Key Concepts: Using `match` Statements in Python

What is a `match` Statement?

- Introduced in **Python 3.10**, it serves as a cleaner alternative to multiple `if-elif-else` statements.
- **Purpose**:
  - Compare a variable to several values (conditions) in a concise, readable way.
  - Simplify complex conditional logic that would otherwise be messy with `if` statements.

Comparison: `if` vs. `match`

1. **`if-elif-else` Statements**:
   - Works well with a **small number of conditions**.
   - Can get **complex and verbose** with many conditions.
   - Example:

```
if http_status == 200:
    print("Success")
elif http_status == 400:
    print("Bad Request")
elif http_status == 500 or http_status == 501:
    print("Server Error")
else:
    print("Unknown")
```

2. **`match` Statements**:
   - Cleaner and more **compact** for handling many conditions.
   - Each case evaluates a condition and executes code if matched.
   - Example:

```
match http_status:
    case 200:
        print("Success")
    case 400:
        print("Bad Request")
    case 500 | 501:  # Equivalent to `or`
        print("Server Error")
    case _:
        print("Unknown")  # Default case
```

How to Use `match` Statements

1. **Basic Syntax**:

```
match variable_name:
    case value1:
        # Action for value1
    case value2:
        # Action for value2
    case _:
        # Default action if no match
```

2. **Key Features**:

- **`case`**:
  - Specifies a condition to test the variable.
- **`|` Operator**:
  - Acts as `or` to test multiple values in one case.
  - Example: `case 500 | 501`.
- **Default Case**:
  - Use `case _` to handle unmatched values (like `else`).

Practical Example: HTTP Error Codes

1. **Code Example**:

```
http_status = 200

# Using if-elif-else
if http_status == 200:
    print("Success")
elif http_status == 400:
    print("Bad Request")
elif http_status == 500 or http_status == 501:
    print("Server Error")
else:
    print("Unknown")

# Using match
match http_status:
    case 200:
        print("Success")
    case 400:
        print("Bad Request")
    case 500 | 501:
        print("Server Error")
    case _:
        print("Unknown")
```

2. **Key Outputs**:
    - `http_status = 200` → Output: `"Success"` (for both).
    - `http_status = 400` → Output: `"Bad Request"` (for both).
    - `http_status = 550` → Output: `"Unknown"` (for both).

Advantages of `match` Statements

- **Readability**: Cleaner, especially with many conditions.
- **Efficiency**: Avoids repetitive comparisons against the variable.
- **Compactness**: Combines multiple values with `|` in a single case.

Key Points to Remember

- The **`match` statement** is available only in Python **3.10 and later**.
- Default behavior is handled by `case _`, equivalent to `else`.
- Combine conditions using the **`|` operator** for `or` logic.
- Works well for fixed, known values like enums, status codes, or menu options.

Conclusion

The `match` statement simplifies the process of testing a variable against many conditions. It is a powerful tool to improve code clarity and maintainability in Python.