Python

* **Introduction**

Python is an easy programming language and popular programming language too. Python is open-source and can get those libraries from python website python.org. In python, function and datatypes were implemented in C, C++. It can be used for many applications like data cleaning, databases and high-performance computing etc. It holds Data libraries like SciPy, NumPy etc.

Python is being used in Data Science technology development like Neural Networks, Artificial Intelligence, Statistics etc.

* **Python – Installation**

You can understand the Installation of python on windows and Ubuntu in this tutorial. Before that let me give provide you the download link for downloading the Python.

For Linux and Unix Systems, below is the link to download Python:

https://docs.python.org/3/using/unix.html

For Windows system, below is the link to download Python:

https://docs.python.org/3/using/windows.html

**Installation on Ubuntu or Linux Systems:**

Python will come pre-installed on most of the Linux distributions. Just you need to give the python3 to start programming in the terminal.

If you don’t have the Python, then get the source from

https://www.python.org/downloads/source/

follow the below commands.

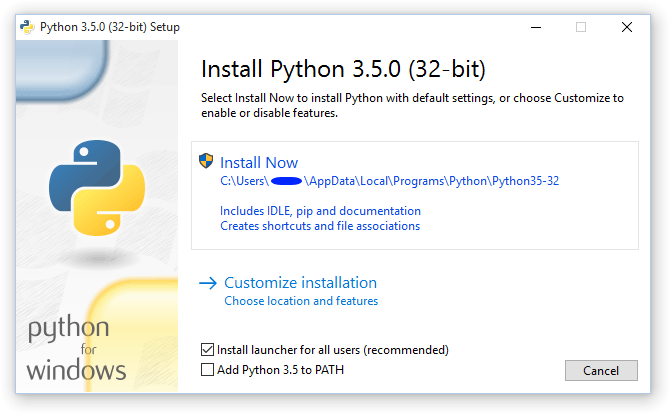
./configure

make

make install

**Installation on windows systems:**

Python installers are available to install python on 32-bit and 64-bit versions. Just download the installer and install the software. Once the installation opens the command prompt and give the python command to test.



Pic source: [www.python.org](http://www.python.org)

* **Python – Syntax**

In this tutorial, you will learn the python syntax and an example about how to write a basic but popular Hello World print program.

**Note:** In this tutorial, we are using python version 3.5. Hence all the examples reflect the same results as like we execute in Python 3.5

**Example:**

>>> x="Hello World"

>>> print(x)

Hello World

>>>

In above Python terminal, x is a variable where we have assigned a value as hello world in double quotation as it is the string value. Then, we used print function to print the variable x which is in parenthesis. Do remember that we have given parenthesis for variable in print function.

**Example:**

>>> x=2

>>> print(x)

2

>>>

* **Python – Variables and Data types**

**Variable:**

A variable is the location in the memory to store the values. A variable may hold different types of values like numbers, strings etc. In Python, no need to declare a datatype for a variable. It will understand by the value that is assigned to the variable.

The name of variable or a function that we define can be called as Identifier. An Identifier must obey the below rules.

1. Identifiers can have letters, digits, underscores.

2. there is no definite in length.

3. All identifiers must start with letter or underscore. You cannot use digits.

4. Identifier should not be a keyword. (which is a reserved word for python)

**Examples of value assignment to a variable:**

>>>x = 10 # x is Integer

>>>y = 10.1 # y is Float

>>>z = "Hello" # z is String

>>>x,y = y,x # assign x value to y and y value to x

>>>a,b,c = 10,20,30 # assign a,b,c values sperated by comma at a time

>>>print(x,y,a,b,c) # printing all the values

**Data Types:**

Python has different types of datatypes as below.

1.Numbers

2.String

3.List

4.Tuple

5.Dictionary

6.Boolean

* **Python – Numbers**

In Python, Numbers can be defined in three ways.

1. Integer

2. Float

3. Complex

**Let us understand how to work with Integer numbers**

>>>x = 10 # assigning the integer value to variable x

>>>x

10

**we can identify which type of value the variable x is holding**

>>>x=10

>>>type(x)

<class 'int'>

**Let us understand how to work with Float numbers**

>>>y=10.1 # assigning the float value to variable y

>>>y

10.1

**Let us identify the type of value the variable y is holding**

>>>y=10.1

>>>type(y)

<class 'float'>

**Let us understand how to work with complex type**

>>>x=4+2y # 4 is the real part and 2y is the imaginary part

We can perform the various calculations with these numbers. Let us see few examples below.

>>>5+5

10

>>>5\*5

25

>>>5-4

1

* **Python – Strings**

In this tutorial, we will work on the Python Strings where we can learn about the manipulation of Strings, using String Operators and string methods and Functions. First, let us understand that how do we declare the strings in python programming language. We can declare and print the strings by placing them in single Quotes ('..'), Double Quotes (".."), and using the print function too. Python Strings are Immutable (An Object with a fixed value).

Using the Strings in single quotes ('...')

>>> 'hello world'

'hello world'

Below command will give an error.

>>> 'let's start' # this will give us an error

File ",stdin>, line 1

'let's do it'

^

SyntaxError: invalid syntax

**To overcome that we must use escape character \**

>>> 'let\'start'

"let's start"

**Using the Strings in Double quotes ("...")**

>>> "let's start" # using double quotes to avoid escape character

"let's start"

**Using the Strings in Print() function**

>>> print("let's start") # we have enclose the strings in double quotation inside print funtion

let's start

**Using the 3 double quotes start and end of the string allows us to print the data including spaces and newlines.**

>>> print("""let's

...start

...Now""")

let's

start

now

**String Concatenation:**

Multiple Strings can be concatenated using (+) symbol. Let us see the example of concatenating the strings.

