CS 112 – Fall 2020 – Programming Assignment 2

Basic Expressions and Branching

Due Date: Sunday, September 13th, 11:59pm

The purpose of this assignment is to get more practice with basic statements and expressions and to start working with selection statements.

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!

Background

Selection statements (**if/elif/else** combinations) allow us to write code that can execute different statements based on the current values seen in a particular run of the program. We will use this to write a program that performs calculations and selectively reports on different properties of the calculated values.

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 selection statements you use, and be sure to test your code with many examples. For instance, multiple `if` statements will not necessarily behave the same as a chain of `if-elif-elif`.
- 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 `int()`, `float()`, `str()` only.
- You are **not** allowed to `import` anything.
- You are **not** allowed to use loops or 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 (age is an integer not a float,

etc.), you don't have to validate them.

- The functions are going to be called with usable values (e.g. r_pizza is a float, etc.), you don't have to validate them.
- 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:	15	# see assignment basics file for how to comment!
Tester calculations correct:	80	# see assignment basics file for how to test!
TOTAL:	100	

Functions

In this project things are a bit different from the previous one. You won't write a monolithic program as you did in Programming Assignment 1, but you're going to implement the following functions. The signature of each function is provided below, do ***not*** make any changes to them otherwise the tester will not work properly. Keep in mind that you must ***not*** write a main body for your program in this project. You should only implement these functions; it is the tester that will be calling and testing each one of them. Last, be reminded that the generic structure of a function is the following:

```
def function_name(arg1, arg2, etc.) # this line is the signature
      # commands go here
      # more commands
      # result = ...
      return result     # this line returns the result of your computations
```

The following are the functions you must implement for PA2:

`def seek_shelter(seconds, inside)`

Description: A common rule of thumb for thunderstorms is that the storm is one mile away from you for every five seconds between the sound of thunder and the sight of lightning. You should seek shelter from the storm when the storm is within five miles of you. Write a function that determines if you should seek shelter from the storm based on this rule, and whether or not you are already inside.

Parameters: **`seconds`** (`int`), the number of seconds between thunder and lightning, **`inside`** (`boolean`) True if you are inside, False otherwise

Return value: one of two strings: "yes" if you should seek shelter, or "no" if you don't have to

Examples:

seek_shelter(30, True) → "no"
seek_shelter(20, False) → "yes"
seek_shelter(18, True) → "no"

`def energy(age, amt_sleep, cals_burned)`

Description: Determine the energy level of a person based on their age and sleep level. People who are 30 and younger are considered "young", anyone 60 or older is "old", and anyone in between is "middle aged." Every age group requires a different amount of sleep to feel "rested." Young people need at least 8 hours, middle aged people need at least 6 hours, and old people need at least 4 hours. Anyone who gets at least half as much sleep as they require will wake up "dazed", any less and they wake up "exhausted."

However, the amount of sleep that a person needs to feel rested changes depending on the number of calories they burned the day before. A young person who burned more than 2200 calories requires 2 extra hours of sleep. A middle-aged person who burns more than 2000 calories needs 3 extra hours of sleep. An old person who burns more than 1800 calories requires 4 extra hours of sleep.

Parameters: **`age`** (`int`) is the age of a person in years (1ft), **`amt_sleep`** (`int`) is the amount sleep that person got last night **`cals_burned`** (`int`) is the number of calories that person burned the day before

Return value: a string based on the person's age, amount of sleep, and number of calories burned

Examples:

energy(20, 8, 2000) → "young and rested"
energy(20, 4, 2000) → "young and dazed"
energy(20, 8, 2600) → "young and dazed"
energy(20, 2, 2000) → "young and exhausted"
energy(43, 5, 1500) → "middle aged and dazed"
energy(73, 1, 1000) → "old and exhausted"

one more function on the next page

`def pizza_coverage(r_pizza, r_pep, num_pep)`

Description: Suppose you work for your local pizza place. You are making a pizza of a certain radius. You are going to cover the pizza with little pepperonis of a certain radius. You only have a certain number of pepperonis left. Happy customers want **at least** 70% of the pizza to be covered with pepperoni. You will have sad customers if you can't cover **at least** 50% of the pizza. Anywhere in between leaves the customer feeling neutral. Complete this python function that returns whether you will have "happy" customers, "neutral" customers, or "sad" customers based on the number of pepperoni you have and the relevant sizes.

Parameters: **`r_pizza`** [integer] the radius (in cm) of the pizza you're making
`r_pep` [integer] the radius (in cm) of each pepperoni
`num_pep` [integer] the number of pepperoni in your shop

Assumptions:

- we only care about the total area of pepperoni on the pizza, overlapping pepperoni doesn't matter
- the radius of the pepperoni will always be smaller than the radius of the pizza
- only a whole pepperoni can go on a pizza, you cannot tear them to place a fraction on the pizza
- use 3.14159 for pi

Return value: one of the following three strings: **`"happy customer", "neutral customer", or "sad customer"`**

Examples:

`pizza_coverage(5, 1, 10)` → **`"sad customer"`**
`pizza_coverage(5, 1, 15)` → **`"neutral customer"`**
`pizza_coverage(5, 1, 20)` → **`"happy customer"`**