Chapter. 03

파이썬 프로그래밍 언어

I고급 주제들2

FASTCAMPUS
ONLINE
금융공학/퀀트I

강사. 서찬웅

이번시간에 배울 내용

1. 클래스
2. 클래스 연산자 중복
3. 반복자, 발생자
4. 코루틴
5. 클로저
6. 데코레이션

Copyright FASTCAMPUS Corp. All Rights Reserved

I 객체지향이란

- 컴퓨터가 등장하고 컴퓨터의 활용이 전 산업으로 확대되면서 컴퓨터의 프로그램이 기능이 많아지고 프로그램의 크기도 엄청나게 증가하면서 효율적으로 코딩할 수 있는 방법이 연구되었습니다.
- 객체지향 프로그래밍(Object Oriented Programming, 줄여서 OOP라고 부름)은 이런 연구 결과로 만들어진 프로그래밍 방법 중 하나입니다. 특징은 객체(Object)라는 단위로 프로그래밍을 합니다.
- 실세계를 컴퓨터로 시물레이션하기 위해서 1960년대에 만들어진 simula67이라는 프로그래밍 언어에서 채택된 방법이며, 모든 사물을 객체(Object)라 하고 그에 따른 기능과 속성을 분석하고 이들을 프로그래밍에서 함수와 변수로 표현이 시초입니다.
- 그 뒤에 객체지향이라는 방법이 여러 프로그래밍 언어로 전파되면서 다양한 기능이 추가되면서 발전된 형태

- 객체지향의 장점
  - 객체지향프로그래밍을 적용하면 프로그램을 작은 객체 단위로 나누어 만들수 있습니다.
  - 같은 기능을 하는 코드를 효율적으로 공유 및 재사용하고 공유할 수 있다.
  - 이미 검증된 코드를 재사용하기 때문에 코드에 대한 신뢰도가 높아집니다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

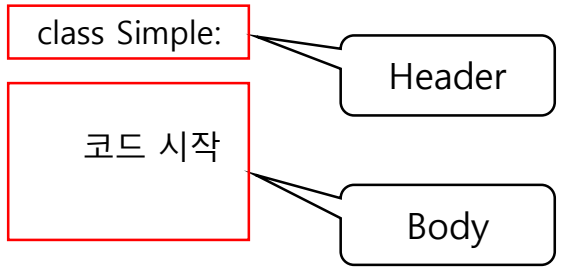
I 클래스란 무엇인가?

- 클래스는 새로운 이름 공간을 지원하는 단위
- 이 이름 공간에는 함수와 변수가 포함될 수 있다. 이러한 점에서는 모듈과 유사하다.
- 차이점은 모듈은 파일 단위로 이름 공간을 구성하는 반면, 클래스는 이름 공간과 클래스가 생성하는 인스턴스 이름 공간을 각각 갖는다.
- 클래스 이름 공간과 인스턴스 이름 공간은 유기적인 관계로 연결되어 있으며 상속 관계에 있는 클래스 간의 이름 공간도 유기적으로 연결되어 있다.
- 클래스를 정의하는 것은 새로운 자료형을 하나 만드는 것이고, 인스턴스는 이 자료형의 객체를 생성하는 것

- 클래스의 특징은?
  - 클래스는 파이썬의 주요 객체지향 프로그래밍 도구
  - 클래스를 알게 되면 프로그래밍이 훨씬 간결해지고 프로그램을 체계적으로 만들 수 있다.
  - 파이썬의 클래스는 다른 언어에 비해서 간결하며 사용하기 쉽다.

Copyright FASTCAMPUS Corp. All Rights Reserved

I 클래스의 생성

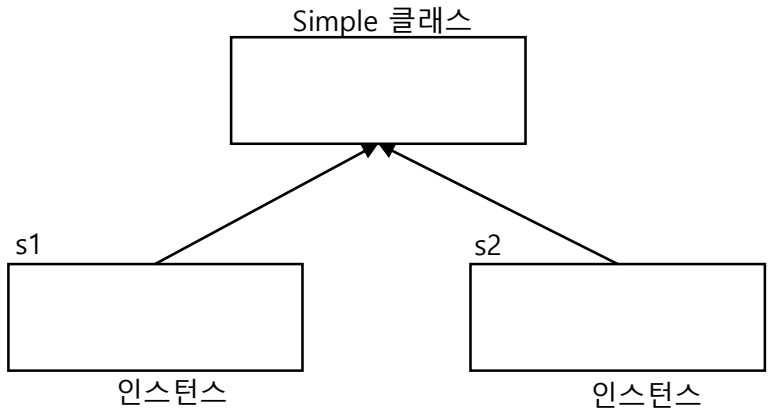


첫 줄은 class 키워드와 클래스 이름이 나온다. 이 줄은 Header라고 한다.
클래스의 이름은 Simple이다.
콜론 : 을 붙이는 것을 잊지 말 것

Body 는 다음 행부터 들여쓰기 된 상태로 기술한다.

