Python-Exception Handling

Errors and Exceptions in Python

Errors are problems in a program that causes the program to stop its execution. On the other hand, exceptions are raised when some internal events change the program's normal flow.

- Syntax Errors in Python
- Python Logical Errors (Exception)
- Common Builtin Exceptions
- Error Handling

Syntax Errors in Python

When the proper syntax of the language is not followed then a syntax error is thrown.

Example: It returns a syntax error message because after the if statement a colon: is missing. We can fix this by writing the correct syntax.

Python3

```
# initialize the amount variable
amount = 10000


# check that You are eligible to
#  purchase Dsa Self Paced or not
if(amount>2999)
    print("You are eligible to purchase Dsa Self Paced")
```

Output:

```
  File "/home/ac35380186f4ca7978956ff46697139b.py", line 4
    if(amount>2999)
                  ^
SyntaxError: invalid syntax
```

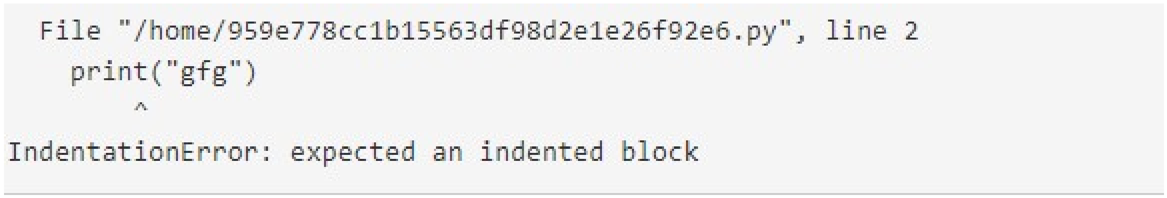
Example 2: When indentation is not correct.

Python

```
if(a<3):
print("gfg")
```

Python-Exception Handling

Output



```
  File "/home/959e778cc1b15563df98d2e1e26f92e6.py", line 2
    print("gfg")
        ^
IndentationError: expected an indented block
```

Python Logical Errors (Exception)

A logical error in Python, or in any programming language, is a type of bug that occurs when a program runs without crashing but produces incorrect or unintended results. Logical errors are mistakes in the program's logic that lead to incorrect behavior or output, despite the syntax being correct.

Characteristics of Logical Errors

1. **No Syntax Error**: The code runs successfully without any syntax errors.
2. **Unexpected Output**: The program produces output that is different from what is expected.
3. **Difficult to Detect**: Logical errors can be subtle and are often harder to identify and fix compared to syntax errors because the program appears to run correctly.
4. **Varied Causes**: They can arise from incorrect assumptions, faulty logic, improper use of operators, or incorrect sequence of instructions.

Example of a Logical Error

Consider a simple example where we want to compute the average of a list of numbers:

Python

```
numbers = [10, 20, 30, 40, 50]

total = 0


# Calculate the sum of the numbers

for number in numbers:

    total += number


# Calculate the average (this has a logical error)

average = total / len(numbers) - 1
```

print("The average is:", average)

Analysis

- **Expected Output**: The average of the numbers [10, 20, 30, 40, 50] should be 30.
- **Actual Output**: The program will output The average is: 29.0.

Cause of Logical Error

The logical error is in the calculation of the average:

average = total / len(numbers) - 1

Instead, it should be:

average = total / len(numbers)

The incorrect logic here is the subtraction of 1, which results in a wrong average calculation.

Common Builtin Exceptions

Exception	**Description**
IndexError	When the wrong index of a list is retrieved.
AssertionError	It occurs when the assert statement fails
AttributeError	It occurs when an attribute assignment is failed.
ImportError	It occurs when an imported module is not found.
KeyError	It occurs when the key of the dictionary is not found.
NameError	It occurs when the variable is not defined.
MemoryError	It occurs when a program runs out of memory.
TypeError	It occurs when a function and operation are applied in an incorrect type.

Python-Exception Handling

Note: For more information, refer to Built-in Exceptions in Python

Error Handling

When an error and an exception are raised then we handle that with the help of the Handling method.

Handling Exceptions with Try/Except/Finally

We can handle errors by the Try/Except/Finally method. we write unsafe code in the try, fall back code in except and final code in finally block.

Python

```
# put unsafe operation in try block
try:
    print("code start")


    # unsafe operation perform
    print(1 / 0)


# if error occur the it goes in except block
except:
    print("an error occurs")


# final code in finally block
finally:
    print("GeeksForGeeks")
```

Output:

```
code start
an error occurs
GeeksForGeeks
```

Raising exceptions for a predefined condition

When we want to code for the limitation of certain conditions then we can raise an exception.

```
# try for unsafe code
```

Python-Exception Handling

```
try:
    amount = 1999
    if amount < 2999:

        # raise the ValueError
        raise ValueError("please add money in your account")
    else:
        print("You are eligible to purchase DSA Self Paced course")

# if false then raise the value error
except ValueError as e:
        print(e)
```

Output:

```
please add money in your account
```

Thank you