WEEK ONE INTRODUCTION TO PYTHON PROGRAMMING

Key Concepts Term	Definition
Python	Versatile, high-level programming language emphasizing readability, created in 1991, ideal for data engineering due to extensive ecosystem
Data Engineering	Practice of building systems that collect, store, and analyze data at scale using automated pipelines
Script	File containing executable Python code that performs specific tasks
Variable	Named container that stores data values that can change during program execution
Variable Naming Rules	Guidelines: start with letter/underscore, avoid keywords, use descriptive names, case-sensitive
Term	Definition
Initialization	Process of assigning an initial value to a variable when first created
Reassignment	Changing the value stored in an existing variable after initial assignment

Syntax	Set of rules defining the correct structure and format of Python code
Print Statement	Built-in function print() used to display output to the console

Variable

a variable is a named storage location or container used to hold data values. These values can be of various types, such as numbers (integers, floats), text (strings), lists, dictionaries, or other objects.

Key characteristics of Python variables:

No explicit declaration:

Unlike some other programming languages, Python does not require you to declare a variable before using it. A variable is created the moment you assign a value to it using the assignment operator (=).

Dynamic typing:

Python is dynamically typed, meaning you don't need to specify the data type of a variable when you create it. The type is inferred based on the value assigned, and a variable can even hold values of different types throughout the program's execution.

Referencing objects:

In Python, variables are best understood as references or pointers to objects in memory, rather than being the objects themselves. When you assign a value to a variable, you are essentially making that variable point to the memory location where the object containing that value resides.

Mutability:

The value stored in a variable can be changed during the program's execution by assigning a new value to it. This new assignment will override the previous value

Example

```
name = "John"

age = 20

is_student = True

print(name)
```

Rules for Python variables:

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (age, Age and AGE are three different variables)

A variable name cannot be any of the Python keywords.

Data types

Python data types are actually classes, and the defined variables are their instances or objects. Since Python is dynamically typed, the data type of a variable is determined at runtime based on the assigned value.

In general, the data types are used to define the type of a variable. It represents the type of data we are going to store in a variable and determines what operations can be done on it.

Each programming language has its own classification of data items. With these datatypes, we can store different types of data values.

Think of data types like different types of containers:

A bottle holds liquid (like water or soda).

A box holds items (like clothes or books).

In programming, a variable is like a container, and the data type tells Python what kind of "thing" is inside.

1. Numeric Data Types

Python numeric data types store numeric values. Number objects are created when you assign a value to them. For example

var1 = 1 # int data type

var2 = True # bool data type

var3 = 10.023 # float data type

var4 = 10+3j # complex data type

Python supports four different numerical types and each of them have built-in classes in Python library, called int, bool, float and complex respectively

 int: Integer numbers (positive or negative)

python

x = 10

y = -5

float: Floating point numbers (with decimals)

python

pi = 3.14159

temperature = 98.6

complex: Complex numbers

python

z = 3 + 5j

A complex number is made up of two parts - real and imaginary. They are separated by '+' or '-' signs. The imaginary part is suffixed by 'j' which is the imaginary number. The square root of -1 (\sqrt{-1}), is defined as imaginary number. Complex number in Python is represented as x+yj, where x is the real part, and y is the imaginary part. So, 5+6j is a complex number.

String Data Types

Python string is a sequence of one or more Unicode characters, enclosed in single, double or triple quotation marks (also called inverted commas). Python strings are immutable which means when you perform an operation on strings, you always produce a new string object of the same type, rather than mutating an existing string.

As long as the same sequence of characters is enclosed, single or double or triple quotes don't matter. Hence, following string representations are equivalent.

A string is a non-numeric data type. Obviously, we cannot perform arithmetic operations on it. However, operations such as slicing and concatenation can be done. Python's str class defines a number of useful methods for string processing. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator in Python.

str = 'Hello World!'

print (str) # Prints complete string

print (str[0]) # Prints first character of the string

print (str[2:5]) # Prints characters starting from 3rd to 5th

print (str[2:]) # Prints string starting from 3rd character

print (str * 2) # Prints string two times

print (str + "TEST") # Prints concatenated string

Sequence Data Types

Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0. It is conceptually similar to an array in C or C++. There are following three sequence data types defined in Python.

