**Games.py – part 1**

We have written a lot of relatively short programs over the past couple of weeks, usually with just 1 function. But programs in the real world are generally not so short. They’re composed of multiple functions often spread across multiple files. Therefore it might be a good idea to create a more complex program in this manner.

In this exercise I will run through the creation of an application that will allow the user to choose between two games to play. There will be a script for each game and another for the top level menu. There will be quite a lot of code to get through so be sure to type it exactly as you see it.

To start off we should concentrate on the two games we’ll include in our application, a number guessing game and rock paper scissors. We’ll start with the former. Create a new file called ‘number\_guess.py’.

Before we get started it is a good idea to understand how our number game should work. The computer will generate a number to be guessed and the player should be able to input their own number. If the player guessed incorrectly the program should tell us how close they are to the number in terms of temperature, e.g. hot, warm, cool, etc, while noting what their previous guess was. The player should have a limited number of guesses to make, say 4. To make it a bit more interesting the player will be able to choose between 3 difficulty levels, easy (0-20), medium (0-50) and hard (0-100). If the player guesses correctly they will be told how many guesses it took them, otherwise failure will present the player with the correct number. This is a lot more complex than the example we covered earlier on.

As usual, we will create a skeleton to start off our number game.

import random

def clear\_screen():

def difficulty\_menu\_ui():

def difficulty():

def guess\_the\_number():

We import the ‘random’ module since we know we will want to generate a random number for the player to guess.

‘clear\_screen()’ will be a small utility to keep the screen clear between guesses. This is relatively optional but I feel it’s worth having.

‘difficulty\_menu\_ui()’ will simply handle outputting our menu displaying the difficulty options available to the player. It is a good idea to separate code for logic and user interface.

‘difficulty()’ will handle the logic for difficulty selection.

‘guess\_the\_number()’ is the game itself using the above functions. This one will be the longest function.

We’ll get the ‘clear\_screen()’ function out of the way first and it’s quite simple.

for x in range(50):

print("")

It simply prints out 50 blank lines. You may want to add more or less lines depending on your own environment.

Next we’ll tackle the first part of the game: difficulty selection. As you can see above there is a function to output the menu and another to handle the logic behind it. We’ll start with the menu, so underneath ‘difficulty\_menu\_ui()’ add the following.

return '''

----------------------------------

Please select a difficulty

Enter a number to select an option

----------------------------------

1 - Easy

2 - Medium

3 - Hard

----------------------------------

'''

This is something new. This is known as a string literal and allows us to print out text exactly as it is laid out in the code, making it perfect for outputting well-formatted menus. One thing to bear in mind is that indentation is taken into account with these so to ensure proper menu placement be sure to remove all indents at the start of each line.

With that done we can now start on writing our ‘difficulty()’ logic. Inside ‘difficulty()’ type the following.

print(difficulty\_menu\_ui())

while(True):

# Handle invalid input type

try:

user\_input = int(input("-> "))

except:

print("Invalid input, please try again")

continue

First we output the menu, then we get input from the user. The method we will be using here is a little different from how we might have done it before, I will explain.

We want a number from the user. Here we have placed that statement inside a ‘try except’ block which will allow us to decide what to do should we get an error back. If ‘input()’ gets a string or a float instead of an integer it will cause an error. When this happens the ‘except’ block is triggered, printing out an error message and executing the ‘continue’ inside. What this does is start a new loop. So if the user enters anything that’s not an integer we will simply prompt the user again until we get an integer. This is a lot more robust than our previous solutions.

Once we get input we can act upon it and return the appropriate difficulty level. Therefore an ‘if’ statement is in order. Add the following inside the loop.

# If input is not 1-3 display error

if((user\_input < 1) or (user\_input > 3)):

print("Invalid input, please try again")

else:

clear\_screen()

if(user\_input == 1):

print("Easy selected\n")

return 1

elif(user\_input == 2):

print("Medium selected\n")

return 2

else:

print("Hard selected\n")

return 3

We first check if the number is out of range since we’re looking for a number between 1 and 3. If that is the case we output an error message and ask the user for input again. Otherwise we clear the screen and print out and return the appropriate difficulty level.

With that done we can begin on writing the bulk of our ‘guess\_the\_number()’ function. Within it add the following.

