23. 고급 문법

반복자 (Iterator)

- For 문을 통해 순차적으로 요소를 추출 할 수 있는 클래스 의 객체

- 데이터 를 순차적으로 반환 하는 반복 가능한 클래스(Iterable) 만들기
  . **Iterable** : 반복을 할 수 있도록 구성되어있는 클래스 (list, tuple, range, dict...)
  . **next()** : 클래스의 __next__ 를 호출 하도록 약속되어있는 내장 함수
  . **__next__** : 특정 규칙에 의한 데이터 반환 (반복 가능한 클래스 는 순차적 요소 반환 로직이 구현됨)

```
1  class MyList:
2      def __init__(self, *data):
3          self.data = data
4          self.index = -1
5      def __next__(self):
6          self.index += 1
7          if self.index >= len(self.data):
8              self.index = -1
9              raise StopIteration
10         return self.data[self.index]
11
12 lists = MyList(10,20,30,40,50,60)
13 while True:
14     try:
15         result = next(lists)
16     except StopIteration :
17         break
18     print(result)
```

>12행 : MyList 클래스 에 가변인수 n 개의 데이터 전달 후 객체를 생성
>2행 : 가변 인자 를 통해 n 개의 데이터 를 전달 받음
>4행 : 57행에서 요소 추출 시 index 를 +1 하고 시작하므로 index 를 0 부터 시작하게 하기 위하여 -1 로 초기화
>15행 : __next__ 를 호출
>6행 : index 1 증가
>7행 : index 가 모두 처리 되었는지 비교
>8행 : index -1 로 초기화 (다음 호출을 위하여 초기화)
>9행 : StopIteration 예외 발생
  * __next__ 를 통한 요소 추출 을 위하여 요소가 없음을 알리는 예외

> 실행 결과

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

10
20
30
40
50
60
```

>While 문을 통하여 열거된 데이터를 하나씩 추출하지만 Next() 함수를 통해서만 구현 가능하므로 완전한 Iterable 클래스라고 할 수 없다.

- For 문
  . 위의 While 문을 간단히 구현 하기 위하여 생성된 문법

```
1  class MyList:
2      def __init__(self, *data):
3          self.data = data
4          self.index = -1
5      def __next__(self):
6          self.index += 1
7          if self.index >= len(self.data):
8              raise StopIteration
9          return self.data[self.index]
10
11
12 lists = MyList(10,20,30,40,50,60)
13 for k in lists:
14     print(k)
15
```

> 실행 결과

```
15  for k in lists:

Exception has occurred: TypeError ×
'MyList' object is not iterable
  File "C:\Python\Source\Remove.py", line 15, in <module>
    for k in lists:
TypeError: 'MyList' object is not iterable
```

. Iterable 형식이 아닌 클래스 를 For 문으로 구동 할 수 없음을 알리는 오류 (클래스 에서 __Iter__ 메서드 를 찾지 못하여 발생)
> For 문은 For 문이 수행 할 수 있는 반복 가능 클래스 임을 선언해 주는 __Iter__ 메서드 를 찾기로 되어있다.
> 따라서 For 문을 통해 반복 클래스를 구현 하기 위하여 __Iter__ 을 선언해 주어야만 한다.

- Iterator (반복자)
. **Iterator** : __iter__ 가 반복 가능한 형태로 만든 상태 의 클래스
. **iter()** : 클래스의 __iter__ 를 호출 (For 문법에서 확인을 위해 호출함)
. **__iter__** : 현재 클래스 를 순차적으로 다음 요소를 확인 할 수 있는 클래스(반복자) 임을 선언
> 클래스 에 __iter__ 가 없으면 다음 요소를 확인할 수 있는 클래스 가 아님을 간주

