Assignment 2: due 8am on Mon, Oct 16, 2017

Summary of Instructions

Note	**Read the instructions carefully and follow them exactly**
Assignment Weight	**As outlined in the syllabus, this assignment is worth 5% of your final grade**
Due Date	**This assignment is due 8am on Monday, Oct 16, 2017**
Important	**As outlined in the syllabus, late submissions will not be accepted**
	Any files with syntax errors automatically be excluded from grading. Be sure to test your code before you submit it
	For all functions, make sure you've written good docstrings that include type contract, function description and the preconditions if any. That includes functions you write both in Part 1 and Part 2.

Read the instructions below carefully. The instructions must be followed. This assignment is worth 5% of your grade. The assignment is due on Monday 16th of October 8AM. No late assignments will be accepted. This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages $15-19$ of course outline. You can find that file on Brightspace under Course Information. While at it, also review Course Policies on pages 13 and 14.

The goal of this assignment is to learn and practice the concepts covered thus far: function design, function calls, branching (i.e. if statements), strings, for-loops. You can make multiple submissions, but only the last submission before the deadline will be graded. What needs to be submitted is explained next. You have seen all the tools/concepts needed to solve this assignment, except possibly Question 9 and 10 in Part 2, where looping over indices of a string is helpful.

The assignment has two parts (in fact 3, if you count the bouns). Each part explains what needs to be submitted. Put all those required documents into a folder called `a2_xxxxxx` where you changed `xxxxxx` to your student number, zip that folder and submit it as explained in Lab 1. In particular, the folder should have the following files:
Part 1: `a2_part1_xxxxxx.py`, `a2_part1_xxxxxx.txt`
Part 2: `a2_part2_xxxxxx.py` and `a2_part2_xxxxxx.txt`
Bonus: and if you do bonus, you need to submitted the three images: `a2_week2_xxxxxx.jpg`, `a2_week3_xxxxxx.jpg` and `a2_week4_xxxxxx.jpg`.

Both of your programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file `a2_part1_xxxxxx.py` with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for Part 1 becomes zero. The same applies to Part 2, when they open and run file `a2_part2_xxxxxx.py`.

Furthermore, for each of the functions (in Part 1 and Part 2), I have provided one or more tests to test your functions with. For example, you should test function `high_school_eqsolver` from Part 1 by making a call in Python shell with `high_school_eqsolver(3,2,15)` To obtain a partial mark your function may not necessarily give the correct answer on these tests. But if your function gives any kind of python error when run on the tests provided below, that question will be marked with zero points.

Section 4, contains tests for Part 1. Tests for Part 2 are provided after each question in Part 2. To determine your grade, your functions will be tested both with examples provided in Part 2 and Section 4 and with some other examples. Thus you too should test your functions with more example than what I provided in Part 2 and Section 4.

Each function has to be documented with docstrings.

Global variables are not allowed. If you do not know what that means, for now, interpret this to mean that inside of your functions you can only use variables that are created in that function. For example, this is not allowed, since variable `x` is not a parameter of function `a_times(a)` nor is it a variable created in function `a_times(a)`. It is a global variable created outside of all functions.

```
def a_times(a):
    result=x*a
    return result

x=float(input("Give me a number: "))
print(a_times(10))
```

1 Part 1: Math quiz-generator / equation-solver - 40 points

You would like to help in your community center teach primary and high school students math. To reach as many kids as possible, you decided to write some software that can automate some tasks for you. As starting point, you decided to make an automatic quiz generator and grader for primary school pupils and a quadratic equation solver for highschool pupils – but unlike the solver you did in your lab, this quadratic equation solver needs to handle all possible cases including those when the given quadratic equation has complex numbers for solutions. In case of primary school pupils, your understanding is that they already know well addition and multiplication but that they struggle with subtraction and exponentiation.

For this part, I provided you with starter code in file called `part1_xxxxxx.py`. Begin by replacing `xxxxxx` in file name with your student number. Then open the file. Your solution (code) for this part must go into that file in clearly indicated spaces. You are not allowed to delete or comment-out any parts of the provided code except for the keywords `pass`.

