

함수형; functional



Photo by Sneaky Elbow on Unsplash

함수도 객체다. 즉, 변수로 참조할 수 있다

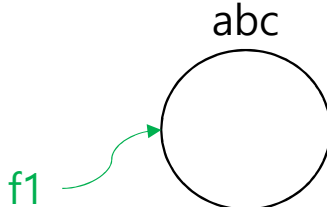
변수에 할당할 수 있다.

```
>>>def abc(text):  
    print(text)  
>>>f1 = abc  
>>>f1("hello")  
hello
```

인자; parameter

abc

f1



간접 방식으로 참조 시키기

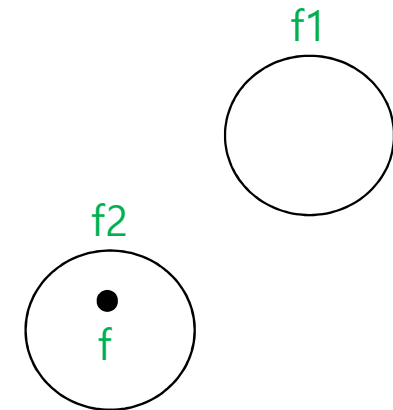
함수의 인자, 함수의 리턴

```
>>>def abc(text):
    print(text)
    return text+" , World"
>>>ret = abc("Hello")  text="Hello"
>>>print(ret)
Hello, World

>>>def add(a, b)
    return a+b
>>>c = add(1, 2)  (a, b) = (1, 2)
```

```
>>>def f1():
    print("f1")
>>>def f2(f):
    return f

>>>a = f2(f1)
>>>type(a)
<class 'function'>
>>>a()
f1
```



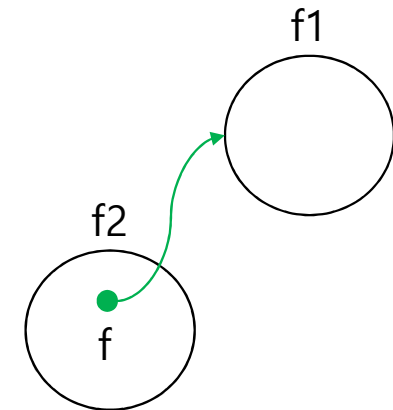
간접 방식으로 참조 시키기

함수의 인자로 함수의 리턴

```
>>>def abc(text):  
    print(text)  
    return text+"", World"  
>>>ret = abc("Hello")  
>>>print(ret)  
Hello, World
```

```
>>>def f1():  
    print("f1")  
>>>def f2(f):  
    return f
```

```
>>>a = f2(f1)  f=f1  
>>>type(a)  
<class 'function'>  
>>>a()  
f1
```

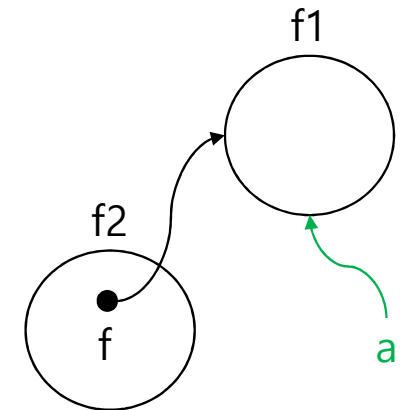


간접 방식으로 참조 시키기

함수의 인자로 함수의 리턴

```
>>>def abc(text):  
    print(text)  
    return text+" , World"  
>>>ret = abc("Hello")  
>>>print(ret)  
Hello, World
```

```
>>>def f1():  
    print("f1")  
>>>def f2(f):  
    return f  
  
>>>a = f2(f1)    f=f1  
>>>type(a)  
<class 'function'>  
>>>a()  
f1
```



함수를 다른 함수의 인자로 넘기고 받고

이런 함수를 functional 이라고 한다.

```
def add(a, b):  
    return a+b  
  
def sub(a, b):  
    return a-b  
  
def cal(f, a, b):  
    return f(a, b)  
  
c = cal(sub, 3, 2)  
print(c)
```

```
def cal(fname):  
    if fname=='add':  
        return add  
    elif fname=='sub':  
        return sub  
    return None
```

```
f = cal('sub')  
c = f(1, 2)  
print(c)
```

```
c = cal('sub')(1, 2)  
print(c)
```

시간만 더 추가된 함수를 만들면,

처리 시간을 출력하고 싶다.

```
import time
def testA():
    time.sleep(1) # 1초가 걸리는 task
def testB():
    time.sleep(2) # 2초가 걸리는 task
```

```
def testA():
    t1 = time.time()
    time.sleep(1) # func for code A
    t2 = time.time()-t1
    print(t2)
def testB():
    t1 = time.time()
    time.sleep(2) # func for code B
    t2 = time.time()-t1
    print(t2)
```

```
>>testA()
1.0
>>testB()
2.0
```

동일한 코드가 testA, testB에
반복해서 들어간다.
간단하게 하고 싶다.