```python
class MyList:
    def __init__(self, *data):
        self.data = data
        self.index = -1
    def __iter__(self):
        return self
    def __next__(self):
        self.index += 1
        if self.index >= len(self.data):
            raise StopIteration
        return self.data[self.index]


lists = MyList(10,20,30,40,50,60)
it = iter(lists)
while True:
    try:
        result = next(it)
    except StopIteration :
        break
    print(result)
```

>15행 : lists 객체 가 반복자 가 될수있는 클래스인지 확인 후 반복자 객체 it 를 생성

> 5행 : iter() 호출 시 __iter__ 메서드 가 있으므로 반복자가 될수 있는 클래스임을 확인

> 실행 결과

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

10
20
30
40
50
60
```

- For 문법으로 적용

```
class MyList:
    def __init__(self, *data):
        self.data = data
        self.index = -1
    def __iter__(self):
        return self
    def __next__(self):
        self.index += 1
        if self.index >= len(self.data):
            raise StopIteration
        return self.data[self.index]


lists = MyList(10,20,30,40,50,60)
for k in lists:
    print(k)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

10
20
30
40
50
60
```

> 정상 동작 되는것을 확인 할 수 있다.

- 요소의 끝을 알리는 예외 상황을 다른것으로 발생한다면 ?
  . For 문에서 받기로 약속된(정의된) 예외 종류가 아니므로 오류가 발생한다.
    * 값은 모두 출력 된상태에서 오류가 발생함 (인터프리터)

```
Remove.py > MyList > __next__
 1  class MyList:
 2      def __init__(self, *data):
 3          self.data = data
 4          self.index = -1
 5      def __iter__(self):
 6          return self
 7      def __next__(self):
 8          self.index += 1
 9          if self.index >= len(self.data):
10              raise ValueError

Exception has occurred: ValueError ×
exception: no description
  File "C:\Python\Source\Remove.py", line 10, in __next__
    raise ValueError
  File "C:\Python\Source\Remove.py", line 14, in <module>
    for k in lists:
ValueError:
```

* MyList 클래스 는 __iter__ **메서드와** __next__ 가 존재하며 열거의 끝을 알리는 ValueError 클래스를 예외를 가지는 Iterable 한 Iterator(반복자) 클래스 라고 할수 있다.

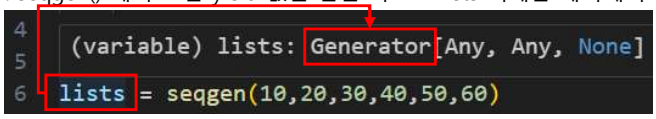
제너레이터(함수 반복자)

- 반복자를 클래스 형식이 아닌 함수 를 이용하여 간편히 구현하는 기능
- 일반적인 함수의 형태로 구현하며 yield 명령으로 값을 리턴한다.
  . yield : return 과 비슷한 기능, 변수의 마지막 값과 상태를 저장하고있다.
- 제너레이터는 내부에서 __iter__() , __next__ 메서드 를 자동으로 생성한다.
  . 제너레이터 의 예제

```
1  def seqgen(*data):
2      for index in range(len(data)):
3          yield data[index]
4
5  # 메서드 를 실행하는 대리자 lists
6  lists = seqgen(10,20,30,40,50,60)
7  for k in lists:
8      print(k)
```

> 실행 결과

> 6행 : seqgen() 메서드 는 yield 값을 반환 하므로 lists 객체는 제너레이터 가 된다.



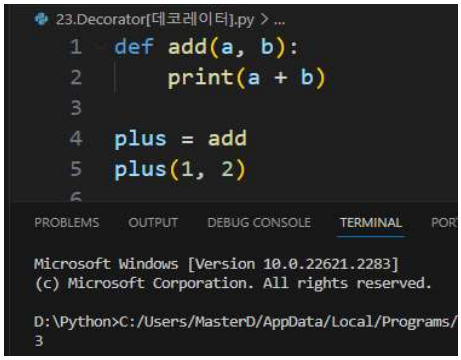
> 7행 : Sequen 제너레이터 를 호출한다.
> 3행 : yield 로 인해 마지막 저장한 index 를 기억후 순차적으로 데이터를 반환한다
> 7행 : 3행에서 반환한 결과를 lists 를 Iterater 객체로 등록
> 8행 : lists 객체 에서 순차적으로 데이터를 추출 하여 표현
> 더 이상 반환할 값이 없으면 자동으로 StopIteration 예외를 발생시켜 빠져나온다.

