IT1007 Introduction to Programming with Python and C — Practical Exam

Submission instructions:

1. Total marks: 100. Duration: 2 hours
2. Complete your code using the skeleton files provided, then test your code on IDLE first before submitting to Coursemology.
3. You must name your functions exactly as the questions stated.
4. Click "Run Code" to test that if your functions work on Coursemology. You have only 5 test runs on each question.
5. To submit your code on Coursemology, click on "PE" in the sidebar followed by the appropriate "Attempt" button.
6. **Copy ONLY the required functions** from your completed skeleton file into the Coursemology code window. We will deduct marks if you pasted extra functions into a particular question.
    - Moreover, you should only submit your functions WITHOUT the test cases. E.g. the last test case in Question 5 will cause a timeout in Coursemology and tell you that your answer is wrong.
7. Because we have numerical answers, Coursemology may tell you that you are "wrong" because your output numbers are not exactly the same with the sample answer. You should just submit with confidence.
8. Click "Finalize Submission" to submit your code. **You will not be able to amend your code after you have finalized your submission.**
9. **We will NOT grant you extra test runs or "finalize submission" if you accidentally used them up.**

You should not import any other functions other than those that are already imported in the given code.

- But of course, you can define or copy-and-paste your own additional functions written by yourself.
- You should manipulate the images with `numpy` and `scipy` packages ONLY, but **not** other library such as PIL. And the package `matplotlib` should be used for showing the image only, namely only functions such as `imread()`, `subplot()`, `imsave()`, `axis()`, `imshow()` and `show()`.

Important Notice:

We will also grade your programs according to your style and performance. Namely, in order to gain full marks you should not have bad styles and slow computations, such as

- Unnecessary lengthy code
- Forbidden functions, such as using `scipy.integrate()` for Question 3, is NOT allowed.
- But you can assume that the inputs of your function will be in the right type. For example, we will **not** call your function as `"compute_e_within_error('abc')"` in Question 1.

Failure to follow each of the instruction will result in 10% deduction of your marks.

Question 1 [10 marks]: Computing the Natural Number *e*

The mathematical constant e is the unique number whose natural logarithm is equal to one. And you can compute the number with the following sum the infinite series,

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \times 2} + \frac{1}{1 \times 2 \times 3} + \frac{1}{1 \times 2 \times 3 \times 4} + \cdots$$

Of course, we cannot compute the perfect e until n equals to infinity (and no one can). Let's say we compute the first $i+1$ terms of e such that

$$e_i = \sum_{n=0}^{i} \frac{1}{n!}$$

So we can actually stop computing e up to the $(i+1)^{th}$ term if $e_{i+1} - e_i$ is smaller than a certain error constant. We assumed that the error we can tolerate must be less than 1.0 .

Write a function '`compute_e_within_error(err)`' such that

- It will compute and return e_i when $e_{i+1} - e_i$ is smaller than the number 'err'.
- It will print a message of how many terms it has computed (namely, the number i)
- And it should NOT waste time on computing extra j^{th} term for $j > i + 2$.

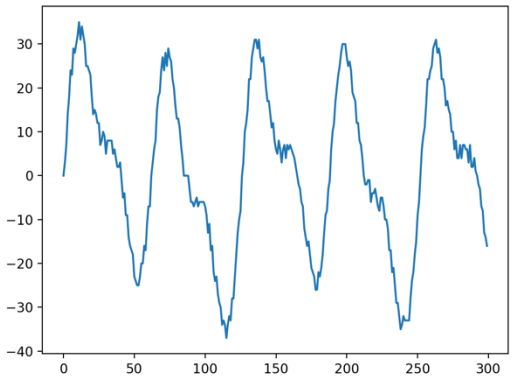
Sample outputs:

```
>>> e = compute_e_within_error(0.99)
No. of terms = 2
>>> print(e)
2.0
>>> e = compute_e_within_error(0.01)
No. of terms = 5
>>> print(e)
2.708333333333333
>>> e = compute_e_within_error(0.0000000000000000001)
No. of terms = 21
>>> print(e)
2.7182818284590455
```

Note that the "No. of terms" is equal to $i + 1$.