함수 안에 함수

함수 안에 함수를 둘 수 있다

<https://blog.naver.com/nonezerok/221642672593>

```
import time
def testA():
    time.sleep(1)
def testB():
    time.sleep(2)
```

기존 testA, testB 함수는 그대로 둔다.

```
def timeTest(f):
    def wrapper():
        t1=time.time()
        f()
        t2=time.time()-t1
        print(t2)
    return wrapper
```

testA, testB를 확장하려는 함수니까,
함수이름을 timeTest로 지었음

그리고, testA, TestB 중에서 골라야
하니까, f 라는 인자를 하나 둬

함수 호출이 아님에 주의

```
timeTestA = timeTest(testA)
timeTestB = timeTest(testB)
timeTestA(); timeTestB()
```


함수 안에 함수; ① 데코레이터

함수 이름을 그대로 사용하여 확장한다.

```
import time
def timeTest(f):
    def wrapper():
        t1=time.time()
        f()
        t2=time.time()-t1
        print(t2)
    return wrapper
```

```
@timeTest
def testA():
    time.sleep(1)

@timeTest
def testB():
    time.sleep(2)

#testB = timeTest(testB)

testA()
testB()
```

데코레이터 예시

함수 이름을 그대로 사용하여 확장한다.

```
def say(msg):
    print(msg)

def say_hello(func):
    def w(msg):
        print('Hello, ', end='')
        func(msg)
    return w # 확장된 함수를 반환

hello = say_hello(say)
hello('Ko') # Hello, Ko
```

```
def say_hello(func):
    def w(msg):
        print('Hello, ', end='')
        func(msg)
    return w

@say_hello # 데코레이터 지정
def say(msg):
    print(msg)

say('Ko') # Hello, Ko
```

```

import time
import psutil
from functools import wraps

def timeTest(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        #start_time = time.time()
        freq = psutil.cpu_freq().current
        start_time_ps= time.perf_counter()

        ret = func(*args, **kwargs)

        end_time_ps = time.perf_counter()
        #end_time = time.time()

        elapsed_time = end_time_ps - start_time_ps
        #elapsed_time = end_time - start_time
        elapsed_clocks = elapsed_time * freq * 1e6

        print(f"Function {func.__name__}")
        print(f"....elapsed time: {elapsed_time:.6f} seconds")
        print(f"....elapsed clocks: {elapsed_clocks:.6f} clocks")

        return ret, elapsed_time, elapsed_clocks

    return wrapper

```

```

def wrapper(*args, **kwargs):
    print("args:", args)
    print("kwargs:", kwargs)

```

```

wrapper(1, 2, a=3, b=4)
# 출력:
# args: (1, 2)
# kwargs: {'a': 3, 'b': 4}

```

```

@timeTest
def test(a, b):
    c = a + b
    return c

ret, elapsed_time, elapsed_clocks = test(a, b)

```

함수 안에 함수; ② 클로저

변수가 그대로 유지된다.

<https://blog.naver.com/nonezerok/221642672593>

<https://blog.naver.com/nonezerok/221895776802>

```
#foo.py
x = 1
def test(func):
    return func()
```

```
>>>import foo
>>>x = 2
>>>def printX():
    print(x)
>>>foo.test(printX)
```

2 1이 나오면 더 좋을 것 같은데...

```
def who_say(name):
    def say(msg):
        print(name, ': ', end='')
        print(msg)
    return say
```

```
kimSay = who_say('Kim')
kimSay('Hello') # Kim : Hello
kimSay('World') # Kim : World
```

```
koSay = who_say('Ko')
koSay('Hello') # Ko : Hello
koSay('World') # Ko : World
```

값만 남기고 함수를 종료 시키지 않는다

③ 제너레이터

yield

```
def get_data():  
    for i in range(3):  
        yield i  
  
    return
```

과는 달리 함수를
종료 시키지 않는다.
대기한다.
사용하는 쪽에서 `__next__()`를 호출하면
다시 코드 수행.

```
>>>a=get_data()  
>>>type(a)  
<class 'generator'>  
>>>next(a)    #a.__next__()  
0  
>>>next(a)  
1  
>>>next(a)  
2
```