# Determine difficulty level

difficulty\_level = difficulty()

# The player has 3 attempts to guess the correct number

guess\_attempts = 4

# The volume of available numbers increases with difficulty

if(difficulty\_level == 1):

random\_number = random.randint(0, 20)

limit = 20

elif(difficulty\_level == 2):

random\_number = random.randint(0, 50)

limit = 50

else:

random\_number = random.randint(0, 100)

limit = 100

The first thing we do is get our difficulty level by calling our ‘difficulty()’ function, then we set ‘guess\_attempts’ to 4. You can give the player more or less guesses by changing this value. We set another variable called ‘limit’ which will help when we need to check whether the player’s input is in the correct range or not, since there are multiple ranges according to each difficulty.

Determining the number to be guessed follows. The range of numbers to choose from increases along with the difficulty.

After this there are two major operations to work on, the next being player input. We will handle this in a manner similar to before with a few changes. Type the following code.

while(guess\_attempts > 0):

# Ask the player to guess a number

print("Guesses remaining: {}".format(guess\_attempts))

# Handle invalid input type

try:

player\_guess = int(input("Please guess a number\n-> "))

except:

print("Invalid input, please try again")

continue

We only want to loop so long as the player has guesses left to make, but if invalid input is given it will continue to ask the player until we get valid input. We only reduce the number of guesses if the player has made a valid guess.

Now we can concentrate on the last major operation which is our win condition. There are several parts to this wrapped up inside an ‘if’ statement. First we have to check that the player’s input is in range or not. Type the following code after the ‘try except’ block but within the loop.

# Check if player guessed correctly

if(not(0 <= player\_guess <= limit)):

print("That number is out of range, please try again")

continue

Here we say ‘if the players guess is not in range tell the player it was out of range and try again’. It’s a good idea to ensure the player’s guess is in range before doing anything else. After that we can check to see if the player guesses correctly.

# Player guess correctly

elif(player\_guess == random\_number):

print("Well done, {} was the correct number!".format(random\_number))

print("You had {} guesses remaining.".format(guess\_attempts))

return 0

If the player’s guess matches the generated number we tell the player that they were correct and how many guesses they had left. Then we say ‘return 0’ which will exit the game and send the player back to the top menu which we have yet to make. When we test the game once we’ve finished writing it this statement will simply exit the script.

If the player didn’t guess correctly we give the player an indication of how close they are to the right answer. We’ll use a nested ‘if’ statement to do this and we’ll need to do it twice since the player’s guess can be larger or smaller than the right answer.

# Player guess is larger than the number

elif(player\_guess > random\_number):

clear\_screen()

if((player\_guess - random\_number) <= 5):

print("Hot\n")

elif(5 < (player\_guess - random\_number) <= 15):

print("Warm\n")

elif(15 < (player\_guess - random\_number) <= 25):

print("Cool\n")

else:

print("Cold\n")

# Player guess is smaller than the number

else:

clear\_screen()

if((random\_number - player\_guess) <= 5):

print("Hot\n")

elif(5 < (random\_number - player\_guess) <= 15):

print("Warm\n")

elif(15 < (random\_number - player\_guess) <= 25):

print("Cool\n")

else:

print("Cold\n")

If the player’s guess is larger than the right answer, then we check to see the difference between the two. We do the same if the player’s guess is smaller but we swap the subtraction around. We clear the screen before outputting the indication for neatness.

After printing out the indication of how close the player is we can deduct a guess and output the previous guess. This goes right after the above ‘if’ statement.

guess\_attempts -= 1

# print out previous guess for reference

print("Previous guess: {}".format(player\_guess))

If the player exhausts their guesses before guessing the correct answer then the loop will end and the player will be told what the correct answer was. Then we can return to the top menu once we write it. This code follows the loop.

# The player runs out of guesses

print("Unfortunately you ran out of guesses")

print("The number was {}".format(random\_number))

return 0

We can test that this works correctly by calling the ‘guess\_the\_number()’ function. Outside the functions (at the bottom with no indentation) type the following.

guess\_the\_number()

Test it using various input to ensure that all possible conditions are handled appropriately. Hopefully it works just fine. Next we will build the second game in our program, rock paper scissors, and this is laid out in part 2.