Functional Programming

함수 복습

In :
```
def greet(name, msg):
    print('안녕', name, msg)
```

In :
```
greet('길동', '좋은 아침!')
```
Out:
```
안녕 길동 좋은 아침!
```

In :
```
def greet(name, msg='별일 없죠?'):
    print('안녕', name, msg)
```

In :
```
greet('길산')
```
Out:
```
안녕 길산 별일 없죠?
```

In :
```
greet('길산', '좋은 아침~')
```
Out:
```
안녕 길산 좋은 아침~
```

함수 복습

In :
```
def calc(x, y, z):
    return x+y+z
```

In :
```
print(calc(1, 2, 3))
```
Out:
```
6
```

In :
```
print(calc(x=1, y=2, z=3))
```
Out:
```
6
```

In :
```
print(calc(y=2, z=3, x=1))
```
Out:
```
6
```

함수 복습

In :	**print(calc(1, y=2, z=3))**
Out:	6

In :	**print(calc(x=1, y=2, 3))**
Out:	**SyntaxError:** positional argument follows keyword argument

함수 복습

In :
```
def sum_all(*args):
    result = 0
    for i in args:
        result += i
    return result
```

In :
```
print(sum_all(1,2,3,4))
```

Out:
```
10
```

함수 복습

In :

```
def sum_all(*args):
    result = 0
    for i in args:
        result += i
    return result
```

In :

```
lst = [1, 2, 3]
print(sum_all(*lst))
```

Out:

```
10
```

함수 복습

In :

```
def func(dx, dy):
    '''함수의 도움말'''
    dx, dy = dy, dx
    return dx, dy
```

In :

```
print(func)
```

Out:

```
<function func at 0x00000281C4E97268>
```

In :

```
print(type(func))
```

Out:

```
<class 'function'>
```

함수 복습

In :	**print(func.__doc__)**
Out:	함수의 도움말

In :

```
help(func)
```

Out:

```
Help on function func in module __main__:

func(dx, dy)
    함수의 도움말
```

In :	**print(func(10, 20))**
Out:	(20, 10)

Procedural programming

In :
```
lst = [1, 2, 3]
```

In :
```
def sum_list(lst):
    result = 0
    for value in lst:
        result += value
    return result
```

In :
```
print(sum_list(lst))
```

Out:
```
6
```

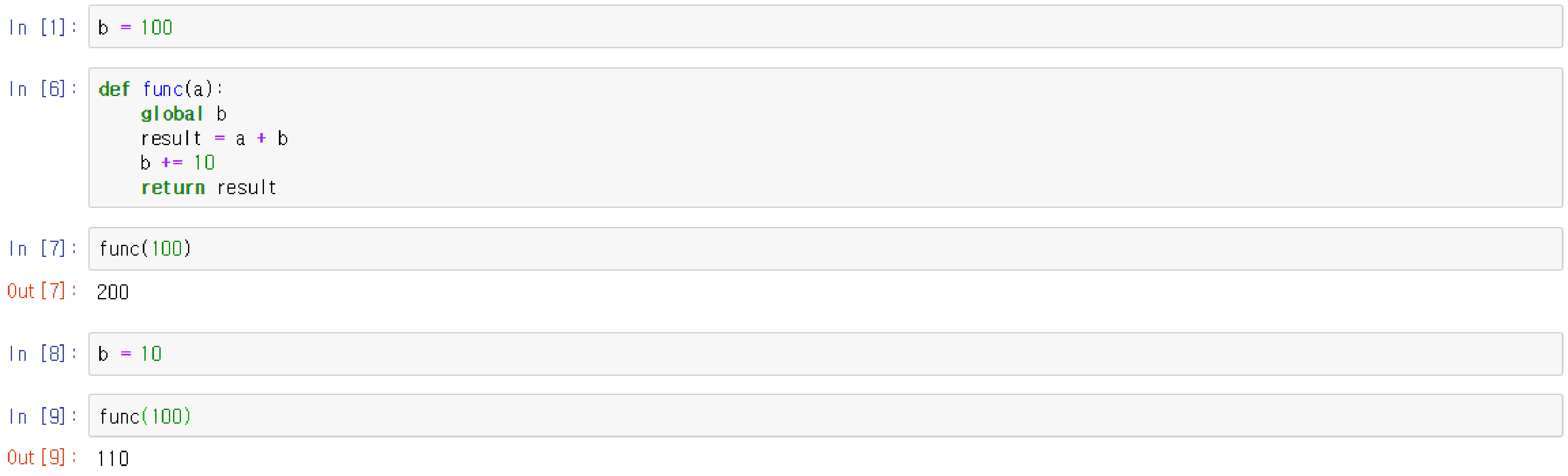
Functional programming

- In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.   -Wikipidia

Functional programming

- 함수형 프로그래밍 특징
  - 상태 표현 피하기
  - 데이터에 대한 변경 불가능
  - first class
  - high order function
  - recursive call

