

자료구조/알고리즘

3주차 온라인 수업

배시영



3주차

파이썬 리뷰



리스트

- 어떤 경우에는 여러 개의 데이터를 하나로 묶어서 저장하는 것이 필요하다.



```
heroes = [ "아이언맨", "토르", "헐크", "스칼렛 위치" ]
```

리스트에 항목을 추가

- 공백 리스트를 생성한 후에 코드로 리스트에 값을 추가하는 것

```
list = []  
list.append(1)  
list.append(2)  
list.append(6)  
list.append(3)
```

```
[1, 2, 6, 3]
```

리스트에 항목을 추가

```
>>> heroes = []  
>>> heroes.append("아이언맨")  
['아이언맨']  
>>> heroes.append("닥터 스트레인지")  
>>> print(heroes)  
['아이언맨', '닥터 스트레인지']
```



점의 의미


- 파이썬에서 모든 것은 객체(object)이다. 객체는 관련되는 변수와 함수를 묶은 것이다.
- 파이썬에서 리스트도 객체이다. 객체 안에 있는 무엇인가를 사용할 때는 객체의 이름을 쓰고 점(.)을 붙인 후에 함수의 이름을 적는다.

```
heroes.append("아이언맨")
```

리스트 항목 접근하기

```
>>> letters = ['A', 'B', 'C', 'D', 'E', 'F']
```

'A'	'B'	'C'	'D'	'E'	'F'
0	1	2	3	4	5



```
>>> print(letters[0])
```

A

```
>>> print(letters[1])
```

B

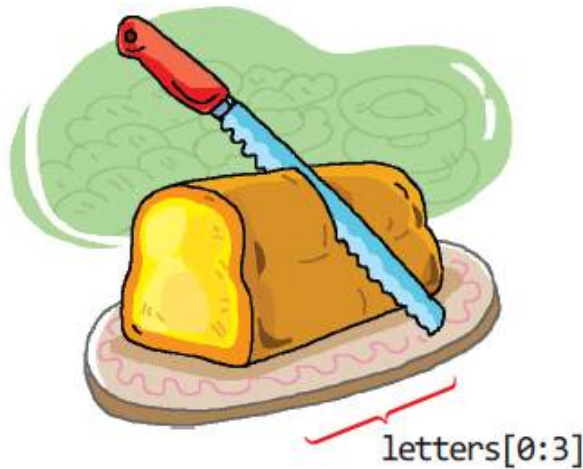
```
>>> print(letters[2])
```

C

슬라이싱

- 슬라이싱(slicing)은 리스트에서 한 번에 여러 개의 항목을 추출하는 기법이다.

```
>>> letters = ['A', 'B', 'C', 'D', 'E', 'F']  
>>> print(letters[0:3])  
['A', 'B', 'C']
```



인덱스 생략

```
>>> print(letters[:3])  
['A', 'B', 'C']
```

```
>>> print(letters[3:])  
['D', 'E', 'F']
```

```
>>> print(letters[:])  
['A', 'B', 'C', 'D', 'E', 'F']
```

리스트 항목(원소) 처음 부터 끝까지

리스트 항목 변경하기

```
>>> heroes = ["아이언맨", "토르", "헐크", "스칼렛 위치"]  
>>> heroes[1] = "닥터 스트레인지"  
>>> print(heroes)  
['아이언맨', '닥터 스트레인지', '헐크', '스칼렛 위치']  
>>>
```

인덱스를 이용한다.

함수를 이용하여 추가하기

```
>>> heroes.append("스파이더맨")
```

```
>>> print(heroes)
```

```
['아이언맨', '닥터 스트레인지', '헐크', '스칼렛 위치', '스파이더맨']
```

```
>>> heroes.insert(1, "배트맨")
```

```
>>> print(heroes)
```

```
['아이언맨', '배트맨', '닥터 스트레인지', '헐크', '스칼렛 위치', '스파이더맨']
```

```
>>>
```

항목 삭제하기

```
heroes = ["아이언맨", "토르", "헐크", "스칼렛 위치"]  
heroes.remove("스칼렛 위치")  
print(heroes)
```

```
['아이언맨', '토르', '헐크']
```

항목이 리스트 안에 있는지 체크

```
if "슈퍼맨" in heroes:  
    heroes.remove("슈퍼맨")
```

del

- del는 인덱스를 사용하여 항목을 삭제한다.

```
heroes = ["아이언맨", "토르", "헐크", "스칼렛 위치"]  
del heroes[0]  
print(heroes)
```

```
['토르', '헐크', '스칼렛 위치']
```

pop()

- pop()은 리스트에서 마지막 항목을 삭제

```
heroes = ["아이언맨", "토르", "헐크", "스칼렛 위치"]  
last_hero = heroes.pop()  
print(last_hero)
```

스칼렛 위치

리스트 탐색하기

- index() 사용

```
heroes = ["아이언맨", "토르", "헐크", "스칼렛 위치"]  
print(heroes.index("헐크"))
```

2

리스트 방문하기

```
heroes = [ "아이언맨", "토르", "헐크", "스칼렛 위치" ]  
for hero in heroes:  
    print(hero)
```

