10.009 The Digital World

Term 3. 2019

Problem Set 8 (for Week 8)

Most recent updated: March 25, 2019

Due dates:

- **Cohort session problems** : Following week: Tuesday 11:59pm.
- **Homework problems** : Same as for the cohort session problems.
- **Exercises**: These are practice problems and will not be graded. You are encouraged to solve these to enhance your programming skills. Being able to solve these problems will likely help you prepare for the end term examination.

Objectives

1. Understand what is object-oriented programming (OOP).
2. Learn classes and methods.

Note: Solve the programming problems listed using your favorite text editor. Make sure you save your programs in files with suitably chosen names, **and try as much as possible to write your code with good style (see the style guide for python code)**. In each problem find out a way to test the correctness of your program. After writing each program, test it, debug it if the program is incorrect, correct it, and repeat this process until you have a fully working program. Show your working program to one of the cohort instructors.

Problems: Cohort sessions

1. *Classes and Methods: Make a Coordinate class.* Implement a class named `Coordinate` that represents the coordinates of a point in two-dimensional space. The class has two attributes: x and y. It also has the following methods:

   - `__init__(self, x=0, y=0)` : This method takes in x and y to initialize the attributes. If x and y are not provided, the attributes are initialized to 0.
   - `magnitude(self)`: This method returns the distance from the origin, which is defined as $\sqrt{x^2 + y^2}$.
   - `translate(self, dx, dy)`: This method translates the point by `dx` and `dy` so that the point now has the coordinates `(x + dx, y + dy)`.
   - `__eq__(self, other)`: This method takes another `Coordinate` object and returns `True` if the attributes of `other` have the same values as the attributes of `self`, and `False` otherwise.

   A sample interactive session is shown below.

```
>>> p = Coordinate()
>>> print(p.x, p.y)
0 0
>>> print(p.magnitude())
0.0
>>> p.x = 3
>>> p.y = 4
>>> print(p.magnitude())
5.0
>>> q = Coordinate(3,4)
>>> print(p == q)
>>> True
>>> q.translate(1, 2)
>>> print(q.x)
4
>>> print(p == q)
>>> False
```

2. *Clases and Methods: Make a Celsius class:* Implement a class named `Celsius` that represents a temperature in degrees Celsius.

   It has one attribute called `_temperature` that stores the temperature value. Note that by convention, an underscore prefix (like `_temperature`) designates a private attribute that should only be accessed within the class itself. Public attributes are those that are intended to be accessed anywhere. While Python does not prevent access to private attributes, it is a convention that is generally followed by Python programmers.

   The class also has a property `temperature` (See Python docs here: `https://docs.python.org/3/library/functions.html#property`).

The methods that the class contains are as follows:

- `__init__(self,temperature)`: This method initializes the property `temperature` according to the argument `temperature`. If it is not specified, the property should be initialized to 0.
- `to_fahrenheit(self)`: This method returns the temperature in degrees Fahrenheit.
- `get_temperature(self)`: This method returns the temperature in degrees Celsius.
- `set_temperature(self, value)`: This method assigns the argument `value` to the attribute `_temperature`. However, if `value` is less than -273, the smallest physically possible temperature of -273 will be assigned to `_temperature`.
- `temperature`: This is a property that allows access to the `_temperature` attribute. Remember that the underscore is a convention denoting a private attribute. Thus `temperature` behaves like a public attribute. Assign the `get_temperature` and `set_temperature` methods as its getter and setter.

A sample interactive session using class `Coordinate` is shown below.

```
>>> c=Celsius()
>>> print(c.temperature)
0
>>> c.temperature=32
>>> print(c.to_fahrenheit())
89.6
>>> c.temperature=-300
>>> print(c.temperature)
-273
```

3. *Classes and Methods: Stopwatch:* Write a class named `StopWatch`. The class contains:

   - Two instance attributes, i.e. `start_time` and `end_time`.
   - During object instantiation, `start_time` should be initialized to the current time and `end_time` should be initialized to -1. You should not use optional arguments in `__init__()` for this problem.
   - A method named `start` that resets the `start_time` to the current time and `end_time` to -1.
   - A method named `stop` that sets the `end_time` to the current time.
   - A method named `elapsed_time` that returns the elapsed time for the stopwatch in seconds as a float. Round the value of the elapsed time to one decimal place. If `end_time` is not valid (ie., it is -1), return `None`.