Functional programming



```
In [1]: b = 100

In [6]: def func(a):
            global b
            result = a + b
            b += 10
            return result

In [7]: func(100)

Out[7]: 200

In [8]: b = 10

In [9]: func(100)

Out[9]: 110
```

Functional programming

```
In :    new_lst = [1, 2, 3, 4]
```

```
In :    def sum_list(lst):
            if len(lst) == 1:
                return lst[0]
            else:
                return lst[0] + sum_list(lst[1:])
```

```
In :    print(sum_list(new_lst))
Out:    10
```

Functional programming

- Python's functional feature
  - lambda
  - map / filter / reduce
  - High order function
  - iterator
  - generator
  - closure
  - decorator

람다 함수(lambda function)

In :

```
def add(x, y):
    return x+y

print(add(10, 10))
```

Out:

```
20
```

In :

```
a = lambda x,y : x+y
print(a(10, 10))
```

Out:

```
20
```

함수는 객체

In :
```
def wow(text):
    return text.upper()
```

In :
```
wow('hi')
```

Out:
```
'HI'
```

In :
```
oh = wow
```

함수는 객체

```
In :  oh('hello')
Out:  'HELLO'
```

```
In :  del wow
```

```
In :  oh('hello~')
Out:  'HELLO~'
```

함수는 객체

In :	**wow('hello?')**
Out:	NameError: name 'wow' is not defined

In :	**oh.__name__**
Out:	'wow'

함수는 객체

In :
```
flst = [oh, str.lower, str.capitalize]
```

In :
```
flst
```

Out:
```
[<function __main__.wow(text)>, <method 'lower' of 'str' objects>,
<method 'capitalize' of 'str' objects>]
```

In :
```
for x in flst:
    print(x('welcome'))
```

Out:
```
WELCOME
welcome
Welcome
```

함수는 객체

In :	**flst[0]('welcome~')**
Out:	'WELCOME~'

함수는 객체

In :
```
def greet(func):
    greeting = func('hi, I love python')
    print(greeting)
```

In :
```
greet(oh)
```

Out:
```
HI, I LOVE PYTHON
```

map 함수

In :	**list(map(oh, ['hi', 'hello', 'welcome']))**
Out:	['HI', 'HELLO', 'WELCOME']

map 함수

In :	**list(map(oh, ['hi', 'hello', 'welcome']))**
Out:	['HI', 'HELLO', 'WELCOME']

reduce( )

```
In :  from functools import reduce
      reduce(lambda x, y: x + y, [0,1,2,3,4])
Out:  10
```

```
In :  reduce(lambda x, y: y+x, 'abcd')
Out:  'dcba'
```

filter( )

```
In :   list(filter(lambda x: x<5, range(10)))
Out:   [0, 1, 2, 3, 4]
```

```
In :   list(filter(lambda x: x>5, range(10)))
Out:   [6, 7, 8, 9]
```

내부 함수

In :
```
def speak (text):
    def wow(t):
        return t.lower()
    return wow(text)
```

In :
```
speak('Hello, World')
```
Out:
```
'hello, world'
```

In :
```
wow('Hi')
```
Out:
```
NameError: name 'wow' is not defined
```

In :
```
speak.wow
```
Out:
```
AttributeError: 'function' object has no attribute 'wow'
```

내부 함수

In :

```
def get_speak_wow(volume):
    def wow(text):
        return text.lower()
    def oh(text):
        return text.upper()
    if volume > 0.5:
        return oh
    else:
        return wow
```

In :

```
speak_func = get_speak_wow(0.8)
speak_func('Hello')
```

Out:

```
'HELLO'
```

클로저(closures)

In :

```
def get_speak_func(text, volume):
    def wow():
        return text.lower()
    def oh():
        return text.upper()
    if volume > 0.5:
        return oh
    else:
        return wow
```

