CMSC 201 Fall 2018
Project 1 – Spooky Camping by The Upside Down

Read this entire document before you begin!

Assignment: Project 1 – Spooky Camping by The Upside Down
Due Date:
Design Document: Monday, October 22nd, 2018 by 8:59:59 PM
Project: Monday, October 29th, 2018 by 8:59:59 PM
Value: 80 points

Collaboration: For Project 1, **collaboration is not allowed** – you must work individually. You may still come to office hours for help, but you may not work with any other CMSC 201 students.

Make sure that you have a complete file header comment at the top of each file, and that all of the information is correctly filled out.

```
# File:    FILENAME.py
# Author:  YOUR NAME
# Date:    THE DATE
# Section: YOUR DISCUSSION SECTION NUMBER
# E-mail:  YOUR_EMAIL@umbc.edu
# Description:
#    DESCRIPTION OF WHAT THE PROGRAM DOES
```

Instructions

For this project, you will be creating a single program, but one that is bigger in size and complexity than any individual homework problem that you have seen so far. This assignment will focus on using functions to break a large task down into smaller parts. You will be required to turn in a design document before you turn in your actual code.

This is the first assignment where you've had to turn in a "design document" in addition to the actual code. The design document is intended to help you practice deliberately constructing your program and how it will work, rather than coding as you go along, or starting without a plan.

A list of functions required for this project are found later in this document. You are ***required* *to follow the provided design exactly***! You may add additional functions, but you must implement all of the specified functions as described in the design overview.

At the end, your Project 1 file must run without any errors.
It must also be called proj1.py (case sensitive).

Additional Instructions – Creating the proj1 Directory

During the semester, you'll want to keep your different Python programs organized, organizing them in appropriately named folders (also known as directories).

You should create a directory in which to store your Project 1 files. We recommend calling it **`proj1`**, and creating it inside a newly-created directory called **`Projects`** inside the **`201`** directory.

If you need help on how to do this, refer back to the detailed instructions in Homework 1.

Objective

Project 1 is designed to give you lots (and lots) of practice with functions, as well as help you become more familiar and comfortable with lists. You'll need to use **`while`** loops, control statements like **`if/else`**, passing in parameters, returning from functions, shallow copies of lists, and algorithmic thinking.

Task

You'll be implementing a text-based adventure game. The premise of this game is that you went camping in the woods only to find out that a monster, ***The Demogorgon***, was set loose from the lab conveniently located in the same forest. The object of this game is to run away from that monster. If you can stay alive for seven days, or if you travel 150 miles to the nearest big city, then you win the game. If you die at the hands of the monster, or by some other means, you lose the game.

The next few pages will describe the basic functionality and details for this game. See the sample output (in a separate file) for detailed examples of how everything works, and continue reading for more information.

Overview

Once you are alerted to ***The Demogorgon's*** presence, you have a limited amount of time to act. Survive for seven days, or travel 150 miles to the closest city, and you will survive. This is harder than it sounds, because the monster is chasing you. If it catches up to you, you must fight it off. This isn't just any old video game that uses nostalgia to get you hooked, this is real life!

You start off with a backpack, which is where you can put things that you find while playing the game. Inside this backpack you have Walkie Talkie and Flashlight, the only items you would ever need while camping. You also start off with 100 health, which is the maximum health you are allowed to have in the game.

The Day

When you wake up for the day, you can complete one of three basic tasks. The player completes as many daily tasks in the morning as they want until they say they are done.

- View your inventory
  - Show the items you have in your backpack.
    - Note: Food is not stored in the backpack, as it is eaten immediately!
  - After you view your inventory, you should have the option to equip or unequip an item.
    - This means you can choose one item to hold.
    - All items in your inventory are equipable.
    - An item that you are holding should remain in your inventory, it should not be removed.
- View your stats
  - Your health
  - Your distance traveled
  - Your equipped item
- Eat breakfast
  - Your psychic abilities allow you to magically create Eggo Waffles out of thin air. They heal you by 10 points.
  - You can only create and eat one Eggo Waffle each morning!

