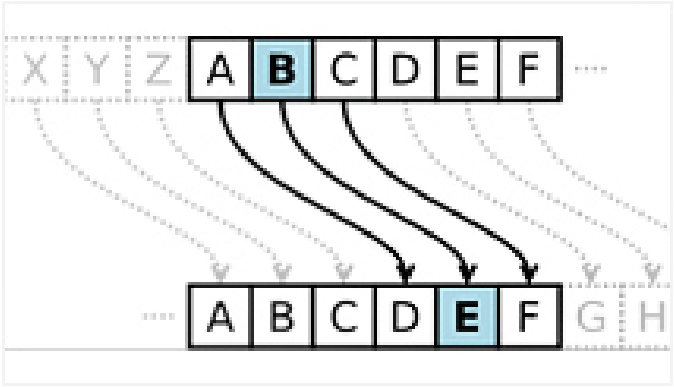
Lecture #09 | 객체지향 프로그래밍 소개

SE213 프로그래밍 (2019)

Written by Mingyu Cho
Edited by Donghoon Shin

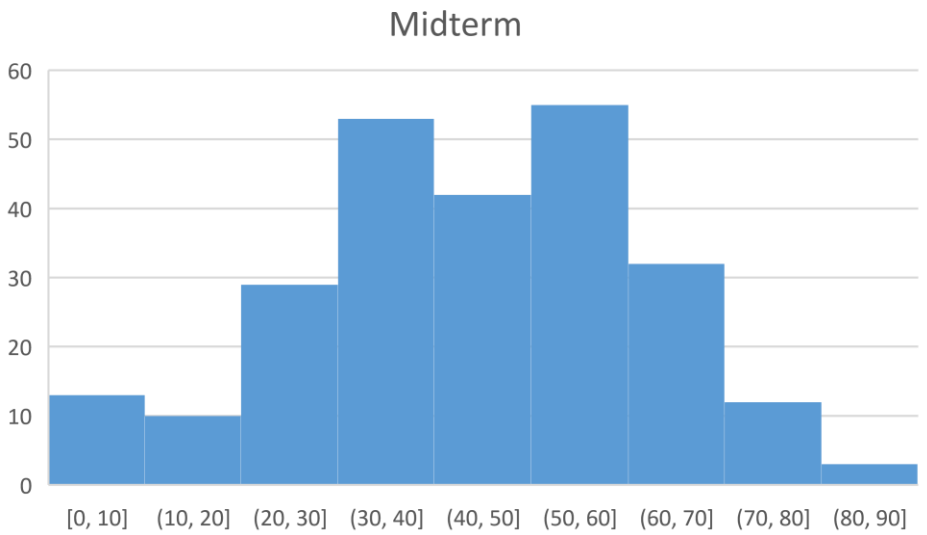
2nd assignment

- 과제 2-1: 순환하는 소수
  - circular(n) → (리스트) 모든 순환된 숫자
  - prime(n) → (bool) n의 소수 여부
  - Circular_prime(n) → (bool) n으로 생성한 순환 숫자가 모두 소수인지 여부
- 과제 2-2: 야구 게임
  - set_number(seed=-1) → (리스트) 0-9 사이의 서로다른 3개의 숫자를 추출하여 반환
  - input_check(input_data) → (리스트) 입력받은 문자열을 인자로 받아서 3개의 숫자로 변환 뒤, 검증 후, 리스트로 반환
  - check_count(rnd, usr) → (문자열) 사용자가 입력한 값과 설정값을 비교하여 Strike, ball 판정
- 과제 2-3: 카이사르 암호
  - find_key(plaintext, ciphertext) → (int) 키 값
  - decrypt(ciphertext, key)→ (문자열) 암호문을 key를 통해 복호화하고 해당 값을 반환 (if문을 26번 사용하지 말것)



Midterm result

- Result
  - Average : 45.66
  - High score : 86
  - Median: 47



- Claim (~5/7)
  - 1: 정승화(shjung at dgist.ac.kr)
  - 2(1,2): 송봉섭 (doorebong at dgist.ac.kr)
  - 2(3,4): 이현태 (freed0m at dgist.ac.kr)
  - 2(5,6): 김민지 (kkkmz at dgist.ac.kr)
  - 3: 박형석 (hyungseok at dgist.ac.kr)
  - 4: 이경민 (rud557 at dgist.ac.kr)
  - 5: 원유민 (wym7787 at dgist.ac.kr)
  - 6: 배진욱 (jinwook.bae at dgist.ac.kr)

