

[Opensource Basic Project]

# 오픈소스 기초 프로젝트

- 6주차: Classes

2024. 4. 17(수).

우창우

[Dr.woo@chungbuk.ac.kr](mailto:Dr.woo@chungbuk.ac.kr)

# □ 주차별 강의계획

※ 4.10(국회의원선거날)

주차(변경 전)	주차(변경 후)	강의계획	과제	번호	제출 마감일
1주(03.06)	1주(03.06)	Python Started	VScode 설치화면, "Hello World" 출력화면 및 작성코드	1	03.06~03.09(토)
2주(03.13)	2주(03.13)	Variables and simple data types	다이아몬드 모양, 진수 변환 프로그램 및 동전 변환 프로그램의 출력화면/작성코드	2	03.13~03.16(토)
3주(03.20)	3주(03.20)	Control Statement	종합계산기, 구구단출력 프로그램의 출력화면/작성코드	3	03.20~03.23(토)
4주(03.27)	4주(03.27)	Lists, Dictionaries, String	음식 궁합 출력, 문자열 거꾸로 출력 프로그램의 출력화면/작성코드	4	03.27~03.30(토)
5주(04.03)	5주(04.03)	Functions and module	GitHub Copilot Extension 설치화면, 로또번호 추첨 출력화면/작성코드	5	04.03~04.06(토)
6주(04.10)	6주(04.17)	Classes	팀프로젝트 과제 제출양식.hwp (조당 1개)	6	04.17~04.20(토)
7주(04.17)	7주(04.24)	Window programming		7	04.24~04.27(토)
8주(04.24)	8주(05.01)	Midterm (20%)		-	-

# □ 주차별 강의계획

※5.15(부처님 오신 날)

주차(변경 전)	주차(변경 후)	강의계획	과제	번호	제출 마감일
9주(05.01)	9주(05.08)	Files and Exceptions		8	05.08~05.11(토)
10주(05.08)	10주(05.22)	Database		9	05.22~05.25(토)
11주(05.15)	11주(05.29)	Data visualization		10	05.29~06.01(토)
12주(05.22)	12주(06.05)	Quiz (10%), Team project implementation I		-	-
13주(05.29)	13주(06.12) 또는 (06.17)	Team project implementation II		-	-
14주(06.05)	14주(06.17) 또는 (06.18)	Team project implementation III		-	-
15주(06.12)	15주(06.18) 또는 (06.20)	Project Presentation (50%)		-	-

# **Chapter 12. 객체지향 프로그래밍**

**Selection 01. 객체지향 프로그래밍**

**Selection 02. 클래스**

**Selection 03. 생성자**

**Selection 04. 인스턴스 변수와 클래스 변수**

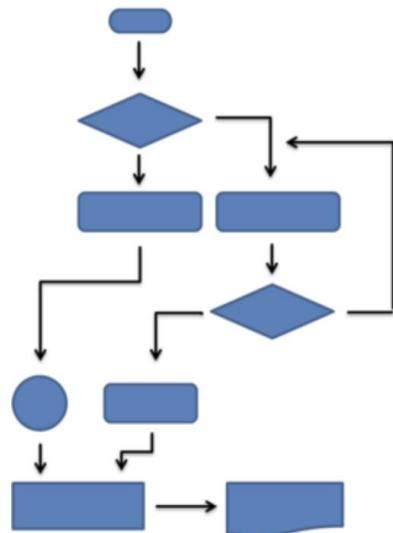
**Selection 05. 클래스의 상속**

**Selection 06. 객체지향 프로그래밍의 심화 내용**

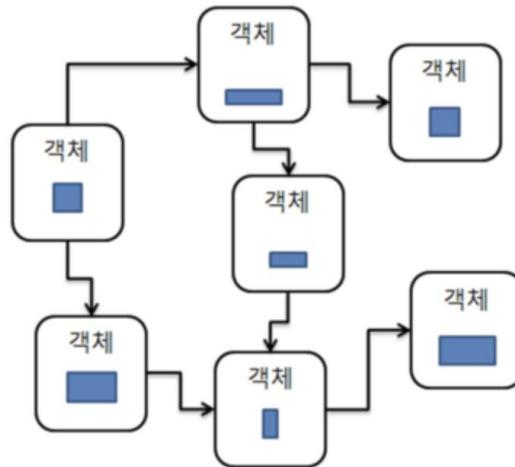
# Selection 01. 객체지향 프로그래밍

## 1. 절차지향(Procedure Oriented) 프로그래밍과 객체지향(Object Oriented) 프로그래밍

- 절차지향 프로그래밍은 순서대로 흐름을 따라가는 프로그래밍 방법
- 객체지향 프로그래밍은 개별적으로 참조되고 호출되는 변수와 함수를 '객체'로 묶어 순서대로 진행하는 프로그래밍 방법



