MAT101 – Programming with Python

Exam - 08.02.24, 9:00-12:00

Gauthier Wissocq

Institute of Mathematics, University of Zurich

- The exam is graded out of 24 points and your final grade will be between 1 and 6.
- For each exercise, produce a script named `'Exercise_n.py'`, with n number of the exercise. Your Python files should be uploaded on `https://exam.math.uzh.ch`.
- Appropriately comment your code to explain what it does, specifying inputs, outputs and algorithms. This will be taken into account when rounding off the final grade.
- No import is allowed except `numpy` and `matplotlib` in Exercise 3.
- Each question can be addressed independently. For example, if question a) requires writing a function called `test`, this function can still be used in other questions even if question a) has not been answered.

Exercise 1 (6p) (No import allowed for this exercise)
In this exercise, a *text* refers to a list of characters. For example, `['h','o','t','e','l']` is a text of length 5.

a) (2p) Without using the test `==` directly on the lists (like 'list1 == list2')[1], write a function called `'equal_texts'` which takes as input two texts and returns the boolean `True` if the two texts are equal, `False` otherwise.

b) (2p) In this question, we assume that all the texts have the same number of characters n. If two texts are not equal, we would like to determine the number of positions at which they differ. In other words, we want to find the number of positions i, $0 \leq i < n$, such that the character at position i differs in both texts. Write a function called `'distance'` which takes as input two texts and returns this result. For example:

```
>>> distance(['v', 'i', 's', 'a'], ['v', 'a', 'i', 's'])
3
>>> distance(['a', 'v', 'i', 's'], ['v', 'i', 's', 'a'])
4
```

c) (2p) Write a function called `'no_common_character'` which takes as input two texts and returns `True` if the set of characters appearing in one text is disjoint from the set of characters appearing in the other one, `False` otherwise. For example:

```
>>> no_common_character(['a', 'v', 'i', 's'], ['v', 'i', 's', 'a'])
False
>>> no_common_character(['a', 'v', 'i', 's'], ['u', 'r', 'n', 'e'])
True
```

[1] same for `!=`: do not write 'not (list1 != list2)'

Exercise 2 (10p) (No import allowed for this exercise)

a) (2p) Write a function `'divisors'` which takes as input a striclty positive integer and returns the list of all its divisors. We recall that the divisors of an integer n is the set of integers $d \leq n$ such that the rest of the Euclidean division of n by d is zero.

b) (2p) Write a function `'GCD'` which takes as input two stricly positive integers and returns their greatest common divisor, *i.e.* the largest positive integer that is a divisor of both integers given as input.

c) Create a new class `Fraction` that contains the following functions:

- (1p) A constructor which takes as input a numerator and a denominator (both assumed to be stricly positive) and defines two attributes in the class: `num` (the numerator) and `den` (the denominator).
- (1p) A method `print`, designed to print the fraction under the form `num/den`, as in the following example:

```
>>> Fraction(5,6).print()
5/6
```

- (1p) A method `inverse`, which returns a `Fraction` that is the inverse of the current one.
- (1p) A method `reduce`, which reduces the current `Fraction` to lowest terms. For example, considering a fraction `F`, the command `'F.reduce'` should reduce `F`.
  Hint: use the function `'GCD'` created above.
- (2p) A method designed to overload the operator `+` such that, if `A` and `B` are two `Fraction`, `A+B` returns the `Fraction` that is the sum of `A` and `B`. The result of the operation should be reduced to lowest terms.

Exercise 3 (8p) (Import `numpy` and `matplotlib`, no other import allowed)
Consider the following sequence defined for any integer $k \geq 1$:

$$a_k = \frac{(2k)^2}{(2k)^2 - 1},$$

and the so-called *partial Wallis product* W_n defined as the product of all the a_k, for $1 \leq k \leq n$:

$$W_n = \prod_{k=1}^{n} a_k.$$

The aim of this exercise is to observe that $W_n \to \pi/2$ when $n \to \infty$.

a) (1p) Create a function called `a_sequence` which takes as input an integer n and returns a numpy array containing all the elements a_k for k ranging from 1 to n included.
Hint: no need to use a loop thanks to numpy functions.

b) (1p) Create a function called `partial_wallis` which takes as input an integer n and returns the partial Wallis product W_n.

c) (4p) Plot the partial products W_n obtained corresponding to $n = 2^k$, for $k \in \{1, 2, \dots 15\}$, and compare it with $\pi/2$. Ensure that your plot look like the Figure 1 below (be careful with the axes, labels, colors, and ranges in x and y).

d) (2p) Write a function called `'convergence_up_to_tolerance'` which takes as input a tolerance ε and an integer m and returns as output the first $n \geq 1$ such that $|W_n - W_{n-1}| \leq \varepsilon$ if $n \leq m$ and stops the execution, with a message for the user, if such condition does not occur for $n \leq m$.

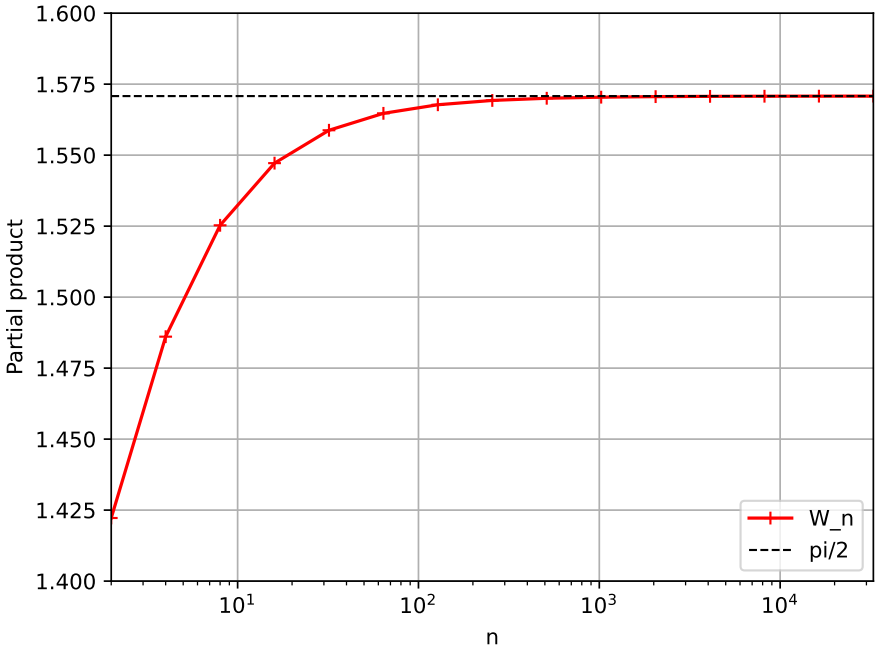

Figure 1: Expected figure in Exercise 3, question c)