Section 12

Debugging

Mohammed Asir Shahid

2021-08-04

Contents

1 The raise and assert Statements

Now we might start finding some more complicated bugs. Debugging is a tool that we need to learn in order to find what we did wrong.

For example, a zero divide error occurs when we try to divide a number by 0. We learned how to handle exceptions with try and except statements to deal with expected errors.

1.1 Raising Your Own Exceptions

We can also raise our own exceptions in our code. This is a way of saying that Python should stop running the code in this function and move to the except statement.

We can raise exceptions using the raise statement.

```
raise Exception("This is the error message.")
```

1.2 Box Example

We want to create a function that creates a box using some supplied characters.

```
def boxPrint(symbol, width, height):

    if len(symbol)!=1:
        raise Exception("symbol needs to be of length 1")

    if width < 2 or height < 2:
        raise Exception("width and height must be greater than or equal to 2")

    print(symbol*width)

    for i in range(height-2):
        print(symbol + (" "*(width-2)) + symbol)

    print(symbol*width)

boxPrint("*", 15, 5)
boxPrint("O", 5, 20)
```

```
***************
*             *
*             *
*             *
***************
OOOOO
O   O
O   O
O   O
O   O
O   O
O   O
O   O
O   O
O   O
O   O
```

```
O   O
O   O
O   O
O   O
O   O
O   O
O   O
O   O
OOOOO
```

Our error message is called a traceback. This is because it shows information showing where the error occurred.

2 The traceback.format$_{\mathbf{exc}}$() Function

We can get the traceback error text as a string vale using this function.

```
import traceback

try:
    raise Exception("This is the error message.")
except:
    errorFile=open("error_log.txt","a")
    errorFile.write(traceback.format_exc())
    errorFile.close()
    print("The traceback info was written error_log.txt")

error=open("error_log.txt")
print(error.read())
```

```
The traceback info was written error_log.txt
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
Exception: This is the error message.
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
Exception: This is the error message.
Traceback (most recent call last):
```

```
  File "<stdin>", line 5, in <module>
Exception: This is the error message.
```

2.1 Assertions and the assert Statement

An assertion is a sanity check that makes sure the code isn't doing something really wrong. These are performed by assert statements. If the sanity check fails, then an assertion error exception is raised. These assertions are for programmer errors and the program should not run after an assertion is raised.

```
assert False, "This is the error message"
```

Let's try to create a simple traffic simulator program with intersections with stop lights.

```
market_2nd={"ns": "green", "ew": "red"}

def switchLights(intersection):
    for key in intersection.keys():
        print(intersection[key])
        if intersection[key]=="green":
            intersection[key]="yellow"
        elif intersection[key]=="yellow":
            intersection[key]="red"
        elif intersection[key]=="red":
            intersection[key]="green"
    assert "red" in intersection.values(), "Neither light is red!"

print(market_2nd)
switchLights(market_2nd)
print(market_2nd)
```