



TECHNICAL REPORT WRITING

PYTHON AS AN OBJECT-ORIENTED PROGRAMMING

NAME: RUPAK SARKAR

MAKAUT ROLL NO.: 14271024036

STREAM: MASTER OF COMPUTER APPLICATION

SUBJECT NAME: PROGRAMMING CONCEPT WITH PYTHON

SUBJECT CODE: MCAN-101

COLLEGE NAME: MEGHNAD SAHA INSTITUTE OF
TECHNOLOGY

UNIVERSITY NAME: MAULANA ABUL KALAM AZAD
UNIVERSITY OF TECHNOLOGY

SUMMARY

Python is a powerful and versatile programming language that fully supports **Object-Oriented Programming (OOP)** principles, enabling the creation of modular, reusable, and scalable software. Core OOP concepts in Python include **encapsulation**, which bundles data and methods within classes and controls access using visibility modifiers; **inheritance**, allowing classes to derive properties and methods from others to promote reusability; **polymorphism**, enabling uniform treatment of different objects through method overriding and operator overloading; and **abstraction**, achieved via abstract base classes using the abc module. Python's dynamic features, such as duck typing and metaprogramming, enhance its flexibility, making it a preferred choice for applications ranging from web development to machine learning. Its simplicity and readability make Python ideal for leveraging OOP principles effectively, catering to both beginners and advanced developers.

This report will cover how Python acts as Object-Oriented Programming.

CONTENTS

- 1) Introduction
- 2) What is OOPS
- 3) Features of OOPS
- 4) Function Overloading
- 5) Inheritance
- 6) Conclusion
- 7) Bibliography
- 8) Acknowledgement

INTRODUCTION

What is Object-Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that organizes software design around **objects**, which represent real-world entities by encapsulating **data** (attributes) and **behaviour** (methods). OOP enables modular, reusable, and scalable software development by emphasizing concepts like:

1. **Encapsulation:** Bundling data and methods into a single unit (object) and restricting direct access to some components to ensure controlled interaction.
2. **Inheritance:** Creating new classes (derived classes) based on existing ones (base classes), allowing the reuse of code and establishment of hierarchical relationships.
3. **Polymorphism:** Allowing objects of different classes to be treated uniformly through shared interfaces, often achieved via method overriding or overloading.
4. **Abstraction:** Hiding complex implementation details and exposing only essential features, enabling simpler interaction with objects.

OOP promotes code clarity, maintainability, and reusability, making it a widely used approach in software development for applications such as web development, game design, and system modelling. Popular OOP languages include Python, Java, C++, and Ruby.

Features of Object-Oriented Programming

The key features of **Object-Oriented Programming (OOP)** include:

1. **Encapsulation:**
 - Bundles data (attributes) and methods (functions) into a single unit called an **object**.
 - Ensures controlled access to an object's data using access modifiers (e.g., public, private, protected).
2. **Inheritance:**
 - Allows a class (child or derived class) to inherit properties and methods from another class (parent or base class).
 - Promotes code reuse and establishes hierarchical relationships.
3. **Polymorphism:**
 - Enables objects of different classes to respond to the same method or operation in different ways.
 - Achieved through method overriding (runtime polymorphism) and operator overloading (compile-time polymorphism).
4. **Abstraction:**
 - Hides the internal implementation details of an object and exposes only essential features.

- Achieved using abstract classes and interfaces in languages like Python, Java, and C++.

5. **Class and Object:**

- **Class:** A blueprint for creating objects that define attributes and methods.
- **Object:** An instance of a class representing a specific entity with data and behaviour.

6. **Dynamic Binding:**

- The process of linking a method call to its definition at runtime, supporting flexibility in polymorphism.

7. **Modularity:**

- Divides the program into smaller, manageable pieces (classes), making code easier to debug, test, and maintain.

8. **Message Passing:**

- Objects communicate with each other by invoking methods (sending messages), facilitating interaction between objects.

These features make OOP a robust and flexible paradigm for designing complex, reusable, and maintainable software systems.

Function Overloading in Python

Python does not support **function overloading** in the traditional sense, as seen in languages like C++ or Java, where multiple functions with the same name but different parameter lists can coexist. Instead, Python uses **dynamic typing** and its flexible function definitions to achieve similar behavior through alternative approaches. These include:

1. Default Arguments

- Python allows specifying default values for function parameters, enabling a single function to handle different numbers of arguments.

```
def greet(name, message="Hello") :  
    print(f"{message}, {name}!")  
  
greet("Alice")  
greet("Alice", "Hi")
```

2. Using *args and **kwargs

- Functions can accept a variable number of positional (*args) or keyword arguments (**kwargs), simulating overloading behaviour.

```
def add(*args):  
    return sum(args)  
  
print(add(1, 2))  
print(add(1, 2, 3, 4))
```

Inheritance in Python

Python supports **inheritance**, a key feature of Object-Oriented Programming (OOP), by allowing classes to derive properties and behaviours from other classes. This promotes code reuse, modularity, and hierarchical organization. Python implements inheritance through the following mechanism:


```
class Animal:  
    def cry(self):  
        print("Animal speaks")  
  
class Dog(Animal):  
    def cry(self):  
        print("Dog barks")  
  
dog = Dog()  
dog.cry()
```

Python supports inheritance through single, multiple, multilevel, hierarchical, and hybrid inheritance patterns. Features like method overriding, the super() function, and MRO enhance flexibility and ensure clarity in class relationships. These capabilities make Python well-suited for building robust and reusable software systems.

CONCLUSION

In conclusion, Python demonstrates its strength as an object-oriented programming language by effectively implementing core OOP principles such as encapsulation, inheritance, polymorphism, and abstraction. Its ability to support function overloading through dynamic typing, default arguments, and decorators like **singledispatch** provides flexibility in handling diverse programming scenarios. Python also excels in inheritance with support for single, multiple, multilevel, and hybrid inheritance patterns, further empowering developers to create modular and reusable code. These features make Python an excellent choice for building scalable, maintainable, and efficient software systems, ensuring its continued relevance in the world of programming.

BIBLIOGRAPHY

 <https://en.wikipedia.org/>
 <https://www.geeksforgeeks.org/>

ACKNOWLEDGEMENT

I would like to acknowledge all those without whom this project would not have been successful. Firstly, I would wish to thank our Mathematics teacher Mr. Saubhik Goswami who guided me throughout the project and gave his immense support. He made us understand how to successfully complete this project and without him, the project would not have been complete.

This project has been a source to learn and bring our theoretical knowledge to the real-life world. So, I would really acknowledge his help and guidance for this project.

I would also like to thank my parents who have always been there whenever needed. Once again, thanks to everyone for making this project successful.