

A PROJECT REPORT ON

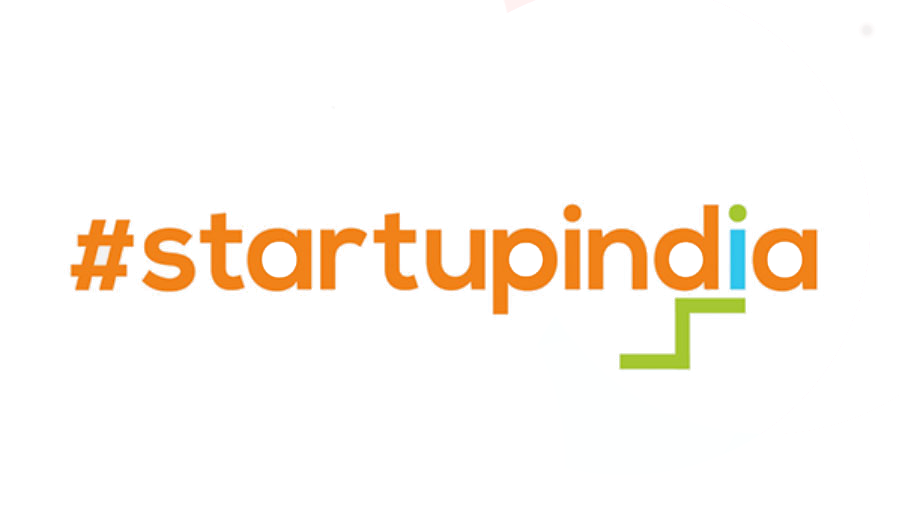
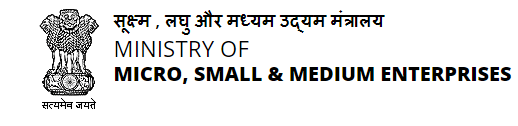
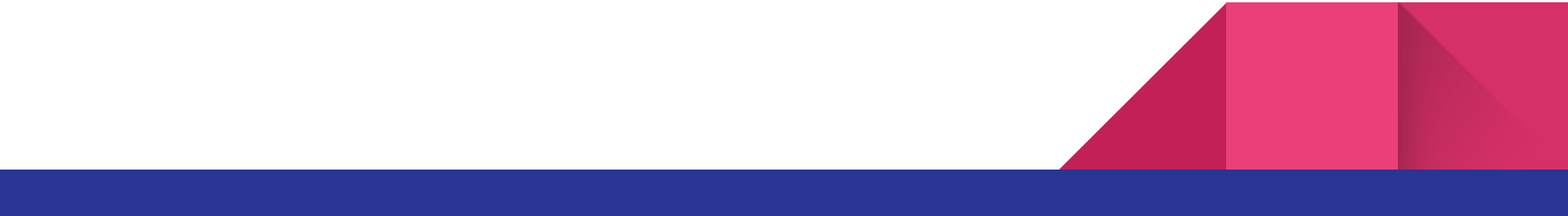
Guess Number (Python)

### Submitted To: Submitted By:

P Abhishek P. Thilak







**Shri Buddha Chandrasekhar**

Chief Coordinating Officer(CCO)

AICTE

**P Abishek**

Human Resources (HR)

Slash Mark IT Startup

ABSTRACT

The number guessing game is an interactive Python program designed to engage users in a fun and challenging activity. The game begins by requesting the user's name, providing a personalized experience. The program then generates a random number between 1 and 200, which the user must guess within six attempts. This game structure not only entertains but also tests the user's logical thinking and deduction skills. By incorporating pauses and feedback after each guess, the program maintains a dynamic and engaging pace, ensuring that users remain invested in the outcome.

Central to the game is the function that handles the user's guesses. This function, named pick(), systematically processes each guess, checking if it falls within the specified range and providing hints whether the guess is too high or too low. If the guess is outside the range, the user is prompted to enter a valid number. The program also includes robust error handling to manage non-numeric inputs gracefully, ensuring a smooth user experience. The feedback mechanism and guess validation not only enhance user interaction but also provide a structured approach to problem-solving within the game.

Upon either successfully guessing the number or exhausting all attempts, the game offers feedback on the outcome, congratulating the user if they guessed correctly or revealing the correct number if they did not. The option to play again is provided, making the game replayable and extending its entertainment value. This replayability, combined with personalized user interactions and a clear, engaging structure, makes the number guessing game a compelling and enjoyable experience for users. The program's design effectively balances simplicity and interactivity, making it accessible for a broad audience while providing a meaningful challenge.

# TABLE OF CONTENTS

#### Declaration 1 Abstract 2 Table of Contents 3



Problem Statement 4 o Purpose 4 o Goals and Objectives 4 o Description \_ 5



#### Scope 6 Introduction 7 o



#### Software Details \_ 8-9



#### Reference Code 10-11

#### Implementation Work Details(Reference code) 12-16

#### Key Functions 17-19

Error modifications 20-21



Updated code 22-23



Input and Output 24-28



Conclusion 29



PROBLEM STATEMENT

Develop an interactive Python program that engages users in a number guessing game, where the program selects a random number between 1 and 200, and the user must guess this number within a maximum of six attempts. The program should provide clear instructions and personalized interaction by asking for the user's name at the start. During the game, it should validate each guess to ensure it falls within the specified range and provide feedback indicating whether the guess is too high or too low. The program must handle non-numeric inputs gracefully and offer the user the opportunity to guess again or quit. Upon either correctly guessing the number or running out of attempts, the program should display an appropriate message indicating the outcome and prompt the user to play again if desired. The design should ensure a smooth and enjoyable user experience through effective error handling and interactive feedback mechanisms.

