A Perfect `match`

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

Brandt Bucher (October 16, 2021)

- Background in computer engineering
- 4 years of Python experience
- 2 years of CPython development
- 1 year of CPython core development
- Implemented and co-authored the new structural pattern matching proposal
- Currently working on the new Python performance team at Microsoft!

- The History
- The Design
- The Implementation
- The Future

The History

Switch Statements

Switch Statements

The History

```
switch (meal.size()) {
    case 2:
        printf("Yay, an entrée and a side!\n");
        break;
    case 1:
        printf("I guess I don't get a side.\n");
        break;
    default:
        printf("Do I have too much food, or none at all?\n");
        break;
}
```

Switch Statements

The History

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

Switch Statements

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?")
```

PEP 275

PEP 275

The History

- "Switching on Multiple Values"
- November 10th, 2001
- Python 2.6
- 1 author
- 1,418 words
- "A similar PEP for Python 3000, PEP 3103, was already rejected, so this proposal has no chance of being accepted either."

PEP 3103

PEP 3103

The History

- "A Switch/Case Statement"
- June 25th, 2006
- Python 3.0
- 1 author
- 4,022 words
- "A quick poll during my keynote presentation at PyCon 2007 shows this proposal has no popular support. I therefore reject it."

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
```

PEP 622

PEP 622

The History

- "Structural Pattern Matching"
- June 23rd, 2020
- Python 3.10
- 6 authors
- 12,706 words

PEPs 634/635/636

PEPs 634/635/636

The History

- "Structural Pattern Matching: Specification"
- "Structural Pattern Matching: Motivation and Rationale"
- "Structural Pattern Matching: Tutorial"
- September 12th, 2020
- Python 3.10
- 4 authors
- 15,472 words

DLS 2020

DLS 2020

The History

- "Dynamic Pattern Matching with Python"
- 16th ACM SIGPLAN International Symposium on Dynamic Languages
- November 17th, 2020
- 5 authors
- 11,217 words

The Design

Dedicated Repository

Dedicated Repository
The Design

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

Syntax

Syntax
The Design

```
# Python 3.10

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 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, 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 _:

        ...
```

```
# Python 3.9

if True:

    ...
```

Syntax
The Design

```
# Python 3.10

match entrée:

    case e:

        ...
```

```
# Python 3.9

if True:

    e = entrée

    ...
```

Syntax
The Design

```
# Python 3.10

match entrée:

    case Food.SPAM:

        ...
```

```
# Python 3.9

if entrée == Food.SPAM:

    ...
```

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 cook():

    case "Spam" | "eggs" as e:

        ...
```

```
# Python 3.9

_subject = cook()

if (

    _subject == "Spam"

    or _subject == "eggs"

):

    e = _subject

    ...
```

Syntax
The Design

```
# Python 3.10
match meal:
    case ["Spam" | "eggs" as e, *_]:
        ...
```

```
# Python 3.9
if (
    isinstance(meal, Sequence)
    and len(meal) >= 1
    and (
        meal[0] == "Spam"
        or meal[0] == "eggs"
    )
):
    e = meal[0]
    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case {"entrée": "Spam", "side": side}:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Mapping)

    and len(meal) >= 2

    and "entrée" in meal

    and meal["entrée"] == "Spam"

    and "side" in meal

):

    side = meal["side"]

    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case {"entrée": "Spam", **rest}:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Mapping)

    and len(meal) >= 1

    and "entrée" in meal

    and meal["entrée"] == "Spam"

):

    rest = dict(meal)

    del rest["entree"]

    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case Meal(entrée="Spam", side=side):

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Meal)

    and hasattr(meal, "entrée")

    and meal.entrée == "Spam"

    and hasattr(meal, "side")

):

    side = meal.side

    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case Meal("Spam", side):

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Meal)

    and hasattr(meal, Meal.__match_args__[0])

    and getattr(meal, Meal.__match_args__[0]) == "Spam"

    and hasattr(meal, Meal.__match_args__[1])

):

    side = getattr(meal, Meal.__match_args__[1])

    ...
```

Syntax
The Design

```
# Python 3.10

match meal:

    case Meal("Spam", side) if side is not None:

        ...
```

```
# Python 3.9

if (

    isinstance(meal, Meal)

    and hasattr(meal, Meal.__match_args__[0])

    and getattr(meal, Meal.__match_args__[0]) == "Spam"

    and hasattr(meal, Meal.__match_args__[1])

):

    side = getattr(meal, Meal.__match_args__[1])

    if side is not None:

        ...
```

Syntax
The Design

```
match get_coordinates():

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

        print("At the origin!")

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

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

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

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

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

Quick Stats

Quick Stats

The Implementation

- 6 languages
- 43 files
- 76 reviews
- 250 commits
- 277 days
- 24,224 lines

Compiled Bytecode

Compiled Bytecode
The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

Compiled Bytecode
The Implementation

```
# Python 3.9

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

Compiled Bytecode
The Implementation

```
# Python 3.9

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

```
>>> import dis

>>> dis.dis(...)
```

Compiled Bytecode
The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)

  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

isinstance

Compiled Bytecode
The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

meal

isinstance

Compiled Bytecode

The Implementation

```
# Python 3.9

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

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

Sequence

meal

isinstance

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

isinstance(meal, Sequence)

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

Compiled Bytecode
The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)

  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

len

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

meal

len

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

len(meal)

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

2

len(meal)

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)

  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

len(meal) == 2

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

Compiled Bytecode
The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

meal

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

meal[0]

meal[1]

Compiled Bytecode
The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

meal[1]

Compiled Bytecode

The Implementation

