

Lab9. Object-Oriented Programming 2

CSED101 LAB

연산자 오버로딩 (Operator Overloading)

- Python의 연산자는 모두 내장된 특수 메소드를 호출함
 - 해당 메소드를 override 하는 것으로 기능 변경 가능
 - 예) $a + b$ 는 자동으로 `a.__add__(b)` 를 호출

```
1 class BankAccount:  
2     def __init__(self, name="none", balance=0):  
3         self.balance = balance  
4         self.name = name  
5  
6     def deposit(self, amount):  
7         self.balance += amount  
8  
9     # 생략
```

```
1 acc = BankAccount("홍길동", 1000)  
2 print(acc)
```

이름: 홍길동, 잔고: 1000

```
1 acc + 1000
```

```
-----  
TypeError                                Traceback  
~\AppData\Local\Temp\ipykernel_26460\3531347346  
----> 1 acc + 1000
```

`TypeError: unsupported operand type(s) for +: 'Ban`

```
1 class BankAccount:  
2     def __init__(self, name="none", balance=0):  
3         self.balance = balance  
4         self.name = name  
5  
6     def deposit(self, amount):  
7         self.balance += amount  
8  
9     def __add__(self, amount):  
10        self.deposit(amount)  
11  
12     def __iadd__(self, amount):  
13        pass  
14  
15     # 생략
```

```
1 acc = BankAccount("홍길동", 1000)  
2 print(acc)
```

이름: 홍길동, 잔고: 1000

```
1 acc + 1000  
2 print(acc)
```

이름: 홍길동, 잔고: 2000

연산자 오버로딩 (Operator Overloading)

연산	호출되는 함수
$a + b$	<code>a.__add__(b)</code>
$a - b$	<code>a.__sub__(b)</code>
$a * b$	<code>a.__mul__(b)</code>
a / b	<code>a.__truediv__(b)</code>
$a // b$	<code>a.__floordiv__(b)</code>
$a \% b$	<code>a.__mod__(b)</code>
$a ** b$	<code>a.__pow__(b)</code>
$a \& b$	<code>a.__and__(b)</code>
$a b$	<code>a.__or__(b)</code>
$a ^ b$	<code>a.__xor__(b)</code>
$-a$	<code>a.__neg__()</code>
$\sim a$	<code>a.__invert__()</code>

연산	호출되는 함수
$a += b$	<code>a.__iadd__(b)</code>
$a -= b$	<code>a.__isub__(b)</code>
$a *= b$	<code>a.__imul__(b)</code>
$a /= b$	<code>a.__itruediv__(b)</code>
$a //= b$	<code>a.__ifloordiv__(b)</code>
$a \%= b$	<code>a.__mod__(b)</code>
$a **= b$	<code>a.__ipow__(b)</code>
$a \&= b$	<code>a.__iand__(b)</code>
$a = b$	<code>a.__ior__(b)</code>
$a ^= b$	<code>a.__ixor__(b)</code>

연산	호출되는 함수
$a < b$	<code>a.__lt__(b)</code>
$a \leq b$	<code>a.__le__(b)</code>
$a == b$	<code>a.__eq__(b)</code>
$a != b$	<code>a.__ne__(b)</code>
$a > b$	<code>a.__gt__(b)</code>
$a \geq b$	<code>a.__ge__(b)</code>

연산	호출되는 함수
$a[key]$	<code>a.__getitem__(key)</code>
$a[key]$	<code>a.__setitem__(key)</code>
<code>len(a)</code>	<code>a.__len__()</code>
<code>print(a)</code>	<code>a.__str__()</code>

Problem 1 (연산자 오버로딩)

- Lab9.py를 Lab9과제에서 다운로드 받아서 BankAccount 클래스 아래의 4개 특수 메서드를 채워 실행 예시와 동일하게 동작하도록 작성하시오.

```
class BankAccount:
```

```
    ... 생략 ...
```

```
    def __add__(self, amount):  
        pass
```

```
    def __iadd__(self, amount):  
        pass
```

```
    def __sub__(self, amount):  
        pass
```

```
    def __isub__(self, amount):  
        pass
```

<실행 예시>

```
1 acc1 = BankAccount("홍길동", 0)  
2 print(acc1) # 이름: 홍길동, 잔고: 0  
3 acc1 + 1000  
4 print(acc1) # 이름: 홍길동, 잔고: 1000  
5 acc1 += 2000  
6 print(acc1) # 이름: 홍길동, 잔고: 3000  
7 acc1 - 500  
8 print(acc1) # 이름: 홍길동, 잔고: 2500  
9 acc1 -= 3000 # 잔액 부족!  
10 print(acc1) # 이름: 홍길동, 잔고: 2500
```

이름: 홍길동, 잔고: 0

이름: 홍길동, 잔고: 1000

이름: 홍길동, 잔고: 3000

이름: 홍길동, 잔고: 2500

잔액 부족!