Question 2 [25 marks]: Filtering a wave

You are given a wave like your lab exercise before. This time, your job is to smooth the wave out by a very simple filter. You are given a wave in a list named '`original_wave`' as follows:



Part 2a [20 marks]

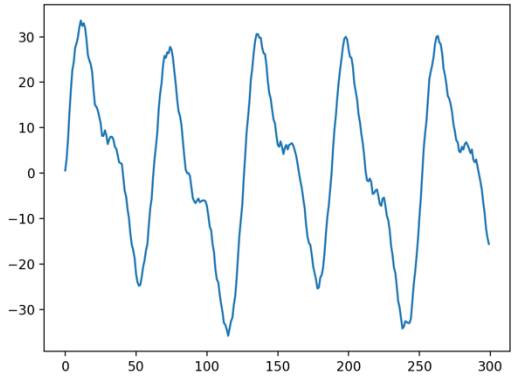
Write a function '`filter_wave(wave)`' and this will produce a new wave which is a smoothed version of the input 'wave'. Following the rules below

- In the new wave, for every position i, it is the weighted sum of three positions of the original wave. More preciously, the new wave value in position i will be equal to

```
new_wave[i] = wave[i-1] * 0.2 + wave[i]*0.6 + wave[i+1]*0.2
```

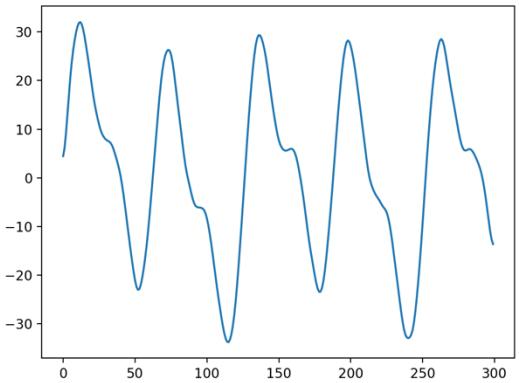
- Let `len(wave)` be `L`. The above method will access `wave[-1]` and `wave[L]` in which do NOT exist in the original wave. So, you can deem the value of `wave[-1]` as `wave[0]` and the value of `wave[L]` as `wave[L-1]`.
- As in the lab before, you should **NOT** modify the original wave input

Here is the expected shape of the wave after `filter_wave(original_wave)`



Part 2b [5 marks]

Write a function '`filter_wave_n(wave,n)`' for **n** ≥ 0. And this will repeat the function `filter_wave()` in Part 2a **n** times to the wave accumulatively and produce an even smoother wave. Here is the expected wave for `filter_wave_n(wave,10)`



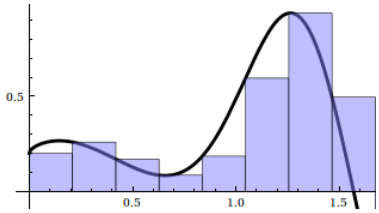
Again, you should not modify the original wave. **And note that you will gain marks for Part 2b ONLY IF your Part 2a also works correctly**.

Question 3 [25 marks]: Agar Agar Integration

Part a and Part b are related. You are recommended to read both parts before starting Part a. In this question, you have to compute the integration by your own code. Namely, you cannot use any library functions like `scipy.integrate()`.

Part 3a [20 marks]

Computing the integral of a function f can be very difficult if it is hard to compute the integration of f. So, avoiding the difficult parts, we want to do our 'Agar Agar Integration', namely, ***numerical integration***.
To compute the integral of f from $x = a$ to $x = b$, it is equal to compute the area under the curve $y = f(x)$ between $x = a$ and b. The numerical integration method (with the rectangular rule) says that, we can compute some rectangles and sum up their areas to approximate the total area below the curve



Of course, if the slices are slimer, we will have better approximation of the integral

You are given a function $q(x)$. Write a function '`numeric_integral(l,r,delta_x)`' that will evaluate $\int_l^r q(x)dx$ numerically from l to r with the width of the rectangles equals to '`delta_x`'.
The function '`int_q(x)`' is the analytical integral of $q(x)$. It is mainly for you to verify your answers. Of course, you cannot use that as your answer.

Sample Outputs:

```
>>> numeric_integral(-1,0,0.0001)
1.0002500166667745
>>> print(numeric_integral(-1,0,0.0000001))
1.0000002505086756
>>> print('True integral of q from -1 to 0 = ' +str(int_q(0)-int_q(-1)))
True integral of q from -1 to 0 = 1
>>> print(numeric_integral(0,2,0.0001))
32.00400013332054
>>> print(numeric_integral(0,2,0.0000001))
31.999996009131905
>>> print('True integral of q from 0 to 2 = ' +str(int_q(2)-int_q(0)))
True integral of q from 0 to 2 = 32
```

Part 3b [5 marks]

You are given more functions on top of the function $q(x)$. Write a function '`numeric_integral_2(l,r,delta_x,f)`' that will evaluate the integral of f as f is also an input.

