

QualityThought Python Training

Saturday, August 25, 2018

Input(), comments

```
Input()
-----
>>> a = input()
12
>>> print(a)
12

>>> age = input("Enter your age \n")
Enter your age
28
>>> print(age)
28
```

COMMENTS:

Python Comments:

=====

1. Single line comment ----> #
2. Multi line comments ----> ''' or """

Command Line Arguments:

While running program arguments which are passing through command line are called as command line arguments .

Here arguments are separated by space.

```
>> python add.py 10 20
```

* These arguments are stored by default in the form of strings in a list with name argv. this is available in sys module.

```
argv[0] --> add.py
```

```
10 --> 10
```

```
20 --> 20
```

```
cmd_line.py
```

=====

```
import sys
```

```
a = int(sys.argv[1])
```

```
b = int(sys.argv[2])
```

```
print(type(a))
```

```
print(type(b))
```

```
add = a + b
```

```
print(add)
```

About Me

Phani

[View my complete profile](#)

Blog Archive

▼ 2018 (5)

▼ August (5)

[Input\(\), comments](#)

[Command line arguments](#)

[Data types](#)

[print\(\), variables](#)

[Module-1](#)

Posted by Phani at 8:51 PM No comments:



Comand line arguments

Command Line Arguments:

While running program arguments which are passing through commnd line are called as command line arguments .

Here arguments are separated by space.

```
>> python add.py 10 20
```

* These argumens are stored by default in the form of strings in a list with name argv. this is available in sys module.

```
argv[0] --> add.py
10 --> 10
20 --> 20
```

```
cmnd_line.py
=====
```

```
import sys
```

```
a = int(sys.argv[1])
b = int(sys.argv[2])
```

```
print(type(a))
print(type(b))
```

```
add = a + b
print(add)
```

NOTE : default it will take string so we have convert to integer

execution :

```
C:\Users\welcome\Desktop>python cmnd_line.py 10 20
```

Posted by Phani at 8:50 PM No comments:



Data types

Python Datatypes :

=====

A datatype represents the type of data stored into a variable or memory.

Buitin datatypes -- Already available in python

User defined datatypes -- Datatypes created by programmers

1. Built in datatypes :

- * None Type
- * Numaric Types --> int,float,complex
- * Sequences --> str,bytes,bytearray,list,tuple,range
- * Sets --> set,frozenset
- * Mappings --> dict

Determine datatype :

```
type(variable name)
```

None :

'None' datatype represents an object that does not contain any value.

In java - NULL

In Python - None

```
>> print(type(None))
>>
```

```
>>> def calc():
...     a=10
...     b=20
...     c=a+b
...
>>> calc()
>>> res = calc()
>>> type(res)
<class 'NoneType'>
```

```
>>> if(res==None):
...     print("does not return any values")
...
does not return any values
```

Numaric datatypes :

1. int
2. float
3. complex

int :

- * Int represents an integer number.
- * It is number with out decimal part and fraction part
- * There is no limit for size of int datatype. It can store any integer number conveniently.

Eg :

[illegible]

Float :

- * Float represents floating number
- * A floating number contains decimal part

Eg:

```
>>> a = 223.345
>>> type(a)
<class 'float'>
>>> a = 22e5
>>> print(a)
2200000.0
```

Converting the datatype explicitly :

```
>>> x = 23
>>> type(x)
<class 'int'>
>>> b = float(x)
>>> print(b)
23.0
>>> c = 23.345
>>> d = int(c)
>>> print(d)
23
```

Complex Datatype :

* complex number is number that is written in the form of a+bj or a+bJ
 a : real part
 b : imaginary part

Eg :

```
>>> a = 1+2j
>>> b = 2+3j
>>> c = a+b
>>> print(c)
(3+5j)

>>> c = 1+2j
>>> d = 1-2j
>>> res = c * d
>>> print(res)
(5+0j)
```

bool datatype :

