

빠르게 활용하는 파이썬3 프로그래밍

클래스

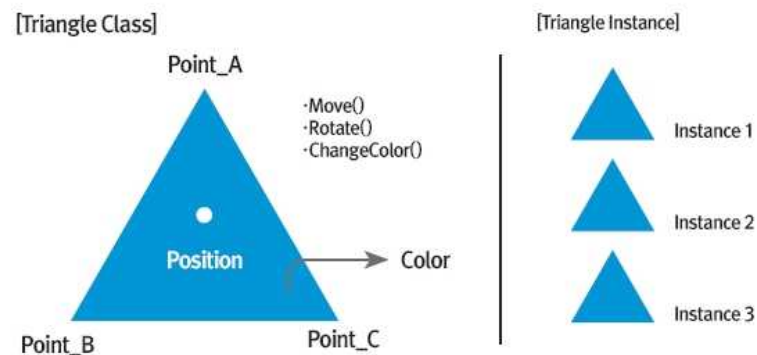


목 차

- 클래스란?
- 클래스 기본
- 이름 공간
- 클래스와 인스턴스 관계
- 생성자와 소멸자
- 메서드 확장
- 연산자 중복
- 상속

클래스란?

- 데이터와 데이터를 변형하는 함수를 같은 공간으로 작성



- 메서드(Method)
- 인스턴스(Instance)
- 정보 은닉(Information Hiding)
- 추상화(Abstraction)
 - 부모클래스
 - 자식 클래스

클래스 기본

- 클래스와 인스턴스

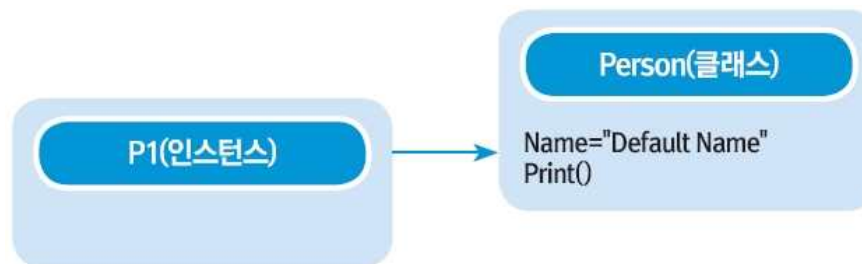
```
class Person: # 클래스 정의
    Name = "Default Name" # 멤버 변수

    def Print(self): # 멤버 메소드
        print("My Name is {}".format(self.Name))

p1 = Person() # 인스턴스 객체 생성

p1.Print() # 멤버 변수값을 출력
```

