SECTION 2. STRUCTURAL PATTERN MATCHING

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : GENERAL SYNTAX

- Structural pattern matching has been added in the form of a match statement and case statements of patterns with associated actions.
- Patterns consist of sequences, mappings, primitive data types as well as class instances.
- Pattern matching enables programs to extract information from complex data types, branch on the structure of data, and apply specific actions based on different forms of data.
- This feature introduce by PEP622 and PEP634 and will be available Python 3.10!

STRUCTURAL PATTERN MATCHING: GENERAL SYNTAX

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : SIMPLE PATTERNS

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot“
        case 401 | 403 | 404:
            return "Not allowed"
        case _:
            return "Something's wrong with the Internet"
```

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

```python
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

```python
match greeting:
    case "":
        print("Hello!")
    case name:
        print(f"Hi {name}!")
```

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

```python
match greeting:
    case "":
        print("Hello!")
    case name:
        print(f"Hi {name}!")
if name == "Santa":        # <-- might raise UnboundLocalError
    ...                    # but works fine if greeting was not empty
```

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

Wildcard Patterns

```
match data:
    case [_, _]:
        print("Some pair")
        print(_)  # Error!
```

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

Sequence Patterns

```
match collection:
    case 1, [x, *others]:
        print("Got 1 and a nested sequence")
    case (1, x):
        print(f"Got 1 and {x}")
```

- To match a sequence pattern the subject must be an instance of collections.abc.Sequence
- it cannot be any kind of string (str, bytes, bytearray). It cannot be an iterator.

STRUCTURAL PATTERN MATCHING : PATTERNS WITH LITERALS AND VARIABLES

- The _ wildcard can be starred to match sequences of varying lengths. For example:
  - [*_] matches a sequence of any length.
  - (_, _, *_), matches any sequence of length two or more.
  - ["a", *_, "z"] matches any sequence of length two or more that starts with "a" and ends with "z".

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : MAPPING PATTERNS

- Mapping pattern is a generalization of iterable unpacking to mappings
- Its syntax is similar to dictionary display but each key and value are patterns "{" (pattern ":" pattern)+ "}“
- A **rest pattern is also allowed, to extract the remaining items. Only literal and constant value patterns are allowed in key positions

STRUCTURAL PATTERN MATCHING : MAPPING PATTERNS

```
import constants

match config:
    case {"route": route}:
        process_route(route)
    case {constants.DEFAULT_PORT: sub_config, **rest}:
        process_config(sub_config, rest)
```

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : CLASS PATTERNS

- A class pattern provides support for destructuring arbitrary objects
- There are two possible ways of matching on object attributes:
  - by position like Point(1, 2)
  - by name like Point(x=1, y=2).
- These two can be combined, but a positional match cannot follow a match by name. Each item in a class pattern can be an arbitrary pattern

STRUCTURAL PATTERN MATCHING : CLASS PATTERNS

```python
match shape:
    case Point(x, y):
        ...
    case Rectangle(x0, y0, x1, y1, painted=True):
        ...
```

STRUCTURAL PATTERN MATCHING : CLASS PATTERNS

```python
class Coordinate:
    __match_args__ = ['x', 'y', 'z']

    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

coordinate = Coordinate(1, 2, 3)
match Coordinate:
    case Coordinate(0, 0, 0):
        print('Zero Coordinate')
    case Coordinate(x, y, z) if z == 0:
        print('Coordinate in the plane Z')
    case _:
        print('Another Coordinate')
```

STRUCTURAL PATTERN MATCHING : ROADMAP

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard

STRUCTURAL PATTERN MATCHING : GUARD

```python
match point:
    case Point(x, y) if x == y:
        print(f"The point is located on the diagonal Y=X at {x}.")
    case Point(x, y):
        print(f"Point is not on the diagonal.")
```

Exercise 4

- Open your favorites IDE for python
- First task:
  - We get some string with current time (e.g. 19:30, 19:30:32 or 19). You need print separately hours, minutes and seconds if it present, otherwise we need print “00” instead
- Second task:
  - Get maximum element from list by pattern matching features
- Run these tasks under Python 3.10 and check result

PATTERN MATCHING : REVIEW

- General Syntax
- Simple patterns
- Patterns with literal and variable
- Mapping Patterns
- Class Patterns
- Guard