List Data Type

Tuple Data Type

Range Data Type

Python sequences are bounded and iterable - Whenever we say an iterable in Python, it means a sequence data type (for example, a list).

Python List Data Type

Python List are the most versatile compound data types. A Python list contains items separated by commas and enclosed within square brackets ([]). To some extent, Python lists are similar to arrays in C. One difference between them is that all the items belonging to a Python list can be of different data type where as C array can store elements related to a particular data type.

A list in Python is an object of list class. We can check it with type() function.

As mentioned, an item in the list may be of any data type. It means that a list object can also be an item in another list. In that case, it becomes a nested list.

[['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]

A list can have items which are simple numbers, strings, tuple, dictionary, set or object of user defined class also.

The values stored in a Python list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tinylist = [123, 'john']

print (list) # Prints complete list

print (list[0]) # Prints first element of the list

print (list[1:3]) # Prints elements starting from 2nd till 3rd

print (list[2:]) # Prints elements starting from 3rd element

print (tinylist * 2) # Prints list two times

print (list + tinylist) # Prints concatenated lists
```

Python Tuple Data Type

Tuple is another sequence data type that is similar to a list. A Python tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses (...).

A tuple is also a sequence, hence each item in the tuple has an index referring to its position in the collection. The index starts from 0.

(2023, "Python", 3.11, 5+6j, 1.23E-4)

In Python, a tuple is an object of tuple class. We can check it with the type() function.

type((2023, "Python", 3.11, 5+6j, 1.23E-4))

As in case of a list, an item in the tuple may also be a list, a tuple itself or an object of any other Python class.

(['One', 'Two', 'Three'], 1,2.0,3, (1.0, 2.0, 3.0))

To form a tuple, use of parentheses is optional. Data items separated by comma without any enclosing symbols are treated as a tuple by default

Example of Tuple data Type

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

tinytuple = (123, 'john')

print (tuple) # Prints the complete tuple

print (tuple[0]) # Prints first element of the tuple

print (tuple[1:3]) # Prints elements of the tuple starting from 2nd till 3rd

print (tuple[2:]) # Prints elements of the tuple starting from 3rd element

print (tinytuple * 2) # Prints the contents of the tuple twice

print (tuple + tinytuple) # Prints concatenated tuples
```

The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed i.e. lists are mutable, while tuples are enclosed in parentheses (( )) and cannot be updated (immutable). Tuples can be thought of as read-only lists.

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )

list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

tuple[2] = 1000 # Invalid syntax with tuple

list[2] = 1000 # Valid syntax with list
```

Python Range Data Type

A Python range is an immutable sequence of numbers which is typically used to iterate through a specific number of items.

It is represented by the Range class. The constructor of this class accepts a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number. Following is the syntax of the function −

```
range(start, stop, step)
```

Here is the description of the parameters used −

start: Integer number to specify starting position, (Its optional, Default: 0)

stop: Integer number to specify ending position (It's mandatory)

step: Integer number to specify increment, (Its optional, Default: 1)

Example of Range Data Type

Following is a program which uses for loop to print number from 0 to 4 −

```
for i in range(5):
print(i)
```

Python Binary Data Types

A binary data type in Python is a way to represent data as a series of binary digits, which are 0's and 1's. It is like a special language computers understand to store and process information efficiently.

This type of data is commonly used when dealing with things like files, images, or anything that can be represented using just two possible values. So, instead of using regular numbers or letters, binary sequence data types use a combination of 0s and 1s to represent information.

Python provides three different ways to represent binary data. They are as follows −

bytes

bytearray

memoryview

Let us discuss each of these data types individually −

(a) Python Bytes Data Type

The byte data type in Python represents a sequence of bytes. Each byte is an integer value between 0 and 255. It is commonly used to store binary data, such as images, files, or network packets.

We can create bytes in Python using the built-in bytes() function or by prefixing a sequence of numbers with b.

Other Data Types

Python Dictionary Data Type

Python dictionaries are kind of hash table type. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Python dictionary is like associative arrays or hashes found in Perl and consist of key:value pairs. The pairs are separated by comma and put inside curly brackets {}. To establish mapping between key and value, the semicolon':' symbol is put between the two.

{1:'one', 2:'two', 3:'three'}

In Python, dictionary is an object of the built-in dict class. We can check it with the type() function.

type({1:'one', 2:'two', 3:'three'})

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print (dict['one']) # Prints value for 'one' key
```

print (dict[2]) # Prints value for 2 key

print (tinydict) # Prints complete dictionary

print (tinydict.keys()) # Prints all the keys

print (tinydict.values()) # Prints all the values

Python Set Data Type

is a Python implementation of set as defined in Mathematics. A set in Python is a collection, but is not an indexed or ordered collection as string, list or tuple. An object cannot appear more than once in a set, whereas in List and Tuple, same object can appear more than once.