**Example**:

>>> x="hello"

>>> y="world"

>>> x+y

'helloworld'

**String Repetition:**

String repetition can be performed by using the (\*) symbol. Let us see the example of repetition of strings.

**Example**:

>>> 3\*"hello"

'hellohellohello'

Strings are indexed with each character in a memory location when assigned to a variable. The indexed number starts from zero '0' from first character till the end of the string. Whereas, reverse indexing starts with '-1' from right to left until the starting character. Let us try few examples of retrieving the characters from a word PYTHON in either ways.

**Example**:

>>> x="P Y T H O N" # Word python is written without spaces

0 1 2 3 4 5

-6-5-4-3-2-1

>>> x[3]

'H'

>>> x[-5]

'Y'

>>> x[:4] # Starting from first character, 4th position excluded

'PYTH'

>>> x[:-4] # Starting from fourth character from right, 4th position excluded

'PY'

>>> X[0:] # Starting from first character till end

'PYTHON'

>>> x[-6:] # Starting from -6th position until start ie., -1 postion

'PYTHON'

**String Methods in Python:**

|  |  |
| --- | --- |
| **Python String Methods** | **Description** |
| capitalize() | Returns the String with first Character as Capital Letter |
| casefold() | Returns a casefolded copy |
| center(width[, fillchar]) | This will pads the string with a character specified |
| count(sub[, start[, end]]) | Returns the number of occurances of substring in string |
| encode(encoding="utf-8", errors="strict") | returns an encoded string |
| endswith(suffix[, start[, end]]) | Check the string if it ends with the specified |
| expandtabs(tabsize=8) | Replace the tab with space |
| find(sub[, start[, end]]) | returns the highest index |
| format(\*args, \*\*kwargs) | formats the string |
| format\_map(mapping) | formats the string except the mapping is directly used |
| index(sub[, start[, end]]) | returns the index of substring |
| isalnum() | checks for alphanumeric Char |
| isalpha() | Checks if all characters are Alphabets |
| isdecimal() | Checks for decimal characters |
| isdigit() | Checks for digit char |
| isidentifier() | checks for valid Identifier |
| islower() | checks for lowercase of all alphabets in string |
| isnumeric() | Checks for Numeric Char |
| isprintable() | Checks for Printable Char |
| isspace() | Checks for Whitespace Characters |
| istitle() | Returns true if the string is titlecased |
| isupper() | Checks if all characters are Uppercase |
| join(iterable) | returns concatenated string |
| ljust(width[, fillchar]) | returns left-justified string |
| lower() | returns lowercased string |
| lstrip([chars]) | Removes Leading Characters |
| partition(sep) | returns a tuple |
| replace(old, new[, count]) | replaces the substring |
| rfind(sub[, start[, end]]) | Returns the Highest Index |
| rindex(sub[, start[, end]]) | Returns Highest Index but raises when substring is not found |
| rjust(width[, fillchar]) | Returns the string right justified |
| rpartition(sep) | Returns a tuple |
| rsplit(sep=None, maxsplit=-1) | Splits String From Right |
| rstrip([chars]) | Removes Trailing Characters |
| split(sep=None, maxsplit=-1) | Splits String from Left |
| splitlines([keepends]) | Splits String at Lines |
| startswith(prefix[, start[, end]]) | Checks if String Starts with the Specified String |
| strip([chars]) | Removes Both Leading and Trailing Characters |
| swapcase() | swap uppercase characters to lowercase and vice versa |
| title() | Returns a Title Cased String |
| translate(table) | returns mapped charactered string |
| upper() | returns uppercased string |
| zfill(width) | Returns a Copy of The String Padded With Zeros |

* **Python – Sequences**

Sequence in Python can be defined with a generic term as an ordered set which can be classified as two sequence types. They are mutable and immutable. There are different types of sequences in python. They are Lists, Tuples, Ranges.

**Lists**: Lists will come under mutable type in which data elements can be changed.

**Tuples**: Tuples are also like Lists which comes under immutable type which cannot be changed.

**Ranges**: Ranges is mostly used for looping operations and this will come under immutable type.

**The common Sequence Operations are listed below:**

x in s : Returns true if an item in s is equal to x

x not in s : Returns false if an item in s is equal to x

s+t : concatenation

s\*n : adding s to itself n number of times

s[i] : ith item of s, index starts from zero

s[i:j]: slice of s from i to j

len(s) : length of s

max(s) : largest item in s

min(s) : smallest item of s

s.count(x): total number of occurrences of x in s

* **Python – Lists**

Python Lists holds the data of any datatype like an array of elements and these are mutable means the possibility of changing the content or data in it. List can be created by giving the values that are separated by commas and enclosed in square brackets. Let us see different types of value assignments to a list.

**Example**:

List1=[10,20,30,40,50];

List2=['A','B','C','D'];

List3=[10.1,11.2,12.3];

List4=['html','java','oracle'];

List5=['html',10.1,10,'A'];

As we know the way strings can be accessed, same way Lists can be accessed. Below is example of indexing in python for your understanding again.

**Example**:

List1=[10,20,30,40,50];

0 1 2 3 4 ---> Forward Indexing

-5 -4 -3 -2 -1 ---> Backward Indexing

**Accessing and slicing the elements in List**

Now let us take a list which holds different datatypes and will access the elements in that list.