아이언맨
토르
헐크
스칼렛 위치

리스트 정렬하기

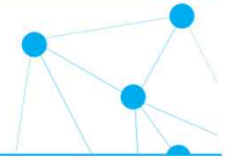
```
heroes = [ "아이언맨", "토르", "헐크", "스칼렛 위치" ]  
heroes.sort()  
print(heroes)
```

['스칼렛 위치', '아이언맨', '토르', '헐크']

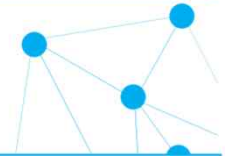


- 리스트(list)

- 스마트한 배열



메소드	설명
s.append(item)	항목 item을 리스트 s의 맨 뒤에 추가한다.
s.extend(lst)	리스트 lst를 s에 추가한다.
s.count(item)	리스트에서 항목 item의 개수를 세고 그 개수를 반환한다.
s.index(item,[시작],[종료])	리스트에서 항목 item을 찾아 가장 작은 인덱스를 반환한다. 탐색의 시작 위치와 종료 위치를 지정할 수도 있다.
s.insert(pos, item)	pos 위치에 항목 item을 삽입한다.
s.pop(pos)	pos 위치의 항목을 s에서 꺼내고 반환한다.
s.remove(item)	항목 item을 s에서 제거한다.
s.reverse()	리스트 항목의 순서를 뒤집는다.
s.sort([key], [reverse])	항목을 정렬한다.
s.clear()	리스트 값 모두 삭제(빈 리스트로 초기화)



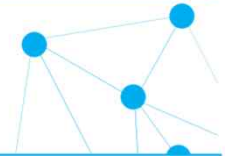
- 튜플(tuple)
 - 리스트(list)와 동일하지만 크기나 값을 변경할 수 없음

```
t = ( 0, 3, 7 )           # 항목이 3개인 튜플
a = ( 2 )                 # 항목이 1개인 튜플
b = ( 'game', 1, 3.14, 2019 ) # 항목이 4개인 복합 튜플
```

```
hobby = ("골프")
age = (36)
score = (94.5)
```

```
print (" 취미=%s, 나이=%d, 학점=%f" % (hobby, age, score))
```

```
취미=골프, 나이=36, 학점 = 94.500000
```



- 딕셔너리(dict)

- 키(key)와 관련된 값(value)로 이루어진 항목(entry)들의 집합

```
map = { '김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구' }  
print(map)  
print('쿠드롱이 뭐하는 사람이지? ', map['쿠드롱'])
```

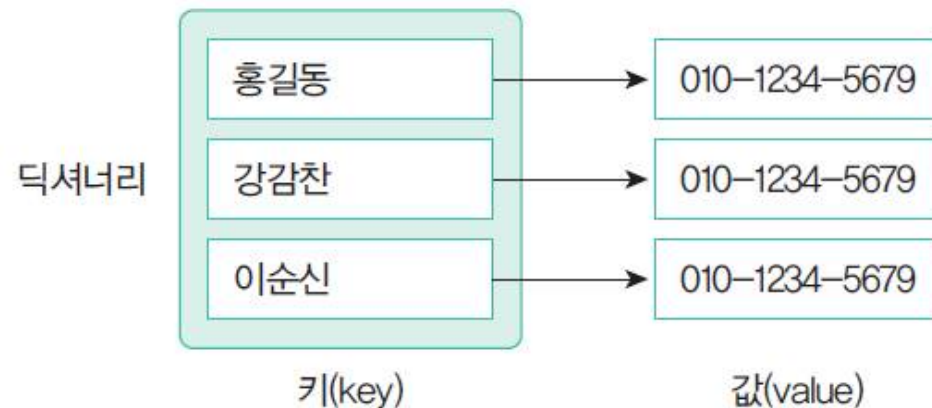
```
C:\WINDOWS\system32\cmd.exe  
{'김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구'}  
쿠드롱이 뭐하는 사람이지?  당구
```

```
map['나달'] = '테니스' # 맵에 항목 추가  
map.update({'최민영': '여자야구', '고진영': '골프'}) # 여러 항목 추가  
print(map)
```

```
C:\WINDOWS\system32\cmd.exe  
{'김연아': '피겨', '류현진': '야구', '쿠드롱': '당구', '메시': '축구', '나달': '테니스', '최민영': '여자야구', '고진영': '골프'}
```

딕셔너리

- 딕셔너리(dictionary)도 리스트와 같이 값을 저장하는 방법이다. 하지만 딕셔너리에는 값(value)과 관련된 키(key)가 있다.