For this part you need to submit two files: `a2_part1_xxxxxx.py` and `a2_part1_xxxxxx.txt`

`a2_part1_xxxxxx.py` needs to contain your program for Part 1 as explained above and `a2_part1_xxxxxx.txt` needs to contain the proof that you tested your two core functions from this part, namely `primary_school_quiz` and `high_school_eqsolver`

1.1 The Core Functions

Your solution in `a2_part1_xxxxxx.py` must have two functions called: `primary_school_quiz` and `high_school_eqsolver`.

You should design and test these functions first before moving onto the `main` part of the program. Here are specifications for the two functions:

`primary_school_quiz`: This function takes two parameters, namely an integer `flag` and a positive integer `n`. If `flag` is `0`, `primary_school_quiz` helps practice subtraction. But if `flag` is `1`, `primary_school_quiz` helps practice exponentiation. The function, `primary_school_quiz` then generates `n` math problems that a pupil must answer in turn. For each question, it generates two random positive, single-digit numbers (check out python's `random` module to see if there's a useful function in there) and asks the pupil for the answer to the math problem with those two numbers (either subtract the second number from the first, or raise the first number to the power of the second number). `primary_school_quiz` then prompts the pupil for the answer, and checks if her answer is correct. At the end of `n` questions, `primary_school_quiz` returns the number of questions answered correctly.

`high_school_eqsolver`: This function has three parameters representing three real numbers for the coefficients of the quadratic equation $ax^2 + bx + c = 0$. The function displays/prints the equation frist and then prints its solutions. The function must display correct and meaningful solutions given any three real numbers for coefficients a, b and c. See examples in Section 4 to understand what that means. Please consider the examples (and the video) to be a part of the function/program specifications that must be followed.

Note that to solve this problem, you do not need to use Python's complex numbers.

1.2 The User Interaction i.e. the `main` part of the program

Now that you have the two functions that perform the core functionality, you want to make it more user friendly for the pupils (after all, alas, the pupil may not know how to write code and call functions in Python shell). In the `main` part of your program, write your code in specified places. You code must follow the behaviour indicated in the example runs in Section 4 (and the video to be provided).

For example, for primary school pupils, called Ena, you will first ask her whether she would like to practice subtraction or exponentiation. Then ask her how many practice questions she'd like (if she says `0`, then your code should not ask her to solve any math questions). Using her responses, call the `primary_school_quiz` function with the appropriate values. When it returns the number of correct answers, display a message to the pupil:

- If she did 90% or better, display on screen: `Congratulations Ena! You'll probably get an A tomorrow. Now go eat your dinner and go to sleep. Good bye Ena!`
- If she did 70% or better, but worse than 90%, display on screen: `You did well Ena, but I know you can do better. Good bye Ena!`
- If she did worse than 70%, display on screen: `I think you need some more practice Ena. Good bye Ena!`

For high school pupils, in the `main` part of your program in the specified places, you need to write some code that asks the pupil for the coefficients a, b and c. Then you need to make the call to `high_school_eqsolver` to display the solutions. After that your program should ask the pupil if they would like another quadratic equation solved. If the pupil says anything but `yes` the program terminates by printing a good bye message as in the examples. Otherwise

(and any form of typing `yes` should be acceptable, including with lots of white space before and after, and with capital letters and lower case letter etc), as long as pupil answers `yes`, she should be asked for the coefficients again and the resulting new quadratic equation should be solved. Since you have not seen while loops yet, I provided the code for that in `part2_xxxxxx.py`.

The rest of the specifications for your program in Part 1 can be inferred from examples and the video. You will notice that the program is required to display greetings surrounded with stars. One of you functions from Assignment 1, may be helpful for that. You can copy paste that function from your Assignment 1 solution (or mine) to your solution in `a2_part1_xxxxxx.py`.

