Python Programming

Python Data Types:

- Data types specify the type of data that can be stored inside a variable.
- You can get the data type of any object by using the type() function.

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either `True` or `False`
Set	set, frozeenset	hold collection of unique items

Python Data Types:

- Since everything is an object in Python programming, data types are actually classes and variables are instances(object) of these classes.

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset

Python Numeric Data Type:

- **Integers, floating-point numbers and complex numbers** fall under Python numbers category. They are defined as **int, float and complex** classes in Python.
  - **int** - holds signed integers of non-limited length.
  - **float** - holds floating decimal points and it's accurate up to 15 decimal places.
  - **complex** - holds complex numbers.
- We can use the **type() function** to know which class a variable or a value belongs to.

Python Numeric Data Type:

```
@author: Balu Laxman
"""
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 2.0
print(num2, 'is of type', type(num2))

num3 = 1+2j
print(num3, 'is of type', type(num3))
```

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is of type <class 'complex'>
```

Python Numbers, Type Conversion and Mathematics:

- The **number data types** are used **to store the numeric values.**
- Python supports integers, floating-point numbers and complex numbers. They are defined as int, float, and complex classes in Python.
  - **int** - holds signed integers of non-limited length.
  - **float** - holds floating decimal points and it's accurate up to 15 decimal places.
  - **complex** - holds complex number

Python Numeric Data Type:

- Integers and floating points are separated by the presence or absence of a decimal point. For instance,
  - 5 is an integer
  - 5.42 is a floating-point number.
- Complex numbers are written in the form, x +yj, where x is the real part and y is the imaginary part.
- We can use the type() function to know which class a variable or a value belongs to.

```
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 5.42
print(num2, 'is of type', type(num2))

num3 = 8+2j
print(num3, 'is of type', type(num3))
```

Number Systems:

- The numbers we deal with every day are of the decimal (base 10) number system.
- But computer programmers need to work with binary (base 2), hexadecimal (base 16) and octal (base 8) number systems.
- In Python, we can represent these numbers by appropriately placing a prefix before that number. The following table lists these prefixes.

Number System	Prefix
Binary	0b or 0B
Octal	0o or 00
Hexadecimal	0x or 0X

```
print(0b1101011)  # prints 107

print(0xFB + 0b10)  # prints 253

print(0o15)  # prints 13
```

Type Conversion in Python:

- In programming, type conversion is the process of converting one type of number into another.
- Operations like addition, subtraction convert integers to float implicitly (automatically), if one of the operands is float. For example,

```
print(1 + 2.0) # prints 3.0
```

- We can also use built-in functions like int(), float() and complex() to convert between types explicitly.

```
num1 = int(2.3)
print(num1)

num2 = int(-2.8)
print(num2)

num3 = float(5)
print(num3)

num4 = complex('3+5j')
print(num4)
```

Python Random Module:

- Python offers the random module to generate random numbers or to pick a random item from an iterator.
- First we need to import the random module.

```
import random

print(random.randrange(10, 20))

list1 = ['a', 'b', 'c', 'd', 'e']

# get random item from list1
print(random.choice(list1))

# Shuffle list1
random.shuffle(list1)

# Print the shuffled list1
print(list1)

# Print random element
print(random.random())
```

```
12
c
['b', 'a', 'e', 'c', 'd']
0.8915375820057005
```

Python List Data Type:

- List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ].

- To access items from a list, we use the index number (0, 1, 2 ...).

```
languages = ["C-Programming", "Java", "Python"]
# access element at index 0
print(languages[0])   # C-Programming
# access element at index 2
print(languages[2])   # Python
```

Python List Data Type:

- In Python, lists are used to store multiple data at once.
- Suppose we need to record the ages of 5 students. Instead of creating 5 separate variables, we can simply create a list.
- A **list is created** in Python **by placing items inside [], separated by commas.**
- A list can have any number of items and they may be of **different types (integer, float, string, etc.)**


```
numbers = [1, 2, 5]
print(numbers)
```

```
# empty list
my_list = []

# list with mixed data types
my_list = [1, "Hello", 3.4]
print(my_list)
```

Access Python List Elements:

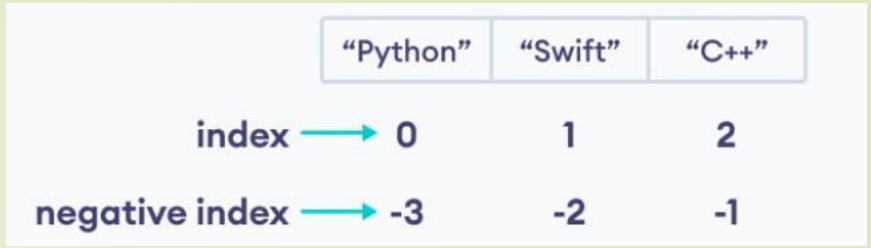
- In Python, each item in a list is associated with a number. The number is known as a **list index.**
- We can access elements of an array using the index number (0, 1, 2 …).

