Technical skill training

Name : VIJAY KUMAR BU

Branch : ELECTRONICs AND
COMMUNICATION

USN : 1SV24EC074

PYTHON PROGRAMME

Day 1

Introduction to Python

Python is a high-level, interpreted programming language known for its simplicity and readability. It is widely used in web development, data science, automation, artificial intelligence, and more.

Key Features of Python

Easy to Learn and Use: Python has a simple syntax similar to English, making it beginner-friendly.

Interpreted Language: Code runs line-by-line, which makes debugging easier.

Cross-Platform: Works on Windows, macOS, and Linux without modification.

Versatile: Used for web development, data analysis, AI, automation, and more.

Large Community & Libraries: Thousands of libraries (e.g., NumPy, Pandas, TensorFlow) help with different tasks.

Day 2

Function

, a function is a block of reusable code that performs a specific task. Functions allow you to organize your code and make it more modular.

Here's a simple example of how to define and use a function in Python:

Syntax

```
def function_name(parameters)
```

Types of arguments

1 position
2 keyword
3 default
4 variable length

. Positional Arguments

- These are the most common type of arguments. The values are assigned to parameters in the function based on their position.

Example

```
def greet(name, age):
    print(f"Hello, {name}. You are {age} years old.")

greet("Alice", 30)  # Position matters: "Alice" goes to name, 30 goes to age
```

2. Keyword Arguments

- These are arguments passed to a function by explicitly specifying the parameter name and its corresponding value. The order of the arguments doesn't matter in this case.

Example

```
def greet(name, age):
    print(f"Hello, {name}. You are {age} years old.")

greet(age=30, name="Alice")  # Order doesn't matter here
```

3. Default Arguments

- These are arguments that have default values. If a value is not provided when calling the function, the default value is used.

Example

```
def greet(name, age=25):  # age has a default value
    print(f"Hello, {name}. You are {age} years old.")

greet("Alice")  # Uses default age (25)
greet("Bob", 30)  # Overrides default age (30)
```

4. Variable-length Arguments

- These allow you to pass a variable number of arguments to a function. There are two types:

Example:

```
def greet(*names):
    for name in names:
        print(f"Hello, {name}!")

greet("Alice", "Bob", "Charlie")  # Accepts multiple arguments

def greet(name, age):
    print(f"Hello, {name}. You are {age} years old.")

greet("Alice", 30)  # Position matters: "Alice" goes to name, 30 goes to age
```

1. `if` Statement

The `if` statement is used to test a condition. If the condition is `True`, the code block under the `if` statement is executed.

Syntax:

```
if condition:
    # Code to execute if condition is True
```

Example:

```
age = 18
if age >= 18:
    print("You are an adult.")
```

2. `else` Statement

The `else` statement is used after an `if` statement. It will execute the code block if the condition in the `if` statement is `False`.

Syntax:

```
if condition:
    # Code to execute if condition is True
else:
    # Code to execute if condition is False
```

Example:

```
age = 16
if age >= 18:
    print("You are an adult.")
else:
    print("You are a minor.")
```

3. `elif` Statement (Else If)

The `elif` statement is used to test multiple conditions. It allows you to check several conditions in sequence. If one of the conditions evaluates to `True`, the corresponding block of code is executed.

Syntax:

```
if condition1:
    # Code to execute if condition1 is True
elif condition2:
    # Code to execute if condition2 is True
else:
    # Code to execute if none of the conditions are True
```

Example:

```
age = 20
if age >= 18:
    print("You are an adult.")
elif age >= 13:
    print("You are a teenager.")
else:
    print("You are a child.")
```

In Python, loops allow you to repeatedly execute a block of code as long as a specified condition is `True`. Python provides two primary types of loops: **`for`** loops and **`while`** loops.

1. `for` Loop

The `for` loop is used to iterate over a sequence (like a list, tuple, dictionary, string, etc.) or other iterable objects. It executes a block of code for each item in the sequence.

Syntax:

```
for item in iterable:
    # Code to execute for each item
```

Example 1: Iterating over a list

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

Output:

```
apple
banana
cherry
```

2. `while` Loop

The `while` loop repeatedly executes a block of code as long as the condition is `True`. It is useful when you don't know in advance how many times you want to loop, but you have a condition that will eventually stop the loop.

Syntax:

python

```
while condition:
    # Code to execute while condition is True
```

Example 1: Simple `while` loop

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Output:

```
0
```

```
1
2
3
4
```

3. Loop Control Statements

You can control the flow of loops using the following statements:

a. `break` Statement

The `break` statement is used to exit the loop when a certain condition is met, even if the loop hasn't finished iterating over all items.