     Hint: To get the current time, you will need to use a function from Python's `time` module. Read the documentation to find out what it is. `https://docs.python.org/3/library/time.html`.

To test, run this script:

```
sw = StopWatch()
time.sleep(0.1)
sw.stop()
print(sw.elapsed_time())
sw.start()
time.sleep(0.2)
print(sw.elapsed_time())
sw.stop()
print(sw.elapsed_time())
```

It should output the following:

```
0.1
None
0.2
```

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

4. *Classes and Methods:* Write a class `Line` that represents the following equation: $y = c_0 + c_1 x$. It will have the following methods:

   (a) The `__init__()` method takes two parameters, c_0 and c_1. During object instantiation, the two parameters should initialize the coefficients c_0 and c_1 in the expression for the straight line: $y = c_0 + c_1 x$.

   (b) The `__call__` method returns the value of y according to $y = c_0 + c_1 x$.

   (c) The `table(L, R, n)` method samples the function at n points for $L \leq x \leq R$ and creates a table of x and y values, with each value formatted to 2 decimal places in a field of width 10.

   Hint: Look up string formatting in Python docs to find out how to do the formatting required by the method `table(L,R,n)`.

Refer to the following sample interactive session.

```
>>> line=Line(1,2)
>>> line(2)
5
>>>
>>> print(line.table(1,5,4))
      1.00      3.00
      2.33      5.67
      3.67      8.33
      5.00     11.00
```

For the test cases below, use the inputs in the same way they were used in the sample interactive session given above. Input 1 refers to the input x, c_0 and c_1 in `line(x)` and for instantiating the class Line, output 1 refers to the output after calling `line(x)`, input

2 refers to the input L, R, and n in `line.table(L,R,n)` and output 2 refers to the output after printing `line.table(L,R,n)`.

Test Cases:

Test case 1

Input 1: $x = 2, c_0 = 1, c_1 = 2$

Output1: 5.0

Input 2: L = 1, R = 5, N = 4

Output 2:

1.00	3.00
2.33	5.67
3.67	8.33
5.00	11.00

Test case 2

Input 1: $x = 2, c_0 = -1, c_1 = 2$

Output 1: 3.0

Input 2: L = -1, R = 5, N = 10

Output 2:

-1.00	-3.00
-0.33	-1.67
0.33	-0.33
1.00	1.00
1.67	2.33
2.33	3.67
3.00	5.00
3.67	6.33
4.33	7.67
5.00	9.00

Test case 3

Input 1: $x = 2, c_0 = 3, c_1 = 4$

Output 1: 11.0

Input 2: L = 1, R = 5, N = 15

Output 2:

1.00	7.00
1.29	8.14
1.57	9.29

1.86	10.43
2.14	11.57
2.43	12.71
2.71	13.86
3.00	15.00
3.29	16.14
3.57	17.29
3.86	18.43
4.14	19.57
4.43	20.71
4.71	21.86
5.00	23.00

Test case 4

Input 1: $x = 2, c_0 = 3, c_1 = 4$

Output 1: 11.0

Input 2: L = 1, R = 1, N = 15

Output 2:

1.00	7.00

Test case 5

Input 1: $x = 2, c_0 = 3, c_1 = 4$

Output 1: 11.0

Input 2: L = 1, R = 5, N = 0

Output 2:

Error in printing table

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

Problems: Homework

1. *Classes and Methods: Time:* Write a class named `Time`. The class contains:

   - The instance attributes `_hours`, `_minutes`, and `_seconds` that represent a time.
   - During object instantiation of a `Time` object, it should initialize the attributes `_hours`, `_minutes`, and `_seconds` using the input parameters.
   - A property named `elapsed_time` with a getter and setter. The getter returns the total number of seconds that has elapsed since the time `00:00:00`, and the setter takes number of seconds elapsed and sets the `_hours`, `_minutes` and `_seconds` instance attributes accordingly.
   - Define the `__str__` method so that when you print a `Time` object you would get the following string representation: `‘‘Time: H:M:S’’`, where H is hour, M is minute, and S is seconds. For example, `‘‘Time: 10:19:10’’`

   Note that `_hours` can only go from 0 to 23, and `_minutes` and `_seconds` can only go from 0 to 59. You can assume that all our test cases will have valid values within those ranges. If the given number of elapsed seconds is so large that it overflows (ie. it is longer than one day), simply set the appropriate time in the different day. For example, if the elapsed time is 555550 seconds, `_hours` is 10, `_minutes` is 19, and `_seconds` is 10. An example of a test program is as follows:

   ```
   t = Time(10, 19, 10)
   print(t.elapsed_time)
   t.elapsed_time = 555550
   print(t.elapsed_time)
   print(t)
   ```

   It should output:

   ```
   37150
   37150
   Time: 10:19:10
   ```

   Submission to Vocareum: Please submit your entire class with all the above methods implemented.

