**DataTypes - Introduction**

Python Syntax

1. Python statement ends with the token NEWLINE character (carriage return). It means each line in a Python script is a statement.
2. Use backslash character \ to join a statement span over multiple lines,Please note that the backslash character spans a single statement in one logical line and multiple physical lines, but not the two different statements in one logical line.
3. Use the semicolon ; to separate multiple statements in a single line.
4. Expressions in parentheses (), square brackets [ ], or curly braces { } can be spread over multiple lines without using backslashes

*#Point 1*

print('id: ', 1)

​

print('First Name: ', 'Steve')

​

print('Last Name: ', 'Jobs')

id: 1

First Name: Steve

Last Name: Jobs

print(2**+**3)

5

*#point 2*

**if** 100 **>** 99 **and** \

200 **<=** 300 **and** \

**True** **!=** **False**:

print('Hello World!')

Hello World!

**if** 100**>**200:

print("Hello")

*#point 2*

print('Hello') \

print(' World!')

**File "<ipython-input-13-3df794e5dd3f>", line 3**

**print(' World!')**

**^**

**SyntaxError:** invalid syntax

*#point 3*

print('id: ', 1);print('First Name: ', 'Steve');print('Last Name: ', 'Jobs')

​

print("Asha"); *#can use semicolon as statement seperator but not needed and not recomended*

id: 1

First Name: Steve

Last Name: Jobs

Asha

lst **=** [1, 2, 3, 4,

5, 6, 7, 8,9,

10, 11, 12]

lst

Out[18]:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

##Indentation in Python

Leading space or tab at the beginning of the line is considered as indentation level of the line, which is used to determine the group of statements. Statements with the same level of indentation considered as a group or block.

For example, functions, classes, or loops in Python contains a block of statements to be executed. Other programming languages such as C# or Java use curly braces { } to denote a block of code. Python uses indentation (a space or a tab) to denote a block of statements.

Indentation Rules

\*Use the colon : to start a block and press Enter. \*All the lines in a block must use the same indentation, either space or a tab. \*Python recommends four spaces as indentation to make the code more readable. Do not mix space and tab in the same block. \*A block can have inner blocks with next level indentation.

​

c,c**++**

void main()

{

printf("Hello world")

printf("Welcome")

printf("How are you")

**if**(100 **>**200)

{

{

print("100 is lesser")

print("100 is not equal to 200")

}

**else**

print("200 is greater")

}

}

​

**if** 10 **>** 5:

print("10 is greater than 5")

print("Now Checking 20 >10")

**if** 20 **>** 10:

print("20 is greater than 10") *# inner block*

print("20 is not lesser than 10")

print("Enter")

**else**: *# 2nd block starts*

print("10 is less than 5") *# 2nd block*

print("This will never print") *# 2nd block*

10 is greater than 5

Now Checking 20 >10

20 is greater than 10

20 is not lesser than 10

Enter

Clanguage

void main()

{

float a **=** 10.1;

int b **=** 20;

int c **=** a**+**b;

}

{

}

println(c);

{

print(c)

}

}

​

a**=**10

b**=**20.1

d **=** "ASha"

a **=** "123"

c**=**a**+**b

print(c)

​

**if** 10**>**5

{

print

print

}

**if** a**>**b:

print("A is greater")

print("B is lessfer")

**else**:

print("B is greater")

print("A is lesser")

print("Outer")

*# comments in python*

1. Single line comments

These start **with** *# (hash) symbol and these are non - executable statements*

2. Multi line comments

These can be written using single **or** double triple quotes

'''

comments

comments

'''

​

"""

comments

"""

​

st **=** 'Asha'

st **=**"Asha"

x **=** """Asha"""

x **=** '''Asha'''

​

​

​

*#To find sum of 2 integers - Single line comment*

*#cvcxvc*

​

​

a **=** 70

print(a)

a**=**20

b**=**30

c **=** a**+**b

print(c)

​

50

​

​

a **=** 10

print(type(a))

a **=** 20.4

print(type(a))

a **=**"Asha"

print(type(a))