Comma separated items in a set are put inside curly brackets or braces {}. Items in the set collection can be of different data types.

{2023, "Python", 3.11, 5+6j, 1.23E-4}

{(5+6j), 3.11, 0.000123, 'Python', 2023}

Note that items in the set collection may not follow the same order in which they are entered. The position of items is optimized by Python to perform operations over set as defined in mathematics.

Python's Set is an object of built-in set class, as can be checked with the type() function.

type({2023, "Python", 3.11, 5+6j, 1.23E-4})

A set can store only immutable objects such as number (int, float, complex or bool), string or tuple. If you try to put a list or a dictionary in the set collection, Python raises a TypeError.

{['One', 'Two', 'Three'], 1,2,3, (1.0, 2.0, 3.0)}

Hashing is a mechanism in computer science which enables quicker searching of objects in computer's memory. Only immutable objects are hashable.

Even if a set doesn't allow mutable items, the set itself is mutable. Hence, add/delete/update operations are permitted on a set object, using the methods in built-in set class. Python also has a set of operators to perform set manipulation. The methods and operators are explained in latter chapters

```
set1 = {123, 452, 5, 6}

set2 = {'Java', 'Python', 'JavaScript'}

print(set1)

print(set2)
```

Python Boolean Data Type

Python boolean type is one of built-in data types which represents one of the two values either True or False. Python bool() function allows you to evaluate the value of any expression and returns either True or False based on the expression.

A Boolean number has only two possible values, as represented by the keywords, True and False. They correspond to integer 1 and 0 respectively.

Example of Boolean Data Type

```
a = True

# display the value of a

print(a)

# display the data type of a

print(type(a))
```

Getting Data Type

To get the data types in Python, you can use the type() function. The type() is a built-in function that returns the class of the given object.

```
# Getting type of values

print(type(123))
```

```
print(type(9.99))

# Getting type of variables

a = 10

b = 2.12

c = "Hello"

d = (10, 20, 30)

e = [10, 20, 30]

print(type(a))

print(type(b))

print(type(c))

print(type(d))

print(type(e))
```

Setting Data Type

In Python, during declaring a variable or an object, you don't need to set the data types. Data type is set automatically based on the assigned value.

Example

The following example, demonstrating how a variable's data type is set based on the given value −

```
# Declaring a variable

# And, assigning an integer value

x = 10

# Printing its value and type

print("x = ", x)

print("type of x = ", type(x))
```

```
# Now, assigning string value to

# the same variable

x = "Hello World!"

# Printing its value and type

print("x = ", x)

print("type of x = ", type(x))
```

Primitive and Non-primitive Data Types

The above-explained data types can also be categorized as primitive and non-primitive.

1. Primitive Types

The primitive data types are the fundamental data types that are used to create complex data types (sometimes called complex data structures). There are mainly four primitive data types, which are −

Integers

Floats

Booleans, and

Strings

2. Non-primitive Types

The non-primitive data types store values or collections of values. There are mainly four types of non-primitive types, which are −

Lists

Tuples

Dictionaries, and

Sets

Python Data Type Conversion

Sometimes, you may need to perform conversions between the built-in data types. To convert data between different Python data types, you simply use the type name as a function.

Example

Following is an example which converts different values to integer, floating point and string values respectively −

```
print("Conversion to integer data type")

a = int(1) # a will be 1

b = int(2.2) # b will be 2

c = int("3.3") # c will be 3

print (a)

print (b)

print (c)

print("Conversion to floating point number")

a = float(1) # a will be 1.0

b = float(2.2) # b will be 2.2

c = float("3.3") # c will be 3.3

print (a)

print (b)

print (c)

print("Conversion to string")
```

```
a = str(1) # a will be "1"

b = str(2.2) # b will be "2.2"

c = str("3.3") # c will be "3.3"

print (a)

print (b)

print (c)
```