

강의교안 이용 안내

- 본 강의교안의 저작권은 도서출판 홍릉에 있습니다.
- 불법한 PDF 파일이 사용되지 않도록 지도 바랍니다.
- 이 자료를 무단으로 전제하거나 배포할 경우 저작권법 136조에 의거하여 벌금에 처할 수 있고 이를 병과(併科)할 수도 있습니다.

단원 09

패키지 활용과 새로운 기능: 패턴 매칭과 타입 힌트

인공지능소프트웨어학과

강환수 교수



Section 1. 패키지 활용

-



모듈 operator 개요와 산술 연산자

- 내장 함수 eval(s)

- 문자열 s로 표현된 파이썬 표현식을 실행하고 결과를 반환

```
>>> a, b = 10, 4
>>> a + b, sum([a, b])
(14, 14)
>>> a - b, eval('a - b')
(6, 6)
>>> a * b, eval('a * b')
(40, 40)
>>> a / b, eval('a / b')
(2.5, 2.5)
```

- 표준 모듈 operator

- 다양한 연산자에 대한 함수를 제공하는 모듈

```
>>> from operator import add, sub, mul, truediv
>>> a, b = 10, 4
>>> add(a, b)
14
>>> sub(a, b)
6
>>> mul(a, b)
40
>>> truediv(a, b)
2.5
```

클래스 defaultdict 개요

클래스 collections.defaultdict는 collections 모듈에 포함된 클래스

- 딕셔너리와 유사하지만 초기값(default)을 가지는 딕셔너리

- 키를 사용하여 값에 접근할 때, 만약 해당 키가 딕셔너리에 존재하지 않는다면

- 초기값으로 설정된 값으로 자동으로 초기화

❖ default_factory는 키가 없는 항목을 조회할 때 자동으로 생성할 기본값의 타입이나 함수를 지정하는 속성

- defaultdict(int)나 defaultdict(list)

- 기본 값인 0이나 빈 리스트 []와 같은 초기(default_factory) 값이 지정되는 사전이 됨

```
>>> from collections import defaultdict
>>> # 디폴트값이 int(값을 지정하지 않은 키에 대해서는 값이 0으로 지정)인 딕셔너리
>>>
>>> int_dict = defaultdict(int)
>>> print(int_dict)
defaultdict(<class 'int'>, {})
>>> print(int_dict['a'])
0
>>> print(int_dict)
defaultdict(<class 'int'>, {'a': 0})
>>> int_dict['a'] += 1
>>> int_dict['a'] += 1
>>> int_dict['b'] += 1
>>> print(int_dict)
defaultdict(<class 'int'>, {'a': 2, 'b': 1})
>>> int_dict['a']
2
>>> int_dict['b']
1
>>> []
```

❖ 딕셔너리의 서브 클래스로, 존재하지 않는 키를 조회할 때 자동으로 지정한 int의 기본값 0을 생성해주는 기능

❖ 일반 사전이라면 오류 발생

클래스 defaultdict의 첫 번째 인자의 초기 값을 list로 지정

- 사전의 키 pl의 값으로 ['java']를 지정해 객체를 생성한 구문

```
>>> from collections import defaultdict
>>> # 디폴트값이 list(값을 지정하지 않은 키에 대해서는 값이 빈 리스트 []로 지정)인 딕셔너리
>>>
>>> lst_dict = defaultdict(list, pl=['java'])
>>> print(lst_dict)
defaultdict(<class 'list'>, {'pl': ['java']})
>>> print(lst_dict['pl'])
['java']
>>> print(lst_dict['env'])
[]
>>> lst_dict['pl'] += ['kotlin']
>>> # lst_dict['pl'].append('python')
>>>
>>> lst_dict['env'] += ['window']
>>> lst_dict['env'] += ['linux']
>>> print(lst_dict)
defaultdict(<class 'list'>, {'pl': ['java', 'kotlin'], 'env': ['window', 'linux']})
>>> print(lst_dict['pl'])
['java', 'kotlin']
>>> print(lst_dict['env'])
['window', 'linux']
>>> []
```

