Exception Handling using Python

Tushar B. Kute,
http://tusharkute.com

Exception

- Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.
- Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.
- Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

Exceptions

- ZeroDivisionError: Occurs when a number is divided by zero.
- NameError: It occurs when a name is not found. It may be local or global.
- IndentationError: If incorrect indentation is given.
- IOError: It occurs when Input Output operation fails.
- EOFError: It occurs when end of file is reached and yet operations are being performed

Uncaught Exception

```
num1 = int(input('Enter first:'))
num2 = int(input('Enter second:'))

result = num1 / num2

print('Division is:', result)
```

```
Enter first:12
Enter second:0
Traceback (most recent call last):
  File "exc.py", line 4, in <module>
    result = num1 / num2
ZeroDivisionError: division by zero
```

Python's exception handling

assert

raise

try

except

else

finally

try… except

- The malicious code (code having exception) is enclosed in the try block.
- Try block is followed by except statement. There can be multiple except statement with a single try block.
- Except statement specifies the exception which occurred. In case that exception is occurred, the corresponding statement will be executed.

try... except

- Handling the exceptions:

```
try:
  code to monitor for exception
except exception-name:
  message to print
```

Using try… except

```
a = int(input("Enter first: "))
b = int(input("Enter second: "))
try:
    a = a / b
except ZeroDivisionError:
    print("Denominator is zero")
    exit(0)
print("Division is:", a)
```

```
Enter first: 12
Enter second: 0
Denominator is zero
```

```
Enter first: 12
Enter second: 4
Division is: 3.0
```

Still … uncaught exceptions

```
a = int(input("Enter first: "))
b = int(input("Enter second: "))
try:
    a = a / b
except ZeroDivisionError:
    print("Denominator is zero")
    exit(0)
print("Division is:", a)
```

```
Enter first: zero
Traceback (most recent call last):
  File "exc1.py", line 1, in <module>
    a = int(input("Enter first: "))
ValueError: invalid literal for int() with base 10: 'zero'
```

Handling multiple exceptions

```
try:
    a = int(input("Enter first: "))
    b = int(input("Enter second: "))
    a = a / b
except ZeroDivisionError:
    print("Denominator is zero")
    exit(0)
except ValueError:
    print("Enter Proper Values")
    exit(0)
print('Division is', a)
```

```
Enter first: 10
Enter second: 3
Division is 3.333
```

```
Enter first: 10
Enter second: 0
Denominator is zero
```

```
Enter first: 12
Enter second: one
Enter Proper Values
```

Handling all exceptions

```
try:
    a = int(input("Enter first: "))
    b = int(input("Enter second: "))
    a = a / b
except:
    print("Error")
    exit(0)
print("Division is:", a)
```

```
Enter first: 12
Enter second: 0
Error
```

```
Enter first: 12
Enter second: zero
Error
```

```
Enter first: 12
Enter second: 3
Division is: 4.0
```

Handling exception with messages

```
try:
    a = int(input("Enter first: "))
    b = int(input("Enter second: "))
    a = a / b
except Exception as e:
    print("Error", e)
    exit(0)
print("Division is:", a)
```

```
Enter first: 12
Enter second: 0
Error division by zero
```

```
Enter first: 12
Enter second: 3
Division is: 4.0
```

```
Enter first: 12
Enter second: three
Error invalid literal for int() with base 10: 'three'
```

Finally block

- In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block.
- Finally block will always be executed irrespective of the exception.
- Generally used for shutdown activities like closing the file, terminating database connections etc.

Example: finally

```
try:
    a = int(input("Enter first: "))
    b = int(input("Enter second: "))
    a = a / b
except ZeroDivisionError:
    print("Denominator is zero")
finally:
    print("Good Bye")

print('Division is', a)
```

```
Enter first: 12
Enter second: 3
Good Bye
Division is 4.0
```

```
Enter first: 12
Enter second: 0
Denominator is zero
Good Bye
Division is 12
```

```
Enter first: 12
Enter second: five
Good Bye
Traceback (most recent call last):
  File "exc3-1.py", line 3, in <module>
    b = int(input("Enter second: "))
ValueError: invalid literal for int() with base 10: 'five'
```

The else block

- The try statement can have a else block too.
- The else statements will be executed when except is not executed.
- Either except (in case of exception) or else (no exception) blocks are executed.

Complete exception handling

try:

Run this code

except:

Execute this code when there is an exception

else:

No exceptions? Run this code.

finally:

Always run this code.

try...except...else...finally

```
try:
    a = int(input("Enter first: "))
    b = int(input("Enter second: "))
    a = a / b
except ZeroDivisionError:
    print("Denominator is zero")
else:
    print('Division is', a)
finally:
    print("Good Bye")
```

```
Enter first: 12
Enter second: 0
Denominator is zero
Good Bye
```

```
Enter first: 12
Enter second: 3
Division is 4.0
Good Bye
```

```
Enter first: 13
Enter second: Two
Good Bye
Traceback (most recent call last):
  File "exc3-2.py", line 3, in <module>
    b = int(input("Enter second: "))
ValueError: invalid literal for int() wi
```

Raising an exception

- We can use raise to throw an exception if a condition occurs.
- The statement can be complemented with a custom exception.