모든 파이썬 클래스는 Object 클래스를 Base 클래스로 한다.

```
class Simple:
    pass

s1 = Simple()
s2 = Simple()
```



클래스의 인스턴스 객체 s1과 s2를 생성하였습니다.
인스턴스 객체를 생성하는 방법은 마치 함수를 호출하듯이
클래스를 호출하면 됩니다.
클래스를 호출하면 인스턴스 객체를 생성한 후 해당하는 참조를 반환한다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 클래스 생성

- S1이라는 공간에 만들어진 a라는 변수에 접근을 할려면 .을 사용해서 접근할수 있습니다.

```
class test_class:
    a = 1
```

class라는 키워드를 사용하여 클래스를 생성합니다.
test_class 이라는 공간에서 내부에서 사용할
a라는 변수에 1을 할당합니다.

```
test_class.a
```

```
1
```

```
test_class.a = 2
```

test_class이라는 공간에 존재하는 a에 2라는 숫자를 대입

```
test_class.a
```

```
2
```

```
x = test_class()
```

test_class과 동일한 클래스를 x라는 변수에 대입하는데 이를 test_class 클래스의 인스턴스 객체 x라고 합니다.
test_class의 a가 2인 상태에서 그대로 복사되어 x.a도 2가 출력됩니다.

```
x.a
```

```
2
```

```
y = test_class()
y.a = 100
```

위와 같은 방식으로 y라는 인스턴스를 생성합니다.
그리고 y.a에 100을 대입합니다.

```
y.a
```

```
100
```

```
x.a
```

객체 x와 객체 y는 서로 다른 공간을 가지고 있습니다.

```
2
```


Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 클래스란 무엇인가?

인스턴스 객체의 이름 공간 안에 stack이라는 리스트를 저장하는 변수를 생성합니다.

```
s1.stack = []
```

stack 변수에 1, 2, 3을 추가합니다.

```
s1.stack.append(1)
s1.stack.append(2)
s1.stack.append(3)
```

s1.stack의 값을 출력합니다.

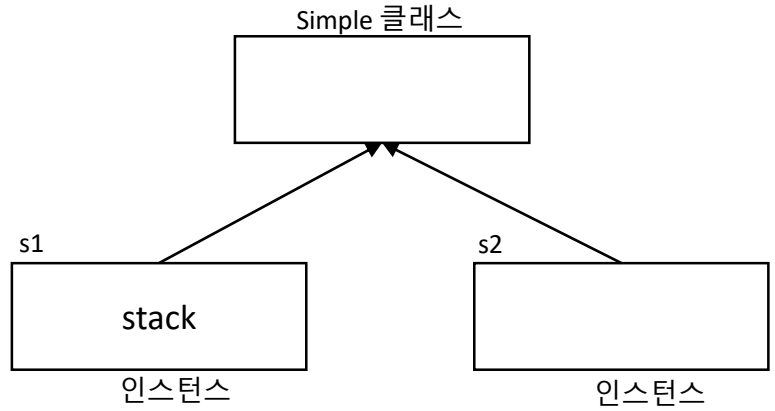
```
s1.stack
```

```
[1, 2, 3]
```

s2에도 stack이라는 변수가 있는지 호출해 봅시다.

```
s2.stack
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-c3ee0a2b3490> in <module>()
----> 1 s2.stack

AttributeError: 'Simple' object has no attribute 'stack'
```



클래스의 인스턴스 객체 s1은 클래스 Simple 안에 내포된 독립적인 이름 공간을 가지며, 이 이름 공간에서는 동적으로 이름을 설정하는 것이 가능하다.