❖ Python의 defaultdict를 사용하여 기본값이 리스트(list)인 딕셔너리를 생성하고, 키 "pl"에 초기값으로 ['java']를 설정하는 코드

❖ 일반 사전이라면 오류 발생

❖ 키가 없으므로 빈 리스트 []를 반환

사전의 메소드 `setdefault()` 활용

- 사전의 메소드 `setdefault()`로 초기값을 지정 가능
 - `setdefault()`
 - 딕셔너리에서 특정 키의 값이 존재하지 않으면 기본값을 설정하고 반환하는 메서드
 - `dict.setdefault(key, default)`
 - key가 딕셔너리에 존재하면 해당 값을 반환
 - key가 존재하지 않으면 default 값을 딕셔너리에 추가하고 반환
- `by_letter.setdefault(letter, []).append(word)`

❖ 첫 글자(letter)를 키로, 값이 없으면 지정한 []을 반환하며, 값이 있으면 그 리스트를 반환

- 첫 글자를 키로 리스트에 추가, 처음 나오는 첫 글자이면 빈 리스트인 []에 word를 추가

```
>>> words = ['강나루', '강가', '나비', '나침반', '파이썬', '파김치']
>>> by_letter = {}
>>> for word in words:
...     letter = word[0]
...     by_letter.setdefault(letter, []).append(word) # 리스트 마지막에 추가
...
>>> by_letter
{'강': ['강나루', '강가'], '나': ['나비', '나침반'], '파': ['파이썬', '파김치']}
```

클래스 defaultdict 활용

- 다음 코드를 더 간단히

```
>>> words = ['강나루', '강가', '나비', '나침반', '파이썬', '파김치']
>>> by_letter = {}
>>> for word in words:
...     letter = word[0]
...     by_letter.setdefault(letter, []).append(word) # 리스트 마지막에 추가
...
>>> by_letter
{'강': ['강나루', '강가'], '나': ['나비', '나침반'], '파': ['파이썬', '파김치']}
```

- 9-2 코딩

```
# %% 코딩 예제 09-02firstlettercount.py
from collections import defaultdict

words = ['강나루', '강가', '나비', '나침반', '파이썬', '파김치']

by_letter = defaultdict(list)
for word in words: # defaultdict는 바로 다음 코드가 가능
    by_letter[word[0]].append(word)

print(by_letter)          # 전체 출력
print(by_letter['강'])    # 각각의 키로 출력
print(by_letter['나'])    # 각각의 키로 출력
print(by_letter['파'])    # 각각의 키로 출력
```


표준 파이썬 모듈 random

던진 주사위의 수처럼 난수(random number)는 주어진 범위의 수 중에서 예측할 수 없는 임의의 수 또는 무작위 수

- 표준 모듈 random은 난수와 관련된 다양한 함수를 제공
 - 함수 randint(시작, 끝)
 - 정수 시작과 끝 수 사이 임의의 정수를 반환
 - 함수 randrange(start, stop, step)
 - start 이상 stop 미만, step 간격의 정수에서 하나의 난수를 반환

```
>>> for i in range(6):
...     n = random.randint(1, 45)
...     print(n, end=' ')
...
17 23 12 22 25 15
```

```
>>> from random import sample
>>> sample(range(1, 46), 6)
[38, 43, 9, 34, 4, 42]
>>> from random import choice, choices
>>> choice(range(1, 46))
25
>>> sorted(choices(range(1, 46), k=6))
[10, 12, 17, 29, 30, 33]
```