이름 없는 함수; lambda

```
>>>def sq(n):  
    return n*n  
>>sq(2)  
4
```

인자 return 대상

```
>>>lambda i : i*i  
<function <lambda> at 0x0000023773161EA0>
```

```
>>>sq = lambda i : i*i  
>>>sq(2)  
4  
>>>(lambda i : i*i)(2)  
4
```

lambda-인자가 두개인 경우

```
>>>def add(a, b):  
    return a+b  
>>>c = add(1, 2)  
3
```

```
>>>(lambda a, b : a + b)(1, 2)  
>>>3
```

[for]

리스트 컴프리헨션 list comprehension 이라고 부른다. 더 빠르다.

```
>>>a=[1,2,3]
>>>b=[]
>>>for i in a:
...     b.append(-i)
>>>print(b)
[-1, -2, -3]
```

```
>>>a=[1,2,3]
>>>b=[-k for k in a]
>>>print(b)
[-1, -2, -3]
```

$-1 * k$
계산식이 올 수 있다

```
>>>b=[abs(i) for i in b]
>>>print(b)
[1, 2, 3]
```

함수가 올 수 있다.

[for if] [if else for]

```
>>>a=[1,2,3]
>>>b=[-i for i in a if i < 3]
>>>print(b)
[-1, -2]
```

```
>>>a=[1,2,3]
>>>b=[-i if i<3 else i for i in a]
>>>print(b)
[-1, -2, 3]
```

lambda 예시

```
>>>a=[1,2,3]
>>>def sq(n):
    return n*n
>>>b=[sq(i) for i in a]
>>>print(b)
[1, 4, 9]
```

```
>>>a=[1,2,3]
>>>sq = lambda i:i*i
>>>b=[sq(i) for i in a]
>>>print(b)
[1, 4, 9]

>>>a=[1,2,3]
>>>b=[(lambda i:i*i)(i) for i in a]
>>>print(b)
[1, 4, 9]
```

map, filter, reduce

함수에 값을 하나씩 전달한다.

```
>>>a=[1,2,3]
>>>b=[i*i for i in a]
[1, 4, 9]

>>>a=[1,2,3]
>>>def sq(n):
    return n*n
>>>b=[sq(i) for i in a]
[1, 4, 9]
```

map(함수, 이터러블 객체) -> 이터러블 객체

```
>>>b = map(sq, a)
>>>type(b)
<class 'map'>
>>>dir(b)
[ '__iter__' ]
>>>list(b)
[1, 4, 9]
```

함수가 와야 한다!

당연히, 이터러블 객체

map

입력이 여러 개인 함수에 차례대로 할당할 때

```
>>>def add(a,b):  
    return a+b  
  
>>>a=[1,2,3]  
>>>b=[1,2,3]  
>>>c=map(add, a, b)  
>>>list(c)  
[2, 4, 6]
```

map, filter, reduce

삼인방!

filter(함수, 이터러블 객체) -> 이터러블 객체

```
>>>a = map(int, ('1','2','3'))
>>>list(a)
[1,2,3]
```

```
>>>a=[]
>>>for i in range(1,11):
>>>    if i % 2 == 0:
>>>        a.append(i)
>>>print(a)
[2, 4, 6, 8, 10]
```

```
>>>a=[i for i in range(1,11) if i%2==0]
>>>print(a)
[2, 4, 6, 8, 10]

>>>a=filter(lambda i: i%2==0, range(1,11))
>>>list(a)
[2, 4, 6, 8, 10]
```

map, filter, reduce

삼인방!

reduce(함수(a, b), 이터러블 객체, 값) -> 값

```
>>>s=0
>>>for i in range(1,11):
>>>    s+=i
>>>print(s)
55
```

```
>>>from functools import reduce
>>>def acc(i, j):
>>>    return i+j
>>>s = reduce(acc, range(1,11), 0)
>>>print(s)
55
```

기타

Photo by Annie Spratt on Unsplash



for - enumerate

인덱스를 만들어 낼 수 있다.

```
>>>a=['abc', 'def', 'ghi']  
>>>for i, v in enumerate(a):  
    print(i, v)
```

```
0 abc  
1 def  
2 ghi
```


zip

iterator 객체 두개 이상을 묶는다. dir(obj) 했을 때 `__iter__()`을 가지고 있는 객체

```
>>>a=[1,2,3]
>>>b=['a', 'b', 'c']
>>>for i, j in c:
    print(i, j)
```

```
1 a
2 b
3 c
```

```
>>>list(zip(a, b))
[(1, 'a'), (2, 'b'), (3, 'c')]
```

```
>>>a=[1,2]
>>>b=['a', 'b', 'c']
>>>list(zip(a, b))
[(1, 'a'), (2, 'b')]
```

