Making a Guess the Number Game (Solutions)

In this section, you're going to make a Guess the Number game. The computer will think of a secret number from 1 to 50 and ask the user to guess it. After each guess, the computer will tell the user whether the number is too high or too low. The user wins if they can guess the number within six tries.

This is a good game to code because it covers many programming concepts in a short program. You'll learn how to convert values to different data types and when you would need to do this. Since this program is a game, from now on we'll call the user the player.

Once you are finished with this module you will have a working interactive game. Here's what the Guess the Number program looks like to the player when it's run. The player's input is marked in bold.

Hello! What is your name?
Albert
Well, Albert, I am thinking of a number between 1 and 20.
Take a guess.
10
Your guess is too high.
Take a guess.
2
Your guess is too low.
Take a guess.
4
Good job, Albert! You guessed my number in 3 guesses!

1. Building the first interaction to learn the players name

Up until now, you've probably only worked with data in files or data you've put into the code before running it. Now we are going to recieve data from the user while the program is running.

The input() function pauses the program until something is typed in. Let's start by saving some user input into a variable and then printing it back to them.

In [ ]:
```
print("What is your favorite number? (Press enter after you type a number)")
their_fav_number = input()
print("Oh really!? I love the number "+their_fav_number + "!" )
```

Notice the timing when the code above is run. The final print out of a statement doesn't appear until you have pressed enter on the input. This is a great example showing the order

of operation and flow of a program. This is an important aspect of programming and it is good to start thinking about how programs flow now.

Next up is making a function that will give us back someone's name that we can reuse later.

In [ ]:
```
def get_players_name_polite():
    print("Hello, what is your name?")
    player_name = input()
    print("It is nice to meet you " + player_name + ".")
    return player_name

get_players_name_polite()
```

Notice how the print statements control the flow of the program. When writing code that has live interaction with users it is important to make sure you give them clear instruction.

Question 1.1 Fill in the function that sounds more like you. How would you ask someone their name? How would you respond.

An example might be (user's response in bold):

Heya! What's your name?
Maria
Nice to make your acquaintance Maria!

In [ ]:
```
# Potential Solution

def get_players_name_my_way():
    print('Hi! What"s your name?')
    player_name = input()
    print('Nice to meet you ' + player_name + '!')
    return player_name

get_players_name_my_way()
```

2. Picking a Number and Flow Control

Random Numbers

Now that you have a player's name we can get to the main part of the game and deal with picking a number and letting the player guess.

In order to do this you will need an import statement. Statements are instructions that perform some action but don't evaluate to a value like expressions do. You've already seen the assignment statement, which stores a value in a variable.

While Python includes many built-in functions, some functions are written in separate programs called *modules*. You can use these functions by importing their modules into your program with an import statement.

For this game you will need to import the random module so that the program can call the randint() function. This function will come up with a random number for the player to guess.

Try running the code below multiple times to see the randomness in action!

In [ ]:
```
import random
random.randint(1,4)
```

For Loop

In previous labs and projects, the program execution started at the top instruction in the program and moved straight down, executing each instruction in order. But with the *for, if, else*, and *break* statements, you can make the execution loop or skip instructions based on conditions. These kinds of statements are **flow control statements**, since they change the flow of the program execution as it moves around your program.

Using Loops to Repeat Code Below is a for statement, which indicates the beginning of a for loop:

```
for step in range(6):
```

Loops let you execute code over and over again. The example here will repeat its code six times. A for statement begins with the `for` keyword, followed by a new variable name (called `step` here), the `in` keyword, a call to the `range()` function that specifies the number of loops it should do, and a colon.

Try running the following and then we'll add a few additional concepts so that you can work with loops.

In [ ]:
```
for step in range(6):
    print("The value of step is: "+ str(step))
    rand_number = random.randint(1,6)
    print("and here is a random number "+ str(rand_number))
```

Now after running this, there are three important pieces to point out.

