Lab #1

Due: 9/26/22 @ 11:59pm

Content Covered

Expressions, Statements, Variables, Functions, and Selection Statements in Python

Getting Started

1. Sign into replit.com
2. Create a new python REPL, name it whatever you like

Editing Your Code

In this lab you will be defining functions to solve 6 problems as described below. All of your function definitions should be written in main.py. It is important that you name your functions exactly as specified by each problem so that the autograder will be able to call them.

Any variables and parameters can be named however you like. You can write any additional functions to help you solve the problems as you see fit. Each problem is more complex than the ones before it.

Submission

The Autolab submission for this lab will open up no later than 9/19/22 @ 5:00pm.

Your final submission for this lab should be a .zip file download from replit.com and submitted to Autolab. You may submit as many times as you like, but only your last submission will count for your final grade.

Submissions close at 11:59pm on 9/26/22 and no late submissions will be accepted.

Problems

Problem 1

Define a function named `double` which, when given a string as an argument, returns the string concatenated with a space and then concatenated with itself.

For example: `double("hello")` must return `"hello hello"`.

Problem 2

Define a function named `longest` which accepts three string arguments and returns the length of the longest one.

For example: `longest("hello","hi","museum")` must return `6`.

Hint: you might find the built-in functions `len` and `max` to be helpful.

Problem 3

Define a function named `median` which accepts three numeric arguments and returns the median of the three (i.e. the middle one).

For example: `median(7,9,4)` must return `7`.

Problem 4

Define a function named `grade` which maps a numeric grade to a letter grade, based on the table below. You may assume that the argument to the function is a float value greater than or equal to zero and less than or equal to 100.

Numeric grade	**Letter grade**
90 <= grade <= 100	A
80 <= grade < 90	B
70 <= grade < 80	C
60 <= grade < 70	D
0 <= grade < 60	F

For example: `grade(80)` must return `"B"`.

Problem 5

Define a function named `repeat` which functions like the `double` function from Problem 1, except it now also accepts an additional argument which specifies how many times the string should be repeated. Assume that the numeric argument is a non-negative integer.

For example: `repeat("wow", 4)` must return `"wow wow wow wow"`

Hint: The `*` operator can be used on strings. For example, `"hi" * 3` evaluates to `"hihihi"`.

Problem 6

Define a function named `dice`, which takes three arguments. The first two arguments are integers, and the third argument is a boolean. The two integers represent the roll of two six-sided dice, and you can assume they will both be greater than or equal to 1, and less than or equal to 6. The function will return a string that describes the dice roll as follows:

The string that the function returns will start with the sum of the two dice rolled.

If the third argument is `True`, then the sum may be followed by a special description, based on what was rolled:

The dice rolled	**The special description**
Two ones	Snake Eyes!
Two sixes	Boxcars!
Any other doubles	Doubles!
Two dice that sum to 7	Lucky Number 7!

The sum and the special description should be separated by a colon and a space.

If the third argument is `False`, or the dice rolled do not have a special description, then just the sum of the dice as a string will be returned.

For example: `dice(4, 3, True)` must return `"7: Lucky Number 7!"`.

Hint: To turn an integer into a string, you may find the built-in function `str` helpful.

Useful Suggestions and Advice

- **Think before you code** - think about the input to the function, and the desired output, and plan out how you would solve the problem in your head or on paper before you start to code. Having an idea of where you are going before you start typing can make the coding process smoother.
- **Write your own tests** - each problem describes an example test case or two that you can use to check if your implementation is correct. But they are not sufficient to guarantee you have a completely correct implementation. It can help to think through and write additional test cases (even before you code) to make sure your functions are behaving as you expect them to. This will also allow you to more quickly test your code as you go, rather than having to wait for Autolab to be open, and to go through the entire submission process just to get feedback.
- **Break down bigger problems into smaller ones** - if you are stuck on a problem, try solving a smaller piece of the problem first. You can even write it as a separate function, and test it out. Then once you have the small pieces working, building the solution to the bigger problem from your smaller solutions should be easier.
- **Ask for help** - during lab sessions, in office hours, and on Piazza. If you do choose to ask questions on Piazza **do not post your own code in public posts.** If you are going to make your question public, make sure it is general/conceptual in nature. Otherwise, make your question private to just the instructors.