지난 시간에 다룬 내용

- 파일 입출력
  - 바이너리 파일 입출력
- 자료구조
  - 집합(`set`)
  - 사전(`dict`)

오늘 다룰 내용

- 객체지향 프로그래밍 (Object-Oriented Programming)
  - 객체지향 프로그래밍 소개
  - 파이썬과 객체지향 프로그래밍
  - 클래스 정의
  - 클래스 변수/함수의 사용
  - 인스턴스 변수/함수의 사용

오늘 다룰 내용

- 객체지향 프로그래밍 (Object-Oriented Programming)
  - 객체지향 프로그래밍 소개
  - 파이썬과 객치지향 프로그래밍
  - 클래스 정의
  - 클래스 변수/함수의 사용
  - 인스턴스 변수/함수의 사용

python에 대해서 지금까지 배운 것

- 참고: 컴퓨터에서 가장 중요한 것은 '데이터'와 '연산'
- python
  - 기본자료형 (예: `int, float, str, bool, …`)
  - 변수
  - 자료 구조 (예: `list, tuple, dict, set`) → 여러 값을 저장
  - 연산자 (& 우선순위)
  - 제어 흐름(control flow)
    - 조건문(`if/elif/else`) → 데이터(조건)에 따라 다른 연산을 수행
    - 반복문(`for, while`) → 여러 데이터에 대해서 (같은/유사한) 연산을 수행
    - 함수 → 데이터(인자)를 이용하여 연산을 수행하고, 그 결과 데이터를 생성(반환값)
  - **클래스** → 데이터 + 연산(함수)의 좀 더 강한 결합

객체지향 프로그래밍이란?

- 절차적 프로그래밍(Procedural Programming)
  - 프로시저(Procedure: 프로그램 언어에 따라 서브루틴, 함수, 메소드 등으로도 불림) 호출을 사용하는 프로그래밍 패러다임
  - 모듈화를 통해 작업하고자 하는 독립적인 범위(scope)를 만들고, 인자와 반환값을 통해 프로시저와 데이터를 교환
  - 어떠한 '작업'을 수행하는 것에 초점을 맞추어 프로그램을 개발
- 객체지향 프로그래밍 (Object-Oriented Programming, OOP)
  - 데이터와 프로시저를 결합한 '객체'라는 개념에 기반한 프로그래밍 패러다임
  - 작업 대상이 되는 '자료'(Data)와 '처리방법'을 동시에 설계
- 참고: 객체지향과 다른 '함수형 프로그래밍'도 있음

왜 객체지향 프로그래밍을 사용하는가?

- 더 높은 수준의 추상화(abstract)를 제공하기 방향으로 프로그래밍 언어가 진화
  - Machine language (i.e., binary code)
  - Assembly language
  - Procedural languages (예: C, Fotran)
  - Object-oriented languages (예: C++, Java, python)
  - Declarative (or functional) languages (예: Lisp, Haskell, ML, prolog, Scala)
- 목적
  - **유지보수의 용이성**: 독립적인 코드의 구성이 쉬워, 유지보수가 용이함
  - **코드 재사용성 향상**: 코드, 디자인 방법을 재사용할 수 있는 언어 기능을 제공
  - **확장성 제공**: 코드에 대한 요구사항이 변경됐을 때, 기능의 추가/변경을 용이하게 함

객체지향 프로그래밍을 지원하는 언어

- 객체 지향을 지원하는 언어 (지원 정도에는 차이가 있음)
  - smalltalk: 최초의 OOP 언어
  - C++: C언어에 객체지향 개념을 지원
  - Java
  - python
  - scala (다중 패러다임 언어: 함수형 & 객체지향 언어)
  - ruby
  - C#
  - Objective-C, …
- 현재 널리 사용되는 대부분의 언어에서 객체지향 프로그래밍의 개념을 지원함

python의 객체지향 프로그래밍 지원

- python은 객체지향을 염두에 두고 설계된 언어
  - 간단한 문법으로 대부분의 객체지향 개념을 지원
  - 엄격한 문법으로 객체 지향을 지원하는 C++, Java 등과는 차이를 보임
- python에서는 모든 것이 객체: 기본자료형, 자료구조, 함수 등
  - 기본자료형: `int, float, str, bool`
  - 자료구조: `list, tuple, dict, set`
  - 함수
- 클래스(class) v.s. 인스턴스(instance) 혹은 객체(object)
  - 클래스: 자료형
    - 예: `int, float`
  - 인스턴스(instance)/객체(object): 특정 클래스에 값이 주어진 실체
    - 예: `42, 1024, 3.14`

