COMP 1012 Fall 2016 Assignment 3

Due Date: Friday, November 4, 2016, 11:59 PM

Material Covered

- functions
- lists and loops
- input
- `if` statements

Notes:

- When programming, follow the posted programming standards to avoid losing marks. A program called `CheckStandardsV2.py` is provided to enable you to check that you are following standards.
- You will hand in two script files for this assignment. Name your script files as follows: `<LastName><FirstName>A3Q1.py` and `<LastName><FirstName>A3Q2.py.` For example, `LiJaneA3Q1.py` is a valid name for a student named Jane Li. If you wish to add a version number to your file, you may add it to the end of the file name. For example, `SmithRobA3Q1V2.py` is a valid name for Rob Smith's script file.
- Submit output in similarly-named output files: e.g., `<LastName><FirstName>A3Q1Output.txt.` To generate this file, open a new empty tab in Spyder and save it under the name above. **Make sure you choose *Text files* from the *Save as type:* list before you save the file!** Then copy and paste the output of your program from the console window to this file, and save it.
- You must complete the checklist for a ***Blanket Honesty Declaration*** in UM Learn to have your assignment counted. This one honesty declaration applies to all assignments in COMP 1012.
- To submit the assignment follow the instructions on the course website carefully (link: http://www.cs.umanitoba.ca/~comp1012/Documents/Handin_UMLearn.pdf). You will upload both script file and output via a dropbox on the course website. There will be a period of several days before the due date when you can submit your assignment. ***Do not be late!*** If you try to submit your assignment within 48 hours ***after*** the deadline, it will be accepted, but you will receive a penalty (of 25% per day late).

Group Work

NOTE: This assignment allows you to work with others as a group.

If you do the assignment all on your own and hand it in, you can get full marks if you achieve about 70% on the assignment. There will be no bonuses for doing better than that.

If you do the work with a friend, then each of you must hand in your own copy of the assignment. They do not have to be the same, but they can be, except for the inner author identification. If the solution you hand in is completely correct, you can get full marks, and similarly for your friend.

If you do the work with two others, then each of you must hand in your own copy of the assignment. Even if the solution you hand in is completely correct, you can get at maximum about 3.5 out of 4 marks. If you are part of a group of four, then each can get a maximum of 3 out of 4 marks. If you are part of a group of six, then each can get a maximum of about 2.5 out of 4 marks. If you are part of a group of nine, then each can get a maximum of 2 out of 4 marks. The more people you work with, the smaller is your possible mark. In general, the maximum mark is $\min(4, 6/\sqrt{n})$ when n people work as a group.

If you work as part of a group containing James Dean, Gwen Worobec and Calvin Wong as well as yourself, you must insert lines like the following into your code's initial doc string to avoid a charge of academic dishonesty. The names deanj12, worobg and wongc23 are the UMnetIDs of your associates.

```
@with deanj12
@with worobg
@with wongc23
```

They in turn should name you and one another in their submitted assignments.

If you do the assignment by yourself, put this line into your initial doc string:

```
@with nobody
```

Question 1—Finding roots of polynomials

Description

There are a number of computational techniques for finding the roots of functions. *Newton's method* takes an initial estimate of a root, and repeatedly calculates a refined estimate using the function and its derivative. We will restrict our solution to finding the roots of polynomials, since their derivatives are easy to calculate.

Given an estimate x_0 for the root of a function $f(x)$, Newton's method calculates refined estimates of the root x_1, x_2, x_3, ... as follows:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until $f(x_n)$ is "close enough" to zero (to some desired precision).

Newton's method is efficient, but it has one drawback: there are certain situations where it fails to find a root from an estimate. If the estimate is too distant from an actual root, or if the root has multiplicity greater than 1, the refined estimates may not converge to a root. To prevent this, we count the number of refinements and give up when the count gets too large (that is, we fail to find a root).[1]

Calculating the derivative of a polynomial (in its standard form) is just a matter of taking the derivative of each monomial ax^n. The derivative of a monomial is found by multiplying the coefficient by its power to get the coefficient of the term in the derivative, and reducing the power by one. That is, if

$$f(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x^1 + c_0 x^0,$$

then the derivative of f is f', given by

$$f'^{(x)} = n c_n x^{n-1} + (n-1) c_{n-1} x^{n-2} + \cdots + 1 c_2 x^1 + c_1 x^0.$$

Note that the resulting polynomial has one fewer term (the constant term $c_0 x^0$ falls off).