2. *Classes and Methods: A bank account class:* Implement the concept of a bank account as a class named `Account`. The bank account has some data, namely the name of the account holder, the account number, and the current balance. The current balance can be a negative number. Three things we can do with an account is withdraw money, deposit money into the account, and print out the account information. These actions are modelled by methods inside the class. Implement the following methods:

- `__init__(self, owner, account_number, amount)`
- `deposit(self, amount)`
- `withdraw(self, amount)`
- `__str__(self)`

Name the attributes of your class as

- `_owner`
- `_account_number`, and
- `_balance`

You may test your class at the Python shell using the following commands:

```
>>> a1 = Account('John Olsson', '19371554951', 20000)
>>> a2 = Account('Liz Olsson', '19371564761', 20000)
>>> a1.deposit(1000)
>>> a1.withdraw(4000)
>>> a2.withdraw(10500)
>>> a1.withdraw(3500)
>>> print(a1)
John Olsson, 19371554951, balance: 13500
>>> print(a2)
Liz Olsson, 19371564761, balance: 9500
```

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

3. *Classes and Methods:* A class for estimating the derivative of a function $f(x)$ at x takes the form:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

where h is a small change in x.

The goal of this exercise is to use the formula above to differentiate a mathematical function $f(x)$ implemented as a Python function `f(x)`.

Implement class `Diff` with two special methods. The `__init__()` method takes in a function object and also an optional argument `h`. The default value of `h` is 1e-4. Implement also the special method `__call__` so that the instance is callable. This method returns the approximation of the derivative of the function using the formula above.

The following code shows you how this class can be used to compare the exact value of the derivative versus its approximation for the function $f(x) = (1/4)x^4$.

```
def f(x):
    return 0.25*x**4

df = Diff(f)    # make function-like object df
```

```
# df(x) computes the derivative of f(x) approximately:
for x in [1, 5, 10]:
    df_value = df(x) # approx value of derivative of f at point x
    exact = x**3     # exact value of derivative
    print("f'({:d})={:g} (error={:.2E})".format(x, df_value, exact-
        df_value))
```

The following are test cases for the derivative of $f(x) = ln(x)$, evaluated at $x = 10$ for various values of h.

Test Cases:

Test case 1

Input: x = 10.0, f = log, h = 0.1,

Output: (0.09950330853167877, 0.0004966914683212365) # derivative, approximation error

Test case 2

Input: x = 10.0, f = log, h = 0.5,

Output: (0.09758032833886343, 0.0024196716611365743)

Test case 3

Input: x = 10.0, f = log, h = 1.0E-5,

Output: (0.09999994996512383, 5.003487617283309e-08)

Test case 4

Input: x = 10.0, f = log, h = 1.0E-9,

Output: (0.1000000082740371, -8.274037094357922e-09)

Test case 5

Input: x = 10.0, f = log, h = 1.0E-11,

Output: (0.0999644811372491, 3.551886275091065e-05)

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

4. *Polynomial class:* This exercise focuses on a class `Polynomial` for polynomials. The coefficients in the polynomial will be given as a list, and is used as a parameter during object instantiation. Index number i in this list represents the coefficients of the x^i term in the polynomial. For example, writing `Polynomial([1,0,-1,2])` defines the polynomial:

$$p(x) = 1 + 0x - 1x^2 + 2x^3 = 1 - x^2 + 2x^3.$$

   (a) Polynomials can be added and subtracted(by adding/subtracting the coefficients). The class should implement the 'magic' `__add__` and `__sub__` methods.