PURPOSE

The purpose of the number guessing game program is to provide an engaging and educational experience for users by combining entertainment with learning fundamental programming concepts in Python. It aims to teach practical skills such as random number generation, user input handling, control structures, and exception handling while encouraging logical thinking and problem-solving. Additionally, the game ensures a smooth and enjoyable user experience through personalized interactions, clear instructions, and effective feedback mechanisms, making it both a fun pastime and a valuable educational tool.

GOALS AND OBJECTIVES

The primary goal of the number guessing game program is to create an enjoyable and interactive game that challenges users to guess a randomly selected number within a limited number of attempts. By providing a personalized experience and immediate feedback on each guess, the program aims to keep users engaged and motivated to continue playing. The game’s structure encourages users to think logically and use deduction skills to narrow down the possible range of numbers, making each guess more informed than the last. This interactive element enhances the overall entertainment value of the game, ensuring that users find it both fun and stimulating.

The objective of the program extends beyond mere entertainment; it is also designed as an educational tool to help learners grasp key programming concepts in Python. By developing and interacting with the game, users can gain hands-on experience with fundamental programming techniques such as random number generation, input validation, loops, conditionals, and exception handling. The program’s design emphasizes good coding practices, including clear instructions, robust error handling, and a user-friendly interface, which are essential skills for any aspiring programmer. Additionally, the program's replayability allows users to repeatedly apply these concepts, reinforcing their learning and building confidence in their programming abilities.

### DESCRIPTION

The number guessing game is an interactive Python program designed to engage users in a fun challenge where they must guess a randomly generated number between 1 and 200 within six attempts. Upon starting the game, users are prompted to enter their name, adding a personalized touch to the experience. The game then provides feedback on each guess, indicating whether it is too high or too low, and ensures valid input through error handling for out-of-range or non-numeric entries. With clear instructions, immediate feedback, and the option to play again, the game combines entertainment with educational value, helping users develop logical thinking and fundamental programming skills in a user-friendly interface.

### SCOPE

1. **User Interaction**: The program starts by requesting the user's name, providing a personalized gaming experience. It guides the user through the process of making guesses, offering clear instructions and immediate feedback after each guess.
2. **Random Number Generation**: The program utilizes Python’s random module to generate a random number between 1 and 200, ensuring that each game session is unique and unpredictable.
3. **Input Validation and Error Handling**: The program validates user inputs to ensure they are within the specified range (1 to 200) and are numeric. It gracefully handles incorrect inputs by informing the user of the error and prompting them to try again.
4. **Game Logic and Feedback**: The core logic of the game includes checking the user's guess against the randomly generated number, providing feedback on whether the guess is too high, too low, or correct. The game limits the number of guesses to six attempts, maintaining a balance between challenge and playability.
5. **Replayability**: After each game session, the program asks the user if they would like to play again, allowing for multiple rounds of the game in a single run of the program. This replay feature enhances the program’s engagement and entertainment value.
6. **User Experience**: The program incorporates time delays using time.sleep() to create a more engaging and paced interaction. It aims to deliver a smooth and enjoyable user experience through thoughtful design and responsive feedback mechanisms.

Overall, the scope includes all essential elements to ensure the game is functional, user-friendly, and educational, providing a solid foundation for learning basic programming concepts and creating an interactive console-based application.

### INTRODUCTION

The number guessing game is a simple yet engaging Python program designed to provide both entertainment and educational value. The game begins by personalizing the experience, asking for the user's name, and explaining the rules: the user must guess a randomly selected number between 1 and 200 within six attempts. This introductory interaction sets the stage for a friendly and engaging game, drawing the user into the challenge.

At the heart of the game is a loop that handles user guesses, validating each one to ensure it is within the specified range and numeric. The program provides immediate feedback on each guess, indicating whether it is too high or too low, and prompts the user to guess again if incorrect. Robust error handling is implemented to manage non-numeric inputs gracefully, ensuring the game runs smoothly even if the user makes a mistake. This interactive feedback mechanism keeps the game dynamic and helps users refine their guesses, enhancing their logical thinking and deduction skills.

