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:

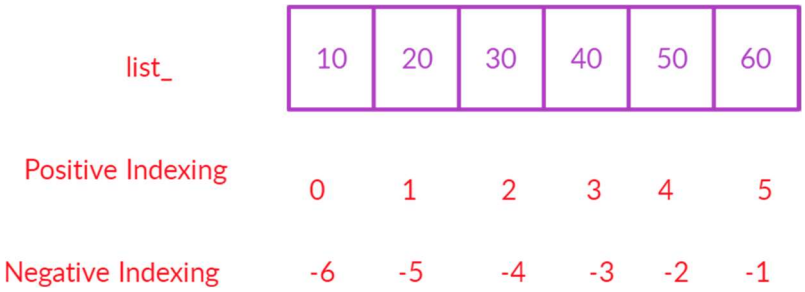
```
 numbers = [1, 2 ,3, 4 ,5]
 languages = ["Urdu", "English",
"French"]
 another_list = [2, "Urdu", "English"]

 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++
```

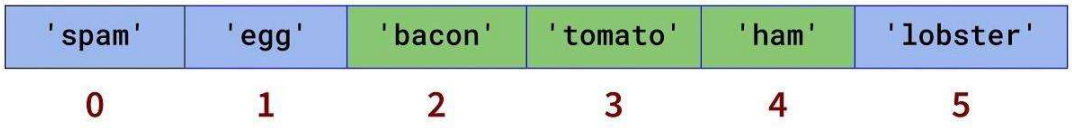
Modify List Items: You can modify the list items as shown below:

```
 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:

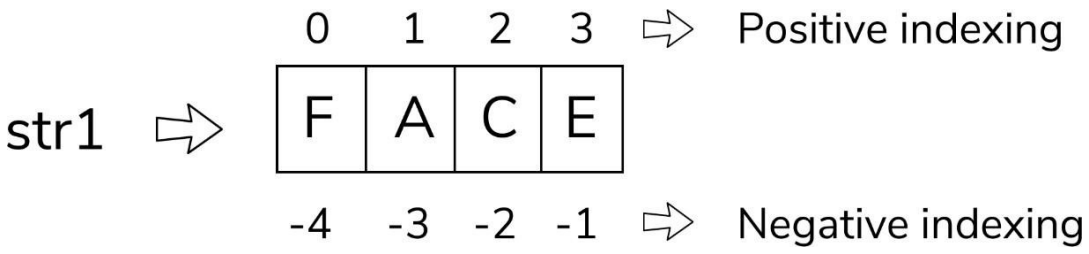
String is a sequence of characters represented by either single or double quotes. For example:

```
 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
```