10.009 The Digital World

Term 3. 2015

Problem Set 8 (for Week 8)

Most recent updated: March 14, 2015

Due dates:

- **Cohort session problems** : Following week: Monday 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 midterm examination.

Objectives

1. Understand what is object-oriented programming (OOP).
2. Learn classes and methods.
3. Understand and learn how to use inheritance.

Note: Solve the programming problems listed below using the IDLE or Canopy editor. Make sure you save your programs in files with suitably chosen names and in an newly created directory. 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 square class.* Implement a class named `Square` that represents a square (a geometrical figure). The constructor method in the class assigns the dimension of the square, i.e., its length or width (both are the same) to the instance's attribute. In the class define a method `getArea()` that returns the area of a square as a float. Define another method `setArea()` which sets the area of the square to a given value. Note that the dimension of a square can be an `int` or a `float`.

   Also implement the `__str__` method that it prints out the dimension of the square as shown in the sample interactive session below. Recall that the `__str__` method is a special and used to print a class. In this problem you are being asked to customize the default `__str__` method so that it prints as shown below. A sample interactive session using class `Square` follows.

```
>>> s = Square(6)
>>> s.getArea()
36
>>> s.setArea(100)
>>> s1 = Square(10)
>>> s1.getArea()
100.0
>>> s2 = Square(100)
>>> s2.getArea()
10000.0
>>> print s1   # Print object s1 using __str__.
Square of height and width 10.
```

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

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

   - The private data fields `startTime` and `endTime` with get methods `getStartTime()` and `getEndTime()`.
   - A constructor that initialises `startTime` with the current time and `endTime` with -1.
   - A method named `start()` that resets the `startTime` to the current time and `endTime` with -1.
   - A method named `stop()` that sets the `endTime` to the current time.
   - A method named `getElapsedTime()` that returns the elapsed time for the stop watch in milliseconds. If the `endTime` is not valid (-1), returns `None`.

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

3. *Classes and Methods: Define a straight line class.* Make a class `Line` whose constructor takes two coefficients c_0 and c_1 as input to represent: $y = c_0 + c_1x$. The constructor initializes the coefficients c_0 and c_1 in the expression for the straight line: $y = c_0 + c_1x$. The `__call__` method evaluates the function $y = c_0 + c_1x$. Also the `table()` method samples the function at n points and creates a table of x and y values, each formatted with 2 decimals in a field of width 10. Following is a 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
```

In general, method `table(L,R,n)` returns a table with n points for $L \leq x \leq R$

Test Cases:

Test case 1

Input: c0 = 1, c1 = 2, x = 2, L = 1, R = 5, N = 4

Output: 5.0

1.00	3.00
2.33	5.67
3.67	8.33
5.00	11.00

Test case 2

Input: c0 = -1, c1 = 2, x = 2, L = -1, R = 5, N = 10

Output: 3.0

-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: c0 = 3, c1 = 4, x = 2, L = 1, R = 5, N = 15

Output: 11.0

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: c0 = 3, c1 = 4, x = 2, L = 1, R = 1, N = 15
Output: 11.0

1.00	7.00

Test case 5

Input: c0 = 3, c1 = 4, x = 2, L = 1, R = 5, N = 0
Output: 11.0

Error in printing table

Submission to Tutor: 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 private data fields `hour`, `minute`, and `second` that represent a time.
   - A constructor that constructs a `Time` object that initializes `hour`, `minute`, and `second` using the input parameters.
   - The `get` methods for the data fields `hour`, `minute`, and `second`, respectively.
   - A method named `setTime(elapseTime)` that sets a new time for the object using the elapsed time in seconds. For example, if the elapsed time is 555550 seconds, the hour is 10, the minute is 19, and the second is 10. Notice that this method set the time with an elapse time calculated from time 0. If the hour exceeds 12 hours, it will start counting from zero again.

   Submission to Tutor: 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, typically the name of the account holder, the account number, and the current balance. Three things we can do with an account is withdraw money, put money into the account, and print out the account information. These actions are modeled by methods inside the class. Implement this bank account class so that we can pack the data and actions together into a new data type with one account corresponding to one variable in a program. Implement the class methods: constructor, `deposit()`, `withdraw()`, and `__str__()`, so that the class `Account` can be used as follows:

```
>>> 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 Tutor: Please submit your entire class with all the above methods implemented.

3. *Classes and Methods: A class for numerical differentiation.* A widely used formula for numerical differentiation of a function $f(x)$ takes the form:

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

The goal of this exercise is to use the above formula to automatically differentiate a mathematical function $f(x)$ implemented as a Python function `f(x)`. More precisely, the following code should work:

```
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)" % (x, df_value, exact-df_value)
```

Implement class `Diff`. Implement also the special method `__call__` so that the instance is callable as shown above. Test that the code above works. Include an optional argument `h` to the constructor in class `Diff` so that one can specify the value of `h` in the approximation. Apply class `Diff` to produce a table of the derivatives and the associated approximation errors for $f(x) = \ln x$, $x = 10$, and $h = 0.5, 0.1, 10^{-3}, 10^{-5}, 10^{-7}, 10^{-9}, 10^{-11}$.

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 Tutor: 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 can be given to the constructor as a list. Index number i

in this list represents the coefficients of the x^i term in the polynomial. That is, writing `Polynomial([1,0,-1,2])` defines a polynomial:

$$1 + 0x - 1x^2 + 2x^3 = 1 - x^2 + 2x^3.$$

(a) Polynomials can be added (by just adding the coefficients) so the class needs to have an `__add__` method.

(b) Implement also the method `__sub__` in class `Polynomial`.

(c) A `__call__` method is natural for evaluating the polynomial, given a value of x.

(d) Implement the `__mul__` method for polynomial multiplications. 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:

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

(e) Implement a method for differentiating the polynomial:

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

Implement two methods for differentiation. Let `p.differentiate()` be an implementation that does not return anything, but the coefficients in the `Polynomial` instance `p` are altered. The other method is implemented by `p.derivative()`, which returns a new `Polynomial` object with coefficients corresponding to the derivative of `p`.

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 Tutor: 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

$$f(x;a,w) = e^{-ax} sin(wx).$$

Class `F` contains the `value(x)` method to compute the values of $f(x)$ for given values of `x`; a and w are class attributes. Test the class with the following program.

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

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

Test Cases:

Test case 1

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

Test case 2

Input:	a = 3.0, w = 0.5
Output:	0.00635214599841

Test case 3

Input:	a = 5.0, w = 1.5
Output:	0.018203081084

Test case 4

Input:	a = 5.0, w = 2.0
Output:	11.8716456895

Test case 5

Input:	a = 10.0, w = 3.0
Output:	0.0872937481106

2. *Classes and Methods: Straight line class based on alternative definition.* Make a class `Line0` whose constructor takes two points `p1` and `p2` (2- tuples or 2-lists) as input. The line passes through these two points. A `value(x)` method computes a value on the line at the point `x`. Following is a sample interactive session.

```
>>> line = Line0((0,-1), (2,4))
>>> print line.value(0.5), line.value(0), line.value(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 # value(x)

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.