The bool datatype in python represents boolean values.

```
>>> a = 20
>>> b = 10
>>> print(a>b)
True
>>> print(a<b)
False
>>> c=print(a>b)
True
>>> print(c)
None
>>> c=a>b
>>> print(c)
True

>>> True + True
2
>>> True - False
1

>>> file1 = True
>>> file2 = True
>>> file3 = True
>>> print(file1+file2+file3)
3
```

Sequences in Python :

A sequence represents a group of elements or items.

eg : group of integer numbers will form sequence

There are six types of sequences in python :

1. str
2. bytes
3. bytearray
4. list
5. tuple
6. range

str datatype :

* str represents string datatype.
 * string represents group of characters
 * string enclosed in single quotes or double quotes('' , '')

* string can also be represent in "" or ""(if assign to variable then string otherwise it would be comment only)

eg :

```
>>> print(word)
welcome
>>> word = "welcome"
>>> print(word)
welcome
>>> word = ""welcome""
>>> print(word)
welcome
>>> word = """"welcome""""
>>> print(word)
welcome
```

```
for = 'Quality Thought!'
```

SyntaxError: invalid syntax (for is key word, variable name should not be key word)

```
name = 'Quality Thought!'
```

```
print(name)      # Prints complete string
print (name[0])  # Prints first character of the string
print (name[2:5]) # Prints characters starting from 3rd to 5th
print (name[2:])  # Prints string starting from 3rd character
print (name * 2)  # Prints string two times
print (name + "INSTITUTE") # Prints concatenated string
```

```
Quality Thought!
Q
ali
ality Thought!
Quality Thought!Quality Thought!
Quality Thought!INSTITUTE
```

Note : explain about file processing , file name contain date

```
>>> name[0:4]
'Qual'
>>> name[0:6:2]
'Qai'
>>> print(name[0:4])
Qual
>>> print(name[0:6:2])
Qai
```

There is no char datatype like C in python.

```
>>> ch = 'A'
>>> type(ch)
<class 'str'>

>>> ch = 'A'
>>> type(ch)
<class 'str'>
>>> name = "ravi"
>>> name[0]
's'
```

bytes datatype :

The bytes datatype represents a group of byte numbers just like an array does.

* should be 0 to 255

* Does not support negative numbers

```
>>> a = [12,23,45]
>>> x = bytes(a)
>>> type(x)
```

```

<class 'bytes'>
>>> x[0]=55
Traceback (most recent call last):
File "<pyshell#120>", line 1, in <module>
x[0]=55
TypeError: 'bytes' object does not support item assignment
>>> a = [12,23,45,345]
>>> x = bytes(a) x
Traceback (most recent call last):
File "<pyshell#122>", line 1, in <module>
x = bytes(a)
ValueError: bytes must be in range(0, 256)
>>> a = [12,23,45,-23]
>>> x = bytes(a)
Traceback (most recent call last):
File "<pyshell#124>", line 1, in <module>
x = bytes(a)
ValueError: bytes must be in range(0, 256)

```

bytearray datatype :

* The bytearray datatype is similar to bytes data type. The difference is that the bytes type array can not be modified but bytearray can be modified

```

>>> a = [12,23,34,54]
>>> x = bytearray(a)
>>> print(x[0])
12
>>> x[0] = 55
>>> for i in x:
print(i)
55
23
34
54

```

List datatype :

List : Store different data type elements, can grow dynamically.