```
languages = ["C-Programming", "Java", "Python"]
# access element at index 0
print(languages[0])   # C-Programming
# access element at index 2
print(languages[2])   # Python
```

Access Python List Elements:

- Python allows **negative indexing** for its sequences. **The index of -1 refers to the last item, -2 to the second last item and so on.**

```
languages = ["C-Programming", "Java", "Python"]
# access element at index 0
print(languages[-1])   # Python
# access element at index 2
print(languages[-2])   # Java
```



- If the specified index does not exist in the list, Python throws the **IndexError exception.**

Slicing of a Python List:

- In Python it is possible **to access a section of items from the list** using the **slicing operator :**, not just a single item.

```
# List slicing in Python
my_list = ['p','r','o','g','r','a','m','i','z']
# items from index 2 to index 4
print(my_list[2:5])
# items from index 5 to end
print(my_list[5:])
# items beginning to end
print(my_list[:])
```

```
['o', 'g', 'r']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

- **When we slice lists, the start index is inclusive but the end index is exclusive.**

Add Elements to a Python List:

- Python List provides different methods to add items to a list.
  - **1. Using append():** The append() method **adds an item at the end of the list.**

```
numbers = [21, 34, 54, 12]

print("Before Append:", numbers)

# using append method
numbers.append(32)

print("After Append:", numbers)
```

```
Before Append: [21, 34, 54, 12]
After Append: [21, 34, 54, 12, 32]
```

Add Elements to a Python List:

- Python List provides different methods to add items to a list.
  - **2. Using extend():** We use the extend() method **to add all items of one list to another.**

```
prime_numbers = [2, 3, 5]
print("List1:", prime_numbers)

even_numbers = [4, 6, 8]
print("List2:", even_numbers)

# join two lists
prime_numbers.extend(even_numbers)

print("List after append:", prime_numbers)
```

```
List1: [2, 3, 5]
List2: [4, 6, 8]
List after append: [2, 3, 5, 4, 6, 8]
```

Change List Items:

- Python lists are **mutable.** Meaning **lists are changeable.** And, we can change items of a list by assigning new values using = operator.

```
languages = ['Python', 'C', 'C++']

# changing the third item to 'Java'
languages[2] = 'Java'

print(languages)  # ['Python', 'Swift', 'C']
```

```
['Python', 'C', 'Java']
```

Remove an Item From a List:

- **1. Using del():** In Python we can use the del statement to remove one or more items from a list.

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the second item
del languages[1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the last item
del languages[-1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']

# delete first two items
del languages[0 : 2]  # ['C', 'Java', 'Rust']
print(languages)
```

```
['Python', 'C++', 'C', 'Java', 'Rust', 'R']
['Python', 'C++', 'C', 'Java', 'Rust']
['C', 'Java', 'Rust']
```

Remove an Item From a List:

- **2. Using remove():** In Python we can also use the remove() method to delete a list item.

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']

# remove 'Python' from the list
languages.remove('Python')

print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust', 'R']
```

Python List Methods:

Method	Description
append()	add an item to the end of the list
extend()	add items of lists and other iterables to the end of the list
insert()	inserts an item at the specified index
remove()	removes item present at the given index
pop()	returns and removes item present at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the specified item in the list
sort()	sort the list in ascending/descending order
reverse()	reverses the item of the list
copy()	returns the shallow copy of the list

Iterating through a List:

- We can use the for loop to iterate over the elements of a list.

```
languages = ['Python', 'Swift', 'C++']
# iterating through the list
for language in languages:
    print(language)
```

```
Python
Swift
C++
```

- We use the **in** keyword to check if an item exists in the list or not.

```
languages = ['Python', 'Java', 'C++']
print('C' in languages)
print('Python' in languages)
```

```
False
True
```

Python List Length:

- In Python, we use the **len() function** to find the number of elements present in a list.

```
languages = ['Python', 'C-Programming', 'C++']

print("List: ", languages)

print("Total Elements: ", len(languages))
```

Python Tuple Data Type:

- Tuple is an ordered sequence of items **same as a list. The only difference is that tuples are immutable.** Tuples once created cannot be modified.
- In Python, we use the **parentheses ()** to store items of a tuple.
- Similar to lists, we use the index number to access tuple items in Python.

```
# create a tuple
product = ('Microsoft', 'Xbox', 499.99)

# access element at index 0
print(product[0])   # Microsoft

# access element at index 1
print(product[1])   # Xbox
```

Python Tuple Data Type:

- A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.
- A tuple is **created** by placing all the items (elements) inside parentheses (), separated by commas.
- The **parentheses are optional**, however, it is a good practice to use them.
- A tuple **can have any number of items** and they may be of different types (integer, float, list, string, etc.).

