OOP (Object-Oriented Programming) and POP (Procedural-Oriented Programming) are two different programming paradigms or approaches for organizing and structuring code. They have distinct characteristics and are suitable for different types of software development. Here are the key differences between OOP and POP:

1. \*\*Basic Unit of Organization\*\*:

- \*\*OOP\*\*: In OOP, the basic unit of organization is an "object." Objects are instances of classes and encapsulate both data (attributes) and behavior (methods).

- \*\*POP\*\*: In POP, the basic unit of organization is a "procedure" or "function." The focus is on writing procedures or functions to perform tasks.

2. \*\*Data and Behavior\*\*:

- \*\*OOP\*\*: OOP emphasizes the bundling of data and the methods that operate on that data into a single unit (i.e., the class). Objects are responsible for managing their own state.

- \*\*POP\*\*: POP treats data and functions as separate entities. Data is typically stored in variables, and functions operate on that data.

3. \*\*Encapsulation\*\*:

- \*\*OOP\*\*: OOP promotes encapsulation, which means hiding the internal state of an object and providing controlled access to it through methods. This helps in data protection and abstraction.

- \*\*POP\*\*: POP does not emphasize encapsulation to the same extent. Data is often shared more openly between functions.

4. \*\*Inheritance\*\*:

- \*\*OOP\*\*: OOP supports inheritance, allowing one class (subclass) to inherit properties and behaviors from another class (superclass). Inheritance is used for code reuse and creating class hierarchies.

- \*\*POP\*\*: POP typically does not support inheritance in the same way. Code reuse is achieved by calling functions or procedures as needed.

5. \*\*Polymorphism\*\*:

- \*\*OOP\*\*: OOP supports polymorphism, allowing objects of different classes to be treated as objects of a common superclass. Method overriding enables dynamic dispatch.

- \*\*POP\*\*: POP does not have built-in support for polymorphism to the same extent. Polymorphic behavior is less common.

6. \*\*Modularity\*\*:

- \*\*OOP\*\*: OOP promotes modularity through the use of classes and objects, making it easier to design and manage complex systems by breaking them down into smaller, reusable components.

- \*\*POP\*\*: POP also supports modularity but may rely more on procedural decomposition, breaking tasks into functions or procedures.

7. \*\*Real-World Modeling\*\*:

- \*\*OOP\*\*: OOP is often used for modeling real-world entities and relationships, making it a natural choice for applications that closely mimic real-world scenarios.

- \*\*POP\*\*: POP is more focused on solving problems algorithmically and may not map as directly to real-world modeling.

8. \*\*Examples\*\*:

- \*\*OOP\*\*: Languages like C++, Java, Python, and C# are commonly associated with OOP.

- \*\*POP\*\*: Languages like C, Pascal, and early versions of Fortran are often associated with POP.

It's important to note that the choice between OOP and POP depends on the specific requirements of a project. OOP is well-suited for large, complex systems where modeling real-world entities and their interactions is essential. POP, on the other hand, can be more straightforward and suitable for smaller, task-oriented programs. In practice, many modern languages support both paradigms to varying degrees, allowing developers to choose the approach that best fits their needs.