A Perfect `match`

The history, design, implementation, and future of Python's structural pattern matching.

Brandt Bucher (April 29, 2022)

Brandt Bucher

Brandt Bucher

- Studied computer engineering
- 5 years of Python experience
- 3 years of CPython development
- Currently part of the CPython performance engineering team at Microsoft!
- Implemented and co-authored Python's structural pattern matching proposal

The History

PEP 622

PEP 622

The History

- June 23, 2020
- Python 3.10
- 6 authors

PEP 622

PEP 634

PEPs 634/635/636

PEPs 634/635/636

The History

- September 12, 2020
- Python 3.10
- 4 authors
- PEP 634: Specification
- PEP 635: Motivation and Rationale
- PEP 636: Tutorial

Dedicated Repository

Dedicated Repository

The History

- GitHub: `gvanrossum/patma`
- An issue tracker
- A collaborative environment
- A source of information

The Design

Structural Pattern Matching

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

Me, hundreds of times

Structural Pattern Matching

The Design

- Control Flow
- Destructuring

Control Flow

Control Flow

The Design

```
if meal[0] == "Spam":
    print("Yay, Spam!")
elif meal[0] == "eggs":
    print("Oh, eggs?")
else:
    print("Hm, something else.")
```

Control Flow

The Design

```
if len(meal) == 2:

    print("Yay, an entrée with a side!")

elif len(meal) == 1:

    print("Oh, only an entrée?")

else:

    print("Hm, something else.")
```

Destructuring

The Design

```
meal = entrée, side
```

Destructuring

The Design

```
entrée, side = meal
```

Destructuring

The Design

```
entrée = meal[0]

side = meal[1]
```

Destructuring

The Design

```
entrée = meal["entrée"]

side = meal["side"]
```

Destructuring

The Design

```
entrée = meal.entrée

side = meal.side
```

Syntax

Syntax
The Design

```
match meal:
    case entrée, side:
        ...
```

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
# Python 3.10

match meal:

    case (entrée, side):

        ...
```

```python
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

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 [_, side]:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    side = meal[1]

    ...
```

Syntax

The Design

```
# Python 3.10

match meal:

    case [_, side]:

        ...

    case _:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    side = meal[1]

    ...

else:

    ...
```

Syntax

The Design

```python
# Python 3.10

match meal:

    case ["Spam", side]:

        ...

    case _:

        ...
```

```python
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

    and meal[0] == "Spam"

):

    side = meal[1]

    ...

else:

    ...
```

Syntax

The Design

```
match meal:

    case ["Spam", side]:

        ...
```

Syntax

The Design

```
match meal:
    case ["Spam" | "eggs", side]:
        ...
```

Syntax

The Design

```
match meal:
    case ["Spam", side] if not self.has_tried(side):
        ...
```

Syntax

The Design

```
match meal:
    case {"entrée": "Spam", "side": side}:
        ...
```

Syntax

The Design

```
match meal:
    case {"meal": ["Spam", side]}:
        ...
```

Syntax

The Design

```
match meal:
    case Meal(Food("Spam"), Food(side)):
        ...
```

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

fn f(n: u64) -> u64 {

    match n {

        0 | 1 => 1,

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

    }

}
```

```
// Scala

def f(n: Int): Int =

  n match {

    case 0 | 1 => 1

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

  }
```

```
# Python

def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

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

Syntax

The Design

```
// Rust

fn f(n: u64) -> u64 {

    match n {

        0 | 1 =>

            return 1,

        _ =>

            return n * f(n - 1),

    }

}
```

```
// Scala

def f(n: Int): Int =

    n match {

        case 0 | 1 =>

            return 1

        case _ =>

            return n * f(n - 1)