Copyright FASTCAMPUS Corp. All Rights Reserved

I 메소드(Method)와 정의와 호출

- 메소드를 정의하는 방법은 사용자 정의 함수를 정의하는 것과 동일합니다. 다른 점이 있다면 메소드의 첫 번째 인수는 반드시 해당 클래스의 인스턴스 객체이어야 합니다. 관례로 우리는 self란 이름으로 첫 번째 인수로 선언합니다.
- 인스턴스 객체를 이용하여 호출하는 방식을 Bound Method Call이라고 합니다.
- 메소드는 클래스 안에서 구현된 함수입니다.

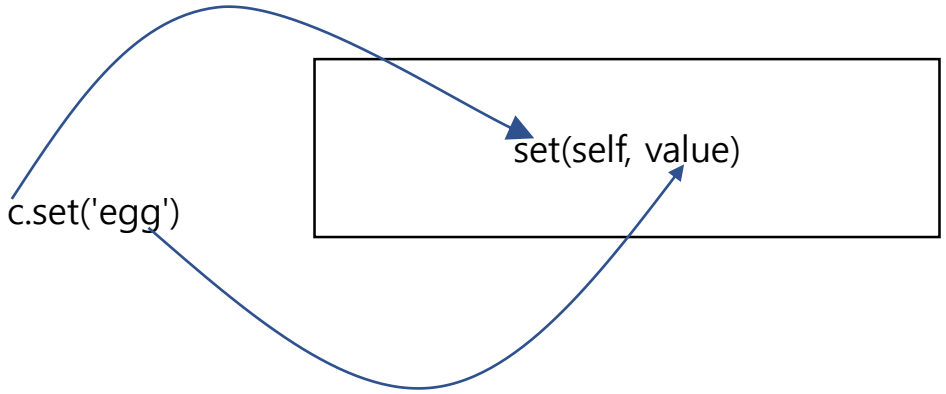
```
class MyClass:
    def set(self, v):
        self.value = v

    def get(self):
        return self.value
```

```
C = MyClass()
```

```
C.set('egg')
```

```
C.get()
```

```
'egg'
```



- c.set()을 호출하면 c의 인스턴스 객체가 self로 암시적으로 전달됩니다. 따라서 c.set('egg')와 같이 두 번째 인수부터 전달하면 됩니다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 내부에서 인스턴스 멤버나 메소드를 호출할 때도 self를 이용합니다.


```
class MyClass2:
    def set(self, v):
        self.value = v

    def get(self):
        return self.value

    def incr(self):
        self.value += 1
        return self.get()
```

```
D = MyClass2()
```

```
D.set(5)
```

```
D.get()
```

```
5
```

```
D.incr()
```

```
6
```

MyClass2 안의 incr() 메서도 안에서 get 함수를 호출하기 위해서 self를 사용하여 self.get() 함수를 호출하였습니다.

만약 get() 함수를 호출한다면 MyClass2 밖에 정의된 함수중에 get() 함수가 있는지 알고 get() 함수를 호출할 것입니다.

클래스 멤버나 메서드를 참조하려면 언제나 self를 이용하는 것을 잊지 말아야 합니다.

Copyright FASTCAMPUS Corp. All Rights Reserved

I 생성자와 소멸자

- 클래스로 객체를 생성 할 때 자동으로 실행되는 __init__라는 이름의 메소드를 사용할 수 있습니다.
- 메소드를 실행은 반드시 객체.메서드() 형식으로 호출되어야 하지만, 별도로 호출되지 않아도 객체가 만들어질 때 자동으로 실행되는 메소드가 있습니다.
- 이러한 메서드를 생성자(constructor) 또는 생성자 메서드, initializer라고 부릅니다.
- 생성자는 객체가 생성될 때 변수에 초기값을 할당하거나, 제일 먼저 실행되어야 하는 명령문을 기술하기 위해 사용되는 특수한 메서드입니다.
- 생성자와 반대로 소멸될 때 자동으로 호출되는 메서드를 소멸자(Destructor)라고 부릅니다.
- 소멸자는 인스턴스 객체에서 사용하고서 메모리에 제거될 때 자동으로 호출됩니다.
- 생성자와 소멸자는 파이썬 클래스에서는 아래와 같은 이름으로 정의 됩니다.
    - 생성자 함수의 이름 : __init__
    - 소멸자 함수의 이름 : __del__