딕셔너리

```
>>> phone_book = { }  
>>> phone_book["홍길동"] = "010-1234-5678"  
  
>>> print(phone_book)  
{'홍길동': '010-1234-5678'}
```

```
>>> phone_book = {"홍길동": "010-1234-5678"}  
>>> phone_book["강감찬"] = "010-1234-5679"  
>>> phone_book["이순신"] = "010-1234-5680"  
  
>>> print(phone_book)  
{'이순신': '010-1234-5680', '홍길동': '010-1234-5678', '강감찬': '010-1234-5679'}
```

딕셔너리에서 탐색

- 키를 가지고 값을 찾는다.

```
>>> print(phone_book["강감찬"])  
010-1234-5679
```


딕셔너리의 모든 키 출력하기

```
>>> phone_book.keys()  
dict_keys(['이순신', '홍길동', '강감찬'])
```

```
>>> phone_book.values()  
dict_values(['010-1234-5680', '010-1234-5678', '010-1234-5679'])
```

예제

- 한 학생에 대한 정보를 딕셔너리로 저장하기

```
dict = {'Name': '홍길동', 'Age': 7, 'Class': '초급'}  
print (dict['Name'])  
print (dict['Age'])
```

홍길동
7

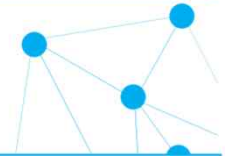
딕셔너리 항목 방문

```
>>> for key in sorted(phone_book.keys()):  
    print(key, phone_book[key])
```

강감찬 010-1234-5679

이순신 010-1234-5680

홍길동 010-1234-5678



- 집합(set)
 - set과 frozenset(내용을 변경할 수 없는 set)

```
s1 = { 1,2,3 }           # 집합 객체
s2 = { 2,3,4,5 }         # 집합 객체
s3 = s1.union(s2)         # 합집합
s4 = s1.intersection(s2)  # 교집합
s5 = s1 - s2              # 차집합

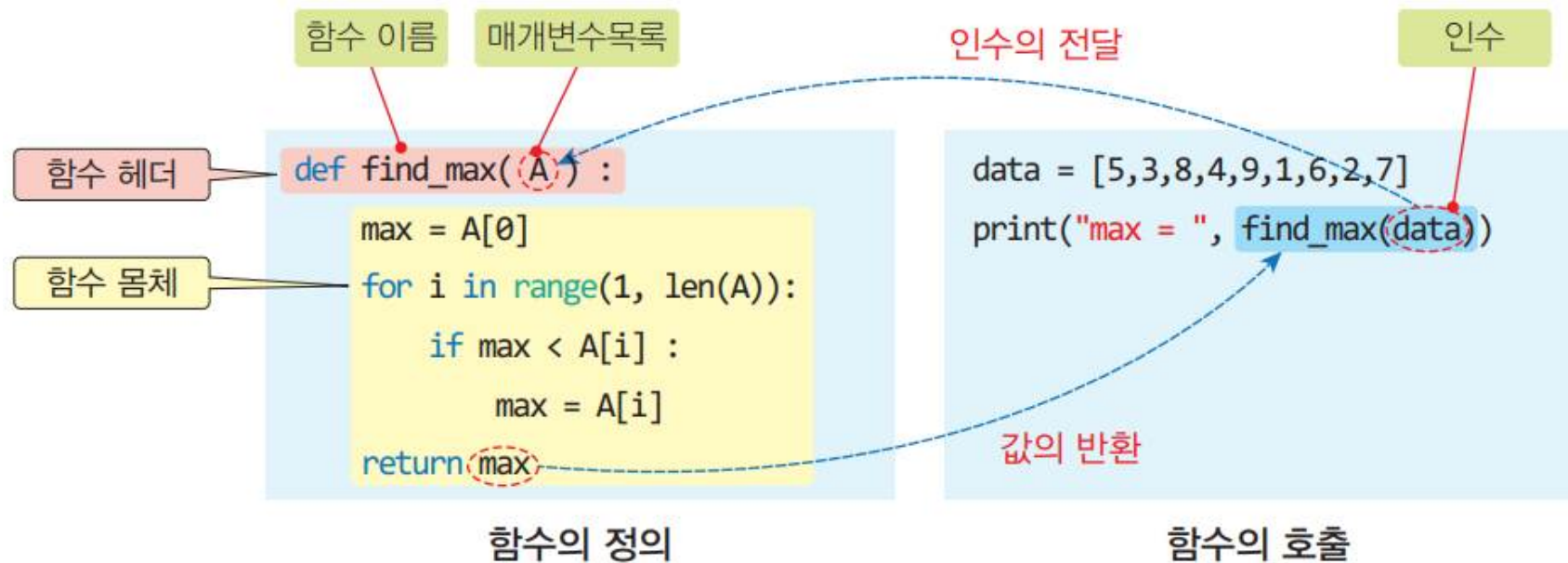
print("s1:", s1)
print("s2:", s2)
print("s3:", s3)
print("s4:", s4)
print("s5:", s5)
```

```
C:\#WIND...
s1: {1, 2, 3}
s2: {2, 3, 4, 5}
s3: {1, 2, 3, 4, 5}
s4: {2, 3}
s5: {1}
```

```
s5 = { 3.14 }           # 원소가 하나인 집합
map = { 3.14 : 'Phi' }  # 엔트리가 하나인 딕셔너리
```

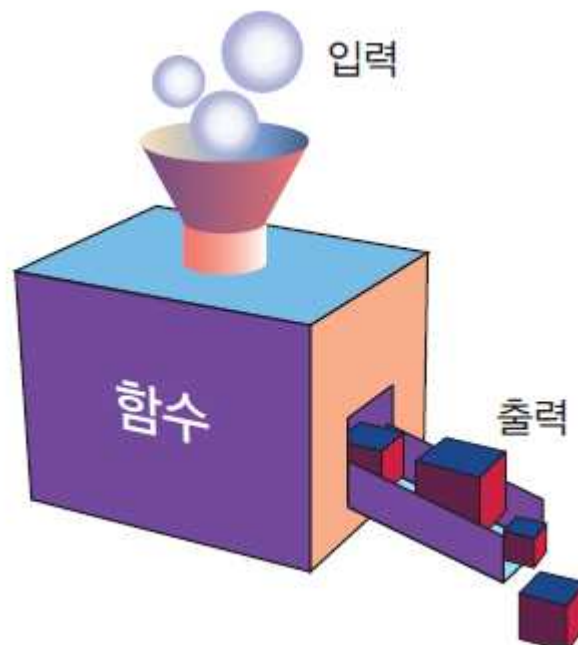
사용자 정의 함수

- 파이썬 내장 함수: type(), len(), ord() 등
- 사용자 정의 함수



함수란?