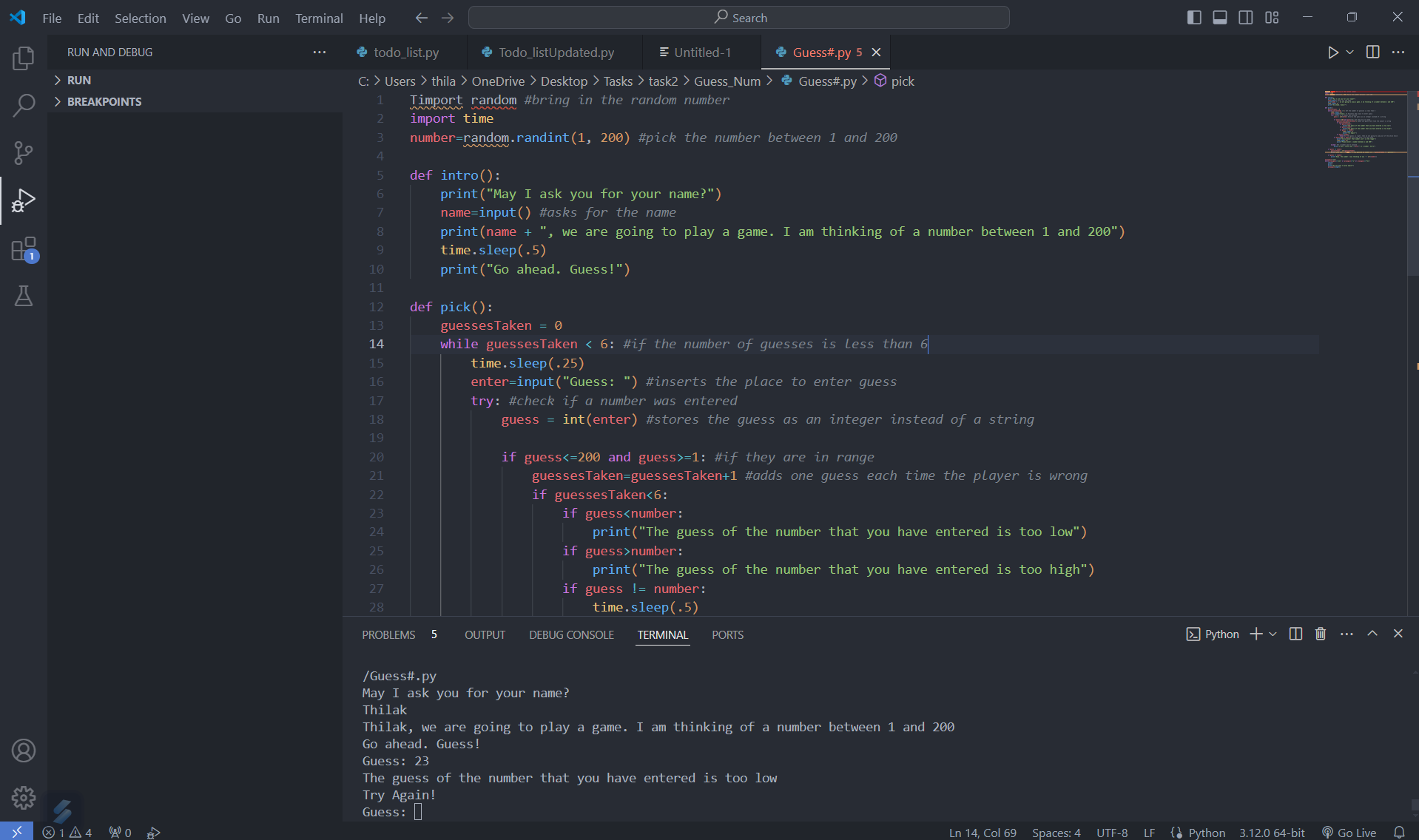
The game concludes by either congratulating the user for correctly guessing the number or revealing the correct number if the user exhausts all attempts. The option to play again is presented, allowing for repeated engagement and enjoyment. This replayability aspect, combined with the educational benefit of understanding Python programming concepts like loops, conditionals, and error handling, makes the number guessing game a versatile and valuable tool. It not only entertains but also serves as an excellent practice ground for those looking to improve their coding skills and logical reasoning.

### SOFTWARE DETAILS

Visual Studio Code (VS Code) is a highly popular, open-source code editor developed by Microsoft. Released in 2015, it has quickly become a favorite among developers due to its lightweight nature and powerful features. VS Code supports a wide range of programming languages and frameworks, making it a versatile tool for various types of development, from web applications to machine learning projects. Its user interface is highly customizable, allowing developers to tweak themes, keyboard shortcuts, and layout to fit their workflow. The built-in Git integration is a significant advantage, enabling seamless version control and collaboration directly within the editor.

One of the standout features of VS Code is its extensive marketplace of extensions. These extensions can enhance functionality with features like advanced debugging, code snippets, linting, and integrations with various development tools and services. The editor also includes IntelliSense, which provides intelligent code completion, parameter info, and quick info on hover, significantly boosting productivity. Moreover, VS Code's live share feature allows real-time collaborative coding, making it easier for teams to work together remotely. Regular updates and a strong community support make Visual Studio Code an evolving tool that adapts to the ever-changing landscape of software development.

VS Code



### COMPLETE GUESS NUMBER (REFERENCE)CODE:

import random

*#bring in the random number*

import time

number=random.randint(1, 200) *#pick the number between 1 and 200*

def intro():

    print("May I ask you for your name?")

    name=input() *#asks for the name*

    print(name + ", we are going to play a game. I am thinking of a number between 1 and 200")

    time.sleep(.5)

    print("Go ahead. Guess!")

def pick():

    guessesTaken = 0

    while guessesTaken < 6: *#if the number of guesses is less than 6*

        time.sleep(.25)

        enter=input("Guess: ") *#inserts the place to enter guess*

        try: *#check if a number was entered*

            guess = int(enter) *#stores the guess as an integer instead of a string*

            if guess<=200 and guess>=1: *#if they are in range*

                guessesTaken=guessesTaken+1 *#adds one guess each time the player is wrong*

                if guessesTaken<6:

                    if guess<number:

                        print("The guess of the number that you have entered is too low")

                    if guess>number:

                        print("The guess of the number that you have entered is too high")

                    if guess != number:

                        time.sleep(.5)

                        print("Try Again!")

                if guess==number:

                    break *#if the guess is right, then we are going to jump out of the while block*

            if guess>200 or guess<1: *#if they aren't in the range*

                print("Silly Goose! That number isn't in the range!")

