DATATYPES IN PYTHON

- Comments increase readability or understand-ability of program.

Comments in Python

1. **Single-line comment**
2. **Multi-line comment**

1. **Single Line Comment**

   #To add numbers
   a=10 #store 10 into variable a

2. **Multi-line Comment or Block Comments**

   Triple double quotes “”” or triple single quotes ‘’’

   For Example:

   “””
   This is python lecture.
   Next lecture is on Thursday.
   “””
   ‘’’
   This is python lecture.
   Next lecture is on Thursday.
   ‘’’

DOCSTRINGS

- If we write strings inside “”” or ‘’’ and if these strings are written as first statements in the module, function, class or a method, then these strings are called **documentation strings or doc-strings.**
- Useful to create an API (Application Programming Interface) documentation file.

VARIABLE AND INDENTIFIERS

- A **variable** is a named location used to store data in the memory.
- Example: a=10; b=15
- Assignment Operator (=)
- The name given to a variable is called **identifier**.
- We need not to declare the datatype of the variable.

How Python see variables

- A variable is seen as a **tag** (or name) that is tied to some value.
- Python considers the values as ‘objects’.

- For example – number=10; number=1.1
- **In python, we don't actually assign values to the variables. Instead, Python gives the reference of the object (value) to the variable.**
- **Python is a type-inferred language.**

Example 1: Declaring and assigning value to a variable

```
website = "python.org"
print(website)
```

Example 2: Changing the value of a variable

```
website = "python.org"
print(website)
#assigning a new value to website
website = "outlook.office.com"
print(website)
```

DATATYPES IN PYTHON

1. Built-in datatypes
2. Use-defined datatypes

BUILT-IN DATATYPES

1. None Type
2. Numeric Type
3. Sequences
4. Sets
5. Mappings

None Type represents an object that does not contain any value.

Numeric Type int, float, complex

1. **int** Datatype: a=-89
- There is no limit for the size of an int datatype.

2. **Float** datatype: num=14.6

3. **Complex** datatype: a+bj, a – real part of the number and b –imaginary part of the number. The suffix 'j' indicates the square root value of -1.

X= -1-8.5j

For Practice: Python Program to display the sum of Two Complex Numbers.

Bool Datatype True as 1 and False as 0 (“”)

Example:

```
a=36>15
print(a)

a=15>36
print(a)
```

Sequences in Python

- Represents a group of elements
- Six-types of sequences in Python.

  - **str**
  - **bytes**
  - **bytearray**
  - **list**
  - **tuples**
  - **range**

1. **str datatype**

- represents string datatype.

Example:

```
str ="Welcome"
str ='Welcome'

str1="""This is Python's Lecture
which discusses all the topics of python."""

str2='''This is Python's Lecture
which discusses all the topics of python.'''

str="""This is 'Python Lecture' session"""
print(str)

str='''This is "Python Lecture" session'''
print(str)
```

- **slice operator** represents square brackets [and] to retrieve pieces of a string.

Example:

```
s='Welcome to Python Lecture'
print(s)

print(s[0]) # display's 0th character from s

print(s[3:7]) #display's from 3rd to 7th characters

print(s[11:]) #display from 11th character onwards till end

print(s[-1]) #display first character from the end
```

- The **repetition operator = '*' symbol**: repeat the string
- S*n repeats the string for n times.

2. bytes Datatype

- Represents a group of byte numbers.
- Store numbers in the range from 0 to 255 (+eve integer).
- It cannot store –eve numbers**.**
- We cannot modify or edit any element.

Example:

```
elements = [10, 26, 0, 45, 15] #this is the list of byte numbers

x= bytes(elements) #convert the list into byte bytearray

print(x[0]) #display 0th element
```

3. Bytearray Datatype

- array can be modified.

Example:

```
elements = [10, 26, 0, 45, 15] #this is the list of byte numbers
x= bytearray(elements) #convert the list into bytearray type array
print(x[0]) #display 0th element
print(x[1])

x[0]=80 #replace 0th element by 80
x[1]=99

print(x[0])
print(x[1])
```

4. List Datatype

- Represents group of elements.
- It can grow dynamically in the memory.
- Using square brackets [ ] .
- we can perform slicing operation with list.

Example:

```
list = [10, -30, 5.6, 'Ram', "Pooja"]
print(list)

#use of slice operator
print(list[0])

print(list[1:3])

print(list[-2])

print(list*3) #use of Repetition operator
```

5. tuple Datatype

- Enclosed in parenthesis ().
- The list can be modified; it is not possible to modify the tuple.
- Slicing operations can be done on lists are also valid in tuples.

6. range Datatype

- Represents a sequence of numbers.
- The numbers in the range are not modifiable.
- Used for repeating a for loop for a specific number of time.

Example:

```
r= range(10) #range object is created from 0 to 9
for i in r: print(i) #display the number from 0 to 9

r=range(30,40,2) #starting number 30 and an ending number 39. The step size is 2.
for i in r: print(i)

#Create a list with a range of numbers
lst=list(range(10))
print(lst)
```

SETS

- Unordered collection of elements.
- The elements may not appear in the same order as they are entered in the set.
- Does not accept duplicate elements.

1. set datatype
2. frozenset datatype

set datatype

Example:

```
s={10, 20 , 30, 20, 50}

print(s)

ch=set("Hello")

print(ch)
```

#convert a list into set

```
lst=[1,2,5,4,3]

s=set(lst)

print(s)
```

#update() method is used to add elements to a set

```
s.update([50,23])

print(s)
```

#remove() method is used to remove any particular element from a set

```
s.remove(50)

print(s)
```

#Slicing and indexing is not possible

```
print(s[0]) #indexing #Type Error

print(s[0:2]) #slicing #Type Error
```

frozenset datatype

- The elements in the frozenset cannot be modified.
- **Update () and remove ()** methods will not work on frozensets since they cannot be modified or updated.

Example:

```
s={50,40,30,20,90}
print(s)

#create frozenset
fs=frozenset(s)
print(fs)

#passing a string to frozenset()
fs=frozenset("abcdefg")
print(fs)
```

<u>MAPPING TYPES</u>

- A **map r**epresents a group of elements in the form of **key value** pairs so that when the key is given, we can retrieve the value associated with it.
- The **dict datatype** is an example for a map.
- The **'dict'** represents a **'dictionary'** that contains pairs of elements such that the first element represents the key and the next one becomes its value.
- Key and its value separated by the **colon (:).**
- All the elements should be enclosed inside curly brackets {}.

Example:

```
#empty dictionary

d={}


d[10]='Pooja'

d[11]='Ram'

print(d)


d={10:'Pooja', 11:'Ram', 12:'Anup', 13:'Reetu', 14:'Sanjay'}

print(d)
print(d[11])
```

#To retrieve only keys

```
print(d.keys())
```

#To get only Values

```
print(d.values())
```

#update the value of key d[key]=new-value

```
d[10]='Shyam'

print(d)
```

#delete the key and corresponding values

```
del d[11]

print(d)
```