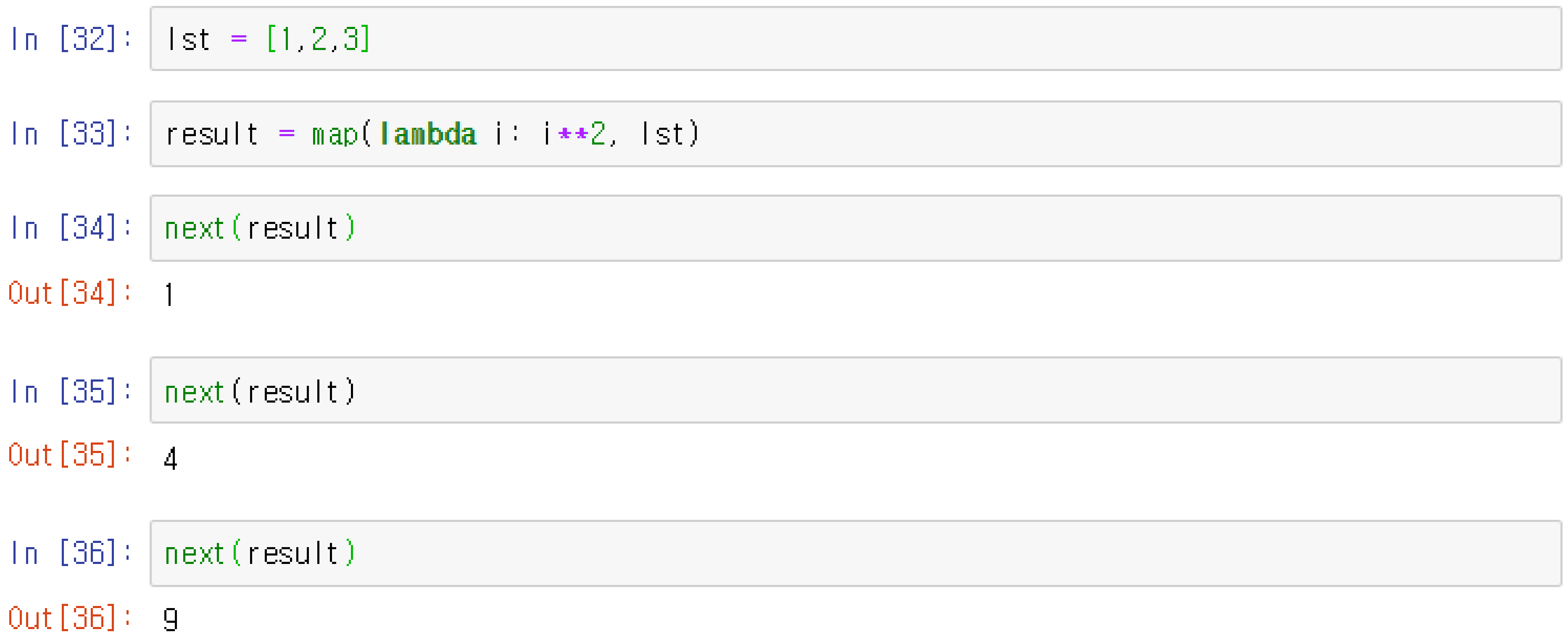
In :

```
get_speak_func('Hello, World', 0.7)()
```

Out:

```
'HELLO, WORLD'
```

map 함수 복습



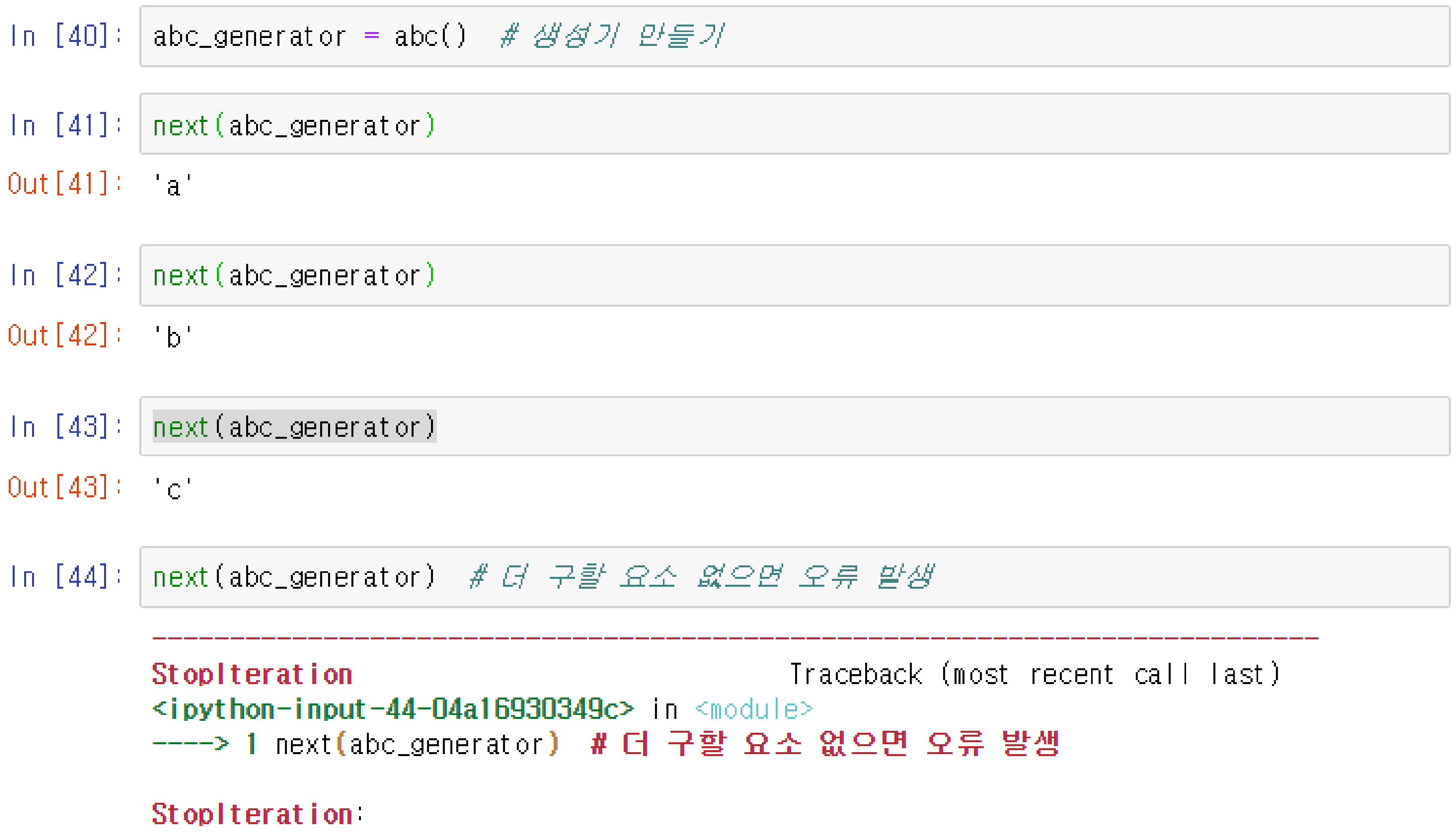
```
In [32]: lst = [1,2,3]

In [33]: result = map(lambda i: i**2, lst)

In [34]: next(result)

Out[34]: 1

In [35]: next(result)

Out[35]: 4

In [36]: next(result)

Out[36]: 9
```

