COMPSCI 105 Summer School 2016
Assignment One

Worth: 6% of your final grade
Deadline: 9:00pm, Monday 18th January

Please submit what you have completed before the deadline
No extensions, unless *exceptional* circumstances apply
(discuss with Paul as soon as possible)

What you need to do

For this assignment, your task is to define *four* Python classes. The classes you must define are:

- Polynomial
- Student
- Course
- Area

All four class definitions should appear in a single source file, which must be called Asst1.py. The general structure of your submission will therefore be:

Asst1.py

```
class Polynomial:
    ....
    ....

class Student:
    ....
    ....

class Course:
    ....
    ....

import json
class Area:
    ....
    ....
```

Please note: it is very important that the Asst1.py source file you submit for marking does not contain any program statements other than the four class definitions (i.e. there should be no main() function and no statements appearing outside the class definitions).

Your submission will be marked using a separate program that imports your class definitions, and tests your implementations by creating instances of the classes and then calls the methods you have defined.

You must submit this single source file to the Assignment Drop Box:

https://adb.auckland.ac.nz

prior to 9:00pm on Monday 18th January.

Paul Denny
January 2016

Question 1: The Polynomial class (40 marks)

For this question you must define a class called Polynomial. This class definition must include 6 methods:

- __init__() - the initialiser for the class
- __str__() - returns a formatted string representation of a Polynomial object
- scale() - scales a Polynomial object by a specified factor
- add() - returns a new Polynomial object that is the sum of two Polynomials
- evaluate() - evaluates a Polynomial object for a given *x*-value
- get_degree() - returns the degree of the Polynomial object

Background

A polynomial is a mathematical expression that is the sum of a series of terms that each consist of variable (we will use *x*) which is taken to a given power and multiplied by a given coefficient. For example, the following are two different polynomials:

$$2x^4 + x^2 - 7x + 13$$

$$x^3 - 5x^2 + 1$$

We can use a *Python list* to easily represent a polynomial. The list consists only of the *coefficient* values, as the position of the values in the list can be used to determine the power of *x*. For example, the two polynomials above could be unambiguously represented by the two lists:

[2, 0, 1, -7, 13]

[1, -5, 0, 1]

For this question you must define a class called Polynomial that represents a polynomial.

The __init__() method

The initialiser method should take *one input* - a Python list - that stores the coefficients of the polynomial, just as illustrated in the examples above.

For example, consider a program that imports the Polynomial class definition from your Asst1.py source file. This program could then create the two polynomials shown above using the following statements:

```
p1 = Polynomial([2, 0, 1, -7, 13])
p2 = Polynomial([1, -5, 0, 1])
```

Once you have completed this initialiser method, you should define the __str__() method so that you can print out Polynomial objects, and see a formatted string representation. This format is described in detail next - please read this description carefully.

The __str__() method
This function is called *automatically* by Python whenever a Polynomial object is provided as input to the print() function. The function should return a string that represents the polynomial.

The string returned by this method should use the '^' character to indicate exponentiation. Several examples are shown below. The following code:

```
p1 = Polynomial([2,3,4])
p2 = Polynomial([10,5,2,0,2])
p3 = Polynomial([1,1,1,1,1])
p4 = Polynomial([6,0,0,0])
p5 = Polynomial([-2,3,-4])
p6 = Polynomial([10,-5,-2,0,2])
p7 = Polynomial([-1,-1,-1,-1,-1])
p8 = Polynomial([1,0,-2])
p9 = Polynomial([1])
p10 = Polynomial([0])

print('p1 =', p1)
print('p2 =', p2)
print('p3 =', p3)
print('p4 =', p4)
print('p5 =', p5)
print('p6 =', p6)
print('p7 =', p7)
print('p8 =', p8)
print('p9 =', p9)
print('p10 =', p10)
```

should produce the output:

```
p1 = 2x^2 + 3x + 4
p2 = 10x^4 + 5x^3 + 2x^2 + 2
p3 = x^4 + x^3 + x^2 + x + 1
p4 = 6x^3
p5 = -2x^2 + 3x - 4
p6 = 10x^4 - 5x^3 - 2x^2 + 2
p7 = -x^4 - x^3 - x^2 - x - 1
p8 = x^2 - 2
p9 = 1
p10 = 0
```

Please pay **very careful** attention to the following points:

- If the coefficient of a term is 1, then the coefficient is *not* shown
  - i.e. x + 2 rather than 1x + 2
- If the coefficient of a term is negative, then a *single minus sign* should be shown rather than a plus and minus sign
  - i.e. x^2 - 2x + 2 rather than x^2 + -2x + 2
- There should be a single space character on either side of a '+' or '-' sign that appears internal to the polynomial. The only exception to this rule is at the very start of the polynomial. For the very first term, if the coefficient is positive then no sign should be shown, but if the coefficient

is negative then a single minus sign should be shown *without any* surrounding spaces
- The terms should appear in *descending order* of exponent value - which is the same order as the corresponding coefficients appear in the list that was used to initialise the object.