Copyright FASTCAMPUS Corp. All Rights Reserved

I 생성자와 소멸자


```
from time import ctime
class Income:

    def __init__(self, price, tax):
        self.price = price
        self.tax = tax

    def calcurator(self):
        self.after_tax = self.price - (self.price * self.tax * 0.01)

    def display(self):
        print ("수입 : ", self.price)
        print ("세율 : ", self.tax)
        print ("세후 : ", self.after_tax)

    def __del__(self):
        print ("메모리에서 삭제되었습니다 - ", ctime())
```

```
Obj1 = Income(2500000, 4.4)
```

```
Obj1.calcurator()
```

```
Obj1.display()
```

```
수입 :  2500000
세율 :  4.4
세후 :  2390000.0
```

```
del Obj1
```

```
메모리에서 삭제되었습니다 -  Thu Jun 21 03:48:34 2018
```

Income 클래스의 인스턴스 객체가 생성되면 자동으로 __init__() 메서드가 호출되어 초기값으로 입력 받은 price, tax를 내부에서 사용할 수있는 price, tax에 대입합니다.

calcurator 메서드는 내부 변수 price, tax를 가지고 계산을 하고 내부 변수 after_tax에 저장합니다.

display 메서드는 내부 변수들을 print 문을 사용하여 출력합니다.

__del__은 소멸자로써 Obj1이 삭제가 될 때 자동으로 실행이 됩니다. 하지만 소멸자는 자주 정의되지 않습니다. 대부분의 메모리나 자원 관리가 자동으로 이루어지기 때문에 특별한 조취를 취하지 않아도 인스턴스 객체가 제거되면서 자원이 원상 복귀되기 때문입니다.

Copyright FASTCAMPUS Corp. All Rights Reserved

I 연산자 중복 메소드에 대해서 알아보겠습니다.

- 연산자에 대해서 이전에 공부하였습니다. 정수나 실수에 대한 연산자를 하면 해당 연산에 대한 결과가 출력됩니다.
  - 파이썬이 기본 자료형은 객체이며, dir() 함수에서 나타는 결과는 해당 객체의 속성(메소드) 목록입니다.
    - print() 함수의 end 옵션의 값을 || 으로 설정해서 출력을 아래와 같이 하였습니다.

```
a = 10
```

```
for x in dir(a):
    print (x, end="||")
```

```
__abs__||__add__||__and__||__bool__||__ceil__||__class__||__delattr__||__dir__||__divmod__||__doc__||__eq__||__float__||__floor__||__floord
iv__||__format__||__ge__||__getattribute__||__getnewargs__||__gt__||__hash__||__index__||__init__||__init_subclass__||__int__||__invert__||
__le__||__lshift__||__lt__||__mod__||__mul__||__ne__||__neg__||__new__||__or__||__pos__||__pow__||__radd__||__rand__||__rdivmod__||__reduce
__||__reduce_ex__||__repr__||__rfloordiv__||__rlshift__||__rmod__||__rmul__||__ror__||__round__||__rpow__||__rrshift__||__rshift__||__rsub_
_||__rtruediv__||__rxor__||__setattr__||__sizeof__||__str__||__sub__||__subclasshook__||__truediv__||__trunc__||__xor__||bit_length||conjug
ate||denominator||from_bytes||imag||numerator||real||to_bytes||
```

- 위의 목록중에서 __xxxx__ 형식으로 되어 있는 메소드는 연산자와 매칭이 되는 메소드입니다.
- 뒤장에서 자세히 목록을 설명하겠습니다. 우선 몇개만 소개하겠습니다.
- 위에 목록중 __add__는 우리가 사용하는 + 연산자와 매칭이 됩니다.
- + 연산자를 사용하면 __add__가 실행이 됩니다.
- 다음 장은 연산자 중복 메소드 목록을 정리했습니다.

```
x = 10; y = 20
```

```
x + y
```

```
30
```

```
x.__add__(y)
```

```
30
```


Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 연산자 중복 메소드 목록

- 아래는 연산자 중복 메소드 목록입니다.

메소드	기호
__add__	+
__sub__	-
__mul__	*
__truediv__	/
__floordiv__	//
__mod__	%
__pow__	**
__gt__	>
__ge__	>=
__lt__	<
__le__	<=
__eq__	==
__ne__	!=

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 컨테이너 자료형의 연산자 중복에 대해서 알아보겠습니다.

