Exception Handling

Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.

Exceptions in Python

Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

Python try...except

try….. except blocks are used in python to handle errors and exceptions. The code in try block runs when there is no error. If the try block catches the error, then the except block is executed.

Syntax:

```
try:
    #statements which could generate
    #exception
except:
    #Soloution of generated exception
```

Example:

```
try:
   num = int(input("Enter an integer: "))
except ValueError:
   print("Number entered is not an integer.")
```

Output:

```
Enter an integer: 6.022
Number entered is not an integer.
```

Finally Clause

The finally code block is also a part of exception handling. When we handle exception using the try and except block, we can include a finally block at the end. The finally block is always executed, so it is generally used for doing the concluding tasks like closing file resources or closing database connection or may be ending the program execution with a delightful message.

Syntax:

```
try:
  #statements which could generate
  #exception
except:
  #solution of generated exception
finally:
   #block of code which is going to
   #execute in any situation
```

The finally block is executed irrespective of the outcome of try……except…..else blocks
One of the important use cases of finally block is in a function which returns a value.

Example:

```
try:
   num = int(input("Enter an integer: "))
except ValueError:
   print("Number entered is not an integer.")
else:
   print("Integer Accepted.")
finally:
   print("This block is always executed.")
```

Output 1:

```
Enter an integer: 19
Integer Accepted.
This block is always executed.
```

Output 2:

```
Enter an integer: 3.142
Number entered is not an integer.
This block is always executed.
```

Raising Custom errors

In python, we can raise custom errors by using the `raise` keyword.

```
salary = int(input("Enter salary amount: "))
if not 2000 < salary < 5000:
   raise ValueError("Not a valid salary")
```

In the previous tutorial, we learned about different built-in exceptions in Python and why it is important to handle exceptions. However, sometimes we may need to create our own custom exceptions that serve our purpose.

Defining Custom Exceptions

In Python, we can define custom exceptions by creating a new class that is derived from the built-in Exception class.

Here's the syntax to define custom exceptions:

```
class CustomError(Exception):
  # code ...
  pass

try:
  # code ...

except CustomError:
  # code...
```

This is useful because sometimes we might want to do something when a particular exception is raised. For example, sending an error report to the admin, calling an api, etc.