ERRORS IN PYTHON
Introduction to Bugs
"*First actual case of bug being found.*"

The story goes that on September 9th, 1947, computer scientist Grace Hopper found a moth in the Harvard Mark II computer's logbook and reported the world's first literal computer bug. However, the term "bug," in the sense of technical error, dates back at least to 1878 and with Thomas Edison.

Python refers to the mistakes within the program as *errors* and will point to the location where an error occurred with a `^` character. When programs throw errors that we didn't expect to encounter, we call those errors *bugs*. Programmers call the process of updating the program so that it no longer produces bugs *debugging*.

During your programming journey, you are destined to encounter innumerable red errors. Some even say that 75% of development time is spent on debugging. But what makes a programmer successful isn't avoiding errors, it's knowing how to solve them. And a good place to start is understanding what they are.

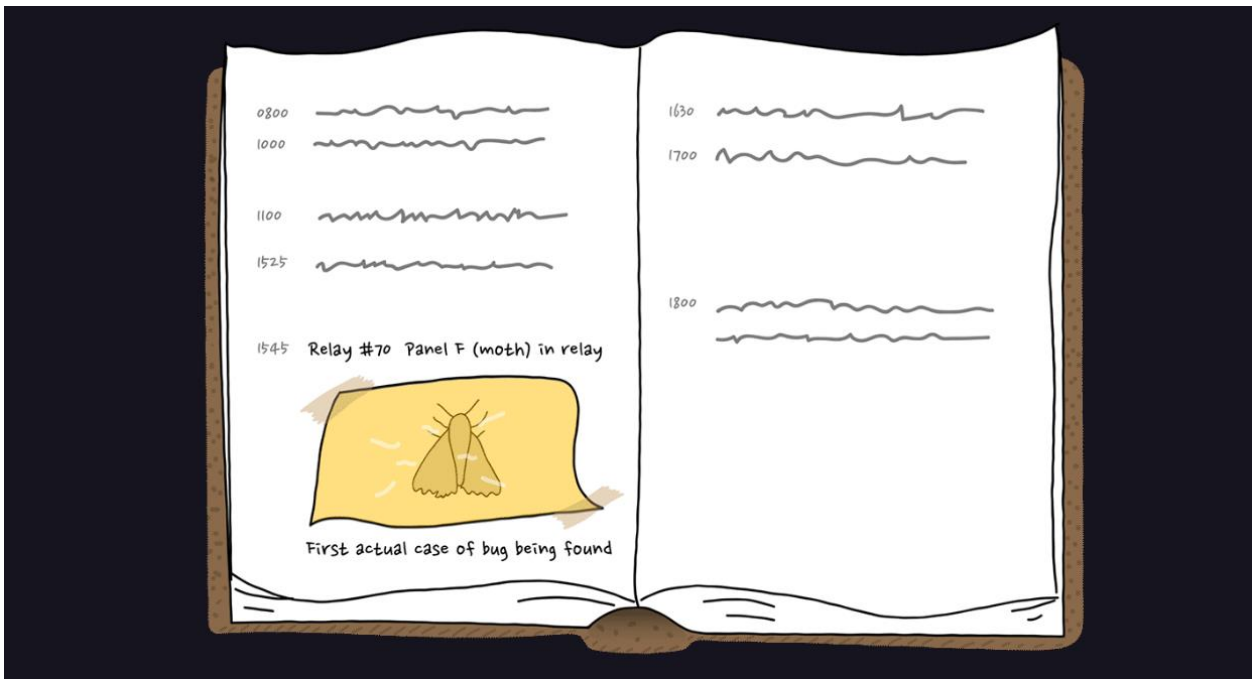
In Python, there are many different ways of classifying errors, but here are some common ones:

- `SyntaxError`: Error caused by not following the proper structure (syntax) of the language.
- `NameError`: Errors reported when the interpreter detects a variable that is unknown.
- `TypeError`: Errors thrown when an operation is applied to an object of an inappropriate type.

In this mini-lesson, we will be looking at these different error messages, and you'll get some practice by debugging them one by one!

Instructions

Click **Next** to continue.