2 Part 2: A Library of Functions

For this part of the assignment, you are required to write and test several functions (as you did in Assignment 1). You need to save all functions in `part2_xxxxxx.py` where you replace xxxxxx by your student number. You need to test your functions (like you did in Assignment 1) and copy/paste your tests in `part2_xxxxxx.txt`. Thus, for this part you need to submit two files: `a2_part2_xxxxxx.py` and `a2_part2_xxxxxx.txt`

2.1 `min_enclosing_rectangle(radius, x, y)` - 5 points

Computing a smallest (axis-aligned) rectangle that encloses a set of objects in the plane is a very common computational problem arising in graphics and thus game development too. Write a function, called `min_enclosing_rectangle`, that has 3 input parameters. The first is a number representing a radius of a circle and the next two are two numbers representing the x- and y-coordinates of its center. Consider the smallest axis-aligned rectangle that contains that circle. The function should return the x- and y-coordinates of the bottom-left corner of that rectangle. If `side` is a negative number, `min_enclosing_rectangle` should return `None`.

```
>>> min_enclosing_rectangle(1,1,1)
(0, 0)
>>> min_enclosing_rectangle(4.5, 10, 2)
(5.5, -2.5)
>>> min_enclosing_rectangle(-1, 10, 2)
>>> min_enclosing_rectangle(500, 1000, 2000)
(500, 1500)
```

2.2 `series_sum()` - 5 points

Write a function called `series_sum()` that prompts the user for an non-negative integer n. If the user enters a negative integer the function should return `None` otherwise the function should return the sumer of the following series $1000 + 1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 + ... + 1/n^2$ for the given integer n. For example, for $n = 0$, the function should return 1000, for $n = 1$, the function should return 1001, for $n = 2$, the function should return 1001.25, for $n = 3$, the function should return 1001.3611111111111, etc.

```
>>> series_sum()
Please enter a non-negative integer: -10
>>> series_sum()
Please enter a non-negative integer: 0
1000
>>> series_sum()
Please enter a non-negative integer: 5
1001.463611111111
```

2.3 `pell(n)` - 5 points

Pell numbers are a mathematical sequence of numbers that help approximate the value of $\sqrt{2}$ (check out the Wikipedia page on Pell Numbers:
https://en.wikipedia.org/wiki/Pell_number). The sequence is defined recursively as shown in eq. 1.

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 2P_{n-1} + P_{n-2} & \text{if } n > 1 \end{cases} \quad (1)$$

Write a function called `pell` that takes one integer parameter `n`, of type `int`. If `n` is negative, `pell` returns `None`. Else, `pell` returns the nth Pell number (i.e. P_n).

```
>>> pell(0)
0
>>> pell(1)
1
>>> pell(2)
2
>>> pell(3)
5
>>> pell(-45)
>>> pell(6)
70
```

2.4 countMembers(s) - 5 points

In this question you are not allowed to use any of the Python's string methods (i.e. those string functions that you call with dot operator).

Say a character is *extraordinary* if it is one of the following: the lower case letter between `e` and `j` (inclusive), the upper case letters between `F` and `X` (inclusive), numerals between `2` and `6` (inclusive), and the exclamation point (`!`), comma (`,`), and backslash (\)

You are required to count how many times these characters appear in a string.

Write a function called `countMembers` that takes a single input parameter `s`, of type `str`. `countMembers` then returns the number of characters in `s`, that are extraordinary. Therefore, if there are two `X`s in `s`, `countMembers` must count two extraordinary characters (one for each occurrence of `X` in `s`).

```
>>> countMembers("\\")
1
>>> countMembers("2\'")
1
>>> countMembers("1\'")
0
>>> countMembers("2aAb3?eE'_13")
4
>>> countMembers("one, Two")
3
```