〈 절차지향 프로그래밍 순서도 〉



〈 객체지향 프로그래밍 순서도 〉

# Selection 02. 클래스

## 1. 클래스의 개념

- 파이썬은 객체지향 개념을 적용할 수 있는 객체지향 프로그래밍 (Object Oriented Programming) 언어에 해당
- 객체지향 프로그래밍 언어에서 가장 핵심은 클래스(Class)로 아래와 같이 구현됨

```
class 클래스명 :  
    # 이 부분에 관련 코드 구현
```

- 클래스는 '현실 세계의 사물을 컴퓨터로 구현하기 위해 고안된 개념'으로 이해
- (예시) 자동차를 클래스로 구현하는 개념



```
class 자동차 :  
    # 자동차의 속성  
    색상  
    속도  
    # 자동차의 기능  
    속도 올리기()  
    속도 내리기()
```

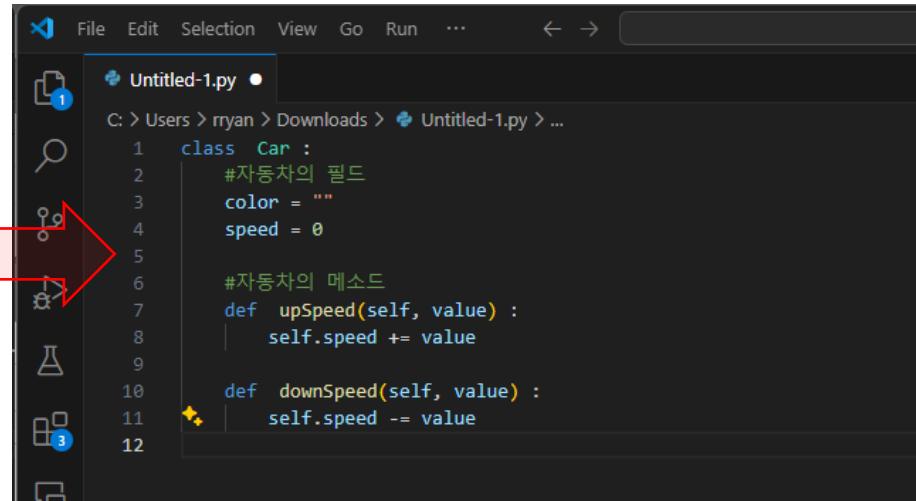
〈자동차를 클래스로 구현〉

# Selection 02. 클래스

## 1. 클래스의 개념

- (예시) 자동차 클래스의 개념을 코드로 구현하는 예시

```
class Car :  
    # 자동차의 필드  
    색상 = ""  
    현재_속도 = 0  
  
    # 자동차의 메서드  
    def upSpeed(증가할_속도량) :  
        # 현재 속도에서 증가할_속도량만큼 속도를 올리는 코드  
  
    def downSpeed(감소할_속도량) :  
        # 현재 속도에서 감소할_속도량만큼 속도를 내리는 코드
```



```
File Edit Selection View Go Run ...  
Untitled-1.py ●  
C: > Users > rryan > Downloads > Untitled-1.py > ...  
1 class Car :  
2     # 자동차의 필드  
3     color = ""  
4     speed = 0  
5  
6     # 자동차의 메서드  
7     def upSpeed(self, value) :  
8         self.speed += value  
9  
10    def downSpeed(self, value) :  
11        self.speed -= value  
12
```

- 자동차의 속성은 지금까지 사용 한 변수처럼 생성 (필드 (field))
- 자동차의 기능은 지금까지 사용 한 함수 형식 (def 함수명(매개변수):)으로 구현
- 다만, 클래스 안에서 구현된 함수는 함수라고 하지 않고 메서드 (Method) 라고 함

# Selection 02. 클래스

## 1. 클래스의 개념

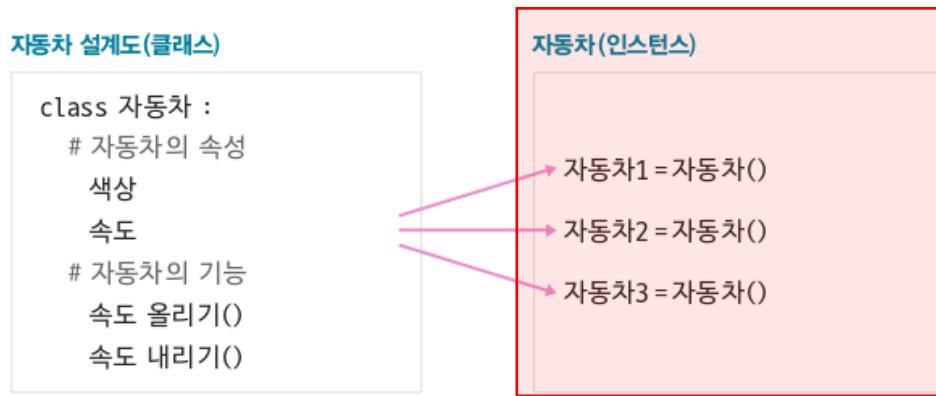
- 인스턴스는 객체와 같은 개념으로 클래스를 만든 후 인스턴스를 여러 개 생성 할 수 있음
- (예시) 클래스와 인스턴스의 개념