- 컨테이너 자료형은 우리가 알고 있는 str, list, tuple, dict, set입니다.
- 컨테이너 자료형을 dir() 함수를 사용해 속성(메소드)를 살펴보면 __xx__ 형식의 속성들이 존재합니다.

```
for x in dir(list):
    print (x, end= "||")
```

```
__add__||__class__||__contains__||__delattr__||__delitem__||__dir__||__doc__||__eq__||__format__||__ge__||__getattribute__||__getitem__||__
gt__||__hash__||__iadd__||__imul__||__init__||__init_subclass__||__iter__||__le__||__len__||__lt__||__mul__||__ne__||__new__||__reduce__||_
_reduce_ex__||__repr__||__reversed__||__rmul__||__setattr__||__setitem__||__sizeof__||__str__||__subclasshook__||append||clear||copy||count
||extend||index||insert||pop||remove||reverse||sort||
```

- __add__, __mul__, __contains__, __len__, __getitem__, __setitem__, delitem__, __iter__은 해당 객체에 연산자 및 len() 함수가 실행될 때 해당 메소드가 실행이 됩니다.
- 간단한 예를 들어보겠습니다. “Python” + “ Seo” 라고 실행하면 결과는 “Python Seo”라고 출력될 것입니다.
  - 이제는 말할 수 있습니다. 문자열 객체의 __add__ 메소드가 실행된 것입니다.
    - "Python".__add__(" Seo")
- 우리가 객체의 능력이라고 생각을 했던 기능들이 사실은 모두 정의된 속성들이 실행이 된 것입니다.
- str, list, tuple은 + 연산자가 되고, set, dict은 + 연산자가 안되는 이유는 뭘까요?
  - __add__ 속성이 정의 되어 있지 않습니다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 컨테이너 연산자 중복의 목록을 정리합시다.

- 아래는 컨테이너형 데이터 유형의 연산자 중복 표입니다.
  - 표 안의 예제를 실행하면 해당 열의 이름의 메소드가 실행이 됩니다.

	__add__	__mul__	__len__	__getitem__	__setitem__	__delitem__	__iter__
str	'apple' + 'banana'	'apple' * 3	len('apple')	'apple' [1]	X	X	O
list	[1,2] + [3,4]	[1,2] * 3	len([1,2])	a = [1,2] a[1]	a[1] = 5	del a[1]	O
tuple	(1,2) + (3,4)	(1,2) * 3	len((1,2))	b = (1,2) b[1]	X	X	O
dict	X	X	len({'a':1, 'b':2})	c ={'a':1, 'b':2} c['a']	c['a'] = 3	del c['a']	O
set	X	X	len({1,2,3})	X	X	X	O

- __iter__ 메소드는 반복자를 생성해주는 메소드입니다. 뒤에 반복자만 따로 공부하겠습니다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 반복자에 대해서 알아보겠습니다.

- 반복자(Iterator)는 next() 함수를 호출할 때 데이터를 순차적으로 한 번에 하나씩 넘겨주는 자료형입니다.
- 반복자는 Lazy Evaluation을 하기 때문에 출력 값이 필요한 시점에서 값을 계산합니다.
  - 앞에서 with open() as f 를 기억하신다면 대용량 파일을 한번에 메모리에 올리지 않고 반복자를 사용하여 출력 값이 필요한 시점에서만 연산을 하는 것을 확인하였습니다. 이런 특징이 Lazy Evaluation의 특징입니다.
- 어떤 객체에 대한 반복자 객체는 iter() 함수로 생성할 수 있습니다. 이 함수에 반응하는 메소드가 바로 __iter__. __iter__ 메소드가 정의되어 있어야 iter() 함수를 사용할 수 있습니다.
- 반복자 객체는 next() 함수에 의해서 호출되는 __next__ 메소드를 가집니다. next() 함수는 순차적으로 데이터를 출력합니다. next() 함수로 더 이상 출력할 데이터가 없을 경우 StopIteration 예외가 발생됩니다.
- “for x in 객체”에서 객체는 __iter__ 메소드가 존재하고 for문은 자동으로 __next__를 호출합니다. 그리고 더 이상 데이터가 없다면 next()가 StopIteration 예외를 발생하고 이를 for문이 처리해 줍니다.
- 반복자는 데이터를 필요한 시점에서 데이터를 준비해서 반환하기 때문에 메모리와 계산 시간을 절약할 수 있습니다. 데이터의 크기가 매우 클수록 더 효과적입니다. (큰 파일 읽기 예제)

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 발생자에 대해서 알아보겠습니다.