                time.sleep(.25)

                print("Please enter a number between 1 and 200")

except: *#if a number wasn't entered*

            print("I don't think that "+enter+" is a number. Sorry")

    if guess == number:

        guessesTaken = str(guessesTaken)

        print('Good job, ' + name + '! You guessed my number in ' + guessesTaken + ' guesses!')

    if guess != number:

        print('Nope. The number I was thinking of was ' + str(number))

playagain="yes"

while playagain=="yes" or playagain=="y" or playagain=="Yes":

    intro()

    pick()

    print("Do you want to play again?")

    playagain=input()

### **IMPLEMENTATION WORK DETAILS** INPUT/OUTPUT SCREENSHOTS

**Inputs and Outputs**

**1. Introduction Phase**

**Input**: The user is asked for their name.

May I ask you for your name?

**Output**: The user's name is used to personalize the game.

[User Name], we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!

**2. Guessing Phase**

**Input**: The user is prompted to guess a number between 1 and 200.

Guess:

**Output**: Depending on the guess, the program provides feedback:

* If the guess is too low:

The guess of the number that you have entered is too low

Try Again!

* If the guess is too high:

The guess of the number that you have entered is too high

Try Again!

* If the guess is not in the range:

Silly Goose! That number isn't in the range!

Please enter a number between 1 and 200

* If the input is not a number:

I don't think that [input] is a number. Sorry

**3. End of Game**

**Output**: After 6 guesses or if the correct number is guessed:

* If the user guesses the number correctly:

Good job, [User Name]! You guessed my number in [X] guesses!

* If the user fails to guess the number within 6 attempts:

Nope. The number I was thinking of was [number]

**4. Play Again**

**Input**: The user is asked if they want to play again.

Do you want to play again?

**Output**: The game either restarts or ends based on the user's input:

* If the user inputs "yes", "y", or "Yes":
  + The game restarts.
* Any other input ends the game.

**Example Interaction**

Here's an example of what the input and output might look like during a typical game session:

1. **Introduction**:

May I ask you for your name?

John

John, we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!

1. **Guessing**:

Guess: 50

The guess of the number that you have entered is too low

Try Again!

Guess: 100

The guess of the number that you have entered is too

low

Try Again!

Guess: 150

The guess of the number that you have entered is too low

Try Again!

Guess: 190

The guess of the number that you have entered is too high

Try Again!

Guess: 170

The guess of the number that you have entered is too

low

Try Again!

Guess: 185

Nope , The Number I was thinking of was 183

1. **Play Again**:

Do you want to play again?