* 여기서 우리가 주목해야할 반복자와 제너레이터 의 핵심 기능은 는 현재 시점에 반환해야 할 값을 매번 순차적으로 하나씩 반환한다는 것이다.
그렇지 않고 만약 모든 데이터 를 미리 재단을 해 두고 하나씩 반환을 하는 로직을 구현하면 중간에 반복을 종료 해야 할 경우
데이터를 모두 가공해 서 처리하는 방식과 시점별로 재단하여 반환하는 방식 의 성능의 차이는 확연히 다를것이다.

함수 의 활용

- 파이썬과 같은 함수형 언어는 함수 도 변수와 똑같은 방식으로 다루어 지며 아래의 동작이 가능하다.
  . 다른 변수 에 대입할 수 있다.
  . 인수 로 전달 할 수 있다.
  . 함수를 리턴 할 수 있다.
  . 컬렉션에 저장 할 수 있다.

* 함수의 정의 와 대입 의 예제

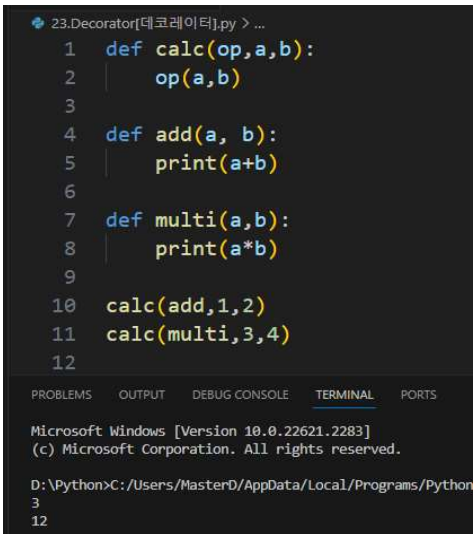


>1행 : 함수의 이름정의

>4행 : 함수를 변수에 대입
* 대입 할때는 () 를 붙이지 않는다.

* 두 함수 plus 와 add 는 같은 객체 이며
plus 는 add 의 별명이다.

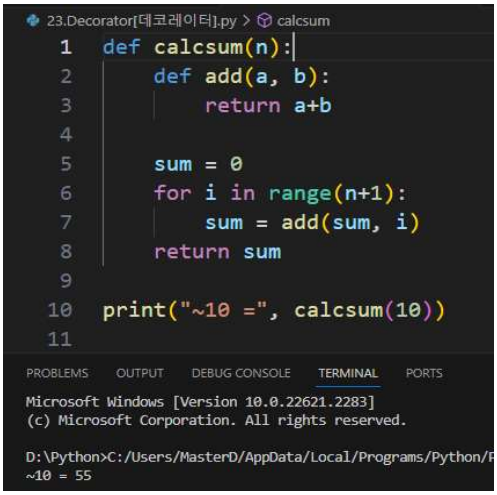
* 함수를 인수 로 전달 예제



>10행 : add 함수 와 1, 2 값을 인수로 전달한다.
>1행 : op 변수에 add 함수를 담고 1, 2 를 각각 a, b 변수 에 담는다.
>2행 : add 함수에 1,2 인자를 전달 하고 add 함수를 실행한다.

* 인수로 전달받은 op 가 어떤 함수를 전달 받느냐에 따라 clac 함수의 기능이 달라진다.
* 함수를 전달 하게 될 경우 프로그램이 더욱 활용성이 높아지며 재사용성 이 증가한다.