* List represent in [] and elements separated by ,

```

>>> a= [101,"ravi",'NRT']
>>> b =[102,"raju","HYD"]
>>> a[0]
101
>>> a[0:2]
[101, 'ravi']
>>> b[:3]
[102, 'raju', 'HYD']

>>> print(a)
[101, 'ravi', 'NRT']
>>> type(a)
<class 'list'>

>>> print(a)
[101, 'ravi', 'NRT']
>>> a[0]
101
>>> a[0] = 111
>>> print(a)
[111, 'ravi', 'NRT']

```

list elements can be modified

```

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = ['john']

```

```

print(list)      # Prints complete list
print(list[0])   # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:])  # Prints elements starting from 3rd element

```

```
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
This produce the following result -
```

```
['abcd', 786, 2.23, 'john', 70.2000000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.2000000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2000000000000003, 123, 'john']
```

tuple datatype :

- * Tuple is similar to list.
- * Can contain different datatypes of elements
- * Represents ()
- * Differene between list and tuple is
can not modify the tuple so tuple is read only list.

```
>>> account_details = (101,"ravi","NRT")
>>> type(account_details)
<class 'tuple'>
>>> account_details[0]
101
>>> account_details[1]
'ravi'
>>> account_details[1] = "raju"
Traceback (most recent call last):
  File "<pyshell#171>", line 1, in <module>
    account_details[1] = "raju"
TypeError: 'tuple' object does not support item assignment
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
```

```
print(tuple)      # Prints complete list
print(tuple[0])   # Prints first element of the list
print(tuple[1:3]) # Prints elements starting from 2nd till 3rd
print(tuple[2:])  # Prints elements starting from 3rd element
print(tinytuple * 2) # Prints list two times
print(tuple + tinytuple) # Prints concatenated lists
```

range data type :

- * range data type represents sequence of numbers.
- * The numbers in range are not modifiable
- * Generally range is used for repeating a for loop for specific number of times.

```
>>> a = range(10)
>>> print(a)
range(0, 10)
>>> type(a)
<class 'range'>
>>> for i in a:
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
>>> r = range(10,30,3)
>>> for i in r:
    print(i)
```

```
10
13
16
19
22
25
28
```

```
>>> r = range(10,30,3)
>>> for i in r:
    print(i)
```

```
10
13
16
19
22
25
28
```

Sets :

A set is an unordered collection of elements much like a set in mathematics.

- * Order of elements is not maintained. It means elements may not appear in the same order as they entered in to set.

- * Set does not accept duplicate elements

- * Two types of sets

1. set datatype
2. frozenset datatype

set datatype :

- * set elements should separated with ,
- * set always print only unique elements.

```
>>> s = {10,20,30,40,50}
>>> print(s)
{40, 10, 50, 20, 30}
>>> type(s)
<class 'set'>
>>> s = {10,10,20,20,30,30}
>>> print(s)
{10, 20, 30}
```

```
>>> str1 = set("srinivas")
>>> print(str1)
{'r', 'v', 'n', 'a', 's', 'i'}
```

```
>>> str2 = list(str1)
>>> print(str2)
['r', 'v', 'n', 'a', 's', 'i']
>>> print(str2[0])
r
>>> print(str2[1])
v
```

```
>>> s = {10,20,30,40}
>>> s.update([50])    // adding element to set
>>> print(s)
{40, 10, 50, 20, 30}
>>>
>>> print(s)
{40, 10, 50, 20, 30}
>>> s.remove(50)      // remove element from set
>>> print(s)
{40, 10, 20, 30}
```


frozenset datatype:

- * Create frozenset by passing set data
- * Can not be modified (update and remove methods will not work)

```
>>> s = {50,60,70,80,90}
>>> fs = frozenset(s)
>>> type(fs)
<class 'frozenset'>
>>> print(fs)
frozenset({80, 50, 70, 90, 60})
>>> s = {50,50,60,60,70}
>>> fs1 = frozenset(s)
>>> type(fs1)
<class 'frozenset'>
>>> print(fs1)
frozenset({50, 60, 70})
```

Mapping Type :

* map represents a group of elements in the form of key values pairs so that when key is given will retrieve a value.