2.5 casual_number(s) - 5 points

Imagine a customer in a bank is asked to enter a whole number representing their approximate monetary worth. In that case, some people use commas to separate thousands, millions etc, and some don't. In order to perform a some calculations the bank needs that number as na integer. Write a function, called `casual_number(s)` that has one parameter, `s`, as input where `s` is a string. Function `casual_number(s)` should return an integer representing a number in `s`. If `s` does not look like a number the function should return `None`. Note that if a string in `s` looks like a number but with commas, you may assume that commas are in meaningful places i.e. you may assume that `s` will not be a string like '$1,1,345$'.

```
>>> casual_number("251")
251
>>> casual_number("1,aba,251")
>>> casual_number("1,251")
1251
>>>casual_number("1251")
1251
>>> casual_number("-97,251")
-97251
>>> casual_number("-97251")
-97251
>>> casual_number("-,,,,")
>>> casual_number("--97,251")
>>> casual_number("-")
>>> casual_number("-1,000,001")
-1000001
>>> casual_number("999,999,999,876")
999999999876
>>> casual_number("-2")
-2
```

2.6 `alienNumbers(s)` - 5 points

Strange communication has been intercepted between two alien species at war. NASAs top linguists have determined that these aliens use a weird numbering system. They have symbols for various numeric values, and they just add all the values for the symbols in a given numeral to calculate the number. NASA's linguists have given you the following table of symbols and their numeric values. Since they have a lot of these numbers to compute, they want you to write a function that they can use to automate this task.

Symbol	Value
`T`	1024
`y`	598
`!`	121
`a`	42
`N`	6
`U`	1

Thus `a!ya!U!NaU` represents a value of `1095` (see table below for an explanation).

Numeral	Value	Occurrences	Total Value	Running Total
`T`	1024	0	$0 \times 1024 = 0$	0
`y`	598	1	$1 \times 598 = 598$	598
`!`	121	3	$3 \times 121 = 363$	961
`a`	42	3	$3 \times 42 = 126$	1087
`N`	6	1	$1 \times 6 = 6$	1093
`U`	1	2	$2 \times 1 = 2$	1095

Write a function called `alienNumbers` that takes one string parameter `s`, and returns the integer value represented by `s`. Since aliens only know these characters you may assume that no character in `s` outside of this set: `{T,y, !,a, N, U}`. *Callenge:* try to make the whole body of this function only one line long.

```
>>> alienNumbers("a!ya!U!NaU")
1095
>>> alienNumbers("aaaUUU")
129
>>> alienNumbers("")
0
```

2.7 `alienNumbersAgain(s)` - 5 points

NASA is very pleased with your proof-of-concept solution in the previous question. Now, they've designed a small chip that runs a very restricted version of python - it doesn't have any of the string methods that you are used to. They want you to now rewrite your `alienNumbers` function to run without using any of those string methods you may have otherwise used. Basically, you can use a loop of some sort and any branching you'd like, but none of the string methods. Use *accumulator variable* as we have seen in class.'

Write a function called `alienNumbersAgain`, that takes a single string parameter `s`, and returns the numeric value of the number that `s` represents in the alien numbering system.

```
>>> alienNumbersAgain("a!ya!U!NaU")
1095
>>> alienNumbersAgain("aaaUUU")
129
>>> alienNumbersAgain("")
0
```

The last three questions may be a more challenging to solve especially the last one.

2.8 `encrypt(s)` - 5 points

Hint: Think of how slicing and reversing of strings can help with this problem. (We saw how to do slicing and reversing at the beginning of Lecture 8)

You want to send a note to your friend in class, and because you want to be respectful in class, you don't want to whip out your phone and send a text or an email (or any other digital communication). Instead, you choose to go old school and write it out on a piece of paper and pass it along to your friend. The problem is, you don't want anyone else to read the note as they pass it along. Luckily, your friend and you have come up with an encryption system so that nobody else can understand your message. Here's how it works: you write out your message backwards (so, `Hello,`

`world` becomes `dlrow ,olleH`). But you don't stop there, because that's too easy to crack - anyone can figure that out! Now that you've written in backwards, Then you start on either side of the string and bring the characters together. So the first and the last characters become the first and the second character in the encrypted string, and the second and the second last characters become the third and the fourth characters in the string, and so on. Thus, `Hello, world` ultimately becomes `dHlerlolwo ,` (notice how all punctuation, special characters, spaces, etc are all treated the same) and `0123456789` becomes `9081726354`.