   (b) Implement `__call__` so that we can evaluate the value of the polynomial expression given a certain value of x

(c) Implement the `__mul__` method for polynomial multiplication. Let $p(x) = \sum_{i=0}^{M} c_i x^i$ and $q(x) = \sum_{j=0}^{N} d_j x^j$ be two polynomials. Their product is:

$$\sum_{i=0}^{M} \sum_{j=0}^{N} c_i d_j x^{i+j}.$$

(d) Implement two different methods for differentiating the polynomial:

$$\frac{d}{dx} \sum_{i=0}^{n} c_i x^i = \sum_{i=1}^{n} i c_i x^{i-1}.$$

The first method is `differentiate` which returns `None` but changes the coefficients of the current polynomial instance on which it is called.

The second method is `derivative`, which returns a new `Polynomial` instance with containing the new coefficients.

An interactive session for `Polynomial` is as follows:

```
>>> p1 = Polynomial([1, -1])
>>> p2 = Polynomial([0, 1, 0, 0, -6, -1])
>>> p3 = p1 + p2
>>> print(p3.coeff)
[1, 0, 0, 0, -6, -1]
>>> p4 = p1*p2
>>> print(p4.coeff)
[0, 1, -1, 0, -6, 5, 1]
>>> p5 = p2.derivative()
>>> print(p5.coeff)
[1, 0, 0, -24, -5]
>>> p = Polynomial([1, 2, 3])
>>> q = Polynomial([2, 3])
>>> r=p-q
>>> print(r.coeff)
[-1, -1, 3]
>>> r=q-p
>>> print(r.coeff)
[1, 1, -3]
>>>
```

Test Cases:

Test case 1

Input: [1, -1], [0, 1, 0, 0, -6, -1] # poly coeffs are added
Output: [1, 0, 0, 0, -6, -1]

Test case 2

Input: [1, -1], [0, 1, 0, 0, -6, -1], x = 3 # poly coeffs are subtracted and evaluated at x = 3
Output: [1, -2, 0, 0, 6, 1], 724 # resultant poly coeff and evaluated value

Test case 3

Input: [1, 2, 3, 4], [1, 2, 3, 4] # multiplication
Output: [1, 4, 10, 20, 25, 24, 16]

Test case 4

Input:	[1, 3, 5, 7, 9] # differentiation
Output:	[3, 10, 21, 36]

Test case 5

Input:	[2, 4, 6, 8, 10] # derivative - differentiation of polynomial copy
Output:	[2, 4, 6, 8, 10]

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

Problems: Exercises

1. *Classes and Methods: Make a function class:* Make a class named `F` that implements the function

$$f(x) = e^{-ax} sin(wx).$$

The function $f(x)$ calculates the value for the given variable x from constant parameters a and w.

Class `F` has instance attributes `a` and `w`. Apart from the `__init__` method, it also has the `__call__` method implemented to compute the values of $f(x)$ for a given value of `x`. Test the class with the following program.

```
from math import *
f = F(a=1.0, w=0.1)
print(f(x=pi))
```

Submission to Vocareum: Please submit your entire class with all the above methods implemented.

Test Cases:

Test case 1

Input: a=1.0, w = 0.1, x=pi
Output: 0.013353835137

Test case 2

Input: a = 3.0, w = 0.5, x=pi/2.0
Output: 0.00635214599841

Test case 3

Input: a = 5.0, w = 1.5 , x=pi/4.0
Output: 0.018203081084

Test case 4

Input: a = 5.0, w = 2.0, x=pi/6.0
Output: 0.06317573987774452

Test case 5

Input: a = 10.0, w = 3.0, x=pi/18.0
Output: 0.0872937481106

2. *Classes and Methods: Straight line class based on alternative definition:* Make a class `Line0` whose `__init__` method takes two points `p1` and `p2` (2- tuples or 2-lists) as input. The line passes through these two points. Implement `__call__` to take an `x` value and return the corresponding `y` value.

```
>>> line = Line0((0,-1), (2,4))
>>> print(line(0.5), line(0), line(1))
0.25 -1.0 1.5
```

Test Cases:

Test case 1

Input:	p1 = (0,-1), p2 = (2,4), x = 0.5,
Output:	0.25

Test case 2

Input:	p1 = (0,-1), p2 = (2,4), x = 0,
Output:	-1.0

Test case 3

Input:	p1 = (0,-1), p2 = (2,4), x = 1,
Output:	1.5

Test case 4

Input:	p1 = (3,3), p2 = (8,8), x = -1,
Output:	-1.0

Test case 5

Input:	p1 = (3,3), p2 = (8,8), x = 4.3,
Output:	4.3

End of Problem Set 8.