- 지역 함수
  . 길고 복잡한 동작을 수행하는 함수 가 있을때 그중 일부 동작을 지역함수로 정의 한다.
  . 함수 내부의 반복되는 코드를 통합할 수 있어소스가 짧아지면서 관리하기 쉬운 이점이 있다.
  . 외부에서 호출 할 수 없다.



>10행 : calcsum 함수에 10 을 전달 후 호출한다.
>1행 : calcsum 함수 가 10을 전달 받는다.
> 6행 : 0 ~ 10 까지 반복한다.
> 7행 : (sum, I) 를 인자로 내부 함수 add 를 호출한다
>3행 : 인자로 전달 받은 두 수를 합하여 반환한다.
>7행 : sum 변수 에 대입한다.
* sum 변수가 매번 갱신되는걱 같아 보이지만
변경 되기 전의 sum 값과 i 의 값을 더하는 add 함수 를 실행하므로
누적 된 결과 값이 할당된다.
>10행 : 결과를 출력한다.

. 지역함수의 활용 (함수의 리턴)

```python
def makeHello(message):
    def hello(name):
        print(message + ", " + name)
    return hello

enghello = makeHello("Good Morning")

enghello("Mr kim")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/pyt
Good Morning, Mr kim
```

>6행 : makeHello 함수 는 인사말 문자열을 전달 받아 인사말과 이름을 같이 출력하는 함수 hello 를 생성하여 리턴한다.
* 자기 자신 의 지역 함수를 리턴
따라서 enghello 는 makeHello -> hello() 함수가 할당된다

>8행 : 할당 된 지역함수 enghello 에 "Mr Kim" 을 인수로 전달후 호출시 2행 의 hello 메서드가 실행되며 "Good Morning, Mr kim" 을 반환하게 된다.

* makeHello 함수의 message 인자는 4행 을 만난후 소멸되어야 정상이지만 이경우 는 지역함수 를 반환하는 특징 상 message 인자 의 값이 소멸 되면 안된다고 판단하여 클로저(Closure) 라는 특수한 구조 를 만들어 유지하게 된다.

함수 데코레이터 ★★★

- 함수에 장식을 붙이듯 앞 뒤로 원하는 코드를 추가하는 기법
- 함수를 실행하기 전에 특정 로직을 수행 하거나 검증 하는등의 일련의 일을 정의 해 두어 간단하게 사용할 수 있도록 만들어 놓은 문법
. 함수 를 래핑(Wrapping) 하여 원하는 코드 를 추가 하고 내부에서 원래 함수를 대리호출하여 기능을 확장하는 함수 로직

```python
def inner():
    print("결과를 출력합니다.")

def outer(func):
    print("-" * 20)
    func()
    print("-" * 20)

outer(inner)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/py
--------------------
결과를 출력합니다.
--------------------
```

>9행 : inner 함수를 감싸는 래퍼 outer 함수를 호출한다.
>6행 : outer 함수 내부에 func 인자로 받은 inner() 함수를 실행한다.

* outer 함수는 앞 뒤에 줄을 긋는 추가 동작을 수행한다.
앞 뒤로 코드를 추가 할 수 있을뿐 inner 함수 자체의 기능은 변경 하지 못한다.
outer 는 인수로 받은 함수 를 대리 호출 할 뿐이다.

* inner 함수 가 주 로직이지만 oputer함수가 인자로 받고 있어 선뜻 주체 가 어떤 로직인지 모호해질 수 있다.

. 주요 로직을 구현하는 직관적인 함수 로직으로 수정

```python
def inner():
    print("결과를 출력합니다.")

def outer(func):
    def wrapper():
        print("-" * 20)
        func()
        print("-" * 20)
    return wrapper

inner = outer(inner)
inner()
```

