Laboratory Assignment 6

For each problem, create your class inside that problem, and return the class.

The problems should not take any parameters unless it is explicitly stated in the question.

Your grade for each problem will be divided by the class methods, so it is possible to get partial credit from a problem if you implemented less than the expected methods.

Problem 0 - Getter & Setter `[0 pts]`

In this function, write a **class** with the following properties and return the class. Your function should not take any parameters. **Check the example in the provided submission python file.**

Class properties:

- Class name should be **`p0`**.
- Class should take one parameter `(x)`, and should define and set an internal variable.

Class methods:

- **`get_value()`** that will return the internal variable value when called.
- **`set_value(x)`** that will update the internal variable with the given value.

Class constraints:

- `inputs : x ∈ ℤ`

Example Doctests:

```
>>> A = problem0()
>>> ainst = A(3)
>>> ainst.get_value()
3
>>> A = problem0()
>>> ainst = A(1)
>>> ainst.get_value()
1
>>> A = problem0()
>>> ainst = A(-4)
>>> ainst.get_value()
-4
>>> ainst.set_value(5)
```

```
>>> ainst.get_value()
5
```

Problem 1 - Integer only Getter & Setter [10 pts]

In this function, write a **class** with the following properties and return the class. Your function should not take any parameters.

Class properties:

- Class name should be **`p1`**.
- Class should take one parameter `(x)`, and should define and set an internal variable. Only integers should be allowed. If the passed parameter is any other type, it should set the internal variable to **`0`**.

Class methods:

- **`get_value()`** that will return the internal variable value when called.
- **`set_value(x)`** that will update the internal variable with the given value. Only integers should be allowed. If the passed parameter is any other type, it should just ignore silently.

Class constraints:

- `inputs : x ∈ ℤ`

Example Doctests:

```
>>> A = problem1()
>>> ainst = A(3.5)
>>> ainst.get_value()
0
>>> A = problem1()
>>> ainst = A([1])
>>> ainst.get_value()
0
>>> A = problem1()
>>> ainst = A(None) # Creating an instance with None as parameter
>>> ainst.get_value()
0
>>> A = problem1()
>>> ainst = A(-4)
>>> ainst.get_value()
-4
>>> ainst.set_value(7.6)
>>> ainst.get_value()
-4
>>> ainst.set_value(3)
>>> ainst.get_value()
3
```

Problem 2 - Rectangle [10 pts]

In this function, write a **class** with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a rectangle with basic properties such as *area* and *perimeter*.
- Class name should be **`p1`**.
- Class should take two parameters `(x, y)`, and should define and set internal variables. These are the sides of the rectangle.

Class methods:

- **`get_area()`** that will return the *area* of the rectangle.
- **`get_perimeter()`** that will return the *perimeter* of the rectangle.

Class constraints:

- `inputs : {x, y} ∈ ℤ`

Example Doctests:

```
>>> A = problem2()
>>> ainst = A(3, 5)
>>> ainst.get_area()
15
>>> ainst.get_perimeter()
16
```

Problem 3 - Grades [12 pts]

In this function, write a class with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a list of grades and calculate and hold the basic properties such as minimum, maximum, median and mean grades.
- Class name should be **`Grades`**.
- Class should not take any parameter, however should initialize internal values to `0.0`.

Class methods:

- **`add_grade(x)`** that will add the given grade to the internal list.
- **`remove_grade(x)`** that will return the grade from the internal list if it exists, if it does not exist, it will just ignore it.
- **`get_min()`** return the minimum grade.

- **`get_max()`** return the maximum grade.
- **`get_mean()`** return the mean(average) grade.
- **`get_median()`** return the median grade. If the number of grades are even, returns the average of the middle two grades (assuming the grades are sorted).

