

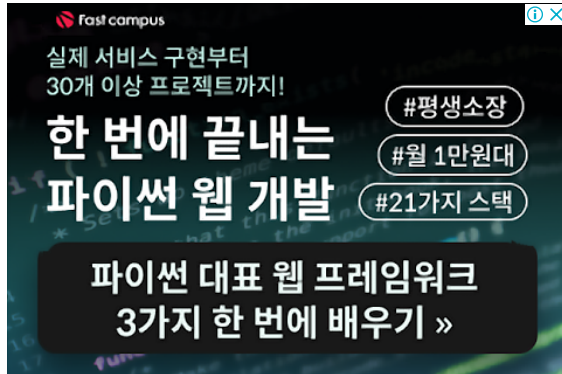


강좌 검색

홈 » 내 강좌 » 파이썬 코딩 도장 » 핵심 정리 » 핵심 정리

◀ 45.7 심사문제: 패키지 사용하기

Q & A ▶



핵심 정리

데코레이터

데코레이터는 함수를 수정하지 않은 상태에서 추가 기능을 구현할 때 사용합니다. 먼저 데코레이터는 호출할 함수를 매개변수로 받고, 호출할 함수를 감싸는 함수 wrapper를 만듭니다. 그리고 wrapper 함수 안에서는 매개변수로 받은 func를 호출하고, 함수 바깥에서는 return을 사용하여 wrapper 함수 자체를 반환합니다. 데코레이터를 사용할 때는 호출할 함수 위에 @데코레이터 형식으로 지정해줍니다.

```
def 데코레이터이름(func):    # 데코레이터는 호출할 함수를 매개변수로 받음
    def wrapper():          # 호출할 함수를 감싸는 함수
        func()              # 매개변수로 받은 함수를 호출
    return wrapper          # wrapper 함수 반환

@데코레이터                  # 데코레이터 지정
def 함수이름():
    코드
```

함수의 매개변수와 반환값을 처리하는 데코레이터

데코레이터에서 함수의 매개변수와 반환값을 처리할 때는 wrapper 함수의 매개변수를 호출할 함수의 매개변수와 똑같이 지정하고, func에 매개변수를 넣어서 호출하고 반환하면 됩니다.

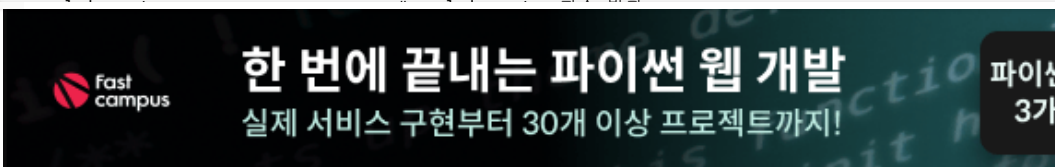
```
def 데코레이터이름(func):    # 데코레이터는 호출할 함수를 매개변수로 받음
    def wrapper(매개변수1, 매개변수2):  # 호출할 함수의 매개변수와 똑같이 지정
        return func(매개변수1, 매개변수2)  # func에 매개변수를 넣어서 호출하고 반환값을 반환
    return wrapper          # wrapper 함수 반환

@데코레이터                  # 데코레이터 지정
def 함수이름(매개변수1, 매개변수2):    # 매개변수는 두 개
    코드
```

매개변수가 있는 데코레이터

매개변수가 있는 데코레이터는 값을 지정해서 동작을 바꿀 수 있습니다. 이때는 데코레이터가 사용할 매개변수를 지정하고, 실제 데코레이터 역할을 하는 real_decorator 함수를 만듭니다. 그다음에 real_decorator 함수 안에서 wrapper 함수를 만들어줍니다.

```
def 데코레이터이름(매개변수):    # 데코레이터가 사용할 매개변수를 지정
    def real_decorator(func):    # 호출할 함수를 매개변수로 받음
        def wrapper(매개변수1, 매개변수2):  # 호출할 함수의 매개변수와 똑같이 지정
            return func(매개변수1, 매개변수2)  # func를 호출하고 반환값을 반환
        return wrapper          # wrapper 함수 반환
```