클래스의 구성 요소

- 클래스의 속성(attributes)
  - 데이터 속성(data attributes): 변수
    - member variable, field, attributes, properties, characteristics
  - 메소드(methods): 함수
    - member functions, methods, responsibilities
- 클래스 & 인스턴스 변수 (class & instance variable)
  - 클래스 변수: 한 종류의 클래스 전체에 공유되는 변수
  - 인스턴스 변수: 클래스의 인스턴스별로 독립적인 변수
- 클래스 & 인스턴스 함수 (class & instance function)
  - 클래스 함수: 한 종류의 클래스 전체에 공유되는 함수로 인스턴스 변수를 접근할 수 없음
  - 인스턴스 함수*: 클래스의 인스턴스 변수를 접근할 수 있는 함수

* 첫 번째 인자의 이름은 어떤 것으로 해도 되나, 관례적으로 `self`를 사용함

type(), id() : 자료형과 식별자를 반환하는 함수

- type(): 어떠한 값 혹은 변수에 저장된 값의 자료형을 반환
- id(): 객체의 식별자*를 반환

```
d1 = 42
t1 = [42, 1024, 23]
t2 = t1
t3 = t1.copy()
print(type(d1))
print(type(t1))
print(type(t2))
print(type(t3))
print(id(t1), id(t2), id(t3))
```

```
<class 'int'>
<class 'list'>
<class 'list'>
<class 'list'>
4554987528 4554987528 4552072968
```

* 객체들을 구별하기 위한 고유한(unique) 값이며, 값 자체는 중요하지 않음

is, is not, ==, !=: 식별자 혹은 객체의 값의 동일 여부 확인

- is, is not: 동일한 객체(인스턴스)를 가르키는지, 즉 식별자로 동일 여부 판단
- ==, !=: 객체가 저장하고 있는 내용이 동일한지를 판단
- (Advanced) 사용자가 만든 클래스의 경우*
  - 특수한 함수(__eq__())를 정의해서 인자로 넘겨진 객체와 현재 객체의 동일 여부에 따라 True 혹은 False를 반환하도록 해야 함
    - 이 함수의 반환값이 == 연산자의 결과로 계산됨
  - 이 함수를 만들지 않았을 경우에는, is연산자의 결과가 사용됨

```
t1 = [42, 1024, 23]
t2 = t1
t3 = t1.copy()
print(t1 == t2, t1 is t2)
print(t1 == t3, t1 is t3)
```

```
True True
True False
```

* 실습에 예제가 있음

클래스의 정의 문법

- 클래스 정의 문법

```
class ClassName:
    <statement_1>
    <statement_2>
    ...
    <statement_n>
```

- 클래스 안에는 임의의 명령문을 사용할 수 있으나, 일반적으로 함수의 정의를 사용함
- 클래스 이름에 관한 규칙
  - 변수/함수의 이름의 규칙과 동일: 알파벳, 숫자, _로 구성되고, 숫자로 시작할 수는 없음
  - 일반적으로 사용자가 만든 클래스는 대문자로 시작함