- 사용자 정의 함수에 return 대신 yield 키워드가 있다면 그 함수는 제너레이터 함수가 됩니다.
- yield 문은 return 문과 비슷해 보이지만 실행 상태를 보존한 상태에서 복귀합니다. 아래 예제를 보겠습니다.

```
def generator_func():
    n = 0
    while n <= 2:
        yield n
        n += 1
```

함수 내부에 yield 키워드가 사용되면 제너레이터 함수가 됩니다.

```
generator_1 = generator_func()
```

함수를 실행하면 제너레이터 객체를 반환

```
generator_1.__next__()
```

```
0
```

반복자처럼 __next__ 메소드를 사용하여
yield 문이 실행될 때까지의 결과를 확인할 수 있다.

```
generator_1.__next__()
```

```
1
```

```
generator_1.__next__()
```

```
2
```

```
generator_1.__next__()
```

다음 값이 존재하지 않을 경우 반복자처럼
StopIteration가 발생합니다.

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-37-2f2c330880a3> in <module>
----> 1 generator_1.__next__()

StopIteration:
```

```
def normal_func():
    n = 0
    while n <= 2:
        return n
        n += 1
```

```
normal_func()
```

```
0
```

```
normal_func()
```

```
0
```


Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 발생자 함수는 일반 함수처럼 종료되는 것이 아닙니다.

- 제너레이터(발생자) 함수는 일반 함수처럼 return 키워드를 만나서 종료되는 것이 아니라, yield 키워드를 만나는 순간 여기까지 계산된 결과를 반환하고 잠시 보류? 이정도로 생각하면 됩니다.
- 잠시라고 말했으니 다음번은 언제 실행하는지는 위의 예제를 보면 __next__를 호출하면 또 다시 yield까지만 실행되고 또 잠시 보류입니다.
- 꼭 기억하실 내용은 yield를 만나 여기까지 결과를 반환하고 중단하고 다시 실행되어도 실행 상태와 지역 변수들을 보존하고 있습니다.

```
def picalc():
    n = 1
    denom = 1
    nom = 1
    acc = 0.0
    while True:
        acc += nom / denom
        yield 2 * acc
        nom *= n
        denom *= (2* n  + 1)
        n += 1
```

```
a = picalc()
```

```
next(a)
```

```
2.0
```

코드 출처 - 파이썬 3 바이블 (이강성)

Copyright FASTCAMPUS Corp. All Rights Reserved

I 발생자 내장에 대해서 알아보겠습니다.

- 리스트 내장(list comprehension) 형식에서 대괄호를 소괄호로 변경하여 실행하면 Generator comprehension이 실행 됩니다.
- [x for x in range(100000000)]를 (x for x in range(100000000)으로 변경하면 발생자 내장이 됩니다.
- 마지막으로 발생자 내장 및 발생자의 장점을 예제를 통해서 설명하고 마치겠습니다.

```
[x for x in range(100000000)]
```

```
a = (x for x in range(100000000))
```

```
sum(a)
```

```
4999999950000000
```

  - 리스트 내장으로 해당 표현식의 리스트를 생성하면 메모리 공간에 데이터를 만들어야 하기 때문에 메모리 사용량이 증가합니다.
  - 하지만 발생자를 사용하면 전체 데이터를 생성하지 않고 필요할 때마다 실행하기 때문에 메모리가 증가되지 않습니다.

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 코루틴이란 무엇인가?

- 코루틴(Coroutine)이란 함수 실행되면서 특정 위치에서 함수의 중단과 실행을 가능하게 만드는 다중 진입점이 있는 함수입니다.
  - 루틴의 국어사전 의미는 “컴퓨터 특정한 작업을 실행하기 위한 일련의 명령. 프로그램의 일부 혹은 전부를 이르는 경우에 쓴다.”
  - yield 키워드와 next() 함수에 의해서 함수의 실행과 중단이 됩니다.
  - 서브 루틴은 메인 루틴에서 호출되는 함수들을 의미하고 서브 루틴이 종료되면 메인 루틴으로 복귀합니다.
- 코루틴을 사용하면 동시에 많은 함수를 실행하는 것처럼 보이게 할 수 있고, 제너레이터를 사용하기 때문에 메모리가 적게 사용하여 구현할 수 있습니다.

Copyright FASTCAMPUS Corp. All Rights Reserved

I 코루틴이란 무엇인가?

- yield 키워드를 사용한 아래 예제는 제너레이터 함수처럼 보이지만 다른 점이 있습니다.
- 외부로부터 데이터를 send() 메소드로 데이터를 코루틴 함수에 전달하는 기능을 사용할 수 있습니다.
  - 코루틴 함수에 전달받기 위해서는 변수 = (yield) 혹은 변수 = (yield average) 식으로 사용합니다.
  - 데이터를 입력받는 기능과 외부로 반환하는 기능 두 기능을 가진다