Class constraints:

- `inputs : None`

Example Doctests:

```
>>> A = problem3()
>>> ainst = A()
>>> ainst.get_min()
0.0
>>> ainst.get_max()
0.0
>>> ainst.get_mean()
0.0
>>> ainst.get_median()
0.0
>>> ainst.add_grade(3)
>>> ainst.add_grade(5)
>>> ainst.get_min()
3.0
>>> ainst.get_max()
5.0
>>> ainst.get_mean()
4.0
>>> ainst.get_median()
4.0
>>> ainst.add_grade(10)
>>> ainst.get_min()
3.0
>>> ainst.get_max()
10.0
>>> ainst.get_mean()
6.0
>>> ainst.get_median()
5.0
>>> ainst.remove_grade(15)
>>> ainst.remove_grade(10)
>>> ainst.get_min()
3.0
>>> ainst.get_max()
5.0
>>> ainst.get_mean()
```

```
4.0
>>> ainst.get_median()
4.0
```

Problem 4 - Movie [14 pts]

In this function, write a class with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a movie and get its properties.
- Class name should be **`Movie`**.
- Class should take mandatory and optional parameters. Mandatory ones are movie_name, director, and year, optional parameters are rating and length.
    - `movie_name` is a string
    - `director` is a string
    - `year` is a number between [1920 - 2021]
    - `rating` is a float. Should be between [0.0 - 10.0]. Other inputs should be ignored. Default is 0 mins. Default is 0.0
    - `length` is an integer in minutes. Should be between [0 - 500]. Other inputs should be ignored. Default is 0 mins.

Class methods:

- **`get_movie_name()`** that will return the movie name.
- **`get_director()`** that will return the director name.
- **`get_year()`** that will return the year that the movie is made.
- **`get_rating()`** return the movie rating.
- **`get_length()`** return the movie length.
- **`set_rating(x)`** set the movie rating. Should be a float and between [0.0 - 10.0]. Other inputs should be ignored.
- **`set_length(x)`** set the movie length. Should be an integer and between [0 - 500]. Other inputs should be ignored.

Problem 5 - Movie Catalog [16 pts]

In this function, write a class with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a movie catalog using the movie class that we defined in Problem 4.
    - You can either re-implement the `Movie` class here, or call `problem4()` within Problem 5 to access that class.
- Class name should be **`MovieCatalog`**.
- Class should take one parameter called filename, which will point to a txt file with a list of movies. You should parse this file, and create a `Movie` class instance for each movie and add it to an internal list. (Thus the list should hold a bunch of `Movie` class instances).

Class methods:

- **`add_movie(movie_name, director, year, rating=0.0, length=0)`** will add the given movie to the list **if it is not already in the list**. (Do not forget to create a new `Movie` instance for this movie.)
- **`remove_movie(movie_name)`** that will remove the movie from the list if it exists. Ignore otherwise.
- **`get_oldest()`** that will return the **name** of the oldest movie in the list.
- **`get_lowest_ranking()`** that will return the **name** of the lowest ranking movie in the list.
- **`get_highest_ranking()`** that will return the **name** of the highest ranking movie in the list.
- **`get_by_director(director)`** that will return **a list of movies** that are made by the given director. If the director has no movies in the movie list, return an empty list.

Example Doctests:

```
>>> A = problem5()
>>> ainst = A('movies.txt')
>>> ainst.add_movie('The Usual Suspects', 'Bryan Singer', 1995, 8.5, 106)
>>> ainst.get_oldest()
'The Godfather'
>>> ainst.get_highest_ranking()
'The Shawshank Redemption'
>>> ainst.get_lowest_ranking()
'The Usual Suspects'
>>> ainst.get_by_director('Nuri Bilge Ceylan')
[]
>>> ainst.get_by_director('Francis Ford Coppola')
['The Godfather', 'The Godfather: Part II']
```

Problem 6 - Node `[12 pts]`

In this function, write a class with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a node in the 3D coordinate space.
- Class name should be **`Node`**.
- Class should take three parameters `(x, y, z)` that will hold the node coordinates in each direction.

