Structural Pattern Matching

"Structural Pattern Matching is *not* a `switch` statement!"

Me, hundreds of times

Structural Pattern Matching

The History

```
if entrée == "Spam":
    print(f"Yum, {entrée}!")
elif entrée == "eggs":
    print(f"Ew, {entrée}.")
else:
    print(f"Hm, {entrée}?")
```

Structural Pattern Matching

The History

```
if len(meal) == 2:
    print("Yay, an entrée and a side!")
elif len(meal) == 1:
    print("I guess I don't get a side.")
else:
    print("Do I have too much food, or none at all?")
```

Structural Pattern Matching

The History

```
entrée, side = meal
```

Structural Pattern Matching

The History

```
entrée = meal[0]

side = meal[1]
```

Structural Pattern Matching

The History

```
entrée = meal["entrée"]

side = meal["side"]
```

Structural Pattern Matching

The History

```
entrée = meal.entrée

side = meal.side
```

The Design

Syntax

Syntax
The Design

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case [entrée, *sides]:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) >= 1

):

    entrée, *sides = meal

    ...
```

Syntax
The Design

```
# Python 3.10

match entrée:

    case "Spam":

        ...
```

```
# Python 3.9

if entrée == "Spam":

    ...
```

Syntax
The Design

```
# Python 3.10

match entrée:

    case "Spam" | "eggs":

        ...
```

```
# Python 3.9

if (

    entrée == "Spam"

    or entrée == "eggs"

):

    ...
```

Syntax
The Design

```
# Python 3.10

match serve():

    case "Spam" | "eggs" as e:

        ...
```

```
# Python 3.9

_subject = serve()

if (

    _subject == "Spam"

    or _subject == "eggs"

):

    e = _subject

    ...
```

Syntax
The Design

```
match p:

    case Point(0, 0) | {"x": 0, "y": 0}:

        print("At the origin!")

    case Point(0, y) | {"x": 0, "y": y}:

        print(f"On the y-axis at {y = }!")

    case Point(x, 0) | {"x": x, "y": 0}:

        print(f"On the x-axis at {x = }!")

    case Point(x, y) | {"x": x, "y": y} if x == y:

        print(f"On the diagonal at x = {y = }")
```

Syntax
The Design

```
def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

            return n * f(n - 1)
```

Syntax
The Design

```rust
// Rust

fn f(n: u64) -> u64 {

    match n {

        0 | 1 => 1,

        _ => n * f(n - 1),

    }

}
```

```scala
// Scala

def f(n: Int): Int =

  n match {

    case 0 | 1 => 1

    case _     => n * f(n - 1)

  }
```

```python
# Python

def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

            return n * f(n - 1)
```

Syntax
The Design

```rust
// Rust

fn f(n: u64) -> u64 {

    match n {

        0 | 1 =>

            return 1,

        _ =>

            return n * f(n - 1),

    }

}
```

```scala
// Scala

def f(n: Int): Int =

    n match {

        case 0 | 1 =>

            return 1

        case _ =>

            return n * f(n - 1)

    }
```

```python
# Python

def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

            return n * f(n - 1)
```

Syntax
The Design

```rust
// Rust

fn f(n: u64) -> u64 {
    match n {
        0 | 1 =>
            return 1,
        _ =>
            return n * f(n - 1),
    }
}
```

```scala
// Scala

def f(n: Int): Int =
    n match {
        case 0 | 1 =>
            return 1
        case _ =>
            return n * f(n - 1)
    }
```

```python
# Python

def f(n: int) -> int:
    match n:
        case 0 | 1:
            return 1
        case _:
            return n * f(n - 1)
```

The Implementation

The Structural Pattern Matching Compiler

The SPaM Compiler

Soft Keywords

Soft Keywords
The Implementation

```
import re
match = re.match(
    r"(.*) is (closed|still under investigation).",
    "The Case of the Missing Spam is still under investigation.",
)
if match is not None:
    case, status = match
    if status == "closed":
        print(f"Wow, they finally solved {case}!")
    elif status == "still under investigation":
        print(f"I wonder when they will solve {case}!")
else:
    print("Why aren't they looking into this?")
```

Soft Keywords
The Implementation

```
import re

match = re.match(
    r"(.*) is (closed|still under investigation).",
    "The Case of the Missing Spam is still under investigation.",
)
match match:
    case case, "closed":
        print(f"Wow, they finally solved {case}!")
    case case, "still under investigation":
        print(f"I wonder when they will solve {case}!")
    case None:
        print("Why aren't they looking into this?")
```

Soft Keywords
The Implementation

```
import re
match = re.match(
    r"(.*) is (closed|still under investigation).",
    "The Case of the Missing Spam is still under investigation.",
)
match match:
    case case, "closed":
        print(f"Wow, they finally solved {case}!")
    case case, "still under investigation":
        print(f"I wonder when they will solve {case}!")
    case None:
        print("Why aren't they looking into this?")
```

The Future

Improved Control Flow

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
match meal:
    case ["Spam" as entrée, side]:
        print(f"Delicious... {entrée} with {side}!")
    case ["eggs" as entrée, side]:
        print(f"Ew... {entrée} with {side}.")
    case [entrée, side]:
        print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
if isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "Spam":
    entrée, side = meal
    print(f"Delicious... {entrée} with {side}!")
elif isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "eggs":
    entrée, side = meal
    print(f"Ew... {entrée} with {side}.")
elif isinstance(meal, Sequence) and len(meal) == 2:
    entrée, side = meal
    print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
if isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "Spam":
    entrée, side = meal
    print(f"Delicious... {entrée} with {side}!")
elif isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "eggs":
    entrée, side = meal
    print(f"Ew... {entrée} with {side}.")
elif isinstance(meal, Sequence) and len(meal) == 2:
    entrée, side = meal
    print(f"That's unexpected... {entrée} with {side}?")
```

Improved Control Flow
The Future

```
if isinstance(meal, Sequence) and len(meal) == 2:
    entrée, side = meal
    if entrée == "Spam":
        print(f"Delicious... {entrée} with {side}!")
    elif entrée == "eggs":
        print(f"Ew... {entrée} with {side}.")
    else:
        print(f"That's unexpected... {entrée} with {side}?")
```

Thank you!

@brandtbucher | brandt@python.org