# Selection 02. 클래스

## 1. 클래스의 개념

- 인스턴스 구현 형식



〈클래스와 인스턴스 코드 구성〉

- 자동차 세 대의 인스턴스 생성 코드

A screenshot of a code editor showing a file named "Untitled-1.py". The code defines a class "Car" and creates three instances: "myCar1", "myCar2", and "myCar3". A red box highlights the instantiation code (lines 13-15), and a red arrow points from the explanatory text above to this highlighted area.

```
File Edit Selection View Go Run ...
Untitled-1.py
C:\Users\rryan\Downloads\Untitled-1.py...
1 > class Car : ...
12
13     myCar1 = Car()
14     myCar2 = Car()
15     myCar3 = Car()
16
17
```

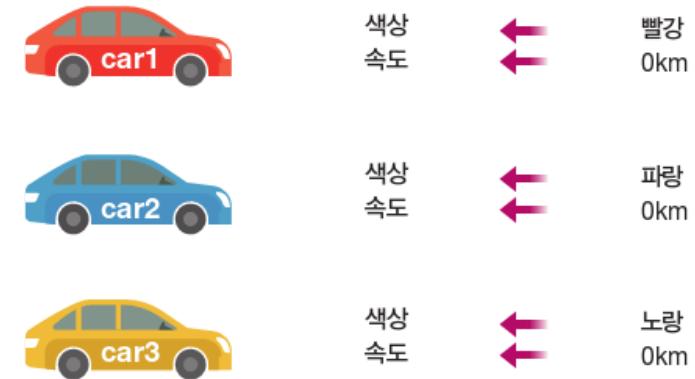
# Selection 02. 클래스

## 1. 클래스의 개념

- 인스턴스가 생성되면 별도의 값 대입이 가능. 필드의 경우 '인스턴스명.필드명' 형식으로 표현 할 수 있고, 메서드의 경우 '인스턴스명.메서드名' 형식으로 표현이 가능

```
File Edit Selection View Go Run ... ← → ⌘ Untitled-1.py X  
C: > Users > rryan > Downloads > Untitled-1.py > ...  
1 > class Car :  
2  
3     myCar1 = Car()  
4     myCar2 = Car()  
5     myCar3 = Car()  
6  
7     myCar1.color = "빨강"  
8     myCar1.speed = 0  
9     myCar2.color = "파랑"  
10    myCar2.speed = 0  
11    myCar3.color = "노랑"  
12    myCar3.speed = 0  
13  
14    myCar1.upSpeed(30)  
15    myCar2.downSpeed(60)
```

메서드의 호출



〈인스턴스의 필드와/메서드에 각각 대응〉

# Selection 02. 클래스

## 2. 클래스의 완전한 작동 구현

- (예시) 자동차 클래스의 개념을 코드로 구현하는 예시

The screenshot shows two side-by-side code editors in VS Code. The left editor contains the following Python code:

```
C: > Users > rryan > Downloads > Untitled-1.py > Car
1  class Car :
2      ## 자동차의 필드 ##
3      color = ""
4      speed = 0
5
6      ## 자동차의 메소드 ##
7      def upSpeed(self, value) :
8          self.speed += value
9
10     def downSpeed(self, value) :
11         self.speed -= value
12
13 myCar1 = Car()
14 myCar2 = Car()
15 myCar3 = Car()
16
17 myCar1.color = "빨강"
18 myCar1.speed = 0
19 myCar2.color = "파랑"
20 myCar2.speed = 0
21 myCar3.color = "노랑"
22 myCar3.speed = 0
23
```

The right editor shows the execution of this code, printing the state of three car instances:

```
C: > Users > rryan > Downloads > Untitled-1.py > ...
23
24 myCar1.upSpeed(30)
25 print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
26
27 myCar2.upSpeed(60)
28 print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
29
30 myCar3.upSpeed(0)
31 print("자동차3의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar3.color, myCar3.speed))
32
```

Below the code editors is a table illustrating the three-step instantiation process:

단계	작업	형식	예
1단계	클래스 선언	class 클래스명: # 필드 선언 # 메서드 선언	class Car : color = "" def upSpeed(self, value) : ...
2단계	인스턴스 생성	인스턴스 = 클래스명()	myCar1 = Car()
3단계	필드나 메서드 사용	인스턴스.필드명 = 값 인스턴스.메서드()	myCar1.color = "빨강" myCar1.upSpeed(30)

# Selection 03. 생성자

## 1. 생성자의 개념

- 생성자(Constructor)는 인스턴스를 생성하면 무조건 호출되는 메서드로, 인스턴스를 생성하면서 필드 값을 초기화 시켜주는 메서드
- 생성자의 기본형태 : `__init__()` ※ Initialize

```
class 클래스명 :  
    def __init__(self) :  
        # 이 부분에 초기화할 코드 입력
```

- (예시) 자동차 클래스 예제에 생성자를 적용
  - `myCar1` 인스턴스를 생성함과 동시에 `color="빨강"`, `speed=0`으로 초기화

```
## 메인 코드 부분 ##  
myCar1 = Car()  
myCar1.color = "빨강"  
myCar1.speed = 0
```

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

# Selection 03. 생성자

## 1. 생성자의 개념

- 매개변수가 self만 있는 기본 생성자
- 매개변수가 있는 생성자

```
File Edit Selection View Go Run ... ← → Untitled-1.py X C: > Users > rryan > Downloads > Untitled-1.py > ... 1 ## 클래스 선언 부분 ## 2 class Car : 3     color = "" 4     speed = 0 5 6     def __init__(self) : 7         self.color = "빨강" 8         self.speed = 0 9 10    def upSpeed(self, value) : 11        self.speed += value 12 13    def downSpeed(self, value) : 14        self.speed -= value 15 16    ## 메인 코드 부분 ## 17    myCar1 = Car() 18    myCar2 = Car() 19 20    print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.col 21    print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.col 22 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PS C:\Users\rryan> & C:/Users/rryan/AppData/Local/Programs/Python/Python312/python 자동차1의 색상은 빨강이며, 현재 속도는 0km입니다. 자동차2의 색상은 빨강이며, 현재 속도는 0km입니다. PS C:\Users\rryan>
```

```
File Edit Selection View Go Run ... ← → Untitled-1.py X C: > Users > rryan > Downloads > Untitled-1.py > ... 1 ## 클래스 정의 부분 ## 2 class Car : 3     color = "" 4     speed = 0 5 6     def __init__(self, value1, value2) : 7         self.color = value1 8         self.speed = value2 9 10    def upSpeed(self, value) : 11        self.speed += value 12 13    def downSpeed(self, value) : 14        self.speed -= value 15 16    ## 메인 코드 부분 ## 17    myCar1 = Car("빨강", 30) 18    myCar2 = Car("파랑", 60) 19 20    print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.col 21    print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.col 22 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PS C:\Users\rryan> & C:/Users/rryan/AppData/Local/Programs/Python/Python312/python 자동차1의 색상은 빨강이며, 현재 속도는 30km입니다. 자동차2의 색상은 파랑이며, 현재 속도는 60km입니다. PS C:\Users\rryan>
```

# Selection 04. 인스턴스 변수와 클래스 변수

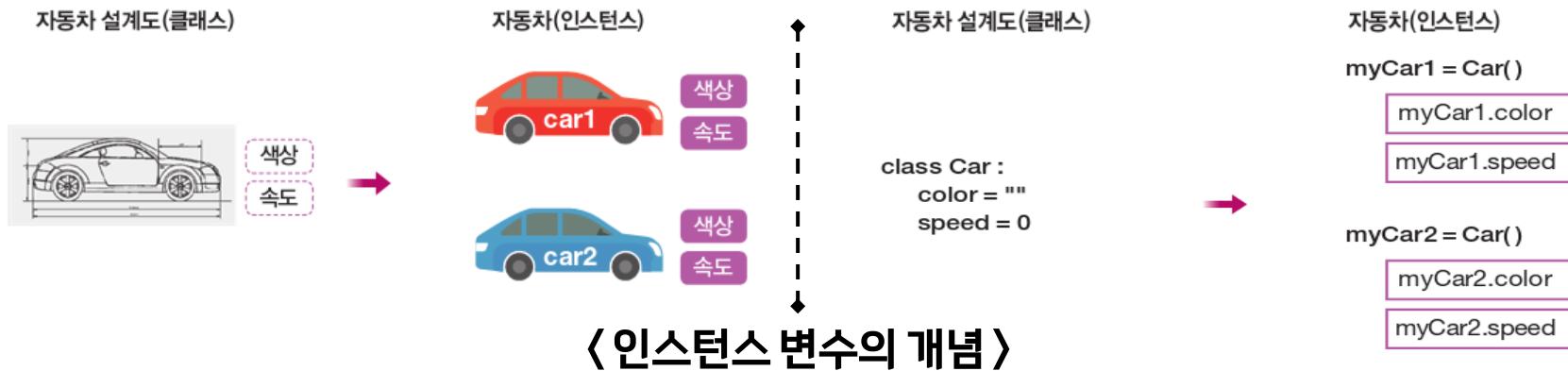
## 1. 인스턴스 변수

- 인스턴스 변수는 인스턴스를 생성해야 사용할 수 있는 변수

```
class Car :  
    color = ""    # 필드 : 인스턴스 변수  
    speed = 0    # 필드 : 인스턴스 변수
```