표준 파이썬 모듈 random 주요 함수

• 주요 함수

이름	설명	예제 코드
<code>random()</code>	0 이상 1 미만($0 \leq x < 1$)의 난수 실수를 반환	<code>random.random()</code>
<code>uniform(a, b)</code>	a 이상 b 미만($a \leq x < b$)의 난수 실수를 반환	<code>random.uniform(10, 20)</code>
<code>randrange(stop)</code>	0 이상 stop 미만($0 \leq x < \text{stop}$)의 정수를 반환	<code>random.randrange(10)</code>
<code>randrange(start, stop, step)</code>	start 이상 stop 미만($\text{start} \leq x < \text{stop}$), step 간격의 정수를 반환	<code>random.randrange(1, 10, 2)</code>
<code>randint(a, b)</code>	a 이상 b 이하($a \leq x \leq b$)의 정수를 반환	<code>random.randint(1, 6)</code>
<code>choice(seq)</code>	주어진 시퀀스에서 임의의 요소를 선택하여 반환	<code>random.choice([1, 2, 3, 4, 5])</code>
<code>choices(population, weights=None, k=1)</code>	모집단에서 중복을 허용하며 가중치를 고려하여 k개의 요소를 무작위로 추출	<code>random.choices([1, 2, 3, 4, 5], weights=[0.1, 0.2, 0.3, 0.2, 0.2], k=3)</code>
<code>shuffle(seq)</code>	시퀀스의 요소를 무작위로 섞음	<code>mylist = [1, 2, 3, 4, 5]</code> <code>random.shuffle(mylist)</code>
<code>sample(population, k)</code>	모집단에서 중복 없이 k개의 요소를 무작위로 추출	<code>random.sample([1, 2, 3, 4, 5], 3)</code>
<code>seed(a=None)</code>	시드 값을 설정하여 재현성 확보	<code>random.seed(42)</code>

외부 패키지 numpy의 난수

서드 파티 패키지가므로 pip install numpy로 설치가 필요

- 패키지 numpy의 모듈 random의 rand()와 random() 함수
 - 모듈 random의 함수 rand()
 - 1을 제외한 [0, 1) 사이의 실수인 난수를 반환
 - 함수 rand(d0, d1, d2, ...) 호출에서 인자가 있다면
 - 지정한 차수의 난수 배열을 생성
 - 모듈 random의 함수 random()
 - 1을 제외한 [0, 1) 사이의 실수인 난수를 반환
 - random((2, 3)) 호출로는 [2행 3열]의 난수 6개를 생성

```
>>> from numpy import random
>>> random.rand()
0.6908239787335428
>>> random.rand(3)
array([0.81305315, 0.81665962, 0.9347946 ])
>>> random.rand(2, 3)
array([[0.80598979, 0.96899341, 0.65393748],
       [0.17379764, 0.64080266, 0.01429333]])
>>> from numpy import random
>>> random.random()
0.6957318601516265
>>> random.random(size=(2,))
array([0.07383698, 0.77860451])
>>> random.random((2, 3))
array([[0.56252375, 0.40111048, 0.22387416],
       [0.99452768, 0.60342314, 0.82124341]])
```

패키지 numpy의 모듈 random의 randint()와 choice() 함수

• 함수 randint(low=0, high, size=None)

- high를 제외한 일정 범위 $\text{low} \leq x < \text{high}$ 내의 정수를 size로 생성

```
>>> from numpy import random
>>> random.randint(2)
0
>>> random.randint(1, 7)
2
>>> random.randint(1, 7, size=(2, 3))
array([[3, 5, 4],
       [1, 3, 1]])
```

[0, 2) 정수인 0, 1 중 하나를 반환한다.

[1, 7) 정수인 1, 2, 3, 4, 5, 6 중 하나를 반환한다.

• 함수 choice(a, size=None, replace=True, p=None)

- 주어진 배열 a에서 중복을 허용해 무작위로 size 차원의 요소를 선택
 - 매개변수 replace로 중복 허용(재선택) 여부를 지정
 - p로 a에서 선택되는 원소들의 확률을 지정