클래스로 데코레이터 만들기

클래스로 데코레이터를 만들 때는 인스턴스를 함수처럼 호출하게 해주는 `__call__` 메서드를 구현하고, `__call__` 메서드에서 호출할 함수의 매개변수를 처리해줍니다.

```
class 데코레이터이름:
    def __init__(self, func):
        self.func = func

    def __call__(self, 매개변수1, 매개변수2):
        return self.func(매개변수1, 매개변수2)

@데코레이터
def 함수이름(매개변수1, 매개변수2):
    코드
```

매개변수가 있는 데코레이터를 만들 때는 `__init__` 메서드에서 데코레이터가 사용할 매개변수를 초깃값으로 받고, `__call__` 메서드에서 호출할 함수를 매개변수를 받습니다. 그리고 `__call__` 함수 안에서 `wrapper` 함수를 만들고 호출할 함수의 매개변수를 처리해주면 됩니다.

```
class 데코레이터이름:
    def __init__(self, 매개변수):
        self.속성 = 매개변수

    def __call__(self, func):
        def wrapper(매개변수1, 매개변수2):
            return func(매개변수1, 매개변수2)
        return wrapper

@데코레이터(인수)
def 함수이름(매개변수1, 매개변수2):
    코드
```

정규표현식

정규표현식은 일정한 규칙을 가진 문자열을 표현하는 방법입니다. 문자열 속에서 특정한 규칙으로 된 문자열을 검색한 뒤 추출하거나 바꿀 때, 문자열이 정해진 규칙에 맞는지 판단할 때 사용합니다.

▼ 표 45-1 정규표현식 메타 문자

메타 문자	설명
[]	문자, 숫자 범위를 표현하며 +, -, . 등의 기호를 포함할 수 있음
{개수}	특정 개수의 문자, 숫자를 표현
{시작개수, 끝개수}	특정 개수 범위의 문자, 숫자를 표현
+	1개 이상의 문자를 표현. 예) a+b는 ab, aab, aaab는 되지만 b는 안 됨
*	0개 이상의 문자를 표현. 예) a*b는 b, ab, aab, aaab
?	0개 또는 1개의 문자를 표현. 예) a?b는 b, ab
.	문자 1개만 표현
^	[] 앞에 붙이면 특정 문자 범위로 시작하는지 판단 [] 안에 넣으면 특정 문자 범위를 제외
\$	특정 문자 범위로 끝나는지 판단
	여러 문자열 중 하나라도 포함되는지 판단
()	정규표현식을 그룹으로 묶음, 그룹에 이름을 지을 때는 ?P<이름> 형식 예) (?P<func>[a-zA-Z_][a-zA-Z0-9_]+)

▼ 표 45-2 정규표현식 특수 문자



한 번에 끝내는 파이썬 웹 개발

실제 서비스 구현부터 30개 이상 프로젝트까지!

파이썬
3가

\d	[0-9]와 같음. 모든 숫자
\D	[^0-9]와 같음. 숫자를 제외한 모든 문자
\w	[a-zA-Z0-9_]와 같음. 영문 대소문자, 숫자, 밑줄 문자
\W	[^a-zA-Z0-9_]와 같음. 영문 대소문자, 숫자, 밑줄 문자를 제외한 모든 문자
\s	[\t\n\r\f\v]와 같음. 공백(스페이스), \t, \n, \r, \f, \v을 포함
\S	[^ \t\n\r\f\v]와 같음. 공백을 제외하고 \t, \n, \r, \f, \v만 포함

정규표현식은 re 모듈을 가져와서 사용하며 다음은 정규표현식 함수 및 메서드입니다.

