Testing

by Sophia

WHAT'S COVERED

In this lesson, you will work through testing code and identify some common problems and issues. Specifically, this lesson covers:

1. Syntax Errors

It's great when all goes well, but what happens when there are issues? Before we answer that, let's explore the first line of code we wrote in the last lesson.

```
print("Hello World")
```

The command **print()** is a function that allows Python to send data to the output screen. Inside of the brackets, we have a string enclosed in double-quotes. The quotes can be single or double, but they must be the same, otherwise you will get an error.

EXAMPLE
For example, this could be:
print("Hello World")
as well as this:
print('Hello World')

TRY IT

However, try mixing the quotes and see what happens:

Directions: In the IDE, enter the following code:

```
print('Hello World")
```

Directions: Then, select the Run button at the top.

In the output window, after you selected the Run button, you received an error that looks like this:

```
 File "/home/main.py", line 1
    print('Hello World")
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

A **syntax error** means that you have violated the "grammar" rules of Python. Python does its best to point right at the line and character where it noticed it was confused. This is a good place to start looking for errors in your code. It may not always be exactly where the error is detected, but it's a good place to start. In the output, you will also see more details about which file and line the error is on. In the code editor, you should see the line numbers to the left of the code. It's important to note that these numbers are not part of the code.

The only tricky bit of syntax errors is that sometimes the mistake that needs fixing is actually earlier in the program than where Python noticed it was confused. So again, the line and character that Python indicates in a syntax error may just be a starting point for your investigation.

Python will also inform you what the potential errors are. With any error message, if you're unsure of what it means, you can always copy it into Google to learn what the potential issue(s) could be. This is a good step to follow if you're not sure about what the error represents for any programming language.

Another important thing to note is that if you see multiple errors (which is quite common in larger programs), you should fix the issues from the top down. In many instances, an issue at the top of the error list could seemingly show that there are many more errors than there really are. Don't worry about the number of errors; put your focus on the first one.

Remember we noted that we needed the quotes around the literal string. If we don't use quotes around the literal string, Python will assume that it is a variable or another keyword. We will get into that in Challenge 2.

TERMS TO KNOW

print()
The `print()` function allows Python to send data to the output screen.

Syntax Error
A syntax error means that you have violated the "grammar" rules of Python.

2. Comments

It's commonplace to see developers add in notes or comments in the code to explain what the code is doing for both themselves and for other developers who review the code later on. In order to add a comment, you would start the comment with a #. Then, Python will ignore everything else on the line.

TRY IT

Directions: In the IDE, enter the following code and run it:

```
#This code outputs Hello World 5 times
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Notice that after running the code, the code does not produce any errors, because Python simply ignores anything after the #. We could have a comment after a line as well.

EXAMPLE

```
#This code outputs Hello World 5 times
print("Hello World")
print("Hello World")
print("Hello World") #The third output line
print("Hello World")
print("Hello World")
```

Developers may also use the # to temporarily remove lines. There may be certain parts of the code that aren't functioning and need to be tested further. For example, let's add in an error with the missing quotes:

```
#This code outputs Hello World 5 times
print("Hello World")
print("Hello World")
print(Hello World)
print("Hello World")
print("Hello World")
```

If we ran this snippet of code, we receive the following error:

```
  File "/home/main.py", line 4
```

```
    print(Hello World)
          ^^^^^^^^^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

If we didn't know how to fix the issue and the line didn't affect the rest of the code, we can simply comment out the line as shown below:

TRY IT

Directions: In the IDE, enter this code and run it:

```
#This code outputs Hello World 5 times
print("Hello World")
print("Hello World")
#print(Hello World)
print("Hello World")
print("Hello World")
```

```
Hello World
Hello World
Hello World
Hello World
```

Since the third line was commented out, the line wasn't output. It's not quite obvious, so let's change the output slightly:

TRY IT

Directions: In the IDE, enter this code and run it:

```
#This code outputs Hello World 5 times
print("Hello World 1")
print("Hello World 2")
#print(Hello World 3)
print("Hello World 4")
print("Hello World 5")
```

```
Hello World 1
Hello World 2
Hello World 4
Hello World 5
```

That now makes it clear which line was commented out.

3. Best Practices With Comments

In other programming languages, you have the ability to use special types of comments that span multiple lines. However, this is not possible in Python. Rather, if you need to create multiline comments in Python, you can use a # (hashtag) for each line of the comment like this:

```
#This is a way of having
#comments that span multiple
#lines of code so that you can
#add your notes without
#scrolling on one line.
```

It's a good practice to include some details about the program in your comments. This may include who developed the program, when it was created, what the program is for, perhaps some details of how to use the program, and any additional information that could be useful for someone using the program later on—even if they don't know what the code is actually doing.

EXAMPLE

Let's take a look at the following program, which can feel a bit daunting:

