Remarks on assignment 1 (worksheet 3)

A complete correction for assignment 1 (that is worksheet 3 about functions) can be found on the website of the course. This document contains remarks about errors that were frequently encountered.

1 Docstring and examples for functions

When you implement a function, ask yourself the following questions in that order

1. Is my function documented?
2. Did I make simple examples?
3. Did I make complicated examples?

Testing your function is the best way to detect errors. In many of your works the function `fib_range` does return n+1 or n+2 numbers. That would have been easily detected by testing `fib_range(0)`, `fib_range(1)` and `fib_range(2)`.

Here is a good examples of what should be done

```
def sign(x):
    "Returns the sign of the number x"
    if x < 0:
        return -1
    elif x > 0:
        return 1
    else:
        return 0
```

Then for the tests, each case of the if/elif/else should be tested. For example

```
In : [sign(n) for n in range(-3, 3)]
Out: [-1, -1, -1, 0, 1, 1]
In : sign(13498295893485)
Out: 1
In : sign(-44434211)
Out: -1
```

2 Never use twice the same name for different things

No two functions should have the same name.

You might propose more than one implementation of a function. For that purpose, use different names. For example `collatz_length` and `collatz_length_bis`.

3 `print` versus `return`

Just take it as a rule: do not use print inside functions.

The reason is the following. Let us consider the examples of the Python functions `f` and `g` below.

```
def f():
    return 1
def g():
    print(1)
```

The function `f` has 1 as a return value, meaning that you can further use this result in other computations

```
In : a = f()
In : a + 1
Out: 2
```

Whereas `g` has no return value (this is the `None` object in Python)

```
In : b = g()   # you will see 1 on the screen because of print
1
In : print(b)
None
In: b + 1     # error since b is None
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

4 Identical instructions in if/elif/else and loops

If you use for loop, while loop and if/elif/else constructions ask yourself whether it is really needed.

The following weirdness are frequent in your assignments

```
if condition1:
    x = 1
    print(x)
elif condition2:
    x = 2
    print(x)
else:
    x = 3
    print(x)
```

or

```
for i in range(10):
    x = x + i
    y = 2*x
print(y)
```

In these examples, the instruction `print(x)` and `y = 2*x` should be moved. Namely

```
if condition1:
    x = 1
elif condition2:
    x = 2
else:
    x = 3
print(x)
```

and

```
for i in range(10):
    x = x + i
y = 2*x
print(y)
```

A concrete example that appear in some of your work is

```
def var(l):
    "Returns the variance of the list of numbers l"
    s = 0
    t = mean(l)
    for i in l:
        s = s + (i - t)**2
        g = s / len(l)
    return g
```

where the instruction `g = s / len(l)` should be moved outside of the for loop. Similarly in

```
def bad_sign(x):
    "Returns the sign of x"
    if x < 0:
        s = -1
        return s
    elif x > 0:
        s = 1
        return s
    else:
        s = 0
        return s
```

the instruction `return s` should be moved outside of the if/elif/else blocks.

5 When do we need to construct lists?

Do not construct lists unless necessary. List construction is expensive and can be often avoided.

Most of the time you can process data without having to construct an intermediate list. For example if you want to compute the sum of the squares of all integers between 1 and 100 just do

```
s = 0
for i in range(1, 101):
    s = s + i**2
```

There is no need to create first the list of the squares and then take their sum. This applies in particular to the `collatz_length` function. The following is a bad version

```
def bad_collatz_length(n):
    "Return the length of the Collatz sequence of n"
    seq = [n]
    while seq[-1] != 1:
        seq.append(collatz(seq[-1]))
    return len(seq)
```

It should be replaced with

```
def collatz_length(n):
    "Return the length of the Collatz sequence of n"
    s = 1
    i = n
    while i != 1:
        i = collatz(i)
        s = s + 1
    return s
```

6 Python is full of shortcut

There are many possible constructions in Python. Sometimes you might save some lines of code by using advantage of the language. You might also gain readability.

Using list comprehension is often good. The following

```
l = []
for i in range(10):
    if (i**2 + 3) % 2 == 0:
        l.append(i**3 + 5)
```

can be replaced by a one line list comprehension

```
l = [i**3+5 for i in range(10) if (i**2 + 3) % 2 == 0]
```