• 01/21

```
def my_sum(a, b):
    result = a + b
    return result

aa = my_sum(10,20)
print(aa)
```

formal parameter 매개변수

a, b result

local variable (지역변수)

memory

Stack

?

<class function>

my_sum

300

aa 70

70

30

객체

300

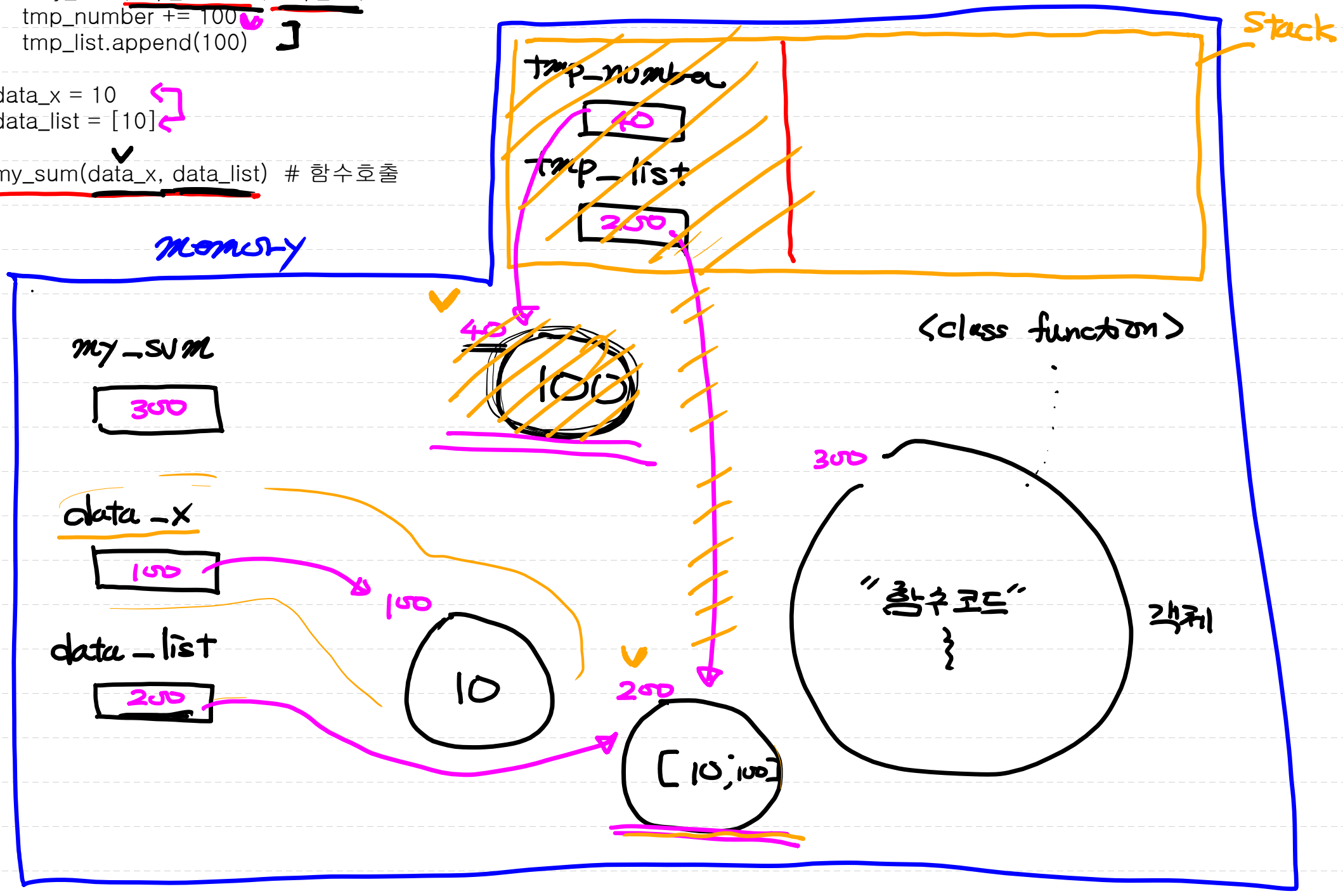
참수의 실행코드

result=a+b

return result

객체

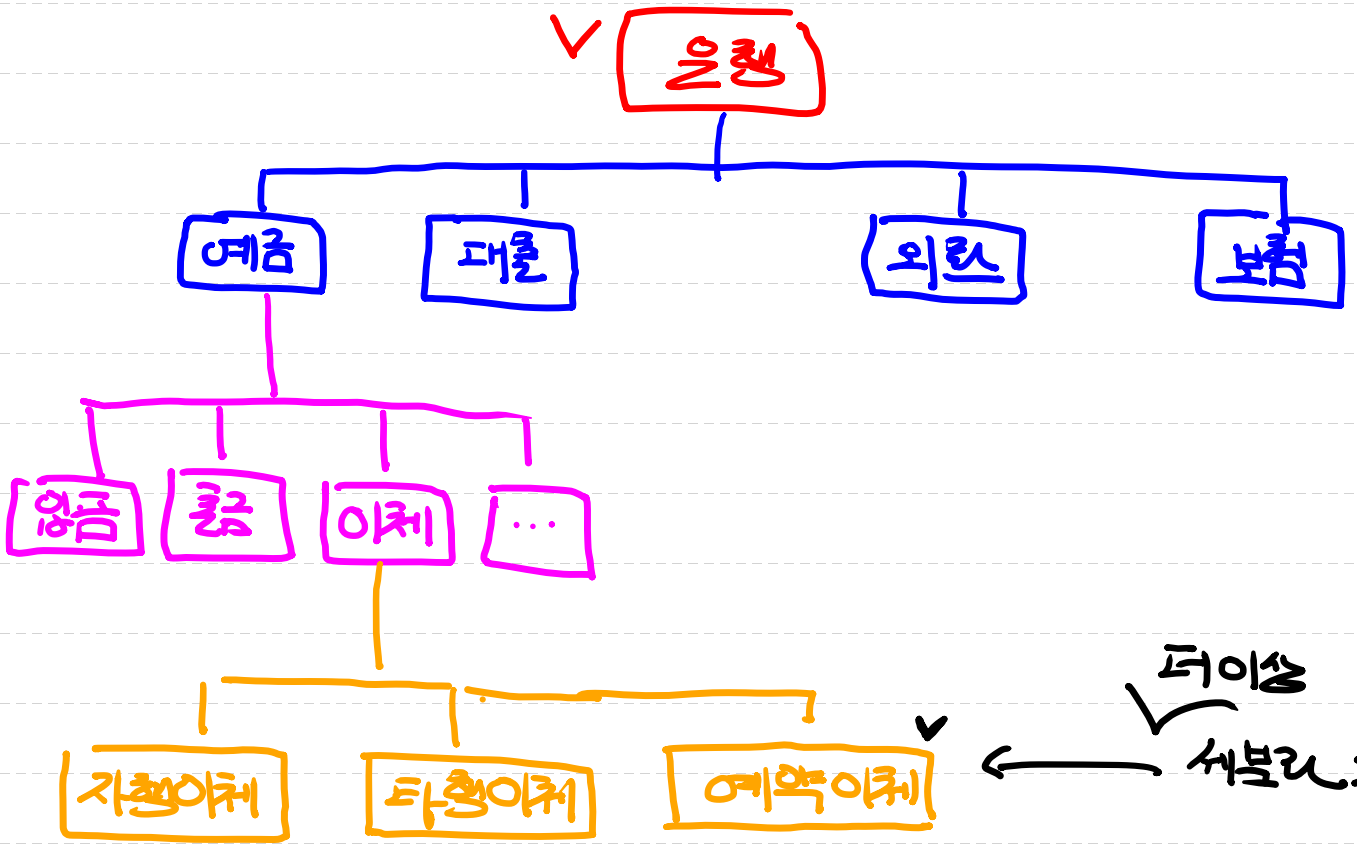
```
def my_sum(tmp_number, tmp_list):
    tmp_number += 100
    tmp_list.append(100)

data_x = 10
data_list = [10]

my_sum(data_x, data_list)  # 함수호출
```



객체지향

→ 프로그램을 구조화하는 분석.

- 과거 (초기 ~1990) : 기능 단위의 프로그램을 구현
