고급 문법

고급 문법

❖ 열거 가능 객체

- for 반복문의 순회 대상 객체
- 해당 객체의 __iter__() 메서드로 열거 가능 객체 획득
  - 열거 가능 개체는 __iter__() 메서드를 정의해야 함
- 매 루프마다 __next()__ 함수를 통해 다음 요소를 추출
- 더 이상 요소가 없는 데 __next()__를 호출하는 경우
  - StopIteration 예외가 발생하고 for 반복문을 끝냅

고급 문법

❖ 열거 가능 객체

```
nums  = [11, 22, 33]

it = iter(nums)
while True:
    try:
        num = next(it)
    except StopIteration:
        break
    print(num)
```

```
11
22
33
```

고급 문법

❖ 열거 가능 객체

```
class Seq:
    def __init__(self, data):
        self.data = data
        self.index = -2

    def __iter__(self):
        return self

    def __next__(self):
        self.index += 2
        if self.index >= len(self.data):
            raise StopIteration

        return self.data[self.index:self.index+2]
```

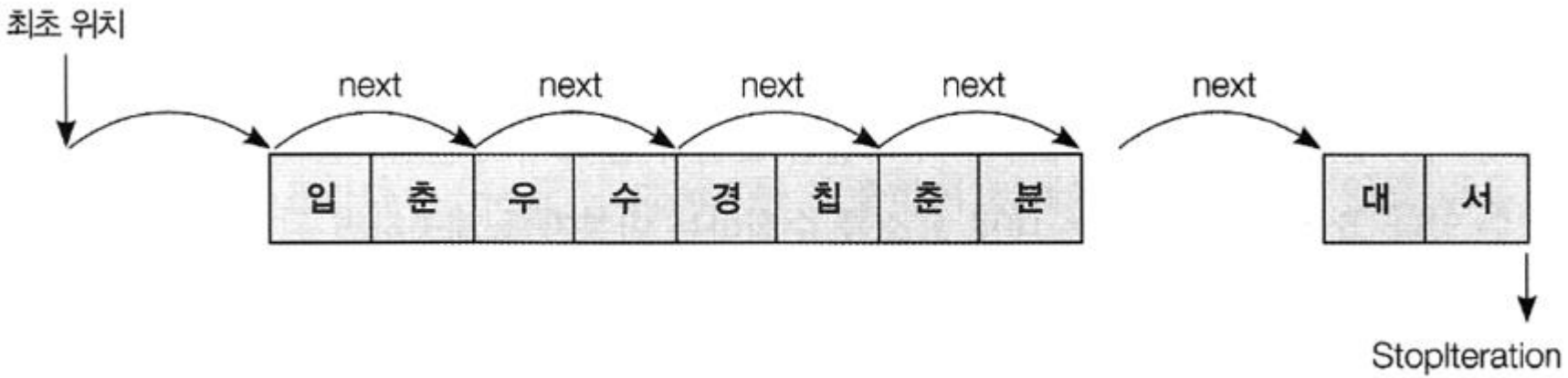
고급 문법

❖ 열거 가능 객체

```
solarterm = Seq("입춘우수경칩춘분청명곡우입하소만망종하지소서대서")

for k in solarterm:
    print(k, end = ',')
```

입춘,우수,경칩,춘분,청명,곡우,입하,소만,망종,하지,소서,대서,



제너레이터

❖ 제너레이터

- 객체로 순회가능한 객체를 만드는 거는 다소 귀찮은 작업
- 제너레이터 함수로 대체 가능
- 함수에서 데이터를 연속해서 리턴(yield)
- 함수가 끝나면(또는 return 실행) StopIteration 예외 발생
- 함수를 호출하면 함수가 실행되는 것이 아니고 순회 가능 객체가 리턴
- 순회 가능 객체로 순회할 때 실제 함수가 실행됨

제너레이터

❖ 제너레이터

```
def seqgen(data):
    for index in range(0, len(data), 2):
        yield data[index:index+2]

solarterm = seqgen("입춘우수경칩춘분청명곡우입하소만망종하지소서대서")

for k in solarterm:
    print(k, end = ',')
```

입춘,우수,경칩,춘분,청명,곡우,입하,소만,망종,하지,소서,대서,

데코레이터

❖ 일급 시민

- 함수도 일반 변수와 동일한 특성을 가짐
- 이름을 가진다.
- 다른 변수에 대입할 수 있다.
- 인수로 전달할 수 있다.
- 리턴값이 될 수 있다.
- 컬렉션에 저장할 수 있다.