```
# Python 3.9

if (

    isinstance(meal, Sequence)

    and len(meal) == 2

):

    entrée, side = meal

    ...
```

```
>>> dis.dis(...)
  3           0 LOAD_NAME                0 (isinstance)
              2 LOAD_NAME                1 (meal)
              4 LOAD_NAME                2 (Sequence)
              6 CALL_FUNCTION            2
  2           8 POP_JUMP_IF_FALSE       18 (to end)
  4          10 LOAD_NAME                3 (len)
             12 LOAD_NAME                1 (meal)
             14 CALL_FUNCTION            1
             16 LOAD_CONST               0 (2)
             18 COMPARE_OP               2 (==)
  2          20 POP_JUMP_IF_FALSE       20 (to end)
  6          22 LOAD_NAME                1 (meal)
             24 UNPACK_SEQUENCE          2
             26 STORE_NAME               4 (entrée)
             28 STORE_NAME               5 (side)
```

STACK

Compiled Bytecode
The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

meal

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

isinstance(meal, Sequence)

meal

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

meal

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

len(meal)

meal

Compiled Bytecode
The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

2

len(meal)

meal

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

len(meal) == 2

meal

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

meal

Compiled Bytecode
The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

meal[0]

meal[1]

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

meal[1]

Compiled Bytecode

The Implementation

```
# Python 3.10

match meal:

    case entrée, side:

        ...
```

```
>>> dis.dis(...)

  2           0 LOAD_NAME                0 (meal)

  3           2 MATCH_SEQUENCE
              4 POP_JUMP_IF_FALSE       12 (to end)
              6 GET_LEN
              8 LOAD_CONST               1 (2)
             10 COMPARE_OP               2 (==)
             12 POP_JUMP_IF_FALSE       12 (to end)
             14 UNPACK_SEQUENCE          2
             16 STORE_NAME               1 (entrée)
             18 STORE_NAME               2 (side)
```

STACK

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}?")
```

Improved Reachability Checks

Improved Reachability Checks

The Future

```
SPAM = "Spam"

EGGS = "eggs"
```

Improved Reachability Checks
The Future

```
match entrée:
    case SPAM:
        print(f"Delicious... {entrée}!")
    case EGGS:
        print(f"Ew... {entrée}.")
    case _:
        print(f"That's unexpected... {entrée}?")
```

Improved Reachability Checks
The Future

```
match entrée:
    case SPAM:
        print(f"Delicious... {entrée}!")
    case EGGS:
        print(f"Ew... {entrée}.")
    case _:
        print(f"That's unexpected... {entrée}?")
```

Improved Reachability Checks
The Future

```
Traceback (most recent call last):
    case SPAM:
         ^^^^
SyntaxError: name capture 'SPAM' makes remaining patterns unreachable
```

Improved Reachability Checks

The Future

```
class Food:

    SPAM = "Spam"

    EGGS = "eggs"
```

Improved Reachability Checks
The Future

```
match entrée:
    case Food.SPAM:
        print(f"Delicious... {entrée}!")
    case Food.EGGS:
        print(f"Ew... {entrée}.")
    case _:
        print(f"That's unexpected... {entrée}?")
```

Improved Reachability Checks
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 Reachability Checks
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 Reachability Checks
The Future

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

Improved Reachability Checks

The Future

```
SyntaxWarning: pattern is unreachable

  case [EGGS as entrée, side]:

SyntaxWarning: pattern is unreachable

  case [entrée, side]:
```

Improved Reachability Checks
The Future

```
for number in range(100):
    match number % 5, number % 3:
        case _, 0: print("fizz")
        case 0, _: print("buzz")
        case 0, 0: print("fizzbuzz")
        case _, _: print(number)
```

Improved Reachability Checks
The Future

```
for number in range(100):
    match number % 5, number % 3:
        case _, 0: print("fizz")
        case 0, _: print("buzz")
        case 0, 0: print("fizzbuzz")
        case _, _: print(number)
```

Improved Reachability Checks
The Future

```
for number in range(100):
    match number % 5, number % 3:
        case 0, 0: print("fizzbuzz")
        case _, 0: print("fizz")
        case 0, _: print("buzz")
        case _, _: print(number)
```

Improved Reachability Checks
The Future

```
match payload:
    case {"meal": ["Spam" as entrée, side] | [entrée, "Spam" as side]}:
        print(f"I'm having {entrée} with {side}.")
    case {"meal": ["Spam" as entrée, "Spam" as side]}:
        print(f"Awesome, I'm having {entrée} with {side}!!!")
    case {"meal": [_, _]}:
        print("Where's the Spam???")
    case _:
        print("Malformed payload!")
```

Improved Reachability Checks
The Future

```
match payload:
    case {"meal": ["Spam" as entrée, side] | [entrée, "Spam" as side]}:
        print(f"I'm having {entrée} with {side}.")
    case {"meal": ["Spam" as entrée, "Spam" as side]}:
        print(f"Awesome, I'm having {entrée} with {side}!!!")
    case {"meal": [_, _]}:
        print("Where's the Spam???")
    case _:
        print("Malformed payload!")
```

Improved Reachability Checks
The Future

```
match payload:
    case {"meal": ["Spam" as entrée, "Spam" as side]}:
        print(f"Awesome, I'm having {entrée} with {side}!!!")
    case {"meal": ["Spam" as entrée, side] | [entrée, "Spam" as side]}:
        print(f"I'm having {entrée} with {side}.")
    case {"meal": [_, _]}:
        print("Where's the Spam???")
    case _:
        print("Malformed payload!")
```

Thank you!

@brandtbucher | brandt@python.org