Python Data Types - Seesion1

- Python is a forerunner in the technical domain, and the reason behind its success is its vast community.
- These communities are super active, trying their best to keep the language simple and updated, making learning easy.
- Python offers ease to the developers with the fast development speed of coding and readability.
- **All this is possible because of the different data types of the Python programming language.**

Libraries, modules, and functions are the ones that make Python multifaceted. Besides, there are basic Python data types that make a difference in the language's design.

Uses of Python

It can be used in different areas:

- Data Science
- Data Analysis
- Machine Learning
- Data Engineering
- Web Development
- Software Development, and other fields.

Numeric

The Numeric data types in Python comprise integers, floating type numbers aka floats, and complex numbers.

- **Integer** includes positive, negative, or zero. For example, 11, 4, -10, -100, etc. There's no restriction on the length of the integer.
- **Floats** are real numbers usually depicted in decimal form like 2.2, 10.9, etc.
- **Complex numbers** include real as well as imaginary elements like a + by where a and by could be imaginary and real parts. Complex numbers could be 1.15k, 3.0 + 2.5j, etc.

```
# Integers examples
print(123123123123123123123123123123123123123123123123 + 1)
print(type(123123123123123123123123123123123123123123123123 + 1))`
```

Define integer value to indicate a base other than 10:

```
# Octal, defined 0 with o small or capital
print(0o10)
print(type(0o10))
print('################')
# Hexadecimal, defined 0 with x small or capital
print(0x10)
print(type(0x10))
print('################')
# Binary, defined 0 with b small or capital
print(0b10)
print(type(0b10))
print('################')`
```

Floating-Point Numbers

The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character e or E followed by a positive or negative integer

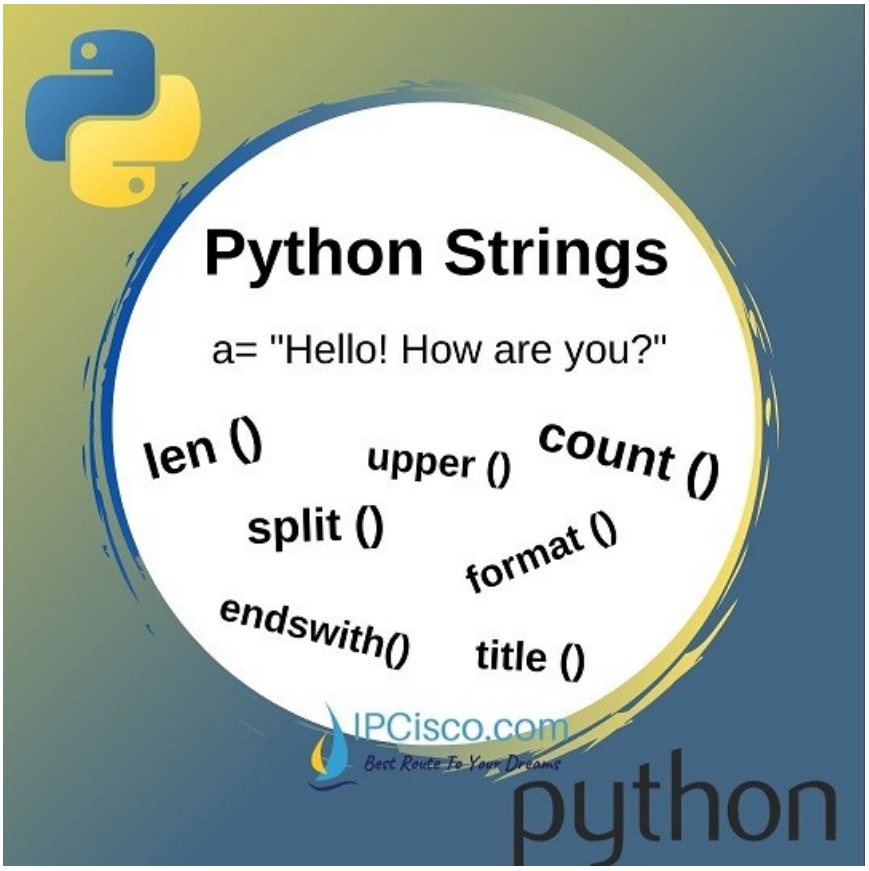
```
# Define floating numbers
print(4.2)
print(type(4.2))
print('################')
# Define floating point with Scientific notation
print(4e7)
print(type(4e7))
print('################')
```

