

Name: \_\_\_\_\_

For this project, you will be creating a turn-based text adventure game. A treasure will be located at some coordinate on a  $5 \times 5$  grid. The player will start at some different coordinate in the center of the game map. The player's goal is to locate the treasure by moving one grid square each turn, while possibly overcoming obstacles. The game will continue until the treasure is found, or the player "loses" by one of several different ways.

## Game Start Setup

1. (5 points) Create a function named `getPlayerName` that takes no input parameters. This function will ask the user for their name. It must validate that the name is at least 3 letters long. If the name is not 3 letters long, loop and continue to ask until the user enters a valid name. Make the first letter of the name capitalized. All other letters should be lowercase. The function will then return the string containing the name.

When displaying output messages to the user throughout the game, be sure to use the player's name as part of the output. For example, if your player's name is "John" and the player moves north, it should display "John moved North." or something similar.

2. (3 points) Write a function named `initGlobals` that takes no input parameters and returns no value. This function will provide initial starting values for all global variables. This function should be called exactly once each time a new game starts.
  - The player's name should be initialized to an empty string.
  - The player's  $x$  and  $y$  coordinates should be reset to 0.
  - The number of turns remaining should be set to 20.
  - The treasure's  $x$  and  $y$  coordinates should be set to a random integer. See the last page for example code...
3. (2 points) Write a function named `showInstructions` that takes no input and returns no value. This function will display instructions about how to play your game to the console. Be sure to include any necessary details, including what keys are valid for movement, etc.
4. (2 points) An empty function named `startGame` has been provided. It is called from `main`. Whenever the `startGame` function is called, it should call the above functions to setup the game world for a new game.
5. (3 points) The `main` function that is provided just calls the `startGame` function. Modify the `main` function so that when the game ends, it will ask the player if they want to play again. If they answer with a 'y' or a "yes" (or any capitalization of that) the program

will call `startGame` again. This should be implemented using a loop so that they player can play as many times as they want. If the user does not provide a *yes* response, the program will end.

## The Coordinate System and Movement

The game world is a  $5 \times 5$  grid. There isn't really a "grid" or squares, though. Everything in the game is *implied* through a coordinate system. That is, every object and location in your game is just a pair of integers. This is why things like the player's location is stored as two integer variables: one for the  $x$ -coordinate, and one for the  $y$ -coordinate.

Because the game world is  $5 \times 5$ , we need to decide how we want to arrange our coordinates. There are two sensible possibilities. The first (and preferred) is to have (0,0) be the center of the game world. This would mean that the coordinate systems in both the  $x$  and  $y$  axes would run from  $-2$  to  $+2$ .

The alternative is to have a coordinate system that runs from 0 to 4 in both axes, which would put the center at (2,2). If you choose this, make sure that you set the correct values inside your `initGlobals` function when the game begins.

Your player will start at the center of the game world. Each game turn, they are allowed to move one unit in either the horizontal or vertical directions. This is just a change of the player's coordinate value  $\pm 1$  depending on the direction that they are moving in. If they are moving horizontally, it will be the  $x$  coordinate that is changing. If they are moving vertically, it will be the  $y$  coordinate that is changing.

## Game Turns and Rules

When the game begins, the player will have 20 turns to find the treasure. The player should know how many turns are remaining. A turn is composed of the following actions:

1. Ask the player for a move
2. Move the player (if allowed)
3. Check to see if the player landed on something, and if so, respond accordingly.

To implement 1) and 2) above, we will have the following:

6. (10 points) The `gameLoop` function will contain a loop that will run for as long as there are turns remaining and the user hasn't found the treasure or lost in some other way.
7. (10 points) Write a function named `getPlayerMove`. This function takes no input parameters. It should ask the user which direction they want to move. Movement is allowed in four directions only. You can call them whatever you want as long as it is reasonable. Common choices are:

- ‘N’, ‘S’, ‘E’, and ‘W’ (for North, South, East, and West)
- ‘W’, ‘A’, ‘S’, and ‘D’ (should be familiar to gamers)
- ‘U’, ‘D’, ‘L’, and ‘R’ (for Up, Down, Left, and Right)

This function will prompt the user to type a valid choice. It should also work for any capitalization. If the user does not type a valid choice, it will continue to ask them until they do.

The function should return an uppercase version of what the user typed.

8. (10 points) Write a function named `movePlayer`. This function will take one input parameter, which should be a string indicating the direction of movement.

This function will modify the `playerX` or `playerY` variables  $\pm 1$  based on the provided direction.

Before moving the player, you must check to make sure that the movement is valid. Because we are on a  $5 \times 5$  map, the player cannot go outside of the map boundaries. There are several ways to deal with this. I don’t care which you choose, but you must implement something to deal with the off-the-map case. Common solutions are:

- Display a message to the user saying they can’t move in that direction (possibly costing them a turn)
- Have the player “fall off the map” and end their game.
- Warp the player to the other side of the map (in this case it would be considered an allowed move).

If the player is allowed to move, update the variables and return `True`. Otherwise, return `False`.