- 클래스와 인스턴스 이름 공간

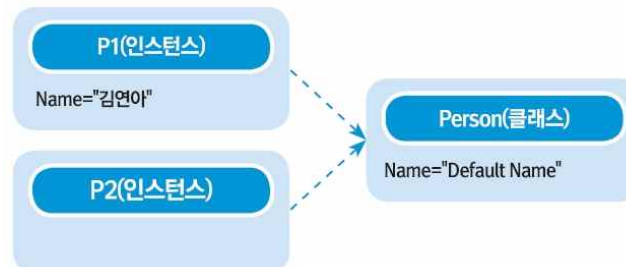


이름 공간

- 인스턴스객체를 통하여 변수나 함수의 이름을 찾는경우 검색 순서
 - 인스턴스 객체 영역 -> 클래스 객체 영역 -> 전역 영역

```
->>> class Person:      # 클래스 정의
    name = "Default Name"

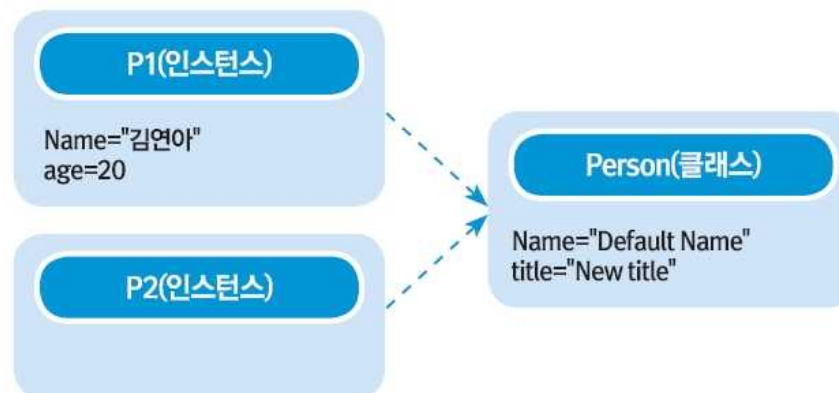
>>> p1 = Person()      # 인스턴스 객체 생성
>>> p2 = Person()
>>> print("p1's name: ", p1.name) # 각 객체의 name 속성 출력
p1's name: Default Name
>>> print("p2's name: ", p2.name)
p2's name: Default Name
>>> p1.name = "김연아"  # p1 인스턴스의 'name' 속성을 변경
>>> print("p1's name: ", p1.name)
p1's name: 김연아
>>> print("p2's name: ", p2.name)
p2's name: Default Name
```



이름 공간 – con'd

- 클래스와 인스턴스에 멤버 데이터 추가
 - 예제 코드

```
>>> Person.title = "New title" # 클래스 객체에 새로운 멤버 변수 title 추가
>>> p1.age = 20 # p1 객체에만 age 멤버 변수를 추가
>>> print("p1's title: ", p1.age)
p1's title: 20
>>> print("p2's title: ", p2.age)
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    print("p2's title: ", p2.age)
AttributeError: 'Person' object has no attribute 'age'
```



클래스와 인스턴스 관계

- isinstance(인스턴스 객체, 클래스 객체)
 - 인스턴스 객체가 어떤 클래스로부터 생성되었는지 확인
 - 불린 형태로 결과 반환

- 예제 코드

```
>>> class Person:
    pass

>>> class Bird:
    pass

>>> class Student(Person):
    pass
```

- 클래스에 상속관계가 있는경우에도 자식 클래스의 인스턴스객체는 부모클래스의 인스턴스로 평가
- 클래스 객체 정의시 어떤 클래스를 상속 받지 않더라도 파이썬 버전 3 이후로는 암시적으로 오브젝트 클래스를 상속

```
>>> p, s = Person(), Student()
>>> print("p is instance of Person: ", isinstance(p, Person))
p is instance of Person: True
>>> print("s is instance of Person: ", isinstance(s, Person))
s is instance of Person: True
>>> print("p is instance of object: ", isinstance(p, object))
p is instance of object: True
>>> print("p is instance of Bird: ", isinstance(p, Bird))
p is instance of Bird: False
```

생성자와 소멸자

- 생성자

- 생성시 초기화 작업을 수행
- 인스턴스 객체가 생성될 때 자동으로 호출
- `__init__()`

- 소멸자

- 소멸시 종료 작업을 수행
- 인스턴스 객체의 참조 카운터가 '0'이 될 때 호출
- `__del__()`

- 예제 코드

```
class MyClass:
    def __init__(self, value): # 생성자 메소드
        self.Value = value
        print("Class is created! Value = ", value)

    def __del__(self): # 소멸자 메소드
        print("Class is deleted!")

def foo():
    d = MyClass(10) # 함수 foo 블록안에서만 인스턴스 객체 d가 존재

foo()

class is created! Value = 10
Class is deleted
```


연산자 중복

- 연산자 중복이란

- 사용자 정의 객체에서 필요한 연산자를 내장 타입과 형태와 동작이 유사하도록 재정의
- 연산자 중복을 위하여 두 개의 밑줄 문자가 앞뒤로 있는 메소드를 미리 정의함

- 예제 코드

```
class GString:
    def __init__(self, init = None):
        self.content = init

    def __sub__(self, str): # '-' 연산자 중복 정의
        for i in str:
            self.content = self.content.replace(i, '')
        return GString(self.content)

    def Remove(self, str):
        return self.__sub__(str)
```