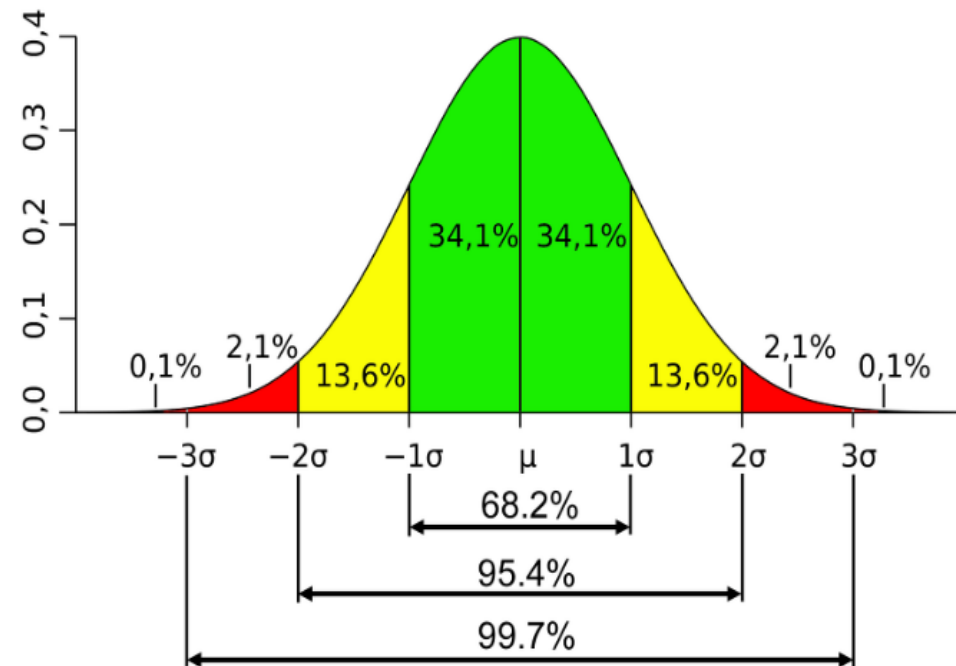
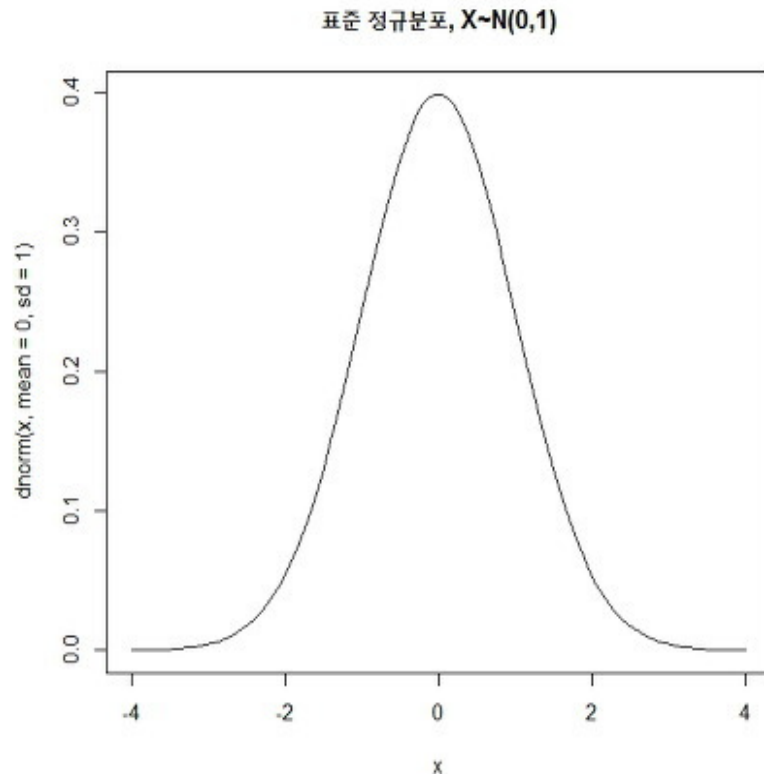
```
>>> from numpy import random
>>> sorted(random.choice(range(1, 46), 6))
[2, 2, 3, 12, 39, 44]
```

중복이 허용되므로 한 개의 수가 여러 번 나올 수 있다.