    }
```

```
# Python

def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

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

Syntax
The Design

```
// Rust

fn f(n: u64) -> u64 {

    match n {

        0 | 1 =>

            return 1,

        _ =>

            return n * f(n - 1),

    }

}
```

```
// Scala

def f(n: Int): Int =

    n match {

        case 0 | 1 =>

            return 1

        case _ =>

            return n * f(n - 1)

    }
```

```
# Python

def f(n: int) -> int:

    match n:

        case 0 | 1:

            return 1

        case _:

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

The Implementation

Soft Keywords

Soft Keywords
The Implementation

```
PATTERN = r"(.*) is (closed|still under investigation)."

match = re.match(PATTERN, "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

```
PATTERN = r"(.*) is (closed|still under investigation)."

match = re.match(PATTERN, "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

```
PATTERN = r"(.*) is (closed|still under investigation)."
match = re.match(PATTERN, "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 Structural Pattern Matching Compiler

The SPaM Compiler

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

STACK

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME            0 (isinstance)
  LOAD_NAME            1 (meal)
  LOAD_NAME            2 (Sequence)
  CALL_FUNCTION        2
2 POP_JUMP_IF_FALSE   18 (to end)
4 LOAD_NAME            3 (len)
  LOAD_NAME            1 (meal)
  CALL_FUNCTION        1
  LOAD_CONST           0 (2)
  COMPARE_OP           2 (==)
2 POP_JUMP_IF_FALSE   20 (to end)
6 LOAD_NAME            1 (meal)
  UNPACK_SEQUENCE      2
  STORE_NAME           4 (entrée)
  STORE_NAME           5 (side)
```

```
STACK
isinstance
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
   STACK
    meal
 isinstance
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
  STACK
 Sequence
   meal
isinstance
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

STACK

```
isinstance(meal, Sequence)
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE 18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE 20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

STACK

The SPaM Compiler
The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
STACK
 len
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE 18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE 20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

```
STACK
meal
len
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME            0 (isinstance)
  LOAD_NAME            1 (meal)
  LOAD_NAME            2 (Sequence)
  CALL_FUNCTION        2
2 POP_JUMP_IF_FALSE   18 (to end)
4 LOAD_NAME            3 (len)
  LOAD_NAME            1 (meal)
  CALL_FUNCTION        1
  LOAD_CONST           0 (2)
  COMPARE_OP           2 (==)
2 POP_JUMP_IF_FALSE   20 (to end)
6 LOAD_NAME            1 (meal)
  UNPACK_SEQUENCE      2
  STORE_NAME           4 (entrée)
  STORE_NAME           5 (side)
```

```
STACK
len(meal)
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE 18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE 20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

```
  STACK
    2
len(meal)
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
    STACK
len(meal) == 2
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE 18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE 20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

STACK

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME            0 (isinstance)
  LOAD_NAME            1 (meal)
  LOAD_NAME            2 (Sequence)
  CALL_FUNCTION        2
2 POP_JUMP_IF_FALSE   18 (to end)
4 LOAD_NAME            3 (len)
  LOAD_NAME            1 (meal)
  CALL_FUNCTION        1
  LOAD_CONST           0 (2)
  COMPARE_OP           2 (==)
2 POP_JUMP_IF_FALSE   20 (to end)
6 LOAD_NAME            1 (meal)
  UNPACK_SEQUENCE      2
  STORE_NAME           4 (entrée)
  STORE_NAME           5 (side)
```

STACK

meal

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME            0 (isinstance)
  LOAD_NAME            1 (meal)
  LOAD_NAME            2 (Sequence)
  CALL_FUNCTION        2
2 POP_JUMP_IF_FALSE   18 (to end)
4 LOAD_NAME            3 (len)
  LOAD_NAME            1 (meal)
  CALL_FUNCTION        1
  LOAD_CONST           0 (2)
  COMPARE_OP           2 (==)
2 POP_JUMP_IF_FALSE   20 (to end)
6 LOAD_NAME            1 (meal)
  UNPACK_SEQUENCE      2
  STORE_NAME           4 (entrée)
  STORE_NAME           5 (side)
```

```
 STACK
meal[0]
meal[1]
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
entrée = meal[0]
```

STACK

