**B1:Explain Exception handling?What is an Error in Python?**

Exceptions handling in Python is very similar to Java. But whereas in Java exceptions are caught by catch clauses, we have statements introduced by an "except" keyword in Python.

Eg : *n = int(input("Please enter a number: "))*

Please enter a number: 23.50 Exception occurs like

ValueError: invalid literal for int() with base 10: '23.5'

*try:*

*n = input("Please enter an integer: ")*

*n = int(n)*

*break*

*except:*

*print("No valid integer! Please try again ...")*

*print("Great, you successfully entered an integer!")*

Three types of errors,

*syntax error:*

print("hy)

print("hy"

prin("hy")

*Logical error:*

Sum of two digits

a=10

b=20

print(a-b)

*Run time error:*

Zero division error

**B2. How many except statements can a try-except block have? Name Some built-in exception classes**:

Except clause , A try statement may have more than one except clause for different exceptions.

But at most one except clause will be executed. Eg :

*import sys*

*try:*

*…… …… ……*

*except IOError:*

*……..*

*except ValueError:*

*…….*

*except :*

*sys.exc\_info()[0]*

|  |  |
| --- | --- |
| Python Built-in Exceptions | |
| **Exception** | **Cause of Error** |
| AssertionError | Raised when assert statement fails. |
| AttributeError | Raised when attribute assignment or reference fails. |
| EOFError | Raised when the input() functions hits end-of-file condition. |
| FloatingPointError | Raised when a floating point operation fails. |
| GeneratorExit | Raise when a generator's close() method is called. |
| ImportError | Raised when the imported module is not found. |
| IndexError | Raised when index of a sequence is out of range. |
| KeyError | Raised when a key is not found in a dictionary. |
| KeyboardInterrupt | Raised when the user hits interrupt key (Ctrl+c or delete). |
| MemoryError | Raised when an operation runs out of memory. |
| NameError | Raised when a variable is not found in local or global scope. |
| NotImplementedError | Raised by abstract methods. |
| OSError | Raised when system operation causes system related error. |
| OverflowError | Raised when result of an arithmetic operation is too large to be represented. |
| ReferenceError | Raised when a weak reference proxy is used to access a garbage collected referent. |
| RuntimeError | Raised when an error does not fall under any other category. |
| StopIteration | Raised by next() function to indicate that there is no further item to be returned by iterator. |
| SyntaxError | Raised by parser when syntax error is encountered. |
| IndentationError | Raised when there is incorrect indentation. |
| TabError | Raised when indentation consists of inconsistent tabs and spaces. |
| SystemError | Raised when interpreter detects internal error. |
| SystemExit | Raised by sys.exit() function. |
| TypeError | Raised when a function or operation is applied to an object of incorrect type. |
| UnboundLocalError | Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable. |
| UnicodeError | Raised when a Unicode-related encoding or decoding error occurs. |
| UnicodeEncodeError | Raised when a Unicode-related error occurs during encoding. |
| UnicodeDecodeError | Raised when a Unicode-related error occurs during decoding. |
| UnicodeTranslateError | Raised when a Unicode-related error occurs during translating. |
| ValueError | Raised when a function gets argument of correct type but improper value. |
| ZeroDivisionError | Raised when second operand of division or modulo operation is zero. |

**B3. When will the else part of try-except-else be executed?**

The else part is only executed on two conditions :

1. The code in the relevant ‘try’ block has completed
2. The code in the relevant ‘try’ block does not raise an exception.

**B4. Can one block of except statements handle multiple exception?**

yes, like except TypeError, SyntaxError [,…].

**B5.When is the finally block executed?**

try statement had always been paired with except clauses. But there is another way to use it as well.

The try statement can be followed by a finally clause.

Finally clauses are called clean-up or termination clauses, because they must be executed under all circumstances, i.e. a "finally" clause is always executed regardless if an exception occurred in a try block or not.

Syntax:

try:

……………… ……………..

finally :

……………..

**B6. What happens when ‘1’ == 1 is executed?**

we get a False, It simply evaluates to False and does not raise any exception.

**b7. How Do You Handle Exceptions With Try/Except/Finally In Python? exaplin with coding snippets**

a=input("Enter 1st number: ")

b=input("Enter 2nd number: ")

try:

a=int(a)

b=int(b)

c=a/b

print(c)

except(ZeroDivisionError):

print("Zero division error.")

except(ValueError):

print("Value error.")

except:

print("Unexpected error")

fially:

print("This is an example of division.")

**I3.Explain Python Errors and Built-in Exceptions with coding snippets**

*#syntax error:*

print("hy)

print("hy"

prin("hy")

*#Logical error:*

#Sum of two digits

a=10

b=20

print(a-b)

*#Run time error:*

a=input("Enter 1st number: ")

b=input("Enter 2nd number: ")

try:

a=int(a)

b=int(b)

c=a/b

print(c)

except(ZeroDivisionError):

print("Zero division error.")

except(ValueError):

print("Value error.")

except:

print("Unexpected error")

**A1.Explain .User-Defined Exception in Python**

In this example, we will illustrate how user-defined exceptions can be used in a program to raise and catch errors.

This program will ask the user to enter a number until they guess a stored number correctly. To help them figure it out, hint is provided whether their guess is greater than or less than the stored number.

*# define Python user-defined exceptions*

*class Error(Exception):*

*"""Base class for other exceptions"""*

*pass*

*class ValueTooSmallError(Error):*

*"""Raised when the input value is too small"""*

*pass*

*class ValueTooLargeError(Error):*

*"""Raised when the input value is too large"""*

*pass*

*# our main program*

*# user guesses a number until he/she gets it right*

*# you need to guess this number*

*number = 10*

*while True:*

*try:*

*i\_num = int(input("Enter a number: "))*

*if i\_num < number:*

*raise ValueTooSmallError*

*elif i\_num > number:*

*raise ValueTooLargeError*

*break*

*except ValueTooSmallError:*

*print("This value is too small, try again!")*

*print()*

*except ValueTooLargeError:*

*print("This value is too large, try again!")*

*print()*

*print("Congratulations! You guessed it correctly.")*

Here is a sample run of this program.

*Enter a number: 12*

*This value is too large, try again!*

*Enter a number: 0*

*This value is too small, try again!*

*Enter a number: 8*

*This value is too small, try again!*

*Enter a number: 10*

*Congratulations! You guessed it correctly.*

Here, we have defined a base class called Error.

The other two exceptions (ValueTooSmallError and ValueTooLargeError) that are actually raised by our program are derived from this class. This is the standard way to define user-defined exceptions in Python programming, but you are not limited to this way only.

**A3.What is Assertions in Python?**

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.

**A5.What is Argument of an Exception?**

An exception can have an *argument*, which is a value that gives additional information about the problem. The contents of the argument vary by exception. You capture an exception's argument by supplying a variable in the except clause as follows −

try:

You do your operations here;

......................

except *ExceptionType, Argument*:

You can print value of Argument here...

If you write the code to handle a single exception, you can have a variable follow the name of the exception in the except statement. If you are trapping multiple exceptions, you can have a variable follow the tuple of the exception.

This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.