- 인스턴스 변수는 구현되기 전 까지 실제 공간이 할당되지 않음

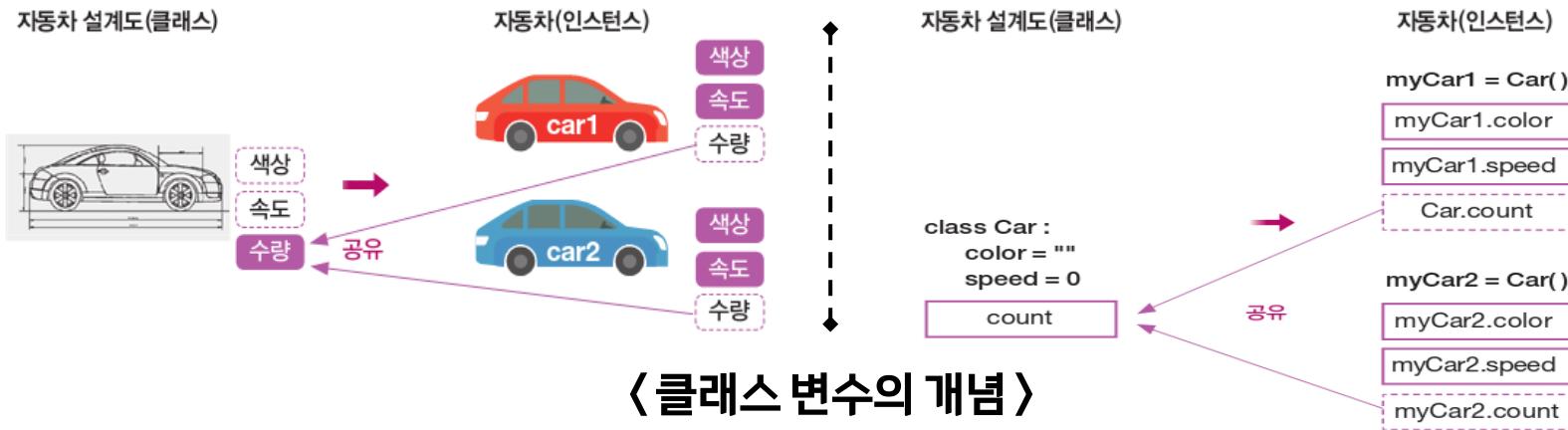
```
myCar1 = Car()  
myCar2 = Car()
```



# Selection 04. 인스턴스 변수와 클래스 변수

## 2. 클래스 변수

- 클래스 안에 공간이 할당된 변수로 여러 인스턴스가 클래스 변수 공간을 함께 사용



- 클래스 변수를 만드는 방법은 인스턴스 변수와 동일하지만, 접근할 때는 '클래스명.클래스변수명' 또는 '인스턴스.클래스변수명' 방식으로 접근해야 함

## Selection 04. 인스턴스 변수와 클래스 변수

## 2. 클래스 변수

- #### ▪ (예시) 자동차 생산 대수를 확인하기 위한 코드 구현의 예시

The screenshot shows a Python script named Untitled-1.py in a code editor. The code defines a Car class with attributes color, speed, and count. It includes an \_\_init\_\_ method that initializes speed to 0 and increments count. Below the class definition, there's sample code creating two instances of Car and printing their speeds and counts.

Annotations highlight specific parts of the code:

- A red box surrounds the assignment statement `self.speed = 0` in the \_\_init\_\_ method, with a callout pointing to the text "Self를 붙이면 인스턴스 변수, 클래스명을 붙이면 클래스 변수가 됨".
- A red box surrounds the expression `Car.count` in the print statement, with a callout pointing to the text "클래스명.클래스변수명".
- A red box surrounds the expression `myCar2.count` in the print statement, with a callout pointing to the text "인스턴스명.클래스변수명".