```
meal[1]
```

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME          0 (isinstance)
  LOAD_NAME          1 (meal)
  LOAD_NAME          2 (Sequence)
  CALL_FUNCTION      2
2 POP_JUMP_IF_FALSE 18 (to end)
4 LOAD_NAME          3 (len)
  LOAD_NAME          1 (meal)
  CALL_FUNCTION      1
  LOAD_CONST         0 (2)
  COMPARE_OP         2 (==)
2 POP_JUMP_IF_FALSE 20 (to end)
6 LOAD_NAME          1 (meal)
  UNPACK_SEQUENCE    2
  STORE_NAME         4 (entrée)
  STORE_NAME         5 (side)
```

```
entrée = meal[0]
 side = meal[1]
```

STACK

The SPaM Compiler

The Implementation

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) == 2
):
    entrée, side = meal
    ...
```

```
3 LOAD_NAME           0 (isinstance)
  LOAD_NAME           1 (meal)
  LOAD_NAME           2 (Sequence)
  CALL_FUNCTION       2
2 POP_JUMP_IF_FALSE  18 (to end)
4 LOAD_NAME           3 (len)
  LOAD_NAME           1 (meal)
  CALL_FUNCTION       1
  LOAD_CONST          0 (2)
  COMPARE_OP          2 (==)
2 POP_JUMP_IF_FALSE  20 (to end)
6 LOAD_NAME           1 (meal)
  UNPACK_SEQUENCE     2
  STORE_NAME          4 (entrée)
  STORE_NAME          5 (side)
```

```
entrée = meal[0]
 side = meal[1]
```

STACK

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

STACK

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
STACK
meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME           0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE  12 (to end)
  GET_LEN
  LOAD_CONST          1 (2)
  COMPARE_OP          2 (==)
  POP_JUMP_IF_FALSE  12 (to end)
  UNPACK_SEQUENCE     2
  STORE_NAME          1 (entrée)
  STORE_NAME          2 (side)
```

```
           STACK
isinstance(meal, Sequence)
           meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME           0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST          1 (2)
  COMPARE_OP          2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE     2
  STORE_NAME          1 (entrée)
  STORE_NAME          2 (side)
```

```
STACK
meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
  STACK
len(meal)
  meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
  STACK
    2
len(meal)
  meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
      STACK
len(meal) == 2
      meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
STACK
meal
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
  STACK
 meal[0]
 meal[1]
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME           0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE  12 (to end)
  GET_LEN
  LOAD_CONST          1 (2)
  COMPARE_OP          2 (==)
  POP_JUMP_IF_FALSE  12 (to end)
  UNPACK_SEQUENCE     2
  STORE_NAME          1 (entrée)
  STORE_NAME          2 (side)
```

```
entrée = meal[0]
```

```
STACK
meal[1]
```

The SPaM Compiler

The Implementation

```
# Python 3.10
match meal:
    case entrée, side:
        ...
```

```
2 LOAD_NAME          0 (meal)
3 MATCH_SEQUENCE
  POP_JUMP_IF_FALSE 12 (to end)
  GET_LEN
  LOAD_CONST         1 (2)
  COMPARE_OP         2 (==)
  POP_JUMP_IF_FALSE 12 (to end)
  UNPACK_SEQUENCE    2
  STORE_NAME         1 (entrée)
  STORE_NAME         2 (side)
```

```
entrée = meal[0]
 side = meal[1]
```