연산자 중복 – con'd

- 수치 연산자

메소드	연산자	사용 예
<code>__add__(self, other)</code>	<code>+</code> (이항)	<code>A + B</code> , <code>A += B</code>
<code>__sub__(self, other)</code>	<code>-</code> (이항)	<code>A - B</code> , <code>A -= B</code>
<code>__mul__(self, other)</code>	<code>*</code>	<code>A * B</code> , <code>A *= B</code>
<code>__truediv__(self, other)</code>	<code>/</code>	<code>A / B</code> , <code>A /= B</code> (3 이상 지원, 그 이하는 버전에서는 <code>__div__</code> 가 사용)
<code>__floordiv__(self, other)</code>	<code>//</code>	<code>A // B</code> , <code>A //= B</code>
<code>__mod__(self, other)</code>	<code>%</code>	<code>A % B</code> , <code>A %= B</code>
<code>__divmod__(self, other)</code>	<code>divmod()</code>	<code>divmod(A, B)</code>
<code>__pow__(self, other[, modulo])</code>	<code>pow()</code> , <code>**</code>	<code>pow(A, B)</code> , <code>A ** B</code>
<code>__lshift__(self, other)</code>	<code><<</code>	<code>A << B</code> , <code>A <<= B</code>



상속

- 상속이란
 - 부모 클래스의 모든 속성(데이터, 메소드)를 자식 클래스로 물려줌
 - 클래스의 공통된 속성을 부모 클래스에 정의
 - 하위 클래스에서는 특화된 메소드와 데이터를 정의
- 장점
 - 각 클래스마다 동일한 코드가 작성되는 것을 방지
 - 부모 클래스에 공통된 속성을 두어 코드의 유지보수가 용이
 - 각 개별 클래스에 특화된 기능을 공통된 인터페이스로 접근 가능



상속

- 예제 코드

```
class Person:
    def __init__(self, name, phoneNumber):
        self.Name = name
        self.PhoneNumber = phoneNumber

class Student(Person):
    def __init__(self, name, phoneNumber, subject, studentID):
        self.Name = name
        self.PhoneNumber = phoneNumber
        self.Subject = subject
        self.StudentID = studentID
```

- 클래스 간의 관계 확인
 - 상속 관계인 두 클래스 간의 관계를 확인
 - `issubclass(자식 클래스, 부모 클래스)`

상속

- 다중 상속

- 2개 이상의 클래스를 상속받는 경우
- 두 클래스의 모든 속성(변수와 메소드)을 전달 받음

- 예제 코드

```
# -*- coding: cp949 -*-  
class Tiger:  
    def Jump(self):  
        print("호랑이처럼 멀리 점프하기")  
  
class Lion:  
    def Bite(self):  
        print("사자처럼 한입에 꿀꺽하기")  
  
class Liger(Tiger, Lion): # 다중 상속  
    def Play(self):  
        print("라이거만의 사육사와 재미있게 놀기")
```

상속

- 클래스 상속과 이름 공간

인스턴스 객체 영역

-> 클래스 객체간 상속을 통한 영역(자식 클래스 영역)

-> 부모 클래스 영역)

-> 전역 영역

- 예제 코드

```
class SuperClass: # 부모 클래스
```

```
    x = 10
```

```
    def printX(self):
```

```
        print(self.x)
```

```
class SubClass(SuperClass): # 자식 클래스
```

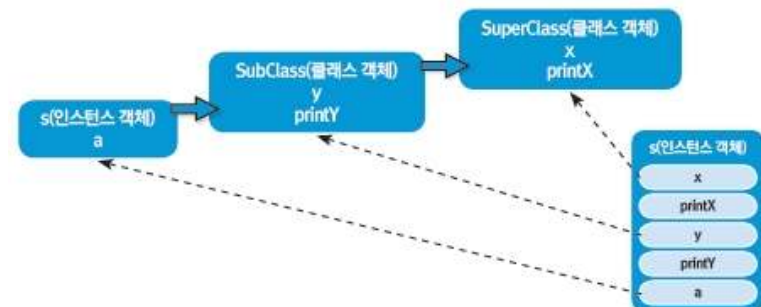
```
    y = 20
```

```
    def printY(self):
```

```
        print(self.y)
```

```
s = SubClass()
```

```
s.a = 30
```