[1] For comparison, there are some root-finding techniques like bisection that converge more slowly but are sure to find a root of a continuous function when one exists.

Sample Output

The sample output below shows the results of finding the roots of a single polynomial, given five different estimates.

```
Polynomial: 5040x**0 + 1602x**1 + 1127x**2 + -214x**3 + -75x**4 + 4x**5 + 1x**6
Derivative: 1602x**0 + 2254x**1 + -642x**2 + -300x**3 + 20x**4 + 6x**5
Starting at 100.00, no root found, last estimate was 6.90, giving value f(6.90)=-2.91038e-11
Starting at 10.00, no root found, last estimate was 6.90, giving value f(6.90)=-2.91038e-11
Starting at 0.00, no root found, last estimate was -2.66, giving value f(-2.66)=8.83976e+03
Starting at -10.00, root found at x = -8.11, giving value f(-8.11)=0.00000e+00

Starting at -100.00, root found at x = -8.11, giving value f(-8.11)=0.00000e+00
Programmed by Instructors
Date: Thu Oct 13 14:28:47 2016
End of Processing
```

What to do in detail

Polynomials will be stored **as a list of coefficients**. A polynomial of degree *n* requires a list with *n+1* values, where the first element is the coefficient of the x^0 term, the second element of the list is the coefficient of the x^1 term, and so on. Your solution will be written with a number of functions; they are detailed below.

- First, write the function `polyToString` and print its results using several different polynomials of different degrees.
- Next, write `findDerivative` and test it on those same polynomials (use `polyToString` to see the results).
- Then, write the function `evaluatePoly` and try several polynomials and several *x* values, and verify the results.
- Next, write the function `newtonsMethod` and try several polynomials and several estimates, checking for both success and failure.
- Finally, write `findRoots`.

Functions

Your solution must have the following functions; match both the names and the parameters. The functions are listed here in alphabetical order by function name; your script must also include them in the same order, to make it easier for the markers to find your functions.

- `evaluatePoly(poly, x_):` Evaluate the polynomial `poly` at *x* = `x_` and return the result as a floating-point number using Horner's rule[2].
- `findDerivative(poly):` Find the derivative of the polynomial `poly` and return it as a list of coefficients (that is, create and return a new polynomial stored in a list; the list will be one item shorter than the original).

[2] See http://mathworld.wolfram.com/HornersRule.html.

- `findRoots(poly, estimates):` Find roots of the polynomial `poly` starting with each of the estimates in the list `estimates`. Use a value of epsilon of 1e-20 and 1000 iterations. The function will print both the polynomial and its derivative neatly using `polyToString`. For each starting estimate in the , call `newtonsMethod` and display either the root, or a message indicating that no root was found starting from that estimate. The messages should have the following forms:
  - `“Starting at 8.00, no root found, last estimate was 8.00, giving value f(8.00) = 6.86400e+04”`
  - `“Starting at -8.00, root found at x = -8.00, giving value f(-8.00) = 0.00000e+00”`
- `newtonsMethod(poly, x_, epsilon, timeout):` Apply Newton’s method to the polynomial `poly` using the initial estimate `x_`. Repeatedly revise the estimate (calculate x_{n+1} and use that as the new value for x). You will need to call `findDerivative` to find the derivative of the polynomial, and `evaluatePoly` to evaluate the polynomial and its derivative at the current estimate. Stop repeating when Newton’s method succeeds or fails. The method stops when either
  1. the absolute value of the polynomial at the current estimate is less than `epsilon`, in which case the method is successful.
  2. the absolute value of the derivative of the polynomial at the current estimate is less than `epsilon` (this could lead to division by 0), in which case the method failed, or
  3. the number of revisions of the estimate has exceeded `timeout` (the estimate is not converging on a solution). This case is also a failure.

