IAE 101 – Introduction to Digital Intelligence
Homework 01

Assigned: 09/06/2024
Due: **Sunday, 09/15/2024, at 11:59 PM**

Total Points: 50 (10 points per problem)

Assignment Objectives:

The goal of this assignment is to help you solve problems and implement algorithms in Python using the tools discussed in class: values, datatypes, and variables, operators, and conditional "if/else" statements.

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 circumstances, change the names of the functions or their parameter lists.

Directions:

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. **Do not change the names of the functions or parameters.**
- 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 of 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 of your completed `hw1problemX.py` files on Brightspace for Programming Homework 01.
- **Do not change the names of the files**.

Problem 1: Estimating the U.S. Population (10 points)

The webcomic xkcd (https://xkcd.com/1047/)gives the following formula to approximate the United States population for a given year (from 2000 onward):

1. Obtain the last two digits of the year (for example, 2017 would give you 17).
   -One way to do this is to calculate the year modulo 100 (`year % 100`).
2. Subtract 10 from this value
3. Multiply the result by 3
4. Add 310 to the product from the previous step

This gives the approximate U.S. population in millions of people for the given year.

For example, consider the year 2006.

1. The last two digits of 2006 are 06, or just 6. If you divide 2006 by 100, the result would be 20 with a remainder of 6—remember that the mod operation (*x* `%` *y*) returns the remainder when dividing *x* by *y*.
2. Subtracting 10 gives us -4.
3. Multiplying -4 by 3 gives us -12.
4. Finally, we add 310 to -12 to get our final answer: 298 (million people).

The "`hw1problem1.py`" file contains a Python function named `population()`, which takes a single argument: a positive integer called `year` that is greater than or equal to 2000, representing a four-digit year. Complete the definition of this function so that it calculates and returns an integer corresponding to the estimated U.S. population for that year in millions. Make sure you check that the input integer is greater than or equal to 2000. If it is, then calculate the population approximation as defined above. If not, then return the integer value `-1`.

Examples:

Function Call	Return Value
`population(2001)`	`283`
`population(2010)`	`310`
`population(2016)`	`328`

Problem 2: Restaurant Tipping (10 points)

After many years, you have developed a system for determining exactly how much to tip when eating out at a restaurant:

- If the final bill is $30.00 or less, you will leave a tip of exactly $8.00 (regardless of the quality of service)
- If the final bill was greater than $30.00 and the service was "good" (according to some criteria that we won't bother describing here), you will leave a tip of 29% of the final bill
- Otherwise (meaning that the final bill was over $30.00 but the service was "bad"), you will leave a tip equal to 16% of the final bill.

The "`hw1problem2.py`" file contains a Python function named `tip_amount()` that can help you decide how much to leave as a tip. It takes two arguments, in the following order: the amount of the final bill (a positive floating-point number) called `bill` and a Boolean value categorizing the quality of the service called `good_service` (`True` indicates "good" service and `False` indicates "bad" service). The function returns a floating-point number corresponding to the amount you will leave as a tip.

For example, `tip_amount(12.56, False)` returns 8.0, representing $8 (even though the service was bad, your system still requires you to leave a $8.00 tip). `tip_amount(48.93, True)` returns 14.189699999999998, representing a 29% tip (for this problem, don't worry about rounding the tip amounts to two decimal places).

Examples:

Function Call	Return Value
`tip_amount(23.37, True)`	`8.0`
`tip_amount(23.37, False)`	`8.0`
`tip_amount(81.75, True)`	`23.7075`
`tip_amount(63.59, True)`	`18.4411`
`tip_amount(63.59, False)`	`10.1744`

Problem 3: Time Conversions (10 points)

The file "`hw1problem3.py`" contains a single function named `getSeconds()`. This function takes exactly three string arguments, called `days`, `hours`, and `minutes`: the first represents the number of days, the second represents the number of hours, and the third represents the number of minutes. These values are derived from an 8-character string representing an amount of time in the form DD:HH:MM, where DD represents a number of days, HH represents a number of hours, and MM represents a number of minutes. This derivation is done for you in the problem code. getSeconds() returns a positive integer representing the total number of seconds in the input.

For example, the time value "02:18:49" (2 days, 18 hours, and 49 minutes) corresponds to a total of 240540 seconds: there are 86400 seconds in a day (60 * 60 * 24), 3600 seconds in an hour (60 * 60), and 60 seconds in a minute: so, we have (2 * 86400) + (18 * 3600) + (49 * 60), which adds up to 240540.