**Example**:

>>> list5=['html',10.1,10,'A'];

>>> list5[0]

'html'

>>> list5[1:2];

[10.1]

>>> list5[-2:-1];

[10]

>>> list5[:-1];

['html', 10.1, 10]

>>> list5[:-2];

['html', 10.1]

>>> list5[1:-2];

[10.1]

>>> list5[1:-1];

[10.1, 10]

>>> list5[-1];

'A'

>>> list5[3:];

['A']

**Using Functions with Lists:**

**Example**:

>>> list5=['html',10.1,10,'A'];

>>> len(list5)

4

>>> 10 in list5

True

>>> 'html' in list5

True

>>> num=[10,20,30,40];

>>> sum(num)

100

>>> max(num)

40

>>> min(num)

10

**Checking if the Lists are mutable:**

**Example**:

>>> score=[10,20,30,80,50]

>>> score

[10, 20, 30, 80, 50]

>>> score[3]=40

>>> score

[10, 20, 30, 40, 50]

**List Comprehension:**

List comprehension works like iterate operations as mentioned below.

**Syntax**:

[x for x in iterable]

**Example**:

>>> var=[x for x in range(5)];

>>> var

[0, 1, 2, 3, 4]

>>> var=[x+1 for x in range(5)];

>>> var

[1, 2, 3, 4, 5]

>>> var=[x for x in range(5) if x%3==0];

>>> var

[0, 3]

**Adding Elements to a list:**

We can add two lists as shown in the below example.

**Example**:

>>> var1=[10,20]

>>> var2=[30,40]

>>> var3=var1+var2

>>> var3

[10, 20, 30, 40]

**Replicating elements in Lists:**

we can replicate elements in lists as shown in the below example.

**Example**:

>>> var1\*2

[10, 20, 10, 20]

>>> var1\*3

[10, 20, 10, 20, 10, 20]

>>> var1\*4

[10, 20, 10, 20, 10, 20, 10, 20]

**Appending elements in Lists:**

We can append an element to an existing list as shown in the below example.

**Example**:

>>> var1.append(30)

>>> var1

[10, 20, 30]

>>> var1.append(40)

>>> var1

[10, 20, 30, 40]

>>> var2

[30, 40]

* **Python – Tuples**

Tuples are generally used to store the heterogeneous data which immutable. Even Tuple looks like Lists but Lists are mutable. To create a tuple, we need to use the comma which separates the values enclosed parentheses.

**Example:**

>>> tup1=() # Creating an empty tuple

>>> tup1

()

>>> tup1=(10)

>>> tup1

10

>>> tup1=(10,20,30);

>>> tup1

(10, 20, 30)

>>> tup1=tuple([1,1,2,2,3,3])

>>> tup1

(1, 1, 2, 2, 3, 3)

>>> tup1=("tuple")

>>> tup1

'tuple'

**Using Functions with Tuples:**

>>> tup1=(10,20,30);

>>> max(tup1)

30

>>> min(tup1)

10

>>> len(tup1)

3

**Operators with Tuples:**

>>> 20 in tup1

True

>>> 30 not in tup1

False

**Slicing in Tuples:**

>>> tup1[0:4]

(10, 20, 30)

>>> tup1[0:1]

(10,)

>>> tup1[0:2]

(10, 20)

* **Python – Dictionary**

Dictionaries are created or indexed by key-value pairs. In which keys are immutable type. Tuples can be used as keys, but lists cannot be used as keys. Just because lists are mutable. Generally, the key-value pairs which are stored in the Dictionary can be accessed with the key. we can delete a key value too. Let us see some examples.

**Example**:

>>> score={'maths':80,'physics':70,'chemistry':85}

>>> score

{'physics': 70, 'maths': 80, 'chemistry': 85}

>>> score['maths']

80

>>> del score['maths']

>>> score['maths']

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'maths'

>>> score

{'physics': 70, 'chemistry': 85}

>>> score.keys()

dict\_keys(['physics', 'chemistry'])

>>> keys=score.keys()

>>> keys

dict\_keys(['physics', 'chemistry'])

>>> list(keys)

['physics', 'chemistry']

* **Python – Ranges**

Range a kind of data type in python which is an immutable. Range will be used in for loops for number of iterations. Range is a constructor which takes arguments and those must be integers. Below is the syntax.

**Syntax**:

class range(stop)

class range(start, stop[, step])

stop: the value of stop parameter.

step: the value of step parameter. If the value is omitted, it defaults to 1.

start: the value of start parameter. If the value is omitted, it defaults to zero.

**Examples**:

>>> list(range(5))

[0, 1, 2, 3, 4]

>>> list(range(10,20))

[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

>>> list(range(10,20,5))

[10, 15]

>>> list(range(10,20,2))

[10, 12, 14, 16, 18]

# It will not take the float numbers

>>> list(range(0,0.1))

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'float' object cannot be interpreted as an integer

>>> list(range(0,2))

[0, 1]

>>> list(range(0,1))

[0]

>>> list(range(0,10,5))

[0, 5]

* **Python – Sets**

In this tutorial, we will learn about sets in python. A set is a datatype which holds an unordered collection with immutable and no duplicate elements. By the name, Set can be used for various mathematical operations. Mathematical operation may be union, intersection or difference, etc. Let us see the example of using the Set below.