클래스가 제공하는 연산: 속성 참조와 객체화

- 속성 참조(attribute reference): 클래스에 속한 변수 혹은 함수를 의미함
  - 클래스 변수/함수

    ClassName.variable_name

    ClassName.function_name()
  - 인스턴스 변수/함수

    instance_name.variable_name

    instance_name.function_name()
- 객체화(instantiation): 클래스 이름을 함수의 형태로 사용하여, 객체를 만듦

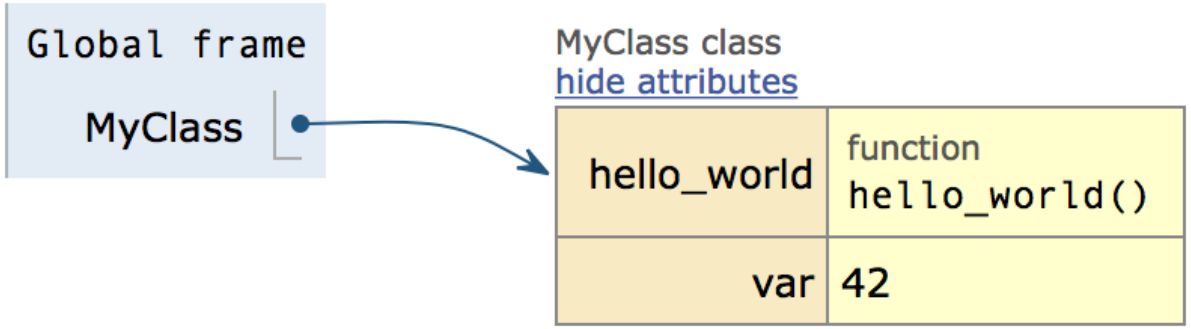
  ClassName()

클래스 변수/함수의 정의와 사용

- 클래스가 선언되면, 그 정보(어떤 변수, 함수를 가지는지)가 전역 네임스페이스에 저장됨
- 클래스 변수/함수의 사용*: 변수/함수 이름 앞에 클래스 이름을 사용
  - 클래스 변수: *ClassName.variable*
  - 클래스 함수: *ClassName.function()*

```
class MyClass:
    var = 42
    def hello_world():
        return 'Hello, world!'

print(MyClass.var)
MyClass.var = 23
print(MyClass.var)
print(MyClass.hello_world())
```

```
42
23
Hello, world!
```