명함 제작 프로그램

```
>>> name = "kimyuna"
>>> email = "yunakim@naver.com"
>>> addr = "seoul"
```

- 파이썬 변수를 이용하여 값을 저장 (바인딩)

```
>>> def print_business_card(name, email, addr):
    print("-----")
    print("Name: %s" % name)
    print("E-mail: %s" % email)
    print("Office Address: %s" % addr)
    print("-----")
```

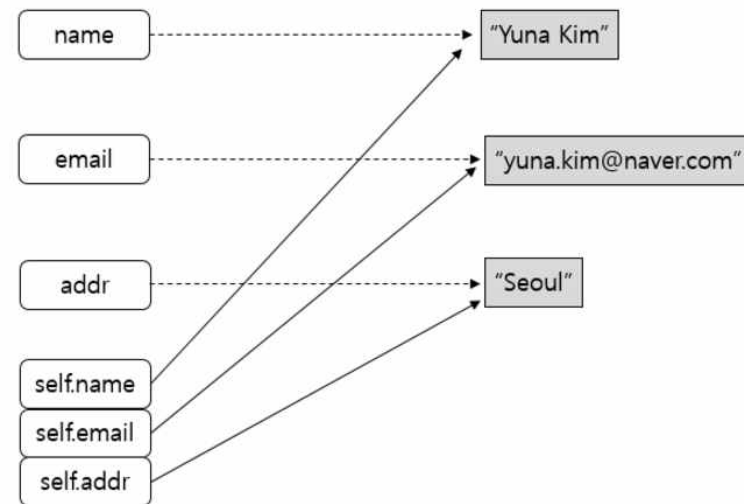
- 명함 정보 출력 함수

```
>>> print_business_card(name, email, addr)
-----
Name: kimyuna
E-mail: yunakim@naver.com
Office Address: seoul
-----
```

명함 제작 클래스

- 위 장의 명함 제작 프로그램을 클래스를 사용하여 생성하시오.
 - 사용자로부터 데이터를 입력받고
 - 저장하는 기능을 하는 함수를 추가
- 클래스에 메소드 추가하기

```
>>> class BusinessCard:  
    def set_info(self, name, email, addr):  
        self.name = name  
        self.email = email  
        self.addr = addr
```

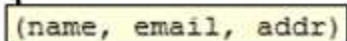


<변수의 바인딩 예>

명함 제작 클래스

- 클래스 인스턴스 생성 및 클래스 인스턴스를 통한 메소드 호출

```
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr
```

```
>>> member1 = BusinessCard()
>>> member1
<__main__.BusinessCard object at 0x030248F0>
>>> member1.set_info()
```

