Day 1 Python Basics Fundamentals and OOP Concepts

Writing our first program

```
print('Bishesh Lal Shrestha')
```

```
Bishesh Lal Shrestha
```

Variables in Python

Variables need not be declared first in python.They can be used directly.Variables in python are case-sensitive as most of the other programming languages.

```
a =3
A = 4
print(a)
print(A)
```

```
3
4
```

Expressions in Python

Arithmetic operations in python can be performed by using arithmetic operators and some of the in-built functions

```
a = 2
b = 3
c = a + b  # Addition
print(c)
d = a * b  # Multiplication
print(d)
e = a - b  # Subtraction
print(e)
```

```
5
6
-1
```

Conditions in Python

Conditional output in python can be obtained by using if-else and elif (else if) statements

```
a = 3
b = 9
c = 10

if a > b and a > c:
    print(f"The greatest number is {a}")
elif b > a and b > c:
    print(f"The greatest number is {b}")
else:
    print(f"The greatest number is {c}")
```

```
The greatest number is 10
```

Functions in Python

A function in python is declared by the keyword 'def' before the name of the function. The return type of the function need not be specified explicitly in python. The function can be invoked by writing the function name followed by the parameter list in the brackets.

```
def checkDivisible(a,b):
    if a % b == 0:
        print(f"{a} is divisible by {b}")
    else:
        print(f"{a} is not divisible by {b}")

checkDivisible(4,2)
```

```
4 is divisible by 2
```

Data Types

Numeric (Integer, Float,Complex Number) Sequence Type (Strings, Tuple,List) Boolean Set Dictionary

type() function checks the type of a data types

```
# Demonstrate numeric value

a = 5
print("Type of a: ",type(a))

b = 5.0
print("\nType of b:",type(b))

c = 2 + 4j
print("\n Type of c:",type(c))
```

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

```
Type of b: <class 'float'>

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

```
## Sequnce Type

#String
string1 = "I am learning python"
print(string1)
print(type(string1))

# List
List = []
print("Initial blank list:")
print(List)

List1 = ["Bishesh","Lal","Shrestha"]
print(List1)
print(List1[0])
print(List1[2])

# Tuple
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('Python', 'Fundamentals')
Tuple3 = (Tuple1, Tuple2)
print(Tuple3)
```

```
I am learning python
<class 'str'>
Initial blank list:
[]
['Bishesh', 'Lal', 'Shrestha']
Bishesh
Shrestha
((0, 1, 2, 3), ('Python', 'Fundamentals'))
```

```
# Boolean

a = True
b = True
c = False
print(a and b)
print(a and c)
print(type(b))
```

```
True
False
<class 'bool'>
```

```
# Set
set1 = set([1,2,'Bishesh',3,4,'Shrestha'])
print(set1)
```

```
{1, 2, 3, 4, 'Shrestha', 'Bishesh'}
```

```
# Dictionary
Dict = {}
print("Empty Dictionary")
print(Dict)

Dict1 = {'name':'Bishesh','course':'python'}
print(Dict1)
```

```
Empty Dictionary
{}
{'name': 'Bishesh', 'course': 'python'}
```

Python Operators

```
# Arithmetic Operator
a = 5
b = 10

# Addition of numbers
add = a + b
# Subtraction of numbers
sub = a - b
# Multiplication of number
mul = a * b
# Division(float) of number
div1 = a / b
# Division of number
div2 = a // b
# Modulo of both number
mod = a % b


print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
```

```
15
-5
50
0.5
```

```
0
5

# Relational Operators
a = 10
b = 13

# a > b is False
print(a > b)

# a < b is True
print(a < b)

# a == b is False
print(a == b)

# a != b is True
print(a != b)

# a >= b is False
print(a >= b)

# a <= b is True
print(a <= b)
```

```
False
True
False
True
False
True
```

```
# Logical Operator
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

```
False
True
False
```