At the bottom, the terminal window shows the execution of the script and its output.

```
File Edit Selection View Go Run ... ← → Search
C: > Users > rryan > Downloads > Untitled-1.py > Car
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = "" # 인스턴스 변수
4      speed = 0 # 인스턴스 변수
5      count = 0 # 클래스 변수
6
7      def __init__(self):
8          self.speed = 0
9          Car.count += 1
10
11 # 변수 선언
12 myCar1, myCar2 = None, None
13
14 # 메인 코드 부분
15 myCar1 = Car()
16 myCar1.speed = 30
17 print("자동차1의 현재 속도는 %dkm, 생산된 자동차는 총 %d대입니다." %(myCar1.speed, Car.count))
18
19 myCar2 = Car()
20 myCar2.speed = 60
21 print("자동차2의 현재 속도는 %dkm, 생산된 자동차는 총 %d대입니다." %(myCar2.speed, myCar2.count))

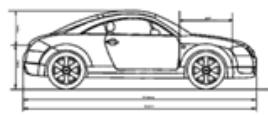
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\rryan> & C:/Users/rryan/AppData/Local/Programs/Python/Python312/python.exe c:/Users/rryan/Downloads/Untitled-1.py
자동차1의 현재 속도는 30km, 생산된 자동차는 총 1대입니다.
자동차2의 현재 속도는 60km, 생산된 자동차는 총 2대입니다.
```

# Selection 05. 클래스의 상속

## 1. 상속의 개념

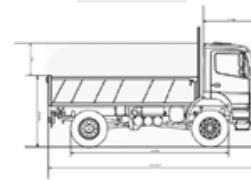
- **클래스의 상속(Inheritance)** : 기존 클래스에 있는 필드와 메서드를 그대로 물려받아 새로운 클래스를 만드는 것
- 상속받은 후에는 클래스에서 필드나 메서드를 추가로 만들 수 있음

승용차 클래스



class 승용차 :  
필드 – 색상, 속도, 좌석 수  
메서드 – 속도 올리기()  
속도 내리기()  
좌석 수 알아보기()

트럭 클래스



class 트럭 :  
필드 – 색상, 속도, 적재량  
메서드 – 속도 올리기()  
속도 내리기()  
적재량 알아보기()

〈승용차와 트럭 클래스의 개념〉

# Selection 05. 클래스의 상속

## 1. 상속의 개념

- 공통 내용을 자동차 클래스에 두고 상속을 받음으로써 일관적인 코딩 가능
- 상위 클래스인 '자동차 클래스'를 슈퍼 클래스(부모 클래스),  
하위 클래스인 '승용차 클래스'와 '트럭 클래스'를 서브 클래스(자식 클래스)라고 부름

```
class 자동차 :  
    필드 - 색상, 속도  
    메서드 - 속도 올리기(),  
              속도 내리기()
```

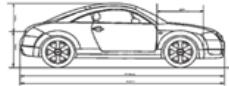
자동차 클래스(공통 내용)



상속

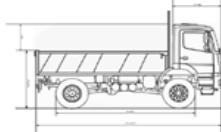
상속

승용차 클래스



```
class 승용차 : 자동차 상속  
    필드 - 자동차 필드, 차석 수  
    메서드 - 자동차 메서드  
              차석 수 알아보기()
```

트럭 클래스



```
class 트럭 : 자동차 상속  
    필드 - 자동차 필드, 적재량  
    메서드 - 자동차 메서드  
              적재량 알아보기()
```

### ▪ 상속을 구현하는 문법

```
class 서브_클래스(슈퍼_클래스) :
```

# 이 부분에 서브 클래스의 내용 코딩

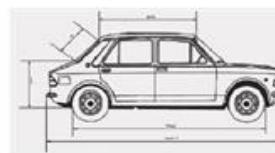
# Selection 05. 클래스의 상속

## 2. 메서드 오버라이딩(Overriding)

- 슈퍼 클래스에 있는 메서드를 서브 클래스에서 다시 만들어서(재정의) 사용하는 과정

class 자동차:  
메서드 – 속도 올리기()

자동차 클래스(공통 내용)



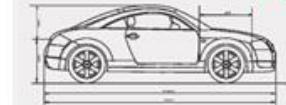
속도 올리기()

상속

상속

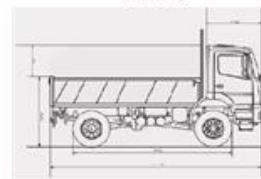
승용차 클래스

속도 올리기()



class 승용차(자동차) :  
메서드 – 속도 올리기() 재정의

트럭 클래스



class 트럭(자동차) :  
메서드 –

〈메서드 오버라이딩의 개념〉

# Selection 05. 클래스의 상속

## 2. 메서드 오버라이딩(Overriding)

The screenshot shows a Python code editor with two files: Untitled-1.py and Untitled-2.py.

