1. Discuss about Python – Syntax and programming modes.

The Python syntax defines a set of rules that are used to create a Python Program. The Python Programming Language Syntax has many similarities to Perl, C, and Java Programming Languages. However, there are some definite differences between the languages.

There are two different modes of Python Programming. (a) Interactive Mode Programming (b) Script Mode Programming.

Python - Interactive Mode Programming

We can invoke a Python interpreter from command line by typing python at the command prompt as following –

$ python3

Python 3.10.6 (main, Mar 10 2023, 10:55:28) [GCC 11.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>>

Here **>>>** denotes a Python Command Prompt where you can type your commands. Let's type the following text at the Python prompt and press the Enter –

>>> print ("Hello, World!")

Python - Script Mode Programming

We can invoke the Python interpreter with a script parameter which begins the execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

write a simple Python program in a script which is simple text file. Python files have **extension .py.** Type the following source code in a test.py file

we should have Python interpreter path set in PATH variable. Now, let's try to run this program as follows –

In visual studio code create new file and type line of code as print(“Hello Worl”)

And save it as test.py.

$ python3 test.py

This produces the following result –

Hello, World!

2.Explain about Python Identifiers and there naming conventions.

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9). ***Python is a case sensitive programming language.***

Python does not allow punctuation characters such as @, $, and % within identifiers.
Naming conventions for Python identifiers –

a) Python Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
b) Starting an identifier with a single leading underscore indicates that the identifier is private identifier.
c) Starting an identifier with two leading underscores indicates a strongly private identifier.
d) If the identifier also ends with two trailing underscores, the identifier is a language-defined special name

3.Explain Python reserved or keywords?

These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	as	assert
break	class	continue
def	del	elif
else	except	False
finally	for	from
global	if	import
in	is	lambda
None	nonlocal	not
or	pass	raise
return	True	try
while	with	yield

4.Explain about Python Comments

Python comments are programmer-readable explanation or annotations in the Python source code. They are added with the purpose of making the source code easier for humans to understand, and are ignored by Python interpreter. Comments enhance the readability of the code and help the programmers to understand the code very carefully.

Python supports three types of comments as shown below –

- Single-line comments
- Multi-line comments
- Docstring Comments

Single Line Comments in Python

Single-line comments in Python start with a hash symbol (#) and extend to the end of the line. They are used to provide short explanations or notes about the code. They can be placed on their own line above the code they describe, or at the end of a line of code (known as an inline comment) to provide context or clarification about that specific line.

Example: Standalone Single-Line Comment

A standalone single-line comment is a comment that occupies an entire line by itself, starting with a hash symbol (#).

```
# Standalone single line comment is placed here
def greet():
  print("Hello, World!")
greet()
```

Multi Line Comments in Python

In Python, multi-line comments are used to provide longer explanations or notes that span multiple lines. While Python does not have a specific syntax for multi-line comments, there are two common ways to achieve this: consecutive single-line comments and triple-quoted strings –

Consecutive Single-Line Comments

Consecutive single-line comments refers to using the hash symbol (#) at the beginning of each line. This method is often used for longer explanations or to section off parts of the code.

```
# This function calculates the factorial of a number
# using an iterative approach. The factorial of a number
 # n is the product of all positive integers less than or
```

Multi Line Comment Using Triple Quoted Strings

We can use triple-quoted strings (''' or """) to create multi-line comments. These strings are technically string literals but can be used as comments if they are not assigned to any variable or used in expressions.

"""

This function calculates the greatest common divisor (GCD)

of two numbers using the Euclidean algorithm. The GCD of

two numbers is the largest number that divides both of them

without leaving a remainder.

""

Python Docstrings

In Python, docstrings are a special type of comment that is used to document modules, classes, functions, and methods. They are written using triple quotes (''' or """) and are placed immediately after the definition of the entity they document.

Docstrings can be accessed programmatically, making them an integral part of Pythons built-in documentation tools"