STACK

The Future

Improved Control Flow

Improved Control Flow

The Future

```
match meal:

    case ["Spam", side]:

        print("Yay, Spam!")

    case ["eggs", side]:

        print("Oh, eggs?")

    case [_, side]:

        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:
    case ["Spam", side]:
        print("Yay, Spam!")
    case ["eggs", side]:
        print("Oh, eggs?")
    case [_, side]:
        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:

    case ["Spam", side]:

        print("Yay, Spam!")

    case ["eggs", side]:

        print("Oh, eggs?")

    case [_, side]:

        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:
    case ["Spam", side]:
        print("Yay, Spam!")
    case ["eggs", side]:
        print("Oh, eggs?")
    case [_, side]:
        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:
    case ["Spam", side]:
        print("Yay, Spam!")
    case ["eggs", side]:
        print("Oh, eggs?")
    case [_, side]:
        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:

    case ["Spam", side]:

        print("Yay, Spam!")

    case ["eggs", side]:

        print("Oh, eggs?")

    case [_, side]:

        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:
    case ["Spam", side]:
        print("Yay, Spam!")
    case ["eggs", side]:
        print("Oh, eggs?")
    case [_, side]:
        print("Hm, something else.")
```

Improved Control Flow

The Future

```
match meal:
    case ["Spam", side]:
        print("Yay, Spam!")
    case ["eggs", side]:
        print("Oh, eggs?")
    case [_, side]:
        print("Hm, something else.")
```

Improved Control Flow

The Future

```
if isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "Spam":
    side = meal[1]
    print("Yay, Spam!")
elif isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "eggs":
    side = meal[1]
    print("Oh, eggs?")
elif isinstance(meal, Sequence) and len(meal) == 2:
    side = meal[1]
    print("Hm, something else.")
```

Improved Control Flow

The Future

```
if isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "Spam":
    side = meal[1]
    print("Yay, Spam!")
elif isinstance(meal, Sequence) and len(meal) == 2 and meal[0] == "eggs":
    side = meal[1]
    print("Oh, eggs?")
elif isinstance(meal, Sequence) and len(meal) == 2:
    side = meal[1]
    print("Hm, something else.")
```

Improved Control Flow

The Future

```
if isinstance(meal, Sequence) and len(meal) == 2:
    side = meal[1]
    if meal[0] == "Spam":
        print("Yay, Spam!")
    elif meal[0] == "eggs":
        print("Oh, eggs?")
    else:
        print("Hm, something else.")
```

Improved Reachability Checks

Improved Reachability Checks

The Future

```
for number in range(100):
    match number % 5, number % 3:
        case _, 0: print("Spam!")
        case 0, _: print("Eggs?")
        case 0, 0: print("Spam and eggs.")
        case _, _: print(number)
```

Improved Reachability Checks

The Future

```
for number in range(100):
    match number % 5, number % 3:
        case _, 0: print("Spam!")
        case 0, _: print("Eggs?")
        case 0, 0: print("Spam and eggs.")
        case _, _: print(number)
```

Improved Reachability Checks

The Future

```
        case 0, 0: print("Spam and eggs.")
```

Improved Reachability Checks

The Future

```
SyntaxWarning: pattern is unreachable

        case 0, 0: print("Spam and eggs.")
```

Improved Reachability Checks

The Future

```
        case 0, 0: print("Spam and eggs.")
```

Improved Reachability Checks

The Future

```
for number in range(100):
    match number % 5, number % 3:
        case _, 0: print("Spam!")
        case 0, _: print("Eggs?")
        case 0, 0: print("Spam and eggs.")
        case _, _: print(number)
```

Improved Reachability Checks

The Future

```
for number in range(100):
    match number % 5, number % 3:
        case 0, 0: print("Spam and eggs.")
        case _, 0: print("Spam!")
        case 0, _: print("Eggs?")
        case _, _: print(number)
```

Thank you!

@brandtbucher

Thank you!

@brandtbucher | brandt@python.org

And now, for something...

And now, for something... else?

```
match meal:
    case [entrée, side]:
        print(f"Yay, {entrée} with {side}!")
    case [entrée]:
        print(f"Oh, only {entrée}?")
    else:
        print("Hm, something else.")
```

```
match meal:
    case [entrée, side]:
        print(f"Yay, {entrée} with {side}!")
    case [entrée]:
        print(f"Oh, only {entrée}?")
else:
    print("Hm, something else.")
```

Thank you!

@brandtbucher | brandt@python.org