Write a function called `encrypt`, that has one parameter `s` where `s` is a string and `encrypt` returns a string which is the encrypted version of `s`.

Fun fact in cryptography, `s` is called "clear text" and the encrypted version you return is called "cipher text"

```
>>> encrypt("Hello, world")
'dHlerlolwo ,'
>>>> encrypt("1234")
'4132'
>>> encrypt("12345")
'51423'
>>> encrypt("1")
'1'
>>> encrypt("123")
'312'
>>> encrypt("12")
'21'
>>> encrypt("Secret Message")
'eSgeacsrseetM '
>>> encrypt(",'4'r")
"r,''4"
```

2.9 `oPify(s)` - 5 points

Hint: Depending on how you plan to solve this problem, accumulator variable initialized as an empty string may help in this question.

Write a function called `oPify`, that takes a single string parameter (`s`) and returns a string. This function considers every pair of consecutive characters in `s`. It returns a string with the letters `o` and `p` inserted between every pair of consecutive characters of `s`, as follows. If the first character in the pair is uppercase, it inserts an uppercase `O`. If however, it is lowercase, it inserts the lowercase `o`. If the second character is uppercase, it inserts an uppercase `P`. If however, it is lowercase, it inserts the lowercase `p`. If at least one of the character is not a letter in the alphabet, it does not insert anything between that pair. Finally, if `s` has one or less characters, the function returns the same string as `s`.

Do `dir(str)` and check out methods `isalpha` (by typing `help(str.isalpha)` in Python shell), and `isupper`

```
>>> oPify("aa")
'aopa'
>>> oPify("aB")
'aoPB'
>>> oPify("ooo")
'oopoopo'
>>> oPify("ax1")
'aopx1'
>>> oPify("abcdef22")
'aopbopcopdopeopf22'
>>> oPify("abcdef22x")
'aopbopcopdopeopf22x'
>>> oPify("aBCdef22x")
'aoPBOPCOpdopeopf22x'
>>> oPify("x")
'x'
>>> oPify("123456")
'123456'
```

2.10 `nonrepetitive(s)` - 5 points

This may be quite a challenging question to solve. Slicing and remembering that you can ask if two strings `(s1==s2)` are the same are key to a short solution to this problem.

A *nonrepetitive* word is a word that does not contain any subword twice in a row. Examples:
ana is nonrepetitive.
borborygmus is not nonrepetitive, since it has subword, bor twice in a row.
abracadabra is nonrepetitive.
repetitive is not nonrepetitive since subword ti is repeated twice in a row.
grammar is not nonrepetitive since subword m is repeated twice in a row.
gaga is not s nonrepetitive since subword ga is repeated twice in a row.
rambunctious is nonrepetitive.
abcab is nonrepetitive.
abacaba is nonrepetitive.
zrtzghtghtghtq is not nonrepetitive since subword ght is repeated twice (in fact three times, but it is enough to find two repetitions to conclude that the word is not nonrepetitive).
aa is not nonrepetitive since subword a is repeated twice.
zatabracabrac is not nonrepetitive since subword abrac is repeated twice in a row.

Write a function, called `nonrepetitive`, that has one parameter, `s`, where `s` is a string. The function returns True if `s` is nonrepetitive and False otherwise.

