

Python for Math and Stat Spring 2025

Final Exam

Assume that all necessary packages have been imported.

1. (12 pts) For the following 4 problems, write down what each code block would display if executed in a Jupyter cell. If the code generates an error or infinite loop, write `Error`.

- (a)

```
num_arr = np.array([1., 2., 3.])
4*num_arr - num_arr**2 - 3
```
- (b)

```
alpha = 'abcdefghijklmnopqrstuvwxyz'
a = {alpha[x]: x+1 for x in range(len(alpha))}
a['x']
```
- (c)

```
list(map(lambda x: x**2 + 1, [-3, 0, 4]))
```
- (d)

```
def func(num):
    print(num, end=' ')
    if num <= 0:
        return 3
    else:
        return 2*num + func(num-3)

func(10)
```

2. (10 pts) The following two questions are related.

- (a) Write a function called **count_threes(num)** which returns the number of digits in positive integer num equal to 3. For example `count_threes(1138)` would return 1 while `count_threes(234353)` would return 3.
- (b) Write code to use your function to find the average (mean) number of 3s in all numbers from 1 to 10^7 .

3. (12 pts) A *Pythagorean triple* (x, y, z) of three positive integers has the property

$$x^2 + y^2 = z^2,$$

corresponding to the sides of a right triangle with hypotenuse z and legs x and y .

Euclid's method generates a Pythagorean triple by taking any two distinct positive integers a and b and producing these values for x and y :

$$x = |a^2 - b^2|$$
$$y = 2ab$$

- (a) Write a function called **Pyth_trip(a, b)** that takes two positive integers a and b and uses Euclid's method to generate a Pythagorean triple. The function returns the answer in the form of an (x, y, z) tuple of ints. You may assume a and b are distinct. For example, `Pyth_trip(1, 2)` returns `(3, 4, 5)`.

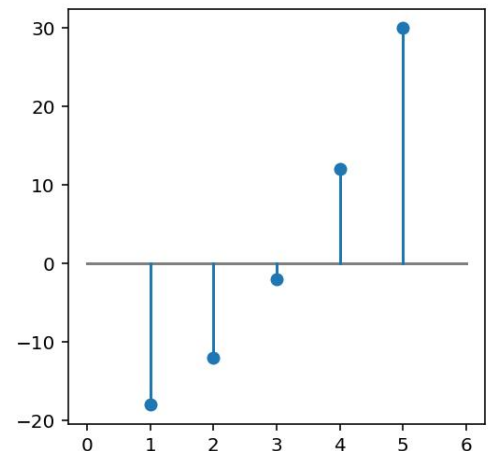
- (b) A Pythagorean triple (x, y, z) is *primitive* if x , y , and z have no common divisor. Write a function **Pyth_reduce(triple)** that takes an (x, y, z) triple in the form of a tuple and returns the corresponding primitive triple. You may call the `gcd(m, n)` function from class to find the greatest common divisor of x and y . For example, `Pyth_reduce((10, 24, 26))` returns `(5, 12, 13)`.

4. (14 pts)

- (a) Write a function **quad_pts(coeffs, npts)** that takes the coefficients (a, b, c) of a quadratic function $f(x) = ax^2 + bx + c$ and returns the function values $f(1), f(2), \dots, f(npts)$ in a numpy array. The code must use **numpy features** such as `arange` and vectorization. It should not use a loop or list comprehension.

Example: `quad_pts((2, 0, -20), 5)` returns `array([-18, -12, -2, 12, 30])` corresponding to $f(x) = 2x^2 - 20$, $f(1) = 18, \dots, f(5) = 30$.

- (b) Write a function **quad_plot(coeffs, npts)** that takes the output of `quad_pts`, plots the points, and draws a vertical line from each point to the x -axis. The code should plot a horizontal line to indicate where the x -axis is. Use default colors.



5. (15 pts) Create a class called **Date**. Each instance of the class corresponds to a date with month, day, and year attributes

- **month**: an integer from 1 to 12
- **day**: an integer from 1 to 31
- **year**: a 4-digit integer

Example: `vars(Date(5, 7, 2025))` returns `{'month': 5, 'day': 7, 'year': 2025}`.

The class includes these methods:

- **month_name()**: returns the name of the month in string format. The code can reference a global variable `months = ['Jan', 'Feb', ..., 'Dec']` which contains the 3-letter abbreviations for month names in an ordered list.

Example: `Date(5, 7, 2025).month_name()` returns `'May'`.

- **format()**: returns the date in `m/d/yy` string format.

Example: `Date(12, 4, 2025).format()` returns `'12/4/25'`.

- **century()**: returns the century the given date is in. (For example, the 21st century includes the years 2001 to 2100 inclusive.)

Examples:

`Date(5, 7, 1776).century()` returns 18, representing the 18th century.

`Date(5, 7, 2000).century()` returns 20, representing the 20th century.

6. (12 pts) The DataFrame **dfplanet** contains information about the eight planets in our solar system. The DataFrame has an index column **Name** and columns **Large_Moons** for the number of large moons, **Icy_Moons** for the number of smaller icy moons, and **Rocks** for the number of asteroid-sized (very small) moons of each planet.

	Large_Moons	Icy_Moons	Rocks
Name			
Mercury	0	0	0
...			
Uranus	0	5	20
Neptune	1	0	20

Write code to do the following:

- Determine the number of planets in **dfplanet** that have more **Large_Moons** than **Icy_Moons**.
- Select the names of all planets that have moons that are **Rocks**. The result should be a pandas index or a list of strings.
- Add a new column to the DataFrame called **Non_Rocks** which equals the number of **Large_Moons** plus **Icy_Moons**.
- Among the planets with **Large_Moons**, one has the largest number of **Rocks**. Identify the name of that planet as a string.