5. Explain about Python 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.

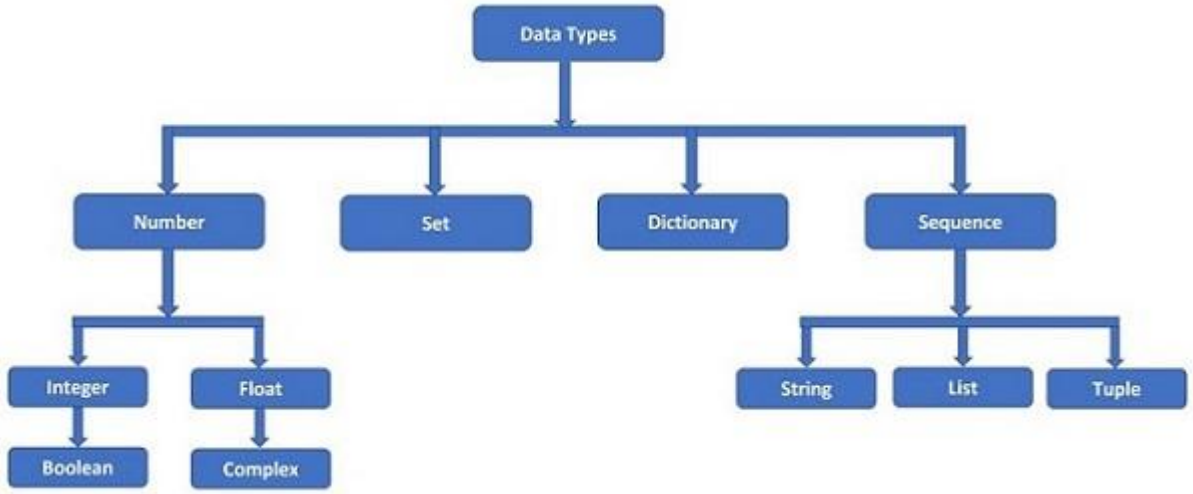
Each programming language has its own classification of data items.With these datatypes, we can store different types of data values.

Types of Data Types in Python

Python supports the following built-in data types –

- Numeric Data Types
  - int
  - flot
  - complex
- String Data Types
- Sequence Data Types
  - list
  - tuple

    - range
- Binary Data Types
    - bytes
    - bytearray
    - memoryview
- Dictionary Data Type
- Set Data Type
    - set
    - frozenset
- Boolean Data Type
- None Type



Python 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 (signed integers)
- float (floating point real values)
- complex (complex numbers)

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.

Ex:- # integer variable.

```
a=100
```

```
print("The type of variable having value", a, " is ", type(a))

# float variable.

c=20.345

print("The type of variable having value", c, " is ", type(c))

# complex variable.

d=10+3j

print("The type of variable having value", d, " is ", type(d))
```

Ex:-

```
>>> type("Welcome To Tutorials Point")

<class 'str'>
```

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.

6. Explain about Sequence Data Types

A sequence is an ordered collection of items, which can be of similar or different data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

a)String Data Type

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

```
s = 'Welcome to the Geeks World'
print(s)

# check data type
print(type(s))

# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

b)List Data Type

Lists are similar to arrays found in other languages. They are an ordered and mutable collection of items. It is very flexible as items in a list do not need to be of the same type.

Creating a List in Python

Lists in Python can be created by just placing sequence inside the square brackets[].

```
# Empty list
a = []
# list with int values
a = [1, 2, 3]
print(a)
# list with mixed values int and String
b = ["Geeks", "For", "Geeks", 4, 5]
print(b)
```

c)Tuple Data Type

Tuple is an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

Creating a Tuple in Python

In Python, tuples are created by placing a sequence of values separated by a 'comma' with or without the use of parentheses for grouping data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).

Note: *Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing* ***'comma'*** *to make it a tuple.*

```
# initiate empty tuple
tup1 = ()

tup2 = ('Geeks', 'For')
print("\nTuple with the use of String: ", tup2)
```

4. Set Data Type

In Python Data Types, Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Create a Set in Python

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'. The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

```
# initializing empty set
s1 = set()
s1 = set("GeeksForGeeks")
print("Set with the use of String: ", s1)

s2 = set(["Geeks", "For", "Geeks"])
```

```
print("Set with the use of List: ", s2)

# loop through set
for i in set1:
  print(i, end=" ") #prints elements one by one

# check if item exist in set
print("Geeks" in set1)
```

5. Dictionary Data Type

A dictionary in Python is a collection of data values, used to store data values like a map, unlike other Python Data Types, a Dictionary holds a key: value pair. Key-value is provided in dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a 'comma'.

Create a Dictionary in Python

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. The dictionary can also be created by the built-in function **dict().**

***Note** - Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.*

```
# initialize empty dictionary
d = {}

d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(d)

# creating dictionary using dict() constructor
d1 = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print(d1)
```

In order to access items of a dictionary refer to its key name. Key can be used inside square brackets. Using get() method we can access dictionary elements.

```
d = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Accessing an element using key
print(d['name'])

# Accessing a element using get
print(d.get(3))
```

Python Boolean Data type is one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true) and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by class bool.