- 클래스 인스턴스를 통한 메소드 호출 및 변수 접근

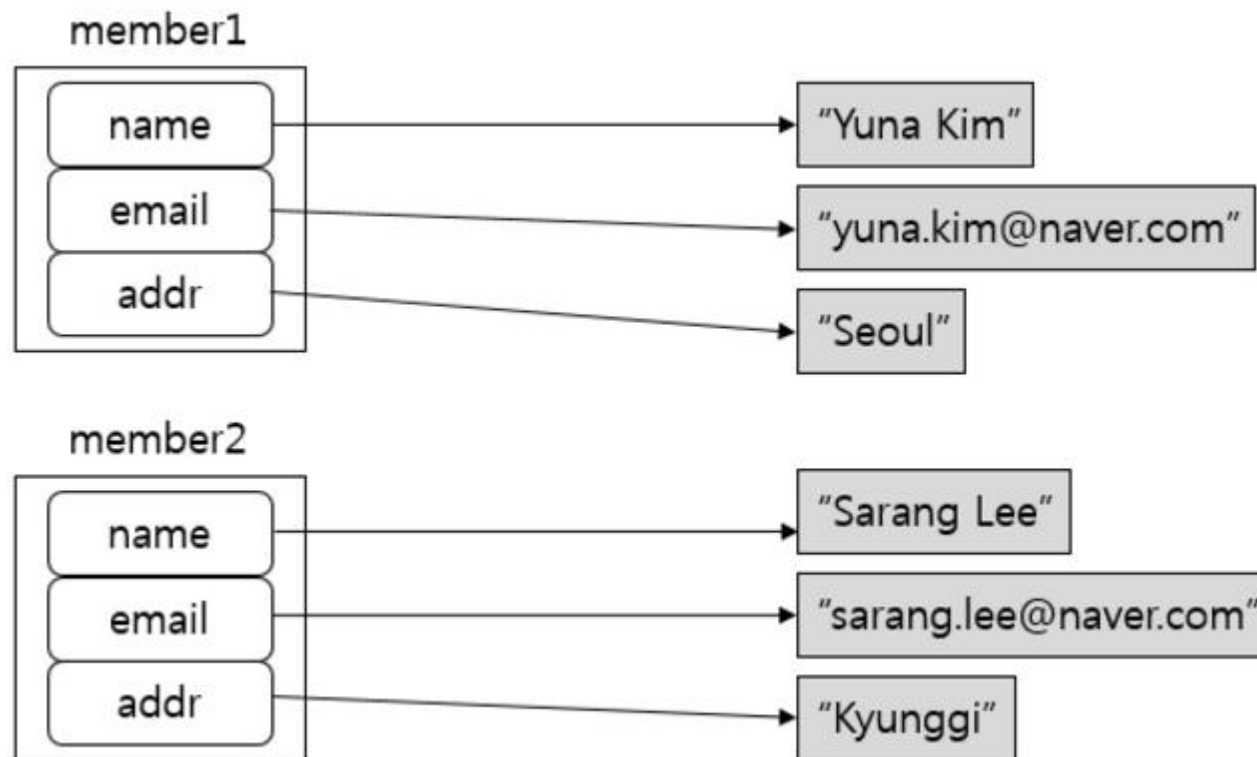
```
>>> member1.set_info("Yuna Kim", "yunakim@naver.com", "Seoul")
```

```
>>> member1.name
'Yuna Kim'
>>> member1.email
'yunakim@naver.com'
>>> member1.addr
'Seoul'
```

```
>>> member2 = BusinessCard()
>>> member2.set_info("Sarang Lee", "sarang.lee@naver.com", "Kyunggi")
```

명함 제작 클래스

- 클래스 인스턴스 상태



명함 제작 클래스

- 클래스 인스턴스 상태

```
>>> class BusinessCard:
    def set_info(self, name, email, addr):
        self.name = name
        self.email = email
        self.addr = addr
    def print_info(self):
        print("-----")
        print("Name: ", self.name)
        print("E-mail: ", self.email)
        print("Address: ", self.addr)
        print("-----")
```

- 인스턴스 변수 접근

```
>>> member1 = BusinessCard()
>>> member1.set_info("YunaKim", "yuna.kim@naver.com", "Seoul")
```

```
>>> member1.print_info()
-----
Name: YunaKim
E-mail: yuna.kim@naver.com
Address: Seoul
-----
```

과제

- 과제1. 다음과 같은 함수와 인스턴스를 호출 했을때 다음과 같은 결과를 출력하는 클래스와 메소드를 생성하시오.

```
>>> lemon = Fruit("lemon", "yellow", "sour", False)
>>> lemon.description()
I'm a yellow lemon and I taste sour.
>>> lemon.is_edible()
Yep! I'm edible.
```




과제 제출 방법

- ▶ 과제1, 명함제작 클래스, 상속 예제 캡처 후 워드or한글파일에 첨부/정리하여 제출 -> 주석 첨부
- ▶ 파일 형식 : [과제번호6]_이름(조이름)_학번
 - ▶ 제출 형식 어길 시 감점처리
- ▶ 제출 : **dbcyy1@gmail.com**로 제출
- ▶ 제출기간 : 5월10일 화요일 23시59분까지