**Example**:

>>> set1={'html','c','java','python','sql'}

>>> print(set1)

{'c', 'python', 'sql', 'html', 'java'}

# Below we have given duplicates

>>> set1={'html','c','java','python','sql','java'}

# we can observe that duplicates are ignored

>>> print(set1)

{'c', 'python', 'java', 'html', 'sql'}

>>> set1

{'c', 'python', 'java', 'html', 'sql'}

**Membership testing in Sets:**

>>> set1={'html','java','python','sql','java'}

>>> set1

{'python', 'java', 'html', 'sql'}

>>> print(set1)

{'python', 'java', 'html', 'sql'}

>>> 'c' in set1

False

>>> 'java' in set1

True

**Sets Operations in Python:**

>>> set1={'html','java','python','sql','java'}

>>> set2={'html', 'oracle', 'ruby'}

# Unique words in set1

>>> set1

{'python', 'java', 'html', 'sql'}

>>> set2

{'ruby', 'html', 'oracle'}

# words in set1 but not in set2

>>> set1-set2

{'python', 'java', 'sql'}

# Words in set1 or set2 or both

>>> set1|set2

{'ruby', 'html', 'oracle', 'python', 'java', 'sql'}

# Words in both set1 and set2

>>> set1&set2

{'html'}

# Words in set1 or set2 but not both

>>> set1^set2

{'oracle', 'python', 'sql', 'ruby', 'java'}

* **Python - Operators**

Operators in Python helps us to perform the mathematical operations with numbers. There are different operators in Python as below.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| // | Integer division |
| + | addition |
| - | subtraction |
| \* | multiplication |
| / | Float division |
| % | Provide remainder after division(Modulus) |
| \*\* | Perform exponent (raise to power) |

Let us try implementing every operator now.

**1. Addition: symbol used (+)**

>>> 10+10

20

>>>20+30

50

>>>50+50

100

**2. Substration: symbol used (-)**

>>>20-10

10

>>>50-40

10

>>>100-30

70

**3. multiplication: Symbol used (\*)**

>>>5\*2

10

>>>10\*2

20

>>>20\*2

40

**4. Float Division: This will divide and provide the result in floating value and the symbol used (/)**

>>>5/2

2.5

>>>10/2

5.0

**5. Integer Division: This will divide and truncate the decimal and provide the Integer value and the symbol used (//)**

>>>5//2

2

>>>7//2

3

**6. Exponentiation Operator: This will help us to calculate a power b and return the result**

>>>10\*\*3 # This mean 10\*10\*10

1000

**7. Modulus Operator: This will provide the remainder after the calucation and symbol used (%)**

>>>10%3

1

What if we want to work with multiple operators at a time. Here comes the Operator precedence in Python.

| **Operator** | **Description** |
| --- | --- |
| [lambda](https://docs.python.org/3/reference/expressions.html#lambda) | Lambda expression |
| [if](https://docs.python.org/3/reference/compound_stmts.html#if) – [else](https://docs.python.org/3/reference/compound_stmts.html#else) | Conditional expression |
| [or](https://docs.python.org/3/reference/expressions.html#or) | Boolean OR |
| [and](https://docs.python.org/3/reference/expressions.html#and) | Boolean AND |
| [not](https://docs.python.org/3/reference/expressions.html#not) x | Boolean NOT |
| [in](https://docs.python.org/3/reference/expressions.html#in), [not in](https://docs.python.org/3/reference/expressions.html#not-in), [is](https://docs.python.org/3/reference/expressions.html#is), [is not](https://docs.python.org/3/reference/expressions.html#is-not), <, <=, >, >=, !=, == | Comparisons, including membership tests and identity tests |
| | | Bitwise OR |
| ^ | Bitwise XOR |
| & | Bitwise AND |
| <<, >> | Shifts |
| +, - | Addition and subtraction |
| \*, @, /, //, % | Multiplication, matrix multiplication division, remainder [[5]](https://docs.python.org/3/reference/expressions.html#id21) |
| +x, -x, ~x | Positive, negative, bitwise NOT |
| \*\* | Exponentiation [[6]](https://docs.python.org/3/reference/expressions.html#id22) |
| await x | Await expression |
| x[index], x[index:index], x(arguments...), x.attribute | Subscription, slicing, call, attribute reference |
| (expressions...), [expressions...], {key: value...}, {expressions...} | Binding or tuple display, list display, dictionary display, set display |

* **Python – If.. Else.. Statements**

In this tutorial, we will discuss about the "if" Condition statement. Let us understand how this "if" statements work. There will be 2 parts in the "if" statement. They are "if" and "elif", "else" which is optional. when the "if" condition satisfies then it executes the program inside that else it execute the program inside "elif" or "else" statements.

**Syntax:**

if Condition:

program of if

elif test expression:

program of elif

else:

program of else

**Below is the example of If.. Else:**

**Example 1:**

>>> x=int(input("Please enter a number:"))

Please enter a number:10

>>> if x<0:

... print("negative is zero")

... elif x==1:

... print("single")

... else:

... print("positive and greater than 1")

...

positive and greater than 1

**Example 2:**

>>> x=int(input("Please enter a number:"))

Please enter a number:1

>>> if x<0:

... print("negative is zero")

... elif x==1:

... print("single")

... else:

... print("positive and greater than 1")

...

Single

* **Python – For Loop**

In this tutorial, we will learn about for loop. In Python, for loop is used to iterate over the sequence of elements (the sequence may be list, tuple or strings.. etc). Below is the syntax.

**Syntax:**

for\_stmt ::= "for" target\_list "in" expression\_list ":" suite

["else" ":" suite]

In the below example, we have given the list of strings as courses and the for loop created to iterate through all the strings to print the course and the length of the course name.

**Example:**

>>> courses=['html','c','java','css']

>>> for i in courses:

... print(i, len(i))

...

html 4

c 1

java 4

css 3

>>>

In the below example, for loop iterates through the list of numbers. In the immediate step, if statement filters only the numbers less than 50, else it will display "no values" for rest of the iterations.

**Example:**

>>> x=[10,20,30,40,50,60]

>>> x

[10, 20, 30, 40, 50, 60]

>>> for i in x:

... if i<50:

... print(i)

... else:

... print("no values")

...