The evaluate() method

The evaluate method takes an input value of *x* (which will be an *integer*) and it should return the *value of the polynomial* if evaluated using that value for *x*. Because the input value is an integer, the output value *must also be an integer*.

For example, the following code:

```
p1 = Polynomial([2,3,4])
p2 = Polynomial([10,5,2,0,2])
p3 = Polynomial([1,1,1,1,1])
p4 = Polynomial([6,0,0,0])

print('p1(4) =', p1.evaluate(4))
print('p2(0) =', p2.evaluate(0))
print('p3(-2) =', p3.evaluate(-2))
print('p4(3) =', p4.evaluate(3))
```

should produce the output:

```
p1(4) = 48
p2(0) = 2
p3(-2) = 11
p4(3) = 162
```

The get_degree() method

The *degree* of a polynomial is simply the value of the greatest exponent. When we print our polynomial, this is simply going to be the exponent value of the *very first* term. For example, the following code:

```
p1 = Polynomial([2,3,4])
p2 = Polynomial([10,5,2,0,2])
p3 = Polynomial([1,1,1,1,1])
p4 = Polynomial([6,0,0,0])

print('Degree of p1 =', p1.get_degree())
print('Degree of p2 =', p2.get_degree())
print('Degree of p3 =', p3.get_degree())
print('Degree of p4 =', p4.get_degree())
```

should produce the output:

```
Degree of p1 = 2
Degree of p2 = 4
Degree of p3 = 4
Degree of p4 = 3
```

The scale() method
The scale method takes a single integer as input, and it should multiply the polynomial by that value as a factor. This simply means that each coefficient of the polynomial is multiplied by the factor.

Please note that in this implementation, Polynomial objects are *mutable* as the scale function *actually changes* the instance on which it is called.

For example, the following code:

```
p1 = Polynomial([2,3,4])
p2 = Polynomial([10,5,2,0,2])
p3 = Polynomial([1,1,1,1,1])
p4 = Polynomial([6,0,0,0])

p1.scale(5)
p2.scale(-5)
p3.scale(5)
p4.scale(5)

print(p1)
print(p2)
print(p3)
print(p4)
```

should produce the output:

```
10x^2 + 15x + 20
-50x^4 - 25x^3 - 10x^2 - 10
5x^4 + 5x^3 + 5x^2 + 5x + 5
30x^3
```

The add() method
Finally, the add method should add two Polynomial objects together and return a new Polynomial object representing their sum. Adding two polynomials is just a matter of combining all of the terms from each polynomial, and adding the coefficients of any terms that have the same exponent.

For example, the following code:

```
p1 = Polynomial([2,3,4])
p2 = Polynomial([10,5,2,0,2])
p3 = Polynomial([1,1,1,1,1])
p4 = Polynomial([6,0,0,0])

sum1 = p1.add(p2)
sum2 = p3.add(p4)

print('sum1 =', sum1)
print('sum2 =', sum2)
```

should produce the output:

```
sum1 = 10x^4 + 5x^3 + 4x^2 + 3x + 6
sum2 = x^4 + 7x^3 + x^2 + x + 1
```

A few things to be careful about

If two polynomials are added together, and one polynomial has a positive coefficient and the other has a negative coefficient, then those may cancel out. An example is as follows:

```
x2 = Polynomial([2,-3,0,4])
y2 = Polynomial([-2,3,1,1])
z2 = x2.add(y2)

print('z2 =', z2)
print('Degree of z2 =', z2.get_degree())
```

In this case, the correct output should be:

```
z2 = x + 5
Degree of z2 = 1
```

Also, each Polynomial object should store its own coefficients in a distinct list. As an example, consider the situation below where two Polynomial objects are created using a single list:

```
my_list = [1,2,3]

share1 = Polynomial(my_list)
share2 = Polynomial(my_list)

share1.scale(2)

print('share1 =', share1)
print('share2 =', share2)
```

In this case, only the first polynomial (share1) is actually scaled - the scale method is called on the instance share1 - so there should be *no change* to the second polynomial (share2). The output here should therefore be:

```
share1 = 2x^2 + 4x + 6
share2 = x^2 + 2x + 3
```

Question 2: The Student class and the Course class (45 marks)

For this question you must define two classes - Student and Course.

