Assignment 4

COMP 208 Fall 2019

posted:	Wednesday, November 20, 2019 at 23:59
due:	Tuesday, December 3, 2019 at 23:59
no penalty until:	Wednesday, December 4, 2019 at 23:59

Primary Learning Objectives

By the end of this assignment, students should be able to:

- work comfortably with NumPy arrays and perform operations on them.
- understand methods of numerical integration and their Python implementations.
- integrate code into an existing object-oriented structure.
- solve a mathematical root-finding problem both by hand and using Python.

Submission Instructions

- Please store all your files in a folder called **Assignment4**, zip the folder and then **submit the file Assignment4.zip** to myCourses. (See the instructions on myCourses for more details on how to zip files.) Inside your zipped folder, there must be the following files (**please use these exact file names**).
  - `image.py`
  - `polynomial.py`
  - `roots.py`
  - `iterations.pdf`

  Any deviation from these requirements will result in deduction of points.
- Don't worry if you realize you made a mistake after submitting your zip file: you can submit multiple times on myCourses. You are encouraged to submit a first version a few days before the deadline (computer crashes do happen and myCourses may be overloaded during rush hours).

Penalties

- Late assignments will be accepted up to 2 days (48 hours) after the due date and will be penalized by 10 points per day. Note that submitting one minute late is the same as submitting 23 hours late. 10 points will be deducted for any student who has to resubmit after the due date (i.e. late) irrespective of the reason, be it wrong file submitted, wrong file format submitted or any other reason. This policy will hold regardless of whether or not the student can provide proof that the assignment was indeed "done" on time.
- **Note that the output obtained by running your code should be exactly as specified in the instructions for each questions. Failure to do so will result in deduction of points.**
- If your program does not work at all, e.g. it gives an error and does not produce any output, you will automatically get zero points for that question. If your program executes without errors but produces incorrect output, partial grades may be awarded based on the correctness of the code.
- You are expected to put **comments in each file**, on average 1 comment for every 5 or 6 lines, explaining what the lines are doing. Failure to comment your code in this manner may result in deduction of points.
- You are expected to use **descriptive names for your variables** whenever possible (depending on the question). Do not use variable names like x, y or z. Instead, use names like user_input, sum_of_numbers, or average_value. Failure to give your variables descriptive names may result in deduction of points.
- If anything is unclear, it is up to you to clarify it by either directly asking a TA during office hours, or making a post on the myCourses discussion board.

Question 1 [40 points]

In this question, you will write some functions to manipulate an image in different ways. Note that you can and should use NumPy arrays, functions and methods for this question. Further, please use the provided `image_tester.py` file to test your functions.

1. [15 points] Write a function `image_to_greyscale` that takes in two arguments: a string `image_filename`, and a tuple `weighting` containing three floats. The function should return a NumPy array representing a greyscale version of the image, that is, an image with only shades of grey. You will have to load the image and perform the greyscale conversion. Note that you are not allowed to use any third-party function for the greyscaling (e.g., you cannot use the `rgb2grey` function as seen in class). Instead, you should transform each pixel element in the image array from an array of three numbers (red, green and blue color intensities) into a single number representing the shade of grey. Use the following formula to convert each pixel:

   $pixel = r * pixel_r + g * pixel_g + b * pixel_b$

   where r, g, b are the three elements of the `weighting` tuple passed in as argument, and $pixel_r$, $pixel_g$, $pixel_b$ are the RGB color intensities of the pixel.

2. [25 points] Write a function `combine_images()` that reads two images from files, selects a rectangular region from each image, and places those two regions side-by-side (horizontally) to produce a new image array. The function takes six arguments: `file1`, `file2`, `topleft1`, `topleft2`, `height` and `width`.

   The arguments `file1` and `file2` are strings containing the names of the image files to be read into memory as NumPy arrays. The argument `topleft1` will be a list of two integers [top, left] indicating the top-left corner of the rectangular region for the first image (the one in `file1`). Similarly, the list `topleft2` will indicate the top-left corner of the rectangular region for the second image (the one in `file2`). For the sake of simplicity, the rectangular regions for both images will have the same size, as specified by the arguments `height` and `width`.