You may assume that the number of days always falls in the range 00-99 (inclusive), the number of minutes always falls in the range 00-59 (inclusive), and the number of seconds always falls in the range 00-59 (inclusive). **Take note, the split function separates out the numbers in the time string, but it does not convert them into integers.**

Examples:

Function Call	Return Value
`getSeconds("11", "11", "14")`	`990840`
`getSeconds("00", "01", "02")`	`3720`
`getSeconds("05", "00", "30")`	`433800`
`getSeconds("00", "00", "00")`	`0`
`getSeconds("02", "18", "49")`	`240540`

Problem 4: Temperature Conversion (10 points)

There are two primary scales for measuring temperature, Fahrenheit and Celsius. I watch a lot of European TV and I am always confused about how hot it is supposed to be in the story because they use Celsius rather than Fahrenheit. I want you to build a program to help me (and my European counterparts who face the opposite problem).

To convert a Fahrenheit temperature value, x, to Celsius:

1. Subtract 32 from x.
2. Multiply the result of line 1 by 5.
3. Divide the result of line 2 by 9.

For example: If the temperature in Fahrenheit is 68 degrees. Then we do the following:

1. `68 - 32 = 36`
2. `36 * 5 = 180`
3. `180 / 9 = 20` degrees Celsius

To convert a Celsius value, x, to Fahrenheit, we do the opposite:

1. Multiply x by 9.
2. Divide the result of line 1 by 5
3. Add 32 to the result of line 2.

For example: if the temperature in Celsius is 25 degrees. Then we do the following:

1. `25 * 9 = 225`
2. `225 / 5 = 45`
3. `45 + 32 = 77` degree Fahrenheit

The file "`hw4problem4.py`" contains a single function called "`temp_converter()`". The function takes exactly two arguments, called `temperature` and `new_unit`. The first argument, `temperature`, is a float value that represents a temperature value. The second argument, `new_unit`, is a single character string, either "`F`" or "`C`". If it is "`F`", then the function will treat the value in temperature as a Celsius value and covert it to Fahrenheit. If it is "`C`", then the function will treat the value in temperature as a Fahrenheit value and convert it to Celsius. temp_converter() returns the converted value.

Examples:

Function Call	Return Value
`temp_converter(100, "C")`	`37.77777777777778`
`temp_converter(0, "F")`	`32.0`
`temp_converter(212, "C")`	`100.0`
`temp_converter(32, "C")`	`0.0`
`temp_converter(50, "F")`	`122.0`

Problem 5: The Depressingly Large and Slow Universe (10 points)

In a vacuum light travels at approximately 186,000 miles per second, which seems fast until you begin to reckon with just how far apart most objects in the universe are. If the speed of light represents a hard limit on the rate at which we can travel, then it seems unlikely we are ever going to travel any significant distance from the Earth. In fact, the fastest that a crewed space vessel has ever traveled (Apollo 10) is less than 1% of the speed of light. I want you to complete the implementation of a program that will help you to share the existential apathy I experience whenever I consider these facts about the world.

The file "`hw1problem5.py`" contains a single function called `how_long()`. This function takes three arguments called `distance`, `fraction`, and `scale`.

- `distance` is a floating-point value representing the distance in miles between the Earth and some other object in the universe.
- `fraction` is a floating-point value representing the speed at which humans can travel as a fraction of the speed of light.
- `scale` is a single character string that determines whether the result is in minutes (`"M"`), hours (`"H"`), days (`"D"`), or years (`"Y"`).

The function includes a variable called `c` that represents the speed of light measured in miles per second. `c` is initialized to 186,000 and should not be changed. The function returns a float representing the amount of time it would take to travel the specified `distance` given the speed humans can travel as specified by `fraction`, in the units of time specified by `scale`.

Function Call	Return Value
`how_long(238900, 0.01, 'M')`	`2.140681003584229`
`how_long(45594000, 0.01, 'H')`	`6.809139784946236`
`how_long(93000000, 1.0, 'M')`	`8.333333333333334`
`how_long(9000000000, 0.01, 'D')`	`56.00358422939068`
`how_long(25000000000000, 0.01, 'Y')`	`426.20688150221224`

Notice that if we could travel at 1% of the speed of light (which is at least twice as fast the fastest a human-crewed space vessel has ever travelled), we could tool around the solar system pretty comfortably—about 2 minutes to the moon, about 7 hours to Mars, about 56 days to the outer edge of our solar system, but that's it. At 1% of light speed it would take us about 426 *years* to reach the nearest star system at Alpha Centauri. We're at the bottom of a gravitational well, and, unless something truly remarkable happens, we will be stuck here forever.