```
def max_of_two( firstNumber, secondNumber ):
  if firstNumber > secondNumber:
    return firstNumber
  return secondNumber
def max_of_three( firstNumber, secondNumber, thirdNumber ):
  return max_of_two( firstNumber, max_of_two( secondNumber, thirdNumber ))
print(max_of_three(20, 30, -10))
```

In looking at the program, can you guess what it's doing? Even if you did, it probably took you a bit of time to figure it out. If you have to take that extra time each time you evaluated a program, it would probably add up quite quickly. Let's take the program and add on some of those comments as a starting point:

EXAMPLE

```
#Author: Sophia
#Created Date: August, 21, 2021
#Description: This program has two functions to find the maximum
#value of three numbers.
#Example of usage: print(max_of_three(20, 30, -10))
#Result: function returns 30

def max_of_two( firstNumber, secondNumber ):
  if firstNumber > secondNumber:
    return firstNumber
  return secondNumber
def max_of_three( firstNumber, secondNumber, thirdNumber ):
  return max_of_two( firstNumber, max_of_two( secondNumber, thirdNumber ) )
print(max_of_three(20, 30, -10))
```

That should help clarify some details as a starting point. You should know who created it so if you do have questions, you can check in with them. Knowing when it was created can also help, as code does get updated. Notice too that we have a description that explains what the code does and an example of how to use it.

Now, we do have two separate functions (`max_of_two` and `max_of three`). Don't worry if you don't know what those are yet; for now, think of them as small, reusable pieces of code. Rather than redeveloping everything from scratch each time, you can make use of these functions later on. Rather than explain what the code does directly, see if you can figure it out based on the comments that are added:

EXAMPLE

```
#Author: Sophia
#Created Date: August, 21, 2021
#Description: This program has two functions to find the maximum
#value of three numbers.
#Example of usage: print(max_of_three(20, 30, -10))
#Result: function returns 30
#Function: max_of_two
#Purpose: This function accepts two numbers, compares them,
#and returns the value that is larger.
def max_of_two( firstNumber, secondNumber ):
  if firstNumber > secondNumber:
    return firstNumber
  return secondNumber
#Function: max_of_three
#Purpose: This function accepts three numbers and sends the second and third
#numbers to max_of_two. It then takes the result of that comparison to
#send the first number and the result to get the larger value
#of all three.
def max_of_three( firstNumber, secondNumber, thirdNumber ):
  return max_of_two( firstNumber, max_of_two( secondNumber, thirdNumber ) )
#Testing the function max_of_three
print(max_of_three(20, 30, -10))
```

Even though you may not understand the logic of the code, you get the gist of what the code is meant to do.

TRY IT

Directions: In the IDE, give it a try with the following code with a test to see if it functioned as you expected it to:
Note: If you copy and paste this code into the IDE, make sure to indent the functions where they are below. For

example, `if firstNumber > secondNumer:` is indented once (typically 2 spaces), `return firstNumber` is indented twice (typically 4 spaces). The importance of indentations will be discussed in a later tutorial.

```
#Author: Sophia
#Created Date: August, 21, 2021
#Description: This program has two functions to find the maximum
#value of three numbers.
#Example of usage: print(max_of_three(20, 30, -10))
#Result: function returns 30
#Function: max_of_two
#Purpose: This function accepts two numbers, compares them,
#and returns the value that is larger.
def max_of_two( firstNumber, secondNumber ):
  if firstNumber > secondNumber:
    return firstNumber
  return secondNumber
#Function: max_of_three
#Purpose: This function accepts three numbers and sends the second and third
#numbers to max_of_two. It then takes the result of that comparison to
#send the first number and the result to get the larger value
#of all three.
def max_of_three( firstNumber, secondNumber, thirdNumber ):
  return max_of_two( firstNumber, max_of_two( secondNumber, thirdNumber ) )
#Testing the function max_of_three
print(max_of_three(20, 30, -10))
```

```
30
```

Try to change around the values that were being passed in as well. Were the results what you expected to see?

4. Edge and Corner Cases

Whenever we have a program, we want to make sure that it works correctly logically. Although we can’t assume every potential input, it’s generally a good idea to also test edge and corner cases. The **edge cases** are values that are at the end of a testing range.

Let’s take the following program that should take in a score for an exam. If the score is between 90 and 100, it should tell the student that they received an A. If the score is between 80 and 89, it should tell the student that they received a B. If the score is between 70 and 79, it should tell the student that they received a C. Any grade less than 70 should tell the student that they did not pass the exam.

As such, this was the (flawed) program that was created for this request:

EXAMPLE

```
score = int(input("What is your exam score (0-100): "))
if score > 90 and score < 100:
  print('You got an A! Congrats!')
elif score > 80 and score < 90:
  print('You got a B! Well done!')
elif score > 70 and score < 80:
  print('You got a C.')
else:
  print("You did not pass the exam.")
```

Let’s just try this program as is to see if it works. First, let’s try entering in 91.