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Output:

```
0
1
2
3
4
```

b. `continue` Statement

The `continue` statement skips the current iteration and proceeds to the next iteration of the loop.

python

Copy

```
for i in range(5):
    if i == 3:
        continue  # Skip the rest of the loop when i is 3
    print(i)
```

Output:

Copy

```
0
1
2
4
```

c. `else` Clause with Loops

In Python, you can also use an `else` clause with both `for` and `while` loops. The `else` block will be executed when the loop finishes executing normally (i.e., without a `break`)

```
for i in range(5):
    print(i)
else:
    print("Loop finished!")
```

```
0
1
2
3
4
Loop finished!
```

Day 3

MODULE

Python, a module is a file that contains Python code, typically including functions, classes, and variables, which can be reused in other programs. Modules help in organizing code and making it more maintainable.

Types of Modules in Python

1. Built-in Modules – These come pre-installed with Python.

2. User-defined Modules – These are created by users to organize their code.

3. Third-party Modules – These are external libraries that need to be installed (e.g., NumPy, Pandas).

1. Built-in Modules

Python includes many modules by default. Some commonly used built-in modules:

math – Mathematical functions

random – Random number generation

datetime – Date and time manipulation

os – Interacting with the operating system

sys – System-specific parameters and functions

Example: Using a Built-in Module

```
import math

print(math.sqrt(25))  # Output: 5.0
print(math.pi)        # Output: 3.141592653589793
```

2. User-defined Modules

You can create your own module by saving Python code in a .py file.

Example: Creating and Using a User-defined Module

```
import my_module

print(my_module.greet("Alice"))  # Output: Hello, Alice!
print(my_module.pi_value)        # Output: 3.14
```

---

3. Third-party Modules

These are external libraries that you can install using pip.

Ways to Import Modules

1. Import the whole module
2. Import with an alias
3. Import specific functions
4. Import everything from a module (not recommended)

Day 4

Lists, Tuples, Dictionaries, and Sets in Python

Python provides several built-in data structures: Lists, Tuples, Dictionaries, and Sets. Each has different properties and use cases.

1. Lists (list)

A list is an ordered, mutable (changeable) collection of elements. Lists allow duplicate values and can store different data types.

Creating a List:

```
my_list = [1, 2, 3, "apple", True]
print(my_list)  # Output: [1, 2, 3, 'apple', True]
```

Accessing List Elements:

```
print(my_list[0])   # Output: 1 (first element)
print(my_list[-1])  # Output: True (last element)
```

Modifying a List:

```
my_list[1] = "banana"
print(my_list)  # Output: [1, 'banana', 3, 'apple', True]
```

List Operations:

```
my_list.append("new")   # Add element at the end
my_list.insert(1, "inserted")  # Insert at a specific index
my_list.remove("apple")  # Remove a specific element
popped_item = my_list.pop()  # Remove and return the last element
print(my_list)
```

List Slicing:

```
print(my_list[1:3])  # Output: ['banana', 3]
print(my_list[::-1])  # Reverse the list
```

Looping through a List

```
for item in my_list:
    print(item)
```

Checking Membership

```
print("apple" in my_list)  # Output: False
```

2. Tuples (tuple)

A tuple is an ordered, immutable (unchangeable) collection. Tuples allow duplicate values.

Creating a Tuple:

```
        my_tuple = (10, 20, 30, "orange")
print(my_tuple)  # Output: (10, 20, 30, 'orange')
```

Accessing Tuple Elements:

```
print(my_tuple[1])  # Output: 20
```

Tuple Operations:

Tuples cannot be modified but can be used with operations like:

```
new_tuple = my_tuple + (40, 50)  # Concatenation
print(new_tuple)  # Output: (10, 20, 30, 'orange', 40, 50)
```

Tuple Unpacking:

```
a, b, c, d = my_tuple
print(a, d)  # Output: 10 orange
```

Checking Membership

```
print(20 in my_tuple)  # Output: True
```

3. Dictionaries (dict)

A dictionary is an unordered collection of key-value pairs. Keys must be unique and immutable (e.g., strings, numbers, tuples).

Creating a Dictionary:

```
my_dict = {
   "name": "Alice",
   "age": 25,
   "city": "New York"
}
print(my_dict)
```

Accessing Dictionary Values

```
print(my_dict["name"])  # Output: Alice
print(my_dict.get("age"))  # Output: 25
```

Modifying a Dictionary:

```
my_dict["age"] = 26  # Modify existing key
my_dict["gender"] = "Female"  # Add new key-value pair
```

Dictionary Operations:

```
del my_dict["city"]  # Delete a key-value pair
my_dict.pop("gender")  # Remove and return a value
print(my_dict.keys())  # Get all keys
print(my_dict.values())  # Get all values
print(my_dict.items())  # Get key-value pairs
```

Looping through a Dictionary

```
for key, value in my_dict.items():
    print(key, "->", value)
```

Checking Key Existence

```
print("name" in my_dict)  # Output: True
```

4. Sets (set)

A set is an unordered collection of unique elements. Sets do not allow duplicates.

