Object-Oriented Programming Overview

Definition and Principles:
Object-Oriented Programming (OOP) views code through a data-centric lens rather than a procedure-centric one. With the advent of languages like C++ and Java, OOP has become a primary organizing principle in programming. Well-written OOP code emphasizes three key characteristics:

1. **Polymorphism**: The ability of different data types to be processed using a unified interface.
2. **Inheritance**: The mechanism by which new classes can derive properties and behaviors from existing ones, fostering code reuse.
3. **Encapsulation**: Bundling data and methods that operate on that data within a single unit, or class.

Advantages:
Together, these principles support the development of large, complex systems by providing a modular structure, enabling easier code maintenance and scalability.

OOP in Python:
Python supports OOP and allows the use of custom data types (classes) to represent objects. Objects, in this context, combine primitive data types into a meaningful, computationally manipulable unit. Python classes also allow:

- **Operator Overloading**: Enables operators (e.g., `+`) to behave differently based on the data types in use (e.g., addition for integers vs. concatenation for strings).
- **Object Inheritance**: New objects can be defined by extending existing ones, allowing focus on unique differences rather than starting from scratch.
- **Controlled Access**: Encapsulation within objects restricts access to certain information, fostering data security and modularity.

Example of Creating a Class and Instances in Python:

```
class Foo:
    pass

# Creating instances
a = Foo()
b = Foo()

# Checking types
print(type(a))  # Output: <class '__main__.Foo'>
```

```
# Attaching variables (attributes) to instances
a.x = 3
a.y = 4
b.x = 5
b.y = a.y

# Modifying attributes
a.y = 6
print(b.y)  # Output: 4
print(a.y)  # Output: 6
```

In this example, Foo is a simple class with instances a and b, demonstrating Python's dynamic attribute assignment and modification.