Session 3

Day 2: Python Programming(contd.)

3. complex:

In python, we can use complex data types. The complex number is in the form: a+bj where 'a' is the real part and 'b' is the imaginary part. Incase of python, only j/J can be used in the imaginary part (i cannot be used). Complex data type is important in its use in scientific and complex applicative areas.

Example:

```
a= 5+3j
print (a)
print(type(a))
```

Output:

```
(5+3j)
<class 'complex'>
```

The complex number cannot be written in the form: 5+j3.

The real part of the complex number can also be written in forms other than decimal. For example:

```
a= 0b1101 + 2j
print(a)
```

Output: `13+ 2j`

However, the imaginary part can only be written in decimal form.

```
a= 5 + 0b1101j
print(a)
```

Output:

```
a = 5+0b1101j
            ^
SyntaxError: invalid binary literal
```

Different arithmetic operations can also be carried out in complex data type:

```
a= 5+2j
b=2+2j
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

Output:

```
(7+4j)
(3+0j)
(6+14j)
(1.75-0.75j)
```

If we want to only show real or imaginary part:

```
a= 5+2j
print(a.real)
print(a.imag)
```

Output:

```
5.0
2.0
```

4. bool:

The bool data type in Python represents one of two values: ‘True’ or ‘False’.

```
a= True
print(type(a))
```

Output:

```
<class 'bool'>
```

Python is case sensitive so ‘true’ cannot be used instead of ‘True’.

```
a= true
print(type(a))
```

Output:

```
 a = true
     ^^^^
NameError: name 'true' is not defined. Did you mean:'True'?
```

Bool data type is used to evaluate expressions and determine the truthness of a statement, as follows:

```
a=10
b=20
c=a>b
```

```
print(c)
```

Output:

```
False
```

Python internally treats True as 1 and False as 0. So, arithmetic calculations can also be carried out with boolean data type.

Example:

```
a=True
b=True
print(a+b)
print(type(a+b))
print(type(a))
```

Output:

```
2
<class 'int'>
<class 'bool'>
```

5. string:

String is known as the array of characters. In python specifically, string is anything enclosed by single quotation, double quotation or triple quotation (triple single quotation or triple double quotation).

Example:

```
a='apple'
print(type(a))
```

Output:

```
apple
<class 'str'>
```

There is no concept of char(character) in Python, be it a single character or multiple character, all are strings.
The string with triple double quotation is also known as docstring, which is a special kind of string which lets us print multi-line strings.

For example:

```
a= """ Learning
Python
is
fun """
print(a)
```

Output:

```
Learning
Python
is
fun
```

- **Slicing of string:**

  String slicing in Python is the process of extracting a portion (or "slice") of a string using a specific range of indices. This is useful for accessing substrings within a larger string.

  Syntax:

  ```
  String[start:end+1]
  ```

  Example:

  If we want to slice the word ‘Python’ and ‘Learning Python’:

  ```
  a= "Learning Python is fun."
  print(a[9:15])
  print(a[:15])
  ```

  Output:

  ```
  Python
  Learning Python
  Learning Python is fun.
  ```

Some practice problems of string:

1.

```
a= "Learning Python is "
b="fun"
print(a+b)
print(3*b)
```

Output:

```
Learning Python is fun.
funfunfun
```

2.

```
a="#"
b="Hello"
print(a*5)
print(b)
print(a*5)
print(len(b))
```

Output:

```
#####
Hello
#####
 5
```

3.

```
b="hello"
print(b[0].upper() + b[1:])
print(b[:-1]+b[-1].upper())
```

Output:

```
Hello
hellO
```

- **Typecasting:**

  In python, there are five fundamental data types:
  - int()
  - float()
  - complex()
  - bool()
  - str()

  The conversion among these five fundamental data types, typecasting, is carried out using following programs:

  1. To int():

```
# float to int
a= 15.8
print(type(a))
print(int(a))
```

Output:

```
<class 'float'>
15
```

```
# bool to int
c= True
print(int(c))
```

Output:

```
1
```

Note: complex cannot be changed to int.

```
# string to int.
d= "10"
print(int(d))
```

Output:

```
10
```

Note: string cannot be changed to int unless the string has integer number with base 10.

---

2. To float():

```
#int to float
b=10
print(float(b))
```

Output:

```
10.0
```

```
#bool to float
c=True
print(float(c))
```

Output:

```
1.0
```

Note: complex cannot be changed to float.

```
#string to float
d = "10"
print(float(d))
```

Output:

```
10.0
```

Note: string cannot be changed to float unless the string has integer number with base 10.

---

3. To complex():

```
#int to complex
b=10
print(complex(b))
```

Output:

```
10+0j
```

```
#float to complex
b=10.5
print(complex(b))
```

Output:

```
10.5 + 0j
```

```
#bool to complex
b=True
c=False
print(complex(b))
print(complex(c))
```

Output:

```
1 + 0j
0 + 0j
```

```
#string to complex
a="10"
print(complex(a))
```

Output:

```
10 + 0j
```

Note: string cannot be changed to float unless the string has integer number with base 10.

---

4. To bool():

```
#int to bool
print(bool(0))
```

Output:

```
False
```

```
#float to bool
print(bool(10.5))  #True for anything other than 0
```
Output:
```
True
```

```
#complex to bool
print(bool(5+2j))
```
Output:
```
 True
```

```
#string to bool
print(bool(""))  #False if empty, True otherwise
```
Output:
```
False
```

```
print(bool("False"))
```
Output:
```
True
```

---

5. To string():
```
#int to string
a=10
b=str(a)
print(b)
print(type(b))
```
Output:
```
10
<class 'str'>
```

```
#float to string
a=10.5
b=str(a)
print(b)
print(type(b))
```
Output:
```
10.5
<class 'str'>
```

```
#complex to string
```

```
a=10 + 5j
b=str(a)
print(b)
print(type(b))
```

Output:

```
 10 + 5j
<class 'str'>
```

```
#bool to string
a=True
b=str(a)
print(b)
print(type(b))
```

Output:

```
True
<class 'str'>
```

- **Fundamental data types vs Immutability:**

All fundamental data types are immutable.
Once we create an object, we cannot perform any changes in that object. If we try to change, then with those changes, a new object is created. This is known as immutability of fundamental data types.

For example:
We know, everything in Python is an object.
Let ,

```
a=10
```

Here, an object is created with value 10 and reference to variable ‘a’.

```
a=a+2
```

Here, instead of replacing the value 10 with 12, a new object with value 12 is created altogether and the reference of variable ‘a’ is changed from 10 to 12.
Then, the object with value 10 is destroyed by the GC(Garbage Collector).

To understand this concept even further, let us consider the following example:

```
a=10
print(id(a))
a=a+2
```

```
print(id(a))
```

Output:

```
4352142448
4352142512
```

Here the id of ‘a’ changes as a new object is created.

If there are two variables with the same value, then Python treats those two objects as the same and just references the object to two or more variables having the same value. For example:

```
a=10
b=10
print(id(a))
print(id(b))
print(a is b)
```

Output:

```
4380355696
4380355696
True
```

Here, the id of both ‘a’ and ‘b’ are the same as they have the same values.