   Thus, given the two rectangular regions, one for each image, the function should then combine both regions side-by-side (horizontally) by selecting all pixels in both regions and putting them into a new NumPy array. The **return value** of the function should be this new NumPy array with the horizontally combined regions.

   You can assume that the lists `topleft1` and `topleft2` will contain valid array indices, i.e., the locations of the top-left corners will be within the image boundaries. However, the `height` and `width` arguments could be such that the region goes *beyond* the image bounds. In such a case, as shown in `Example 4`, pixels outside the image should be considered as black (RGB value of [0, 0, 0]).

File to submit

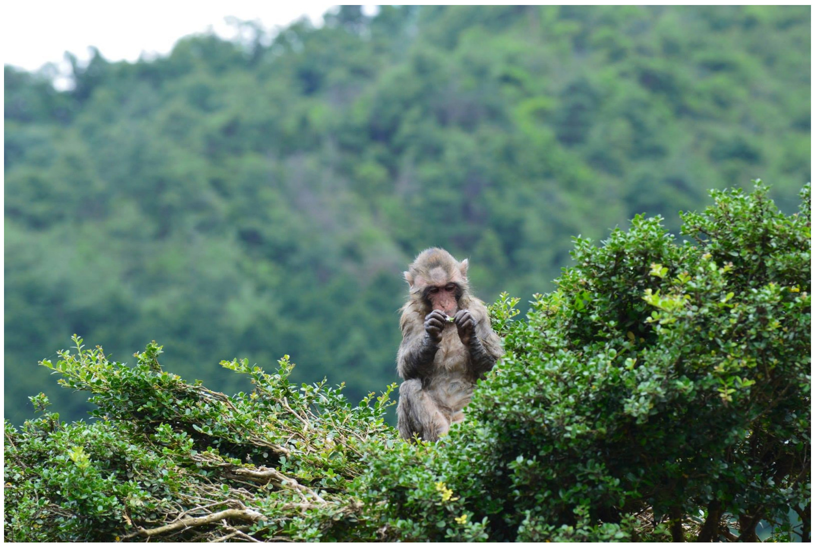
Both functions `image_to_greyscale()` and `combine_images()` must be written and submitted in a file called **image.py**. You do not need to submit the `image_tester.py` file.

Images in examples

All the images used in the examples below are also provided with this Assignment.

`flower.jpg`

`monkey.jpg`

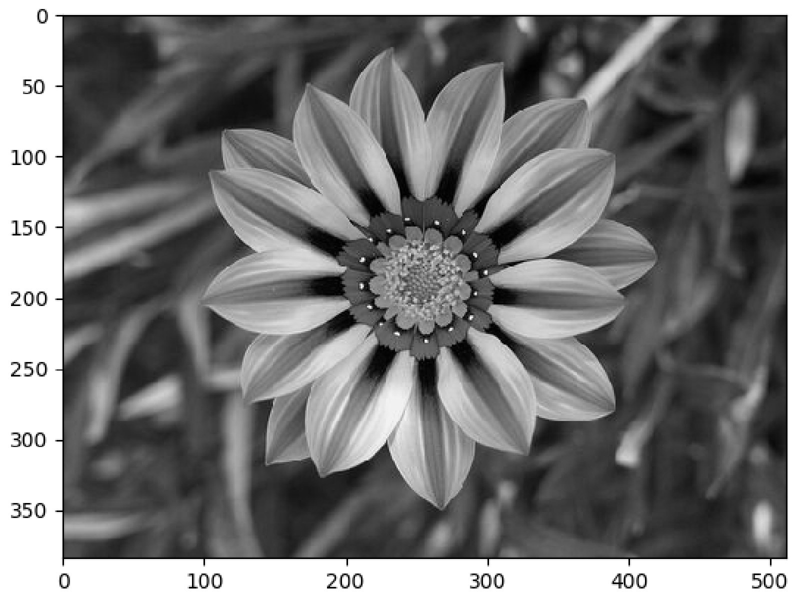


```
tiger.jpg
```

Examples (also given in `image_tester.py`)

