ADVANCED PYTHON HOMEWORK 3

Comparing programming styles

The deadline of the homework as well as its submission is handled centrally via SKOS. Please make sure you are submitting through that.

Every exercise is worth **4** points for this sheet, and you can **submit at most two.**

This homework sheet is equivalent to the Polish one; if you do not understand anything, feel free to ask your lab teacher or consult the Polish sheet.

The three tasks below ask you to implement functions that return a list of natural numbers satisfying the given conditions. Each exercise must be solved in three versions: an imperative version, a version with list comprehension and a functional version.

- In the *imperative version* we are using instructions such as: `while`, `for in` etc. and completing the resulting list with the method `append`.
- The version with *list comprehension* should be in a form of *one* list comprehension, or a chain of nested list comprehensions.
  In the case of nesting sublists may also be chained e.g. like this:

  ```
  def given_function(n):
      list_temp = [ list_comprehension ]
      return [ list_comprehension_containing list_temp ]
  ```

- In the *functional implementation* you should use functions dedicated to operations on list or (list generators) such as: `filter`, `range`, `sum` or `reduce`.

Use the module `timeit` to measure the performance of each version for different data. *Note: Be thorough and use various kinds of inputs, where applicable! Large-sized inputs, medium-sized inputs, fully random input or some input that is designed to take a long time are all good choices.*

Also, please make sure that your function returns the right output, for instance a list, when a list is required.

Exercise 1. Implement three unary functions `prime_imperative(n)`, `prime_comprehension(n)`, `prime_functional(n)` that return a list of primes smaller than n. For example:

```
>>> prime(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

Exercise 2. Implement three unary functions `perfect_imperative(n)`, `perfect_comprehension(n)`, `perfect_functional(n)` that return a list of perfect numbers smaller than n. For example:

```
>>> perfect(1000)
[6, 28, 496, 8128]
```

Exercise 3. Implement three unary functions `factors_imperative(n)`, `factors_comprehension(n)`, `factors_functional(n)` that find a prime factorization of n. The output is a list of pairs

$$[(p_1, w_1), (p_2, w_2), \ldots, (p_k, w_k)],$$

such that $n = p_1^{w_1} * p_2^{w_2} \ldots p_k^{w_k}$ where $p_1, p_2, \ldots, p_k$ denote different primes. For example:

```
>>> factorization(756)
[(2, 2), (3, 3), (7, 1)]
```

Since for this task you may need a list of primes, you can implement a helper function that tests primality or returns a list of primes. For this helper function, the implementation can be of your choosing.

Exercise 4. Implement three unary functions `amicable_imperative(n)`, `amicable_comprehension(n)`, `amicable_functional(n)` that return a list of pairs of amicable numbers smaller than n. For example:

```
>>> amicable(1300)
[(220, 284), (1184, 1210)]
```

All definitions can be found, for example, at Polish or English Wikipedia.