* The 'dict' datatype is an example of map

* dict represents dictionary that contains of pair of elements first one is Key and second one is Value

* key and value separated by ':'

```
>>> d = {101:"Ram",102:"Ravi",103:"Rani"}
>>> print(d)
{101: 'Ram', 102: 'Ravi', 103: 'Rani'}
>>> d[101]
'Ram'
>>> d[102]
'Ravi'
>>> type(d)
<class 'dict'>
```

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
```

```
tinydict = {'name': 'john', 'code': 6734, 'dept': 'sales'}
```

```
print(dict['one'])    # Prints value for 'one' key
print(dict[2])       # Prints value for 2 key
print(tinydict)      # Prints complete dictionary
print(tinydict.keys()) # Prints all the keys
print(tinydict.values()) # Prints all the values
```

Difference between List and Tuple:

1. Literal

```
someTuple = (1,2)
someList = [1,2]
```

2. permitted operations

```
b = [1,2]
b[0] = 3    # [3, 2]
```

```
a = (1,2)
a[0] = 3    # Error
```

```
>>>
>>> a = [1,2]
```

```
>>> a = a + [3]
>>> print(a)
[1, 2, 3]
```

```
>>> a = (1,2)
>>> a = a + (3,)
>>> print(a)
(1, 2, 3)
```

you could add new element to both list and tuple with the only difference that you will change id of the tuple by adding element

```
a = (1,2)
b = [1,2]
```

```
id(a) # 140230916716520
id(b) # 748527696
```

```
a += (3,) # (1, 2, 3)
b += [3] # [1, 2, 3]
```

```
id(a) # 140230916878160
id(b) # 748527696
```

2. User defined datatypes :
array, class or a module is user defined datatypes

Posted by Phani at 7:40 PM No comments:



Monday, August 6, 2018

print(), variables

Python print()

Print Statement :

eg:

```
>>print('Welcome to python')
>>print('Python is OpenSource')
```

```
>>print("welcome to python")
>>print("Python is OpenSource")
```

above two are valid

Below are invalid.

```
>> print("welcome to python")
>>print("Python is OpenSource")
```

Error due to indentation

** Statement should start in first column

** Printing double quotes with in string

```
>>> print("Welcome to \"QualityThought\" ")
Welcome to "QualityThought"
>>> print("Welcome to \'QualityThought\' ")
Welcome to 'QualityThought'
```

** Printing multiple times

```
>>>print("QualityThought"*5)
QualityThoughtQualityThoughtQualityThoughtQualityThoughtQualityThought
```

```
>>>print("QualityThought\n"*5)
QualityThought
QualityThought
QualityThought
QualityThought
QualityThought
```

Python Variable:

variable :

Think of a number. Any number. Now, before you forget that number, let's store it for later.

When you think of a number, you are holding that value in your head.

If you want to remember it you will write it down on a piece of paper.

And if it's really important, you will put it in a safe place.

In computer science, that safe place is a variable. They're called variables because, well, they're "capable of being varied or changed"

--A variable is a memory location where a programmer can store a value. Example : roll_no, amount, name etc.

--Value is either string, numeric etc. Example : "Sara", 120, 25.36

--Variables are created when first assigned.

--The value stored in a variable can be accessed or updated later.

--No declaration required

--The type (string, int, float etc.) of the variable is determined by Python

--The interpreter allocates memory on the basis of the data type of a variable.

Rules for python variables :

=====

--Must begin with a letter (a - z, A - B) or underscore (_)

```
>>> @account_number = 34525
SyntaxError: invalid syntax
```

```
>>> _account_number=34525
>>> print(_account_number)
34525
```

--Other characters can be letters, numbers or _

-- Must not contain any special characters !, @, #, \$, %

```
>>> a@ = 20
SyntaxError: invalid syntax
>>> a$ = 49
SyntaxError: invalid syntax
```

--Case Sensitive

```
>>> product_name = 'TV'
>>> print(product_name)
TV
>>> print(Product_name)
```

Traceback (most recent call last):

```
File "<pyshell#26>", line 1, in <module>
    print(Product_name)
NameError: name 'Product_name' is not defined
>>> print(PRODUCT_NAME)
```

Traceback (most recent call last):

```
File "<pyshell#27>", line 1, in <module>
    print(PRODUCT_NAME)
NameError: name 'PRODUCT_NAME' is not defined
```

--There are some reserved words which you cannot use as a variable name because Python uses them for other things.

