Python Basics - 1

OVERVIEW & PURPOSE

In this session, participants will be introduced to Python's history, features, and its significance in data science. We will explore core programming elements, such as variables, data types, expressions, and operators, complemented by hands-on activities. The lesson will culminate in an understanding of Python's primary data structures.

OBJECTIVES

1. Introduction to Python: history, features, and advantages
2. Variables and data types: integers, floats, strings, Booleans, and None
3. Expressions and operators: arithmetic, assignment, comparison, and logical
4. Understanding type() function and type inference
5. Brief Introduction to data structures: lists, tuples, and dictionaries

Introduction to Python: history, features, and advantages

History: Python was created in the late 1980s by Guido van Rossum in the Netherlands. It was released as Python 0.9.0 in February 1991. The name "Python" was inspired by the British comedy group "Monty Python," which Guido enjoyed.

Features: Python is known for its simplicity and readability. It uses English-like commands that make the language easy to understand and write. It's a versatile language, capable of web development, data analysis, artificial intelligence, and much more.

Advantages:

1. Easy to learn and use: Python's clear syntax makes it great for beginners.
2. Versatile: It can be used for various applications, from web development to data science.
3. Large Community: If you have a question, chances are someone has already answered it. There are numerous resources, libraries, and tools available.

Basic Concepts of Python Programming

1. Variables and data types

Variables are like containers where you store information. The kind of information they hold is called a data type.

Data Types	Class	Description
Numeric	int, float	Holds numeric values
String	str	Holds sequence of characters
Sequence	list, tuple	Holds collection of items
Mapping	dict	Holds data in key-value form
Boolean	bool	Holds either True or False
Set	set	Hold collection of unique items

1.1 Variables in Python:

In other programming languages like C, C++, and Java, you will need to declare the type of variables but in Python you don't need to do that. Just type in the variable and when values will be given to it, then it will automatically know whether the value given would be an int, float, or char or even a String.

Python Code:

```
#initialising variable directly
a = 5
#printing value of a
print ("The value of a is: " + str(a))
```

Output:

```
The value of a is: 5
```

Python Code:

```
# Assigning multiple values in single line
a,b,c="This is ","a ","Programming Session"
print(a+b+c)
```

Output:

```
This is a Programming Session
```

1.2 Python Data Types:

Data types represent the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are classes and variables are instances (object) of these classes. The following are the standard or built-in data types in Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary
- Binary Types

Python Code:

```
#Python program to demonstrate numeric value

a = 5

print("Type of a: ", type(a))

b = 5.0

print("\nType of b: ", type(b))

c = 2 + 4j

print("\nType of c: ", type(c))
```

Output:

```
Type of a:  <class 'int'>

Type of b:  <class 'float'>

Type of c:  <class 'complex'>
```

Note: type() function is used to determine the data type of variable.

2. Expressions and operators

Expressions are combinations of values (like numbers or text) and operators that can be evaluated to produce another value.

2.1 Arithmetic operators:

Addition (+): If you have 2 apples and someone gives you 3 more, you'd use addition to find out you now have 5 apples. In Python, you'd write it as 2 + 3.

Subtraction (-): Suppose from your 5 apples, you gave away 2. You'd subtract 2 from 5 and find out you have 3 apples left: 5 - 2.

Multiplication (*): If you have 5 boxes, and each box contains 3 oranges, the total number of oranges is 5 * 3 = 15.

Division (/): When you divide 15 oranges between 3 friends, each one will get 15 / 3 = 5 oranges.

Modulus (%): If 10 candies are to be shared between 3 kids, each kid would get 3 candies, and there would be 1 candy remaining. The modulus operator tells you what's left: 10 % 3 results in 1.

Exponentiation ():** If you want to know the result of 2 raised to the power of 3 (2 multiplied by itself twice), you'd use 2 ** 3, which equals 8.

2.2 Assignment operators:

Give values to variables (=, +=, -=, etc.)

Assignment operators in Python are used to perform operations on variables or operands and assign values to the operand on the left side of the operator.