```
print(type(True))
print(type(False))
```

7. Explain about type casting?

Python Type Casting is a process in which we convert a literal of one data type to another data type. Python supports two types of casting – implicit and explicit.

Python Implicit Casting

When any language compiler/interpreter automatically converts object of one type into other, it is called automatic or implicit casting. Python is a strongly typed language. It doesn't allow automatic type conversion between unrelated data types.

Implicit **int** to **float** casting takes place when any arithmetic operation on **int** and **float** operands is done.

Consider we have an ,**int** and one **float** variable

```
<<< a=10   # int object
<<< b=10.5 # float object
```

To perform their addition,

Python Explicit Casting

Although automatic or implicit casting is limited to **int** to **float** conversion, you can use Python's built-in functions int(), float() and str() to perform the explicit conversions such as string to integer.

Python int() Function

Python's built-in **int()** function converts an integer literal to an integer object, a float to integer, and a string to integer if the string itself has a valid integer literal representation.

Using **int()** with an int object as argument is equivalent to declaring an **int** object directly.

```
<<< a = int(10)
<<< a
```

inary String to Integer

The string should be made up of 1 and 0 only, and the base should be 2.

```
<<< a = int("110011", 2)
<<< a
51
```

The Decimal equivalent of binary number 110011 is 51.

Octal String to Integer

The string should only contain 0 to 7 digits, and the base

The Decimal equivalent of octal 20 is 16.

Hexa-Decimal String to Integer

The string should contain only the Hexadecimal symbols i.e., 0-9 and A, B, C, D, E or F. Base should be 16.

```
<<< a = int("2A9", 16)
<<< a
```

Decimal equivalent of Hexadecimal 2A9 is 681. You can easily verify these conversions with calculator app in Windows, Ubuntu or Smartphones.

Following is an example to convert number, float and string into integer data type:

```
a = int(1)     # a will be 1
b = int(2.2)   # b will be 2
c = int("3")   # c will be 3

print (a)
print (b)
print (c)
```

This produce the following result –

```
1
```

Binary String to Integer

The string should be made up of 1 and 0 only, and the base should be 2.

```
<<< a = int("110011", 2)
<<< a
51
```

The Decimal equivalent of binary number 110011 is 51.

Octal String to Integer

The string should only contain 0 to 7 digits, and the base should be 8.

```
<<< a = int("20", 8)
<<< a
16
```

The Decimal equivalent of octal 20 is 16.

Hexa-Decimal String to Integer

The string should contain only the Hexadecimal symbols i.e., 0-9 and A, B, C, D, E or F. Base should be 16.

```
<<< a = int("2A9", 16)
<<< a
681
```

Decimal equivalent of Hexadecimal 2A9 is 681. You can easily verify these conversions with calculator app in Windows, Ubuntu or Smartphones.

Python float() Function

The **float()** is a built-in function in Python. It returns a float object if the argument is a float literal, integer or a string with valid floating point representation.

Using float() with an float object as argument is equivalent to declaring a float object directly

```
<<< a = float(9.99)
<<< a
9.99
<<< type(a)
<class 'float'>
```

Python str() Function

The str() function works the opposite. It surrounds an integer or a float object with quotes (') to return a str object. The str() function returns the string representation

of any Python object.The str() function has three parameters. First required parameter (or argument) is the object whose string representation we want. Other two operators, encoding and errors, are optional.
Ex:-
a=str(11.10)
print(a)
Print(type(a))

Python **ord() function**-Converts a single character to its integer value.

Python **hex() function** - Converts an integer to a hexadecimal string.

Python **oct() function** - Converts an integer to an octal string.

8.Disucss about various Python Operators

Python operators are special symbols used to perform specific operations on one or more operands. The variables, values, or expressions can be used as operands. For example, Python's addition operator (**+**) is used to perform addition operations on two variables, values, or expressions.

The following are some of the terms related to **Python operators**:

- **Unary operators**: Python operators that require one operand to perform a specific operation are known as unary operators.
- **Binary operators**: Python operators that require two operands to perform a specific operation are known as binary operators.
- **Operands**: Variables, values, or expressions that are used with the operator to perform a specific operation.

Operator	Name	Example
+	Addition	a + b = 30
-	Subtraction	a b = -10
*	Multiplication	a * b = 200
/	Division	b / a = 2
%	Modulus	b % a = 0
**	Exponent	a**b =10**20

//	Floor Division	9//2 = 4

Python Comparison Operators

Python Comparison operators compare the values on either side of them and decide the relation among them. They are also called Relational operators.

Operator	Name	Example
==	Equal	(a == b) is not true.
!=	Not equal	(a != b) is true.
>	Greater than	(a > b) is not true.
<	Less than	(a < b) is true.
>=	Greater than or equal to	(a >= b) is not true.
<=	Less than or equal to	(a <= b) is true.

Python Logical Operators

Python logical operators are used to form compound Boolean expressions. Each operand for these logical operators is itself a Boolean expression. For example,

Example

age > 16 and marks > 80

percentage < 50 or attendance < 75

Along with the keyword False, Python interprets None, numeric zero of all types, and empty sequences (strings, tuples, lists), empty dictionaries, and empty sets as False. All other values are treated as True.There are three logical operators in

Python Bitwise Operators

Python bitwise operators are normally used to perform bitwise operations on integer-type objects. However, instead of treating the object as a whole, it is treated as a string of bits. Different operations are done on each bit in the string.

Python has six bitwise operators - &, |, ^, ~, << and >>. All these operators (except ~) are binary in nature, in the sense they operate on two operands. Each operand is a binary digit (bit) 1 or 0.ython. They are "and", "or" and "not". They must be in lowercase.

The following are the bitwise operators in Python -

- Bitwise AND Operator
- Bitwise OR Operator
- Bitwise XOR Operator
- Bitwise NOT Operator
- Bitwise Left Shift Operator

- Biwtise Right Shift Operator

Bitwise AND operator is somewhat similar to logical and operator. It returns True only if both the bit operands are 1 (i.e. True).

Let us take two integers 60 and 13, and assign them to variables a and b respectively.

