Chapter 8
Number Guessing Game

8.1 Introduction

In this chapter we are going to bring everything we have learned so far together to create a simple number guessing game.

This will involve creating a new Python program, handling user input, using the `if` statement as well as using looping constructs.

We will also, use an additional library or module, that is not by default available by default to your program; this will be the random number generator module.

8.2 Setting Up the Program

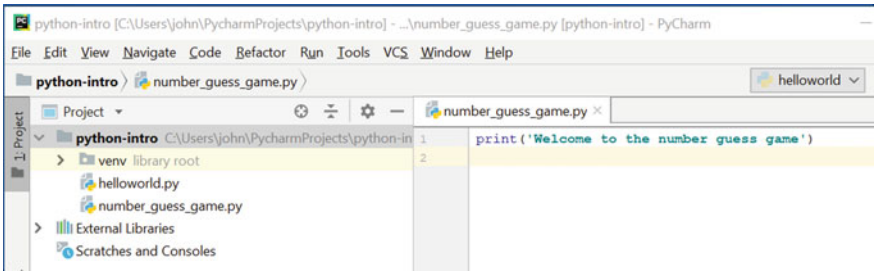
We want to make sure that we don't overwrite whatever you have done so far, and we would like this Python program to be separate from your other work. As such we will create a new Python file to hold this program. The first step will be to create a new Python file. If you are using the PyCharm IDE you can do it using the New>PythonFile menu option.

8.2.1 Add a Welcome Message

To make sure everything is working correctly we will add a simple `print()` statement to the file so that we can test out running it.

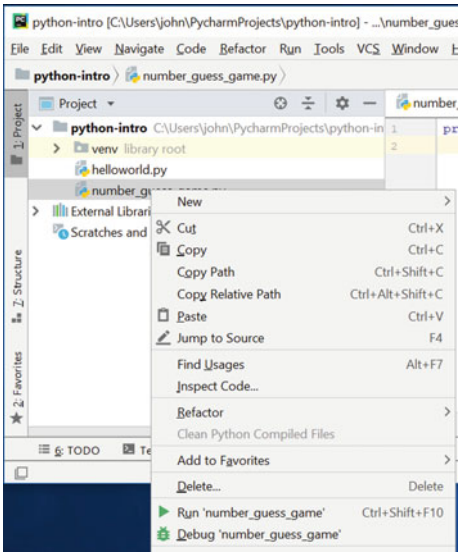
You can add anything you like, including printing out 'Hello World', however the example shown below shows a welcome message that can be used at the start of the game in the PyCharm IDE:

J. Hunt, *A Beginners Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-20290-3_8

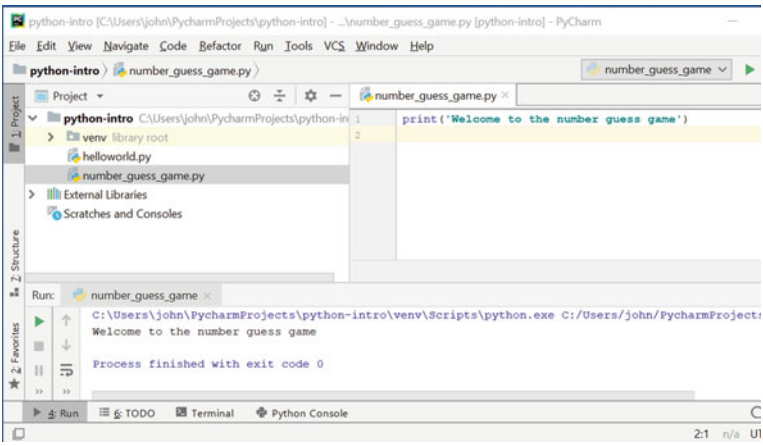


8.2.2 *Running the Program*

We can now run our embryonic program. If you are using PyCharm then to do this we can select the file in the left-hand view and use the right mouse menu to select the 'Run' option:



When we do this the Python console will be opened at the bottom of the IDE and the output displayed:

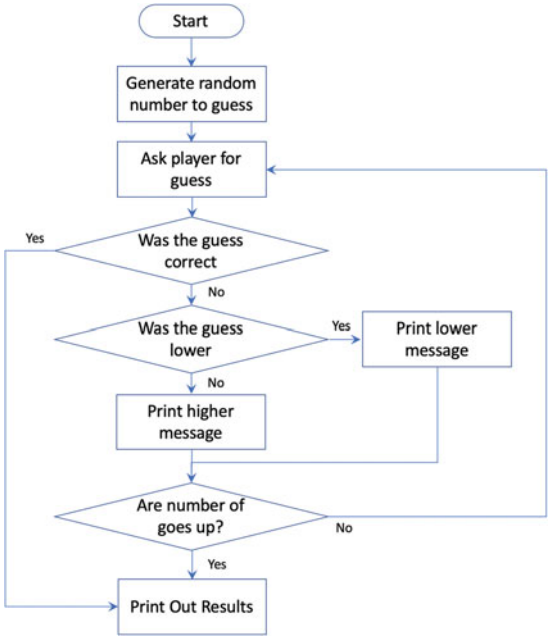


From now on you can rerun the `number_guess_game` program merely by clicking on the little green arrow at the top right-hand side of PyCharm (it allows you to rerun the last program that PyCharm executed).