모듈 random의 확률분포의 난수 발생 함수 randn()과 uniform()

함수 randn()

- 표준 정규분포의 난수를 반환
 - randn에서 마지막 문자 n은 정규 분포에서 normal을 의미
 - 표준 정규분포는 평균이 0, 표준 편차가 1인 분포
 - 0 주위의 실수가 선택될 확률이 높으면 -3에서 3사이의 실수가 선택될 확률이 99.7%인 확률 분포

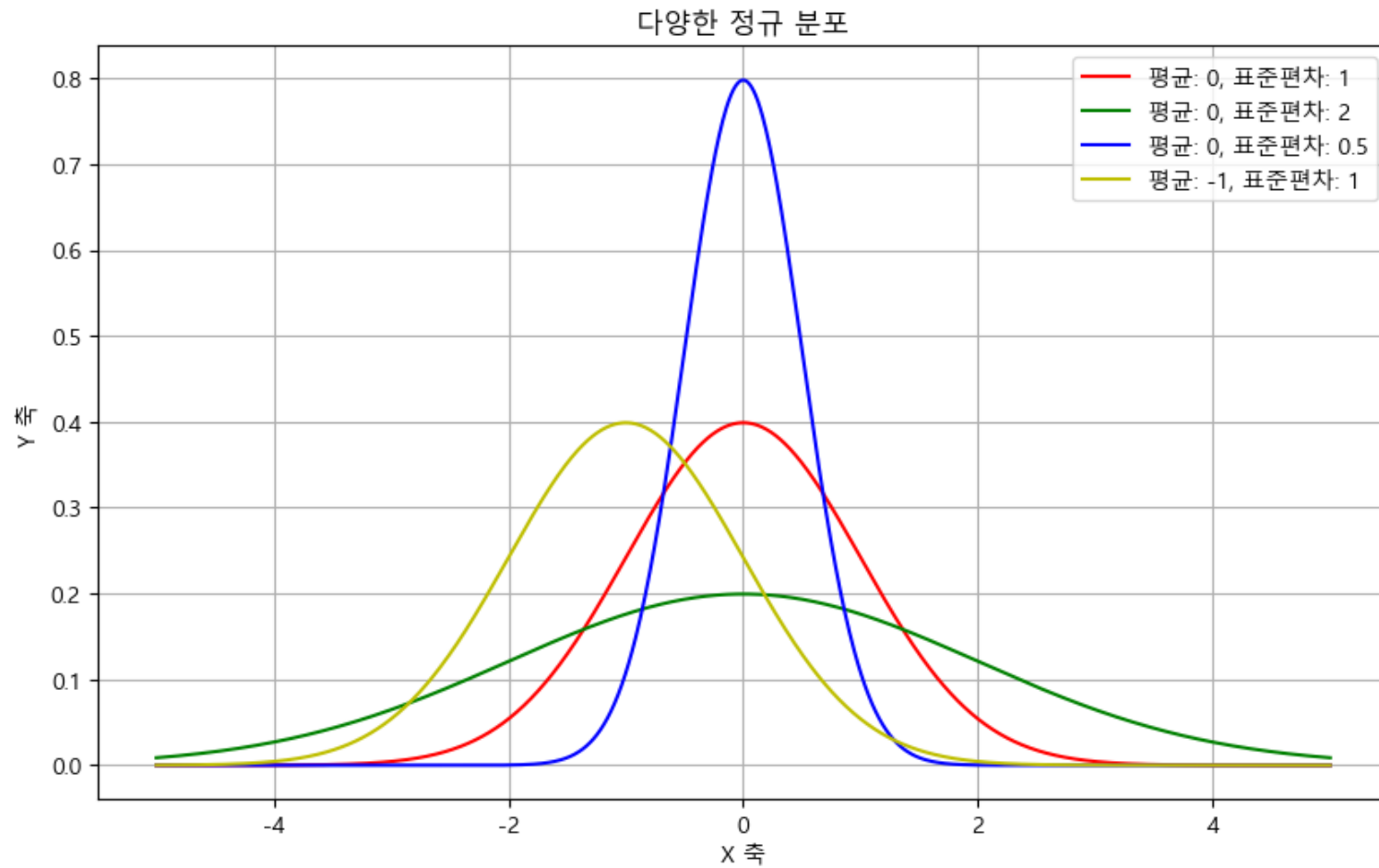


파이썬 numpy의 모듈 random 함수 정리

이름	설명	예제 코드
<code>rand(d0, d1, ..., dn)</code>	주어진 차원의 0에서 1 사이 [0.0, 1.0)의 실수인 난수 배열을 생성	<code>np.random.rand(2, 3)</code>
<code>random(size=None)</code>	0에서 1 사이 [0.0, 1.0)의 실수인 난수를 생성	<code>np.random.random((3, 3))</code>
<code>randn(d0, d1, ..., dn)</code>	주어진 차원의 표준 정규 분포에서 난수 배열을 생성	<code>np.random.randn(4, 5)</code>
<code>randint(low=0, high=None, size=None)</code>	일정 범위 $low \leq x < high$ 내의 정수를 생성, 정수 인자가 하나면 [0, 인자] 사이의 정수 난수	<code>np.random.randint(1, 100, 10)</code>