When you have decided that you are done for the morning, you then have a choice. You can pack up your camp and leave to find civilization, or you can stay put for the day.

If you stay put for the day, there is a 70% chance that ***The Demogorgon*** reaches your camp and challenges you to a fight. The other 30% of the time you will successfully evade the monster, and your health should be restored to full.

If you leave for the day, there is a 60% chance that an event happens where you find food, an item, or you fall into a ditch. There is a 30% chance that ***The Demogorgon*** catches up to you and challenges you to a fight. There is a 10% chance that nothing happens.

The distance in miles that you travel in a day is a function of your health. You can compute the distance you will travel in a day by dividing your health by 4, and then adding 5. So, if you have a health of 100 on a day you choose to pack up and leave, then you would travel 30 miles toward civilization:

```
((100 / 4) + 5)
```

Randomness in Python is covered later in this document.

Random Walking Events

There are a few events that can randomly occur when you decide to spend your day walking towards civilization instead of staying put. (Follow the random number selection process on page 11.)

Event 1: Found Backpack with Food (20% chance)

You stumble upon someone's backpack that has been left behind. In this backpack you find one of the food items in the table on the right (each one has the same chance of appearing). You can then choose to eat the item, or to put it back.

Food	Health
Reese's Pieces	-30
Pop Rocks	-5
Ovaltine	+15
Wonder Bread	+25
Twinkies	+30

Event 2: Old Shed with Shelves (20% chance)

You stumble upon an old shed lined with shelves. You find one of the following items on the shelves (each item has an equal probability of appearing) and you put it in your inventory. When an item is in your inventory, it should have the effect described.

Item	Effect
Sword	Equipable weapon
Bicycle	Travel 1.5 times as far as you normally would each day
Hi-C	Halves ***The Demogorgon's*** health at the start of a fight
Heelys	Travel 1.25 times as far as you normally would each day
Walkman	Decreases ***The Demogorgon's*** attack by 25%
Laser Cannon	Equipable weapon
Rubber Band	Equipable weapon

The "Bicycle" takes precedence over the "Heelys," so if you have both in your backpack, your distance multiplier should be the one for "Bicycle."

Event 3: Fall into a Trench (20% chance)

You only make it half the distance you could have because you fall into a trench. You weren't paying attention to where you were going! You must take one extra day to recover (skip the next day). So, if you fell in the trench on day 3, you would wake up on day 5. If your health was 80 on day 3, you would travel 12.5 miles instead of the 25 miles you would have gone if you hadn't fallen in.

Event 4: Demogorgon Fight! (30% chance)

The Demogorgon catches up to you. You must fight it. The fight mechanic is described on the next page.

Event 5: Nothing Happens (10% chance)

It's better than dying!

The Fighting Mechanic

When ***The Demogorgon*** attacks you, you're going to want to fight back. Here's how it should work.

The Demogorgon's base health is 300. Its base attack is 20. If you have "Hi-C" in your inventory, then its starting health is halved. If you have a "Walkman" in your inventory, then its attack is decreased by 25%.

You and ***The Demogorgon*** should remain fighting as long as both of you have positive health. Each round of fighting the following things happen in this order:

1. Display your health, and the monster's health.
2. Decide if you want to "Fight", "Flail", or "Flee"
    a. If you choose to "Fight":
        i. Hit the monster with your equipped item (*if you have one*)
        ii. The monster hits you with its attack strength
        iii. Update Health for the player and the monster
    b. If you choose to "Flail":
        i. The monster hits you and knocks you out
        ii. Your health is set to zero (and you lose the game)
    c. If you choose to "Flee":
        i. There is a 30% chance that you actually escape
        ii. If you don't manage to escape, the monster hits you with half of its attack strength
            1. Update Health for the player and the monster

You cannot kill ***The Demogorgon*** for good. If the fight ends because the monster was killed, it can still come back and fight you again any day after that. Its health resets to a full 300 at the beginning of each fight.