이름: 홍길동, 잔고: 2500

상속 (Inheritance)

- 기존 클래스의 속성을 물려 받아 새로운 클래스를 만드는 것
 - 새 클래스는 기존 클래스의 모든 변수 및 메서드를 가짐
 - 주로 기존 클래스를 확장하는 용도로 사용
 - 다중 클래스도 상속 가능, ","로 구분

```
class DerivedClassName(BaseClassName):  
    <statement-1>  
    ...  
    <statement-N>
```

상속 예시

```
class A:  
    a = 1  
    def print_a(self):  
        print("A class")
```

```
class B:  
    b = 1  
    def print_b(self):  
        print("B class")
```

```
class C(A):  
    c = 1  
    def print_c(self):  
        print("C class")
```

```
class D(C, B):  
    d = 1  
    def print_d(self):  
        print("D class")
```

Method
Resolution
Order

```
c = C()  
d = D()
```

```
D.mro()
```

```
[__main__.D, __main__.C, __main__.A, __main__.B, object]
```

메서드 오버라이딩 (Overriding)

- 기존 클래스 메서드를 상속 클래스에서 재정의 하는 것
 - 재정의하지 않은 메서드는 기존 클래스의 것을 그대로 사용
 - 오버라이딩한 경우, 기존 클래스의 메서드를 사용하고자 하는 경우 `super()` 사용

```
1  class A:  
2      def __init__(self):  
3          print("this is [A] class and [__init__] function")  
4          self.v1 = 0  
5          self.v2 = 0  
6  
7  class B(A):  
8      def __init__(self):  
9          super().__init__()  
10         print("this is [B] class and [__init__] function")  
11         self.v3 = 0  
12         self.v4 = 0
```

```
1  a = A()  
2  b = B()
```

```
this is [A] class and [__init__] function  
this is [A] class and [__init__] function  
this is [B] class and [__init__] function
```

실습

- Person, Student, Professor 클래스 구현
 - Student, Professor는 Person을 상속받음
- Person
 - 속성: name(str), age(int), department(str)
 - 메서드: __init__, get_name
- Student
 - 속성: id(int), GPA(float), advisor(Professor)
 - 메서드: __init__, print_info, reg_advisor
- Professor
 - 속성: position(str), laboratory(str), student(list of Student)
 - 메서드: __init__, print_info, reg_student

```
Stu1 = Student('김학생', 30, 'CSE', 20191234, 4.3)
stu2 = Student('이학생', 25, 'CSE', 20221234, 3.5)
prof1 = Professor('박교수', 50, 'CSE', 'Full', 'CSLab')

stu1.reg_advisor(prof1)
stu2.reg_advisor(prof1)
prof1.reg_student(stu1)
prof1.reg_student(stu2)

stu1.print_info()
stu2.print_info()
prof1.print_info()

# 실행 예시
제 이름은 김학생, 나이는 30, 학과는 CSE,
지도교수님은 박교수 입니다
제 이름은 이학생, 나이는 25, 학과는 CSE,
지도교수님은 박교수 입니다
제 이름은 박교수, 나이는 55, 학과는 CSE,
지도학생은 김학생, 이학생 입니다
```

Problem 2 (상속 및 오버라이딩)

- 최소 잔액을 유지해야 하는 계좌 MinimumBalanceAccount 클래스를 정의하시오.
 - 기본적으로 Problem1에서 정의한 BankAccount 클래스와 기능 동일함
 - 하지만 출금시 잔액이 최소 잔액 미만이 되면 출금 못함
 - 그리고 "최소 잔액을 유지해야 합니다" 메시지 출력
- 반드시 상속 및 오버라이딩 활용할 것
- Problem1과 동일한 파일에 코드를 작성할 것
- 제출파일명: Lab9_학번.py

<실행 예시>

```
1 class MinimumBalanceAccount(??):  
2     def __init__(self, name="none", balance=0, min_bal=0):  
3         pass  
4  
5     def withdraw(self, amount):  
6         pass
```

```
1 acc = MinimumBalanceAccount("홍길동", 2000, 1000)  
2 acc.deposit(1000)  
3 acc.withdraw(2500)  
4 acc.get_info()
```

최소 잔액을 유지해야 합니다
이름: 홍길동, 잔고: 3000