generator

```
In [37]: def abc():
             '''a, b, c를 출력하는 생성기'''
             yield 'a'
             yield 'b'
             yield 'c'
```

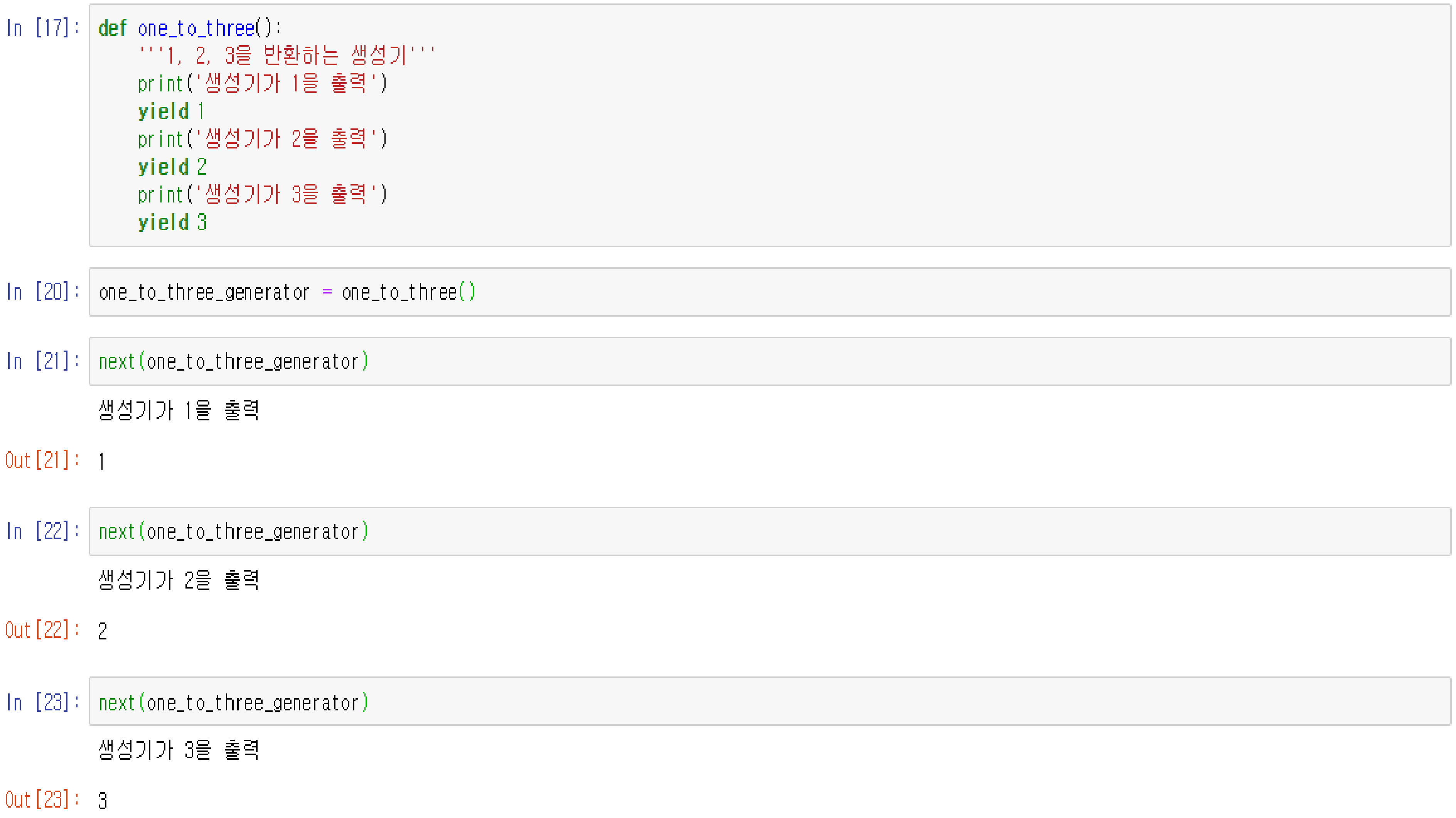
```
In [38]: abc()    #생성기 만들기

Out[38]: <generator object abc at 0x0000023C261BF660>
```