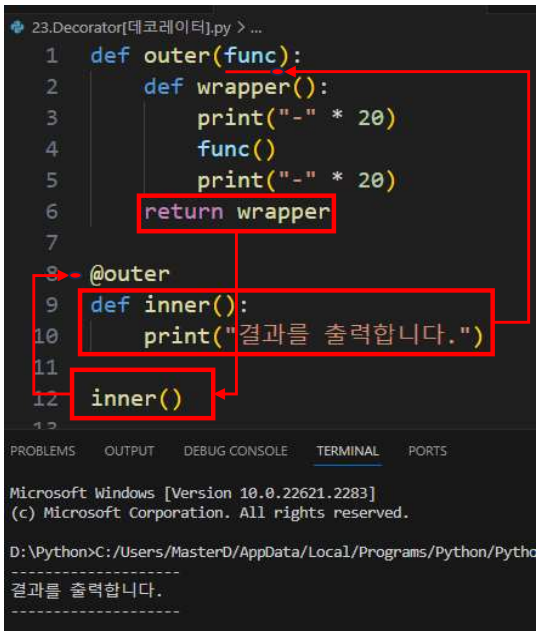
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Py
--------------------
결과를 출력합니다.
--------------------
```

>11행 : outer 함수에 inner 함수를 인자로 전달한다.
>9행 : func 함수는 클로저 가 기억을 하고 지역함수 wraper() 를 반환한다.
>11행 : inner 함수 가 신규 기능이 적용된 wraper 함수 를 새로 할당 받는다.
* 기존의 함수 기능을 wrapper() 로 덮어쓰기 한다.
>12행 : 전달받은 wrapper() 메서드 를 실행한다.
* 이때 outer 가 전달 받은 func 인자의 인수 inner() 는 클로저가 기억 하고 있으므로 wrapper() 의 func() 는 변경되기 이전의 inner() 를 기억하고 실행한다.
* inner2 라는 변수로 받아 실행해도 된다.

```python
inner2 = outer(inner)
inner2()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/
--------------------
결과를 출력합니다.
--------------------
```

. 래핑을 하는 로직을 데코레이터 로 표현
> 위 래핑 로직은 굉장히 실용적이다 따라서 간단히 표현 할 수 있도록 데코레이터 로 기능을 만들어 두었다.

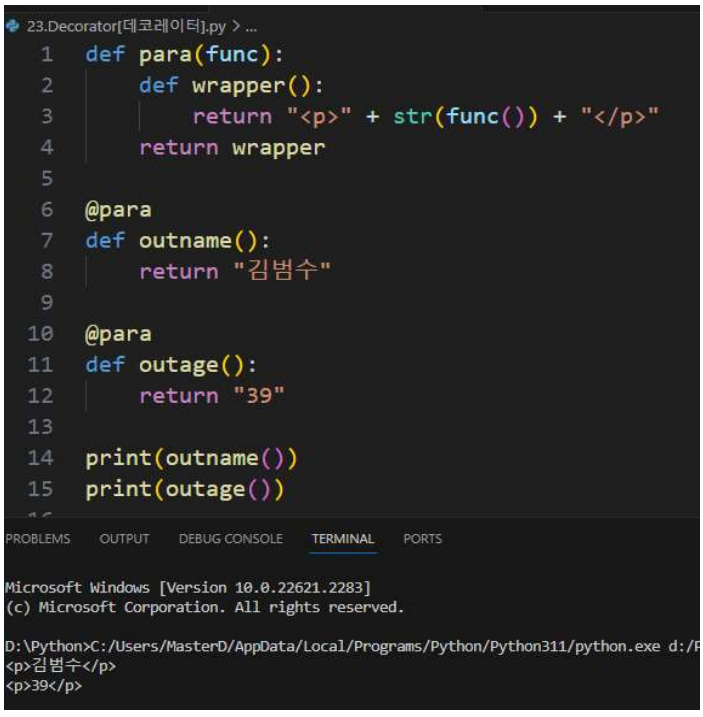


>12행 : @outer 호출
@outer : def inner() 메서드를 outer 메서드의 인자로 전달
>6행 : inner() 메서드를 래핑한 wrapper 메서드를 반환
>12행 : wrapper 를 inner() 라는 이름으로 실행

>8행 : @outer
* outer 함수 로 전달 하여 wrapper() 메서드를 다시 받는것을 정의해둠

데코레이터의 활용

- 웹페이지의 태그를 자동으로 생성해 주는 데코레이터



>6행 : <p>"김범수"</P> 를 반환하도록 함수를 꾸밈

>10행 : <p>"39"</P> 를 반환하도록 함수를 꾸밈

- 웹페이지의 태그를 자동으로 생성해 주는 데코레이터 2

