Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Exceptions

Learning outcomes:

What is Exception?

Handling an Exception

The except Clause with No Exceptions

The except Clause with Multiple Exceptions

The try-finally Clause

List of Standard Exceptions

Raising an Exception

Argument of an Exception

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an Exception

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 elegantly as possible. Exceptional handling is used to handle the other part of the code can be executed without any disruption. If we do not handle the exception, the interpreter doesn't execute all the code that exists after that

Handling an Exception

Syntax:

```
try:
    #block of code
except Exception1:
    #block of code
except Exception2:
    #block of code
 #other code
```

Handling an Exception

Here are few important points about the above-mentioned syntax:

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.

The *except* Clause with No Exceptions

You can also use the except statement with no exceptions defined as follows:

```
try:
You do your operations here;
.....................
except:
If there is any exception, then execute this block.
.....................
else:
If there is no exception then execute this block.
```

The *except* Clause with No Exceptions

This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

The *except* Clause with *Multiple* Exceptions

You can also use the same *except* statement to handle multiple exceptions as follows:

try:
You do your operations here;
.....
except(Exception1[, Exception2[,...ExceptionN]]]):
If there is any exception from the given exception list, then execute this block.
.....
else:
If there is no exception then execute this block.

The try-finally Clause

You 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. The syntax of the try-finally statement is this:

```
try:
You do your operations here;
.....................
Due to any exception, this may be skipped.
finally:
This would always be executed.
```

List of Standard Exceptions

Some of the standard exceptions are as follows:

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StandardError	Base class for all built-in exceptions except *StopIteration* and *SystemExit.*
ArithmeticError	Base class for all errors that occur for numeric calculation
ZeroDivisonError	Raised when division or modulo by zero takes place for all numeric types.
NameError	Raised when an identifier is not found in the local or global namespace.

List of Standard Exceptions

EXCEPTION NAME	DESCRIPTION
IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
SyntaxError	Raised when there is an error in Python syntax.
KeyError	Raised when the specified key is not found in the dictionary.
IndexError	Raised when an index is not found in a sequence .
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.

List of Standard Exceptions

Let's see the some examples.

Raising an Exception

You can raise exceptions in several ways by using the raise statement. The general syntax for the **raise** statement is as follows.

Syntax

raise [Exception [, args [, traceback]]]

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.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Raising an Exception

Example:

```
try:
    age = int(input("Enter the age?"))
    if age<18:
        raise ValueError;
    else:
        print("the age is valid")
except ValueError:
    print("The age is not valid")
```

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 can 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...
```

Argument of an Exception

Example:

```
def temp_convert(var):
  try:
    return int(var)
  except ValueError:
    print ("The argument does not contain
numbers\n")

# Call above function here.
temp_convert("xyz");
```