- 함수는 코드의 묶음에 이름을 붙인 것
- 함수는 입력을 받아서 출력을 내보내는 박스로 생각할 수 있다.



함수 작성하고 호출하기

함수 정의

```
def print_address():  
    print("서울특별시 종로구 1번지")  
    print("파이썬 빌딩 7층")  
    print("홍길동")
```

함수 호출

```
print_address()
```

서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동

함수의 장점

- 한 번만 함수를 정의하면 언제든지 필요할 때면 함수를 불러서 일을 시킬 수 있다.

```
print_address()  
print_address()  
print_address()
```

서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동
서울특별시 종로구 1번지
파이썬 빌딩 7층
홍길동

함수에 입력 전달하기

- 우리는 함수에 값(정보)을 전달할 수 있다. 이 값을 인수(argument)라고 한다.

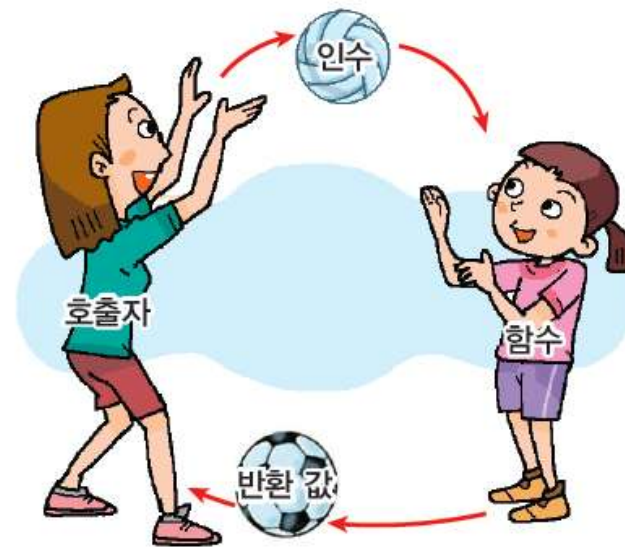


인수 전달

```
def print_address(name):  
    print("서울특별시 종로구 1번지")  
    print("파이썬 빌딩 7층")  
    print(name)  
  
print_address("홍길동")
```

값 반환하기

- 함수는 값을 반환할 수 있다.



값 반환

```
def calculate_area (radius):  
    area = 3.14 * radius**2  
    return area
```

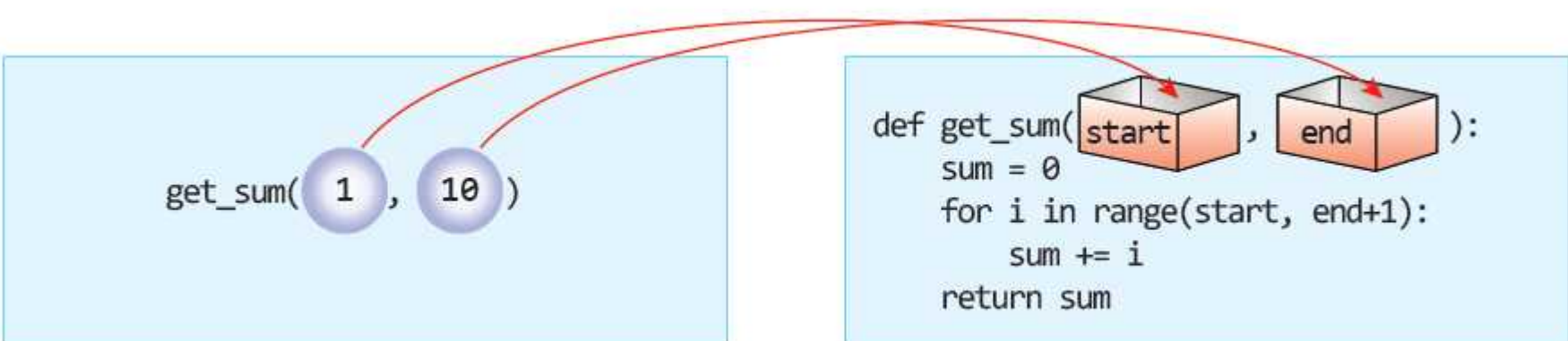
```
c_area = calculate_area(5.0)
```

함수에 여러 개의 입력 전달하기

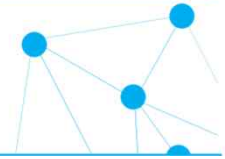
```
def get_sum(start, end):  
    sum = 0  
    for i in range(start, end+1):  
        sum += i  
    return sum
```

```
print(get_sum(1, 10))
```

get_sum(1 , 10)



```
def get_sum( start , end ):  
    sum = 0  
    for i in range(start, end+1):  
        sum += i  
    return sum
```