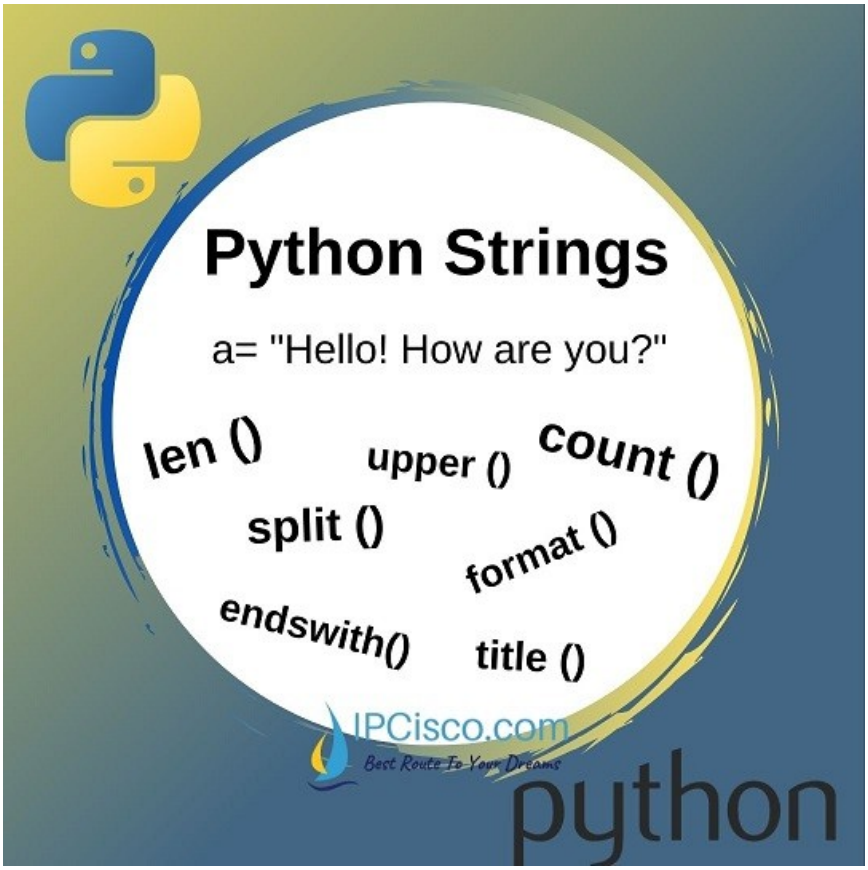
Complex Numbers

Complex numbers are specified as (real part)+(imaginary part)j

```
print(2+3j)
print(type(2+3j))
```

Strings





A string is a sequence of characters. String literals in Python are enclosed by either double or single quotes.my_string = 'Hello'

Python strings are immutable which means they cannot be changed after they are created.

Creation

```
# Use singe or double quotes
my_string = "Hello"
my_string = "I' m  a 'Geek'"
# Escaping backslash
my_string = 'I\' m  a "Geek"'
my_string = 'I\' m a \'Geek\''
print(my_string)

# Triple quotes for multiline strings
my_string = """Hello
World"""
print(my_string)

# Backslash if you want to continue in the next line
my_string = "Hello \
World"
print(my_string)
```

Concatenate two or more strings

```
# concat strings with +
greeting = "Hello"
name = "my name"
sentence = greeting + ' '+ name
print(sentence)`
```