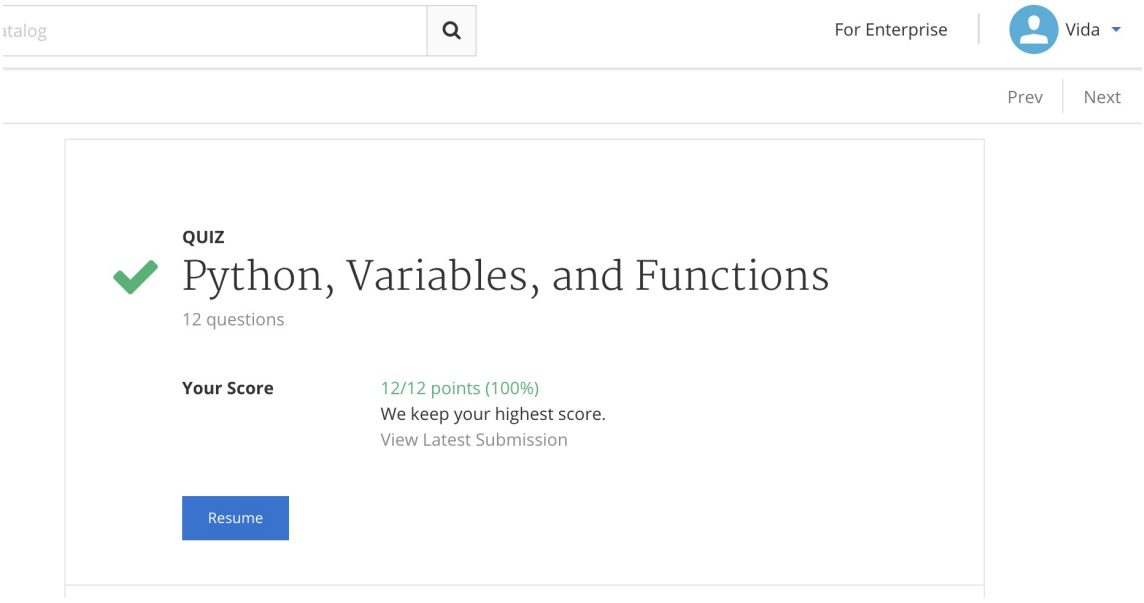
```
>>> nonrepetitive("")
True
>>> nonrepetitive("a")
True
>>> nonrepetitive("zrtzghtghtghtq")
False
>>> nonrepetitive("abcab")
True
>>> nonrepetitive("12341341")
False
>>> nonrepetitive("44")
False
```

3 Bonus - 9 bonus points

To get bonus go to `https://www.coursera.org/learn/learn-to-program` and complete the quizzes of Week 2, Week 3 and Week 4 (like you did Week 1 in Lab 2). We will need some small proof that you did that and I will explain below how to do that.

Think of this part as practice for your quiz and exams.

If you have difficulties getting a good grade in any of the three quizzes (Week 2, Week 3, and/or Week 4), I suggest strongly studying more (by reviewing lectures, labs, reading the textbook). I also suggest you watch all the short coursera videos that relate to the given week.



Complete Week 2, Week 3, and Week 4 quizzes. You can do each quiz many times (3 times every 8 hours – it seems). When you are satisfied with you quiz grade take a photo (screen capture) of your results with your name visible in top right corner. See for example my screen capture on the pervious page. Save the three images as `a2_week2_xxxxxx.jpg`, `a2_week3_xxxxxx.jpg` and `a2_week4_xxxxxx.jpg` and submit them with your assignment. Consider each quiz as being worth 3 points. For each of the three quizzes that you obtain 75% or more for, you will get 3 bonus points. Quizzes with grade less than 75% do not get bonus.

Make sure the following is visible in each of your images:
1. that your name on the top right corner
2. that the title of the quiz is visible
3. that the score for the last attempt is visible for that week

Important: Each image cannot be more than 200 KB (KB not MB!! we have limited space on the Brightspace server). For me the screen capture produced an jpg image of about 104KB.

FYI. the titles of quizzes are:
Week 2: Strings and Designing Functions
Week 3: Booleans, Import, Namespaces, and if Statements
Week 4: For Loops and Fancy String Manipulation