​

a **=** 70

print(id(a),a)

a **=** 80

print(id(a),a)

<class 'int'>

<class 'float'>

<class 'str'>

1578133680 70

1578133840 80

a **=** 70

print(a)

a **=** 80

print(a)

​

a**=** 70

1000**--->** a**--->** 80

print(a)

a **=**80

1000**--->** a**--->** 80

print(a)

​

a **=** 70

​

1000**-->**70

a**=** 80

​

2000**--->**a**-->**80

print(a)

70

80

a **=** 70

print(id(a),a)

a **=** 80

print(id(a),a)

1611163824 70

1611163984 80

​

​

​

'''This program is to find sum of

2 interger'''

a**=**20

b**=**30

c **=** a**+**b

print(c)

50

**def** function():

"""This program is to find

sum of 2 integers"""

a **=** 20

b**=**30

c **=** a**+**b

​

print(c)

​

​

​

50

The triple double quotes or triple single quotes are actually not multiline comments but they are regular strings with the exception that they can span multiple lines .That means memory will be alloted to these string internally.

If these strings are not assigned to any variable then they are removed from memory by the garbage collector and hence can be used as comments

​

So using """ or ''' are not recomended for comments in python people since they internally occupy memory and would waste time of the interpreter since the interpreter has to check them.

​

If we write strings inside """ or ''' and if these strings are written as first statements in a module , function, class or method then these strings are called as documentation strings or doc strings. These doc strings are useful to create an API documentation file from a python program.

​

An API documentation file is a text file or html file that contains description of all features of software , language or a product .

​

Steps to create API documentation using doc strings

def Add(x,y):

  """This function takes two paramters which are numbers and find their sum

    It displays the sum as result.

  """

  print('Sum=',(x+y))

Let this code be in a file ex.py

now compile the file using python command , since we need to generate API documentation , we need to use module called pydoc

python -m pydoc -w ex

​

Observation is their is no mention of .py extension .

-m indication module is used and the module used is pydoc , -w indicates html file is be generated

​

​

**%%**writefile Google.py

**def** Add(x,y):

"""This function takes two paramters which are numbers and find their sum

It displays the sum as result.

"""

print('Sum=',(x**+**y))

*#call the function*

Add(10,20)

Writing Google.py

**!** python **-**m pydoc **-**w Google

Sum= 30

wrote Google.html

How Does python sees the varaiables

​

int a = 20

float a = 20

​

int a;

a=20

​

int d;

d = 30

​

int a=20 statically typed language

​

int a

a = 20

int a=20

​

​

Dynamically typed lang

a = 20.4

​

a = "Asha"

​

a = 10

​

a = 20

a= 20.4

​

a =20

a = 20

print(type(a))

a ="Asha"

a =20.4

​

int b

b = 20

​

int a

a = 20

​

int a

a=20

a ="ASha"

a=20.3

​

​

a = 20

a = 30.2

​

​

Dynamically typed language

​

b = 20

​

​

​

​

​

​

in other languages

means , in the ram , memory will be allocated for 'a' and in that memory value 10 will be stored.

if we change the value as 20, then in the same memory where a is pointing , the value will be changed to 20

​

if we specify

​

int b=a

then new memory box for b will be created and the value of a is copied into that.

​

How ever in Python a variable is seen as a tag(or name) that is tied to some value.

a= 1   -> 1000

​

​

a = "20.5" -->2000

​

means the value '1' is created first in the memory and then it is tagged by the name 'a'

​

1

​

Python considers the values ie 1 or 2 etc as objects . if we change the value of 'a' to new value 2

​

a   -> 20.5   -->2000

​

b -->

then the tag is simply changed to the new value or object and then the value '1' becomes unreferenced object , which will be

removed by the garbage collector.