8.3 What Will the Program Do?

The aim of our number guess game is to guess the number that the program has come up with.

The main flow of the program is shown in the following diagram:



Essentially the program logic is

- The program randomly selects a number between 1 and 10.
- It will then ask the player to enter their guess.
- It will then check to see if that number is the same as the one the computer randomly generated; if it is then the player has won.
- If the player's guess is not the same, then it will check to see if the number is higher or lower than the guess and tell the player.
- The player will have 4 goes to guess the number correctly; if they don't guess the number within this number of attempts, then they will be informed that they have lost the game and will be told what the actual number was.

8.4 Creating the Game

8.4.1 Generate the Random Number

We will start off by looking at how we can generate a random number. Up to this point we have only used the built-in functions that are provided by Python automatically. In actual fact Python comes with very many modules provided by the Python organisation itself, by third party vendors and by the open source (typically free) software community.

The Python `random` module or library is one that is provided with Python as part of the default environment; but the functions within it are not automatically loaded or made available to the programmer. This is partly because there are so many facilities available with Python that it could become overwhelming; so for the most part Python only makes available by default the most commonly used facilities. Programmers can then explicitly specify when they want to use some facilities from one of the other libraries or modules.

The `random` module provides implementations of pseudo-random number generators for use in application programs. These random number generators are referred to as *pseudo* because it is very hard for a computer to truly generate a series of random numbers; instead it does its best to mimic this using an algorithm; which by its very nature will be based on some logic which will mean that it is not impossible to predict the next number—hence it is not truly random. For our purposes this is fine but there are applications, such as security and encryption, where this can be a real problem.

To access the `random` module in Python you need to *import* it; this makes the module visible in the rest of the Python file (in our case to our program). This is done using

```
import random
number_to_guess = random.randint(1,10)
```

Once we have imported it, we can use the functions within this module, such as `randint`. This function returns a random integer between the first and second arguments. In the above example it means that the random number generated will be between 1 and 10 inclusive.

The variable `number_to_guess` will now hold an integer which the player of the game must guess.

8.4.2 Obtain an Input from the User

We now need to obtain input from the user representing their guess. We have already seen how to do this in previous chapters; we can use the `input()` function which returns a string and then the integer function that will convert that string into an integer (we will ignore error checking to make sure they have typed in a number at this point). We can therefore write:

```
guess = int(input('Please guess a number between 1 and 10: '))
```

This will store the number they guessed into the variable `guess`.

8.4.3 Check to See If the Player Has Guessed the Number

We now need to check to see whether the player has guessed the correct number.

We could use an `if` statement for this, however we are going to want to repeat this test until either they have guessed correctly or until they have run out of goes.

We will therefore use a `while` loop and check to see if their guess equals the number to be guessed:

```
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
```

The loop above will be repeated if the number they entered did not match the number to be guessed.

We can therefore print a message telling the player that their guess was wrong:

```
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
    print('Sorry wrong number')
```

Now we need the player to make another guess otherwise the program will never terminate, so we can again prompt them to enter a number:

```
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
    print('Sorry wrong number')
    # TBD ...
    guess = int(input('Please guess again: '))
```

8.4.4 Check They Haven't Exceeded Their Maximum Number of Guess

We also said above that the player can't play forever; they have to guess the correct number within 4 goes. We therefore need to add some logic which will stop the game once they exceed this number.

We will therefore need a variable to keep track of the number of attempts they have made.

We should call this variable something meaningful so that we know what it represents, for example we could call it `count_number_of_tries` and initialise it with the value 1:

```
count_number_of_tries = 1
```

This needs to happen before we enter the `while` loop.

Inside the `while` loop we need to do two things

- check to see if the number of tries has been exceeded,
- increment the number of tries if they are still allowed to play the game.