```python
def div(func):
    def wrapper():
        return "<div>" + str(func()) + "</div>"
    return wrapper

def para(func):
    def wrapper():
        return "<p>" + str(func()) + "</p>"
    return wrapper

@div
@para
def outname():
    return "김범수"

@para
@div
def outage():
    return "39"

print(outname())
print(outage())
```

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe d:/Python
<div><p>김범수</p></div>
<p><div>39</div></p>
```

> 11행 : para 의 결과를 받아서 div 로 감싸는 함수 wrapper 를 outname 에 할당

> 16행 : div 의 결과를 받아서 para로 감싸는 함수 wrapper 를 outage 에 할당

* 실제 구현하는 함수 자체의 로직은 변경하지 않고 꾸미기만 해준다.

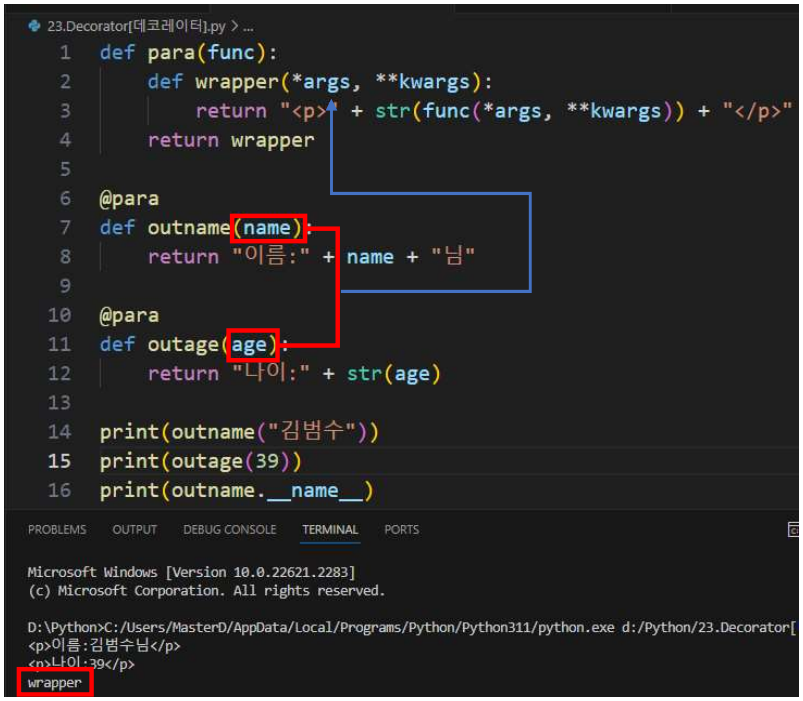
- 인수를 가지는 함수 를 래핑하는 데코레이터 예제 (오류의 확인)

```python
def para(func):
    def wrapper():
        return "<p>" + str(func()) + "</p>"
    return wrapper

@para
def outname(name):
    return "이름:" + name + "님"

@para
def outage(age):
    return "나이:" + str(age)

print(outname("김범수"))
print(outage(39))
```

