Python Control Flow Statements: Match Statement (Python 3.10+)

What is Structural Pattern Matching?

Introduced in Python 3.10, `match-case` is a modern way to handle data-driven decision-making. It goes beyond simple `if-elif-else` chains by letting you check the **structure** of data (like dictionaries, lists, or objects) and extract values from them. Think of it as a supercharged `switch` statement!

Why Use `match-case`?

- 🎯 **Cleaner code**: Replace messy nested `if` statements with readable patterns.
- 💡 **Extract data**: Directly pull values from complex structures (e.g., JSON).
- 🛠️ **Handle multiple cases**: Match data types, values, and even conditions in one block.

Understanding the Basics: Syntax of Match Case

The match case statement in Python begins with the keyword match, followed by an expression. This expression is the value that you want to evaluate and compare against different possibilities [1]. It's important to note that this expression is evaluated only once at the very beginning of the match statement [11].

Inside the match block, you'll find one or more case blocks. Each case starts with the keyword case and is followed by a "pattern." This pattern is what Python will try to match against the value of the expression you provided in the match statement [1]. If the pattern in a case block matches the expression's value, the block of code associated with that case will be executed [1].

Here's a basic example to illustrate the syntax. Let's say we want to check the day of the week:

Basic Syntax

```
match variable_to_check:
    case Pattern1:
        # Action for Pattern1
    case Pattern2:
        # Action for Pattern2
    case _:
        # Default action
```

Example: Basic

```
day = "Monday"
match day:
    case "Monday":
        print("Start of the week!")
    case "Friday":
        print("Almost weekend!")
    case _:
        print("Just another day.")
```

- In this example, the match statement takes the value of the day variable. It then checks each case.
- The first case checks if day is equal to "Monday". Since it is, the code print("Start of the week!") is executed.
- The case _: at the end is special. The underscore _ acts as a wildcard, representing the "default case" or a "catch-all" [1]. If none of the preceding case patterns match the value of the day variable, the code under case _: would be executed. This is similar to the else statement in an if-elif-else structure.

A significant advantage of Python's match case is that it `automatically exits` the match block as soon as it finds the first case that matches 1. This is different from the switch statement in some other programming languages where you might need to use explicit break statements to prevent the code from "falling through" to the next case. This automatic exiting makes the syntax cleaner and helps reduce potential errors for beginners [1].

Pattern Matching in Action: More Than Just Equality

While the basic example above shows simple equality checks, the real power of match case lies in its ability to perform more sophisticated "pattern matching" 1. This means you can check for more than just whether a value is equal to a specific literal.

Let's look at some examples of matching with literal values:

Example: Integers - You can easily check if a number matches specific integer values [1].

```
http_code = 404
match http_code:
    case 200:
        print("OK")
    case 404:
        print("Not Found")
    case 500:
        print("Server Error")
    case _:
        print("Unknown Status")
```

Example: Strings - Matching against specific text inputs or commands is straightforward [1].

```
command = "start"
match command:
    case "start":
        print("Starting the process...")
    case "stop":
        print("Stopping the process...")
    case "help":
        print("Showing help documentation...")
```

```
    case _:
        print("Unknown command.")
```

Example: Booleans - You can also match against the boolean values True or False 11.

```
is_weekend = True
match is_weekend:
    case True:
        print("It's the weekend!")
    case False:
        print("Back to work.")
```

The match case statement also allows you to use the `OR operator`, represented by the pipe symbol `\|`, within a case to check against multiple values at once 1. This can make your code even more concise when you want to perform the same action for several different values.

```
day = "Sunday"
match day:
    case "Saturday" | "Sunday":
        print("It's a weekend day.")
    case "Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday":
        print("It's a weekday.")
    case _:
        print("Invalid day.")
```

for more details, Say Goodbye to Long If-Elif Chains with Python's Match Case