- Domain 분석 → 설계 → 구현 → 테스트 → 배포



C 언어

⇒ 구조적 프로그래밍
(절차적 프로그래밍)

장점 : 분석/설계가 쉬워요!
↓
프로그램의 개발속도가 빨라요
↓
비용이 적게들어요

단점 :

- 더이상 세분화할 수 없는 기능 ← "함수"
- 코드의 결합도 ↑
  → 프로그램의 유지보수가 힘들어져요!

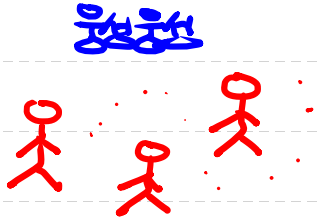
1990 ~ "인터넷 보급"

↓

"정보량이 증가" → 사회가 급변 → 프로그램의 유지보수가 빈번

↓

비용의 증가.



⇒ "OO" (Object Oriented)

✓ 장점 : 유지보수가 상대적으로 좋아요

단점 : 분석/설계가 어려워요

(프로그램을 만들기 어려워요)

↓

Java, C++, C#

어떻게 분석/설계를 하나요??

현실세계의 해결해야 하는 문제를 그대로 프로그램으로 표현 (modeling)

"요점"

① 현실세계의 요점을 구성하고 있는 구성요소파악.

② 이 구성요소들 간의 데이터 통신이 어떻게 되는지 파악/구현

→ 분석/설계가 사람마다 달라져요

→ 프로그램을 만들기가 어려워요.