```
def co_avg():
    total = 0.0
    count = 0
    average = None
    while True:
        term = (yield average)
        total += term
        count += 1
        average = total / count
```

```
corotine_avg = co_avg()
```

코루틴 객체 생성

```
corotine_avg.__next__()
```

함수를 실행하고 첫번째 yield 문까지 진행

```
corotine_avg.send(10)
10.0
```

term 변수에 10을 전달하고 while 한번 실행후 yield average 값을 반환하고 정지

```
corotine_avg.send(20)
15.0
```

term에 20을 전달하고 다시 while 한번 실행
이때 이전 값들이 남아 있기 때문에 total의 값은 10 + 20이 되고 average 변수는 평균을 계산한다.
그리고 yield average으로 값을 반환하고 정지

Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 클로저에 대해서 알아봅시다.

- 일반적인 우리가 배운 사용자 함수(서브루틴)은 그 함수의 실행이 종료가 되면 그 함수에서 사용된 모든 변수와 객체의 정보가 사라지는데 클로저(Clousures)는 함수의 실행이 완전히 종료가 되어도 함수에 대한 정보를 계속 가지고 있어서 함수 밖에서 함수 내부에 정의된 변수를 사용할 수 있습니다.
- nonlocal 문은 함수 내부에서 정의된 함수안에서 바깥 쪽 함수안에 있는 지역 변수를 사용하기 위해서 사용되는 키워드입니다.

  n은 click() 함수 내부에 정의된 지역변수이고, count_click() 함수는 그 함수 안에 정의된 중첩 함수입니다. 이 함수에서 n에 접근하기 위해서는 nonlocal 키워드를 사용합니다.
- 옆의 예제처럼 클로저는 함수가 종료가 되어도 n에 대한 정보가 남아 있어 a1()을 실행할 때마다 n의 값이 1씩 증가되는 것을 확인할 수 있습니다.
- a1과 a2가 각각 독립적인 공간으로 동작하는 것을 옆의 예제에서 확인할수 있습니다.

```
def click():
    n = 0
    def count_click():
        nonlocal n
        n += 1
        return n

    return count_click
```

```
a1 = click()
```

```
a1()
```

```
1
```

```
a1()
```

```
2
```

```
a1()
```

```
3
```

```
a1()
```

```
4
```

```
a2 = click()
```

```
a2()
```

```
1
```


Life Changing Education

Copyright FASTCAMPUS Corp. All Rights Reserved

I 데코레이터에 대해서 알아보겠습니다.

- 데코레이터(decorator)는 기존 함수에 새로운 기능을 추가해주는 함수를 의미합니다.
- 기존 함수의 기능에 새롭게 추가하고 싶은 기능이 있다면 기존 함수를 수정하지 않고 추가할 수 있습니다.
  - 데코레이터라는 단어도 이런 의미에서 새 기능을 추가하여 장식한다 뜻으로 이해하기 바랍니다.

```
def click(func):
    n = 0
    def count_click():
        nonlocal n
        n += 1
        func(n)
        return n

    return count_click
```

```
def hello(cnt):
    print ("본 사이트에 {}번 방문을 환영합니다".format(cnt))
```

```
a3 = click(hello)
```

```
a3()
```

```
본 사이트에 1번 방문을 환영합니다

1
```

장식자 문법(@)을 사용해서 동일한 효과를 만들수 있다.

```
@click
def hello2(cnt):
    print ("Hello2 함수입니다. {}번째 방문을 환영합니다.".format(cnt))
```

```
hello2()
```

```
Hello2 함수입니다. 1번째 방문을 환영합니다.

1
```


Life Changing Education
Fast campus
Copyright FASTCAMPUS Corp. All Rights Reserved

I 정리

- 클래스
- 멤버 변수 접근 방법
- 연산자 중복 메소드, 컨테이너 중복 메소드
- 반복자
- 생성자
- 코루틴
- 클로저
- 데코레이터

Copyright FASTCAMPUS Corp. All Rights Reserved

감사합니다

FAST CAMPUS
ONLINE
서찬웅 강사.

Copyright FASTCAMPUS Corp. All Rights Reserved