Syntax Errors
When we are writing Python programs, the interpreter is our first line of defense against errors.

`SyntaxError` means there is something wrong with the way your program is written — punctuation that does not belong, a command where it is not expected, or a missing parenthesis can all trigger a `SyntaxError`.

Here's an example of a `SyntaxError` error message:

```
File "script.py", line 1
  print(Hello world!)
                  ^
SyntaxError: invalid syntax
```

The interpreter will tell us where (the file name and line number) it got into trouble and its best guess as to what is wrong.

After reading the error message, we can assume that the cause for this error is a lack of quotation marks around `Hello world!`.

Some common syntax errors are:

- Misspelling a Python keyword.
- Missing colon `:`.
- Missing closing parenthesis `)`, square bracket `]`, or curly brace `}`.

Instructions

1.

In **script.py**, your coworker Carolyn wrote a Fortune Cookie program that is supposed to print out a possible fortune based on a random number and an `if`/`elif`/`else` statement.

Run the program to check it out.

Hint

Here's the pseudocode that they wrote on the whiteboard:

```
if fortune == 0:
  // prints one fortune
elif fortune == 1:
  // prints another fortune
elif fortune == 2:
  // prints another fortune
elif fortune == 3:
  // prints another fortune
elif fortune == 4:
  // prints another fortune
```

And `random.randint(a, b)` basically spits out a random integer *N* such that `a` <= *N* <= `b`.

2.

Oh no, there are three `SyntaxError` errors inside **script.py**!

Can you find them all?

Hint

There will be an error message when the program is run:

```
File "script.py", line 7
  if fortune == 0
                ^
SyntaxError: invalid syntax
```

In **script.py**, on line 7, a colon `:` is expected at the end of the `if` condition.

Can you find the other two bugs? 

Note: Usually the error is on the exact line indicated by the interpreter, or the line of code just before it; however, if the problem is incorrectly nested braces, the actual error may be at the beginning of the nested block.

script.py

```
# Fortune Cookie Program

import random

fortune = random.randint(0, 4)

if fortune == 0:
  print("May you one day be carbon neutral")
elif fortune == 1:
  print("You have rice in your teeth")
elif fortune == 2:
  print("No snowflake feels responsible for an avalanche")
elif fortune == 3:
  print("You can only connect the dots looking backwards")
elif fortune == 4:
  print("The fortune you seek is in another cookie")
```

Name Errors

A `NameError` is reported by the Python interpreter when it detects a variable that is unknown.

This can occur if a variable is used before it has been assigned a value or if a variable name is spelled differently than the point at which it was defined. The Python interpreter will display the line of code where the `NameError` was detected and indicate which name is found that was not defined.

Here's an example of a `NameError` error message:

```
File "script.py", line 1, in <module>
    print(score)
NameError: name 'score' is not defined
```

This error is suggesting that the variable `score` was never defined in the program. Oops.

Some common name errors are:

- Misspelling a variable name.
- Forgetting to define a variable.

Instructions

1.

In **script.py**, another teammate Alex wrote a Who Wants to Be A Millionaire question and four options. If the answer is an uppercase or lowercase "A", then the score goes up.

Run the program to check it out.

Hint

In case you were wondering, the `\n` escape sequence simply prints a new line in the terminal.

2.

Oh no, there are two `NameError` errors!

Can you find them both?

Hint

There will be an error message when the program is run:

```
Traceback (most recent call last):
  File "script.py", line 13, in <module>
    print("C.", option3)
NameError: name 'option3' is not defined
```

In **script.py**, on line 13, a variable `option3` is not defined.

Can you find the other bug? 🐛

Note: Usually the error is on the exact line indicated by the interpreter, or the line of code just before it; however, if the problem is incorrectly nested braces, the actual error may be at the beginning of the nested block.

script.py

```
# Who Wants To Be A Millionaire 💰

score = 0

option1 = 'Fresca'
option2 = 'V8'
option3 = 'Pepsi'
option4 = 'A&W'

print("For ordering his favorite beverages on demand, LBJ had four buttons instal
led in the Oval Office labeled 'Coffee', 'Tea', 'Coke', and what?\n")

print("A.", option1)
```

```
print("B.", option2)
print("C.", option3)
print("D.", option4)

answer = 'a'

if answer == 'A' or answer == 'a':
  score += 100
  print("\nCorrect!")
else:
  print("\nNope, sorry!")
```

Type Errors
A `TypeError` is reported by the Python interpreter when an operation is applied to a variable of an inappropriate type.