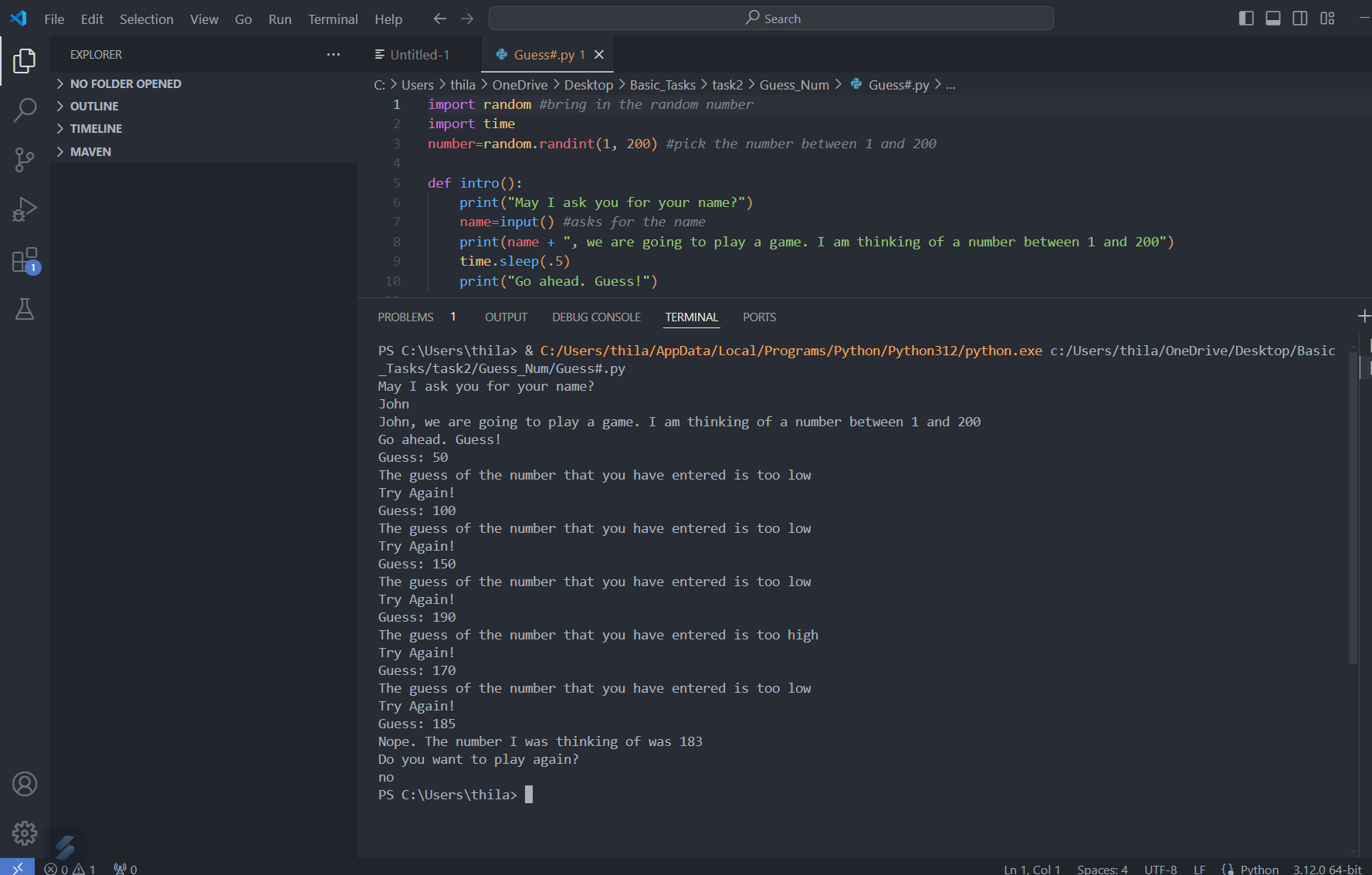
Yes

May I ask you for your name?

John

John, we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!



**Error Scenarios**

1. **Non-number Input**:

Guess: abc

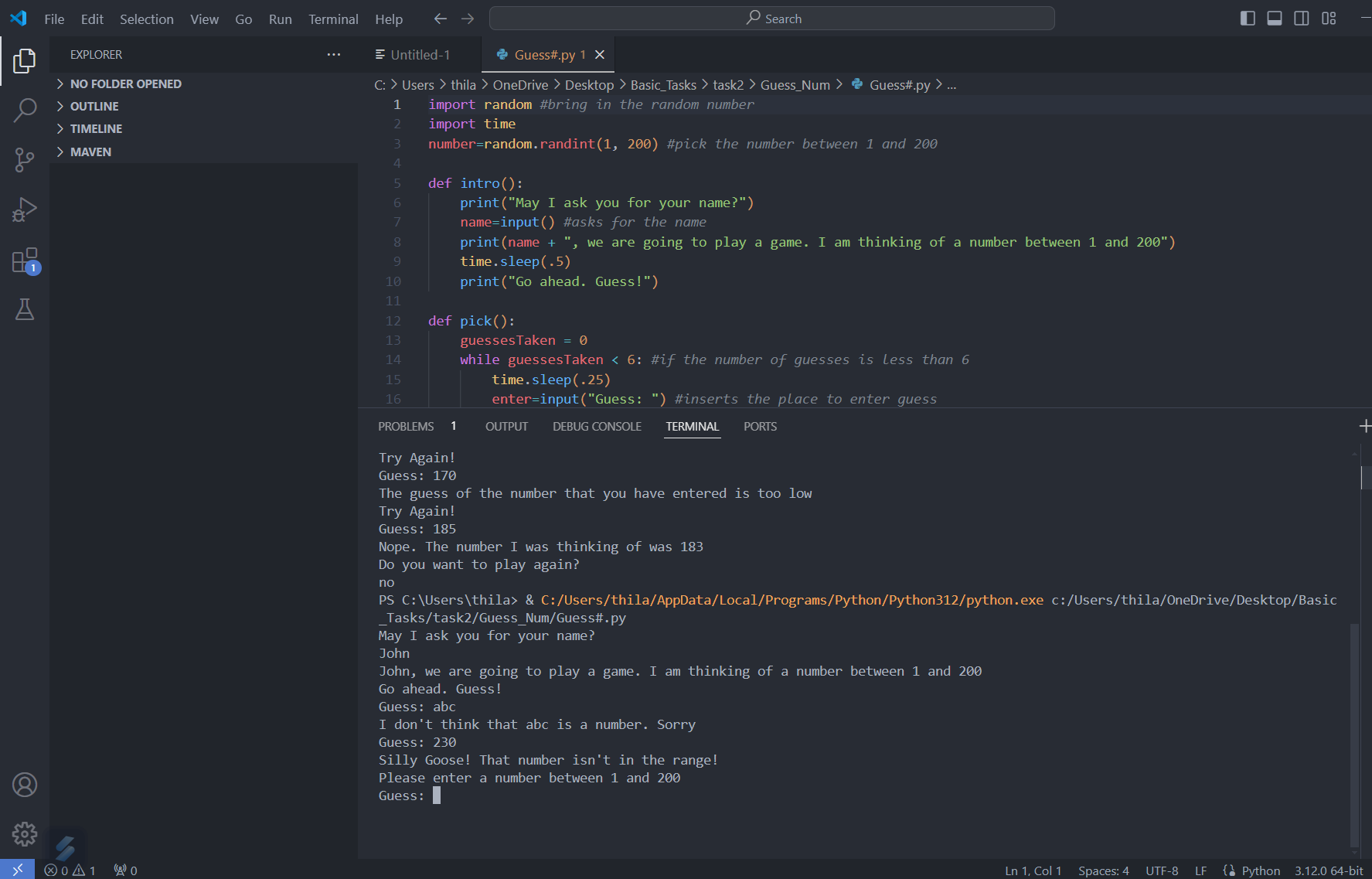
I don't think that abc is a number. Sorry

1. **Out of Range Input**:

Guess: 250

Silly Goose! That number isn't in the range!

Please enter a number between 1 and 200



By addressing these inputs and outputs, you can see the flow of the game and how the user interacts with it at each step.