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe d:/Py
Traceback (most recent call last):
  File "d:\Python\23.Decorator[데코레이터].py", line 14, in <module>
    print(outname("김범수"))
          ^^^^^^^^^^^^^^
TypeError: para.<locals>.wrapper() takes 0 positional arguments but 1 was given
```

* outname 과 outage 는 인자를 가지지만 3행의 func() 는 인자를 받지 않는 함수 호출 로 구현되어있어 오류 가 발생한다.

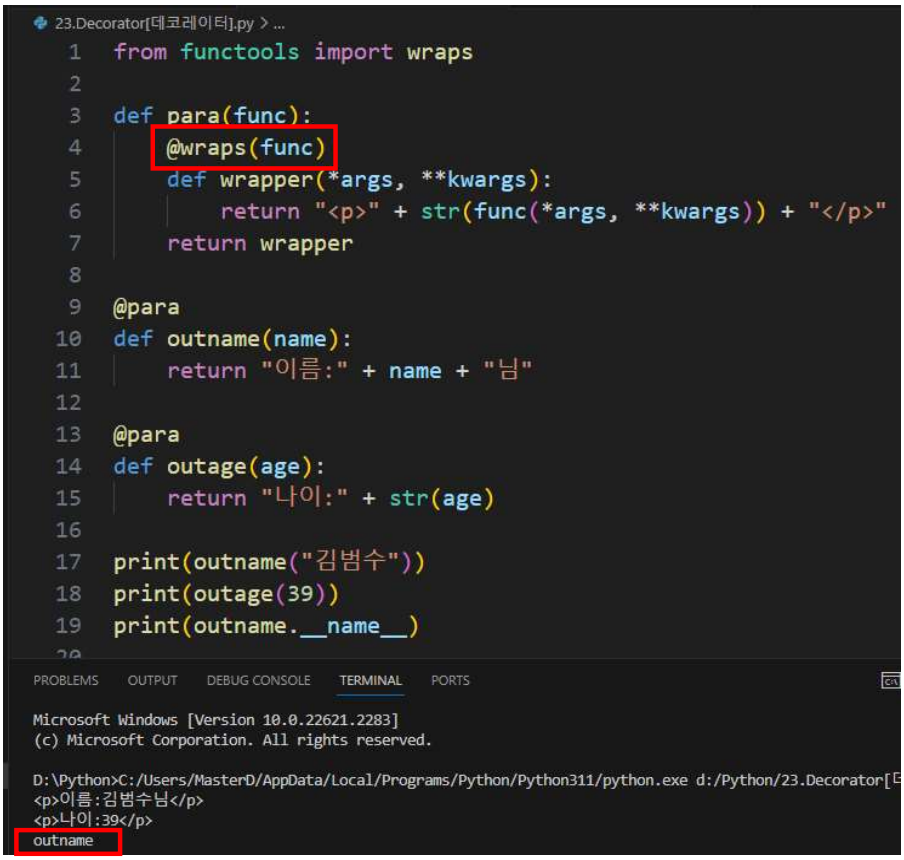
- 인수를 가지는 함수 를 래핑하는 데코레이터 예제 (오류 의 수정)



>2행 : 가변인수 *args 와
선택적 가변인수 **kwargs 를 설정하여
인수의 개수와 데이터 유형에 관계없는
인자 를 가진 함수로 변경후 실행

>16행 : outname.__name__ 는 실제 실행하는 함수의
이름을 표현한다.
주체 메서드 가 명확해 지지 못하여 프로그램
을 구현할 때 혼돈이 생길 수 있다.

- 원본 함수의 속성을 그대로 유지 하면서 기능만 갱신되는 데코레이터



>4행 :래핑하여 결과를 반환하여 줄 함수 wrapper()
에 @warps() 데코레이터를 추가 하였다.
래핑 하여 반환 받기 전 함수의 속성 을 유지 한체
내용 만 추가 되어 반환 된다.

>19행 : outname.__name__ 의 결과 가 outname 인
것을 확인 할 수 있다.

* 모듈 내에서 함수를 호출 하므로 함수간의 의존도가 높다 (유지보수성이 떨어진다)

클래스 데코레이터

- 클래스 의 객체 를 호출 전, 호출후 수행하는 로직을 작성해 두는 기능

. 클래스 래핑의 예제

23.Decorator[데코레이터].py > ...

