IAE 101 – Introduction to Digital Intelligence
Homework 01
Fall, 2019

Assigned: Saturday, 09/28/2019
Due: Sunday, 10/04/2019, at 11:55 PM

Assignment Objectives:

The goal of this assignment is to help you solve problems and implement algorithms in Python using the tools discussed in class.

Getting Started:

This assignment requires you to write Python code to solve computational problems. To help you get started on the assignment, we will give you a basic starter file for each problem. These files contain function stubs ('function stubs' are functions that have no bodies; the function has a name and a list of parameters, but it is up to you to complete the body of the function) and a few tests you can try out to see if your code seems to be correct (note that the test cases we give you in the starter files are just examples; we will use different test inputs to grade your work!). You need to complete (fill in the bodies of) these functions for the assignments. Do not, under any circumstance, change the names of the functions or their parameter lists.

Directions:

There are five problems. Each problem is worth 10 points. Solve each of the following problems to the best of your ability.

- Each starter file has a comment block at the top with various important information. Be sure to add your information (name, ID number, and NetID) to the first three comment lines, so that we can easily identify your work.
- Each of your functions must use the names and parameter lists indicated in the starter code file.
- Each of your functions must explicitly return its final answer; the grading program will ignore anything that your code prints out. Along those lines, do NOT use input() anywhere within your functions; your functions should get all their input from their parameters.
- Be sure to submit your final work as directed by the indicated due date and time. Late work will not be accepted for grading.
- Programs that crash will likely lose a lot of credit, so make sure you thoroughly test your work before submitting it.
- Submit all your completed problemX.py files on Blackboard for Programming Homework 02.

Problem 1: Testing for divisibility

Suppose I want to know whether a number, *x* is evenly divisible by a number *y*. When I use the phrase "evenly divisible", I mean division without any remainder. So, *x* is evenly divisible by *y* if we can divide *x* by *y* without leaving any remainder behind (the remainder is equal to 0).

Sometimes it is easy to tell whether one number evenly divides another just by looking. If *y* is 2, then we know that all even numbers (numbers ending with 0, 2, 4, 6, or 8) can be evenly divided by 2 and that all odd numbers (numbers ending with 1, 3, 5, 7, or 9) cannot be evenly divided by 2. The same is true if *y* is equal to 5. All numbers ending in 5 or 0 can be evenly divided by 5.

However, there are many cases of divisibility that are harder to check. Determining whether a number is evenly divisible by 3 or 7 can be difficult to do just by looking, especially for larger numbers.

Computer can always do this easily. Recall that the modulus operator, %, is used to return just the remainder from an integer division operation. The expression, *x % y*, which is read as *x* modulo *y* (or *x* mod *y*), integer divides *x* by *y* and returns just the remainder of the division operation. We can use this to easily determine whether *x* is evenly divided by *y*. If *x % y* evaluates to 0, then we can conclude that *x* is evenly divisible by *y*.

The file "hw2problem1.py" contains a Python function called `divisible()`, which takes one argument, an integer *n*. Complete the body of the function to do the following: First, create an empty list. Then for each of the integers 1 through 9, determine whether it evenly divides *n*. If so, then add True to the list. If not, then add False to the list. Once you have checked whether *n* is divisible by each of the integers from 1 through 9, return the list of Booleans.

For example, suppose the input *n* is equal to 126. 126 is evenly divisible by 1, 2, 3, 6, 7, and 9. So, I will add True at the first three elements, then two False entries for 4 and 5, then True for 6 and 7, False for 8, and True for 9. The result list can be seen in the table below. I then return the list.

Examples:

Function Call	Return Value
`divisible(126)`	`[True, True, True, False, False, True, True, False, True]`
`divisible(20)`	`[True, True, False, True, True, False, False, False, False]`
`divisible(1024)`	`[True, True, False, True, False, False, False, True, False]`
`divisible(17)`	`[True, False, False, False, False, False, False, False, False]`
`divisible(539`	`[True, False, False, False, False, False, True, False, False]`

Problem 2: Character Frequency

An important part of automated document analysis is measuring the frequency at which words and characters appear in the document. Search engines and other tools can use this to try to determine whether two documents are related to each other in a significant way (so that both are returned as search results, etc.)

We are going to build up to producing frequency analysis for words in documents, by beginning with frequency counts for characters in a string. The frequency with which a character appears is the count of the number of times the character appears in the string. The relative frequency is the frequency divided by the total number of letters in the string.

The file "hw2problem2.py" contains a function called `frequency()`, which takes two arguments: a single character string named *c*, and a target string named *s*. Complete the body of the function to compute the relative frequency with which character *c* appears in string *s* by doing the following:

1. First, count the number of times that *c* appears in *s*.
2. Second, compute the total number of characters in *s*.
3. Third, divide the result of the step 1 by the result of step 2.
4. Fourth, multiply the result of step 3 by 100.
5. Fifth, round the result to two decimal places (Study the built-in function `round()`).

Return the final value as the result of the function—the relative frequency with which *c* appears in *s*.

For example, if I call frequency("i", "Mississippi"). The letter "i" appears 4 times in the string "Mississippi". The total number of characters in "Mississippi" is 11.
$4 \div 11 = 0.36\overline{36}$.
$0.36\overline{36} \times 100 = 36.36\overline{36}$.
Rounded to two decimals the result is 36.36 %.