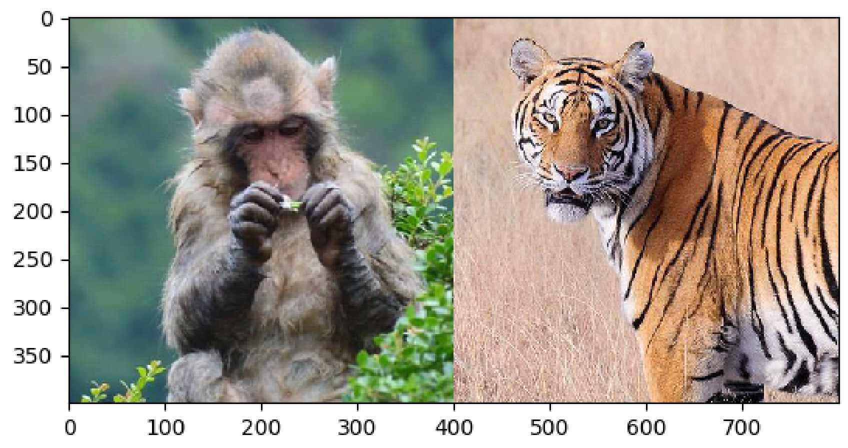
EXAMPLE 1:

```
greyscale = image_to_greyscale("flower.jpg", (0.2125, 0.7154, 0.0721))
plt.imshow(greyscale)
plt.show()
```



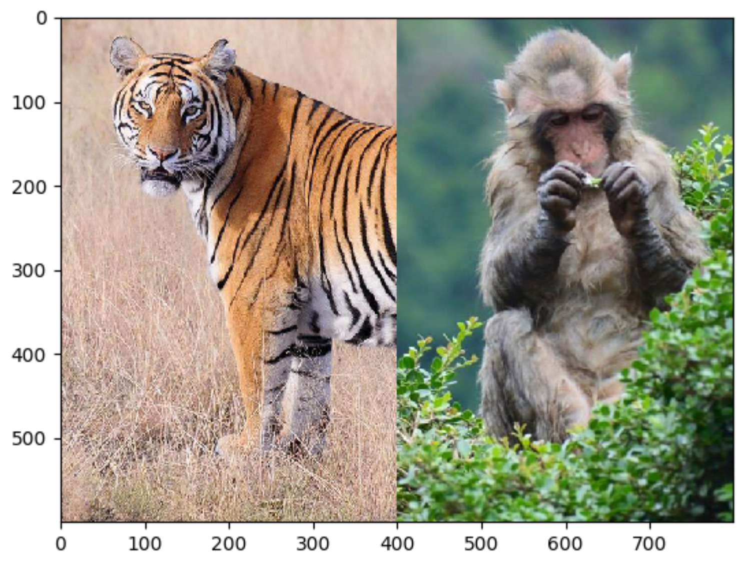
Example 2:

```
image = combine_images("monkey.jpg", "tiger.jpg", [600, 900], [50, 550], 400, 400)
plt.imshow(image)
plt.show()
```



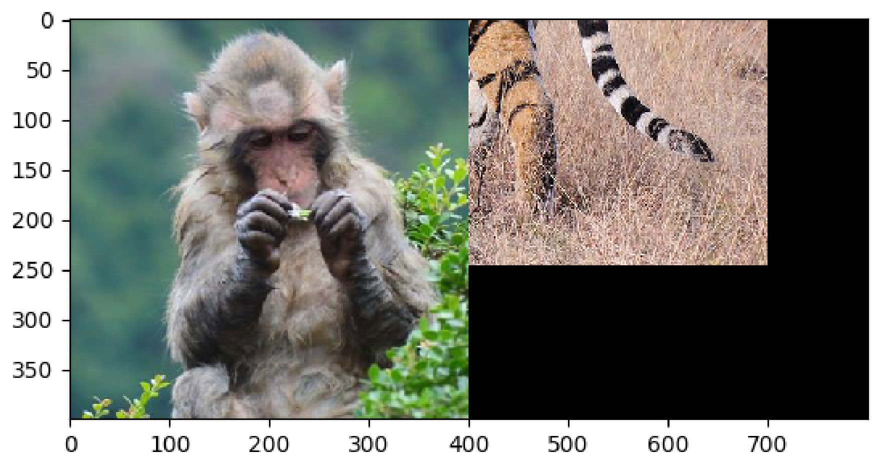
Example 3:

```
image = combine_images("tiger.jpg", "monkey.jpg", [50, 550], [600, 900], 600, 400)
plt.imshow(image)
plt.show()
```