Weapons

The items in the table to the right deal damage when equipped. They must be equipped in order to deal damage, they cannot just be in your inventory. Remember, you can only equip one item. All other items deal 0 damage

Item	Value
Flashlight	5
Walkie Talkie	10
Rubber Band	25
Sword	50
Laser Cannon	100

Randomness

Built-in functions like **`print()`** and **`input()`** are automatically recognized by Python when you use them because they are standard functions that almost every program will need to use. There are quite a few built-in functions in Python that are not available to you automatically. You must manually import them into your code.

You'll notice at the very top of the starter design file (details on page 13) there is a line of code that looks like this:

```
from random import randint
```

This line of code allows you to use the **`randint()`** function that is built into Python. This function takes two integer arguments: a start and stop for a range. It returns a random integer within that range (inclusive of the start and the stop).

Here are some example runs:

```
linux1[4]% python3
Python 3.6.6 (default, Jul 19 2018, 14:25:17)
[GCC 8.1.1 20180712 (Red Hat 8.1.1-5)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> from random import randint
>>> randint(1, 5)
1
>>> randint(1, 5)
5
>>> randint(1, 5)
4
>>> randint(1, 5)
1
>>>
```

This is the only random function that you are allowed to use.

This is the only imported function that you are allowed to use.

Once you have a way to pick random numbers, how can you use that to make an event occur in your code with some probability? Random numbers generated by this function are uniform, meaning that each number has an equal probability of being returned by this function. We can use this to our advantage.

For example, suppose that there is a 40% chance that my program will say "Hello, world!" If we select a random number from 10 choices, each number has a 10% chance of appearing. Therefore, if I have my program print this message when any one of four numbers are selected, it will happen 40% of the time:

```
from random import randint

def main():
    num = randint(1,10)

    # print 40% of the time
    if num <= 4:
        print("Hello world!")
```

At the top of the design file that is given to you will also see a call to the **`seed()`** function. *<u>This function is for testing purposes only.</u>* If you want to match the given sample output, the number passed into this function must be 100. You can change that number to get different results for the randomness.

Your final submission of `proj1.py` must include both lines as they are seen in `design1.txt`, so the number used must be 100.

Randomness in this Project

As you've read, there is a lot of randomness in this project. To make it easy, you should follow this breakdown of what to do with the random number you generate. **Every random number you generate should either be**

a) **within the range 1 to 10**
b) **0 to the length of a list minus 1 (a random valid index)**

When choosing a random number for an action after choosing to walk towards civilization:

1) If the action random number is 1 or 2, then the player finds the food.
   a) To pick a random food, pick a random number that is a valid index in the list of food (use this exact list in this exact order):
   ```
   ["Reese's Pieces", "Pop Rocks", "Ovaltine",
   "Wonder Bread", "Twinkies"]
   ```

2) If the action random number is 3 or 4, then the player finds the shed.
   a) To pick a random item, pick a random number that is a valid index in the list of items (use this exact list):
   ```
   ["Sword", "Bicycle", "Hi-C", "Heelys",
   "Walkman", "Laser Cannon", "Rubber Band"]
   ```

3) If the action random number is 5 or 6, then the player falls in the trench.

4) If the action random number is 7, 8, or 9, then the player fights ***The Demogorgon***.
   a) When choosing to flee from ***The Demogorgon***, select another random number from 1 to 10.
      i. If the fleeing random number is less than or equal to 3, you successfully run away.
      ii. Otherwise, you do not succeed in running away.

5) If the number is 10, then nothing happens.

When choosing a random number after deciding to stay at your camp:
If the staying random number chosen is less than or equal to 7, then ***The Demogorgon*** attacks. Otherwise, you get to rest at camp.

Coding Standards

Prior to this assignment, you should re-read the Coding Standards, linked on the course website at the top of the "Assignments" page.

For now, you should pay special attention to the sections about:

- Comments
  - ***Function header comments***
  - Please note that the "Input" and "Output" in the function header comment do NOT mean what is shown on the screen with **`print()`**, or what is gotten from the user with **`input()`**. They refer to the **parameters** taken in, and the **return value**. (Both "Input" and/or "Output" may be None if appropriate.)
- Constants
  - You must use constants instead of magic numbers or strings!!!
- Make sure to read the last page of the Coding Standards document, which prohibits the use of certain tools and Python keywords.

