# Lab8. Object-Oriented Programming I

*CSED101 LAB*

# 객체 지향 프로그래밍

- 객체를 중심으로 프로그래밍 하는 기법
  - 객체와 그 객체들간의 상호작용 관점을 서술

- 장점
  - 어떤 하나의 '객체'에 '객체'가 필요로 하는 데이터와 기능들을 모두 넣어 코드의 재사용성, 확장성, 가독성 개선

# Python은 모든 것이 객체

- int, string, list, dict 등 모두가 객체

```
list_ex = list([1,2,3])
```

```
list_ex.
```

append
clear
copy
count
extend
index
insert
pop
remove
reverse

```
dict_ex = dict({"dog":1})
```

```
dict_ex.
```

clear
copy
fromkeys
get
items
keys
pop
popitem
setdefault
update

```
int_ex = int(5)
```

```
int_ex.
```

bit_length
conjugate
denominator
from_bytes
imag
numerator
real
to_bytes

# 클래스 (Class)

- 변수와 함수를 묶어서 객체로 만들어 주는 개념
- 객체의 기반이 되는 틀을 정의하는 개념
- 정의

```
class ClassName:
    <statement-1>
    ...
    <statement-N>
```

- 클래스의 구성 요소
  - 변수(데이터) – 속성(attribute)
  - 메서드(데이터를 조작하는 행위)

```
# Dog 클래스 정의
class Dog:
    # 생성자
    def __init__(self, name, age):
        self.name = name
        self.age  = age

    # 메서드
    def bark(self):
        return 'wal wal'
```

# 인스턴스 (Instance)

```
1  class Dog:                    # 클래스 정의
2      def __init__(self, name):
3          self.name = name
4
5      def bark(self):
6          return 'wal wal'
```

```
1  dog1 = Dog('Mongja')      # 인스턴스 생성
```

```
1  dog1.     # .뒤에서 tab키 누름
```

bark
name

```
1
2  dog1.name
```

```
'Mongja'
```

```
1  dog1.bark()
```

```
'wal wal'
```

# 메서드 (Method)

- 클래스에 종속되어 있는 함수
- 대상이 되는 객체가 있어야 하며, 모든 메서드는 첫번째 인자가 항상 존재해야 함(self)

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

    def print_name_age(self):
        print(self.name, self.age)
```

```python
dog1 = Dog('Mongja', 3)

dog2 = Dog('Jindo', 1)
```

동등한 결과

```python
dog1.print_name_age()
dog2.print_name_age()
```

```
Mongja 3
Jindo 1
```

```python
Dog.print_name_age(dog1)
Dog.print_name_age(dog2)
```

```
Mongja 3
Jindo 1
```

- self는 클래스의 인스턴스를 말함
- self를 통해서 인스턴스의 메서드와 속성에 접근 가능

# 특수 메서드 (Special Method)

- 특수 메서드의 특징
  - 항상 "__메서드이름__"과 같은 이름을 가짐
  - 내장 함수와 동일하게 미리 동작이 정의 되어 있음
  - 일반적인 메서드의 사용 방법과는 다르게 특정 상황에서 자동으로 작동함
- 가장 대표적인 특수 메서드 __init__
  - 인스턴스 생성시 자동으로 호출되는 메서드
  - 인스턴스가 갖게 될 변수를 초기화 해 줌

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

# 클래스 변수

- 모든 인스턴스가 공유하는 변수
- 생성된 인스턴스 수를 알고 싶은 경우:

```
class Dog:

    count = 0   # 클래스 변수

    def __init__(self, name, age):
        self.name = name
        self.age  = age
        Dog.count += 1  # Dog 클래스 변수 값 변경
```

```
dog1 = Dog("Mongja", 3)
dog2 = Dog("Jindo", 1)

# 클래스 변수에 접근
print(Dog.count)  # 2  <-- 추천
print(dog1.count) # 2
```

# 실습 1

- 은행 계좌 클래스 만들기
  - 계좌 클래스의 __init__() 인자로 balance와 name을 가짐
    - 계좌 생성시 초기값을 name와 balance를 지정 가능
    - 지정하지 않으면 각 초기값은 "none"과 0을 가지도록 할 것
  - 계좌의 기능은 입금, 출금, 계좌정보출력
  - 출금은 출금하고자 하는 금액이 잔고에 있을 때 출금할 것

```
1  class BankAccount:
2      def __init__(self, ?):
3          pass
4
5      def deposit(self, amount):
6          pass
7
8      def withdraw(self, amount):
9          pass
10
11     def get_info(self):
12         pass
```

실행 예시1)

```
1  acc1 = BankAccount("홍길동", 100) # 계좌 생성
2  ??   # 400원 입금
3  ??   # 600원 출금
```

잔액 부족!

```
1  print(acc1.balance)
```

500

실행 예시2)

```
1  acc2 = BankAccount() # 계좌 생성
2  ?? # 1000원 입금
3  ?? # 900원 출금
4  ?? # 계좌 정보 출력
```

이름: none, 잔고: 100

# Problem

- 실습 1에서 구현한 BankAccount 클래스에 아래 메서드 추가 구현
- 계좌이체 메서드 구현
  - def transfer(self, other, amount)

**실행 예시1)**

```
1  acc1 = BankAccount("고길동", 1000)
2  acc2 = BankAccount("고영희", 200)
3  acc1.transfer(acc2, 300)
4  acc1.get_info()
5  acc2.get_info()
```

```
이름: 고길동, 잔고: 700
이름: 고영희, 잔고: 500
```

**실행 예시2)**

```
1  acc1 = BankAccount("고길동", 1000)
2  acc2 = BankAccount("고희동", 0)
3  acc1.transfer(acc2, 2000)
4  print(acc1)
5  print(acc2)
```

```
잔액 부족!
이름: 고길동, 잔고: 1000
이름: 고희동, 잔고: 0
```

- 제출파일명: 학번_Lab8.py