Example 4:

```
image = combine_images("monkey.jpg", "tiger.jpg", [600, 900], [450, 1100],
                       400, 400)
plt.imshow(image)
plt.show()
```



Question 2 [35 points]

In Assignment 2, you were asked to create functions that dealt with polynomials, and later it was shown how these functions, written in the procedural style of programming, could be expressed in terms of object-oriented programming with a Polynomial class. In this question, you will add methods to the Polynomial class to increase its functionality, such as adding two polynomials, plotting a polynomial, etc.

You must write the methods for this question inside this Polynomial class, in the **polynomial.py** file provided to you. Please take a minute to re-familiarize yourself with the Polynomial class. Recall that each object of the Polynomial class has a dictionary as an attribute, which contains the powers and coefficients of the polynomial.

After you have written code for a method, you may want to test it. Please use the **polynomial_tester.py** file provided to you to test your code. It contains the useful function `create()` to create a Polynomial object from a string.

The Polynomial class should contain the following new methods:

- **evaluate()** — this method evaluates the polynomial at a given value `x`. It takes one argument `x` (a floating-point number), and returns the value of the polynomial at that `x`.
- **add()** — this method adds two polynomials and produces the resulting polynomial; i.e., it adds terms of equal powers together. It takes one argument `other_poly`, which will be a Polynomial object. It then adds the terms from the two Polynomial objects, `self` and `other_poly`, and returns the result as a new Polynomial object. (Note: Do not modify either of the two summands, that is, `self` or `other_poly`.)
- **plot()** — this method plots the polynomial using Matplotlib. It takes four arguments: `a`, `b`, `N`, `filename`. The interval `[a, b]` define the range of x-values that should be used for plotting. The range should be divided into `N` equally spaced numbers (including `a` and `b`). For each of the x-values, y-values can be obtained by evaluating the polynomial at that x-value (using the evaluate() method above). The plot should have a title as shown in `Example 4` below; it is formed using the string representation of the polynomial object. Finally, the plot should be saved to a file using the name given in the argument string `filename`.
- **integrate()** — this method takes two arguments, numbers `a` and `b` and computes the definite integral of the polynomial between those limits `[a, b]`. First, it computes the indefinite integral in a closed-form, using the same approach as in Assignment 2, by manipulating powers and coefficients in the dictionary. Then, it evaluates that closed-form integral at the limits `a` and `b`, and returns the definite integral as a floating-point number.
- **trapz_integrate()** — this method computes the definite integral approximation of the polynomial using the trapezoidal method. It takes three arguments: `a`, `b`, and `N`. The numbers

`a` and `b` form the limits of the integral, while `N` is the number of sub-intervals or segments used for the trapezoidal method. Note that the method arguments do not include the function `f` (unlike the trapezoidal method shown in the lecture slides). In this method, you must use the Polynomial class in a way that fulfills the role of the function `f`.

File to submit

You should submit the file **polynomial.py**.

Examples (as given in polynomial_tester.py)

Example 1:

```
poly1 = create("1x3 1x0")
poly2 = create("-1x2")
poly = poly1.add(poly2)
print(poly)
print(poly.evaluate(1.5))
```

```
1.00x3 1.00x0 -1.00x2
2.125
```

Example 2:

```
poly = create("-4x3 3x2 25x1 6x0")
print('Closed-form integral:', poly.integrate(-2.0, 2.0))
```

```
Closed-form integral: 40.0
```

Example 3:

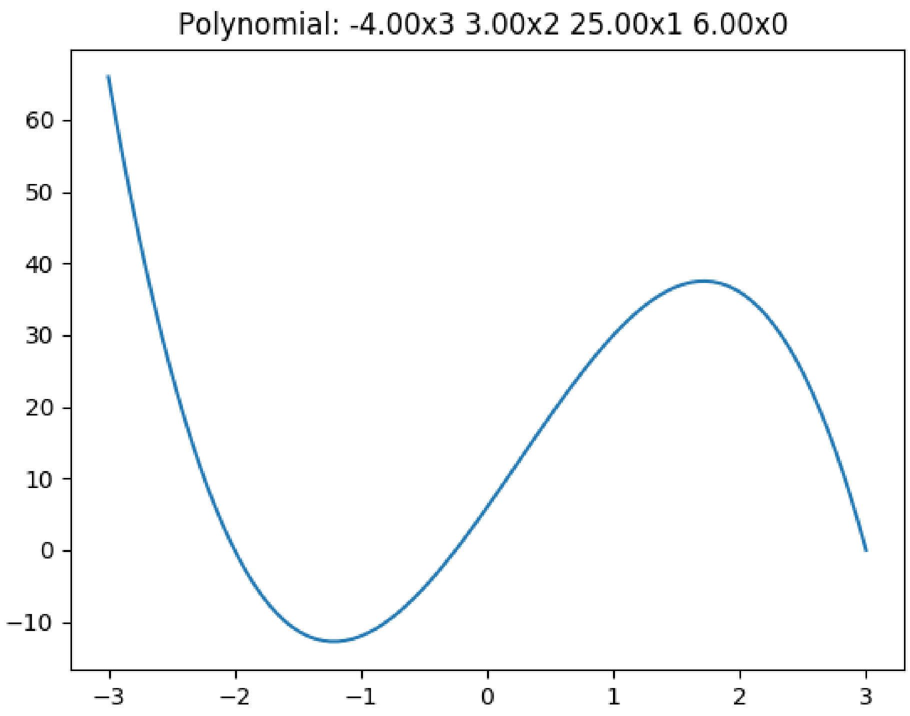
```
poly = create("-4x3 3x2 25x1 6x0")
print('Approximate integral:', poly.trapz_integrate(-2.0, 2.0, 50))
```

```
Approximate integral: 40.012800000000034
```

Example 4:

```
poly = create("-4x3 3x2 25x1 6x0")
poly.plot(-3.0, 3.0, 100, 'myplot.png')
print('open the file myplot.png to see the plot')
```

```
open the file myplot.png to see the plot
```



Question 3 [25 points]

The speed v of a Saturn V rocket in vertical flight near the surface of the earth can be approximated by the following formula:

$$v = u \cdot log\left(\frac{M_0}{M_0 - \dot{m}t}\right) - gt$$

where

- $u = 2510$ m / s (velocity of exhaust relative to rocket)
- $M_0 = 2.8$ x 10^6 kg (mass of rocket at liftoff)
- $\dot{m} = 13.3$ x 10^3 kg/s (rate of fuel consumption)
- $g = 9.81$ m/s^2 (gravitational acceleration)
- $t =$ time measured from liftoff

Using the secant root-finding method and its associated Python code seen in class, determine the time when the rocket reaches the speed of sound (335 m/s).

Write your code in `roots.py`. You can copy the root-finding method code from the slides as part of your program. You can also use NumPy arrays and functions.

After you write the code for the above calculation, in the same file `roots.py` plot a graph with the speed of the rocket on the y-axis and time on x-axis, for the time interval (0, 0.25). Provide appropriate titles for the two axes and the plot itself. Save this plot to disk as rocket.png using the `plt.savefig("rocket.png")` function. Do not submit the rocket.png file – when your code is run during grading, the file, if saved properly, will be automatically created at that time. (Be sure to verify that your code does produce the correct image saved on disk when run.)

Finally, answer the same rocket question above (finding the time when it reaches the speed of sound), but this time do the calculation manually, on paper or on computer, using the steps shown in class. Use the Newton-Raphson method with epsilon as 10^{-9}, the first approximation x_0 as 0, and 4 iterations. For each iteration, you should write the value of x, $f(x)$, $f'(x)$, and the difference between each root approximation. Include the steps in your submission under the name iterations.pdf. Your answer should be precise within 5 decimal places. Note that hard copies of your answer will not be accepted. If you decide to write down the steps by hand, you must either scan your pages into the computer, take a very legible picture with your phone or other device, or in some other way digitize the answer.

Files to submit

You must write your program in a file called **roots.py**. Separately, you must include the file **iterations.pdf** containing your manual steps to find the root as outlined above.