<code>uniform(low=0, high=1, size=None)</code>	주어진 범위 내의 실수인 난수를 생성	<code>np.random.uniform(2, 5, (2, 2))</code>
<code>shuffle(x)</code>	배열의 요소를 무작위로 섞기	<code>mylist = [1, 2, 3, 4, 5]</code> <code>np.random.shuffle(mylist)</code>
<code>choice(a, size=None, replace=True, p=None)</code>	주어진 배열에서 무작위로 요소를 선택	<code>np.random.choice([1, 2, 3, 4, 5], 3)</code>

9-4 코딩

- 결과



Section 2. 새로운 기능: 구조적 패턴 match와 타입 힌트



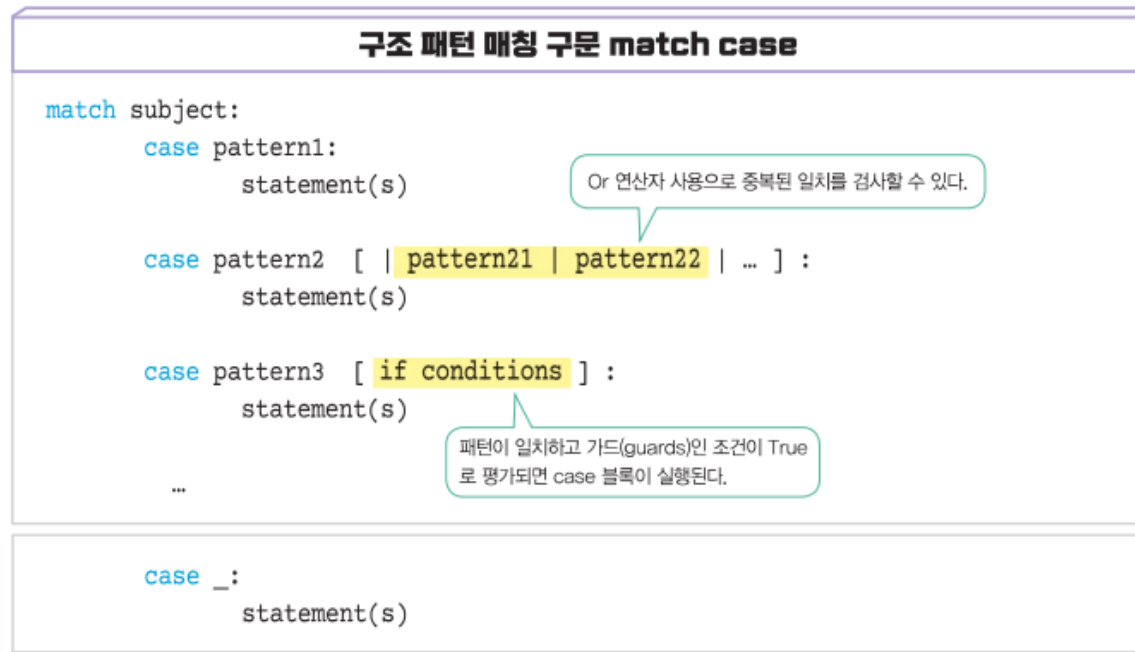
패턴 match 구문 개요

• match 구문

- C나 자바의 switch 기능보다 훨씬 더 강력한 기능을 지원
 - 상수나 리터럴 뿐만 아니라 비교할 패턴으로써 시퀀스나 클래스 인스턴스를 사용 가능
- 2020년에 발표된 파이썬 3.10의 새로운 기능으로 추가