벡터 계산

리스트 +는 이어 붙이기, numpy는 벡터 및 행렬계산 패키지

```
>>>a=[1,2,3]
>>>b=[4,5,6]
>>>c=a+b
>>>print(c)
[1,2,3,4,5,6]
```

```
>>>c=[]
>>>for i in range(len(a)):
>>>    c.append(a[i]+b[i])
>>>print(c)
[5, 7, 9]
```

```
>>>c=[i+j for i, j in zip(a, b)]
>>>print(c)
[5, 7, 9]
>>>import numpy as np
>>>a=[1,2,3]; b=[4,5,6]
>>>a = np.array(a); b = np.array(b)
>>>c = a + b
>>>type(c)
<class 'numpy.ndarray'>
>>>c
array([5, 7, 9])
```

날짜

```
>>>import datetime
>>>datetime.datetime.now()
datetime.datetime(2020, 1, 1, 16, 59, 7, 91684)
>>dir(datetime.datetime)
['__add__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__ne__', '__new__', '__radd__', '__reduce__', '__reduce_ex__',
'__repr__', '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__',
'__subclasshook__', 'astimezone', 'combine', 'ctime', 'date', 'day', 'dst', 'fold',
'fromordinal', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday',
'max', 'microsecond', 'min', 'minute', 'month', 'now', 'replace', 'resolution',
'second', 'strftime', 'strptime', 'time', 'timestamp', 'timetuple', 'timetz',
'today', 'toordinal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset',
'utctimetuple', 'weekday', 'year']
```

import

```
def f1():  
    print("abc")  
def f2():  
    print("def")
```

hello.py

```
>>>import hello  
>>>hello.f1()  
abc  
>>>import hello as h  
>>>h.f1()  
abc
```

```
>>>from hello import f1, f2  
>>>f1()  
abc  
  
>>>dir(hello)  
['__name__', 'f1', 'f2']  
>>>hello.__name__  
'hello'
```

import

```
def f1():  
    print("abc")  
def f2():  
    print("def")  
print('hello.py')  
print(__name__)
```

hello.py

```
C:\>python hello.py  
hello.py  
__main__
```

```
import hello  
print('a.py')  
print(__name__)
```

a.py

hello.py에 있는 것도 다 실행된다.

```
C:\>python a.py  
hello.py  
hello  
a.py  
__main__
```

__name__, __main__

```
def f1():  
    print("abc")  
def f2():  
    print("def")  
if __name__ == '__main__':  
    print('hello.py')  
    print(__name__)
```

hello.py

```
C:\>python hello.py  
hello.py  
__main__
```

```
import hello  
print('a.py')  
print(__name__)  
hello.f1()
```

a.py

a.py 것만 실행된다.

```
C:\>python a.py  
a.py  
__main__  
abc
```

함수 호출 시기를 미룬다

지정한 함수를 정해진 시간 뒤에 호출하도록 할 수 있다.

```
import threading
import datetime

def fa():
    print(datetime.datetime.now(), "in fa")
#fa()
t = threading.Timer(3, fa)
t.start()
print(datetime.datetime.now(), "in main")
print("end")
```

```
threading.Timer(3, fa).start()
```

```
2020-01-01 22:43:50.712549 in main
end
2020-01-01 22:43:53.714868 in fa
```

스레드 함수라고 한다

그전에 취소할 수도 있다.

```
import threading
import datetime

def fa():
    print(datetime.datetime.now())

t = threading.Timer(3, fa)
t.start()
t.cancel()
print(datetime.datetime.now())
print("end")
```

```
2020-01-01 22:43:50.712549
end
```


스레드 함수에 값을 넘길 수 있다

리스트 객체 활용

```
import threading
def fa(a):
    print(a)

t = threading.Timer(1, fa, args=['hello'])
t.start()
print("end")
```

```
end
hello
```

시간 제약이 있는 덧셈 테스트

시간 내에 못 맞추면 틀린 것으로 판정

```
import random
import threading
def addTest(a, b):
    global TimeOut
    c = a + b
    print(a,"+",b,"= ", end="")
    ans = int(input())
    if ans == c:
        print("correct")
        if TimeOut==True:
            print("but, time is over"); return 0
        return 1
    else:
        print("incorrect"); return 0
```

```
TimeOut = False
def timeOut():
    global TimeOut
    TimeOut = True
score = 0
for i in range(3):
    a = random.randint(0,10)
    b = random.randint(0,10)
    threading.Timer(3, timeOut).start()
    TimeOut = False
    score = score + addTest(a, b)
print("\nYour score is", score)
```



Photo by Annie Spratt on Unsplash