- 여러 개의 값의 반환

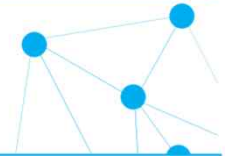
```
def find_min_max(A) :  
    min = A[0]  
    max = A[0]  
    for i in range(1, len(A)) :  
        if max < A[i] : max = A[i]  
        if min > A[i] : min = A[i]  
    return min, max
```

최댓값과 최솟값을 동시에 찾아 반환
i : 1 ~ len(A)-1
최댓값 갱신
최솟값 갱신
최솟값과 최댓값을 반환

```
data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]  
x, y = find_min_max(data)  
print("(min,max) = ", (x,y))
```

최솟값과 최댓값을 반환받음
x와 y를 튜플로 만들어 출력

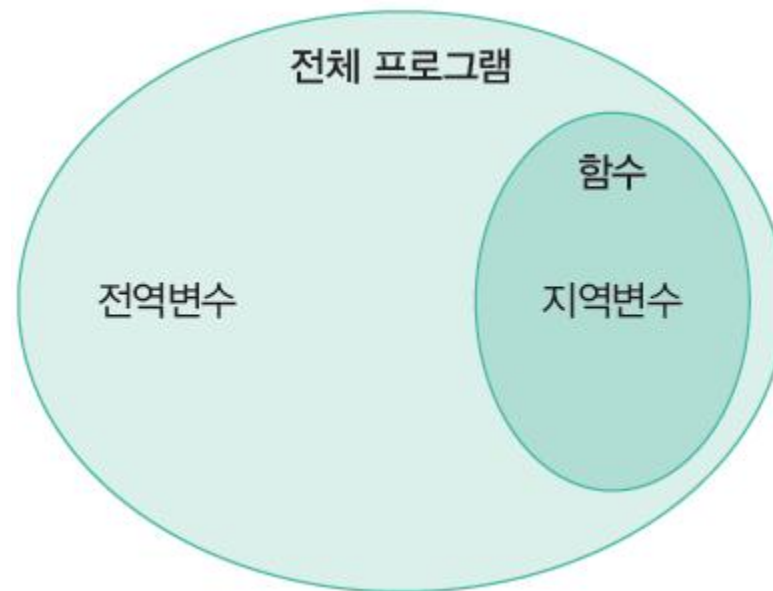
변수의 범위



- 전역 범위(global scope)
 - 소스 파일의 맨 꼭대기 레벨(함수나 클래스 밖)에서 생성
 - 프로그램의 어디에서나 사용할 수 있다.
- 지역 범위(local scope)
 - 함수나 클래스의 멤버함수(메소드) 안에서 생성
 - 그 안에서만 사용. 함수의 매개변수들도 지역범위

변수의 종류

- 지역 변수(local variable): 함수 안에서 선언되는 변수
- 전역 변수(global variable): 함수 외부에서 선언되는 변수



지역 변수의 범위

- 지역 변수는 함수 안에서만 사용이 가능하다.
- 아래의 코드에서 지역 변수를 찾아보자.

```
def calculate_area(radius):  
    result = 3.14 * radius**2  
    return result
```

```
r = float(input("원의 반지름: "))  
area = calculate_area(r)  
print(result)
```

오류가 없을까?

지역 변수의 범위

```
def calculate_area (radius):  
    result = 3.14 * radius**2  
    return result  
  
r = float(input("원의 반지름: "))  
area = calculate_area(r)  
print(result)
```

원의 반지름: 10

Traceback (most recent call last):

File "C:\Users\sec\AppData\Local\Programs\Python\Python35-32\z.py", line 7, in <module>

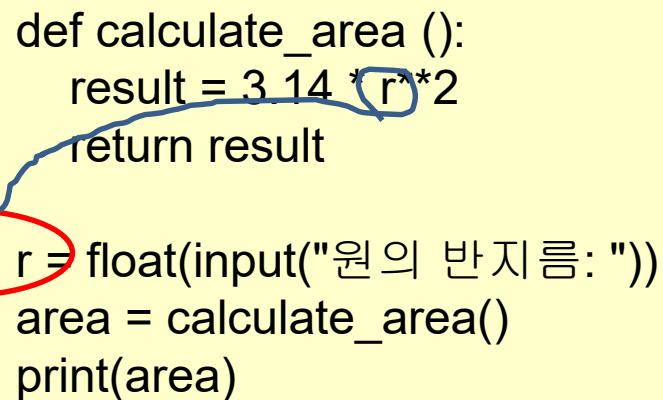
print(result)

NameError: name 'result' is not defined

전역 변수

- 전역 변수는 어디서나 사용할 수 있다.
- 아래의 코드에서 전역 변수를 찾아보자.

```
def calculate_area ():  
    result = 3.14 * r**2  
    return result  
  
r = float(input("원의 반지름: "))  
area = calculate_area()  
print(area)
```



함수 안에서 전역 변수 변경하기

```
def calculate_area (radius):  
    area = 3.14 * radius**2 # 전역변수 area에 계산값을 저장하려고 했다!  
    return
```

```
area = 0  
r = float(input("원의 반지름: "))  
calculate_area(r)  
print(area)
```

원의 반지름: 10

0

>>>

왜 0이 나올까?

