클래스

클래스

❖ 클래스

- 관련 정보와 정보의 조작 함수(메서드)를 묶어서 관리

```
balance = 8000

def deposit(money):
    global balance
    balance += money

def inquire():
    print("잔액은 %d원 입니다."%balance)

deposit(1000)
inquire()
```

```
잔액은 9000원 입니다.
```

클래스

❖ 클래스 정의

- class 키워드로 정의
  - 사용하기 위해서는 인스턴스를 생성한 후 사용

```
class Account:
    def __init__(self, balance):  # 생성자 함수
        self.balance = balance

    def deposit(self, money):
        self.balance += money

    def inquire(self):
        print("잔액은 %d원 입니다."%self.balance)

account = Account(8000)   # Account의 인스턴스 생성
account.deposit(1000)
account.inquire()
```

```
잔액은 9000원 입니다.
```

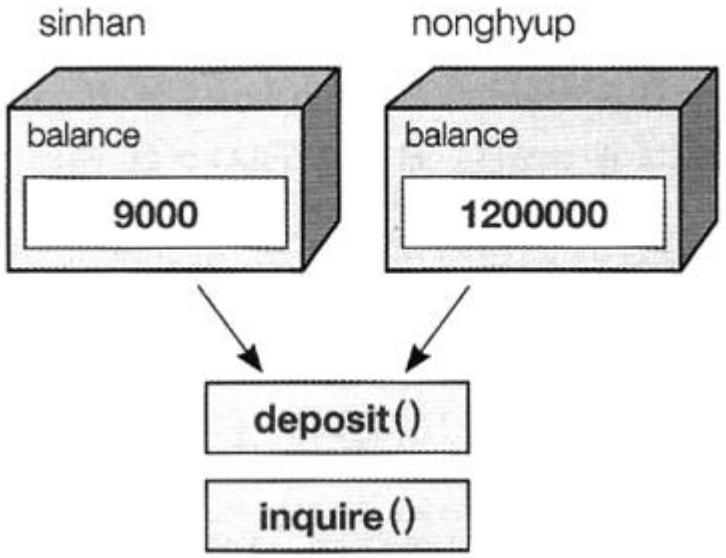
클래스

❖ 클래스 정의

```
shihan = Account(8000)   # Account의 인스턴스 생성
shihan.deposit(1000)
shihan.inquire()

nonghyup = Account(1200000)
shihan.inquire()
```

```
잔액은 9000원 입니다.
잔액은 1200000원 입니다.
```



클래스

❖ 생성자

- `__init__(self)`
  - 클래스의 인스턴스를 생성할 때 자동으로 호출
  - 멤버 변수 정의 및 초기화 역할

```
class 이름:
    def __init__(self, 초기값):
        멤버 초기화

    메서드 정의

객체 = 객체명(인수)
```

클래스

❖ 생성자

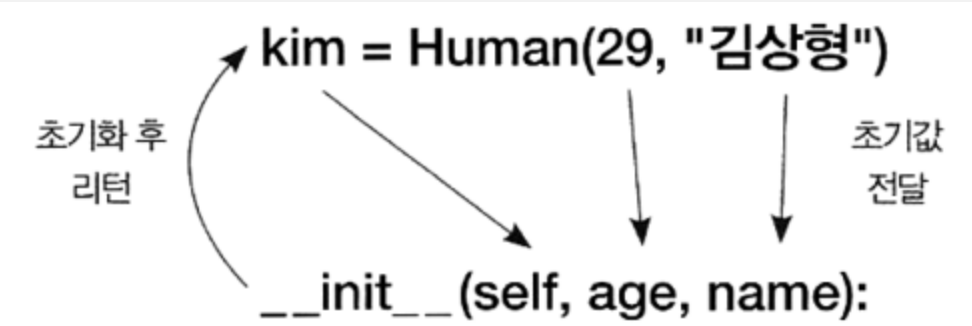
```
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def intro(self):
        print(str(self.age) + "살 " + self.name + "입니다.")


kim = Human("김상형", 29)
kim.intro()

lee = Human("이승우", 45)
lee.intro()
```

```
29살 김상형입니다.
45살 이승우입니다.
```



클래스

❖ 상속

- 기존 클래스 정의를 그대로 자신의 것으로 취하는 방법

```
class 자식클래스명(부모클래스명):
    ... # 자식 클래스 정의
```

클래스

❖ 상속

```
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def intro(self):
        print(str(self.age) + "살 " + self.name + "입니다.")


class Student(Human):
    def __init__(self, name, age, stunum):
        super().__init__(name, age)
        self.stunum = stunum

    def intro(self):
        super().intro()
        print("학번: " + str(self.stunum))

    def study(self):
        print("하늘천 따지 검을 현 누를 황")
```

