Introduction to Functional Programming in Python

Functional programming is a programming paradigm that focuses on the use of **pure functions** and avoids modifying data outside the scope of a function. This approach emphasizes clean, consistent, and maintainable code. Here's a breakdown of the key concepts and examples discussed in the video.

1. Functions and Their Roles

- Functions take inputs, process them, and return outputs.
- Two types of functions:
  - **Traditional Functions**:
    - Can modify global state variables.
    - Outputs may not depend solely on inputs.
  - **Pure Functions**:
    - Do not modify any global state.
    - Always produce the same output for the same input (deterministic behavior).
    - Do not cause side effects.

Example of a Pure Function:

```
def add(a, b):
    return a + b
```

- **Properties**:
  - No side effects.
  - Output depends only on inputs.

2. Principles of Functional Programming

- **No Side Effects**: Functions avoid modifying data or states outside their scope.
- **Stand-Alone Functions**: Functions are independent and modular, enhancing code clarity and reusability.
- **First-Class Functions**: Functions can:
  - Be assigned to variables.
  - Be passed as arguments.
  - Be returned by other functions.

Example: Assigning Functions to Variables

```
def greet(name):
    return f"Hello, {name}!"

say_hello = greet
print(say_hello("Alice"))  # Output: Hello, Alice!
```

3. Built-in Functions in Python

Functional programming leverages reusable built-in functions to save development time. Examples include:

`sorted()` Function

- **Use**: Sorts a list of items.
- **Example**:

```
coffees = ["Espresso", "Latte", "Cappuccino", "Mocha"]
sorted_coffees = sorted(coffees)
print(sorted_coffees)  # Output: ['Cappuccino', 'Espresso', 'Latte', 'Mocha']
```

`map()` Function

- **Use**: Applies a given function to each item in an iterable (e.g., list).
- **Example: Reversing Strings**

```
def reverse(string):
    return string[::-1]

coffees = ["Espresso", "Latte", "Cappuccino", "Mocha"]
reversed_coffees = map(reverse, coffees)
print(list(reversed_coffees))
# Output: ['osserpsE', 'ettaL', 'oniccappaC', 'ahcoM']
```

4. Writing Custom Functions

You can create your own functions for specific tasks, enhancing the functional programming approach.

Reversing Strings with a Custom Function

```
def reverse(string):
    return string[::-1]
```

```
print(reverse("Python"))  # Output: nohtyP
```

5. Advantages of Functional Programming

1. **Clean and Elegant Code**: Functions are independent and modular.
2. **Reusable Logic**: Built-in functions handle common tasks efficiently.
3. **Maintainability**: Pure functions are predictable and easier to debug.

Summary

- Functional programming emphasizes pure functions and avoids side effects, leading to clean, maintainable code.
- Python supports functional programming through **first-class functions** and built-in utilities like `map`, `filter`, and `sorted`.
- By understanding and applying functional programming principles, you can write efficient, reusable, and elegant Python code.