* 클래스의 메소드에서도 클래스 이름을 사용해야 함

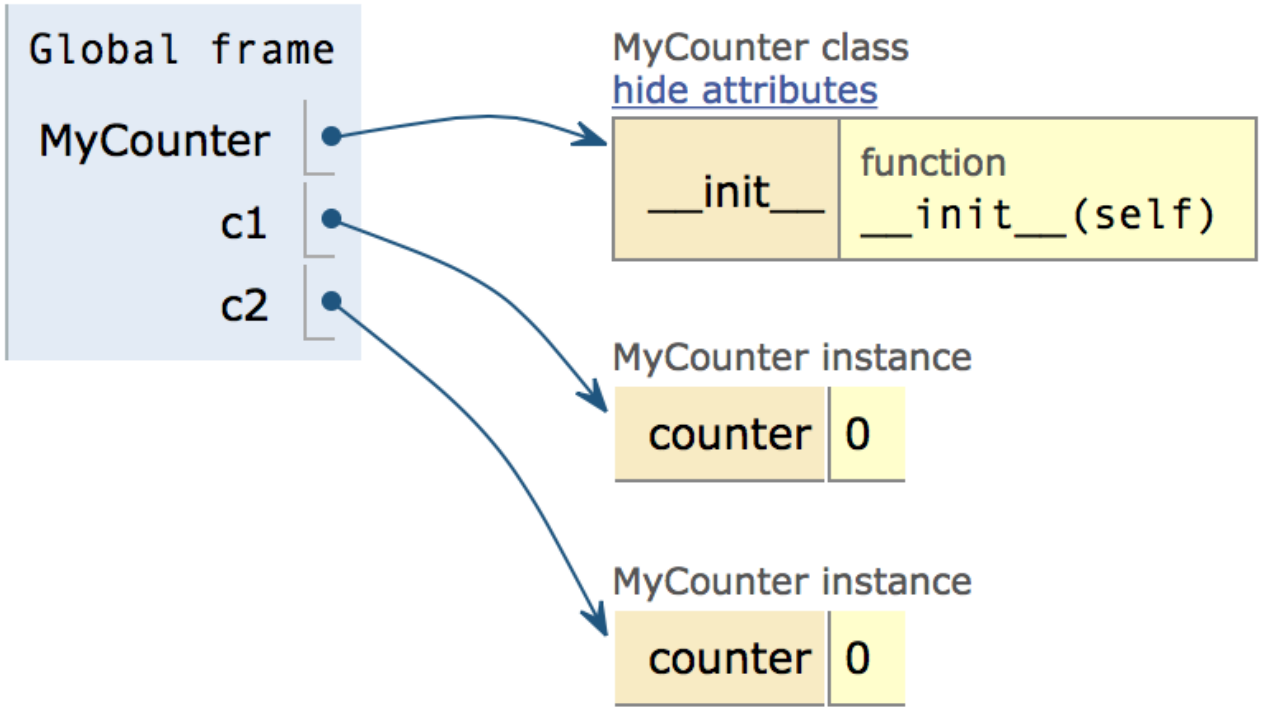
클래스의 인스턴스화: 세부적인 사항

- 인스턴스화 할 때, python이 하는 작업
  1. 인스턴스를 저장할 공간(일종의 네임스페이스)을 만들고
  2. `__init__(self, ...)` 함수를 호출
     a. 첫 번째 인자(`self`)는 1번에서 만든 공간을 가르킴
     b. 다른 인자들은 함수 호출과 같은 규칙으로 `__init__()`의 매개변수에 전달됨
  3. 새로 만들어진 공간(즉 `self`가 가르키는 공간)이 반환값처럼 사용됨
     → 변수에 대입 혹은 표현식의 값으로 사용됨
- `__init__()` 함수의 특성
  – 이름은 바꿀 수 없음
  – 반환값을 가질 수 없음
  – 참조값을 전달받는 첫 번째 매개 변수의 이름은 관례적으로 `self`를 사용함

예시: 클래스의 인스턴스화

- 인스턴스 변수: 인스턴스마다 독립적으로 가지고 있는 변수
  - 클래스를 인스턴스화할 때 만들어진 공간에 저장됨
  - self를 사용하여 __init__() 혹은 다른 인스턴스 함수에서 접근 가능

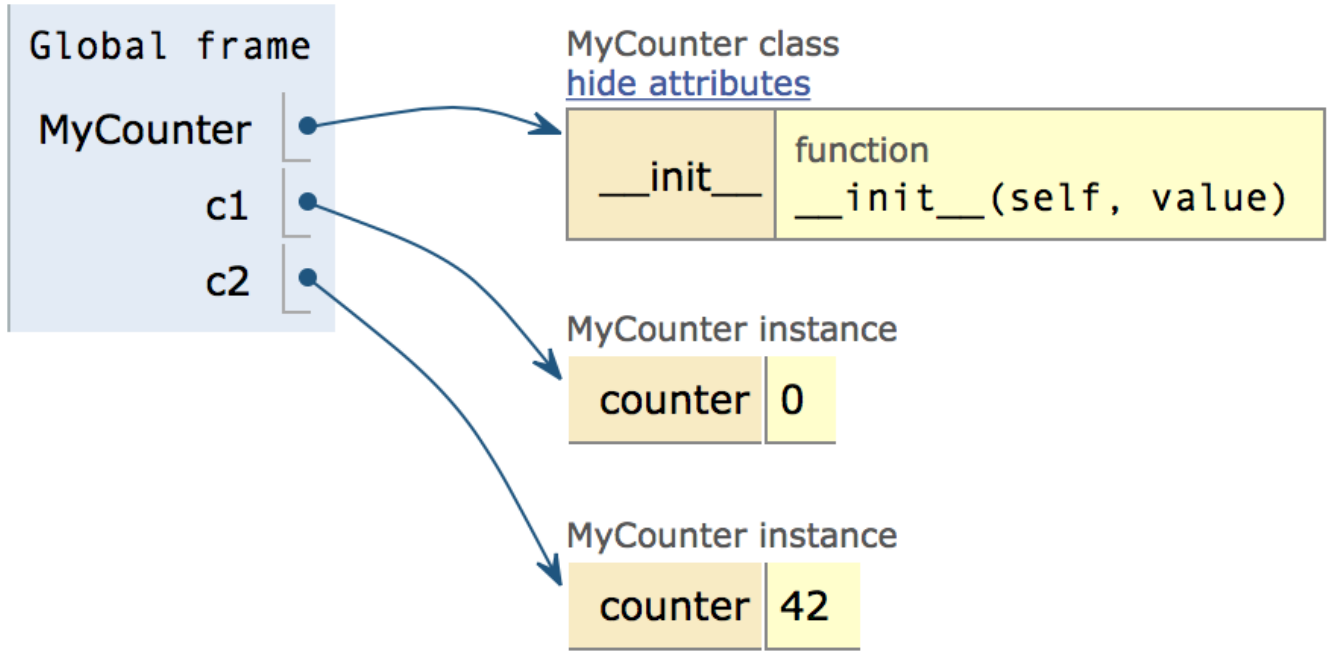
```
class MyCounter:
    def __init__(self):
        self.counter = 0


c1 = MyCounter()
c2 = MyCounter()
```