**Below is the output:**

10

20

30

40

no values

no values

* **Python – While Loop**

In this tutorial, we will learn about while loop. In python, while loop is used to iterate until the condition is satisfied. If the condition given is not satisfied in the first iteration itself, the block of code inside the loop will not get executed.

In the below example, we have assigned the value of x as zero and started the while loop until the value of x is less than 10 and print the values.

**Example**:

>>> x=0

>>> while x<10:

... x=x+1

... print(x)

...

**Below is the Output:**

1

2

3

4

5

6

7

8

9

10

Just changed the values for the above example and below is the output.

>>> x=100

>>> while x<110:

... x=x+1

... print(x)

...

**Below is the output:**

101

102

103

104

105

106

107

108

109

110

* **Python – Break**

In this tutorial, we will learn about the Break statement in Python. Break statement is used to terminate the loop program at a point.

Let us understand the below example which do not have "break" statement will go through multiple iterations till the value of i becomes 109.

**Example:**

>>> for i in range(100,110):

... print('Assigned value of i is ',i)

... for num in range(100,i):

... print(i,num)

...

**Below is the output:**

Assigned value of i is 100

Assigned value of i is 101

101 100

Assigned value of i is 102

102 100

102 101

Assigned value of i is 103

103 100

103 101

103 102

Assigned value of i is 104

104 100

104 101

104 102

104 103

Assigned value of i is 105

105 100

105 101

105 102

105 103

105 104

Assigned value of i is 106

106 100

106 101

106 102

106 103

106 104

106 105

Assigned value of i is 107

107 100

107 101

107 102

107 103

107 104

107 105

107 106

Assigned value of i is 108

108 100

108 101

108 102

108 103

108 104

108 105

108 106

108 107

Assigned value of i is 109

109 100

109 101

109 102

109 103

109 104

109 105

109 106

109 107

109 108

Now, let us break the loop when the value of i becomes 105. Below is the code and output for clarification.

**Example:**

>>> for i in range(100,110):

... print('Assigned value of i is ',i)

... for num in range(100,i):

... print(i,num)

... if i==105:

... break

...

**Below is the Ouput:**

Assigned value of i is 100

Assigned value of i is 101

101 100

Assigned value of i is 102

102 100

102 101

Assigned value of i is 103

103 100

103 101

103 102

Assigned value of i is 104

104 100

104 101

104 102

104 103

Assigned value of i is 105

105 100

105 101

105 102

105 103

105 104

* **Python – Continue**

In this tutorial, we will learn about the Continue statement in Python. Continue Statement is used to take the control to top of the loop for next iteration leaving the rest of the statements in the loop without execution.

**Below is the similar example of a for loop:**

>>> for i in range(10):

... if (i==4 or i ==8):

... continue

... print(i)

...

**Below is the Output:**

0

1

2

3

5

6

7

9

**Below is another example where the printing of even numbers are ignored:**

>>> for i in range(100,110):

... if i%2==0:

... continue

... print(i,' is an even number')

... else:

... print(i,' is an Odd number')

...

**Below is the output:**

101 is an Odd number

103 is an Odd number

105 is an Odd number

107 is an Odd number

109 is an Odd number

* **Python – Pass**

Pass statement is used when there is a situation where a statement is required for syntax in the code, but which should not to be executed. So that, When the program executes that portion of code will not be executed.

In the below example, we can observe that the pass statement in if condition was not executed.

**Example:**

>>> for i in range(100,104):

... print('Assigned value of i is ',i)

... for num in range(100,i):

... print(i,num)

... if num==102:

... pass

...

...

**Below is the Output:**

Assigned value of i is 100

Assigned value of i is 101

101 100

Assigned value of i is 102

102 100

102 101

Assigned value of i is 103

103 100

103 101

103 102

In the below example, we can observe that the pass statement is not mentioned. Hence resulted an error.

**Example:**

>>> for i in range(100,104):

... print('Assigned value of i is ',i)

... for num in range(100,i):

... print(i,num)

... if num==102:

...

File "<stdin>", line 6

^

IndentationError: expected an indented block

* **Python - Date & Time**

In python, datetime is a module which provides different classes to work with dates and times.

**The types of objects in datetime are as below.**

1. date

2. time

2. datetime

3. timedelta

4. tzinfo

5. timezone

**Date Object:**

Date object depicts the date as date(year, month, day) in an ideal Gregorian calendar. Syntax of the Date class is represented as below.

**Syntax**:

class datetime.date(year,month,day)

**All the arguments are integers. Every argument has its own range of values as below.**

1. YEAR: MINYEAR - MAXYEAR (1 - 9999)

2. MONTH: 1 - 12

3. DAY: 1 - number of days in the given month and year.

Now let us work with date object and its methods which serves different requirements with an example. The below example shows the current date in different formats.

**Example:**

# need to import the date

>>> from datetime import date

# method today() shows the today's date

>>> today=date.today()

# let us see the date

>>> today

datetime.date(2017, 11, 7)

>>> today=x

>>> x

time.struct\_time(tm\_year=2017, tm\_mon=8, tm\_mday=15, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=1, tm\_yday=227, tm\_isdst=-1)

# assigning the date

>>> d=date(2017,11,7)

>>> x=d.timetuple()

# Print the date values from the tuple by year, month, day ..etc

>>> for i in x:

... print(i)

...

**Below is the output:**

2017

11

7

0

0

0

1 # week day which is TUESDAY ( where '0' is MONDAY)

311 # 311 th day in the year 2017

-1

# print the iso format of date

>>> d.isoformat()

'2017-11-07'

# String formats of date

>>> d.strftime("%d/%m/%u")

'07/11/2'

>>> d.strftime("%A%d.%B %Y")

'Tuesday07.November 2017'

>>> 'The {1} is {0:%d}, the {2} is {0:%B}.'.format(d, "day", "month")

'The day is 07, the month is November.'