```
What is your exam score: 91
You got an A! Congrats!
```

Look, it seems to work, right? Let’s try 92, 93 and 94.

```
What is your exam score: 92
You got an A! Congrats!

What is your exam score: 93
You got an A! Congrats!

What is your exam score: 94
You got an A! Congrats!
```

All of these seem to work as expected. However, all of these tests are interior tests. The **interior tests** are test values that are within a specific range where the precise values that we entered within that range did not matter. In our case, we tested 92, 93 and 94, which fall in the same range. Although it is important to have a test of the interior tests, we should also be testing edge cases, or the values at the beginning or the end of the range, to ensure that the program works. Let’s try to enter in 100, as that is at the end of the first range, and see what happens.

```
What is your exam score: 100
You did not pass the exam.
```

Oops! You’d expect a student that had a perfect grade to have the same congratulatory message as the one that had a 94, right? We’ll dig into the code a bit further in later tutorials, but the issue here is that in the check, the score doesn’t include 100; it only checks if the score is less than 100. As a quick fix, we’ll change that second line.

EXAMPLE

```
score = int(input("What is your exam score (0-100): "))
if score > 90 and score <= 100:
  print('You got an A! Congrats!')
elif score > 80 and score < 90:
  print('You got a B! Well done!')
elif score > 70 and score < 80:
  print('You got a C.')
else:
  print("You did not pass the exam.")
```

Let’s test the code again by entering in 100.

```
What is your exam score (0-100): 100
You got an A! Congrats!
```

Success! Well, for now. We have to continue testing all of the other edge and corner cases as well. A **corner case** is when the value you are testing is in between two edge cases. In our scenario, a value of 90 and 80 would be considered corner cases, as they are in between those two edge cases. Let’s see what happens if we enter in 90.

```
What is your exam score (0-100): 90
You did not pass the exam.
```

Oops! Similar to our check on 100, we need to ensure that the value 90 is part of one of the ranges. At this point, 90 is not in either range. Let’s go ahead to do the same thing as we did on the 100 and change line 4 to include 90.

EXAMPLE

```
score = int(input("What is your exam score (0-100): "))
if score > 90 and score <= 100:
  print('You got an A! Congrats!')
elif score > 80 and score <= 90:
  print('You got a B! Well done!')
elif score > 70 and score < 80:
  print('You got a C.')
else:
  print("You did not pass the exam.")
```

Let’s try it again:

```
What is your exam score (0-100): 90
You got a B! Well done!
```

That’s not quite right, as a score of 90 should be an A. Instead, we should have changed line 2 to include 90 as part of the range.

EXAMPLE

```
score = int(input("What is your exam score (0-100): "))
if score >= 90 and score <= 100:
  print('You got an A! Congrats!')
elif score > 80 and score < 90:
  print('You got a B! Well done!')
elif score > 70 and score < 80:
  print('You got a C.')
else:
  print("You did not pass the exam.")
```

Let’s try it again:

```
What is your exam score (0-100): 90
You got an A! Congrats!
```

We’ll also need to do the same thing for the next two conditions as well, so let’s fix those issues.

EXAMPLE

```
score = int(input("What is your exam score (0-100): "))
if score >= 90 and score <= 100:
  print('You got an A! Congrats!')
elif score >= 80 and score < 90:
  print('You got a B! Well done!')
elif score >= 70 and score < 80:
  print('You got a C.')
else:
  print("You did not pass the exam.")
```

TRY IT

Directions: Enter the fixed code above into the IDE, making sure to indent the print functions.
Go ahead and test those corner and edge cases within that range and see what happens. Once you’ve done so, try to consider what happens if the user enters in 101 or -1, which are on the edge cases of the entire program. With a negative number, having a note that the user didn’t pass the example is acceptable. However, if the number is set to a value larger than 100, it’s most likely a larger error that we have to handle.

```
What is your exam score (0-100): 101
You did not pass the exam.
```

Try to think about how you could handle that scenario. What should the result be? Should the user be informed that they entered an incorrect value?

TERMS TO KNOW

Edge Case
Values that are at the end of a testing range.

Interior Test
Test values that are within a specific range where the precise values that we entered within that range did not matter.

Corner Case
The value you are testing is in between two edge cases.

SUMMARY

In this lesson, we learned what a **syntax error** is and how Python does its best at helping us identify the right line and character where it notices a violation of the "grammar" rules. We then looked at how to create **comments** within code to help with testing and documenting, and we identified some of the **best practices for commenting**. Finally, we learned how to test a program using interior testing and testing values at **edge and corner cases**.

Best of luck in your learning!

TERMS TO KNOW

Corner Case
The value you are testing is in between two edge cases.

Edge Case
Values that are at the end of a testing range.

Interior Test
Test values that are within a specific range where the precise values that we entered within that range did not matter.

Syntax Error

A syntax error means that you have violated the “grammar” rules of Python.

print( )

The print() function allows Python to send data to the output screen.