Complex Data Type in Python

In Python, complex data types provide a versatile and powerful way to represent complex numbers. A complex number comprises a real part and an imaginary part, denoted as a + bi, where a is the real part, b is the imaginary part, and i is the imaginary unit.
a complex number is formatted as "a + bi." In Python, the imaginary part of a complex number is indicated using the suffix **'j'** or **'J'** instead of **'i.'** For instance, 3 + 3j illustrates a complex number in Python.

Example:

```
a = 5+6j
print(a)
print(type(a))
```

Output:

```
(5+6j)
<class 'complex'>
```

Operation between Complex number

```
x = 5+7j
y = 7-6j

print(x+y)
print(x-y)
print(x*y)
print(x/y)
```

Output:

```
(12+1j)
```

```
(-2+13j)
(77+19j)
(-0.08235294117647059+0.9294117647058823j)
```

String Slicing in Python

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end.

Using the List/Array slicing [::] method

In Python, indexing syntax can be used as a substitute for the slice object. This is an easy and convenient way to slice a string using list slicing and Array slicing both syntax-wise and execution-wise. A start, end, and step have the same mechanism as the slice() constructor.

Below we will see string slicing in Python with examples.

Syntax:

```
arr[start:stop]        # items start through stop-1
arr[start:]            # items start through the rest of the array
arr[:stop]             # items from the beginning through stop-1
arr[:]                 # a copy of the whole array
arr[start:stop:step]   # start through not past stop, by step
```

Example:

```
String = "ILOVEMYCOUNTRYNEPAL"
print(String[:3])
print(String[1:5:2])
print(String[-1:-12:-2])
print(String[::-1])
```

```
print(String[14:19])
print(String[5:])
```

Output:

```
ILO
LV
LPNRNO
LAPENYRTNUOCYMEVOLI
NEPAL
MYCOUNTRYNEPAL
```

Type Cast between fundamental Data Types in Python

1. int()

   -It is a function to cast other data types to integer type.

   Syntax:

```
int(value, base=10)
```

value : The value you want to convert to an integer

base: Optional. The base of the number in string format(default is 10)

Data types the can be converted to int:

1. Float: A floating-point number can be converted to an integer by truncating the decimal part.

Example:

```
float_num = 7.89
int_num = int(float_num)  # Output: 7
```

2. Boolean: True is converted to 1 and False is converted to 0.

Example:

```
bool_value = False
int_num = int(bool_value) # Output: 0
```

3. String: A string that represents a valid integer or float can be converted to an int. The string must not contain invalid characters or be in the wrong format.

Example:

```
string_num = "56"
int_num = int(string_num) # Output: 56
```

4. Hexadecimal: A string representing a hexadecimal number can be converted to an integer by specifying the base (16).

Example:

```
hex_str = "0x1A"
int_num = int(hex_str, 16)  # Output: 26
```

2. float()

   The float() method in Python is used to convert a value to a floating-point number (decimal number). The value can be a string, integer, or other types that can be interpreted as a float.

   Syntax:

   ```
   float(value)
   ```

   value: The value you want to convert to a floating-point number.

   Data types that can be converted to float:

   1. Integer: An integer can be converted to a float, simply appending a .0 to the value.

      ```
      int_num = 25
      float_num = float(int_num)  # Output: 25.0
      ```

   2. String: A string that represents a valid floating-point number can be converted to a float. The string must follow the standard float format.

      Example:

      ```
      string_num = "76.89"
      float_num = float(string_num)  # Output: 76.89
      ```

3. Boolean: True is converted to 1.0, and False is converted to 0.0.

```
Example:
bool_value = False
float_num = float(bool_value)  # Output: 0.0
```

4. Complex: Only the real part of a complex number can be converted to a float. The imaginary part is ignored.
   Example:

```
complex_num = 5 + 3j
float_num = float(complex_num.real)  # Output: 5.0
```

Conversion restrictions:

- Strings must represent valid numbers; otherwise, ValueError will be raised.
- Complex numbers must be handled with care as only the real part is extracted.

3. bool()

In Python, the bool() method is used to convert a value to a boolean, which can either be True or False. In general, most values evaluate to True, while specific values like **0**, **None**, and **empty collections (e.g., [], (), {}, etc.)** evaluate to False.

Examples of using bool():

```
# Converting an integer to a boolean
int_num = 10
bool_value = bool(int_num)  # Output: True

# Converting a string to a boolean
string_value = "Hello"
bool_value = bool(string_value)  # Output: True
```

```
# Converting an empty list to a boolean
empty_list = []
bool_value = bool(empty_list)  # Output: False
```

4. str()

In Python, the str() method is used to convert a value to a string. Below are examples of how various data types can be converted to strings.

```
# Integer to String
int_num = 123
string_value = str(int_num)  # Output: '123'

# Float to String
float_num = 45.67
string_value = str(float_num)  # Output: '45.67'
```

```
# Boolean to String
bool_value = True
string_value = str(bool_value)  # Output: 'True'

# List to String
list_value = [1, 2, 3]
string_value = str(list_value)  # Output: '[1, 2, 3]'

# None to String
none_value = None
string_value = str(none_value)  # Output: 'None'
```

5. complex()

   In python complex() method is used to convert other data types into complex data types.

   Examples:

```
# Integer to Complex
int_num = 5
complex_value = complex(int_num)  # Output: (5+0j)

# Float to Complex
float_num = 4.5
complex_value = complex(float_num)  # Output: (4.5+0j)

# String (representing a number) to Complex
string_value = "3.14"
complex_value = complex(string_value)  # Output: (3.14+0j)

# Two numbers (real and imaginary) to Complex
real = 3
imag = 4
complex_value = complex(real, imag)  # Output: (3+4j)
```

```
# Boolean to Complex
bool_value = True
complex_value = complex(bool_value)  # Output: (1+0j)

# Complex to Complex
complex_num = 2 + 3j
complex_value = complex(complex_num)  # Output: (2+3j)
```

Note:

- For integers, floats, booleans, and strings representing numbers, the imaginary part defaults to 0j unless specified.
- None cannot be converted to a complex number directly. It will raise a TypeError.
- A complex number remains the same when passed to the complex() function.