This can occur in Python when one attempts to use an operator on something of the incorrect type.

For example, let's see what happens when we try and add together two incompatible types:

```
piggy_bank = '2' + 0.25
```
There will be an `TypeError` error message:

```
Traceback (most recent call last):
  File "script.py", line 1, in <module>
    piggy_bank = '2' + 0.25
TypeError: must be str, not float
```
This error is reporting that `0.25` is not a string.

Some common type errors are:

- Accidentally adding or subtracting a string value.
- Call a function on something of the incorrect type.

Instructions

1.
The word got out in the office that you are a pro bug catcher!

Another peer Alisha pops out of the blue and hands you a program that calculates the area of a triangle, a rectangle, and a circle.

Run the program to check it out.

Hint
Triangle area:

area = \dfrac{1}{2} \cdot base \cdot height$area = \frac{1}{2} \cdot base \cdot height$
Rectangle area:

area = length \cdot width$area = length \cdot width$
Circle area:

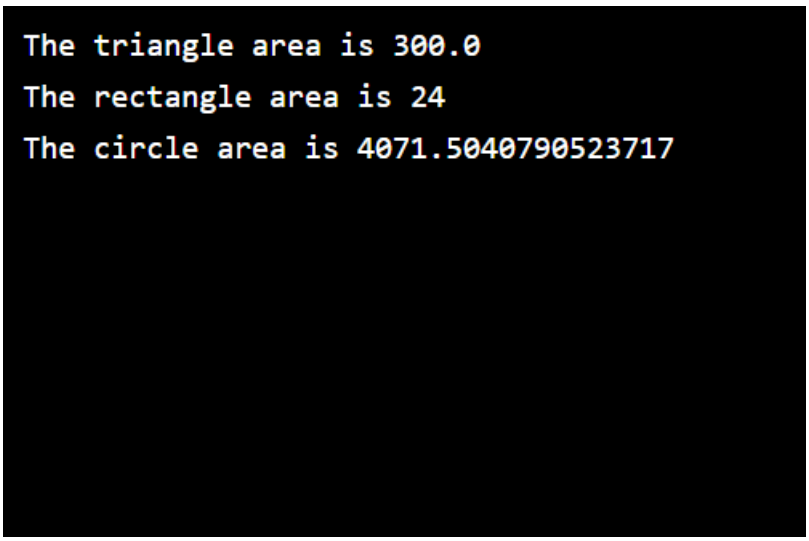
area = \pi r^2$area = \pi r^2$
2.
Oh no, there's one `TypeError` error!

Can you find it?

Hint
Anything interesting about one of the `print()` statements?

script.py

```
# Area Calculator 📏

import math

base = 20
height = 30
area = base * height / 2

print("The triangle area is", area)

length = 2
width = 12
area = length * width

print("The rectangle area is", area )

radius = 36
area = math.pi * radius * radius

print("The circle area is", area)
```



Review

Finding bugs is a huge part of a programmer's life. Don't be intimidated by them! In fact, errors in your code mean you're trying to do something cool.

In this lesson, we have learned about the three types of Python errors:

- `SyntaxError`: Error caused by not following the proper structure (syntax) of the language.
- `NameError`: Errors reported when the interpreter detects an object that is unknown.
- `TypeError`: Errors thrown when an operation is applied to an object of an inappropriate type.

There is also another type of error that doesn't have error messages that we will cover down the line:

- Logic errors: Errors found by the programmer when the program isn't doing what it is intending to do.

Remember, Google and Stack Overflow are a programmer's BFFs (best friends forever) in situations where an error is giving you a lot of trouble. For some more motivation, check out this blog post.

We wish you the best of luck in your bug-squashing journey! 🔎

Instructions

An empty **review.py** file is provided in the code editor. Try experimenting and see what new errors you can find!