Folha de apoio à aula prática 7 (exercícios para praticar antes da aula).

7.1 Defina uma função `forte(passwd)` que verifica se uma palavra-passe, dada pela cadeia de caracteres `passwd`, é forte. Considera-se que a palavra-passe é forte se tiver 8 caracteres ou mais, e incluir pelo menos uma letra maiúscula, uma letra minúscula e um algarismo. O resultado deve ser um valor lógico (`True` ou `False`).

```
>>> forte('9EwL56')
False
>>> forte('HXKW1393')
False
>>> forte('ffu4G7Fghjk')
True
```

7.2 Escreva uma função `ocorrencias(txt,c)` que retorna uma lista com os índices das ocorrências de um caracter `c` na cadeia `txt`. Por exemplo:

```
>>> ocorrencias('banana', 'a')
[1, 3, 5]
```

7.3 Defina a função `divisores(n)` que retorna uma lista com todos os divisores do inteiro `n`. Por exemplo:

```
>>> divisores(12)
[1, 2, 3, 4, 6, 12]
```

7.4 Recorde que um número inteiro d é um *divisor próprio* de n se o resto da divisão de n por d for zero e d for inferior a n. Escreva a função `div_prop(n)` que calcula a lista dos divisores de próprios de `n`, por ordem crescente. Por exemplo, `div_prop(12)` retorna `[1, 2, 3, 4, 6]`.

7.5 Escreva a função `maximo2(xs)` que calcula o segundo maior valor distinto numa lista `xs`. Verifique que o procedimento retorna o valor correcto quando o maior valor ocorre mais do que uma vez. Exemplos:

```
>>> maximo2([3, -2, 1, 0, -2, 1])
1
>>> maximo2([1, 3, 2, 3, 0])
2
```

7.6 Um número inteiro é *perfeito* se for igual à soma dos seus divisores próprios. Exemplo: 6 é perfeito porque $6 = 1 + 2 + 3$; mas 10 não é, porque $10 \neq 1 + 2 + 5$. Escreva uma função `perfeito(n)` que testa se n é perfeito ou não; o resultado deve ser um valor lógico.

7.7 Escreva uma função `repetidos(lista)` que testa se há elementos repetidos numa lista; o resultado deve ser um valor lógico. A sua função deve funcionar com listas de vários tipos (e.g. de números ou de cadeias de carateres). Exemplos:

```
>>> repetidos(['ola', 'ole', 'abba', 'ole'])
True
>>> repetidos([3, 2, -5, 0, 1])
False
```

7.8 *Universal Product Code* (UPC) é um standard para códigos de produtos, em que o algarismo à direita é um dígito de controlo (*check digit*) calculado da seguinte forma. Considerando todos os algarismos exceto esse último:

- Adiciona-se os algarismos em índices pares (0, 2, 4, ...) e multiplica-se o resultado por 3.
- Adiciona-se a esse valor cada um dos algarismos em índices ímpares (1, 3, 5, ...).

- Seja r o resto da divisão desse resultado por 10.
  - Caso r seja 0, o dígito de controlo é 0;
  - Caso contrário, o dígito de controlo é $10 - r$.

Implemente a função `calc_check_digit(code)` que, dada uma *string* `code` só com algarismos retorna o dígito (um carater) a juntar a essa `string` para formar um UPC.

Por exemplo, `calc_check_digit("12345")` deverá retornar

$$10 - ((1 + 3 + 5) \times 3 + (2 + 4))\%10 = 10 - 33\%10 = 7$$

7.9 *International Standard Book Number* é um código de 10 dígitos para identificar livros. Os primeiros nove dígitos são algarismos, e o último é um dígito de controlo calculado da seguinte forma. Considerando todos os algarismos exceto esse último:

- Determina-se a posição de cada algarismo em relação à direita; ou seja, arbitra-se a posição 1 para o dígito de controlo, 2 para o algarismo à sua esquerda, e assim sucessivamente.
- Excluindo o dígito de controlo, multiplica-se cada algarismo pela sua posição e faz-se a soma destes valores; seja n o resultado.
- Um ISBN é válido se a soma de n com o dígito de controlo for divisível por 11. Para isso, o dígito de controlo por vezes tem ser igual a 10; nesse caso, representa-se por `"X"`.

Implemente a função `is_isbn(code)` que, dada uma *string* `code` só com algarismos e, eventualmente, um `"X"` à direita, retorna `True` se `code` for um identificador ISBN válido, `False` no caso contrário. Por exemplo, o código `0-19-510756-X` é identificador ISBN válido porque

$$10 + 6 \times 2 + 5 \times 3 + 7 \times 4 + 0 \times 5 + 1 \times 6 + 5 \times 7 + 9 \times 8 + 1 \times 9 + 0 \times 10 = 187$$

e 187 é divisível por 11. Assim, `is_isbn("019510756X")` deverá retornar `True`.

7.10 O algoritmo de Luhn é um método utilizado para validar números de identificação. Entre muitas outras aplicações, é utilizado para determinar um dígito de controlo em números de cartões de crédito. Dado um código numérico (sem dígito de controlo), o algoritmo consiste no seguinte:

- Define-se a posição 1 como a do algarismo mais à direita;
- O algarismo à sua esquerda tem a posição 2, o seguinte a posição 3, e assim sucessivamente;
- Para cada algarismo em posições ímpares: multiplica-se o seu valor por dois e, se o resultado for maior do que 9, subtrai-se 9;
- Soma-se esse valores, e adiciona-se cada um dos valores das posições pares; seja n o resultado.
- O dígito de controlo (a colocar à direita do código) é o resto da divisão de $9 \times n$ por 10.

Implemente a função `calc_luhn_digit(code)` que, dada uma *string* `code` só com algarismos, retorna o dígito de controlo de Luhn (um carater) a juntar à direita de `code`.

Por exemplo, `calc_luhn_digit("12345")` deverá retornar

$$9 \times (((2 \times 5 - 9) + (2 \times 3) + (2 \times 1)) + (4 + 2))\%10 = 135\%10 = 5$$

Na aula terá exercícios de avaliação contínua baseados nesta folha.