Assignment :Advance Python Programming

Q 1. Introduction to the print() function in Python.

Ans.

The print() function is one of the most commonly used built-in functions in Python. It is used to display output on the screen (typically the console).

Code:

print("Hello, World!")

Q 2. Formatting outputs using f-strings and format().

Ans.

Python provides powerful ways to format strings — the most common are:

* f-strings (formatted string literals) – introduced in Python 3.6
* .format() method – older but still widely used

1. Using f-strings (Fast & Readable):

Code:

name = "Alice"

age = 25

print(f"My name is {name} and I am {age} years old.")

1. Using .format() Method:

Code:

print("My name is {} and I am {} years old.".format(name, age))

Q 3. Using the input() function to read user input from the keyboard.

Ans:

The input() function is used to take input from the user via the keyboard. It always returns the input as a string, so conversion is often needed.

Code:

name = input("Enter your name: ")

print(f"Hello, {name}!")

Q 4. Converting user input into different data types (e.g., int, float, etc.).

Ans.

1. Using input() to Read Input

The input() function is used to take input from the user via the keyboard. It always returns the input as a string, so conversion is often needed.

Code:

name = input("Enter your name: ")

print(f"Hello, {name}!")

2. Converting User Input to Other Data Types

Since input() returns a string, use functions like int(), float(), etc., to convert the input.

🔹 Convert to int

age = int(input("Enter your age: "))

print(f"You are {age} years old.")

🔹 Convert to float

price = float(input("Enter the price: "))

print(f"The price is ₹{price:.2f}")

🔹 Convert to bool (Note: This is tricky)

answer = input("Do you want to continue? (yes/no): ")

is\_continue = answer.lower() == "yes"

print(f"Continue: {is\_continue}")

Q 5. Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').

Ans.

File Opening Modes in Python

The open() function is used to open files. The second argument specifies the mode.

Code:

file = open("filename.txt", "mode")

1. 'r' – Read

Code:

file = open("data.txt", 'r')

print(file.read())

file.close()

1. 'w' – Write

Code:

file = open("data.txt", 'w')

file.write("This will overwrite existing content.")

file.close()

1. 'a' – Append

Code:

file = open("data.txt", 'a')

file.write("\nThis will be added at the end.")

file.close()

1. 'r+' – Read & Write

Code:

file = open("data.txt", 'r+')

content = file.read()

file.seek(0)

file.write("Start: ")

file.close()

1. 'w+' – Write & Read

Code:

file = open("data.txt", 'w+')

file.write("New content!")

file.seek(0)

print(file.read())

file.close()

Q 6. Using the open() function to create and access files.

Ans.

1. Create a New File (Write Mode)

Code:

file = open("example.txt", 'w')

file.write("Hello, this is a new file.")

file.close()

1. Access (Read) an Existing File

Code:

file = open("example.txt", 'r')

content = file.read()

print(content)

file.close()

Q 7. Closing files using close().

Ans.

Always close a file after using it to free up system resources.

Code:

file.close()

Q 8. Reading from a file using read(), readline(), readlines().

Ans.

Reading from a File

To read data from a file, use the file object’s read(), readline(), or readlines() methods.

1. read()

Reads the entire content of the file.

Code:

with open("example.txt", 'r') as file:

content = file.read()

print(content)

1. readline()

Reads one line at a time.

Code:

with open("example.txt", 'r') as file:

line1 = file.readline()

line2 = file.readline()

print(line1, line2)

1. readlines()

Reads all lines into a list.

Code:

with open("example.txt", 'r') as file:

lines = file.readlines()

print(lines)

Q 9. Writing to a file using write() and writelines().

Ans.

1. write()

Writes a single string to the file.

Code:

with open("output.txt", 'w') as file:

file.write("Hello, world!\n")

file.write("Second line.")

1. writelines()

Writes a list of strings (no automatic newline).

Code:

lines = ["Line 1\n", "Line 2\n", "Line 3\n"]

with open("output.txt", 'w') as file:

file.writelines(lines)

Q 10. Introduction to exceptions and how to handle them using try, except, and finally.

Ans.

Exceptions are errors that occur at runtime, causing the program to crash unless they are handled.

Example of an exception:

x = 10 / 0 # ZeroDivisionError

Code:

try:

num = int(input("Enter a number: "))

result = 10 / num

print(result)

except ZeroDivisionError:

print("Cannot divide by zero.")

except ValueError:

print("Invalid input! Please enter a number.")

finally:

print("Program finished.")

Q 11. Understanding multiple exceptions and custom exceptions.

Ans.