Class methods:

- **`get_node()`** that will return the x, y, and z coordinates as a tuple in `x, y, z` order.
- **`get_distance()`** that will return the distance from the *origin* `(0, 0, 0)`.
- **`__add__`** to add two nodes and return a new Node instance.
- **`__str__`** to print the coordinates using `<x, y, z>` format. Make sure there is a space only after commas.

- **__gt__** greater than check. Return True if the magnitude of the first Node is greater than the second one. False otherwise.
- **__ge__** greater equal check. Return True if the magnitude of the first Node is greater than or equal to the second one. False otherwise.
- **__lt__** less than check. Return True if the magnitude of the first Node is less than the second one. False otherwise.
- **__le__** less equal check. Return True if the magnitude of the first Node is less than or equal to the second one. False otherwise.
- **__eq__** equality check. Return True if both nodes are at the same coordinate. False otherwise.

Class constraints:

- `inputs : {x, y, z} ∈ ℤ`

Example Doctests:

```
>>> N = problem6()
>>> ninst1 = N(3, 4, 5)
>>> ninst1.get_node()
(3, 4, 5)
>>> res = ninst1.get_distance()
>>> import math
>>> math.isclose(res, 7.07, rel_tol=1e-2)
True
>>> print(ninst1)
<3, 4, 5>
>>> ninst2 = N(-2, 3, 5)
>>> ninst3 = ninst1 + ninst2
>>> print(ninst3)
<1, 7, 10>
>>> ninst2 > ninst1
False
>>> ninst2 >= ninst1
False
>>> ninst2 < ninst1
True
>>> ninst2 <= ninst1
True
>>> ninst2 == ninst1
False
>>> ninst2 = N(-3, 4, 5)
>>> ninst2 == ninst1
False
>>> ninst2 = N(3, 4, 5)
>>> ninst2 == ninst1
```

```
True
>>> ninst2 is ninst1
False
```

Problem 7 - Node Cloud [14 pts]

In this function, write a class with the following properties and return the class. Your function should not take any parameters.

Class properties:

- The purpose of the class is to define a Node Cloud that consists of Nodes. These Nodes are the ones that we defined in Problem 6.
    - You can either re-implement the Node class here, or call `problem6()` within Problem 7 to access that class.
- Class name should be **`NodeCloud`**.
- Class should take one parameter `(n)`, and this parameter is used to tell how many Nodes there are in the Node Cloud.
- It should create a list of Nodes with random x, y, z coordinates. (These should be Node class instances)
    - You can use the `randint` function from the `random` library for generating random numbers. (`from random import randint`)
    - Coordinate values should be between `-20` and `20` (both included).

Class methods:

- **`get_nodes()`** that will return all the node coordinates as a list of tuples. Node coordinates should be in `x, y, z` order.
- **`get_outermost()`** that will return the node coordinates as a tuple that is furthest from the origin `(0, 0, 0)`. Node coordinates should be in `x, y, z` order.
- **`add_node(x, y, z)`** that will create a Node instance using the given parameters as coordinates and add it to its list of Nodes **if the Node does not already exist.**
    - if a node with the given coordinates exists, the function should ignore it.
- **`get_sum()`** that will sum all the Nodes in x, y, z coordinates, create a Node instance with the final values and return that instance.

Class constraints:

- `inputs : n` $\in \mathbb{Z}^+$

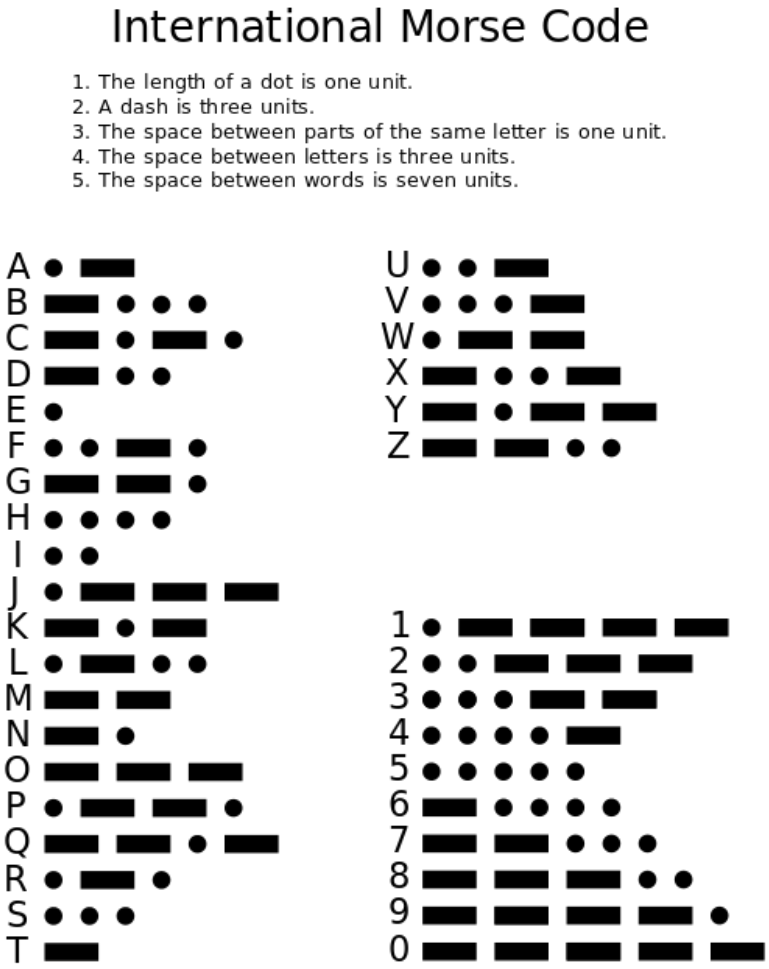
Example Doctests:

```
>>> A = problem7()
>>> ainst = A(5)
>>> nodes = ainst.get_nodes()
>>> len(nodes)
5
>>> isinstance(nodes, list)
True
```

```
>>> isinstance(nodes[0], tuple)
True
>>> ainst.add_node(21, 21, 21)
>>> nodes = ainst.get_nodes()
>>> len(nodes)
6
>>> ainst.get_outermost()
(21, 21, 21)
>>> s = ainst.get_sum()
>>> len(s.get_node())
3
```

Problem 8 - Encoder `[12 pts]`

In this function, write a class with the following properties and return the class. Your function should not take any parameters.



Class properties:

- The purpose of the class is to apply given encodings on a string.
- Class name should be **`Encoder`**.
- Class should take one parameter `(x)` that will be used as the source string.

- Class should accept only alphanumeric characters. `{a-z,A-Z,0-9}.` Other characters should be ignored.

Class methods:

- **`__str__`** to print the stored string.
- **`morse()`** that will return the morse encoded version **as a list of strings**. Conversion table is given above. Encode long dashes with a dash character, and dots with a dot character.
- **`binary()`** that will return the 7-bit ascii binary encoded version **as a string**. [1]
- **`hex()`** that will return the hex encoded version as a string. [1]

Class constraints:

- `inputs : {x} ∈ english letters and numbers.`

Example Doctests:

```
>>> A = problem8()
>>> ainst = A("test")
>>> print(ainst)
test
>>> ainst.morse()
['-', '.', '...', '-']
>>> ainst.binary()
'11101001100101111100111110100'
>>> ainst.hex()
'74657374'
>>> ainst = A("çELık-!")
>>> print(ainst)
ELk
>>> ainst.morse()
['.', '.-..', '-.-']
>>> ainst.binary()
'100010110011001101011'
>>> ainst.hex()
'454c6b'
```

[1] https://www.asciitable.com