  In all three cases, return a 2-tuple containing **both** the current estimate **and** a boolean value indicating if the method succeeded.
- `polyToString(poly):` Return a string containing the polynomial `poly` in standard form. For example, the poly [-1 2 -3 4 -5] would be returned as `"-5x^4 + 4x^3 – 3x^2 + 2x^1 - 1x^0"` (if you want to, you can omit the `"^1"` and the `"x^0"`).
  Note: this description does NOT match the sample output above. You may choose to either match the sample output, or return a string in the standard form shown here.
- `showTermination():` This function prints out the three lines of terminating output required in each program run. At the end of your script, call `showTermination` instead of printing these lines directly.

Handin

You will hand in your program script file. Also hand in the output produced by the following main program:

```
~~findRoots([ -5040, 1602, 1127, -214, -72, 4, 1 ], [-100] + range(-10, 11) + [100])~~
findRoots([ 5040, 1602, 1127, -214, -75, 4, 1 ], [-100] + list(range(-10, 11)) +
[100])
findRoots([ 1, 2, -3 ], [ 0, 1 ])
findRoots([ -1, 0, 0, 0, 0, 2 ], [1, 0.5, 0])
findRoots([ -0.03125, 0.3125, -1.25, 2.5, -2.5, 1 ], [1, 0.5, 0])
findRoots([ -30240, 0, 302400, 0, -403200, 0, 161280, 0, -23040, 0, 1024 ],
          [0, 0.5, 1, 1.5, 1.75, 2, 2.5, 3, 3.5])
showTermination()
```

Question 2—Ten Pin Bowling

Scoring (ten pin) bowling is a complex affair. In each frame, ten pins are set up and the bowler has two throws to knock down as many pins as possible. The three possible outcomes of a frame are:

1. open frame: pins are left standing at the end of the frame.
2. spare: all ten pins are knocked down using two throws.
3. strike: all ten pins are knocked down using one throw.

Each game consists of ten frames.

For each frame, the bowler scores one point for each pin knocked down, but is awarded bonus points in the case of a strike or spare. In a strike, the bowler is given ten points plus the number of pins knocked down in the next two throws. For a spare, the number of pins knocked down in the next throw is given as bonus points, in addition to the ten points earned for the spare. Strikes are denoted by X and spares by /.

So if the sequence of throws in the first three frames was

```
X3/90
```

then the score at the end of the three frames would be

- frame 1 (strike, one throw): 10 points (all ten pins knocked over) + 3 + 7 (bonus for next two throws)
- frame 2 (spare, two throws): 10 points (all ten pins knocked over) + 9 (bonus for next throw)
- frame 3 (open, two throws): 9 points
- total: 48 points

On the last (tenth) frame, a strike or spare may yield extra throws: if the bowler gets a strike on the last frame, they throw two extra balls for bonus points. If the bowler gets a spare on the tenth frame, one extra ball is thrown. No bonus points are given for the extra balls, so if one of the extra balls is a strike, no additional throws are given. In this way, a game of bowling will always (thankfully) end. Note that the extra balls count towards the bonus only, and are not themselves added to the score. So if a bowler has two extra throws, and scores 8 points on those two throws, their final score is increased by 8 for the bonus of frame ten, but not 16 (bonus on frame ten + score on bonus throws).

Write a program that accepts a sequence of frame scores and gives the total score. Your program should have at least one function that accepts a string and returns the score for the game. You should also have a function that validates a game: it should return a boolean value, and check whether the string parameter contains only digits and the characters 'X' and '/'.

Make sure your functions have appropriate names. Again, list them in alphabetical order, and copy and paste your `showTermination()` function from the first question.

Finally, your main program should accept user input of game scores, and either report that the game scores are invalid or the score from the game. The program should end when the user types the string "QUIT".

Handin

Submit your output for the program on the following inputs:

```
9/X12XX9/18333333
9/X12XX9/1833333/X
9/X12XX9/183333X72
XXXXXXXXXXXX
XXXXXXXXXX00
```

```
~~XXXXXXXXX93~~  XXXXXXXXX9/3
XXXXXXXXXXY
QUIT
```