The Student class definition must include 5 methods:

- __init__() - the initialiser for the class
- __str__() - returns a formatted string representation of a Student object
- get_id_number() - returns the ID number of a Student object
- set_marks() - updates the "practical" and "theory" marks of a Student object
- passes() - determines whether a student's marks would earn a passing grade

The Course class definition must include 7 methods:

- __init__() - the initialiser for the class
- __str__() - returns a formatted string representation of a Course object
- set_practical_requirement() - indicates whether or not both "theory" and "practical" mark passes are required for an overall pass in the course
- add_students() - enrols a list of Student objects into the course
- set_marks() - updates the marks for an enrolled Student object
- passing_ids() - returns a string consisting of the ID numbers of all students passing the course
- class_list() - returns a string displaying all students in the course

Background

A Student object stores the name, ID number, practical mark and theory mark of a student. A Course object stores a list of students who are currently enrolled in the course. You could imagine a course management system that uses a Course object to keep track of the progress of students in a course. In particular, methods could be called on the Course object to update the marks of the students, and to produce a list of all passing students.

The Student class

The initialiser method should take *two inputs* - the name of the student and their ID number (both of these are strings). In addition to storing this information, a Student object should also store two marks that represent the "practical" mark (out of 100) and the "theory" mark (out of 100) for a given student. The "practical" mark represents performance on the labs and assignments, and the "theory" mark represents performance on the test and final exam. Initially, both "practical" and "theory" marks should be 0.

For example, consider four Student objects created as follows:

```
s1 = Student('ann', '1234')
s2 = Student('bob', '2345')
s3 = Student('clare', '3456')
s4 = Student('dave', '4567')
```

The __str__() method of the Student class should return a formatted string representing information about the student. The format should display the student's name, followed by an underscore character, followed by the student's "practical" mark, followed by an underscore character, followed by the student's "theory" mark.

For example, using the four Student objects created as in the example above, the following code:

```
print(s1)
print(s2)
print(s3)
print(s4)
```

should produce the output:

```
ann_0_0
bob_0_0
clare_0_0
dave_0_0
```

The get_id_number() method of the Student class should simply return the ID number of a student. For example, the following code:

```
print('ID of s1 =', s1.get_id_number())
print('ID of s2 =', s2.get_id_number())
print('ID of s3 =', s3.get_id_number())
print('ID of s4 =', s4.get_id_number())
```

should produce the output:

```
ID of s1 = 1234
ID of s2 = 2345
ID of s3 = 3456
ID of s4 = 4567
```

The set_marks() method of the Student class takes two inputs - a "practical" mark and a "theory" mark. You can assume both of these inputs will always be a number between 0 and 100. This method should simply update the "practical" and "theory" marks for a student. For example, using the same four Student objects as in the previous example, this code:

```
s1.set_marks(60, 75)
s2.set_marks(30, 45)
s3.set_marks(45, 75)
s4.set_marks(80, 10)

print(s1)
print(s2)
print(s3)
print(s4)
```

would produce the output:

```
ann_60_75
bob_30_45
clare_45_75
dave_80_10
```

Finally, the passes() method of the Student class returns either True or False depending on whether or not the student's "practical" and "theory" marks would earn them a passing grade. The passes() method takes one input, which is either True or False, and which indicates whether or not the course has a "practical pass requirement". If a course *does* have a practical pass requirement, then in order to pass the course a student must have a passing "practical" mark (i.e. 50 or more) as well as having a passing "theory" mark (i.e. 50 or more). The COMPSCI 105 course is an example of such a course. If a course *does not* have a practical pass requirement, then in order to pass the course the *average* of the "theory" and "practical" marks must be at least 50 (i.e. even if a student fails one component, as long as their mark in the other component is high enough they can still pass the course).