generator



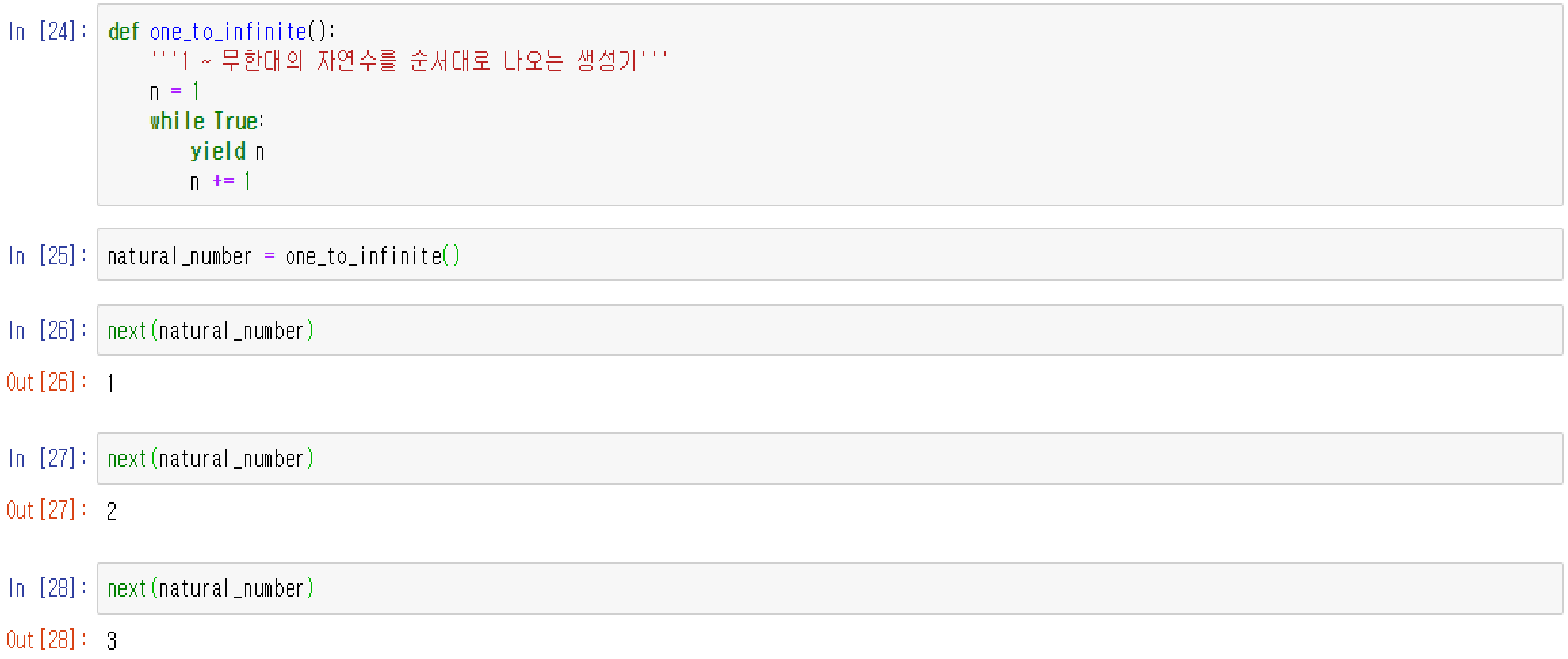
```
In [40]: abc_generator = abc()  # 생성기 만들기

In [41]: next(abc_generator)
Out[41]: 'a'

In [42]: next(abc_generator)
Out[42]: 'b'

In [43]: next(abc_generator)
Out[43]: 'c'

In [44]: next(abc_generator)  # 더 구할 요소 없으면 오류 발생

         ---------------------------------------------------------------------------
         StopIteration                             Traceback (most recent call last)
         <ipython-input-44-04a169303490c> in <module>
         ----> 1 next(abc_generator)  # 더 구할 요소 없으면 오류 발생

         StopIteration:
```