Type Conversion

```
# Implicit Type Conversion
'''
In Implicit type conversion of data types in Python,
the Python interpreter automatically converts one data type to another
without any user involvement.
'''
x = 10
print("x is a type of",type(x))

y = 10.8
print("y is a type of",type(y))

z = a + b
print("z is a type of",type(z))
```

```
x is a type of <class 'int'>
y is a type of <class 'float'>
z is a type of <class 'int'>
```

```
# Explicit conversion
'''
In Explicit Type Conversion in Python, the data type is manually
changed by the user as per their requirement.
With explicit type conversion, there is a risk of data loss since we
are forcing an expression to be changed in some specific data type.
'''

num = 10
num_str = str(num)
print(num_str, type(num_str))

str1 = "25"
str_num = int(str1)
print(str_num, type(str_num))
```

```
10 <class 'str'>
25 <class 'int'>
```

```
# Bitwise operators
a = 5
b = 10

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
```

```
print(~a)

# print bitwise XOR operation
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```

```
0
15
-6
15
1
20
```

Range Function

range() function allows user to generate a series of numbers within a given range. Depending on how many arguments user is passing to the function.

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
for i in range(2,8):
    print(i)
```

```
2
3
4
5
6
7
```

Lambda Function

Lambda functions in Python are anonymous functions that are defined using the lambda keyword. They are often used for short, simple operations where a full function definition would be excessive.

```
# A lambda function to add 10 to a given number
add_10 = lambda x: x + 10
print(add_10(10))
```

```
20
```

```
# Lambda function to add two numbers
add = lambda x, y: x + y
print(add(5, 3))
```

```
8
```

```
# Square each number in a list
numbers = [1,2,3,4]
square = list(map(lambda x: x**2,numbers))
print(square)
```

```
[1, 4, 9, 16]
```

OOP Concept in Python

Classes and Objects

Class: A blueprint for creating objects. Object: A specific thing created from a class.

```
class Person:
    def __init__(self,name,age):
        self.name = name
        self.age = age

p1 = Person('Bishesh',23)
print(p1.name)
print(p1.age)

# Note: __init__() function is called automatically every time the
class is being used to create a new object
```

```
Bishesh
23
```

```
class Person:
    def __init__(self,name):
        self.name = name

    def printName(self):
        print(f"My name is {self.name}")

p1 = Person('Bishesh Lal Shrestha')
p1.printName()

#The self parameter is a reference to the current instance of the
class, and is used to access variables that belong to the class.
```

```
My name is Bishesh Lal Shrestha
```

Encapsulation

Wrapping data (attributes) and methods (functions) together inside a class. You can restrict access to some parts of the data using private attributes (_ or __).

```
class BankAccount:
    def __init__(self,balance):
        self._balance= balance

    def deposit(self,amount):
        self._balance += amount

    def withdraw(self,amount):
        if amount <= self._balance:
            self._balance -= amount
        else:
            print("Insufficient funds")

    def check_balance(self):
        return self._balance

account = BankAccount(1000)
account.deposit(2000)
print(account.check_balance())
account.withdraw(2000)
print(account.check_balance())
```

```
3000
1000
```

Inheritance

A class can inherit attributes and methods from another class (like passing down traits in a family).

```python
class Animal:
    def sound(self):
        print("All animal make different sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")

dog = Dog()
dog.sound()
```

```
Dog barks
```

Polymorphism

An object can behave differently depending on the context.

```python
class Bird:
    def fly(self):
        print("Birds can fly")

class Penguin(Bird):
    def fly(self):
        print("Penguins cannot fly")

bird = Bird()
penguin = Penguin()

bird.fly()
penguin.fly()
```

```
Birds can fly
Penguins cannot fly
```

Abstraction

Hiding complex details and only showing what's necessary.

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Woof Woof!"
```

```
class Cat(Animal):
    def make_sound(self):
        return "Meow!"

dog = Dog()
cat = Cat()

print("Dog:", dog.make_sound())
print("Cat:", cat.make_sound())
```

```
Dog: Woof Woof!
Cat: Meow!
```