• 구동 방식

- 여러 case 패턴 중에서 먼저 일치하는 하나의 패턴 블록만 실행한 후 종료
- (옵션) 문자 `_`
 - 일치하는 항목이 없을 때 실행되는 와일드카드 문자



패턴 match 구문 실행

구동 방식

- 여러 case 패턴 중에서 먼저 일치하는 하나의 패턴 블록만 실행한 후 종료
- (옵션) 문자 `_`
 - 일치하는 항목이 없을 때 실행되는 와일드카드 문자

구조 패턴 매칭 구문 `match case`

```
match subject:
    case pattern1:
        statement(s)

    case pattern2 [ | pattern21 | pattern22 | ... ] :
        statement(s)

    case pattern3 [ if conditions ] :
        statement(s)
    ...

case _:
    statement(s)
```

Or 연산자 사용으로 중복된 일치를 검사할 수 있다.

패턴이 일치하고 가드(guards)인 조건이 True로 평가되면 case 블록이 실행된다.

```
>>> status = 400
>>> match status:
...     case 200:
...         print("OK!")
...     case 400:
...         print("Bad request")
...     case 404:
...         print("Not found")
...     case _:
...         print("Something's wrong with the internet")
...
Bad request
>>>
```

여러 항목의 구조적 패턴 캡처(capture)

- match 구문에서 case 패턴 값은 리스트나 튜플 등 모임(collection)도 가능
 - 구조적 match 구문(왼쪽)과 대입 문장에서의 *b 사용(오른쪽)

```
>>> lst = [1, 2, 3, 4]
>>> match lst:
...     case (a, *b):
...         print(a, b)
...
1 [2, 3, 4]
```

```
>>> lst = [1, 2, 3, 4]
>>> a, *b = lst
>>> a, b
(1, [2, 3, 4])
```

- case "영희" | "미수" as name:
 - 일치하는 패턴을 변수 name에 저장
- case (name, *others):
 - friends의 첫 항목은 name에 담고, 나머지 항목을 목록으로 묶어 others에 저장

```
>>> def hello(friends):
...     match friends:
...         case "영희" | "미수" as name:
...             print(f"안녕, {name}!")
...         case (name, *others):
...             print(f"안녕, {name}, {others} 친구들!")
... 
```

구조적 패턴이 일치하고 조건이 만족하면 실행하는 보호(guards)

- 패턴 뒤에 if 조건 형식의 보호(또는 가드, guards)가 올 수 있음
 - 패턴이 일치하고 가드가 True로 평가되면 case 블록이 실행

```
>>> item = 'python'
>>>
>>> match item:
...     case float(x) if x < 3:
...         print(f"{x} < 3")
...     case float(x) if x >= 3:
...         print(f"{x} >= 3")
...     case str(x) if len(x) >= 5:
...         print(f"문자열 len('{x}')
...             >= 5")
...
문자열 len('python') >= 5
```

```
>>> item = 3.0
>>>
>>> match item:
...     case float(x) if x < 3:
...         print(f"{x} < 3")
...     case float(x) if x >= 3:
...         print(f"{x} >= 3")
...     case str(x) if len(x) >= 5:
...         print(f"문자열 len('{x}')
...             >= 5")
...
3.0 >= 3
```

▲ 그림 4 자료형과 조건 검사 match문

변수 타입 힌트

타입 어노테이션(유형 주석, type annotation) 또는 타입 힌트(유형 힌트, type hint)라 부름

- 단일 변수의 타입 힌트

- var: type_hint = value

```
>>> age = 1
>>> age: int = 1

>>> age: int
>>> age
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
```

```
>>> point : int = 77
>>> ispass : bool
>>> if point >= 60:
...     ispass = True
... else:
...     ispass = False
...
>>> ispass
True
```

모임 자료형에 대한 타입 힌트

- list, tuple, dict, set 등의 자료형 타입 힌트

- var : list[int]

For collections on Python 3.9+, the type of the collection item is in brackets

```
lst: list[int] = [1]
```

```
lst
```