Creating a Set:

```
my_set = {1, 2, 3, 4, 4, 5}
print(my_set)  # Output: {1, 2, 3, 4, 5}
```

Day 5

Class

In Python, a class is a blueprint for creating objects. Classes encapsulate data (attributes) and behaviors (methods) into a single unit, allowing for code reusability and organization.

Defining a Class

A class is defined using the class keyword, followed by its name (usually in PascalCase).

```
class Car:
    # Constructor method (initializer)
    def __init__(self, brand, model, year):
        self.brand = brand  # Attribute
        self.model = model  # Attribute
        self.year = year    # Attribute

    # Method to display car details
    def display_info(self):
        print(f"{self.year} {self.brand} {self.model}")
```

Creating Objects

Once a class is defined, you can create objects (instances) of that class.

```
# Creating instances of the Car class
car1 = Car("Toyota", "Corolla", 2022)
car2 = Car("Honda", "Civic", 2023)

# Calling a method on an object
car1.display_info()  # Output: 2022 Toyota Corolla
car2.display_info()  # Output: 2023 Honda Civic
```

Key Concepts in Classes

1. Attributes: Variables associated with an object (e.g., brand, model, year).

2. Methods: Functions defined within a class that operate on its attributes (e.g., display_info).

3. __init__ Method: A special method (constructor) that initializes an object when it is created.

4. self Keyword: Represents the instance of the class and is used to access attributes and methods.

Programmes

1)
positive or negative.py

```
# find the positive or negative
```

```
a= int(input("Enter a number "))

if a>0:

   print("positive")

elif a<0:

   printf("negativr")

else:

    print("Zero”
```

2)

palandromechecker.py

```
def is_palindrome(s):

     s = s.lower().replace(" ", "")

     return s == s[::-1]

 text = input("Enter a string: ")



 if is_palindrome(text):

     print("The given string is a palindrome.")

 else:

     print("The given string is not a palindrome.")
```

3)

prepositionalargumnet.py

```
def greet(name, age):

     print(f"Hello {name}, you are {age} years old.")
```

```
 greet("krishna", 19)  # Correct order

 # greet(19, "krishna")  # Wrong order, incorrect output
```

4)
creatingmultipleobject.py

```
class Game:

  def __init__(self, name, genre):

  self.name = name

  self.genre = genre



  def display(self):

  print(f"Game: {self.name}, Genre: {self.genre}, Created by Vinayak")

 game1 = Game("BGMI", "Battle Royale")

 game2 = Game("FIFA", "Sports")

 game1.display()

 game2.display()
```

5)
bankbalance.py

```
class BankAccount:

  def __init__(self, account_number, balance=0):

   self.account_number = account_number

   self.balance = balance

  def deposit(self, amount):

   self.balance += amount
   print(f"Deposited {amount}. New balance: {self.balance}")

  def withdraw(self, amount):

    if amount <= self.balance:

      self.balance -= amount

      print(f"Withdrew {amount}. New balance: {self.balance}")

    else:

     print("Insufficient balance")

 account = BankAccount("12345")

 account.deposit(120)

 account.withdraw(52)
```

6)
areaofcircaleandtriangle.py

```
class Shape:

     def area(self, radius=None, length=None, breadth=None):

         if radius is not None:

             return 3.14 * radius * radius

         elif length is not None and breadth is not None:

             return length * breadth

         else:

             return "Invalid input"



 s = Shape()
 print("Area of circle:", s.area(radius=5))

 print("Area of rectangle:", s.area(length=4, breadth=6))
```

7)
defaultargument.py

```
def greet(name, age=18):  # Default age is 18

     print(f"Hello {name}, you are {age} years old.")

 greet("krishna")  # Uses default age (18)

 greet("radha", 20)  # Overrides default age
```

22)encapsulation.py

```
#Encapsulation

 class vikas:

   def __init__(self):

    self.pub="ok"

    self._yashes="not ok"

   def Pavan_private(self):

    print(self._yashes)

 vikas = vijay()

 print(vikas.pub)

 Vikas.yashes_private()
```

8)
add of two num.py

```
#arthmetic code

 a=int(input("enter the value of a"))
 b=int(input("enter the value of b"))
 c=a+b
 d=a-b
 e=a*b
 f=a/b
 print("addition",c)
 print("subtraction",d)
 print("multiplication",e)
 print("devision",f)
```

9)
withoutclassconstructer

```
class Calculator:
 def add(self, a, b):
             return a + b  calc = Calculator()
  result = calc.add(3, 5)
    print("Sum:", result)
```

10)
fibonacci.py

```
def fibonacci(n):
 sequence = []
 a, b = 0, 1
 for _ in range(n):
 sequence.append(a)
```

```
a, b = b, a + b

return sequence

terms = int(input("Enter the number of terms: "))

print("Fibonacci sequence:", fibonacci(terms))
```