Additional Specifications

For this assignment, **you must follow the design overview** in this document.

The use of Python constructs that we have not covered in a formal lecture is explicitly forbidden in this project. Some things you should definitely not use include: try/except statements, default parameters, and dictionaries.

For this assignment, you do need to worry about "input validation." You may assume that the user will enter an integer, but it may be negative or outside of the allowable range.

If the user enters a different type of data than what you asked for, your program may crash. This is acceptable.

You are welcome to add additional print statements to your game to add your own creative spin, ***but do not deviate from the specifications!*** If you have any questions, ask them early!

Project

The project is worth a total of 80 points. Of those points 10 will be based on your design document, 10 will be based on following the coding standards, and the other 60 will be based on the functionality and completeness of your project.

Design Document

The design document will make sure that you begin thinking about your project in a serious way early on in the process. This will not only give you important experience doing design work, but will help you gauge the number of hours you'll need to set aside to be able to complete the project. **Your design document must be called design1.txt.**

For Project 1, the design overview is included in this document, and you are **required to follow it**. For future projects, you will be creating the design entirely on your own, and may choose to design it however you like.

Your design document must have four separate parts:

1. A file header, similar to those for your assignments
2. Constants
   a. A list of all the constants your program will need, including a short comment describing what each "group" of constants is for
3. Function headers
   a. A complete function header comment for each function
   b. You do not need to include the function's code in your design
4. Pseudocode for `main()`
   a. A brief but descriptive breakdown of the steps your `main()` function will take to completely solve the problem; note function calls under the relevant comment (if applicable)

Although you will be presented with a design overview, you must still create the function headers and the pseudocode for `main()` on your own.

A start for your design is provided on Blackboard under "Assignments". Follow the layout and format of that document. You can also copy it using:

```
cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/design1.txt .
```

NOTE: The sample design provided is not complete, and is missing many constants and function header comments. You must add them!

Your **`design1.txt`** file will be compared to the **`proj1.py`** file that you submit. Minor changes to the design are acceptable. A minor change might be the addition of another function, or a small change to **`main()`**.

Major changes between the design and your project will lose you points. This would indicate that you didn't give sufficient thought to your design. *(If your submitted design doesn't work, it is generally better to lose the points on the design, and to have a functional program, rather than turning in a broken program that follows the design. The decision is ultimately up to you.)*

To submit your design document, use

```
linux1[4]% submit cs201 PROJ1_DESIGN design1.txt
Submitting design1.txt...OK
linux1[5]% █
```

Sample Output

The sample output is available as a separate file under "Assignments" on Blackboard, and is called "sample1.txt". Look at the sample output before reading the notes below. More sample output will be posted as it becomes available.

(Yours does not have to match the sample output exactly, but it should be *very* similar.)

Other Hints

- Writing up and testing each of your functions individually will make your life much, much easier.
- Commenting your code as you write it or even before you write it (think pseudocode) will make the process much simpler, and will also allow the TAs to help you more effectively during office hours.

- This project lends itself very well to a "top down" implementation, where functions are first made as dummy functions (functions that only print something), and you ensure that everything works together.

Design Information

You are required to implement and use at least the following five functions for Project 1, in addition to **`main()`**. You may implement more functions if you think they are necessary, but the five below must be implemented **and used**, or you will lose significant points on your project grade. The information for each function is below.

Printing Functions

As the name suggests, these types of functions print information to the user. They do **not** return values – they only print information (which does not count as output in your function header). Printing functions do not always take in parameters, but this one does.

- **`def displayMenu(choices)`**
  - Prints out any of the menus in the program. Here is an example menu this function might generate:

    ```
    Your options are:
    1 - View Inventory
    2 - View Current Stats
    3 - Eat an Eggo Waffle
    4 - Nothing else
    ```
  - Takes in a list; the choices for menu options
  - Returns nothing, since it is a print function

Helper Functions

As the name suggests, these functions "help" other functions, by performing small tasks that will be needed by many pieces of the program. They are often called from functions other than **`main()`**.

