CS 112 – Fall 2020 – Programming Assignment 4

Repetition (For and While Loops)

Due Date: Sunday, September 27th, 11:59pm

The purpose of this assignment is to get more practice with writing repetitive algorithms using the for loop and while loop structures.

See the **"assignment basics"** file for more detailed information about getting assistance, running the test file, grading, commenting, and many other extremely important things. Each assignment is governed by the rules in that document.

Note: the tester is now integrated with Gradescope!
You also get an offline version so you can test without the internet.

Background

The repetition structures (for/while) allow the programmer to repeat a block of code for a specific amount of time. In this assignment you will practice using these structures to solve problems. Bear in mind that you may still need the decision structure from previous weeks to solve these problems! These problems ramp up in difficulty, **start early and ask questions on Piazza often**.

Guidelines

- Think carefully about the order you will check different properties. You might want to write some pseudocode or perhaps draw a flowchart if this works better for you. Think first, then implement.
- Be careful what kinds of loops you use, and be sure to test your code with many examples. Remember that some problems are easier to solve with a for loop instead of a while, and vice-versa.
- When a specific test case isn't working, plug your code into the `visualizer` to watch what the code does, which lines run, which branches are taken.
- From built-in functions, you are allowed to call `range()`,`round()`,`int()`, `float()`, `bool()`, `str()` only. This means you are **not allowed** to use any built in list methods, including but not limited to `list.count()`
- You are **not** allowed to `import` anything.
- You are **not** allowed to any feature that hasn't been covered in class yet.
- Don't forget to review the "**assignment basics**" file
- Insert comments in the lines you deem necessary (hint: more than 0 are necessary)

General Assumptions

You may assume that:

- The types of the values that are sent to the functions are the proper ones you don't have to validate them.
- The functions are going to be called with usable values, you don't have to validate them (e.g. velocity and acceleration won't both be zero).
- Be sure to read the assumptions made in individual problems!

Testing

In this project testing will be slightly different from the previous one. There will be **no user input or print** statements this time. You'll given a number of tasks and for each of them you must implement one Python function. A template with the functions is provided to you. The tester will be calling your functions with certain arguments and will be examining the return values to decide the correctness of your code. Your functions should not ask for user input and should not print anything, just return a value based on the specification of the tasks. When you submit your code to Gradescope, you can see the results of the tests.

Grading Rubric

Submitted correctly:	5	# see assignment basics file for file requirements!
Code is well commented:	5	# see assignment basics file for how to comment!
Tester calculations correct:	90	# see assignment basics file for how to test!
TOTAL:	100	

The following are the functions you must implement for PA4:

`def sum_divisors(n):` **(15 points)**

Description: Compute **the sum** of all the divisors of n. A divisor of n is a number x that evenly divides into n (there is no remainder when dividing n by x).

Parameters: **`n`** (`int`)

Return value: an integer, the sum of the divisors of n

Examples:

```
sum_divisors(1)      →   1   # only 1 divides itself
sum_divisors(5)      →   6   # 5 + 1
sum_divisors(10)     →   18  # 1 + 2 + 5 + 10
sum_divisors(49)     →   57  # 1 + 7 + 49
```

`def negative_product(nums):` **(15 points)**

Description: Compute the product of the negative numbers in a list

Parameters: **`nums`** (`list of ints`)

Return value: an integer, the product of all negative numbers in a list

Examples:

```
negative_product([1,2,3,4])     →   1    # no negatives
negative_product([1,2,3,-4])    →   -4
negative_product([1,-2,3,-4])   →   8
negative_product([1,-2,-3,-4]) →   -24
```

`def heater(fuel, temp):` **(15 points)**

Description: You want to heat your home from its current temperature to 80 degrees Fahrenheit. You have a certain amount of fuel to burn. You can increase your home's temperature by 5 degrees by burning one unit of fuel. You cannot burn partial units of fuel. Write a function to figure out if you can heat your home to the desired temperature.

Parameters: **`fuel`** (`int`) number of units of fuel you have, **`temp`** (`int`) is the current temperature of your home in degrees Fahrenheit

Return value: a string based on the fuel you burned and temperature of your house

Examples:

```
heater(10, 50)   →   "success, 4 leftover fuel!"
heater(5, 55)    →   "success, no leftover fuel!"
heater(5, 50)    →   "failure, highest temp is 75"
heater(1, 90)    →   "success, 1 leftover fuel!"
```

def analyze(activity, time_period): **(25 points)**

Description: Write a function that determines what percentage of your time you spent doing a certain activity. You track the time spent working, eating food, taking a break, and sleeping. You track every 20 minutes you work **("w")**, every 15 minutes you eat **("f")**, every 10 minutes you take a break **("b")**, and every 60 minutes you sleep **("s")**. When tracking time, you write down a letter to represent how you spent that time.

For example, if you worked for 40 minutes, then took a 10-minute break, ate for 30 minutes, and then slept for 2 hours, the string for that period of time would be **"wwbffss"** The total time covered in this period is 40 + 10 + 30 + 120 = 200 minutes. The percentage of this time that is spent working is 40/200 = 20%

Parameters: **activity** [str] a single character that represents an activity **("w", "f", "b", or "s")**
time_period [str] a long string that represents the things done in a period of time, will be comprised only of activity characters

Return value: a float (rounded to two decimal places) that is the percentage of that activity for that period

Examples:

```
analyze("w", "wbfs")       →   19.05
analyze("f", "wbfs")       →   14.29
analyze("b", "wwbbwwff")   →   15.38
analyze("w", "wwbbwwff")   →   61.54
```

---

def travel(position, velocity, acceleration): **(20 points)**

Description: Imagine that a ball is traveling on a line. This line is marked every meter from 0 to 1000. The ball starts at one of these markers, and changes its position each second according to some velocity (the rate of change of position). The velocity is also changing each second according to the acceleration. Write a function that counts the number of steps the ball takes to reach the beginning or the end of the line, using a loop to simulate the changes in position and velocity with each passing second.

Parameters: **postion** (int) where the ball starts on the line; **velocity** (int) how fast the ball travels per second and in what direction (negative or positive) **acceleration** (int) how velocity changes (slower or faster)

Return value: an integer, the number of steps the ball takes until reaching the end of the line

Examples:

```
travel(10, -1, 0)          →   10
travel(10, 10, -1)         →   22
travel(900, -50, 10)       →   13
travel(500, 75, 25)        →   5
```