함수 안에서 전역 변수 변경하기

```
def calculate_area (radius):  
    area = 3.14 * radius**2 # 전역변수 area에 계산값을 저장하려고 했다!  
    return  
  
area = 0  
r = float(input("원의 반지름: "))  
calculate_area(r)  
print(area)
```

여기서 새로운 지역 변수 area가 생성된다.

함수 안에서 전역 변수 변경하기

- `global`을 사용하여 전역 변수에 값을 저장한다고 알려야 한다.

```
def calculate_area (radius):  
    global area  
    area = 3.14 * radius**2  
    return  
  
area = 0  
r = float(input("원의 반지름: "))  
calculate_area(r)  
print(area)
```

디폴트 인수

- 파이썬에서는 함수의 매개변수가 기본값을 가질 수 있다. 이것을 디폴트 인수(default argument)라고 한다.

```
def greet(name, msg="별일없죠?):  
    print("안녕 ", name + ', ' + msg)  
  
greet("영희")
```

```
안녕 영희, 별일없죠?  
>>>
```

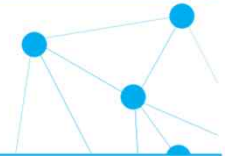
키워드 인수

- 키워드 인수는 인수의 이름을 명시적으로 지정해서 전달하는 방법이다.

```
def calc(x, y, z):  
    return x+y+z
```

```
>>> calc(y=20, x=10, z=30)  
60
```


모듈과 이름 공간(namespace)



```
# 파일명: min_max.py
def find_min_max(A) :
    ...
    return min, max
```

다음 함수가 min_max.py에 저장되어 있음
최댓값과 최솟값을 동시에 찾아 반환
최솟값과 최댓값을 반환

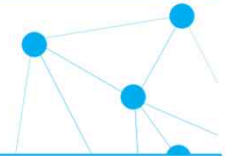
```
# 파일명: sum.py
def sum_range(begin, end, step=1) :
    ...
    return sum
```

다음 함수가 sum.py에 저장되어 있음
매개변수 step이 기본 값을 가짐

```
# 파일명: my_job.py
import min_max
import sum

data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]
print("(min,max) = ", min_max.find_min_max(data))
print("sum = ", sum.sum_range(1, 10))
```

min_max.py 모듈을 사용함
sum.py 모듈을 사용함
min_max 모듈의 함수 사용
sum 모듈의 함수 사용

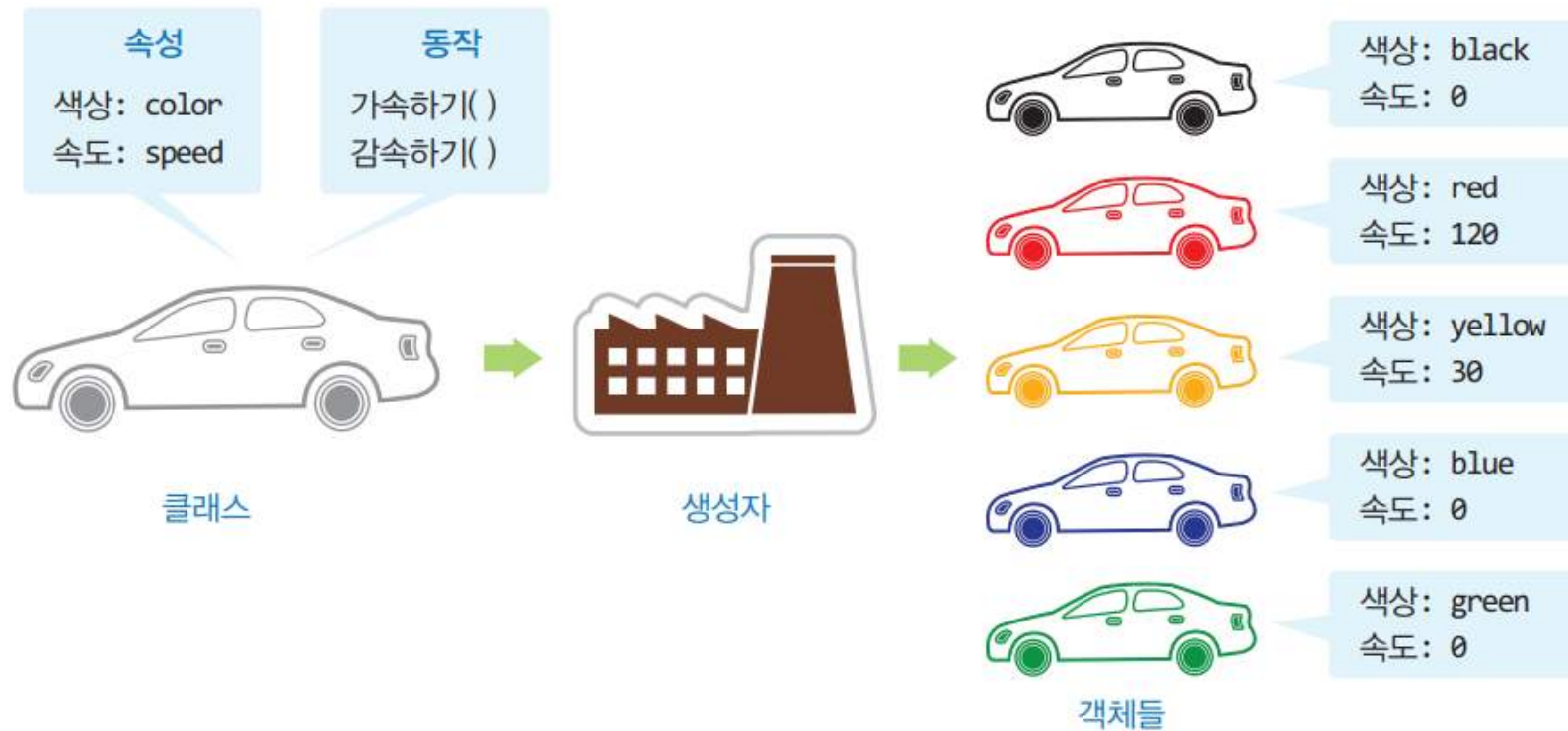
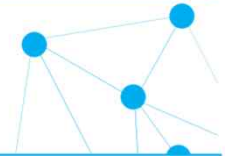


```
from math import pow, sqrt      # math 모듈에서 pow, sqrt 식별자를 사용
result = pow(2,10)              # pow 식별자를 바로 사용 가능
dist = sqrt(1000)              # sqrt 식별자를 바로 사용 가능
```

```
# 파일명: my_job.py
from min_max import *          # min_max 모듈의 모든 식별자 사용 가능
from sum import *              # sum 모듈의 모든 식별자 사용 가능

data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7 ]
print("(min,max) = ", find_min_max(data))    # 바로 사용
print("sum = ", sum_range(1, 10))           # 바로 사용
```