### KEY FUNCTIONS OF CODE:

Key Functions

intro():

Purpose: Introduce the game to the player and prompt for their name.

Behavior:

Asks the player for their name and stores it in a variable.

Displays a message explaining the game and instructs the player to start guessing.

Parameters: None

Returns: None

pick():

Purpose: Handle the main gameplay loop where the player guesses the number.

Behavior:

Initializes a variable guessesTaken to count the number of guesses.

Runs a loop that allows the player to guess up to 6 times.

Checks if the input is a valid number within the range 1 to 200.

Provides feedback if the guess is too low, too high, or correct.

Handles invalid inputs by prompting the user to enter a valid number.

If the player guesses the correct number, it exits the loop and congratulates the player.

If the player does not guess the correct number within 6 attempts, it reveals the number.

Parameters: None

Returns: None

Main Program Loop

Purpose: Control the flow of the game, allowing the player to play multiple rounds.

Behavior:

Initializes the variable playagain to "yes".

Enters a loop that continues as long as the player wants to play again.

Calls the intro() function to start a new game.

Calls the pick() function to handle the guessing game.

Asks the player if they want to play again and stores the response in playagain.

Example Usage

Introduction:

Calls the intro() function.

Asks for the player's name and explains the game.

Guessing Game:

Calls the pick() function.

The player enters guesses, and the function provides feedback.

If the player guesses correctly within 6 tries, they are congratulated.

If the player does not guess correctly, the correct number is revealed.

Play Again:

After the game ends, asks the player if they want to play again.

If the player responds with "yes" or "y", the game restarts.

If the player responds with anything else, the game ends.

### ERRORS AND DRAWBACKS OF CODE:

1. **Global Variable Usage**:
   * **Error/Drawback**: The variable number is declared globally and could lead to unintended side effects if accessed or modified elsewhere in the code.
2. **Name Scope Issue**:
   * **Error/Drawback**: The variable name is defined in the intro() function but is used in the pick() function without being passed explicitly, which can lead to a NameError if name is not defined in the expected scope.
3. **Magic Numbers**:
   * **Error/Drawback**: The number of allowed guesses (6) and the range (1-200) are hard-coded, making the code less flexible and harder to maintain or modify.
4. **General Exception Handling**:
   * **Error/Drawback**: The except block catches all exceptions, which is not recommended because it can suppress unexpected errors and make debugging difficult. Specific exceptions should be caught instead.
5. **Repeated Code for Play Again**:
   * **Error/Drawback**: The expression playagain=="yes" or playagain=="y" or playagain=="Yes" is repeated, making the code unnecessarily verbose and prone to errors.
6. **Unnecessary time.sleep Calls**:
   * **Error/Drawback**: Excessive use of time.sleep can make the program feel slow and unresponsive, especially in a game where quick feedback is desired.
7. **Potential Infinite Loop**:
   * **Error/Drawback**: The main loop could potentially run indefinitely if the play again input is not handled correctly or if an invalid input is given.

**Recommendations for Fixes**

* **Encapsulate Variables**: Encapsulate the number variable within a function or class to avoid global state issues.
* **Pass Variables**: Ensure that variables such as name are passed explicitly to functions that need them.
* **Use Constants**: Define configurable parameters like the number of guesses and the range as constants.
* **Specific Exceptions**: Use specific exceptions like ValueError to handle known error conditions.
* **Simplify Logic**: Simplify conditional checks and avoid repetitive code.
* **Reduce time.sleep**: Minimize the use of time.sleep to improve user experience.

These are the key errors and drawbacks in the current code that need to be addressed to improve its efficiency and maintainability.

### COMPLETE GUESS NUMBER (UPDATED)CODE:

import random

import time

def generate\_number():

    return random.randint(1, 200)

def intro():

    print("May I ask you for your name?")

    name = input()  *# asks for the name*

    print(f"{name}, we are going to play a game. I am thinking of a number between 1 and 200.")

    time.sleep(0.5)

    print("Go ahead. Guess!")

    return name

def get\_guess():

    while True:

        try:

            guess = int(input("Guess: "))

            if 1 <= guess <= 200:

                return guess

            else:

                print("Silly Goose! That number isn't in the range! Please enter a number between 1 and 200.")

        except ValueError:

            print("I don't think that is a number. Sorry.")

def pick(*number*, *name*):

    guesses\_taken = 0

    while guesses\_taken < 6:

        guess = get\_guess()

        guesses\_taken += 1

        if guess < *number*:

            print("The guess of the number that you have entered is too low.")

        elif guess > *number*:

            print("The guess of the number that you have entered is too high.")

        else:

            print(f"Good job, {*name*}! You guessed my number in {guesses\_taken} guesses!")

            return

        if guesses\_taken < 6:

            print("Try Again!")

    print(f'Nope. The number I was thinking of was {*number*}.')

def play\_game():

    play\_again = "yes"

    while play\_again.lower() in ["yes", "y"]:

        number = generate\_number()

        name = intro()

        pick(number, name)

        print("Do you want to play again?")

        play\_again = input().strip()

if \_\_name\_\_ == "\_\_main\_\_":

    play\_game()

### INPUT/OUTPUT SCREENSHOTS

### (UPDATED CODE)

**Inputs and Outputs**

**1. Introduction Phase**

**Input**: The user is asked for their name.

May I ask you for your name?

**Output**: The user's name is used to personalize the game.

[User Name], we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!