Iterating

```
# Iterating over a string by using a for in loop
my_string = 'Hello'
for i in my_string:
    print(i)
```

Check if a character or substring exists

```
if "e" in "Hello":
    print("yes")
if "llo" in "Hello":
    print("yes")`
```

Useful methods

```
my_string = "     Hello World "

# remove white space
my_string = my_string.strip()
print(my_string)

# number of characters
print(len(my_string))

# Upper and lower cases
print(my_string.upper())
print(my_string.lower())

# startswith and endswith
print("hello".startswith("he"))
print("hello".endswith("llo"))

# find first index of a given substring, -1 otherwise
print("Hello".find("o"))

# count number of characters/substrings
print("Hello".count("e"))

# replace a substring with another string (only if the substring is found)# Note: The original string stays the same
message = "Hello World"
new_message = message.replace("World", "Universe")
print(new_message)

# split the string into a list
my_string = "how are you doing"
a = my_string.split() # default argument is " "print(a)
my_string = "one,two,three"
```

```
a = my_string.split(",")
print(a)

# join elements of a list into a string
my_list = ['How', 'are', 'you', 'doing']
a = ' '.join(my_list) # the given string is the separator, e.g. ' ' between each argumentprint(a)
```

Common Python Data Structures

List: Mutable Dynamic Arrays

- Python's lists are implemented as **dynamic arrays** behind the scenes.
- This means a list allows elements to be added or removed, and the list will automatically adjust the backing store that holds these elements by allocating or releasing memory.

List is a collection data type which is ordered and mutable. Unlike Sets, Lists allow duplicate elements. They are useful for preserving a sequence of data and further iterating over it. Lists are created with square brackets.

Creating a list

```
# Creating a list with []
list_1 = ["banana", "cherry", "apple"]
print(list_1)
type(list_1)
# Creating a list with list - built in function
list_1 = list(["banana", "cherry","apple"])
print(list_1)
type(list_1)
# Define new list
list_1 = ["banana", "cherry", "apple"]
print(list_1)

# Or create an empty list with the list function
list_2 = list()
print(list_2)

# Lists allow different data types
list_3 = [5, True, "apple"]
print(list_3)

# Lists allow duplicates
list_4 = [0, 0, 1, 1]
print(list_4)
```

Access elements
You access the list items by referring to the index number. Note that the indices start at 0In [27]:

```
test_list=[1,2,3,4,8,7,9,66]
test_list
```

```
# Get nd element
var=test_list[1]
var
```

```
# Iterate over the list
for element in test_list:
    print(element)
```

Change items

Just refer to the index number and assign a new value

```
# Lists can be altered after their creation
print(list_1)
list_1[2] = "lemon"
print(list_1)
```

Useful methodsIn

```
my_list = ["banana", "cherry", "apple",[1,2,3,4]]

# len() : get the number of elements in a list
print("Length:", len(my_list))

# append() : adds an element to the end of the list
my_list.append("orange")

# insert() : adds an element at the specified position
my_list.insert(1, "blueberry")
print(my_list)

# pop() : removes and returns the item at the given position, default is the last item
item = my_list.pop()
print("Popped item: ", item)

# remove() : removes an item from the list
my_list.remove("cherry") # Value error if not in the listprint(my_list)

# clear() : removes all items from the list
my_list.clear()
print(my_list)

# reverse() : reverse the items
my_list = ["banana", "cherry", "apple"]
my_list.reverse()
print('Reversed: ', my_list)

# sort() : sort items in ascending order
my_list.sort()
```