__init__() 함수에서 인자 사용하기

- 다른 함수와 마찬가지로 매개변수, 디폴트 인자 등을 가질 수 있음
- 참고: 첫 번째 인자인 self는 python이 자동으로 채워주기 때문에, 프로그램에서 따로 제공해주지 않아도 됨

```
class MyCounter:
    def __init__(self, value=0):
        self.counter = value

c1 = MyCounter()
c2 = MyCounter(42)
```



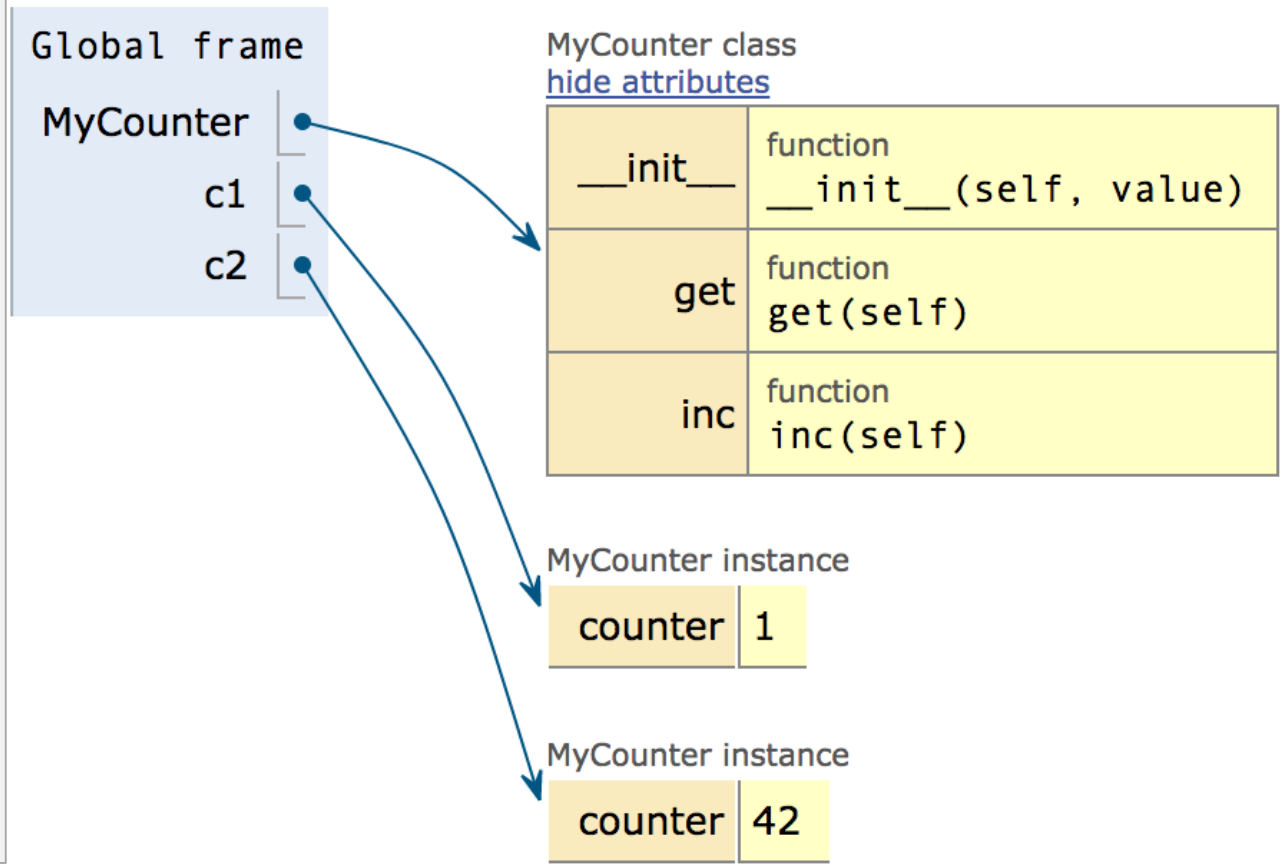
인스턴스 함수/변수의 사용

- 인스턴스 함수/변수 사용: 객체 뒤에 '.'과 함수/변수 이름을 사용하면 됨, 예:
  - `instance_variable.function()`
  - `instance_variable.variable`
- 인스턴스 함수
  - 첫 번째 인자(관례적으로 `self`라는 이름을 사용)로 객체의 참조값이 전달됨 (python이 자동으로 전달하므로 프로그래머가 신경쓰지 않아도 됨)
  - 인스턴스 함수의 내부에서 아래와 같이 인스턴스 변수 혹은 다른 인스턴스 함수를 사용할 수 있음
    - *`self.function()`*
    - *`self.variable`*

예시: 인스턴스 변수/함수

```
class MyCounter:
    def __init__(self, value=0):
        self.counter = value

    def get(self):
        return self.counter

    def inc(self):
        self.counter += 1

c1 = MyCounter()
c2 = MyCounter(42)
print(c1.counter, c2.counter)
c1.inc()
print(c1.get(), c2.get())
```

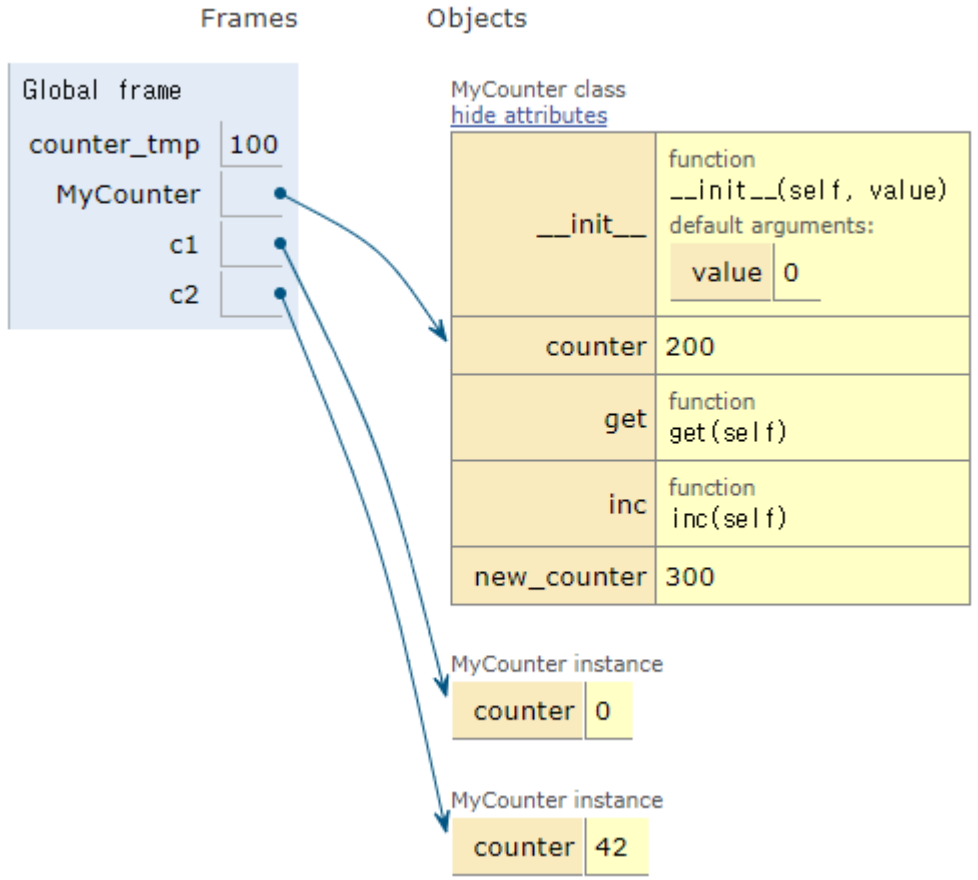
```
0 42
1 42
```



Namespace 관계

```
counter_tmp = 100
class MyCounter:
    counter = 200
    new_counte = 300

    def __init__(self, value=0):
        self.counter = value

    def get(self):
        return self.counter

    def inc(self):
        self.counter += 1

c1 = MyCounter()
c2 = MyCounter(42)
print(c1.counter)
print(c1.new_counter)
print(MyCounter.counter)
```

