▷ **6.1** Defina uma função `forte(passwd)` que verifica se uma palavra-passe (dada como uma cadeia de carateres) tem 8 carateres ou mais e pelo menos uma letra maiúscula, uma letra minúscula e um algarismo. O resultado deve ser um valor lógico (`True` ou `False`).

6.2 Defina uma função `dup_vogais(txt)` que duplica as vogais (letras 'a', 'e', 'i', 'o' e 'u', minúsculas ou maíusculas e sem acentos) duma cadeia de carateres; o resultado deve ser uma nova cadeia. Exemplo:

```
>>> dup_vogais('Abracadabra!')
'AAbraacaadaabraa!'
```

6.3 Recorde que um número inteiro d é *divisor* de n (ou equivalente, n é *múltiplo* de d) se e só se o resto da divisão de n por d for zero.

(a) Escreva uma função `divisores(n)` que calcula a lista dos divisores de n inferiores a n, por ordem crescente.
Exemplo: `divisores(12)` dá `[1, 2, 3, 4, 6]`

(b) Um número inteiro é *perfeito* se for igual à soma dos seus divisores. 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.

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

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

▷ **6.5** Escreva uma função `repetidos(lista)` que testa se há (pelo menos) dois elementos iguais numa lista; o resultado deve ser um valor lógico. A sua função deve funcionar com listas de vários tipos (e.g. números ou cadeias de carateres). Exemplos:

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

6.6 Escreva uma função `palavras(txt)` que retorna a lista das palavras na cadeia de caracteres `txt`. As palavras devem incluir apenas letras maiúsculas ou minúsculas; assuma ainda que a cadeia não tem letras acentuadas. Exemplo:

```
>>> palavras("---A Maria tinha um cordeirinho?")
['A', 'Maria', 'tinha', 'um', 'cordeirinho']
```

6.7 (T) Defina uma função `texto(n)` para converter um inteiro positivo inferior a um milhão para texto em português. Alguns exemplos:

```
>>> texto(21)
'vinte e um'
>>> texto(1234)
'mil duzentos e trinta e quatro'
>>> texto(123456)
'cento e vinte e três mil quatrocentos e cinquenta e seis'
```

Sugestão: começe por definir funções auxiliares para converter para texto número inferiores a 100 e 1000.

▷ **6.8** O *triângulo de Pascal* é constituido pelos valores $\binom{n}{k}$ das combinações de n em k em que n é a linha e k é a coluna. As primeiras 5 linhas do triângulo são:

$$\begin{array}{ccccccc} & & & 1 & & & (n=0) \\ & & 1 & & 1 & & (n=1) \\ & 1 & & 2 & & 1 & (n=2) \\ 1 & & 3 & & 3 & & 1 \quad (n=3) \\ 1 & 4 & & 6 & & 4 & 1 \quad (n=4) \end{array}$$

Para construir o triângulo de Pascal podemos usar as seguintes igualdades:

$$\binom{n}{0} = \binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad \text{se } n > 0 \text{ e } 0 < k < n$$

Ou seja: exceptuando os extremos, cada valor numa linha é obtido pela soma de dois valores na linha anteiror.

Escreva uma função `pascal(n)` cujo resultado é uma lista com os coeficientes da n-ésima linha do triângulo de Pascal. Por exemplo:

```
>>> pascal(4)
[1, 4, 6, 4, 1]
```

6.9 Numa turma com 25 alunos, qual a probabilidade de que o aniversário de pelo menos dois deles seja no mesmo dia? E se forem 50 alunos? O *paradoxo dos aniversários* é que esta probabilidade é muito maior do que parece ao senso comum.[1] Vamos estimar experimentalmente a probabilidade de aniversário comum com n alunos escrevendo um programa que repetidamente gere n dias aleatoriamente (de 1 a 365) e verifique a ocorrência de repetições.

Use o módulo `random` para gerar aleatoriamente dias e o exercício 6.5 para testar repetições. Para um grande número de experiências, a frequência relativa dá um valor aproximado da probabilidade.

[1] `http://en.wikipedia.org/wiki/Birthday_paradox`.