​

a -> 2   1 is unreference

​

if we assign one variable to other

a = 20

b = a

b=a

​

then it makes new tag 'b' to point to the same object

​

a ->

      2

b ->

​​

​

​

​

**Datatypes in Python**

Python lang has both builtin datatypes and userdefined datatypes

Builtin Datatypes : These are of 5 types

1. None Type

2. Numeric Type

3. Sequences

4. Sets

5. Mappings

**The None Type**

None datatype represents an object that does not contain any value. in other langs it is called as null object. if no value need to be provided then None can be used

**Numeric Types**

To represent numeric values , and these are 3 subtypes

1. int

2. float

3. complex

int datatype

Represents integer number with out any fractional part

e.g a = 57

float datatype

Represents floating point numbers, floating point numbers can also be written in scientific notations using 'e' or 'E'

e.g a = 55.27

b= 22.52e3

complex datatype

Represents the number in the form of a+bj i.e real part and imaginary part

e.g c1= 1 + 5j

You can convert one datatype to other by type conversion or coercion method

​

*# x = 15.56*

*# print(int(x))*

​

*# print(type(x))*

​

*# print(type(x))*

*# # print(int(x)) #converting float to int*

​

*# y = int(x)*

*# print(type(y))*

​

*# str1 ="abc"*

*# print(type(str1))*

​

*# g = int(str1)*

*# print(type(g))*

​

*# print(g)*

​

​

​

*# print("Hello World","to the world","of python")*

​

st **=** 16.23

print(type(st))

st **=** int(st)

print(type(st),st)

<class 'float'>

<class 'int'> 16

num **=** 15.24

print(type(num))

v **=**int(num) *#converting int to float*

print(type(v))

<class 'float'>

<class 'int'>

n **=** 10

print(complex(n)) *#converting int to complex*

print(n)

(10+0j)

10

Under numeric types you can represent binary number prefixed with 0b or 0B

e.g 0b1101

Hexadecimal values can be written by prefixing 0x or 0X

e.g:0xA180

Octal values can be written by prefixing 0o or 0O

e.g: 0o145

bool datatype - represents boolean values , there are 2 boolean values True and False and internally these values are represented as 0 and 1

a **=** 10

b**=**20

print(a**<**b)

True

Extra information:

C has two kinds of integers; short and long.

​

A short integer is, at least, 16 bits. So, on a 16-bit machine, it coincides with the long integer format.

The short integer format ranges from -32,767 to 32,767 for the signed version and from 0 to 65,535 for the unsigned.

​

​

Java supports only signed versions of integers. They are:

​

byte (8 bits)

short (16 bits)

int (32 bits)

long (64 bits)

So, with the long integer format we can reach 2^{64-1} as with C on a 64-bit machine but,

this time, on every machine architecture.

But with Java, we can go further with a little hack so we can represent very large integer numbers

through the BigInteger class library. This library combines arrays of smaller variables to build up huge numbers.

The only limit is the physical memory, so we can represent a huge, but still limited, range of integers

​

​

Python directly supports arbitrary precision integers, also called infinite precision integers or bignums, as a top-level construct.

This means, as with Java BigInteger, we use as much memory as needed for an arbitrarily big integer. So, in terms of programming language, this question doesn’t apply at all for Python, cause the plain integer type is, theoretically unbounded.

​

What is not unbounded is the current interpreter’s word size, which is the same as the machine’s word size in most cases. That information is available in Python as sys.maxsize, and it’s the size of the largest possible list or in-memory sequence, which corresponds to the maximum value representable by a signed word.

​

On a 64-bit machine, it corresponds to 2^{64-1} = 9,223,372,036,854,775,807.