1. Multiple Exceptions

You can handle different exceptions separately:

Code:

try:

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

b = 10 / a

except ValueError:

print("Please enter a valid number.")

except ZeroDivisionError:

print("Cannot divide by zero.")

1. Custom Exceptions

You can define your own exceptions using classes:

🔹 Creating a Custom Exception

Code:

class NegativeNumberError(Exception):

pass

num = int(input("Enter a positive number: "))

if num < 0:

raise NegativeNumberError("Negative numbers are not allowed.")

🔹 Handle the Custom Exception

Code:

try:

num = int(input("Enter a positive number: "))

if num < 0:

raise NegativeNumberError("Negative numbers are not allowed.")

except NegativeNumberError as e:

print(e)

Q 12. Understanding the concepts of classes, objects, attributes, and methods in Python.

Ans.

1. Class

A class is a blueprint for creating objects. It defines attributes (data) and methods (functions).

Code:

class Person:

def \_\_init\_\_(self, name, age): # Constructor

self.name = name # Attribute

self.age = age

def greet(self): # Method

print(f"Hello, my name is {self.name} and I am {self.age} years old.")

2. Object

An object is an instance of a class.

Code:

p1 = Person("Alice", 25) # p1 is an object

p1.greet() # Calling method

3. Attributes

* Variables inside a class.
* Defined using self. keyword.

Code:

self.name, self.age

4. Methods

* Functions defined inside a class.
* Always take self as the first argument.

Q 13 . Difference between local and global variables.

Ans.

1. Global Variable

* Declared outside of any function.
* Accessible anywhere in the program.

Code:

x = 10 # global

def show():

print(x) # uses global x

show()

2. Local Variable

* Declared inside a function.
* Accessible only within that function.

Code:

def show():

y = 20 # local

print(y)

show()

# print(y) → Error: y is not defined outside the function

Q 14. Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.

Ans.

1. Single Inheritance

A child class inherits from one parent class.

Code:

class Animal:

def speak(self):

print("Animal speaks")

class Dog(Animal):

def bark(self):

print("Dog barks")

d = Dog()

d.speak()

d.bark()

2. Multilevel Inheritance

A class inherits from a derived class, creating a chain.

Code:

class Animal:

def sound(self):

print("Animal sound")

class Dog(Animal):

def bark(self):

print("Dog barks")

class Puppy(Dog):

def weep(self):

print("Puppy weeps")

p = Puppy()

p.sound()

p.bark()

p.weep()

3. Multiple Inheritance

A class inherits from more than one parent class.

Code:

class Father:

def skill(self):

print("Father: Gardening")

class Mother:

def talent(self):

print("Mother: Cooking")

class Child(Father, Mother):

def hobby(self):

print("Child: Drawing")

c = Child()

c.skill()

c.talent()

c.hobby()

4. Hierarchical Inheritance

Multiple child classes inherit from the same parent class.

Code:

class Animal:

def eat(self):

print("Eating")

class Dog(Animal):

def bark(self):

print("Dog barks")

class Cat(Animal):

def meow(self):

print("Cat meows")

d = Dog()

d.eat()

c = Cat()

c.eat()

5. Hybrid Inheritance

Combination of two or more types of inheritance.

Code:

class A:

def display(self):

print("Class A")

class B(A):

def show(self):

print("Class B")

class C:

def greet(self):

print("Class C")

class D(B, C): # Hybrid: Multilevel (A→B→D) + Multiple (C, B)

pass

d = D()

d.display()

d.show()

d.greet()

Q 15. Using the super() function to access properties of the parent class

Ans.

The super() function is used to call a method or constructor from the parent class.

Code:

class Person:

def \_\_init\_\_(self, name):

self.name = name

def display(self):

print("Name:", self.name)

class Student(Person):

def \_\_init\_\_(self, name, grade):

super().\_\_init\_\_(name) # Call parent constructor

self.grade = grade

def show(self):

super().display() # Call parent method

print("Grade:", self.grade)

s = Student("Alice", "A")

s.show()

Q 16. Method overloading: defining multiple methods with the same name but different parameters.

Ans.

1.Method Overloading (Same Method, Different Parameters)  
Python doesn't support true method overloading like Java or C++.

2.Method Overriding (Same Method, Redefined in Child Class)

Method overriding happens when a child class redefines a method from the parent class.

Code:

class Animal:

def speak(self):

print("Animal speaks")

class Dog(Animal):

def speak(self): # Overriding the speak() method

print("Dog barks")

a = Animal()

a.speak() # Output: Animal speaks

d = Dog()

d.speak() # Output: Dog barks