--> 위와 같은 특성을 가지는 것을 일급시민이라고 함

데코레이터

❖ 일급시민

```
def add(a, b):
    print(a + b)

plus = add   # 변수에 저장 할 수 있다
plus(1, 2)
```

```
3
```

데코레이터

❖ 일급시민

```
def calc(op, a, b):   # 함수의 인자로 전달할 수 있다.
    op(a, b)

def add(a, b):
    print(a + b)

def multi(a, b):
    print(a * b)

calc(add, 1, 2)
calc(multi, 3, 4)
```

```
3
12
```

데코레이터

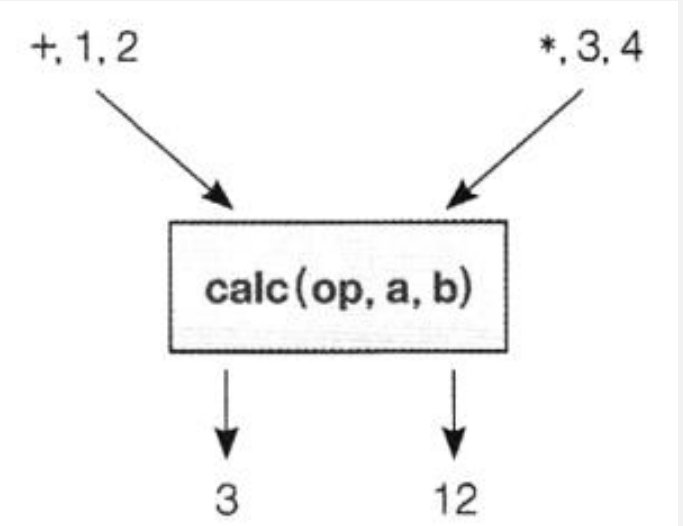
❖ 일급시민

```
def calc(op, a, b):   # 함수의 인자로 전달할 수 있다.
    op(a, b)

def add(a, b):
    print(a + b)

def multi(a, b):
    print(a * b)

calc(add, 1, 2)
calc(multi, 3, 4)
```




```
3
12
```

데코레이터

❖ 지역 함수

- 함수 안에 함수를 정의해서 사용
  - 함수가 정의된 함수 내에서만 사용 가능

  --> 함수의 이름 충돌 방지
  - 함수를 리턴한 경우 함수 밖에서도 사용 가능

데코레이터

❖ 지역 함수

```
def calcsum(n):
    def add(a, b):
        return a + b

    total = 0
    for i in range(n+1):
        total = add(total, i)

    return total

print("~ 10 = ", calcsum(10))
```

```
~ 10 =  55
```

```
def add(a, b):
    return a + b

def calcsum(n):
    sum = 0
    for i in range(n + 1):
        sum = add(sum, i)
    return sum
```

데코레이터

❖ 지역 함수

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

enghello = makeHello("Good Morning")
kohello = makeHello("안녕하세요")

enghello("Mr Kim")
kohello("홍길동")
```

```
Good Morning, Mr Kim
안녕하세요, 홍길동
```

데코레이터

❖ 함수 데코레이터

- 이미 만들어진 함수에 동작을 추가하는 파이썬의 고급 기법
- 함수를 래핑(Wrapping)하여 함수의 앞 또는 뒤에 코드를 자동으로 추가
- 함수를 호출하면 추가된 앞, 뒤의 코드도 같이 실행됨

데코레이터

❖ 함수 데코레이터

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

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

outer(inner)
```

```
--------------------
결과를 출력합니다.
--------------------
```

```
def hello():
    print("안녕하세요")

outer(hello)
```

```
--------------------
안녕하세요
--------------------
```