▼ 표 45-3 정규표현식 함수 및 메서드

함수 및 메서드	설명
match('패턴', '문자열')	문자열의 시작부터 패턴에 매칭되는지 판단, 매칭되면 매치 객체 반환
search('패턴', '문자열')	문자열의 일부분이 패턴에 매칭되는지 판단, 매칭되면 매치 객체 반환
group(그룹)	그룹에 매칭된 문자열 반환
groups()	각 그룹에 해당하는 문자열을 튜플로 반환
findall('패턴', '문자열')	패턴에 매칭된 문자열을 리스트로 반환
sub('패턴', '바꿀문자열', '문자열', 바꿀횟수)	패턴으로 특정 문자열을 찾은 뒤 다른 문자열로 바꿈, 찾은 문자열을 사용하려면 바꿀 문자열에서 \\숫자 형식으로 사용, 그룹에 이름을 지었다면 \\g<이름> 형식으로 사용(\\g<숫자> 형식도 가능). 예) re.sub('(\w+) (?P<x>\d+)', '\\g<x> \\1', 'Hello 1234')
compile('패턴')	정규표현식 패턴을 객체로 만듦
match('문자열')	문자열의 시작부터 패턴에 매칭되는지 판단, 매칭되면 매치 객체 반환
search('문자열')	문자열의 일부분이 패턴에 매칭되는지 판단, 매칭되면 매치 객체 반환

모듈 사용하기

모듈은 `import 모듈` 형식으로 가져오며 `모듈.변수`, `모듈.함수()`, `모듈.클래스()` 형식으로 사용합니다.

```
>>> import math
>>> math.sqrt(2.0)
1.4142135623730951
```

또한, `import 패키지.모듈` 형식으로 패키지의 모듈도 가져올 수 있으며 `패키지.모듈.변수`, `패키지.모듈.함수()`, `패키지.모듈.클래스()` 형식으로 사용합니다.

```
>>> import urllib.request
>>> response = urllib.request.urlopen('http://www.google.co.kr')
```

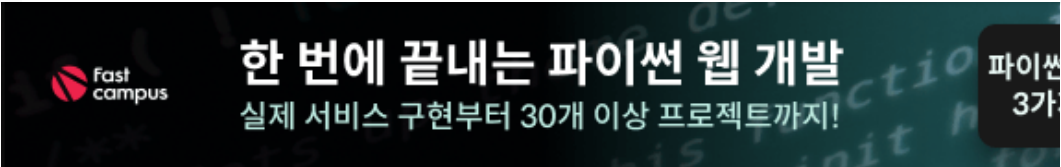
`import as`를 사용하면 모듈의 이름을 지정할 수 있습니다.

```
>>> import math as m      # math 모듈을 가져오면서 이름을 m으로 지정
>>> m.sqrt(2.0)           # m으로 제공된 함수 사용
1.4142135623730951
```

모듈의 일부만 가져오기

`from import`는 모듈의 일부만 가져옵니다. 이때는 모듈 이름을 붙이지 않고 변수, 함수, 클래스를 그대로 사용합니다.

```
>>> from math import sqrt  # math 모듈에서 sqrt 함수만 가져옴
>>> sqrt(2.0)             # sqrt 함수를 바로 사용
1.4142135623730951
```



```
>>> from math import sqrt as s      # math 모듈에서 sqrt 함수를 가져오면서 이름을 s로 지정
>>> s(2.0)                          # s로 sqrt 함수 사용
1.4142135623730951
```

from import에 *를 지정하면 해당 모듈의 모든 변수, 함수, 클래스를 가져옵니다.

```
>>> from math import *              # math 모듈의 모든 변수와 함수를 가져옴
>>> sqrt(2.0)                      # sqrt 함수 사용
1.4142135623730951
```

패키지 설치하기

파이썬 패키지 인덱스(PyPI)에서 패키지를 다운로드하여 설치할 때는 pip install 패키지 형식으로 사용합니다. 또는, python에 -m 옵션을 지정하여 pip를 실행할 수도 있습니다(리눅스, macOS에서는 python3를 사용하며 sudo를 붙여서 관리자 권한으로 실행).

```
pip install requests
python -m pip install requests
```

pip install로 설치한 패키지는 import 패키지 또는 import 패키지.모듈 형식으로 사용할 수 있습니다.

```
>>> import requests                # pip로 설치한 requests 패키지를 가져옴
>>> r = requests.get('http://www.google.co.kr')  # requests.get 함수 사용
```

모듈 만들기

모듈은 .py 파일 안에 변수, 함수, 클래스를 넣어서 만들며 스크립트 파일에서 확장자 .py를 제외하면 모듈 이름이 됩니다.