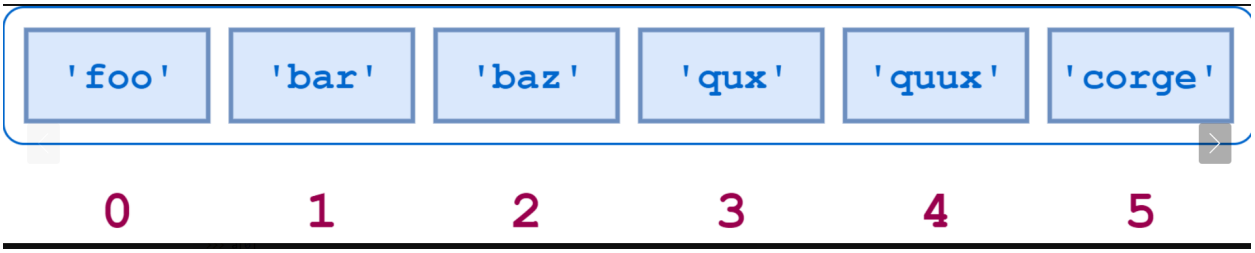
```
print('Sorted: ', my_list)

# use sorted() to get a new list, and leave the original unaffected.
# sorted() works on any iterable type, not just lists
my_list = ["banana", "cherry", "apple"]
new_list = sorted(my_list)

# create list with repeated elements
list_with_zeros = [0] * 5
print(list_with_zeros)

# concatenation
list_concat = list_with_zeros + my_list
print(list_concat)

# convert string to list
string_to_list = list('Hello')
print(string_to_list)
```

Lists Conclusion

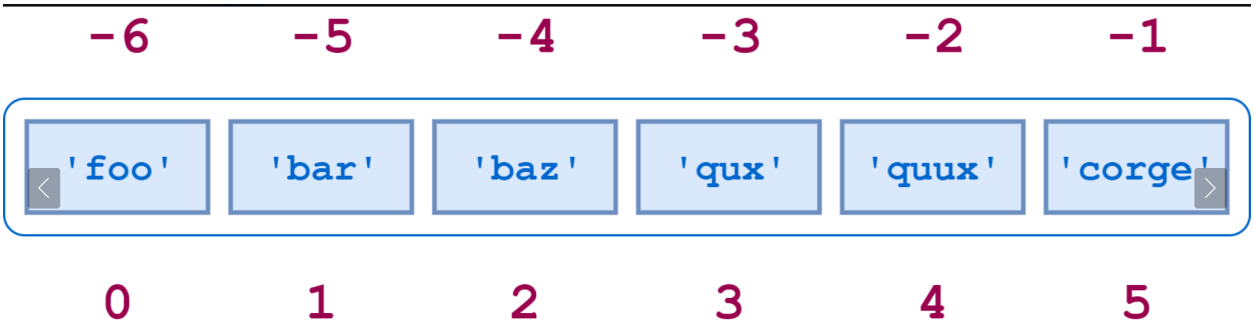
- **Lists are ordered.**
- **Lists can contain any arbitrary objects.**
- **List elements can be accessed by index.**
- **Lists can be nested to arbitrary depth.**
- **Lists are mutable.**
- **Lists are dynamic**

Indexing and Slicing

```
list_index= ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
```




```
# Retreving using +ve indexing
print(list_index[0])
print(list_index[2])
print(list_index[5])
```




```
# Retreving using -ve indexing
print(list_index[-1])
print(list_index[-3])
print(list_index[-5])
```

Slicing

Slicing is: If a is a list, the expression a[m:n] returns the portion of a from index m to, but not including, index n

```
a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']
# Try slice m = 2 and n = 5
a[2:5]
# Try both +ve and -ve slicing
print('-ve slicing',a[-5:-2])
print('+ve slicing',a[1:4])
print(a[-5:-2] == a[1:4])
```

Note
Omitting the first index starts the slice at the beginning of the list, and omitting the second index extends the slice to the end of the list:

```
print(a[:4], a[0:4])
print(a[2:], a[2:len(a)])
a[:4] + a[4:]
```

Note
The syntax for reversing a list works the same way it does for strings:

```
a[::-1]
```

Also, You can specify a stride—either positive or negative:

```
print(a[0:6:2])
print(a[1:6:2])
print(a[6:0:-2])
```