3.1 Consider the *Collatz sequence* [1]: we start with a given positive integer n; each new value is obtained from the previous one:

- if n is even: we divide n by two and continue;
- if n is odd: we multiply n by 3, add 1, and continue;
- we stop when $n = 1$.

Example: for an initial value of $n = 6$, the generated values are $6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$.

Write a function `collatz(n)` that determines this sequence for a given integer; the result should be the list of generated values. Example:

```
>>> collatz(6)
[6, 3, 10, 5, 16, 8, 4, 2, 1]
```

3.2 A number n is said to be triangular if $n = 1 + 2 + \cdots + k$ for some natural k. The first five triangular numbers are $1, 3, 6, 10$ and 15. Write a function `triangular(n)` whose result is `True` or `False` depending on whether n is triangular or not.

Suggestion: perform a loop that calculates $1+2+\cdots+k$ for successive values of k while the sum does not exceed n.

3.3 Write a definition of the function `only_letters(txt)` that tests whether a string contains only uppercase or lowercase letters (without accents). The result should be `True` or `False`. Examples:

```
>>> only_letters("Abracadabra")
True
>>> only_letters("Hello, world!")
False
```

3.4 Write a definition of the function `filter_letters(txt)` that returns a string containing only the uppercase or lowercase letters from the string `txt`. Example:

```
>>> filter_letters('Hello!, -- said he...')
'Hellosaidhe'
```

3.5 Write a function `reverse(txt)` that returns the string given in reverse order. For example:

```
>>> reverse('Hello World!')
'!dlroW olleH'
```

[1] Proving that this process terminates or not for *all* $n > 0$ is still an open mathematical problem. More information at `http://en.wikipedia.org/wiki/Collatz_conjecture`

3.6 A string is a *palindrome* if the sequence of characters read from left to right and from right to left are exactly the same. Example: `"reviver"` is a palindrome.

Write a definition of the function `palindrome(txt)` that checks if a string is a palindrome; the result should be `True` or `False`.

3.7 More generally, a string is a palindrome if it reads the same way in both directions considering only the letters (i.e., ignoring other characters such as spaces, punctuation marks, etc.) and considering uppercase and lowercase letters equivalent. Thus, the following strings are palindromes:

```
"Amora me tem aroma."
"Madam, I'm Adam."
"A man, a plan, a canal:  Panama"
```

Write a function `palindrome(txt)` to test whether a string `txt` is a palindrome in this more general sense.

Hint: You can solve this problem by combining the `lower()` method for strings and the solutions to exercises 3.4 and 3.6.

3.8 Write a function `remove_spaces(txt)` that removes two or more consecutive spaces in a string `txt`, replacing them with a single space; other characters should remain unchanged. Example:

```
>>> remove_spaces(' Hello,     World   !')
' Hello, World !'
```

3.9 Write a function `arithmetic_mean(xs)` that, given a list of n numerical values `xs`, returns the arithmetic mean of its values, that is,

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}.$$

3.10 Write a function `geometric_mean(xs)` that, given a list of n numerical values `xs`, returns the geometric mean of its values, that is,

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}.$$

3.11 Write a function `standard_deviation(xs)` whose result is the sample standard deviation of a list of n values, that is,

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2},$$

where $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ is the arithmetic mean of the values. You may assume that $n > 1$.

3.12 Write a function `count_in_range(xs, a, b)` whose result is the count of the values in the list `xs` that are between a and b inclusive. You may assume that $a \leq b$.

3.13 Write a function `repeated(lst)` that tests if there are (at least) two equal elements in a list; the result should be a boolean value. Your function should work with lists of various types (e.g. numbers or strings). Examples:

```
>>> repeated(['hello', 'hey', 'abba', 'hey'])
True
>>> repeated([3, 2, -5, 0, 1])
False
```

3.14 Write a function `is_contained(a, b)` that checks whether all elements of list `a` are contained in list `b`, returning `True` in that case and `False` otherwise. For example:

```
>>> is_contained([3,4,1],[1,2,4,1,3])
True
```

Start your definition by writing a few more test cases.

3.15 Write a function `remove_adjacent(xs)` to remove adjacent repeated elements from a list, that is, those for which `xs[i] == xs[i+1]`. The result should be a new list; the original list must not be modified. Examples:

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

Note that, unlike the examples shown in class, this function should not remove all duplicates, only consecutive ones. Start your definition by writing a few more test cases.

3.16 Write a function `occurrences(txt, c)` that returns a list with the indices of occurrences of the character `c` in the string `txt`. For example:

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

3.17 Write a function `words(txt)` that returns a list of words from the string `txt`. The words should only include uppercase or lowercase letters; assume that the string has no accented letters. Example:

```
>>> words("---Did Maria have a little lamb?")
['Did', 'Maria', 'have', 'a', 'little', 'lamb']
```