Exception Handling

Errors can also occur at runtime and these are called exceptions. They occur, for example, when a file we try to open does not exist dividing a number by zero, module we try to import is not found etc.

Whenever these type of runtime error occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Common 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 the file is reached and yet operations are being performed.

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as much as possible.

Syntax

```
try:
    statement
except Exception1:
    statement
except Exception2:
    statement
....
....
except ExceptionN:
    statement
else:
    execute code if exception is not found
```

Example	**Output**

try:
    print a
except NameError:
        print "a is not defined"
else:
    print "Value of a is",a
	a is not defined

Except with no Exception

Except statement can also be used without specifying Exception

Example	**Output**
try:	a is not defined

print a except: print "a is not defined" else: print "Value of a is",a	

Example	Output

try:
    a=1/0
    print a
except:
        print "Divide by zero exception"
else:
    print "Value of a is",a
	Divide by zero exception

Handling Multiple Exception with single except statement

We can also use the same except statement to handle multiple exceptions.

Example	Output
try: a=1/0 print a except ArithmeticError,NameError: print "Divide by zero exception"	Divide by zero exception

else: print "Value of a is",a	

Finally Block

We can use a **finally:** block along with a **try:** block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

Example	**Output**
try: a=1/0 print "Divide by zero exception" finally: print "finally block is executed"	Code to be executed Traceback (most recent call last): File "main.py", line 2, in <module> a=10/0; ZeroDivisionError: integer division or modulo by zero

Raise an Exceptions

In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.

```
raise Exception,<args>
```

Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

Example	**Output**

try:
   raise NameError('Exception')
except NameError:
   print('exception is raised')
	exception is raised

User-Defined Exceptions

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Example	**Output**

class CustomError(Exception):
    def __init__(self, data):
       self.data = data
try:
   raise CustomError("Exception")
except CustomError as e:
   print "User defined Exception:", e.data
	User defined Exception: Exception