Python

**Python Keywords**

* Reserved Words
* Can’t use for variable, functions and other identifiers
* Case Sensitive and lower case (except True, False and None)

|  |
| --- |
|  |
| False | class | finally | is | return |
| None | continue | for | lambda | try |
| True | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass |  |
| break | except | in | raise |  |

**Identifiers:**

* Names given to entities like class, functions, variables etc.
* Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.
* An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
* Keywords cannot be used as identifiers.
* We cannot use special symbols like !, @, #, $, % etc. in our identifier.
* Identifier can be of any length.

## Python Statement

Instructions that a Python interpreter can execute are called statements.

### Multi-line statement

a = 1 + 2 + 3 + \

4 + 5 + 6 + \

7 + 8 + 9

a = (1 + 2 + 3 +

4 + 5 + 6 +

7 + 8 + 9)

colors = ['red',

'blue',

'green']

a = 1; b = 2; c = 3

## Python Indentation

* A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unindented line.

for i in range(1,11):

print(i)

if i == 5:

break

if True:

print('Hello')

a = 5

if True: print('Hello'); a = 5

## Python Comments

* In Python, we use the hash (#) symbol to start writing a comment.

#This is a comment

#print out Hello

print('Hello')

### Multi-line comments

#This is a long comment

#and it extends

#to multiple lines

"""This is also a

perfect example of

multi-line comments"""

### Docstring in Python

* Docstring is short for documentation string.
* It is a string that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring.
* Triple quotes are used while writing docstrings. For example:

Refer: IntellPython\src\GettingStarted\FunctionStatements.py

## Data types in Python

* There are various data types in Python. Some of the important types are listed below.
* Datatypes are defined as int, float and complex class in Python.
* type() and the isinstance() function to check if an object belongs to a particular class
* Mutable
* Refer IntellPython\src\GettingStarted\Datatype1.py

### Python List

* Ordered Sequence.
* Mostly used
* All items can be different data type
* >>> a = [1, 2.2, 'python']
* Refer /IntellPython/src/GettingStarted/DatatypeList.py

### Python Tuple

* Ordered Sequence.
* Immutable
* Usually faster than list as it cannot change dynamically.
* All items can be different data type
* Refer /IntellPython/src/GettingStarted/DatatypeTuple.py

### Python Strings

* We can use single quotes or double quotes to represent strings.
* Multi-line strings can be denoted using triple quotes, ''' or """.
* Refer /IntellPython/src/GettingStarted/DatatypeString.py

### Python Set

* UnOrdered Sequence.
* Set is defined by values separated by comma inside braces { }.
* Refer /IntellPython/src/GettingStarted/DatatypeSet.py

### Python Dictionary

* Un Ordered Sequence.
* Kind of key value pair- map
* Fast to retrieve data when we have huge amount of data.
* Refer /IntellPython/src/GettingStarted/DatatypeDict.py
* Also Refer data conversion
* Refer /IntellPython/src/GettingStarted/DatatypeCoversion.py

## Exception Handling in Python

### Some Built-in Python Exceptions

List of some built-in python exceptions are given below.

1. **Exception :** This is the base class for all kind of the exceptions. All kind of exceptions should be derived from this class
2. **ArithmeticError :** This is the base class for the exception raised for any arithmetic errors.
3. **EOFError :** This exception raise when input() function read End-of-File without reading any data.
4. **ZeroDivisionError :** This exception raise when the second argument of a division or modulo operation is zero
5. **AssertionError :** This exception raise when an **assert** statement fails.
6. **FloatingPointError :** This exception raise when a floating point operation fails.
7. **KeyError :** This exception raise when a mapping (dictionary) key is not found in the set of existing keys.

### Python try expect

While writing the code, some statements might suspicious for raising an error. Hence, those statements should be surrounded with try-except-else block. For example, we will now raise an exception by our code. The following code will raise IndexError Exception.

name = 'Imtiaz Abedin'

print(name[15])

print('This will not print')

If you try running the code, you will get below exception.

Traceback (most recent call last):

File "/home/imtiaz/ExceptionHandling.py", line 2, in

print(name[15])

IndexError: string index out of range

Because the size of the string type object ‘name’ is less than 15 and we are try to access the index no 15. Have a look, the second print statement is not executed for that exception. So program crashes due to exception. So, in the next code we will handle this exception.

name = 'Imtiaz Abedin'

try:

print(name[15])

except IndexError:

print('IndexError has been found!')

print('This will be printed print.')

### Basic Structure of Python Exception Handling

In the previous section, we demonstrate about how exception raised and how to handle that. In this section we will discuss about the basic coding structure for handling exceptions. Therefore, the basic coding structure for Python Exception Handling is given below.

name = 'Imtiaz Abedin'

try:

# Write the suspicious block of code

print(name[15])

except AssertionError: # Catch a single exception

# This block will be executed if exception A is caught

print('AssertionError')

except (EnvironmentError, SyntaxError, NameError) as E: # catch multiple exception

# This block will be executed if any of the exception B, C or D is caught

print(E)

except :

print('Exception')

# This block will be executed if any other exception other than A, B, C or D is caught

else:

# If no exception is caught, this block will be executed

pass

finally:

# This block will be executed and it is a must!

pass

# this line is not related to the try-except block

print('This will be printed.')

### Python Exception Handling Important Points

For undergoing a professional python project you need to be careful about exceptions. A simple exception can ruin your code. So, you need to handle those exceptions. A few important points about handling exceptions are given below.

1. It is better to surround the suspicious code with try-except.
2. Using one try-except block for one line of suspicious code is better that using one try-except block for a block of suspicious code.
3. It is better to catch specific exception class. Using generalized exception class is not that much useful for handling.

### Raising an Exception

You can raise an existing exception by using **raise** keyword. So, you just simply write **raise** keyword and then the name of the exception. If we modify the previous code, we get

def input\_age(age):

try:

if(int(age)<=18):

raise ZeroDivisionError

except ValueError:

return 'ValueError: Cannot convert into int'

else:

return 'Age is saved successfully'

print(input\_age('23')) # This will execute properly

print(input\_age('18')) # This will not execute properly