- **`def getUserChoice(choices)`**
  - Gets an integer from the user that falls within the valid range for the given list of choices. That range goes from 1 to **`len(choices)`** inclusive.
  - Takes in a list; of possible choices
  - Returns an integer

- **`def calcDamage(item)`**
  - Computes the amount of damage that the item passed would inflict when used
  - Takes in a string; the item that you want to calculate damage for
  - Returns an integer; the amount of damage that the item inflicts

General Functions

These are the “heavy lifters” of the program, and do the majority of the work, and are almost always called from **`main()`**. They often call other functions, often more than one.

- **`def eat(food, player_health)`**
  - Computes the health boost offered by a food that a player has chosen to eat based on the health value of that food and player health
    - A player's health cannot exceed 100
  - Takes in a string and a number
    - **`food`** – the name of the food the player is going to eat
    - **`player_health`** – the amount of health the player has *before* eating the food item
  - Returns an integer; the new health of player after eating the item

- **`def fight(player_health, item, inventory)`**
  - Allow the player to fight ***The Demogorgon***
    - Fight until someone dies, or you successfully run away
    - Certain items in the player inventory provide special boosts
    - The player can also "Flail" (a.k.a. do nothing)
  - Takes in an integer, string, and a list
    - **`player_health`** – the starting amount of health
    - **`item`** – the item that the player has equipped
    - **`inventory`** – the list of items the player has available
  - Returns an integer; the remaining health the player has after the fight is concluded

  - **WARNING:** This function is the most complex one (other than **`main()`**, of course) in the whole project. Take some time to think carefully about what it needs to do, and plan how it will accomplish that before you start writing any code for it.

<u>Helpful Checklist</u>

We understand that the first big project can be overwhelming, which is why we've provided you with a detailed description of all the functions, along with sample output, a starter design, and more. Below, you'll find a summary of the key actions and where important documents can be found.

<u>Design:</u>

- ☐ Download the starter design from Prof. Neary's public directory:
  `cp /afs/umbc.edu/users/m/n/mneary1/pub/cs201/design1.txt .`
- ☐ Read through the function descriptions provided in this PDF document, and create function header comments for each function in your design file
- ☐ Look at the sample output on Blackboard, and begin sketching out an idea in pseudocode for how your **`main()`** will work
  - ☐ Look at the function headers you wrote to gain more insight

<u>Coding:</u>

- ☐ [Optional step] Make a copy of your design (or your design so far), as a framework for starting the project
- ☐ Going off of your design, start coding up your project piece by piece
  - ☐ Use top-down or bottom-up implementation – your choice!
  - ☐ <u>Don't</u> code up everything at once, and <u>do</u> test as you go
- ☐ If you make changes to your design while coding, and it's before the design due date, update your design and resubmit it!
- ☐ Come to office hours when you need help or get stuck

<u>General:</u>

- ☐ Submit the design to PROJ1_DESIGN, <u>**not**</u> PROJ1
- ☐ Start working on the project itself <u>before</u> October 22nd, since the act of coding it up will improve your design as you go
- ☐ Test your code with the user choices made in the sample output

Submitting

Once your **`proj1.py`** or **`design1.txt`** file is complete, it is time to turn it in with the **`submit`** command. (You may also turn the project in multiple times, as you reach new milestones. To do so, run **`submit`** as normal.)

To submit your design file (which is due Monday, October 22nd, 2018 by 8:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ1_DESIGN design1.txt
Submitting design1.txt...OK
linux1[5]%
```

To submit your project file (which is due Monday, October 29th 2018 by 8:59:59 PM), use the command:

```
linux1[4]% submit cs201 PROJ1 proj1.py
Submitting proj1.py...OK
linux1[5]%
```

If you don't get a confirmation like the one above, check that you have not made any typos or errors in the command.

You can check that your assignment was submitted by following the directions in Homework 0. Double-check that you submitted your homework correctly, since **an empty file will result in a grade of zero for this assignment.**