For example, using the four Student objects with the "practical" and "theory" marks as set in the previous example, the following code:

```
print('With a practical pass requirement:')
print('s1 passes?', s1.passes(True))
print('s2 passes?', s2.passes(True))
print('s3 passes?', s3.passes(True))
print('s4 passes?', s4.passes(True))

print('With no practical pass requirement:')
print('s1 passes?', s1.passes(False))
print('s2 passes?', s2.passes(False))
print('s3 passes?', s3.passes(False))
print('s4 passes?', s4.passes(False))
```

would produce the output:

```
With a practical pass requirement:
s1 passes? True
s2 passes? False
s3 passes? False
s4 passes? False
With no practical pass requirement:
s1 passes? True
s2 passes? False
s3 passes? True
s4 passes? False
```

The Course class

This class represents a course that can have a number of enrolled students. The initialiser method should take *two inputs* - the name of the course and a True/False value indicating whether or not the course has a practical pass requirement (the meaning of this was described in the previous section).

For example, a course on “Basket weaving” that does have a practical pass requirement could be created as follows:

```
basket_weaving = Course('Basket weaving', True)
```

The __str__() method of the Course class should return a formatted string that displays the name of the course as well as the number of enrolled students in parentheses. For example, using the Course object created above, the following code:

```
print(basket_weaving)
```

should produce the output:

```
Basket weaving (0)
```

Because there are no students currently enrolled. OK, let’s add some students. Consider the following 6 students:

```
s1 = Student('ann', '1234')
s2 = Student('bob', '2345')
s3 = Student('clare', '3456')
s4 = Student('dave', '4567')
s5 = Student('edna', '5678')
s6 = Student('frank', '6789')
```

Students can be enrolled into a course by calling the add_students() method of the Course class. This method takes one input, which is a list of Student objects to enrol. For example, the first three students (s1, s2 and s3) could be enrolled into the “Basket weaving” course by calling the add_students() method with the input list [s1, s2, s3]. In this case, the following code:

```
basket_weaving.add_students([s1, s2, s3])
print(basket_weaving)
```

would produce the output:

```
Basket weaving (3)
```

as there are now three students enrolled in the course.

It should not be possible for any given student to be enrolled in a course more than once. So you must ensure that any Student objects that are added to a Course object are not already enrolled.

For example, consider the Course object from the earlier example which currently has 3 students enrolled (the objects s1, s2 and s3). If the following code was then executed:

```
basket_weaving.add_students([s4, s5])
print(basket_weaving)

basket_weaving.add_students([s1, s2, s3, s4, s5, s6])
print(basket_weaving)
```

the output would be:

```
Basket weaving (5)
Basket weaving (6)
```

Notice that when the add_students() method was called for the second time above, all 6 students were in the list that was provided as input - however, only one student (s6) was actually enrolled into the course, as all of the other students were already enrolled.

The Course class should also support a method called class_list which returns a string that consists of the individual string representations of each student in the course. These strings should be separated by a single space. For example, using the Course object from above, the following code:

```
print(basket_weaving.class_list())
```

would produce the output:

```
ann_0_0 bob_0_0 clare_0_0 dave_0_0 edna_0_0 frank_0_0
```

The course class should also support a method called set_marks which is given three inputs - the ID number of the student whose marks will be updated, and the “practical” mark and the “theory” mark for the student. For example, the code below sets the marks of 5 of the 6 students:

```
basket_weaving.set_marks('1234', 80, 90)
basket_weaving.set_marks('2345', 10, 95)
basket_weaving.set_marks('3456', 50, 50)
basket_weaving.set_marks('4567', 48, 52)
basket_weaving.set_marks('6789', 100, 49)

print(basket_weaving.class_list())
```

the output of the code above would therefore be:

```
ann_80_90 bob_10_95 clare_50_50 dave_48_52 edna_0_0 frank_100_49
```

An important function of the Course class is to produce a list of the ID numbers of all passing students. The method passing_ids() should return a string containing these ID numbers, where each individual ID number is separated by a single space character.

Given the Course object, and the student marks as initialised in the example so far, the following code:

```
print(basket_weaving.passing_ids())
```

should produce the output:

```
1234 3456
```

This is because the “Basket weaving” course was initialised with a practical pass requirement, and only Ann and Clare have passed both the “practical” and “theory” components of the course.

It is possible to modify the practical pass requirement for a course, by calling the set_practical_requirement method. This method takes one input - either True or False - and this updates the practical pass requirement of the Course object. For example, the following code:

```
basket_weaving.set_practical_requirement(False)
print(basket_weaving.passing_ids())
```

would produce the output:

```
1234 2345 3456 4567 6789
```

because all students (except for Edna, whose marks were not updated) would pass the “Basket weaving” course in the absence of a practical pass requirement.

