IPA 주관 인공지능센터 기본(fundamental) 과정

- GitHub link: here
- E-Mail: windkyle7@gmail.com

데코레이터 (Decorator)

덕 타이핑(Duck-Typing), 다이나믹 타이핑(Dynamic-Typing)이라는 특징을 살려 일급 객체(first-class) 함수, 클로저(closur), 데코레이터(decorator) 등 다양한 방식으로 함수를 정의할 수 있다.

다음과 같이 함수를 반환하는 함수를 정의한다.

In [1]:
```
def x():
    def y():
        print(a)
        a = 3
    return y
```

함수 x를 호출하여 반환된 함수 y를 실행하면 다음과 같이 UnboundLocalError가 발생한다.

In [2]:
```
x()()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-2-f060ad47541e> in <module>
----> 1 x()()

<ipython-input-1-93e0b6be07c0> in y()
      1 def x():
      2     def y():
----> 3         print(a)
      4         a = 3
      5     return y

UnboundLocalError: local variable 'a' referenced before assignment
```

함수 내부에서 식별자 a가 먼저 정의되어 있지 않은 상태 즉, 바인딩(bind)을 하지 않은 상태에서 print 함수를 호출했기 때문에 에러가 발생하게 된다.

식별자 a가 global로 선언되어있다면 에러가 발생하지 않는다.

In [3]:
```
a = 10
```

In [4]:
```
def x():
    def y():
        global a
        print(a)
        a = 3
    return y
```

In [5]:
```
x()()
```

```
10
```

함수 안에 함수가 정의되어 있다면 이를 데코레이터 (Decorator)로 선언할 수 있다.

In [6]:
```
def x(func):
    def y():
        func()
    return y
```

함수 x를 호출하려고 하면 인자값을 넣어주지 않았으므로 아래와 같이 TypeError가 발생한다.

In [7]:
```
x(lambda x: x + 1)()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-302c81f7a9ee> in <module>
----> 1 x(lambda x: x + 1)()

<ipython-input-6-d8dbaeca5a9c> in y()
      1 def x(func):
      2     def y():
----> 3         func()
      4     return y

TypeError: <lambda>() missing 1 required positional argument: 'x'
```

함수 y에 인자값을 맞춰주지 않았으므로 TypeError가 발생한다.

In [8]:
```
x(lambda x: x + 1)(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
<ipython-input-8-b413fb330c82> in <module>
----> 1 x(lambda x: x + 1)(1)

TypeError: y() takes 0 positional arguments but 1 was given
```

In [9]:
```
def x(func):
    def y(z):
        return func(z)
    return y
```

In [10]:
```
x(lambda x: x + 1)(1)
```

Out[10]: 2

위처럼 호출한 함수 `x` 는 결론적으로 아래의 함수 `add` 와 동일한 역할을 수행한다.

In [11]:
```
def add(num):
    return num + 1
```

In [12]:
```
add(1)
```

Out[12]: 2

다음과 같이 두 식이 있다고 했을 때,

- `f(x) = (x + 1) ^ 2`
- `g(x) = (x - 10) ^ 2`

최종적으로 제곱하는 부분은 동일하기 때문에 이를 간단히 하기 위해 데코레이터를 활용해서 구현해본다면 다음과 같이 작성할 수 있다.

In [13]:
```
def squared(func):
    return lambda x: func(x) ** 2
```

In [14]:
```
@squared
def f(x):
    return x + 1
```

In [15]:
```
@squared
def g(x):
    return x - 10
```

In [16]:
```
f(1)
```

Out[16]: 4

In [17]:
```
g(1)
```

Out[17]: 81

데코레이터를 중첩해서(chaning) 사용할 수는 있지만, 권장하는 방식은 아니다.

In [18]:
```
def decor_1(func):
    def wrapper_1():
        print('call wrapper_1')
        func()
    return wrapper_1
```

In [19]:
```
def decor_2(func):
    def wrapper_2():
        print('call wrapper_2')
        func()
    return wrapper_2
```

In [20]:
```
@decor_1
@decor_2
def f():
    print('call f')
```

In [21]:
```
f()
```

```
call wrapper_1
call wrapper_2
call f
```

위의 코드처럼 중첩해서 사용하면 다음과 같은 절차가 이루어진다.

1. 함수 `f` 호출 시 선언된 데코레이터 `decor_2` 에 인자값으로 함수 `f` 가 넘어간다.
2. 함수 `decor_2` 는 `wrapper_2` 함수를 반환한다.
3. `decor_2` 를 통해 반환된 `wrapper_2` 함수가 데코레이터 `decor_1` 에 인자값으로 넘어간다.
4. `decor_1` 을 통해 `wrapper_1` 함수를 반환한다.
5. 즉, 최종 리턴값은 `decor_1` 을 통해 반환된 함수 `wrapper_1` 이므로 `wrapper_1` 를 수행하게 된다.
6. 함수 `wrapper_1` 에 인자값으로 `decor_2` 를 통해 반환된 함수 `wrapper_2` 가 넘어갔으므로 `wrapper_2` 를 수행하게 된다.
7. 최종적으로 함수 `f` 를 수행한다.

다시 살펴보는 함수를 반환하는 함수

In [22]:
```
def f():
    def g(x, y):
        return x + y
    return g
```

In [23]:
```
g = f()
```

In [24]:
```
g
```

Out[24]:
```
<function __main__.f.<locals>.g(x, y)>
```

In [25]:
```
g(1, 2)
```

Out[25]:
```
3
```

In [26]:
```
import inspect
```

In [27]:
```
print(inspect.getsource(g))
```

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