IMPORTANT: Do ***not*** use the built-in function count() in your solution.

Examples:

Function Call	Return Value
frequency("a", "supercalifragilisticexpialidocious")	8.82
frequency("i", "antidisestablishmentarianism")	17.86
frequency("i", "mississippi")	36.36

Problem 3: Reversing a String

String manipulation is an important part of record and document analysis. A simple string manipulation task is reversing a string—constructing a new string in which the letters appear in the opposite order.

The file, "hw2_problem3.py", contains two functions, reverse1() and reverse2(). Each function takes a single argument, *s*, which is a string. Both functions are going to construct the reverse of *s* and then return it. Complete the body of both functions in the following ways. In reverse1() use a loop to construct a reversed string. Then return the reversed string. In reverse2() use slicing to construct the reversed string, and then return the reversed string

Examples:

Function Call	Return Value

reverse1("Mississippi")
reverse2("Mississippi")
	"ippississiM"

reverse1("Connetquot")
reverse2("Connetquot")
	"touqtennoC"

reverse1("Wyandanch")
reverse2("Wyandanch")
	"hcnadnayW"

reverse1("Ronkonkoma")
reverse2("Ronkonkoma")
	"amoknoknoR"

Problem 4: How Far Did the Ball Go?

Suppose that your throw a ball off a rooftop. The distance that the ball will travel depends upon how far it can travel horizontally before it finally hits the ground. There is a simple equation that describes the distance that the ball will move after a given interval of time, given its initial position and velocity, and the forces acting upon the ball as it travels, which will determine how it accelerates in flight.

$$d = d_0 + (v_0 * t) + \frac{1}{2}(a * t^2)$$

Where d is the total distance; d_0, is the initial position of the ball; v_0, is the initial velocity of the ball; a, is the acceleration acting on the ball; and, t, is the time. All times are in seconds; all distances are in meters; velocity is meters per second; and, acceleration is meters per second per second.

So, if you want to know how far the ball will travel you need to use this equation twice. In the first case you will use it to determine how long the ball travels before it strikes the ground. Once you know the duration of the ball's flight, then you can use that to determine how far the ball travels horizontally.

We are making at least two simplifying assumptions. First, all the force of the throw is directed horizontally, so the force of the throw does not affect the rate at which the ball falls. Second, we are also ignoring the effects of air resistance on the movement of the ball in both the horizontal and vertical directions.

The file, "hw2problem4.py" contains a function, `how_far()`, which takes two float arguments: the initial horizontal velocity, *v*, when the ball leaves the hand, and the height, *h*, from which the ball is thrown. Complete the function by doing the following:

1. Use a while loop to determine the time it takes for the ball to strike the ground.
    a. The initial position is the height, h.
    b. The initial vertical velocity is 0, so that term drops out.
    c. The downward acceleration is due to gravity, -9.8 meters per second per second. I have already defined a variable called "gravity" with this value.
    d. The loop should run until the current distance is 0 or less; that's when it hits the ground.
    e. During each iteration add one to the value of the time and recalculate the balls position.
    f. When the loop is done, you will know how many whole seconds it takes for the ball to hit the ground.
2. Use the value for the time taken to reach the ground to calculate how far the ball will travel horizontally.
    a. The initial horizontal velocity is given as the input parameter, *v*.
    b. The horizontal acceleration is 0. Once the ball leaves your hand you are no longer applying any force to it. The third term of the equation drops out.
    c. The initial position is also 0. The first term of the equation also drops out.
3. Return the distance describing how far the ball traveled as the result of the function.

NOTE: This method will overestimate the time taken to hit the ground by up to 1 second. So, the horizontal distance travelled will also be slightly overestimated.

Examples:

Function Call	Return Value
how_far(10.0, 100.0)	50.0
how_far(45.0, 2.0)	45.0
how_far(20.0, 100.0)	100.0
how_far(30.0, 100.0)	150.0
how_far(20.0, 10000.0)	920.0

Problem 5: Same Backwards as Forwards

A palindrome is a word or phrase that reads the same both forwards and backwards. "racecar" is a palindrome. So is, "a man a plan a canal, panama" if you ignore the spacing and the comma. Discovering how to automate the detection of palindromes is a good way to test your understanding of loops and strings.

The file, "hw2problem5.py", contains a function called `is_palindrome()`, which has a single string parameter, *s*. If *s* is a palindrome, then `is_palindrome()` returns the value True. If *s* is not a palindrome, then the function `is_palindrome()` returns the value False. Complete the body of the function by adding statements that will determine whether the input string *s* is a palindrome or not. Use a loop to perform this test. Return True if it is. Return False if it is not.

BE CAREFUL: The input string, *s*, can be a word or a phrase. If it is a phrase, like the example above, then you must ignore the spacing when testing whether it is a palindrome: only the letters and numbers matter. You can assume that none of the input strings will contain any punctuation. You can also assume that all the characters in the input string will be lowercase.

Examples:

Function Call	Return Value
is_palindrome("racecar")	True
is_palindrome("raceboat")	False
is_palindrome("a man a plan a canal panama)	True
is_palandrome("a place to call up")	False
is_palindrome("deified")	True