클래스

❖ 상속

```
kim = Human("김상형", 29)
kim.intro()

lee = Student("이승우", 45, 930011)
lee.intro()
lee.study()
```

```
29살 김상형입니다.
45살 이승우입니다.
학번: 930011
하늘천 따지 검을 현 누를 황
```

클래스

❖ 액세스

- 파이썬은 기본적으로 정보 은폐 기능 지원하지 않음

```
kim = Human(29, "김상형")
kim.name = "정우성"
kim.age = 46
```

- getter/setter로 정보(프로퍼티)를 보호
  - @property
    - 함수명이 프로퍼티명이 되며 getter 함수로 동작
  - @프로퍼티명.setter
    - 프로퍼티의 setter() 함수 정의

클래스

❖ 액세스

```
class Date:
    def __init__(self, month):
        self.inner_month = month

    @property
    def month(self):
        return self.inner_month

    @month.setter
    def month(self, month):
        if 1 <= month <= 12:
            self.inner_month = month

today = Date(8)
today.month = 15

print(today.month)
```

```
8
```

클래스

❖ 액세스

- 프로퍼티명 = property(getter 함수, setter 함수)

```
class Date:
    def __init__(self, month):
        self.__month = month

    def getmonth(self):
        return self.__month

    def setmonth(self, month):
        if 1 <= month <= 12:
            self.inner_month = month

    month = property(getmonth, setmonth)
```

클래스

❖ 액세스

- 프라이빗 멤버 변수
  - 멤버 변수 앞에 __을 붙이면 외부에서 바로 접근 불가

```
today = Date(8)
today.month = 15

print(today.month)
print(today.__month)  # 예외 발생
```

여러가지 메서드

❖ 클래스 메서드

- 일반적인 메서드는 인스턴스 메서드
  - 반드시 인스턴스를 만든 후 사용 가능
  - 첫 번째 인자는 항상 인스턴스에 대한 참조(self)
- 클래스 메서드는 인스턴스와 무관하게 존재
  - 인스턴스 없이도 클래스명을 통해 접근 가능
  - 첫 번째 인자는 클래스에 대한 참조(cls)
- @classmethod로 정의

❖ 클래스 멤버 변수

- class 안에서 self와 무관하게 정의되는 멤버 변수
- 인스턴스와 무관하게 존재하며 모든 인스턴스가 공유하는 정보

여러가지 메서드

❖ 클래스 메소드

```
class Car:
    count = 0

    def __init__(self, name):
        self.name = name
        Car.count += 1

    @classmethod
    def outcount(cls):
        print(cls.count)

pride = Car("프라이드")
korando = Car("코란도")
Car.outcount()
```

```
2
```

여러가지 메서드

❖ 정적 메서드

- 단순히 클래스 내에 정의되는 일반함수
  - 클래스에 대한 어떠한 정보도 제공하지 않음
  - 첫 번째 인자가 정해져 있지 않음
- 비슷한 성격의 함수를 묶어서 관리하는 역할

여러가지 메서드

❖ 정적 메소드

```
class Car:
    @staticmethod
    def hello():
        print("오늘도 안전 운행 합시다.")

    count = 0

    def __init__(self, name):
        self.name = name
        Car.count += 1

    @classmethod
    def outcount(cls):
        print(cls.count)

Car.hello()
```

```
오늘도 안전 운행 합시다.
```

여러가지 메서드

❖ 연산자 메서드

- 연산자를 재정의할 수 있는 함수
- 연산자 별로 함수명이 정해져 있음
- == : __eq__
- != : __ne__
- < : __lt__
- > : __gt__
- <= : __le__
- >= : __ge__
- + : __add__, __radd__
- - : __sub__, __rsub__
- * : __mul__, __rmul__
- / : __div__, __rdiv__
- // : __floordiv__, __rfloordiv__
- % : __mod__, __rmod__
- ** : __pow__, __rpow__
- << : __lshift__
- >> : __rshift__

여러가지 메서드

❖ 연산자 메소드

```
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __eq__(self, other):
        return self.name == other.name and self.age == other.age

kim = Human("김상형", 29)
sang = Human("김상형", 29)
moon = Human("문종민", 44)

print(kim == sang)
print(kim == moon)
```

```
True
False
```

여러가지 메서드

❖ __str__()

- 객체의 정보를 출력하기위해 객체의 내부 정보를 문자열로 리턴하는 함수
- print() 함수에 객체를 지정했을 때 해당 객체의 __str__() 호출됨

```
class Human:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return "이름 %s, 나이 %d" % (self.name, self.age)

kim = Human("김상형", 29)
print(kim)
```

```
이름 김상형, 나이 29
```