```
>>> for = 10
```

```
SyntaxError: invalid syntax
```

Good Variable Name :

```
=====
```

****Choose meaningful name instead of short name. roll_no is better than rno.**

****Maintain the length of a variable name. Roll_no_of_a-student is too long?**

****Be consistent; roll_no or RollNo**

****Begin a variable name with an underscore(_) character for a special case.**

Multi assignment

```
a=10
```

```
print(a)
```

```
Name='RAVI'
```

```
Age=21
```

```
a = b = c = 1
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

```
a,b,c = 10,20,"QualityThought"
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

Posted by [Phani](#) at [6:45 PM](#) [No comments:](#)



Sunday, August 5, 2018

Module-1

What is Programming Language?

1.A programming language is a set of rules that provides a way of telling a computer what operations to perform.

2.A programming language is a set of rules for communicating an algorithm

3.It provides a linguistic framework for describing computations
provides a way of telling a computer what operations to perform.

Types of computer languages:

Machine Language :

- 1.The fundamental language of the computer's processor, also called Low Level Language.
- 2.All programs are converted into machine language before they can be executed.
- 3.Consists of combination of 0's and 1's that represent high and low electrical voltage.

Assembly Language :

- 1.A low level language that is similar to machine language.
- 2.Uses symbolic operation code to represent the machine operation code.

Eg :

```
LOAD r1,b
```

```
LOAD r2,h
```

```
MUL r1,r2
```

```
DIV r1,#2
```

```
RET
```

High level Language :

1. Computer (programming) languages that are easier to learn.
2. Uses English like statements.
3. Examples are Python, C++, Visual Basic, Pascal, Fortran and

What is Python?

1. Python is an easy to learn, powerful programming language. The application development process much faster and easier
2. The programming language Python was conceived in the late 1980s, and its implementation was started in December 1989 by Guido van Rossum at Netherlands as a successor to the ABC (programming language)
3. Python First release happened in 1991.
4. Python was named for the BBC TV show Monty Python's Flying Circus.

Why python?

1. Easy to understand
2. Beginners language
3. Portable
4. Less lines of code
5. Simple to implement
6. Huge libraries supports

8. Large standard library(numpy, scipy)
9. GUI programming (Tkinter)

Python Implementation alternatives:

1. CPython (standard implementation of python)
2. Jython (Python for java)
3. IronPython (Python for .net)
4. Stackless (Python for concurrency)
5. PyPy (Python for speed)

Python Packages :

1. Web development - Django, Flask frameworks, Pylons, Web2py frameworks.
2. Artificial Intelligence : Scikit-learn, Keras, TensorFlow, OpenCV
3. GUI - Tkinter
4. Desktop Applications : Jython, WxPython
5. Game Development : pygame
6. Testing : Splitter Tool, pytest framework
7. Bigdata : Pydoop, DASK, PySpark Libraries
8. DataScience : NumPy, Pandas, matplotlib, seaborn libraries
9. AWS : boto
10. Robotic process : pyro
11. Web Scraping : BeautifulSoup4, urllib2, mechanize
12. Devops & System Admin : Os, Sys, Shutil, Glob, Subprocess, PathLib, fabric
13. Networking : Twisted, socket, client and server

Who uses Python :

Data engineers, data scientists, System administrators and developers. Python is not industry specific, but task specific—great for data processing, business intelligence, and some application development

Google, YouTube,
Instagram, Dropbox,
Survey Monkey, Quora,
Pinterest, Reddit
Yahoo Maps,

Duration

Posted by Phani at 6:36 PM 1 comment:



[Home](#)

Subscribe to: [Posts \(Atom\)](#)

Simple theme. Powered by [Blogger](#).