We will use an `if` statement to check to see if the number of tries has been met; if it has we want to terminate the loop; the easiest way to this is via a `break` statement.

```
if count_number_of_tries == 4:
    break
```

If we don't break out of the loop we can increment the count using the `'+='` operator. We thus now have:

```
count_number_of_tries = 1
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
    print('Sorry wrong number')
    if count_number_of_tries == 4:
        break
    # TBD ...
    guess = int(input('Please guess again: '))
    count_number_of_tries += 1
```

8.4.5 Notify the Player Whether Higher or Lower

We also said at the beginning that to make it easier for the player to guess the number; we should indicate whether their guess was higher or lower than the actual number. To do this we can again use the `if` statement; if the guess is lower we print one message but if it was higher we print another.

At this point we have a choice regarding whether to have a separate `if` statement to that used to decide if the maximum goes has been reached or to extend that one with an `elif`. Each approach can work but the latter indicates that these conditions are all related so that is the one we will use.

The `while` loop now looks like:

```
count_number_of_tries = 1
guess = int(input('Please guess a number between 1 and 10:
'))
while number_to_guess != guess:
    print('Sorry wrong number')
    if count_number_of_tries == 4:
        break
    elif guess < number_to_guess:
        print('Your guess was lower than the number')
    else:
        print('Your guess was higher than the number')
    guess = int(input('Please guess again: '))
    count_number_of_tries += 1
```

Notice that the `if` statement has a final `else` which indicates that the guess was higher; this is fine as by this point it is the only option left.

8.4.6 End of Game Status

We have now covered all the situations that can occur while the game is being played; all that is left for us to do is to handle the end of game messages.

If the player has guessed the number correctly, we want to congratulate them; if they did not guess the number, we want to let them know what the actual number was. We will do this using another `if` statement which checks to see if the player guessed the number of not. After this we will print an end of game message:

```
if number_to_guess == guess:
    print('Well done you won!')
    print('You took', count_number_of_tries ,
          'goes to complete the game')
else:
    print('Sorry - you loose')
    print('The number you needed to guess was',
          number_to_guess)
print('Game Over')
```

8.5 The Complete Listing

For ease of reference the complete listing is provided below:

```
import random

print('Welcome to the number guess game')

# Initialise the number to be guessed
number_to_guess = random.randint(1,10)

# Initialise the number of tries the player has made
count_number_of_tries = 1

# Obtain their initial guess
guess = int(input('Please guess a number between 1 and 10: '))
while number_to_guess != guess:
    print('Sorry wrong number')

    # Check to see they have not exceeded the maximum
    # number of attempts if so break out of loop otherwise
```

```
    # give the user come feedback
    if count_number_of_tries == 4:
        break
    elif guess < number_to_guess:
        print('Your guess was lower than the number')
    else:
        print('Your guess was higher than the number')

    # Obtain their next guess and increment number of attempts
    guess = int(input('Please guess again: '))
    count_number_of_tries += 1

# Check to see if they did guess the correct number
if number_to_guess == guess:
    print('Well done you won!')
    print('You took', count_number_of_tries , 'goes to complete
the game')
else:
    print("Sorry - you loose")
    print('The number you needed to guess was',
          number_to_guess)

print('Game Over')
```

And a sample run of the program is shown here:

```
Welcome to the number guess game
Please guess a number between 1 and 10: 5
Sorry wrong number
Your guess was higher than the number
Please guess again: 3
Sorry wrong number
Your guess was lower than the number
Please guess again: 4
Well done you won!
You took 3 goes to complete the game
Game Over
```

8.6 Hints

8.6.1 Initialising Variables

In Python it is not necessary to declare a variable before you assign to it; however, it is necessary to give it an initial value before you reference it. Here *reference it* refers to obtaining the value held by a variable. For example

```
count = count + 1
```

what this says is obtain the value held by `count`, add 1 to it and then store the new value back into `count`.

If `count` does not have a value before you try to do this then you are trying to get hold of nothing and add 1 to it; which you can't do and thus an error will be generated in your program, such as:

```
NameError: name 'count_number_of_tries' is not defined
```

This is also true for

```
count += 1
```

Remember this is just a shorthand form of `count = count + 1` and thus still relies on `count` having a value before this statement.

This is why we needed to initialise the `count_number_of_tries` variable before we used it.

8.6.2 Blank Lines Within a Block of Code

You may have noticed that we have used blank lines to group together certain lines of code in this example. This is intended to make it easier to read the code and are perfectly allowable in Python. Indeed, the Python layout guidelines encourage it.

It is also possible to have a blank line within an indented block of code; the block of statements is not terminated until another line with valid Python on it is encountered which is not indented to the same level.

8.7 Exercises

For this chapter the exercises all relate to adding additional features to the game:

1. Provide a cheat mode, for example if the user enters `-1` print out the number they need to guess and then loop again. This does not count as one of their goes.
2. If their guess is within `1` of the actual number tell the player this.
3. At the end of the game, before printing 'Game Over', modify your program so that it asks the user if they want to play again; if they say yes then restart the whole thing.