**2. Guessing Phase**

**Input**: The user is prompted to guess a number between 1 and 200.

Guess:

**Output**: Depending on the guess, the program provides feedback:

* If the guess is too low:

The guess of the number that you have entered is too low

Try Again!

* If the guess is too high:

The guess of the number that you have entered is too high

Try Again!

* If the guess is not in the range:

Silly Goose! That number isn't in the range!

Please enter a number between 1 and 200

* If the input is not a number:

I don't think that [input] is a number. Sorry

**3. End of Game**

**Output**: After 6 guesses or if the correct number is guessed:

* If the user guesses the number correctly:

Good job, [User Name]! You guessed my number in [X] guesses!

* If the user fails to guess the number within 6 attempts:

Nope. The number I was thinking of was [number]

**4. Play Again**

**Input**: The user is asked if they want to play again.

Do you want to play again?

**Output**: The game either restarts or ends based on the user's input:

* If the user inputs "yes", "y", or "Yes":
  + The game restarts.
* Any other input ends the game.

**Example Interaction**

Here's an example of what the input and output might look like during a typical game session:

1. **Introduction**:

May I ask you for your name?

John

John, we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!

1. **Guessing**:

Guess: 50

The guess of the number that you have entered is too low

Try Again!

Guess: 100

The guess of the number that you have entered is too

low

Try Again!

Guess: 150

The guess of the number that you have entered is too low

Try Again!

Guess: 190

The guess of the number that you have entered is too high

Try Again!

Guess: 170

The guess of the number that you have entered is too

low

Try Again!

Guess: 185

Nope , The Number I was thinking of was 183

1. **Play Again**:

Do you want to play again?

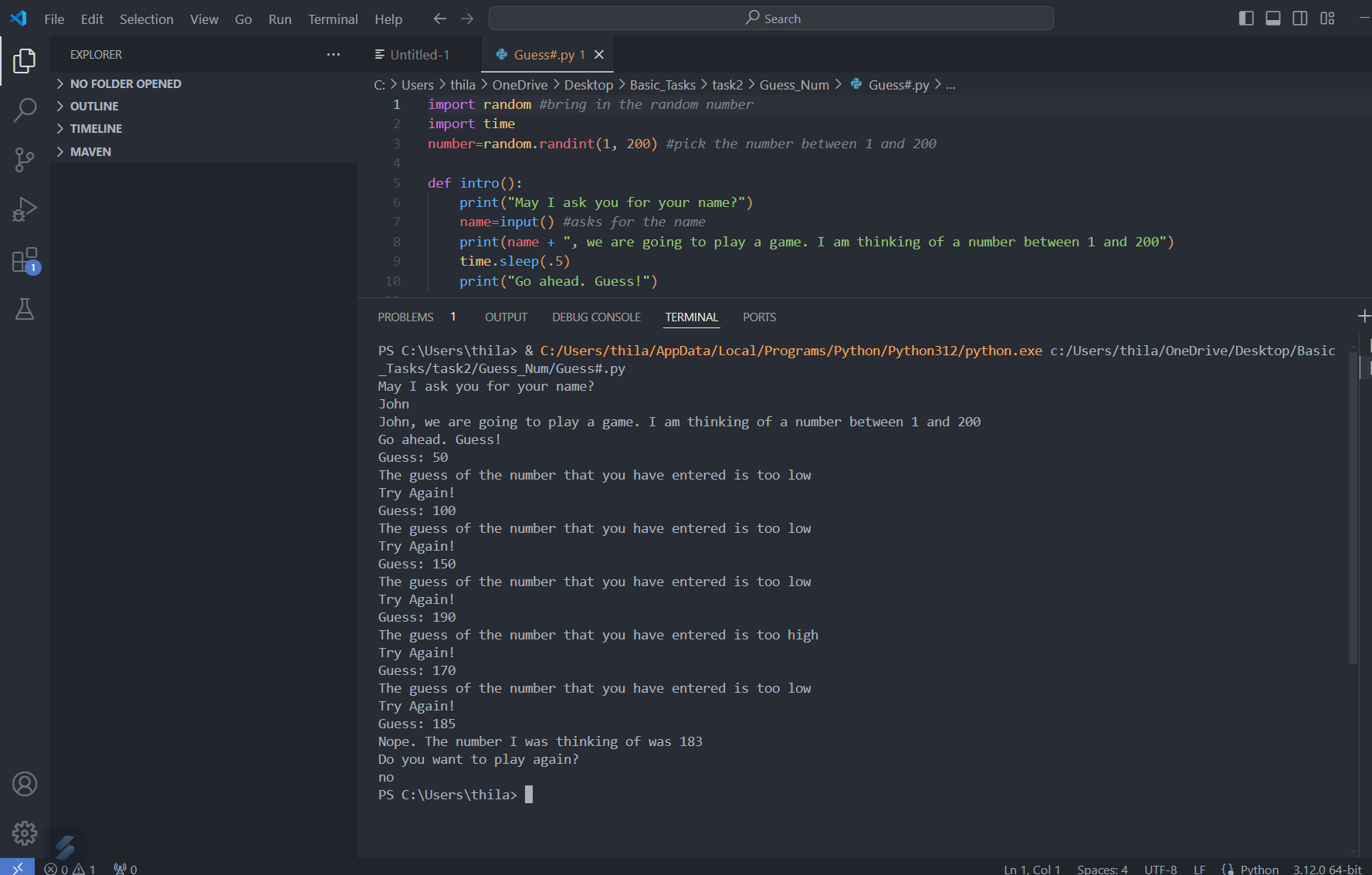
Yes

May I ask you for your name?