**Untitled-1.py:**

```
## 클래스 선언 부분 ##
class Car :
    speed = 0
    def upSpeed(self, value):
        self.speed += value
        print("현재 속도(슈퍼 클래스) : %d" % self.speed)

class Sedan(Car):
    def upSpeed(self, value):
        self.speed += value
        if self.speed > 150:
            self.speed = 150
        print("현재 속도(서브 클래스) : %d" % self.speed)

class Truck(Car):
    pass
```

**Untitled-2.py:**

```
## 변수 선언 부분 ##
sedan1, truck1 = None, None

## 메인 코드 부분 ##
truck1 = Truck()
sedan1 = Sedan()

print("트럭 --> ", end="")
truck1.upSpeed(200)
print("승용차 --> ", end="")
sedan1.upSpeed(200)
```

The code demonstrates inheritance where the `Sedan` class inherits from the `Car` class. It also shows method overriding where the `upSpeed` method is redefined in the `Sedan` class to limit the speed to 150. The `Truck` class is defined but contains only a placeholder `pass`. The `main` code creates instances of `Truck` and `Sedan` and calls their `upSpeed` methods with a value of 200.

# Selection 06. 객체지향 프로그래밍의 심화 내용

## 1. 클래스의 특별한 메서드

- `__init__()` 메서드 : 인스턴스(객체)가 생성할 때 자동으로 실행되는 메서드
- `__del__()` 메서드 : 소멸자(Destructor)로 인스턴스를 삭제할 때 자동 호출
- `__repr__()` 메서드 : 인스턴스를 `print()` 문으로 출력할 때 실행
- `__add__()` 메서드 : 인스턴스 사이에 덧셈 작업이 일어날 때 실행
- **두 개의 클래스를 비교 할 때 쓰는 메서드**

메서드명	의미	비고
<code>__lt__()</code>	Less than	<
<code>__le__()</code>	Less or equal	<code>≤</code>
<code>__gt__()</code>	Great than	<code>&gt;</code>
<code>__ge__()</code>	Great or equal	<code>≥</code>
<code>__eq__()</code>	Equal	<code>==</code>
<code>__ne__()</code>	Not equal	<code>!=</code>

## Selection 06. 객체지향 프로그래밍의 심화 내용

## 1. 클래스의 특별한 메서드

# Selection 06. 객체지향 프로그래밍의 심화 내용

## 2. 추상 메서드

- 서브 클래스에서 메서드를 오버라이딩 하는 경우, 슈퍼 클래스에서는 빈 껍질의 메서드만 만들어 놓고 내용은 pass로 채움으로써 추상 메서드를 구현

The image shows two side-by-side screenshots of the Visual Studio Code (VS Code) interface, both displaying Python code in files named 'Untitled-1.py'.

**Left Window (Original Code):**

```
C: > Users > rryan > Downloads > Untitled-1.py > ...
1 class SuperClass :
2     def method(self) :
3         pass
4
5 class SubClass1(SuperClass) :
6     def method(self) :
7         print("SubClass1에서 method()를 오버라이딩함")
8
9 class SubClass2(SuperClass) :
10    pass
11
12 ## 메인 코드 부분 ##
13 sub1 = SubClass1()
14 sub2 = SubClass2()
15
16 sub1.method()
17 sub2.method()
```

**Right Window (Modified Code with Error):**

```
C: > Users > rryan > Downloads > Untitled-1.py > ...
1 class SuperClass :
2     def method(self) :
3         raise NotImplementedError("SubClass에서 오버라이딩 해야 함")
4
5 class SubClass1(SuperClass) :
6     def method(self) :
7         print("SubClass1에서 method()를 오버라이딩함")
8
9 class SubClass2(SuperClass) :
10    pass
```

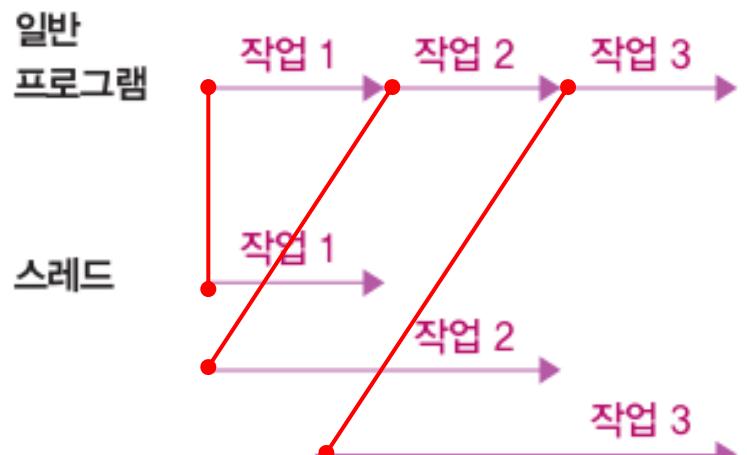
The right window's terminal output shows an error:

```
PS C:\Users\rryan> & C:/Users/rryan/AppData/Local/Programs/Python/Python312/python Untitled-1.py
SubClass1에서 method()를 오버라이딩함
Traceback (most recent call last):
  File "c:/Users/rryan/Downloads/Untitled-1.py", line 17, in <module>
    sub2.method()
  File "c:/Users/rryan/Downloads/Untitled-1.py", line 3, in method
    raise NotImplementedError("SubClass에서 오버라이딩 해야 함")
NotImplementedError: SubClass에서 오버라이딩 해야 함
PS C:\Users\rryan>
```