```
a=60
b=13
print ("a:",a, "b:",b, "a&b:",a&b)
```

It will produce the following **output** –

```
a: 60 b: 13 a&b: 12
```

Python Bitwise OR Operator (|)

The "|" symbol (called pipe) is the bitwise OR operator. If any bit operand is 1, the result is 1 otherwise it is 0.

Python Bitwise XOR Operator (^)

The term XOR stands for exclusive OR. It means that the result of OR operation on two bits will be 1 if only one of the bits is 1.

```
a=60
b=13
print ("a:",a, "b:",b, "a^b:",a^b)
```

Python Bitwise NOT Operator (~)

This operator is the binary equivalent of logical NOT operator. It flips each bit so that 1 is replaced by 0, and 0 by 1, and returns the complement of the original number.

Python Bitwise Left Shift Operator (<<)

Left shift operator shifts most significant bits to right by the number on the right side of the "<<" symbol. Hence, "x << 2" causes two bits of the binary representation of to right.

```
a=60
print ("a:",a, "a<<2:", a<<2)
```

Python Bitwise Right Shift Operator (>>)

Right shift operator shifts least significant bits to left by the number on the right side of the ">>" symbol. Hence, "x >> 2" causes two bits of the binary representation of to left.

```
a=60
print ("a:",a, "a>>2:", a>>2)
```

```
a: 60 a>>2: 15
```

Python Membership Operators

The membership operators in Python help us determine whether an item is present in a given container type object, or in other words, whether an item is a member of the given container type object.

Types of Python Membership Operators

Python has two membership operators: **in** and **not in.** Both return a Boolean result. The result of in operator is opposite to that of not in operator.

The 'in' Operator

The "in" operator is used to check whether a substring is present in a bigger string, any item is present in a list or tuple, or a sub-list or sub-tuple is included in a list or tuple.

Ex:-

```
var = "TutorialsPoint"
a = "P"
b = "tor"
c = "in"
d = "To"
print (a, "in", var, ":", a in var)
print (b, "in", var, ":", b in var)
print (c, "in", var, ":", c in var)
print (d, "in", var, ":", d in var)
```

The 'not in' Operator

The "not in" operator is used to check a sequence with the given value is not present in the object like string, list, tuple, etc.

Ex:-

```
var = "TutorialsPoint"
a = "P"
b = "tor"
c = "in"
d = "To"
print (a, "not in", var, ":", a not in var)
print (b, "not in", var, ":", b not in var)
print (c, "not in", var, ":", c not in var)
print (d, "not in", var, ":", d not in var)
```

Python Identity Operators

The identity operators compare the objects to determine whether they share the same memory and refer to the same object type (data type).

Python provided two identity operators; we have listed them as follows:

'is' Operator
'is not' Operator

Python 'is' Operator

The 'is' operator evaluates to True if both the operand objects share the same memory location. The memory location of the object can be obtained by the "id()" function. If the "id()" of both variables is same, the "is" operator returns True.

Ex:-

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a
# Comparing and printing return values
print(a is c)
print(a is b)
# Printing IDs of a, b, and c
print("id(a) : ", id(a))
print("id(b) : ", id(b))
print("id(c) : ", id(c))
```

Python 'is not' Operator

The 'is not' operator evaluates to True if both the operand objects do not share the same memory location or both operands are not the same objects.

Ex:-

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a

# Comparing and printing return values
print(a is not c)
print(a is not b)

# Printing IDs of a, b, and c
print("id(a) : ", id(a))
print("id(b) : ", id(b))
print("id(c) : ", id(c))
```