Use raise to force an exception:

Example: raising an exception

```
num = int(input('Enter a number: '))
if num > 100:
    raise Exception('Large Number !!!')

print('Square is:', num * num)
```

```
Enter a number: 12
Square is: 144
```

```
Enter a number: 123
Traceback (most recent call last):
  File "exc4.py", line 3, in <module>
    raise Exception('Large Number !!!')
Exception: Large Number !!!
```

Creating your own exception

- Sometimes you may need to create custom exceptions that serves your purpose.
- In Python, users can define such exceptions by creating a new class.
- This exception class has to be derived, either directly or indirectly, from Exception class.
- Most of the built-in exceptions are also derived from this class.

Negative Number Exception

```
class NegNumException(Exception):
    def __init__(self, data):
        self.data = data
    def __str__(self):
        return repr('NEG number')

try:
    num = int(input('Enter number: '))
    if num < 0:
        raise NegNumException(num)
except NegNumException as ae:
    print("Error:", ae)
    exit(0)
print('Square:', num * num)
```

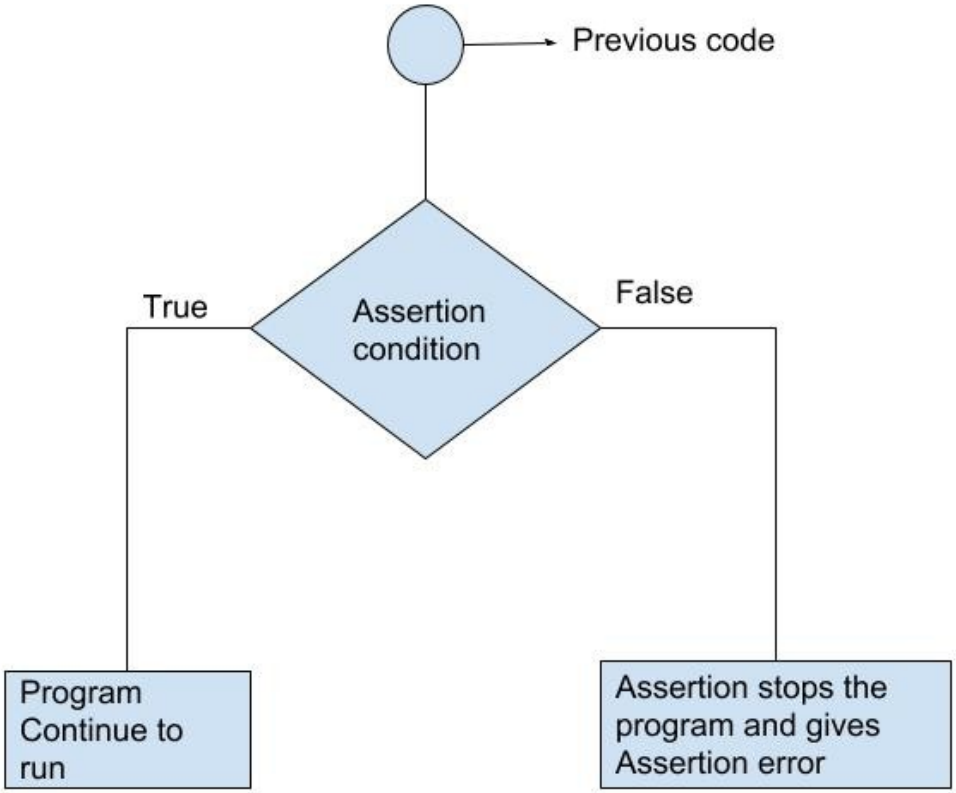
```
Enter number: -33
Error: 'NEG number'
```

```
Enter number: 12
Square: 144
```

Assertions

- Assertions are statements that assert or state a fact confidently in your program.
- For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are simply boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.
- It is also a debugging tool as it brings the program on halt as soon as any error is occurred and shows on which point of the program error has occurred.

How assertions work ?



When to use assert ?

- In Python we can use assert statement in two ways as mentioned above.
- assert statement has a condition and if the condition is not satisfied the program will stop and give AssertionError.
- assert statement can also have a condition and a optional error message. If the condition is not satisfied assert stops the program and gives AssertionError along with the error message.

Example:

```
def divide(num1, num2):
    assert num2 != 0
    return num1 / num2

x = int(input('Enter first number:'))
y = int(input('Enter second number:'))
print('Division is', divide(x,y))
```

```
Enter first number:12
Enter second number:2
Division is 6.0
```

```
Enter first number:4
Enter second number:0
Traceback (most recent call last):
  File "exc8.py", line 7, in <module>
    print('Division is', divide(x,y))
  File "exc8.py", line 2, in divide
    assert num2 != 0
AssertionError
```

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

c/MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com