​

So there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has, as are all things, but beyond that an integer can be as long as you need it to be:

​

>>> print(123123123123123123123123123123123123123123123123 + 1)

123123123123123123123123123123123123123123123124

​

​

*# import sys*

*# print(sys.maxsize)*

2147483647

**Sequences in Python**

Sequences represents a group of elements or items in sequence, there are basically 6 subtypes

1. str

2. bytes

3. bytearray

4. list

5. tuple

6. range

str datatype

​

  str means string datatype , which is sequence or group of characters enclosed either in single or double quotes

  e.g: str = "welcome"

        str = 'welcome'

      str = """Welcome"""

      str = '''Welcome'''

  There is no character datatype , we can get characters from string through slicing

bytes datatype

  Represents group of byte numbers just like an array. A byte is positive number from 0 to 255 and it cannot store negative numbers

  we can not modify or edit any element in the byte type array , we can loop to print the bytes

bytearray datatype

  It is similar to byte type but it can be editable or modifiable

list datatype :

  list represents group of elements , the elements are modifiable

  the main difference between list and array is that list can store different types of

  elements but array can store only one type of elements

  lists can grow dymanically in memory where as arrays size is fixed

  lists are represented using []

  e.g: lst = [10,20,'mary',-1,20.3]

  print(lst[1])

  lst[0]= 99

  array = ["First","Seoncd","third"]

  list()

tuple datatype

  A tuple is similar to list which contains group of different types of elements seperated by commas, but it can not be modifiable

  it is created using ()

  e.g: tp1 = (10,20,'mary')

      tp1[0] = 99 # will give error

range datatype:

      It represents a sequence of numbers . the numbers in the range are not modifiable

      1 - to 10

      range(startindex,endindex,incement)

      range(0,10,1)

      0,10,2

      0,2,6,8

      e.g r = range(0,10) # will create numbers starting from 0 to 9 and we can display using loop,

      0,1,2,3,4,5,6,7,8,9

​

​

​

​

print(range(0,10,1))

​

print(range(0,10,1))

​

*# for item in range(0,10):*

*# print(item)*

print(tuple(range(10,20,2)))

range(0, 10)

range(0, 10)

(0, 2, 4, 6, 8)

range(10)

Out[4]:

range(0, 10)

*# Sets*

​

A set **is** an unordered collection of elements , the order **is** **not** maintained **in** the sets ,it means the elements may

**not** maintain the same order **as** we see.

There are 2 subtypes of sets

1. set datatypes

2. frozenset datatypes

it **is** created by {} brackets , it allows duplicate values **while** entering , but when set **is** getting created then

duplicate values will **not** be stored **or** displayed . The values **in** the set datatype can be modified.

s **=** {10,20,50,20}

print(s[0]) **--** **not** possible

​

​

​

frozenset **-** **is** similar to set datatype , but the difference **is** the values can **not** be modified.

s **=**{50,20,50,60}

fs **=** frozenset(s) *# the set is made as frozenset*

s **=** set(fs)

s **=** set() *#empty set*

y **=** {} *#empty dict*

​

print(type(s))

print(type(y))

​

x **=** {} *#empty set/empty dict*

y **=** {}

​

*# objects can be created by using the constructor of each of the class*

x **=** set()

​

y **=** dict{}

​

x **=** list[]

​

x **=**tuple()

​

<class 'set'>

<class 'dict'>

​

**Mapping Types**

A Map represents a group of elements in the form of key and value pairs, so that we can retrive the value by specifying key dict datatype is example of map.

it is created by {key:value} with colon inside for key and value.

e.g

d ={10 : 'kamal',20:'vimal'}

print(d[10])

**Literals :**

Literal is constant value that is stored into a variable

a = 15 where a is a variable where as 15 is literal , since it is integer it is called as integer literal

1. Numeric literal
2. Boolean literal
3. String literal

Datatype of the variable can be deterimed using type() function

Few of the userdefined datatypes in python are class or a module

**Constants**

Constants are similar to that of varibales , but it can not be modified. once defined its value can not be changed. in other lang we have keywords to represent constant values , but in python it is convention only

we refer constant varaible by writing it in all capital letters and it more than one word is present then it is seperrated by

underscore

e.g : MAX\_VALUE

it is just convention , it can be changed also. usually in python all these type of constant variables are created in one single file and called it as constant.py and it will be imported to the program where it is used . so that the chances of changing the values will be reduced.

PI **=**2.42 *#constat*

constants.py