Common Exceptions

Python provides the number of built-in exceptions, but here we are describing the common standard exceptions.

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

### **The try-expect statement**

If the Python program contains suspicious code that may throw the exception, we must place that code in the **try** block. The **try** block must be followed with the **except** statement, which contains a block of code that will be executed if there is some exception in the try block.

**Syntax**

1. **try**:
2. #block of code
4. **except** Exception1:
5. #block of code
7. **except** Exception2:
8. #block of code
10. #other code
11. **try**:
12. a = int(input("Enter a:"))
13. b = int(input("Enter b:"))
14. c = a/b
15. **except**:
16. **print**("Can't divide with zero")

try-except-else:

We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.

1. **try**:
2. #block of code
4. **except** Exception1:
5. #block of code
7. **else**:

     #this code executes if no except block is executed

1. **try**:
2. a = int(input("Enter a:"))
3. b = int(input("Enter b:"))
4. c = a/b
5. **print**("a/b = %d"%c)
6. # Using Exception with except statement. If we print(Exception) it will return exception class
7. **except** Exception:
8. **print**("can't divide by zero")
9. **print**(Exception)
10. **else**:
11. **print**("Hi I am else block")

## try...finally block

Python provides the optional **finally** statement, which is used with the **try** statement. It is executed no matter what exception occurs and used to release the external resource. The finally block provides a guarantee of the execution.

1. **try**:
2. # block of code
3. # this may throw an exception
4. **finally**:
5. # block of code
6. # this will always be executed
7. **try**:
8. a = int(input("Enter a:"))
9. b = int(input("Enter b:"))
10. c = a/b
11. **print**("a/b = %d"%c)
12. # Using Exception with except statement. If we print(Exception) it will return exception class
13. **except** Exception:
14. **print**("can't divide by zero")
15. **print**(Exception)
16. **finally**:
17. **print**("Hi I am finally block")

-----5 marks -----

## Raising exceptions

An exception can be raised forcefully by using the **raise** clause in Python. It is useful in in that scenario where we need to raise an exception to stop the execution of the program.

**Syntax**

1. **raise** Exception\_class,<value>
2. **try**:
3. age = int(input("Enter the age:"))
4. **if**(age<18):
5. **raise** ValueError
6. **else**:
7. **print**("the age is valid")
8. **except** ValueError:
9. **print**("The age is not valid")

## Custom Exception

The Python allows us to create our exceptions that can be raised from the program and caught using the except clause. However, we suggest you read this section after visiting the Python object and classes.

1. **class** ErrorInCode(Exception):
2. **def** \_\_init\_\_(self, data):
3. self.data = data
4. **def** \_\_str\_\_(self):
5. **return** repr(self.data)
7. **try**:
8. **raise** ErrorInCode(2000)
9. **except** ErrorInCode as ae:
10. **print**("Received error:", ae.data)