모듈.py

```
변수 = 값

def 함수이름():
    코드

class 클래스이름:
    코드
```

```
import 모듈      # 모듈을 가져옴. 스크립트 파일에서 확장자 .py를 제외하면 모듈 이름이 됨
모듈.변수        # 모듈의 변수 사용
모듈.함수()      # 모듈의 함수 사용
모듈.클래스()    # 모듈의 클래스 사용
```

모듈과 시작점

파이썬은 최초로 시작하는 스크립트 파일과 모듈의 차이가 없으며 어떤 스크립트 파일이든 시작점도 될 수 있고, 모듈도 될 수 있습니다. 스크립트 파일을 파이썬 인터프리터로 직접 실행하면 __name__에는 '__main__'이 들어가고, 스크립트 파일을 import로 가져왔을 때는 모듈의 이름이 들어갑니다.

if __name__ == '__main__':처럼 __name__의 값이 '__main__'인지 확인하는 코드는 스크립트 파일이 메인 프로그램으로 사용될 때와 모듈로 사용될 때를 구분하기 위한 용도입니다.

패키지 만들기

패키지는 폴더(디렉터리)로 구성되어 있으며 여러 개의 모듈이 들어갑니다. 폴더 안에 __init__.py 파일이 있으면 해당 폴더는 패키지로 인식됩니다.

```
__init__.py

# __init__.py 파일은 내용을 비워 둘 수 있음
```


패키지/모듈.py

```
변수 = 값

def 함수이름():
    코드

class 클래스이름:
    코드
```





한 번에 끝내는 파이썬 웹 개발

실제 서비스 구현부터 30개 이상 프로젝트까지!

파이썬
3가

from 패키지 import *로 패키지의 모든 변수, 함수, 클래스를 가져오려면 패키지의 `__init__.py` 파일에서 모듈 안의 변수, 함수, 클래스를 가져오도록 만들어야 합니다. 여기서 `.모듈의.(점)`은 현재 패키지를 뜻합니다.

```
__init__.py

from .모듈 import 변수, 함수, 클래스 # 현재 패키지의 모듈에서 각 변수, 함수, 클래스를 가져옴
from .모듈 import *                 # 또는, 모듈의 모든 변수, 함수, 클래스를 가져옴
```

이렇게 만들면 `import` 패키지와 같이 패키지만 가져온 뒤 `패키지.변수`, `패키지.함수()`, `패키지.클래스()` 형식으로도 사용할 수 있습니다.

```
import 패키지 # 패키지만 가져옴
패키지.변수   # 패키지 안에 있는 모듈의 변수 사용
패키지.함수() # 패키지 안에 있는 모듈의 함수 사용
패키지.클래스() # 패키지 안에 있는 모듈의 클래스 사용
```



내비게이션

홈

코딩 도장

내 강좌

- 파이썬 코딩 도장
 - 참여자
 - 🏆 배지
 - ▲ 역량
 - 📊 성적
 - 일반
 - Unit 1. 소프트웨어 교육과 파이썬
 - Unit 2. 파이썬 설치하기
 - Unit 3. Hello, world!로 시작하기
 - Unit 4. 기본 문법 알아보기
 - 핵심 정리
 - Unit 5. 숫자 계산하기
 - Unit 6. 변수와 입력 사용하기
 - Unit 7. 출력 방법 알아보기
 - 핵심 정리
 - Unit 8. 불과 비교, 논리 연산자 알아보기

▼