Sample Outputs:

```
>>> print(numeric_integral_2(0,2,0.0000001,sin))
1.41614679100176
>>> print(cos(0)-cos(2))
1.4161468365471424
>>> print(numeric_integral_2(0,2,0.0000001,f))
9.999999400577833
>>> print(int_f(2)-int_f(0))
10.0
>>> print(numeric_integral_2(0,2,0.0000001,q))
31.999996009131905
>>> print(int_q(2)-int_q(0))
32
```

And note that you will gain marks for Part 3b ONLY IF your Part 3a also works correctly.

Question 4 [20 marks]: Image Composition

Welcome to the Marval Studio! (Oops, you noticed that? We are 'Marval', not 'Marvel'…) You are the one who is responsible to make the movie ***AveNUgerS*** !
Here is a scene captured in the green screen studio on the left, and the COM1 building on the right with some modifications.

"avergers green.jpg"

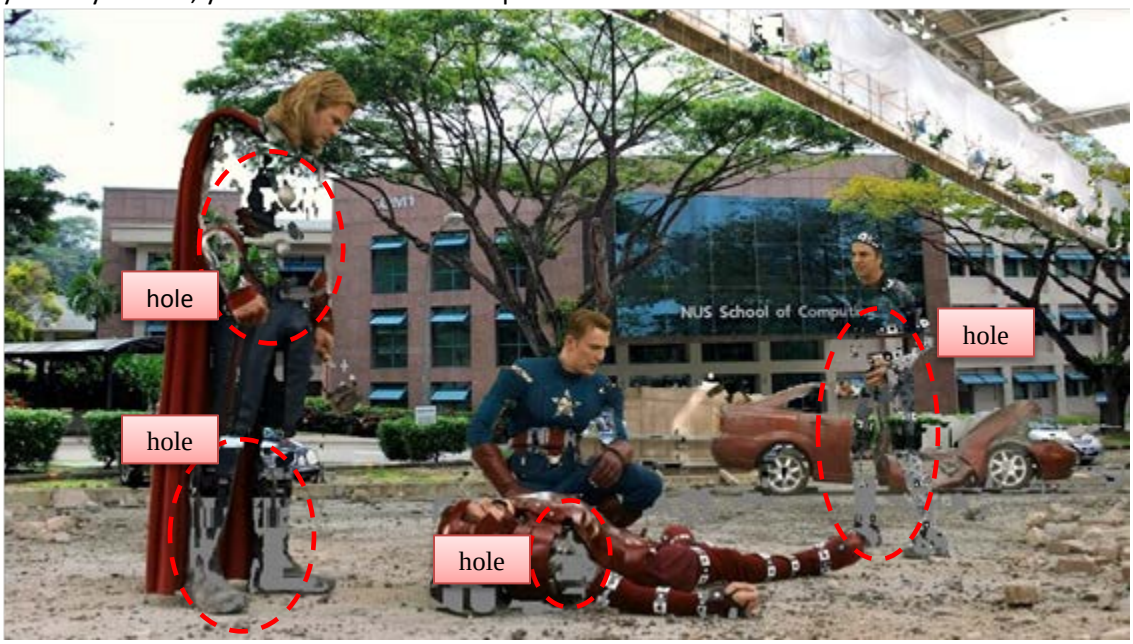
"background.jpg"

Your job is, to remove the green pixels in the left picture and replace them by the background at the right.

The two pictures are aligned at the top left corner. And the background picture is "larger" than the foreground picture. And you do not need to care about the extra part in the background that is larger than the foreground picture. Therefore, your resultant picture should have the same size as the original one, NOT the background.

First, you should try to figure out all the pixels that are "green". The criteria for a "green" pixel is that the green component G in its color [R, G, B], is larger than the other two, G > R and G > B.