**Time object:**

A time object which gives the information about time of any particular day subject to the requirements. The syntax of the time object constructor is given below.

**Syntax**:

1. HOUR: 0 - 24

2. MINUTE: 0 to < 60

3. SECOND: 0 to < 60

4. MICROSECOND: 0 to < 1000000

5. fold in [0,1]

**Example:**

>>> from datetime import time

>>> t=time(12,12,12)

>>> t

datetime.time(12, 12, 12)

>>> t.isoformat()

'12:12:12'

**Datetime Object:**

Datetime object is a combination of both date and time information which can provide the functions from date object and time object.

**Syntax**:

class datetime.datetime(year, month, day, hour=0, minute=0, second=0, microsecond=0, tzinfo=None, \*, fold=0)

All the arguments are integers. Each argument has its own range of values as below.

1. YEAR: MINYEAR – MAXYEAR (1 - 9999)

2. MONTH: 1 - 12

3. DAY: 1 - number of days in the given month and year.

4. HOUR: 0 - 24

5. MINUTE: 0 to < 60

6. SECOND: 0 to < 60

7. MICROSECOND: 0 to < 1000000

8. fold in [0,1]

Now let us work with datetime object and its methods which serves different requirements with an example.

**Example:**

# import the datetime, date, time

>>> from datetime import datetime, date, time

# date

>>> d=date(2017,11,7)

# time

>>> t=time(10,10)

# combine both date and time

>>> datetime.combine(d,t)

datetime.datetime(2017, 11, 7, 10, 10)

# current date and time

>>> datetime.now()

datetime.datetime(2017, 11, 7, 17, 21, 24, 338804)

>>> datetime.utcnow()

datetime.datetime(2017, 11, 7, 11, 51, 37, 256627)

>>> dt=datetime.now()

>>> dt

datetime.datetime(2017, 11, 7, 17, 22, 58, 626832)

# displaying the time tuple

>>> tt=dt.timetuple()

>>> tt

time.struct\_time(tm\_year=2017, tm\_mon=11, tm\_mday=7, tm\_hour=17, tm\_min=22, tm\_sec=58, tm\_wday=1, tm\_yday=311, tm\_isdst=-1)

>>> for i in tt:

... print(i)

...

2017 # year

11 # month

7 # day

17 # hour

22 # minute

58 # second

1 # weekday ( 0 = Monday)

311 # 311 th day in the year 2017

-1

# string format of date and time

>>> dt.strftime("%A, %d. %B %Y %I:%M%p")

'Tuesday, 07. November 2017 05:22PM'

>>> 'The {1} is {0:%d}, the {2} is {0:%B}, the {3} is {0:%I:%M%p}.'.format(dt, "day", "month", "time")

'The day is 07, the month is November, the time is 05:22PM.'

>>> dt.isoformat()

'2017-11-07T17:22:58.626832'

* **Python – Functions**

**There are 2 types of functions.**

1. User-defined functions

2. Pre-defined functions

**User-Defined functions:**

In Python, User-defined function is a block of code which can reusable. Once they are defined or written, that can be used multiple times and in other applications too.

Generally, the syntax of user-defined functions is represented by keyword def. Below is the syntax for defining the function without arguments.

**Syntax:**

def function\_name( args ):

statement 1

statement 2

return

**Let us create a Function for returning a value of x after processing a loop.**

>>> def prim(n):

... for x in range(2,n):

... for i in range(2,x):

... if x%i==0:

... break

... else:

... print(x)

...

**Let us call the User-defined Function:**

>>> prim(10)

2

3

5

7

**Using the argument value as 20:**

>>> prim(20)

2

3

5

7

11

13

17

19

**Using the argument value as 50:**

>>> prim(50)

2

3

5

7

11

13

17

19

23

29

31

37

41

43

47

**Pre-defined Functions:**

Pre-defined functions are already existing functions which cannot be changed. But still we can make our own custom functions using those pre-defined functions.

abs()

all()

any()

ascii()

bin()

bool()

bytearray()

bytes()

callable()

chr()

classmethod()

compile()

complex()

delattr()

dict()

dir()

divmod()

enumerate()

eval()

exec()

filter()

float()

format()

frozenset()

getattr()

globals()

hasattr()

hash()

help()

hex()

id()

input()

int()

isinstance()

issubclass()

iter()

len()

list()

locals()

map()

max()

memoryview()

min()

next()

object()

oct()

open()

ord()

pow()

print()

property()

range()

repr()

reversed()

round()

set()

setattr()

slice()

sorted()

staticmethod()

str()

sum()

super()

tuple()

type()

vars()

zip()

\_\_import\_\_()

* **Python - Packages and modules**

In this tutorial, we will learn about Modules in Python. Modules in Python are none other than .py extension files which contains various statements and functions. In order to import a module, we will use the "import" command.

**There are many modules in Python. Below are the very few examples just for giving an idea for you.**

1. math - this is a mathematical module

2. statistics - This is for Statistical functions

3. pathlib - this is for filesystem paths

4. urllib - this is for handling urls

5. zlib - this is for data compression

6. csv - this is for file reading and writing

The above all are few pre-existing modules which are in python.

We can also write our own modules. Let us see how to create a module which helps us to process the prime numbers under any given value.

**First create a file called "prime.py" and write the below code into the file.**

def prim(n):

for x in range(2,n):

for i in range(2,x):

if x%i==0:

break

else:

print(x)

Now connect to python3 and import the module called "prime" using the keyword import. Then, call the function by passing the integer value as an argument to list the prime numbers for the given value.