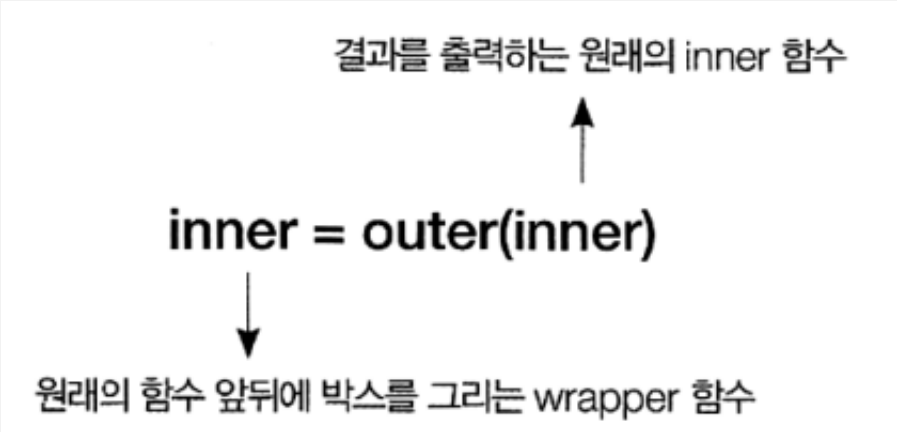
데코레이터

❖ 함수 데코레이터

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

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

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




```
--------------------
결과를 출력합니다.
--------------------
```

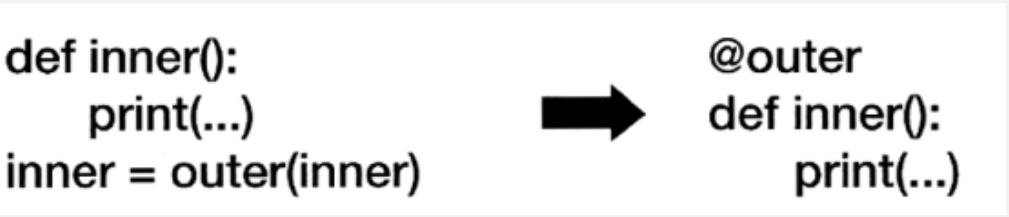
데코레이터

❖ 함수 데코레이터 @함수명

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

@outer
def inner():
    print("결과를 출력합니다.")

inner()
```




```
--------------------
결과를 출력합니다.
--------------------
```

데코레이터

❖ 함수 데코레이터 @함수명

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

@para
def outname():
    return "김상형"

@para
def outage():
    return "29"

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

```
<p>김상형</p>
<p>29</p>
```

데코레이터

❖ 함수 데코레이터 @함수명

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

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

데코레이터

❖ 함수 데코레이터 @함수명

```
@div
@para
def outname():
    return "김상형"
@div
@para
def outage():
    return "29"

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

```
<div><p>김상형</p></div>
<div><p>29</p></div>
```

데코레이터

❖ 함수 데코레이터 @함수명

```
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("김상형"))
print(outage(29))
```

```
Traceback (most recent call last):
  File, line 14, in <module>
    print(outname("김상형"))
TypeError: wrapper() takes 0 positional arguments but 1 was given
```

데코레이터

❖ 함수 데코레이터 @함수명

```
def para(func):
    def wrapper(*args, **kwargs):
        return "<p>" + str(func(*args, **kwargs)) + "</p>"
    return wrapper

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

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

print(outname("김상형"))
print(outage(29))
print(outname.__name__)
```

```
<p>이름: 김상형님</p>
<p>나이: 29</p>
wrapper
```

데코레이터

❖ 함수 데코레이터 @함수명

```
from functools import wraps
def para(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        return "<p>" + str(func(*args, **kwargs)) + "</p>"
    return wrapper

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

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

print(outname("김상형"))
print(outage(29))
print(outname.__name__)
print(outage.__name__)
```

```
<p>이름: 김상형님</p>
<p>나이: 29</p>
outname
outage
```

데코레이터

❖ 클래스 데코레이터

- ○ __callable__ 메서드
  - ▪ 클래스를 함수 호출하듯이 사용했을 때 호출되는 메서드

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

    def __call__(self):
        print("-"*20)
        self.func()
        print("-"*20)

def inner():
    print("결과를 출력합니다.")

inner = Outer(inner)
inner()
```

```
--------------------
결과를 출력합니다.
--------------------
```

데코레이터

❖ 클래스 데코레이터

- ○ __callable__ 메서드
  - ▪ 클래스를 함수 호출하듯이 사용했을 때 호출되는 메서드

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

    def __call__(self):
        print("-"*20)
        self.func()
        print("-"*20)

@Outer
def inner():
    print("결과를 출력합니다.")

inner()
```

```
--------------------
결과를 출력합니다.
--------------------
```