However, if you only do this, you will have a resultant picture like this:

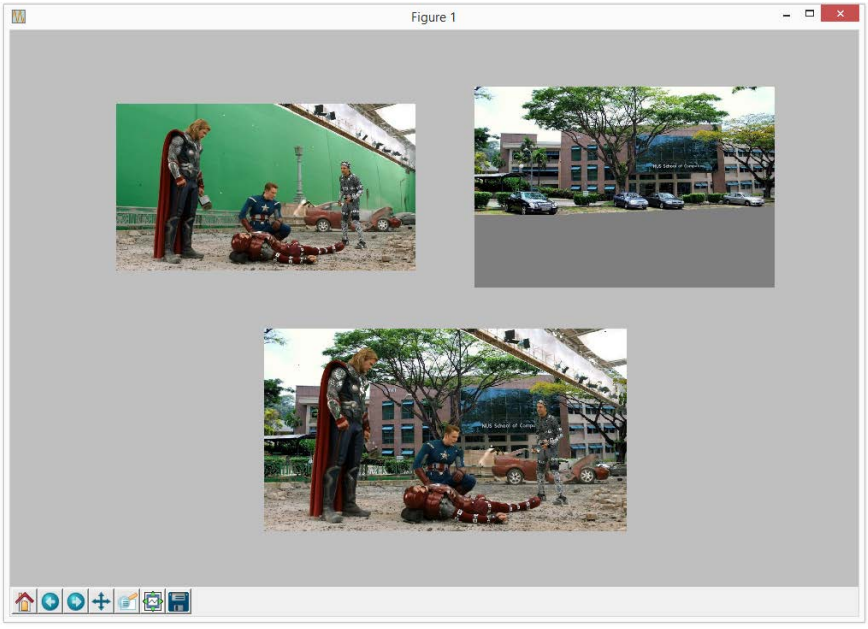


There are a lot of holes because those regions have very dark colors and are considered as “green”. So in order to mend the holes, we need to include one more criteria that the pixel is replaced when it is “green” AND the green value G is > 110. Combining these two criteria, we will have a resultant image like this:

Although there are still some holes on Bruce and other heroes… but let’s forget it and leave it to the next stage of compositions with the additions of armors and Hulk.

So your **tasks** are

a. Write a function “`image_composition(filename1, filename2)`” that takes in the foreground (filename1) and remove the green and replaced by the background image (filename2)
b. Save your resultant image as `'aveNUgerS.jpg'` in the function “`image_composition`”.
c. Also, in the function “`image_composition`”, display the original foreground and background images, as well as the final resultant images, in a single window (without axis) ***exactly*** like the layout shown below:



Question 5 [20 marks]: Squared Sum Number

A positive integer *n* is a *squared sum number* (SSN) if *n* is the sum of the squares of other two positive non-zero integers (can be the same integer). For example, 13 is an SSN because 13 = 4 + 9, namely, 13 = 2^2 + 3^2.

Write a function "`is_squared_sum_num(n)`" to check if a positive integer *n* is an SSN.

- If so, print out the two numbers such that one of them is the smallest possible number. For example, if 200 = 2^2 + 14^2 and 200 = 10^2 + 10^2, you should report 2 and 14 only because 2 is the smallest possible number among 2, 10 and 14. And remember to return the value `True`.
- Otherwise, report that it is not an SSN and return the value `False`.

Remember NOT to include the test cases function call into your submission in Coursemology

Sample outputs:

```
>>> is_squared_sum_num(18)
18 is a squared sum number!
Which is the sum of the squares of 3 and 3
True
>>> is_squared_sum_num(19)
19 is NOT a squared sum number!
False
>>> is_squared_sum_num(200)
200 is a squared sum number!
Which is the sum of the squares of 2 and 14
True
>>> is_squared_sum_num(20000)
20000 is a squared sum number!
Which is the sum of the squares of 20 and 140
True
>>> is_squared_sum_num(1356446145698)
1356446145698 is a squared sum number!
Which is the sum of the squares of 823543 and 823543
True
>>> is_squared_sum_num(1356446145699)
1356446145699 is NOT a squared sum number!
False
>>> is_squared_sum_num(390982172680606)
390982172680606 is NOT a squared sum number!
False
>>> is_squared_sum_num(39098217268060608)
39098217268060608 is a squared sum number!
Which is the sum of the squares of 823543 and 1977326743
True
```

Just for fun [0 mark]

(For Question 4) Well, if you are a perfectionist, you will notice that there are some leftover studio lightings on the top right corner of the resultant pictures. You can use an edited foreground to improve your image by:

```
image_composition('avengers green triangle.jpg','background.jpg')
```

And the resultant image will be better.

- *End of PE* -