Python Tuple Data Type:

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

```
my_tuple = 1, 2, 3
my_tuple = 1, "Hello", 3.4
print(my_tuple)
```

Create a Python Tuple With one Element:

- In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.
- We will **need a trailing comma** to indicate that it is a tuple.

```
thistuple = ("apple",)
print(type(thistuple))

thistuple = ("apple")
print(type(thistuple))
```

```
var1 = ("hello")
print(type(var1))

var2 = ("hello",)
print(type(var2))

var3 = "hello",
print(type(var3))
```

Access Python Tuple Elements:

- Like a list, each element of a tuple is represented by index numbers (0, 1, ...) where the first element is at index 0.
- We use the index number to access tuple elements.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

- The index must be an integer, so we cannot use float or other types. This will result in TypeError.
- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Python Tuple Methods:

- In Python, **methods that add items or remove items are not available with tuple.** Only the following two methods are available.
- my_tuple.count('p') - counts total number of 'p' in my_tuple
- my_tuple.index('l') - returns the first occurrence of 'l' in my_tuple

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # prints 2
print(my_tuple.index('l'))  # prints 3
```

Iterating through a Tuple in Python:

- We can use the **for** loop **to iterate over the elements of a tuple.**

```
languages = ('Python', 'Swift', 'C++')

# iterating through the tuple
for language in languages:
    print(language)
```

- We use the **in** keyword **to check if an item exists in the tuple or not.**

```
languages = ('Python', 'Java', 'C++')

print('C' in languages)    # False
print('Python' in languages)    # True
```

Advantages of Tuple over List in Python:

- Since tuples are quite similar to lists, both of them are used in similar situations.
- However, there are certain advantages of implementing a tuple over a list:
  - We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
  - Since tuples are immutable, iterating through a tuple is faster than with a list. So there is a slight performance boost.
  - Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
  - If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Python String Data Type:

- String is a sequence of characters represented by either single or double quotes.

```
name = 'Python'
print(name)

message = "Python for beginners"
print(message)
```

Python Set Data Type:

- Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces {}.

```
# create a set named student_id
student_id = {112, 114, 116, 118, 115}

# display student_id elements
print(student_id)

# display type of student_id
print(type(student_id))
```

```
{112, 114, 115, 116, 118}
<class 'set'>
```

Python Dictionary Data Type:

- Python dictionary is an ordered collection of items. It stores elements in key/value pairs.
- Here, keys are unique identifiers that are associated with each value.

```
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city)
```

Python Dictionary Data Type:

- We use keys to retrieve the respective value. But not the other way around. For example,

```
capital_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital_city['Nepal'])  # prints Kathmandu

print(capital_city['Kathmandu'])
```

Quick Review:

- The following code example would print the data type of x, what data type would that be**?**

```
x = 5
print(type(x))
```

```
x = "Hello World"
print(type(x))
```

```
x = 20.5
print(type(x))
```

```
x = True
print(type(x))
```

```
x = ["apple", "banana", "cherry"]
print(type(x))
```

```
x = ("apple", "banana", "cherry")
print(type(x))
```

```
x = {"name" : "John", "age" : 36}
print(type(x))
```

Python Operators:

- Operators are special symbols that perform operations on variables and values.
  - Arithmetic operators
  - Assignment Operators
  - Comparison Operators
  - Logical Operators
  - Bitwise Operators
  - Special Operators

Python Arithmetic Operators:

Operator	Operation	Example
+	Addition	5 + 2 = 7
-	Subtraction	4 - 2 = 2
*	Multiplication	2 * 3 = 6
/	Division	4 / 2 = 2
//	Floor Division	10 // 3 = 3
%	Modulo	5 % 2 = 1
**	Power	4 ** 2 = 16

Python Arithmetic Operators:

```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)

# floor division
print ('Floor Division: ', a // b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

```
Sum:  9
Subtraction:  5
Multiplication:  14
Division:  3.5
Floor Division:  3
Modulo:  1
Power:  49
```

Python Assignment Operators:

Operator	Name	Example
`=`	Assignment Operator	`a = 7`
`+=`	Addition Assignment	`a += 1 # a = a + 1`
`-=`	Subtraction Assignment	`a -= 3 # a = a - 3`
`*=`	Multiplication Assignment	`a *= 4 # a = a * 4`
`/=`	Division Assignment	`a /= 3 # a = a / 3`
`%=`	Remainder Assignment	`a %= 10 # a = a % 10`
`**=`	Exponent Assignment	`a **= 10 # a = a ** 10`