The Simple assignment operator in Python is denoted by = and is used to assign values from the right side of the operator to the value on the left side.

```
a = b + c
```

- Add and equal operator:

```
a += 10 is same as a = a + 10.
a = 5
a += 10
print (a)
```

Output: 15

- Subtract and equal operator: a -= 5 is same as a = a – 5.

```
a = 10
a -= 5
print (a)
```

Output: 5

- Exponent assign operator: a **= 5 is same as a = a ** 5.

```
a = 2
a **= 5 print (a)
```

Output: 32

2.3 Comparison Operators

Comparison operators let you compare things:

Equal to (==): Determines if two things are the same. For example, is 5 the same as 5? (5 == 5 would be True)

Not equal to (!=): Checks if two things are different. Asking if 5 is not equal to 4? (5 != 4 would be True)

Greater than (>): Is 6 more than 5? (6 > 5 is True)

Less than (<): Is 4 less than 5? (4 < 5 is True)

Greater than or equal to (>=), Less than or equal to (<=): Just like the above, but inclusive. So, 5 <= 5 is True.

```
 # Comparison
Operators x = 10
 y = 5

 # Equal to
 print("x == y:", x == y)

 # Not equal to
 print("x != y:", x != y)

 # Greater than
print("x > y:", x > y)

 # Less than
 print("x < y:", x < y)

 # Greater than or equal
to print("x >= y:", x >=
y)

 # Less than or equal to
print("x <= y:", x <= y)
```

Output:

```
x == y: False
x != y: True
x > y: True
x < y: False
x >= y: True
x <= y: False
```

2.4 Logical Operators

Logical operators are used to perform logical operations. The following are the logical operators in Python:

- and (logical AND)
- or (logical OR)
- not (logical NOT)

Combining Conditional Statements using Logical Operators:

AND:

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

Output:

```
Both conditions are True
```

OR:

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```

Output:

```
At least one of the conditions is True
```

3. Understanding the type() function and type inference

In Python, you don't have to tell the computer what type of data a variable will hold (like in some other languages). Python figures this out on its own, called type inference.

The type() function helps you know what data type a particular value or variable is.

For example:

```
 x = 42
 y = 3.14

 print("x has type",
type(x)) print("y has type",
type(y))
```

Output:

```
x has type <class 'int'>
y has type <class 'float'>
```

4. Introduction to data structures: lists, tuples, and dictionaries

Think of data structures as more advanced containers (than variables) to store and organize data.

Lists: An ordered collection of items, which can be changed (or mutable). Written within square brackets. Example: [1, 2, 3, "apple"]

Tuples: Like lists, but once you make them, you can't change them (or immutable). Written within parentheses. Example: (1, 2, 3, "apple")

Dictionaries: A collection of key-value pairs. Like a real dictionary where you look up a word (the key) to find its definition (the value). Written within curly braces. Example: {"name": "John", "age": 30}

4.1 List Data Type:

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

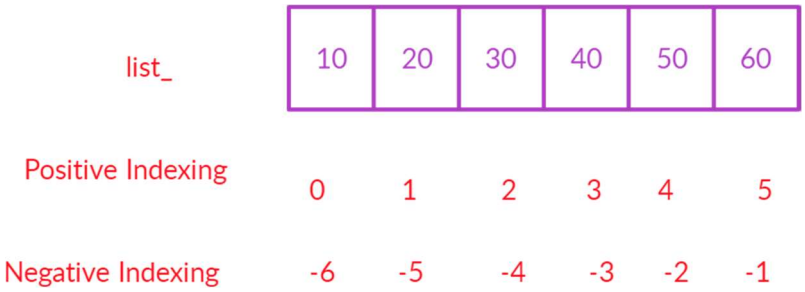
```
 numbers = [1, 2 ,3, 4 ,5]
 languages = ["Urdu", "English",
"French"]
 another_list = [2, "Urdu", "English"]

 print(numbers)
print(languages)
print(another_list)
print(type(numbers))
```

Output:

```
[1, 2, 3, 4, 5]
['Urdu', 'English', 'French']
[2, 'Urdu', 'English']
<class 'list'>
[6, 2, 3, 4, 5]
```

Access List Items:

To access items from a list, we use the index number (0, 1, 2 ...). The first element has the index 0. An element in the list can be accessed by putting its index in square brackets after the list name. For example, in the list numbers, numbers [0] accesses the first element "1".




```
 languages = ["Swift", "Java", "Python", "C++"]

 # access element at index 0
print(languages[0])   # Swift

 # access element at index 2
print(languages[2])   # Python

 # access element at the
end print(languages[-1])
     # C++
print(another_list)
print(type(numbers))
```

Output:

```
Swift
Python
C++
```

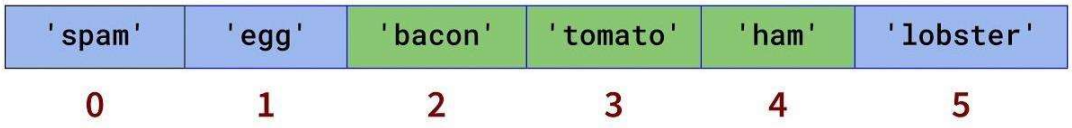
Modify List Items: You can modify the list items as shown below:

```
 numbers = [1, 2 ,3, 4
,5] print(numbers)

 #replaces the first index with 6
numbers[0] = 6

 #replaces the third index with 0
numbers[2] = 0

 print(numbers)
```

Output:

```
[1, 2, 3, 4, 5]
[6, 2, 0, 4, 5]
```

- **Slicing:** You can also access a range of values using the colon ':' in square brackets [].
  `a[2:5]`




```
 languages = ["Swift", "Java", "Python", "C++"]

 #accesses the elements at index 0 and 1.
print(languages[0:2])

 #accesses the elements at index 0, 1 and 2.
print(languages[0:3])

 #accesses the elements from index
1. print(languages[1:])

 #accesses the elements upto index
2. print(languages[:3])
```

Output:

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

4.2 Tuple Data Type:

Tuple is an ordered sequence of items the 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. We can access tuple items in the same way as list's.

```
 # 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
```

Output:

```
Microsoft
Xbox
```

4.3 String Data Type:

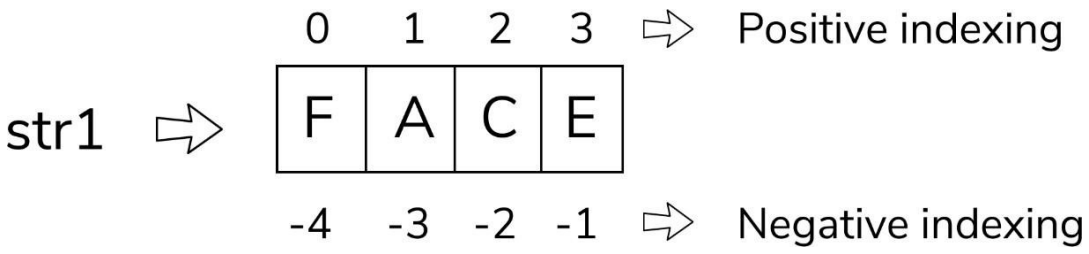
String is a sequence of characters represented by either single or double quotes. For example:

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

 message = 'Python for
```

Output:

```
Python
Python for Beginners
```

String Indexing: String indexing is also done in the same way as list's and tuples.




```
 message = 'Python for
beginners' print(message[0:7])
```

Output:

```
Python
```

Expressions, Conditional Statements & Python Advanced Data Types

Overview & Purpose

In this session, participants will delve into the world of expressions, uncovering the intricacies of operator precedence and associativity. We'll be introducing conditional statements, learning how to effectively use if, elif, and else to guide our code's decision-making. Advanced data structures: lists, tuples, and dictionaries, learning how to effectively use and iterate through each.

Objectives

- Evaluating expressions: operator precedence and associativity
- Introduction to conditional statements: if, elif, and else
- Executing code based on conditionals.
- Understand the fundamental data structures in Python: lists, tuples, and dictionaries.
- Perform basic operations on data structures, such as adding, modifying, and accessing elements.
- Manipulate Python strings effectively, including concatenation, slicing, and formatting.
- Working with Strings build-in functions

1. Evaluating Expressions: Operator Precedence and Associativity

When doing math, we know that multiplication and division come before addition and subtraction. Similarly, in Python, certain operations happen before others due to their "precedence." If operators have the same precedence, then we consider "associativity" (left to right or right to left).

The combination of values, variables, operators is termed as an expression in Python. For example:

x = 5 - 7 + 10

To evaluate these types of expressions, there is a rule of precedence in Python. It guides the order in which these operations are carried out. For example, multiplication has higher precedence than subtraction.

The operator precedence in Python is listed in the following table:

Data Types	Class
()	Parenthesis
**	Exponent
Positive *x*, negative *x*	Unary plus, Unary minus
*, /, //, %	Multiplication, Division, Floor division, Modulus
+, -	Addition, Subtraction
>, >=, <, <=	Comparison Operators
==, !=	Equality Operators
=, %= , /= , //= ,-= ,+= ,*= ,**=	Assignment Operators
not	Logical NOT
and	Logical AND

or	Logical OR

Examples:

```
 x = 2**3 + 3 * 4 / 10
print(x)
```

Output: 9.2

```
 y = 13 == 5 and (not10)> 12
print(y)
```

Output: False

```
 z = (5 % 2) + 30 <= 31 or True and False
print(z)
```

Output: True

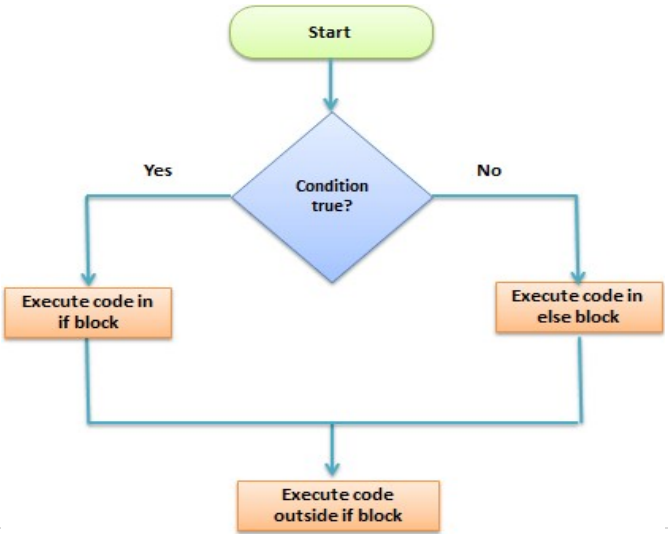
2. Introduction to Conditional Statements: if, elif, and else

Python **if** statement is used for decision-making operations. It contains a body of code which runs only when the condition given in the **if** statement is true. If the condition is false, then the optional **else** statement runs which contains some code for the else condition.

The **else** keyword catches anything which isn't caught by the preceding conditions.

Syntax:

if expression:
Statement
Else expression:
Statement



Example:

```
x = 10
y = 20

if x < y:
  print("x is less than y")
else:
    print("x is greater than or equal to y")
```

Output

```
x is less than y
```

Elif Statement:

The **"elif"** keyword is Python's way of saying "if the previous conditions were not true then try this condition". For example:

Example:

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

Output:

```
a and b are equal
```

Nested If:

You can have if statements inside if statements, this is called *nested* if statements.

```
x = 41
  if x > 10:
 print("Above ten,")
 if x > 20:
      print("and also above
 20!") else:
        print("but not above 20.")
```

Output:

```
Above ten,
and also above 20!
```

Sequence Data Type in Python

The sequence Data Type in Python is the ordered collection of similar or different data types. Sequences allow storing of multiple values in an organized and efficient fashion.

There are five collection data types in the Python programming language:

- **String** A string is a collection of one or more characters put in a single, double, or triple-quote. In python there is no character data type, a character is a string of length one.
- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

1. String Data Type

A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Strings Manipulation: Strings are like sentences. Manipulating them means playing with the words or characters in the sentence. For example, turning "hello" into "HELLO", replacing "cat" with "dog" in the sentence "The cat sat on the mat", or checking how long the word "elephant" is.

1.1 Creating String

Strings in Python can be created using single quotes or double quotes or even triple quotes. Python does not have a character data type, a single character is

simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example:

```
# Python Program to Access characters of String

String1 = "Python Programming"
print("Initial String: ")
print(String1)

# Printing First character
print("\nFirst character of String is: ")
print(String1[0])

# Printing Last character
print("\nLast character of String is: ")
print(String1[-1])
```

Output:

```
Initial String:
Python Programming

First character of String is:
P

Last character of String is:
g
```

To add a newline in a string, use the character combination \n:

Example:

```
print("Languages:\nPython\nC\nJavaScript")
```

Output:

```
Languages:

Python

C

JavaScript
```

1.2 String Slicing

In Python, the String Slicing method is used to access a range of characters in the String. Slicing in a String is done by using a Slicing operator, i.e., a colon (:).

NOTE: One thing to keep in mind while using this method is that the string returned after slicing includes the character at the start index but not the character at the last index.

Example:

```
# Program to demonstrate String slicing

#The [3:12] indicates that the string slicing will start from the 3rd index of the string to the 12th index,12th character not included.

        String1 = "Learning Python for Data Science"

        print("Initial String: " + String1)

        #Printing 3rd to 12th character

        print(String1[3:12])

        # Printing characters between 3rd and 2nd last character

        Print(String1[3:-2])
```

Output:

```
Initial String: Learning Python for Data Science
rning Pyt
rning Python for Data Scien
```

1.3 Python String constants Built-In Functions

1. String.digits - Returns all the digits '0123456789'.
2. String.lower() - Converts the entire string into lowercase letters.
3. String.endswith() - Returns True if a string ends with the given suffix otherwise returns False.
4. String.startswith() - Returns True if a string starts with the given prefix otherwise returns False.
5. String.isdigit() - Returns "True" if all characters in the string are digits, Otherwise, It returns "False".
6. String.isalpha() - Returns "True" if all characters in the string are alphabets, Otherwise, It returns "False".
7. String.index - Returns the position of the first occurrence of substring in a string.
8. String.swapcase() - Method converts all uppercase characters to lowercase and vice versa of the given string, and returns it.
9. String.replace() - Returns a copy of the string where all occurrences of a substring is replaced with another substring.
10. String.len() - Returns the length of the string.

2. List Data Type

Lists are just like arrays, which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

2.1 Creating List:

Example:

```
my_list = ["carrot", False, "cabbage", 19, 3.4, "cucumber", 22]
```

```
print(my_list)
```

Output:

```
['carrot', False, 'cabbage', 19, 3.4, 'cucumber', 22]
print(len(my_list)) #len gives the no. of items in a list.
7
```

2.2 Accessing Elements in a List:

Lists are ordered collections, so you can access any element in a list by telling Python the position, or index, of the item desired. To access an element in a list, write the name of the list followed by the index of the item enclosed in square brackets.

Example:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

Output:

```
trek
```

Python considers the first item in a list to be at position 0, not position 1. The second item in a list has an index of 1 and so on.

2.3 Adding Elements to a List

The simplest way to add a new element to a list is to append the item to the list. When you append an item to a list, the new element is added to the end of the list.

Example:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
```

```
motorcycles.append('ducati')

print(motorcycles)
```

The append() method at u adds 'ducati' to the end of the list without affecting any of the other elements in the list:

Output:

```
['honda', 'yamaha', 'suzuki']

['honda', 'yamaha', 'suzuki', 'ducati']
```

2.4 Python has a set of built-in methods that you can use on lists/arrays.

Method	***Description***
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

3. Tuples in Python

Tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists.

Example:

```
tup = (1, "a", "string", 1+2)
print(tup)
print(tup[1])
```

Output:

```
(1, 'a', 'string', 3)
a
```

3.1 Creating a Tuple

In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping the data sequence. Tuple items are ordered, unchangeable, and allow duplicate values.

Example:

```
mytuple = ("apple", "banana", "cherry")

print(mytuple)
```

Output:

```
('apple', 'banana', 'cherry')
```

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered
When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable
Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Note: To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.