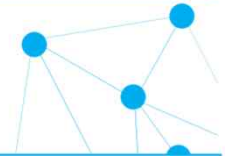
클래스



- 클래스 정의

```
class Car :
```

```
# 자동차와 관련된 클래스 정의 블록이 이어짐. 들여쓰기에 유의할 것
```

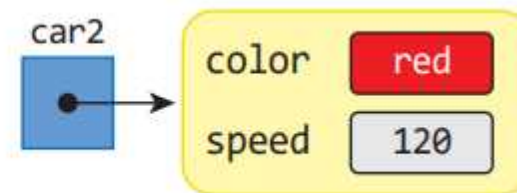
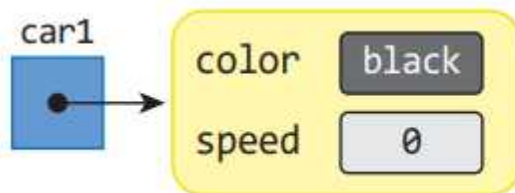


- 생성자

```
class Car :  
    def __init__(self, color, speed = 0) :  
        self.color = color  
        self.speed = speed
```

생성자 함수
데이터 멤버 color 정의 및 초기화
데이터 멤버 speed 정의 및 초기화

```
car1 = Car('black', 0)      # 검정색, 속도 0  
car2 = Car('red', 120)     # 빨간색, 속도 120  
car3 = Car('yellow', 30)   # 노란색, 속도 30  
car4 = Car('blue', 0)      # 파랑색, 속도 0  
car5 = Car('green')        # 초록색, 속도 0 (디폴트 인수 사용)
```



객체 지향 프로그래밍

- 객체(object)는 함수와 변수를 하나의 단위로 묶을 수 있는 방법이다. 이러한 프로그래밍 방식을 객체지향(object-oriented)이라고 한다.



객체란?

- 객체는 하나의 물건이라고 생각하면 된다. 객체는 속성(attribute)과 동작(action)을 가지고 있다.

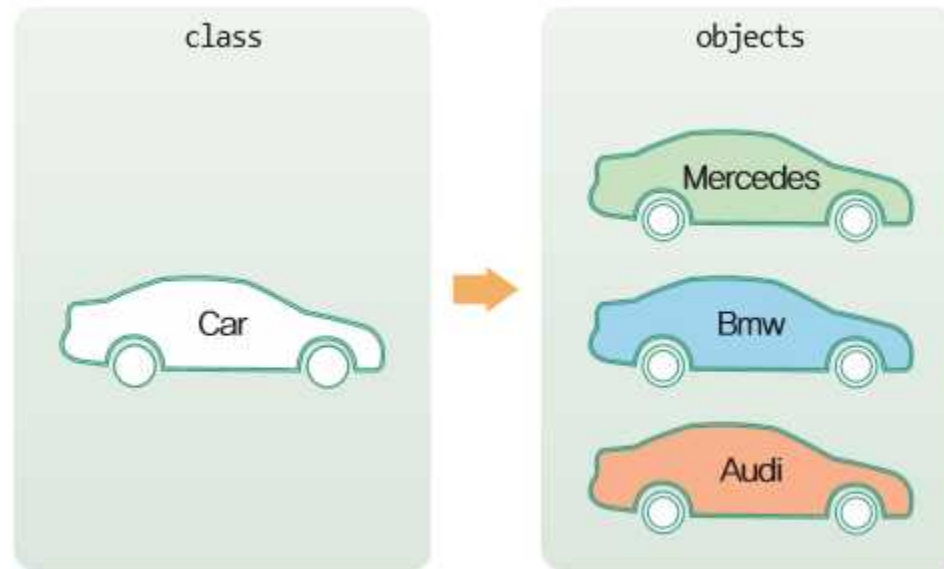
속성
메이커
모델
색상
연식
가격



동작
주행하기
방향바꾸기
주차하기

객체 생성하기

1. 객체의 설계도를 작성하여야 한다.
2. 클래스로부터 객체를 생성하여야 한다.