```
1  class Outer:
2      def __init__(self, func):
3          self.func = func
4
5      def __call__(self):
6          print("-" * 20)
7          self.func()
8          print("-" * 20)
9
10 def inner():
11     print("결과를 출력합니다.")
12
13 inner = Outer(inner)
14 inner()
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/
--------------------
결과를 출력합니다.
--------------------
```

> 13행 : 생성자 인자 에 inner 함수를 전달 후 인스턴스화 하여
inner 이름으로 할당한다.

>14행 : inner() 호출

>5행 : __call__ 메서드가 실행 되며 클래스 인스턴스 변수
func() 메서드(inner()) 를 꾸며주는 print() 내용과 함께 결과를 출력한다.

* 래핑하는 로직을 class 로 만들어 외부에서 호출하여 사용하게 함으로서
재사용성과 보안성, 유지 보수성을 높일 수 있다.

. 클래스 래핑을 위한 데코레이터 의 활용

23.Decorator[데코레이터].py > ...

```
1  class Outer:
2      def __init__(self, func):
3          self.func = func
4
5      def __call__(self):
6          print("-" * 20)
7          self.func()
8          print("-" * 20)
9
10 @Outer
11 def inner():
12     print("결과를 출력합니다.")
13
14 inner()
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python> cmd /C "C:\Users\MasterD\AppData\Local\Programs\Pytho
s-python.python-2023.16.0\pythonFiles\lib\python\debugpy\adapter
레이터].py "
--------------------
결과를 출력합니다.
--------------------
```

>10행 : Outer 클래스를 인스턴스화 하면서 inner() 함수를
Outer 인스턴스 변수 로 등록함

>14행 : inner() 호출 시 __call__ 함수 를 실행함

동적 코드 실행

- eval()

. 프로그램 실행 중 생성되는 결과 로 실시간 연산 처리 하거나 형변환 하는 함수

```python
result = eval("2 + 3 * 4")
print(result)

a = 2
print(eval("a + 3"))

city = eval("['seoul', 'osan', 'suwon']")
for c in city:
    print(c, end = ', ')
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe "d:
14
5
seoul, osan, suwon,
```

>1행: 2+ 3 * 4 의 문자열을 수식으로 바꾸어 출력한다.
  * 문자열의 조합 을 수식으로 바꾸어 준다.

>5행 : a+3 의 연산을 처리한다. a=2 의 값이 할당되어 있으나
  실제로 a 가 어떤 값을 가지느냐에 따라 결과가 달라진다.

>7행 : 문자열 로 표현된 리스트 를 실제 리스트 형식으로 변환한다.
>8행 : 변형 된 리스트 로 for 문을 이용하여 데이터 추출 후 표현

. eval 함수 를 이용한 예제

> 문자열로 입력 받은 내용을 동적으로 처리하여 결과를 반환하는 로직

```python
import math

while True:
    try:
        expr = input("수식을 입력하세요(끝낼 때 0) : ")
        if expr == '0':
            break
        print(eval(expr))
    except:
        print("수식이 잘못되었습니다.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

D:\Python>C:/Users/MasterD/AppData/Local/Programs/Python/Python311/python.exe "d:/Python/24.,Dy
수식을 입력하세요(끝낼 때 0) : 8**
수식이 잘못되었습니다.
수식을 입력하세요(끝낼 때 0) : 1+2
3
수식을 입력하세요(끝낼 때 0) : 1+2*4+6+2
17
수식을 입력하세요(끝낼 때 0) : 일 더하기 삼
수식이 잘못되었습니다.
수식을 입력하세요(끝낼 때 0) : ['123','22242ㄹ']
['123', '22242ㄹ']
수식을 입력하세요(끝낼 때 0) : math.sqrt(2)
1.4142135623730951
수식을 입력하세요(끝낼 때 0) :
```

>5행 : 사용자 로 부터 수식을 입력 받는다.

>8행 : 수식을 받아 연산 결과를 출력한다.
>9행 : 연산 할 수 없는 내용 입력 시 예외 처리를 한다.