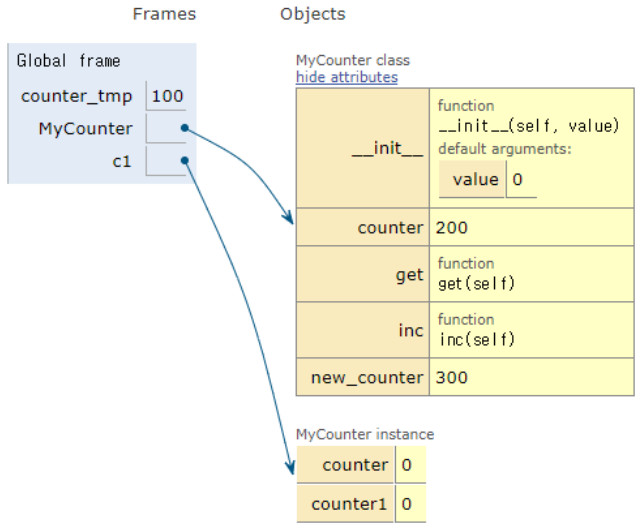
```
0
300
200
```



dir():사용하는 이름 목록 반환 함수

```
counter_tmp = 100
class MyCounter:
    counter = 200
    new_counte = 300

    def __init__(self, value=0):
        self.counter = value
        self.counter1 = value

    def get(self):
        return self.counter

    def inc(self):
        self.counter += 1

c1 = MyCounter()
print(dir(MyCounter))
print(dir(c1))
print(c1.__dict__)
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__return__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'counter', 'get', 'inc', 'new_counter']
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__le__',
'__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__return__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'counter', 'counter1', 'get', 'inc', 'new_counter']
{'counter': 0, 'counter1': 0}
```



예시: 클래스 디자인

- 수업 관리
  - 학생
  - 과목 명
  - 수강생출력
  - 특정학생점수
  - 평균
  - 정렬

- 학생
  - 이름
  - 학번
  - 과제점수
  - 중간고사 점수
  - 기말고사 점수
- 반영비율
- 학점 계산
- 비교연산
- …

```
class Student:
    rate = (40, 30, 30)
    def __init__(self, name, id,
                 homework=0, mid=0, final=0):
        self.name = name
        self.id = id
        self.homework = homework
        self.mid = mid
        self.final = final

    def __repr__(self):
        return self.name

    def __eq__(self, obj):
        if self.id == obj.id:
            return True
        else:
            return False
```

예시: 클래스 디자인

- 수업 관리
  - 학생
  - 과목 명
  - 수강생출력
  - 특정학생점수
  - 평균
  - 정렬

- 학생
  - 이름
  - 학번
  - 과제점수
  - 중간고사 점수
  - 기말고사 점수
- 반영비율
- 학점 계산
- 비교연산
- …

```
class ClassManager:
    """수업관리 class"""

    def __init__(self):
        pass

    def sort(self):
        pass

    def get_grade(self, name):
        pass

    def print_all_students(self):
        pass
```

오늘 다루지 않은 내용

- 객체지향 프로그래밍의 여러 기법
  - 은닉화(encapsulation)
  - 상속(inheritance)
  - 다형성(polymorphism)
  - 추상화(abstraction)
- 왜 객체지향을 써야 하는가에 대한 고찰
- 디자인 패턴
- 다양한 예제
- …
- 참고: 2학기에 객체지향 프로그래밍 과목 개설
  - C++를 사용하여 진행 예정

읽을 거리

- 클래스에 대한 추가적인 설명
  - http://www.flowdas.com/thinkpython/15-classes-and-objects/
- Advanced: (python을 이용한) 객체지향 프로그래밍에 대한 소개
  - 파이썬 - OOP (한국어)
  - python3 OOP (영어)

ANY QUESTIONS?