객체 생성 코드

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

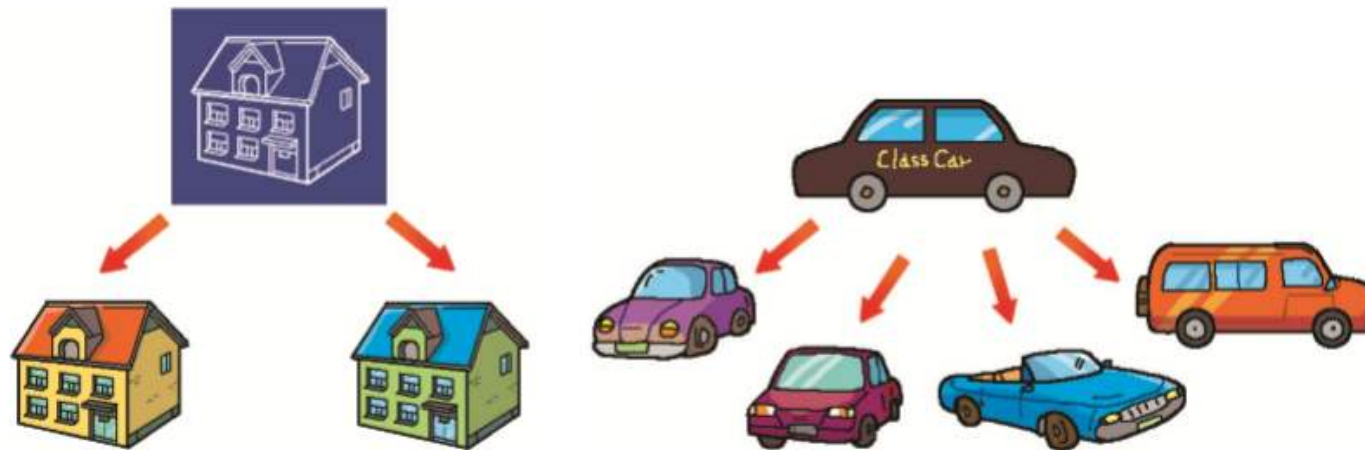
myCar = Car(0, "blue", "E-class")

print("자동차 객체를 생성하였습니다.")
print("자동차의 속도는", myCar.speed)
print("자동차의 색상은", myCar.color)
print("자동차의 모델은", myCar.model)
print("자동차를 주행합니다.")
myCar.drive()
print("자동차의 속도는", myCar.speed)
```

자동차 객체를 생성하였습니다.
자동차의 속도는 0
자동차의 색상은 blue
자동차의 모델은 E-class
자동차를 주행합니다.
자동차의 속도는 60

하나의 클래스로 객체는 많이 만들 수 있다.

- 우리는 하나의 클래스로 여러 개의 객체를 생성할 수 있다



하나의 클래스로 객체는 많이 만들 수 있다.

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60

dadCar = Car(0, "silver", "A6")
momCar = Car(0, "white", "520d")
myCar = Car(0, "blue", "E-class")
```

self는 무엇인가?

```
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model

    def drive(self):
        self.speed = 60c

myCar = Car(0, "blue", "E-class")
yourCar = Car(0, "white", "S-class")

myCar.drive()
yourCar.drive()
```

```

class 클래스이름:
    def __init__(self, 인자1, ...): # 객체 생성자
        self.인스턴스 변수1 = 인자1
        :
    def
        :
    def

```

객체에 대한 메소드

- **인스턴스 변수**: 객체에 정보를 저장
- **객체 생성자(Constructor)**: 객체를 생성
- **메소드**: 객체 혹은 객체 내부 인스턴스 변수에 대한 연산 수행

[예제] 학생을 객체로 표현하기 위해 Student 클래스

- 각 Student 객체는 이름과 학번을 각각 저장하는 인스턴스 변수 name과 id를 가진다고 가정

```
01 class Student:
02     def __init__(self, name, id):
03         self.name = name
04         self.id = id
05     def get_name(self):
06         return self.name
07     def get_id(self):
08         return self.id
09
10 best = Student('Lee', 101)
11 print(best.get_name())
12 print(best.get_id())
```


Student 객체
생성자

인스턴스 변수

객체의 name을
리턴하는 메소드

객체의 id를
리턴하는 메소드

best 학생의
name과 id 출력

Console  PyUnit

<terminated> main.py [C:\Users\wsbyang\AppData\Local\Programs\Python\Python36-32\

Lee

101

[프로그램 1-1]의 수행 결과

파이썬 리뷰 끝