John

John, we are going to play a game. I am thinking of a number between 1 and 200

Go ahead. Guess!



**Error Scenarios**

1. **Non-number Input**:

Guess: abc

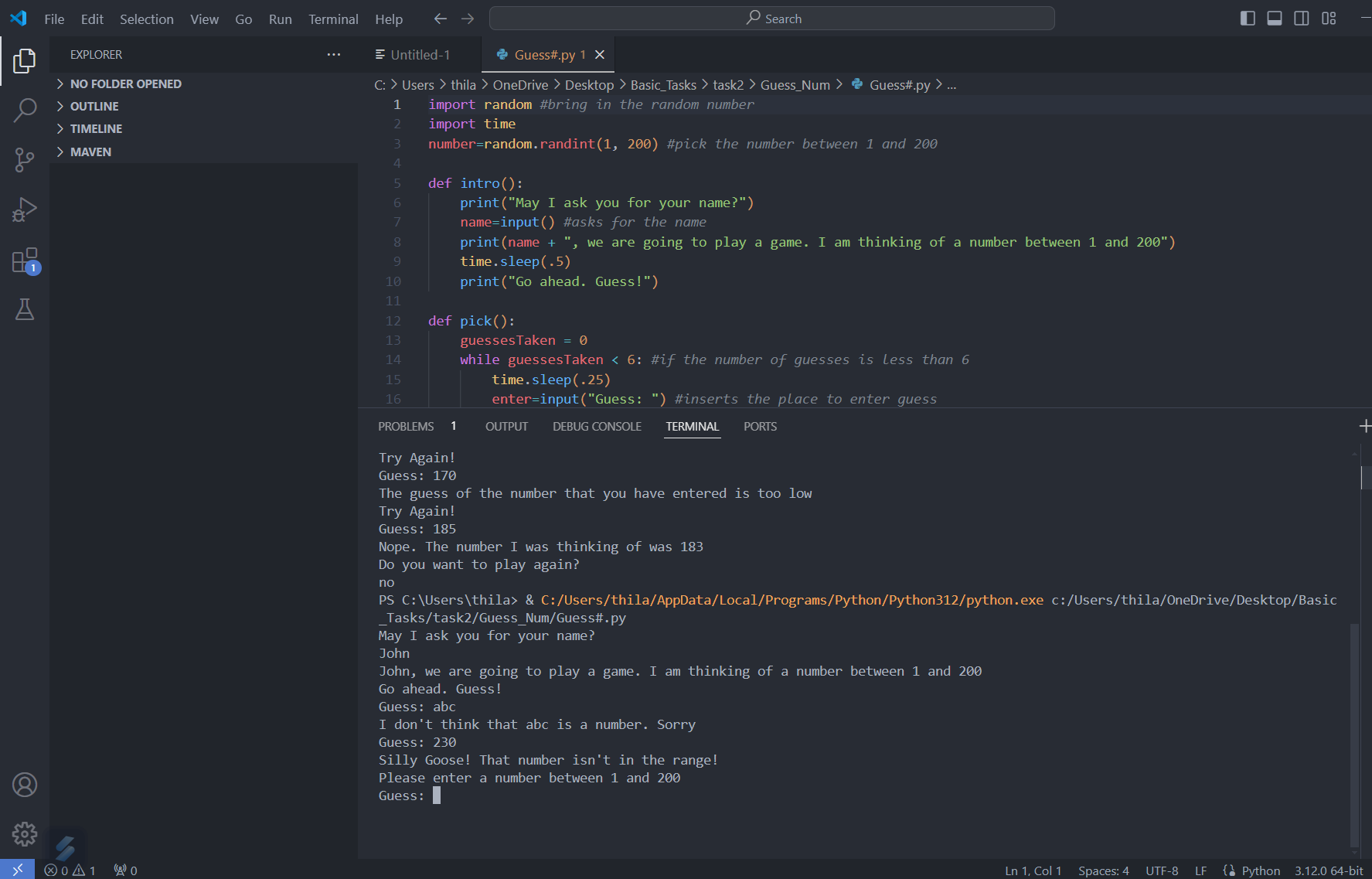
I don't think that abc is a number. Sorry

1. **Out of Range Input**:

Guess: 250

Silly Goose! That number isn't in the range!

Please enter a number between 1 and 200



By addressing these inputs and outputs, you can see the flow of the game and how the user interacts with it at each step.

### CONCLUSION

The original version of the script (Guess#.py) has several drawbacks which have been addressed in the updated version (Guess#Updated.py). One of the major issues in the original code is the handling of user input and validation. The original script lacks modular functions for generating guesses and validating inputs, leading to repetitive and error-prone code. This has been improved in the updated version by introducing a dedicated get\_guess function that ensures the input is a valid number within the specified range before proceeding. This reduces redundancy and improves code readability and maintainability.

Additionally, the original script uses a global variable for storing the player's name and relies heavily on hard-coded strings for user interaction. The updated version adopts a more structured approach by encapsulating the game logic within functions, which include generate\_number, intro, and pick, and uses formatted strings for better clarity and flexibility. The main game loop is also more robust in the updated script, with clearer and more concise control flow, making the program easier to understand and extend. Overall, the refactor enhances the program’s structure, error handling, and user experience. ​