>>> import prime

>>> prime.prim(10)

2

3

5

7

Calling the function with value 20:

>>> prime.prim(20)

2

3

5

7

11

13

17

19

>>>

dir(module\_name) will list us all types of variables, modules, funtions used for the given module.

>>> dir(prime)

['\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'prim']

**Packages:**

Packages are the namespaces within which consists many modules and packages. Every package is none other than a directory that consists a file called "--init\_\_.py". This file describes the directories as packages.

For example, we have created the prime.py module in the above example. Now let us create the package for the same.

**To create a package, we will to follow the below steps.**

1. first have the module ready, that is prime.py

2. Create a directory called "primenum" and keep the above module in that directory.

3. Create a file called \_\_init\_\_.py

Let us try accessing the same.

Now we can import the prime module in 2 ways as below.

>>> import primenum.prime

>>> from primenum import prime

* **Python – Reading a File**

**Reading a File in Python:**

In this tutorial, we will learn about how to read the date from the file in python. In Python, we will use the method called "open" to open the file and "read" method to read the contents of the file.

**Open Method take 2 arguments.**

1. File name - refers to the name of the file.

2. Mode - refers to the mode of opening the file which may be either write mode or read mode.

**Modes:**

r - opens for reading the file

w - opens for writing the file

r+ - opens for both reading and writing

**Read Methods:**

**read() -** This method reads the entire data in the file. if you pass the argument as read(1), it will read the first character and return the same.

**readline() -** This method reads the first line in the file.

**Let us work on an example:**

**First creating the data in a file called "techsal.csv". Below is the data how it looks.**

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

**Now let us import the module called "csv", open the file and read the data:**

>>>import csv

>>> file=open("techsal.csv","r")

>>> print(file.read())

**Below is the output:**

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

**Let us read the first character of the data by passing the numeric argument:**

>>> file=open("techsal.csv","r")

>>> print(file.read(1))

d

**Let us read the first line of the file by using the readline() method:**

>>> file=open("techsal.csv","r")

>>> print(file.readline())

designers, 100, salary, 10000

**Below is another way of writing the code to read the data:**

>>> with open('techsal.csv') as data:

... out=data.read()

...

>>> print(out)

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

Once we complete the work, we need to close the file. Otherwise, it is just waste of memory. So, below is the way to close the file. Once you close the file, you cannot read the data. To read it again, you need to open it again.

>>> data.closed

True

>>>

* **Python – Writing into File**

In this tutorial, we will learn how to write into a file in Python. Writing the data into a file can be done by using the "write()" method. The write() method takes string as an argument. Let us see an example of writing the data into the file called "techsal.csv".

**First creating the data in a file called "techsal.csv". Below is the data how it looks.**

designers, 100, salary, 10000

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

**Let us write a line into the same file.**

>>> data=open('techsal.csv','r+')

>>> data.write('\nthis is the new line entry\n')

28

**Let us read the data from the file.**

>>> with open('techsal.csv') as data:

... out=data.read()

... print(out)

...

**Below is the output:**

this is the new line entry

0

programmers, 1000, salary, 15000

Dataadmins, 10, salary, 12000

**Finally closing the file:**

>>> data.close()

* **Python - Class & Objects**

In this tutorial, we will understand about the Classes and Objects in Python. A class is depicted in different syntax as how the functions are defined. Below is the syntax of the Class.

**Syntax:**

class ClassName:

<statement-1>

.

.

.

<statement-N>

The statements inside the class are function definitions and also contain other required statements. When a class is created, that creates a local namespace where all data variables and functions are defined.

**Below is an example of a class.**

>>> class MyFirstClass:

... """ This is an example class """

... data=127

... def f(self):

... return 'hello !! you just accessed me'

...

**Accessing the variable data:**

>>> print(MyFirstClass.data)

**Output:**

127

**Accessing the function f:**

>>> print(MyFirstClass.f)

**Output:**

<function MyFirstClass.f at 0x7f79c1de5158>

**Accessing the doc String:**

>>> print(MyFirstClass.\_\_doc\_\_)

**Output:**

This is an example class

**Object:**

Now let us see how to create an object. Creation of an object is an instance of the class. Below is how we creating an Object of the class MyFirstClass.

>>> x = MyFirstClass()

In the above line of code, we have created an Object for the class "MyFirstClass" and its name is "x".

Just trying to access the object name and it gives you information about object.

>>> x

<\_\_main\_\_.MyFirstClass object at 0x7f79c1d62fd0>

>>> print(x.f)

<bound method MyFirstClass.f of <\_\_main\_\_.MyFirstClass object at 0x7f79c1d62fd0>>

Below is how you can access the attributes like data variables and functions inside the class using the Object name, which return some value.

>>> x.f()

'hello !! you just accessed me'

>>> x.data

127

* **Python – Exceptions**

In this tutorial, we will learn about the handling exceptions in Python. It is quite common that for any program written in any programming language may hit the error during the execution due to any reasons.

Reasons may be the syntactical error, or conditional or operational errors caused due to filesystem or due to the lack required resources to execute the program.

So, we need to handle those kind of exceptions or errors by using different clauses while we do programming.

In Python, we can handle an exception by using the raise exception statement or using the try, except clauses.

**Syntax:**

try:

raise statement

except x:

statement

**Exceptions are 2 types:**

**1. Pre-defined exceptions:**

These are the Exceptions which are existing within the python programming language as Built-in Exceptions. Some of them are arithmetic errors like ZeroDivisionError, FloatingPointError, EOF errors... etc.

