```
sudoku(grelha: List[List[int]]) -> bool
```

O resultado deve ser `True` se a grelha está totalmente preenchida corretamente e `False` caso contrário. Pode consultar a página da Wikipedia para obter alguns exemplo de soluções: `https://pt.wikipedia.org/wiki/Sudoku`