yield



```
In [17]: def one_to_three():
             '''1, 2, 3을 반환하는 생성기'''
             print('생성기가 1을 출력')
             yield 1
             print('생성기가 2을 출력')
             yield 2
             print('생성기가 3을 출력')
             yield 3

In [20]: one_to_three_generator = one_to_three()

In [21]: next(one_to_three_generator)

         생성기가 1을 출력

Out[21]: 1

In [22]: next(one_to_three_generator)

         생성기가 2을 출력

Out[22]: 2

In [23]: next(one_to_three_generator)

         생성기가 3을 출력

Out[23]: 3
```

generator



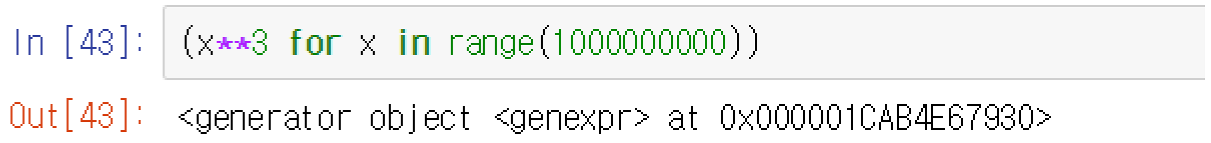
```
In [24]: def one_to_infinite():
             '''1 ~ 무한대의 자연수를 순서대로 나오는 생성기'''
             n = 1
             while True:
                 yield n
                 n += 1

In [25]: natural_number = one_to_infinite()

In [26]: next(natural_number)

Out[26]: 1

In [27]: next(natural_number)

Out[27]: 2

In [28]: next(natural_number)

Out[28]: 3
```

generator expression

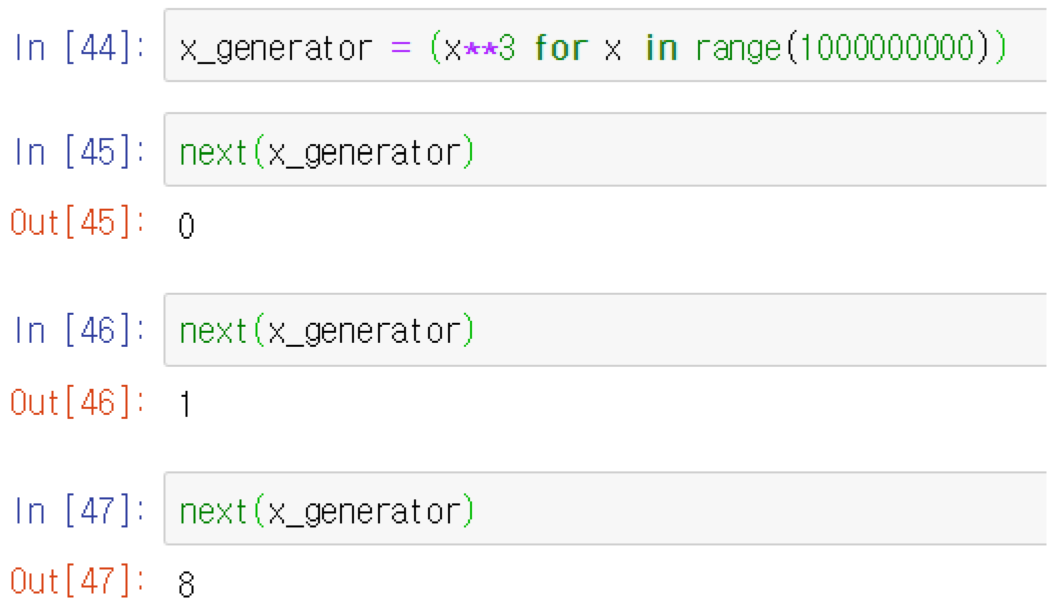
```
In [41]: [x**3 for x in range(10)]

Out[41]: [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

```
In [42]: [x**3 for x in range(1000000000)]

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-42-5a0850146680> in <module>
----> 1 [x**3 for x in range(1000000000)]

<ipython-input-42-5a0850146680> in <listcomp>(.0)
----> 1 [x**3 for x in range(1000000000)]