4 Testing your code in Part 1

Here is how you should test your two functions from Part 1 in Python shell.

```
>>> primary_school_quiz(0,2)
Question 1:
What is the result of 3-9? -6
Question 2:
What is the result of 9-7? 2
2
>>>
>>>
>>> primary_school_quiz(1,3)
Question 1:
What is the result of 1^4? 1
Question 2:
What is the result of 9^9? 50
Question 3:
What is the result of 4^7? 50
1
>>>
>>>
>>> high_school_eqsolver(1,3,1)
The quadratic equation 1·x^2 + 3·x + 1 = 0
has the following real roots:

-0.3819660112501051  and -2.618033988749895
>>>
>>>
>>> high_school_eqsolver(3,2,15)
The quadratic equation 3·x^2 + 2·x + 15 = 0
has the following two complex roots:
-0.3333333333333333 + i 2.2110831935702664
 and
-0.3333333333333333 - i 2.2110831935702664
>>>
>>>
>>> high_school_eqsolver(3,2,-15)
The quadratic equation 3·x^2 + 2·x + -15 = 0
has the following real roots:
1.9274433277084226  and -2.5941099943750894
>>>
>>>
>>> high_school_eqsolver(1,2,1)
The quadratic equation 1·x^2 + 2·x + 1 = 0
has only one solution, a real root:
-1.0
>>>
>>>
```

```
>>> high_school_eqsolver(0,2,4)
The linear equation 2·x + 4 = 0
has the following root/solution: -2.0
>>>
>>>
>>> high_school_eqsolver(0,0,0)
The quadratic equation 0·x + 0 = 0
is satisfied for all numbers x
>>>
>>>
>>> high_school_eqsolver(0,0,10)
The quadratic equation 0·x + 10 = 0
is satisfied for no number x
```

Here is what pressing Run on your program (Part 1) should look like:

```
>>>
************************************************************
*                                                          *
*  __Welcome to my math quiz-generator / equation-solver__  *
*                                                          *
************************************************************

What is your name? Vida
Hi Vida. Are you in? Enter
1 for primary school
2 for high school or
3 for none of the above?
1

************************************************************************
*                                                                      *
*  __Vida, welcome to my quiz-generator for primary school students.__  *
*                                                                      *
************************************************************************

Vida what would you like to practice? Enter
0 for subtraction
1 for exponentiation
0
How many practice questions would you like to do? 4
Vida, here is your 4 questions:
Question 1:
What is the result of 1-7? -6
Question 2:
What is the result of 1-7? -6
Question 3:

What is the result of 3-10? -7
Question 4:
What is the result of 8-6? 100
You did well Vida, but I know you can do better.
Good bye Vida!
```

Another example run:

```
************************************************************
*                                                          *
*  __Welcome to my math quiz-generator / equation-solver__  *
*                                                          *
************************************************************

What is your name? Vida Dujmovic
Hi Vida Dujmovic. Are you in? Enter
1 for primary school
2 for high school or
3 for none of the above?
2

************************************************************************
*                                                                      *
*  __quadratic equation, a·x^2 + b·x + c= 0, solver for Vida Dujmovic__  *
*                                                                      *
************************************************************************
```

```
Vida Dujmovic, would you like a quadratic equation solved?    YeS
Good choice!
Enter a number the coefficient a: 1
Enter a number the coefficient b: 2
Enter a number the coefficient c: 3
The quadratic equation
1.0·x^2 + 2.0·x + 3.0 = 0
has the following two complex roots:
-1.0 + i 1.4142135623730951
 and
-1.0 - i 1.4142135623730951
Vida Dujmovic, would you like a quadratic equation solved?        yeS
Good choice!
Enter a number the coefficient a: -1
Enter a number the coefficient b: 10
Enter a number the coefficient c: 36
The quadratic equation -1.0·x^2 + 10.0·x + 36.0 = 0
has the following real roots:
-2.810249675906654  and 12.810249675906654
Vida Dujmovic, would you like a quadratic equation solved?   no
Good bye Vida Dujmovic!
```

Another example run:

```
>>>
************************************************************
*                                                          *
*  __Welcome to my math quiz-generator / equation-solver__  *
*                                                          *
************************************************************

What is your name? Arya Stark
Hi Arya Stark. Are you in? Enter
1 for primary school
2 for high school or
3 for none of the above?
3
Arya Stark you are not a target audience for this software.
Good bye Arya Stark!
```