Python Comparison Operators:

Operator	Meaning	Example
`==`	Is Equal To	`3 == 5` gives us **False**
`!=`	Not Equal To	`3 != 5` gives us **True**
`>`	Greater Than	`3 > 5` gives us **False**
`<`	Less Than	`3 < 5` gives us **True**
`>=`	Greater Than or Equal To	`3 >= 5` give us **False**
`<=`	Less Than or Equal To	`3 <= 5` gives us **True**

Python Comparison Operators:

```
a = 5

b = 2

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

```
a == b = False
a != b = True
a > b = True
a < b = False
a >= b = True
a <= b = False
```

Python Logical Operators:

- Logical operators are used to check whether an expression is True or False. They are used in decision-making.

Operator	Example	Meaning
`and`	a **and** b	**Logical AND:** `True` only if both the operands are `True`
`or`	a **or** b	**Logical OR:** `True` if at least one of the operands is `True`
`not`	**not** a	**Logical NOT:** `True` if the operand is `False` and vice-versa.

Python Logical Operators:

- Logical operators are used to check whether an expression is True or False. They are used in decision-making.

```
# logical AND
print(True and True)
print(True and False)

# logical OR
print(True or False)

# logical NOT
print(not True)
```

```
True
False
True
False
```

Python Bitwise Operators:

In the table below: Let `x` = 10 (`0000 1010` in binary) and `y` = 4 (`0000 0100` in binary)

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (`0000 0000`)
|	Bitwise OR	x | y = 14 (`0000 1110`)
~	Bitwise NOT	~x = -11 (`1111 0101`)
^	Bitwise XOR	x ^ y = 14 (`0000 1110`)
>>	Bitwise right shift	x >> 2 = 2 (`0000 0010`)
<<	Bitwise left shift	x << 2 = 40 (`0010 1000`)

Python Special operators:

- Python language offers some special types of operators like the **identity** operator and the **membership** operator.

Identity operators:

- In Python, **is** and **is not** are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

print(x1 is not y1)  # prints False

print(x2 is y2)  # prints True

print(x3 is y3)  # prints False
```

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory although they are equal.

Membership operators:

- In Python, **in** and **not in** are the membership operators. **They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).**
- In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
`in`	`True` if value/variable is **found** in the sequence	`5 in x`
`not in`	`True` if value/variable is **not found** in the sequence	`5 not in x`

Membership operators:

```
x = 'Hello world'
y = {1:'a', 2:'b'}

# check if 'H' is present in x string
print('H' in x)

# check if 'hello' is present in x string
print('hello' not in x)

# check if '1' key is present in y
print(1 in y)

# check if 'a' key is present in y
print('a' in y)
```

```
True
True
True
False
```

Python Basic Input and Output:

- In Python, we can simply use the **print()** function to print output.

```
print('Python is powerful')
```

- In the above code, the print() function is taking a single parameter. However, the actual syntax of the print function accepts 5 parameters.

```
print(object= separator= end= file= flush=)
```

- **object** - value(s) to be printed
- **sep (optional)** - allows us to separate multiple objects inside print().
- **end (optional)** - allows us to add add specific values like new line "\n", tab "\t"
- **file (optional)** - where the values are printed. It's default value is sys.stdout (screen)
- **flush (optional)** - boolean specifying if the output is flushed or buffered. Default: False

Python Basic Input and Output:

```
print('Good Morning!')
print('Welcome to Python Class')
```

```
print('Good Morning!' , end= ' ')
print('Welcome to Python Class.')
```

```
Good Morning! Welcome to Python Class.
```

```
print('Good Morning', "Welcome to DoCSE", 'SVNIT, Surat', 395007)
```

```
print('Good Morning', "Welcome to DoCSE", 'SVNIT, Surat', 395007, sep= '. ')
```

```
Good Morning. Welcome to DoCSE. SVNIT, Surat!. 395007
```

Python Basic Input and Output:

```
number = -10.6

name = "Programiz"

# print literals
print(5)

# print variables
print(number)
print(name)
```

```
print('Programiz is ' + 'awesome.')
```

Output formatting:

```
x = 5
y = 10

print('The value of x is {} and y is {}'.format(x,y))
```

Here, the curly braces {} are used as placeholders. We can specify the order in which they are printed by using numbers (tuple index).

Python Input:

- While programming, we might want to take the input from the user. In Python, we can use the **input()** function.

```
input(prompt)
```

- Here, prompt is the string we wish to display on the screen. It is optional.

```
num = input('Enter a number: ')

print('You Entered:', num)

print('Data type of num:', type(num))
```

- To convert user input into a number we can use **int()** or **float()** functions as:

```
num = int(input('Enter a number: '))
```

Thank you