KeyboardInterrupt:
```

generator expression

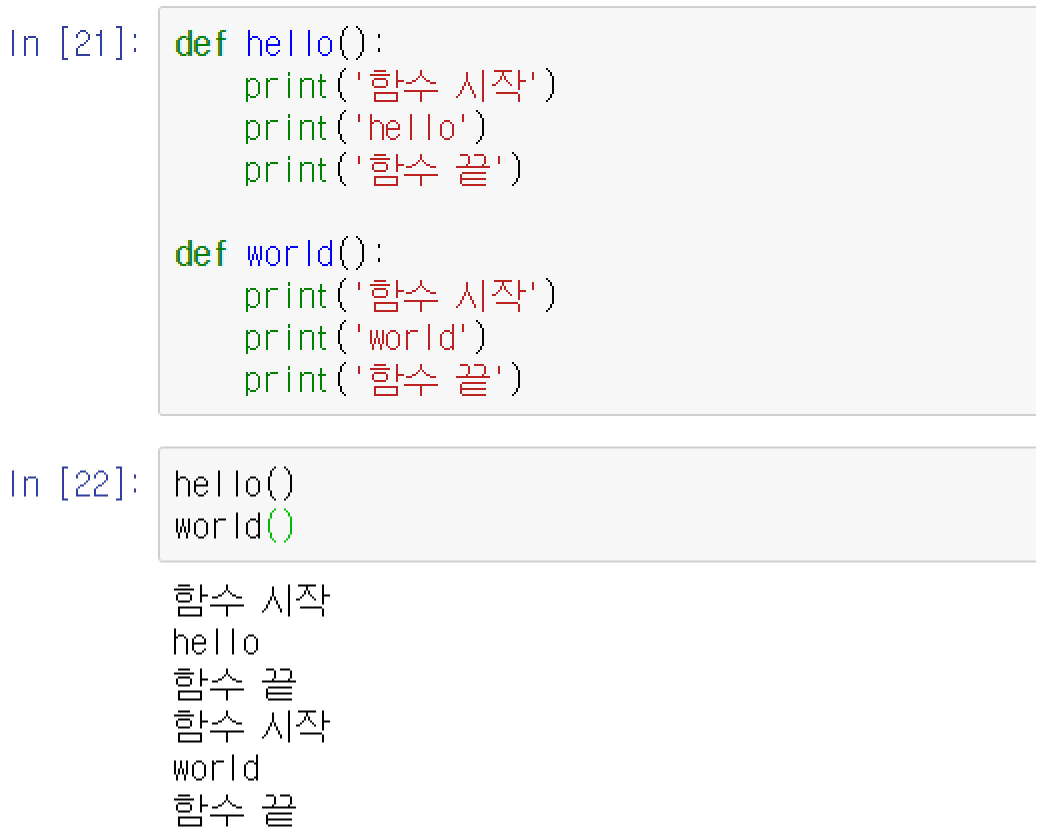


```
In [43]: (x**3 for x in range(1000000000))
Out[43]: <generator object <genexpr> at 0x000001CAB4E67930>
```



```
In [44]: x_generator = (x**3 for x in range(1000000000))

In [45]: next(x_generator)
Out[45]: 0

In [46]: next(x_generator)
Out[46]: 1

In [47]: next(x_generator)
Out[47]: 8
```

decorator

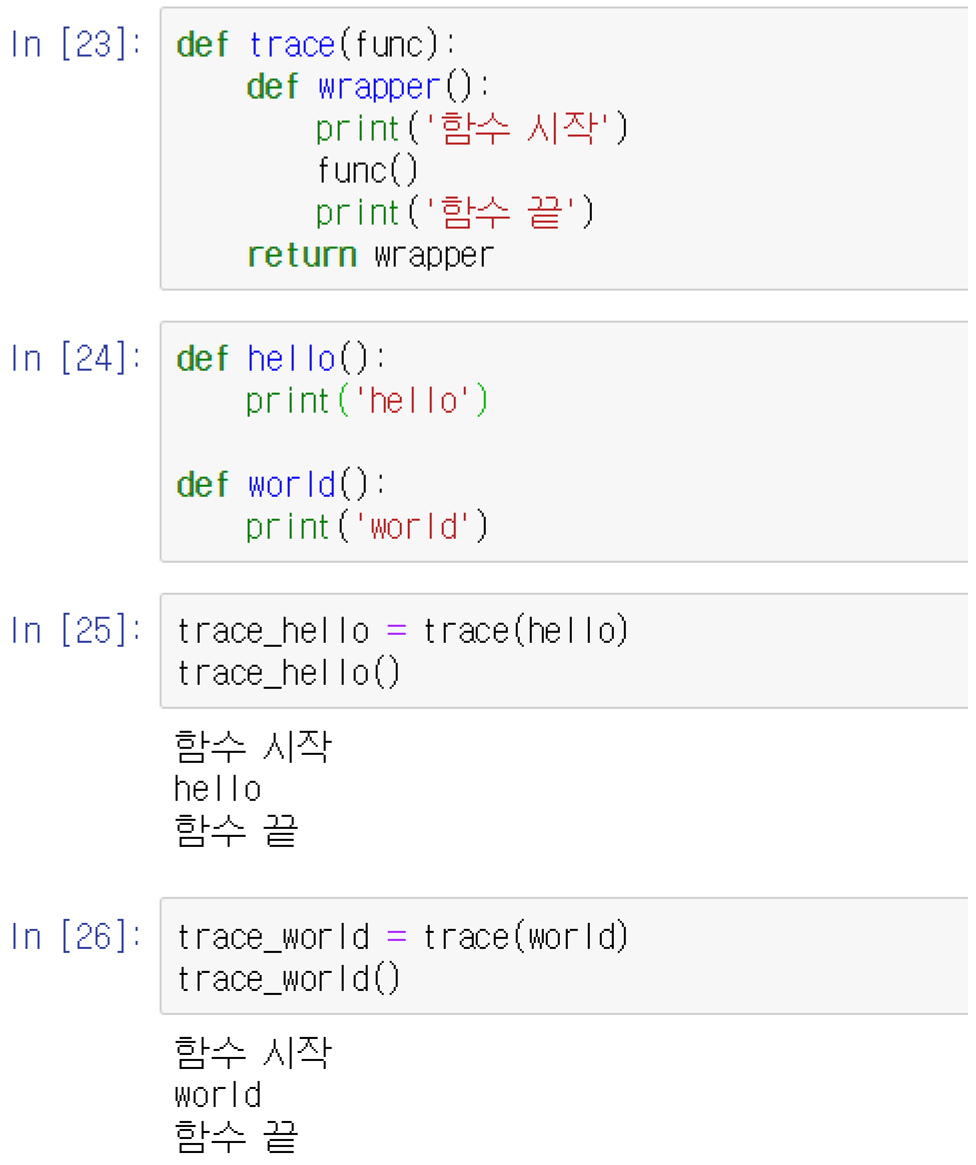