For tuples of fixed size, we specify the types of all the elements

```
tp1: tuple[int, str, float] = (3, "yes", 7.5) # Python 3.9+
```

```
tp1
```

For tuples of variable size, we use one type and ellipsis

```
tp2: tuple[int, ...] = (1, 2, 3) # Python 3.9+
```

```
tp2
```

For mappings, we need the types of both keys and values

```
dt: dict[str, int] = {"python": 2021} # Python 3.9+
```

```
dt
```

```
myset: set[int] = {6, 7}
```

```
myset
```

Union Optional

- 함수 Union, 연산자 |
 - 또는 의미
- 함수 Optional[str]
 - str | None 의미

```
from typing import Union, Optional
from typing import Union
```

```
value: list[Union[int, str]] = [3, 5, "test", "fun"]
value
```

```
help(Union)
```

❖ 간결하므로 권장

```
value: list[int | str] = [3, 5, "test", "fun"] # Python 3.10+
value
```

```
from typing import Optional
```

```
# Optional[X] is the same as X | None or Union[X, None]
value: Optional[str] = 'py' # str | None
value
```

```
help(Optional)
```

함수 주석

• 함수 주석(function annotations)

- 매개변수와 반환 값에 타입 힌트(type annotations)를 작성하는 것
 - 매개변수는 변수의 타입 힌트 방법과 동일
 - 함수 헤더 뒤에 `-> return_type:` 으로 반환 값에 대한 타입 힌트

함수 주석 구문 function annotation

```
def funcname( var1: type1, var2: type2 = default_value, ...) -> return_type:
    statement(s)
```

• `var1: type1`

매개변수 `var` 뒤에 콜론과 공백 이후 자료형을 기술

• `var2: type2 = default_value`

매개변수에 기본값을 할당하는 경우, 변수: 주석 = 기본값의 형태로 할당

• `-> return_type`

매개변수 목록 괄호와 콜론 사이에 `->`와 공백, 반환 값 자료형을 기술

매개변수와 반환 값에 타입 힌트

```
>>> def plus(num1: int, num2: int) -> int:  
...     return num1 + num2
```

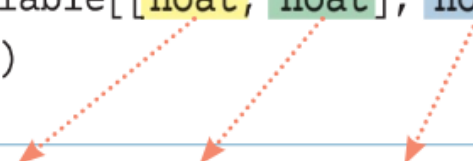
```
>>> def show(value: str, excitement: int = 10) -> None:  
...     print(value + "!" * excitement)  
...  
>>> show('Wow')  
Wow!!!!!!!!!!!!
```

매개변수인 함수의 타입 힌트

• 매개변수인 함수 func의 함수 주석

- func: Callable[[float, float], float]
 - Callable[] 내부의 [float, float]
 - 함수 func의 매개변수의 유형이며 뒤의 float는 반환 값 유형

```
>>> from typing import Callable
>>> def addmul(func: Callable[[float, float], float], a: float, b: float) -> float:
...     return func(a, b)
...
```



```
def add(a: float, b: float) -> float:
    return a + b
```

```
def addmul(func, a, b):
    return func(a, b)
```

파이썬 3.12의 새 기능

• 친절한 오류 메시지

```
>>> from math import py
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'py' from 'math' (unknown location). Did you mean: 'pi'?
```

• f-문자열

- 파이썬 3.6에서 도입
- 버전 3.12에서 기능 향상
 - 큰따옴표 "..."로 구분한 경우라도 내부 표현식에서 큰따옴표 "..."를 계속 사용 가능
 - 작은 따옴표도 가능
- 유니코드 특수 문자를 삽입 가능
 - `\N{BLACK HEART SUIT}`

```
>>> pl = {"java": 1995, "python": 1990}
>>> f"{pl["java"]} {pl["python"]}"
'1995 1990'
>>> pl = {"java": 1995, "python": 1990}
>>> f'{pl["java"]} {pl["python"]}'
'1995 1990'
```

```
>>> print(f"주요 프로그래밍 언어: {\n{BLACK HEART SUIT}"}.\n".join(pl_lst))
주요 프로그래밍 언어: java♥kotlin♥c♥go♥typescript
```