① 지점 ∨ ○ →
② 사람 ∨ ○ →
③ 텔러 ∨ ×
④ 계좌 ∨ ○ →
⋮
99 ㅌㅈㅏ ∨ ×

현실세계의 개체(객체) → 프로그램으로 표현

"객체 모델링"

? ∨
- 속성값 ∨
  - 계좌번호
  - 예금주 ∨
  - 잔액 ∨
  - 개설일 ∨
  - 은행지점
  - 거래내역
  ⇒ 변수
- 동작 ∨
  - 입금한다 ∨
  - 출금한다 ∨
  - 이체한다 ∨
  - 계좌개설한다
  - 계좌삭제한다

→ 함수

"추상화"

abstraction

Class

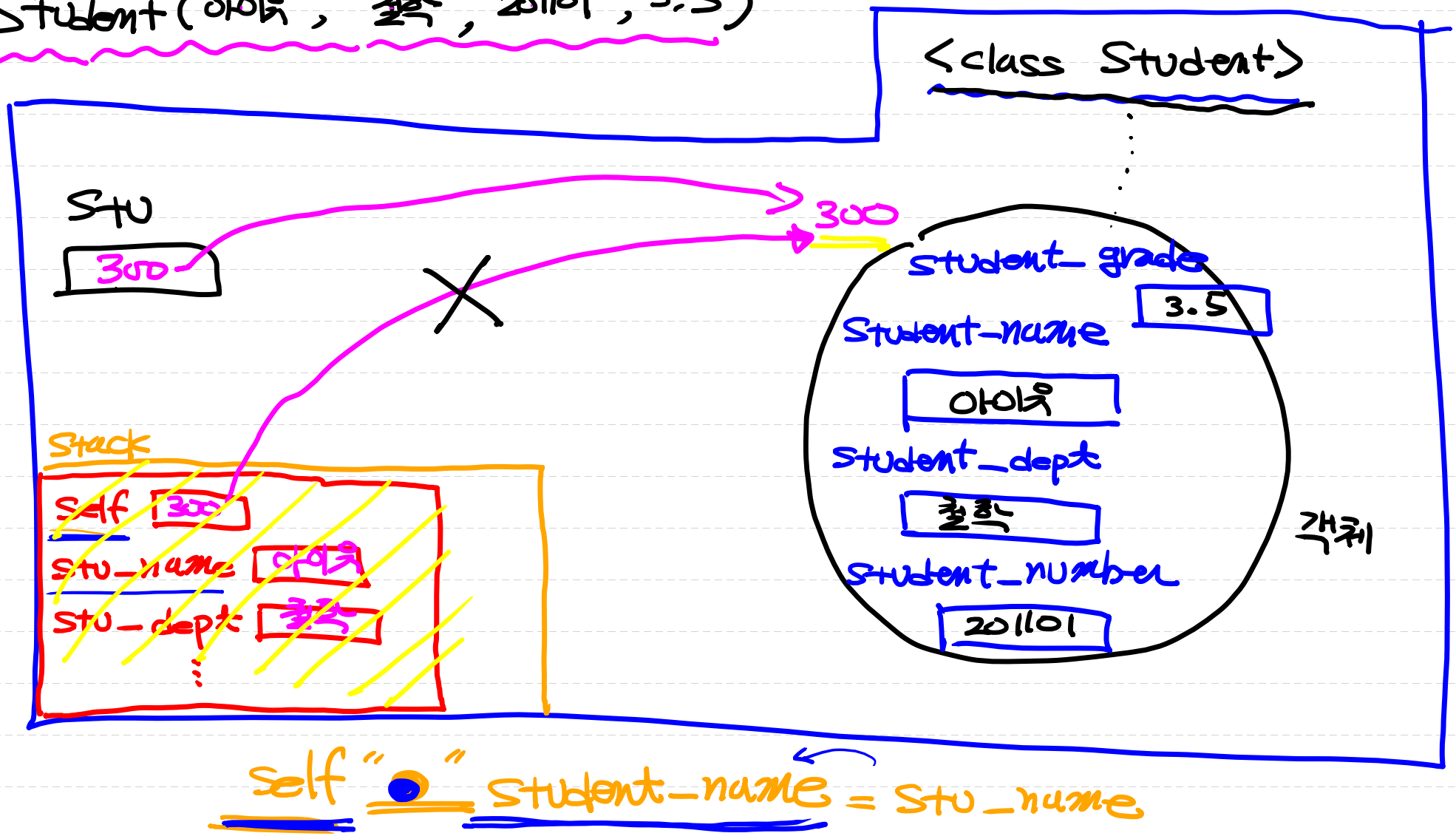
"객체 모델링의 수단"

<class Student>

```
class Student(object):

    # 특별한 함수를 하나 정의할 꺼예요! 이름이 정해져 있어요!
    def __init__(self, stu_name, stu_dept, stu_number, stu_grade):
        self.student_name = stu_name
        self.student_dept = stu_dept
        self.student_number = stu_number
        self.student_grade = stu_grade
```

코드에서 직접호출하지 않아요!!

stu = Student('아이유', '물리', 20110l , 3.5)



self "." student_name = stu_name