**2. User-defined exceptions:**

These exceptions are created by programmer which are derived from Exception class.

**Example of Handling an Exception:**

>>> while True:

... try:

... x = int(input("Please enter a number: "))

... print("The number entered is :", x)

... break

... except ValueError:

... print("Sorry !! the given number is not valid number. Try again...")

...

**Output:**

Please enter a number: abc

Sorry !! the given number is not valid number. Try again...

Please enter a number: apple

Sorry !! the given number is not valid number. Try again...

Please enter a number: a123

Sorry !! the given number is not valid number. Try again...

Please enter a number: 123

The number entered is : 123

**Example by using the Predefined Exception**

>>> try:

... print(100/0)

... except ZeroDivisionError as error:

... print(" we Handled predefined error:", error)

...

**Output:**

we Handled predefined error: division by zero

* **Python - Regular Exp**

Regular Expressions can be used for searching a word or a character or digits from the given data and several patterns. These can be called as RREs, regex patterns. We just need to import the module "re" to work with regular expressions.

**Regular expression syntax:**

\d - Matches any decimal digit [0-9].

\D - Matches any non-digit character [^0-9].

\s - Matches any whitespace character [ \t\n\r\f\v].

\S - Matches any non-whitespace character [^ \t\n\r\f\v].

\w - Matches any alphanumeric character [a-zA-Z0-9\_].

\W - Matches any non-alphanumeric character [^a-zA-Z0-9\_].

**Handling White Spaces:**

\n = Used for new line

\s = Used for space

\t = Used for tab

\e = Used for escape

\f = Used for form feed

\r = Used for carriage return

Let us see the below example which uses the find all method to get the search result of salary from the data given as input to techdata variable.

**Example:**

import re

>>> techdata = '''

... working on design technologies gives you $10000 salary.

... working on programming technologies gives you $15000 salary.

... working on Latest technologies give you $20000 salary.

... '''

>>> salary = re.findall(r'\d{1,10}',techdata)

**Let us print the search result from data.**

>>> print(salary)

['10000', '15000', '20000']

* **Python – Mathematics**

In python, we have the module called "math" which provides the access to mathematical functions which are defined in C programming.

**There are 2 types of modules for mathematical functions.**

1. math - This is used for normal numbers

2. cmath - This is used for complex numbers

**There are different types of mathematical functions available in math module.**

1. Number Functions

2. Power and logarthmic Functions

3. Trigonometric Functions

4. Angular conversion

5. Hyperbolic Functions

6. Special Functions

7. Constants

**Examples of Number Functions:**

>>> from math import ceil, factorial, floor, gcd, fsum, trunc

# ceiling the value

>>> ceil(10.3)

11

>>> ceil(9.9)

10

# Factorial of given value

>>> factorial(3)

6

>>> factorial(10)

3628800

# Floor of value given

>>> floor(10.3)

10

>>> floor(10.9)

10

# calculates the GCD of given numbers

>>> gcd(5,10)

5

>>> gcd(3,7)

1

# Floating point sum of of values

>>> fsum([5,4,5,1])

15.0

# Truncating the values

>>> trunc(9.4)

9

>>> trunc(10.5)

10

>>>

Examples of Power and logarthmic Functions:

>>> math.exp(2)

7.38905609893065

>>> math.log(2,10)

0.30102999566398114

>>> math.log(2,4)

0.5

>>> math.log2(4)

2.0

>>> math.log10(2)

0.3010299956639812

>>> math.pow(2,3)

8.0

>>> math.sqrt(64)

8.0

**Examples of Trigonometric Functions:**

>>> from math import sin, cos, tan

# Values are in radians

>>> sin(30)

-0.9880316240928618

>>> cos(90)

-0.4480736161291701

>>> tan(0)

0.0

**Examples of Angular conversion:**

>>> from math import degrees,radians

>>> degrees(10)

572.9577951308232

>>> radians(572)

9.983283321407566

* **Python - Internet access**

In this tutorial, we will learn about how to access the internet using the python. In python, we will have a module called "urllib" that provides various Objects and functions to access the internet. We can perform many activities using this "urllib" module like accessing the webpage data of any website, sending an email... etc.

Let us try fetching the 100 bytes of code behind the google.com page. Below is the example.

**Example:**

>>> import urllib.request

>>> f = urllib.request.urlopen("http://google.com/")

>>> print(f.read(100).decode('utf-8'))

**Output:**

<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta cont

**Let us try fetching the 500 bytes of code behind the google.com page.**

>>> print(f.read(500).decode('utf-8'))

**Output:**

ent="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/logos/doodles/2017/sitara-devis-97th-birthday-6469056130449408.5-l.png" itemprop="image"><meta content="Sitara Devi&#8217;s 97th birthday" property="twitter:title"><meta content="Celebrating Sitara Devi&#8217;s 97th birthday! #GoogleDoodle" property="twitter:description"><meta content="Celebrating Sitara Devi&#8217;s 97th birthday! #GoogleDoodle" property="og:description"><meta content="summary\_large\_image" property="twitte

* **Python - Data Compression**

In this tutorial, we will learn about the data compression in Python programming language. In python, the data can be archived, compressed using the modules like zlib, gzip, bz2, lzma, zipfile and tarfile. To use the respective module, you need to import the module first. Let us look at below example.

**Example:**

>>> import zlib

>>> s = b'you learn learnt learning the data daily '

>>> len(s)

41

>>>

>>> t = zlib.compress(s)

>>> len(t)

39

>>>

>>> zlib.decompress(t)

b'you learn learnt learning the data daily '

>>>

>>> zlib.crc32(s)

2172471860

>>>