## The Treasure

Once the player has moved to the next location, we need to check to see if the player has landed at the same location as the treasure.

9. (5 points) Write a function named `isTreasure` that takes no input parameters and returns `True` if the player’s coordinates are the same as the treasure’s coordinates, or `False` otherwise.

You will call this function from your `gameLoop` function. If the treasure has been found, it should display a congratulatory message and the game loop should end. Otherwise, the game loop should continue.

## Obstacles

The treasure isn't the only thing that your player could land on. There will be at least three obstacles located on your game map. Each obstacle will have its own  $x/y$  coordinates.

When you land on an obstacle, the user will be asked a question and given three attempts at answering. If the user fails to provide the correct answer after the third attempt, they lose.

10. (20 points) You must provide at least one obstacle of each of the following types in your game:

- An obstacle that has the user guess a number. The number should be between 1 and 5.
- An obstacle that presents the user with a True or False question about Python. The acceptable answers will be "T" or "F".
- An obstacle that generates two random integers from 1 to 20. It will then ask the user a simple math problem based on those numbers, which the user must correctly solve.

Each of the above should be implemented in a separate function. These functions should return `True` if the player succeeded in answering the question, or `False` if they failed. The functions should validate user input and treat invalid responses as a failed attempt at answering the question.

One additional obstacle should be implemented for when the player lands on the treasure. It should be moderately more difficult than the other obstacles.

### **When should obstacles be encountered?**

This is up to you. You can create them with random coordinates when you initialize your global variables at the beginning of the game. You can also write code to make it chance-based (every so many moves, some percentage of the time, etc.).

## Ways The Game Can End

There are only a few ways where the game can end:

- The player uses all of their turns and does not find the treasure.
- The player finds the treasure.
- The player goes out off the map boundaries.
- The player answers incorrectly at and of the game obstacles.

## Exit Survey

When your game has finished and the player no longer wants to continue playing, your program should display some messages providing feedback about the course. Let me know what you liked, what you didn't like, things you would like to see done differently, etc.

## Extra Credit

All of the above describe the *minimum* requirements for this project. You are encouraged to go beyond this and add additional features to your game. Various amounts of extra credit will be provided for additional features. The maximum amount of extra credit is 100 points (meaning that the maximum grade for this project will full extra credit is 200 points). The amount of extra credit depends on the amount and complexity of the features that are added.

Some examples of things students have done in the past are:

- Additional (and more complex) obstacles
- Power-ups (extra moves, tools that help the player to find the treasure, tools that allow them to skip an obstacle, etc.)
- Map displays in the console

You are encouraged to add a storyline to your game, but extra-credit will not be provided for just text by itself. It must be functional.

## Grading Notes

Some items in the project description have point values associated with them. To get those points, your program must do what is described and work correctly. The points listed do not sum up to 100. Programming style and comments will play a role in your grade. A large number of points are reserved for my own judgement about how your game works as a whole. Your game must be playable and work correctly to get a passing grade ( $\geq 60$ ) for this exam.

## Hints

- Start small (and early), and get one feature working at a time.
- Small functions that do a single task are easier to test than large functions that do many things. Consider writing functions and testing them individually to make sure they do what they are supposed to do.
- When working with your global variables inside functions, make sure to use the `global` keyword in the function. Otherwise, you will be creating new local variables instead!
- For items with random locations, consider putting them at a fixed location at first to make it easier to test your game. Make them random once it works.

- There is nothing really new in this project. Games (and all software) are just combinations of very simple things, like if statements, loops, etc.
- The use of *temporary* print functions to see the values of your variables is encouraged and can help make it easier to figure out what is going on in your program. You may also want to explore using the debugger if your development tools support that.

## Summary

- All code must be in functions. The only code allowed outside of functions is a call to the `main` function and any global variable declarations.
- You must acquire the name of the player and use it throughout your game.
- Player will always start in the center of the coordinate system.
- The treasure will be at a random coordinate.
- Player has 20 turns to find the treasure.
- A turn is: asking the player where to move, moving them, and reacting to anything the player lands on. You should display how many turns are remaining at the beginning of each turn.
- Movement commands must be validated. Loop until a valid command is entered.
- The player is not allowed to move outside the game coordinates.
- There must be at least 3 obstacles in your game that meet the requirements stated earlier.
- The player is allowed up to 3 attempts to solve the obstacle problem. If they fail, the game ends.
- The treasure should also present the player with an obstacle before allowing them to claim it.
- If the player finds the treasure and solves the obstacle, they should see a congratulatory message and the game will end.
- If the player runs out of turns, loses an obstacle, or “dies” by any other means, the game should end with a “Game over” message.

## Generating Random Integers

```
# This goes at the top of your program
import random

# Later on in your code...
# Use this to generate a random integer. The parameters
# for randint represent the lowest and highest integers
# that will be generated. In this example, it will
# produce a random integer between 1 and 10 (inclusive).

someVariable = random.randint(1, 10)

# If you try to print the value of this variable, it
# it will display a random number:

print(someVariable)
```

**Deliverables:** You should submit the following:

1. A single python (.py) file submitted in Blackboard.