```
In [21]: def hello():
             print('함수 시작')
             print('hello')
             print('함수 끝')

         def world():
             print('함수 시작')
             print('world')
             print('함수 끝')
```

```
In [22]: hello()
         world()
```

```
함수 시작
hello
함수 끝
함수 시작
world
함수 끝
```

decorator



```
In [23]: def trace(func):
             def wrapper():
                 print('함수 시작')
                 func()
                 print('함수 끝')
             return wrapper
```

```
In [24]: def hello():
             print('hello')

         def world():
             print('world')
```

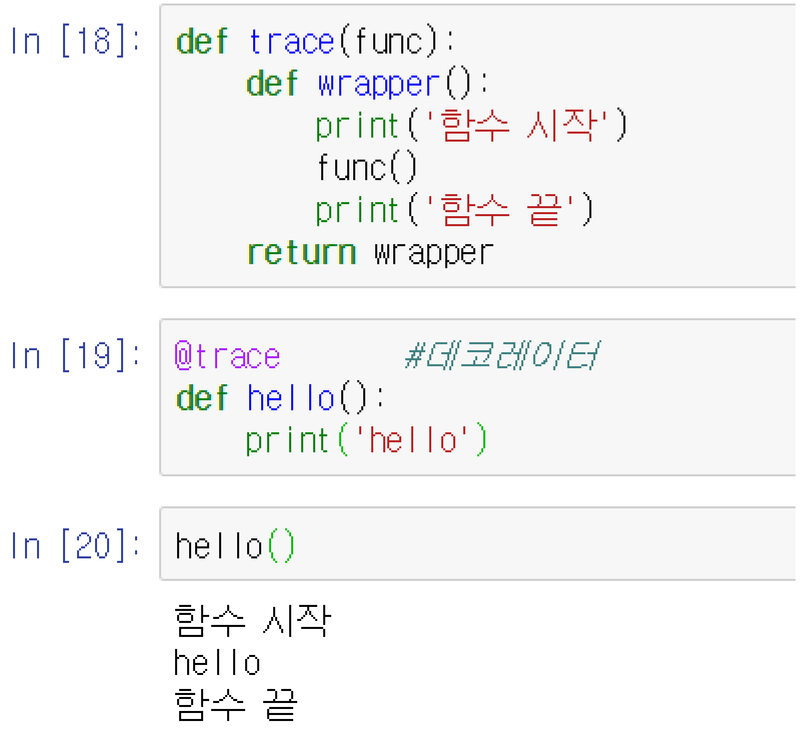
```
In [25]: trace_hello = trace(hello)
         trace_hello()
```

```
함수 시작
hello
함수 끝
```

```
In [26]: trace_world = trace(world)
         trace_world()
```

```
함수 시작
world
함수 끝
```

decorator



```
In [18]: def trace(func):
             def wrapper():
                 print('함수 시작')
                 func()
                 print('함수 끝')
             return wrapper
```

```
In [19]: @trace        #데코레이터
         def hello():
             print('hello')
```

```
In [20]: hello()

         함수 시작
         hello
         함수 끝
```