Question 3: The Area class (15 marks)

For this question you must define one class called Area. The initialiser for the Area class is passed one input, which is a JSON-encoded string.

The string should be encoding a JSON *array*, so it should have this general format:

```
[ item1, item2, item3, ... ]
```

where there is at least one item in the array.

Some of the items in the array *may* be JSON objects, in which case they will have this format:

```
{ "key1" : value1, "key2" : value2, "key3" : value3 }
```

In particular, some of these objects *may* have a field called "area":

```
{ "key1" : value1, "key2" : value2, "area" : value3 }
```

Your task for this question is to complete the Area class such that when an Area object is printed (i.e. when its __str__() method is called), the output is either a formatted error message or a string representing a single number. The number should be the sum of the "area" field values, calculated across all valid items in the JSON-encoded array that have an "area" field.

Regardless of the string passed to the initialiser when an Area object is created, your class should not generate any unhandled exceptions. When printing an Area object (i.e. print(a)) the only allowable outputs are the three possibilities listed below:

1) An error message stating: "Invalid JSON"
If the JSON-encoded string does not represent a valid JSON array, then the __str__() method should return the error message "Invalid JSON".

2) An error message stating: "No areas"
If there are no objects in the JSON array, or if none of the JSON objects in the array have an "area" field, the __str__() method should return the error message "No areas"

3) The actual value that is the sum of all 'area' fields in the objects of the JSON array
If there are JSON objects in the JSON array that do have an "area" field, the sum of these should be returned. In this case, any JSON objects in the JSON array that do not have an "area" field should be ignored in the calculation.

No other outputs or errors should be generated.

Some examples will make this clearer:

In the following, neither of the input strings are valid JSON arrays:

```
a = Area('hello world')
b = Area('[a,b,c')

print('a =', a)
print('b =', b)
```

so the output is:

```
a = Invalid JSON
b = Invalid JSON
```

In the following, although the input strings are valid JSON arrays, none of the items are JSON objects:

```
c = Area('[1, 2, 3]')
d = Area('["Hello", "World"]')

print('c =', c)
print('d =', d)
```

so the output is:

```
c = No areas
d = No areas
```

In the following, the input strings are valid JSON arrays, and at least one of the items is a valid JSON object containing an “area” field:

```
e = Area('[1, 2, 3, {"area": 123, "weight": 1000}]')
f = Area('["Hello", "World", {"area": 999}]')

print('e =', e)
print('f =', f)
```

so the output is:

```
e = 123
f = 999
```

In the following, the input strings are valid JSON arrays containing valid JSON objects, but none of the objects contains an “area” field:

```
g = Area('[{"height": 100}, {"size": 123, "weight": 1000}]')
h = Area('["Hello", "World", {"volume": 999}]')

print('g =', g)
print('h =', h)
```

so the output is:

```
g = No areas
h = No areas
```

In the following, the input strings are valid JSON arrays containing valid JSON objects, and the sum of the “area” fields is computed:

```
i = Area('[{"area": 100}, {"area": 123, "weight": 1000}]')
j = Area('["Hello", "World", {"area": 999}, {"area": 1}]')
k = Area('[{"area": -1000}, {"area": 999}, {"area": 1}]')

print('i =', i)
print('j =', j)
print('k =', k)
```

so the output is:

```
i = 223
j = 1000
k = 0
```

You can structure the Area class as you see fit, as long as it behaves as described above. Obviously, the json.loads() function is going to be helpful - note that the “import json” statement has already been provided to you in the Asst1.py source file, so you can make use of the json.loads() function in your Area class.

Your Area class should not generate any exceptions, regardless of the string input provided to the initialiser. Make sure you handle all exceptions.

IMPORTANT

Your own work

This is an individual assignment. You must write the code for your classes on your own. Under no circumstances should you copy source code from anyone else, or allow your work to be copied. Discussing ideas or algorithms in general is acceptable, but you must write all of the code you submit yourself.

This will be enforced, so please don't try to copy and make superficial changes to make your code "appear" different to the code you copied. It is simple to detect such copying and modifications, and all submissions will be checked.

Submitting your assignment

When you have finished, submit your source file Asst1.py to the Assignment Drop box:

https://adb.auckland.ac.nz/

Your submitted classes will be marked automatically, so make sure that you adhere to the formats exactly as described in this document.

Good luck and enjoy!