1. Notice that step starts at 0. Each step is called one interation
2. The code that is repeated is indented compared to the for statement
3. We used random this time without an import statement. You only need to run import once and you can keep using functions from that module!

You will get lots of practice counting from 0, but at first this can feel very weird. It will get easier, just keep practicing.

For the indentation, this is called a *Block* of code.

Code Blocks

Several lines of code can be grouped together in a block. Every line in a block of code

begins with at least the number of spaces as the first line in the block. You can tell where a block begins and ends by looking at the number of spaces at the front of the lines. This is the line's indentation.

Python programmers typically use four additional spaces of indentation to begin a block. Any following line that's indented by that same amount is part of the block. The block ends when there's a line of code with the same indentation as before the block started. There can also be blocks within other blocks.

For example, the following block of code runs 5 times and prints out the even numbers from 2 to 10. Run this block for yourself to see what the output is.

In [ ]:
```
x = 0
for step in range(5):
    x = x + 2
    print(x)
```

Decisions with if, elif, and else

These let you make decisions based on values or variables in your program. See below an example using random numbers.

In [ ]:
```
first_number = random.randint(1,10)
second_number = random.randint(1,10)

if first_number < second_number:
    print("The first number is smaller.")
elif second_number < first_number:
    print("The second number is smaller.")
else:
    print("The numbers are equal.")

print("The first number is: "+ str(first_number))
print("The second number is: " +str(second_number))
```

The `elif` statement stands for else if which helps you make more than just two cases with if and else.

Question 2.1 Fill in the if, elif, else statement like the previous code, but check for and print which number is larger rather than smaller.

In [ ]:
```
# Potential Solution

first_number = random.randint(1,10)
second_number = random.randint(1,10)

if first_number > second_number:
    print('The first number is larger')
elif second_number > first_number:
    print('The second number is larger')
else:
    print("The numbers are equal.")
```

```
print("The first number is: " + str(first_number))
print("The second number is: " + str(second_number))
```

When we are making comparisons between two values or variables we have a few options on what to use:

Operator	Operation
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

The Break Statement

There are times when we may want a for loop to stop and not perform anymore iterations. This is where the *break* statement comes to our aid. To do this we usually need a decision and so we will use an if statement

In [ ]:
```
print("Please enter a number less than ten for me to count up to.")
max_number = input()
max_number = int(max_number)
print("Thank you, I'll start counting from 0")
for step in range(11):
    print(str(step))
    if(step==max_number):
        break
```

Note that the `input()` function takes in a string, so the line `max_number = int(max_number)` above converts that string into an integer number. This is all you need to know to complete the rest of this exercise, but for a more in depth overview of data types and converting between them in Python see Basic Data Types in Python (e.g. integer, float, complex, string, boolean).

3. Putting it together to make the game!

First let's lay out the flow of the game.

1. Learn the player's name
2. Setup for the game (pick a number)
3. Let the player guess up to 6 times
4. Finish the game with a farewell

The code below is incomplete. Fill in code wherever you see an ...

In [ ]:

```
# Potential Solution

# First part
# Players Name
# Use one of the functions made in section 1 for introductions
player_name = get_players_name_polite()

# Second part
# Setup
# Start by telling them the idea of the game
print("Ok, "+ player_name + " I'm thinking of a number between 1 and 50.")

# Have the computer pick a random number between 1 and 50 using the random.rand
secret_number = random.randint(1, 50)

# Third part
# Let the players pick
# Use a for loop to have the following block happen at most 6 times
for guesses_taken in range(6):
    print("Take a guess.")
    guess = input()
    guess = int(guess)

    if guess < secret_number:
        print("Your guess is too low.")
    elif guess > secret_number:
        print("Your guess is too high")
    else:
        break

# Fourth part
if guess == secret_number:
    guesses_taken = str(guesses_taken + 1) # NOTE: Add 1 here because guesses_t
    print('Good job, ' + player_name + '! You guessed my number in ' + guesses_

if guess != secret_number:
    number = str(secret_number)
    print('Nope. The number I was thinking of was ' + number + '.')
```