# Selection 06. 객체지향 프로그래밍의 심화 내용

## 3. 멀티 스레드/멀티 프로세싱

- 스레드(Thread)는 프로그램 하나에서 여러 개를 동시에 처리할 수 있도록 제공하는 기능으로, 멀티 스레드(Multi Thread)는 동시에 스레드 여러 개를 작동하는 것을 의미
- 파이썬에서 스레드는 내부적으로 CPU를 1개만 사용하지만, 멀티 프로세싱(Multi Processing)은 동시에 CPU 여러 개를 사용 가능하게 함



〈 일반 프로그램과 스레드의 차이 〉

# Selection 06. 객체지향 프로그래밍의 심화 내용

## 3. 멀티 스레드/멀티 프로세싱

〈자동차가 경주하는 코드의 예시〉

The screenshot shows a split-screen view of the Visual Studio Code interface. On the left, the code editor displays a file named 'Untitled-1.py' containing Python code for racing cars. On the right, the terminal window shows the execution of the code and its output.

**Code Editor (Untitled-1.py):**

```
C:\Users\rryan> C:/Users/rryan/AppData/Local/Programs/Python/Python312/python Untitled-1.py
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
PS C:\Users\rryan>
```

**Terminal Output:**

```
PS C:\Users\rryan> & C:/Users/rryan/AppData/Local/Programs/Python/Python312/python
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
@자동차1~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
#자동차2~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
$자동차3~~ 달립니다.
PS C:\Users\rryan>
```

# Selection 06. 객체지향 프로그래밍의 심화 내용

## 3. 멀티 스레드/멀티 프로세싱

〈자동차가 경주하는 코드의 멀티 스레드 예시〉

〈자동차가 경주하는 코드의 멀티 프로세싱 예시〉

The image shows two side-by-side code editors, both titled "Untitled-1.py".

**Left Editor (Threads):**

```
C: > Users > rryan > Downloads > Untitled-1.py > ...
1 import threading
2 import time
3
4 ## 클래스 정의 부분
5 class RacingCar :
6     carName = ''
7     def __init__(self, name) :
8         self.carName = name
9
10    def runCar(self) :
11        for _ in range(0, 3) :
12            carStr = self.carName + '~~ 달립니다.\n'
13            print(carStr, end = '')
14            time.sleep(0.1) # 0.1초 멈춤
15
16 ## 메인 코드 부분
17 car1 = RacingCar('@자동차1')
18 car2 = RacingCar('#자동차2')
19 car3 = RacingCar('$자동차3')
20
21 th1 = threading.Thread(target = car1.runCar)
22 th2 = threading.Thread(target = car2.runCar)
23 th3 = threading.Thread(target = car3.runCar)
24
25 th1.start()
26 th2.start()
27 th3.start()
28
```

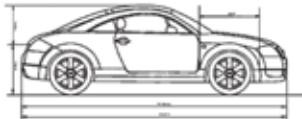
**Right Editor (Processes):**

```
C: > Users > rryan > Downloads > Untitled-1.py > ...
1 import multiprocessing;
2 import time
3
4 ## 클래스 정의 부분
5 class RacingCar :
6     carName = ''
7     def __init__(self, name) :
8         self.carName = name
9
10    def runCar(self) :
11        for _ in range(0, 3) :
12            carStr = self.carName + '~~ 달립니다.\n'
13            print(carStr, end = '')
14            time.sleep(0.1) # 0.1초 멈춤
15
16 ## 메인 코드 부분
17 if __name__ == "__main__":
18     car1 = RacingCar('@자동차1')
19     car2 = RacingCar('#자동차2')
20     car3 = RacingCar('$자동차3')
21
22     mp1 = multiprocessing.Process(target=car1.runCar)
23     mp2 = multiprocessing.Process(target=car2.runCar)
24     mp3 = multiprocessing.Process(target=car3.runCar)
25
26     mp1.start()          30      mp1.join()
27     mp2.start()          31      mp2.join()
28     mp3.start()          32      mp3.join()
```

# Chapter 12. 요약

- ✓ 자동차 클래스는 자동차의 설계도와 같으며, 설계도를 토대로 제작한 객체를 인스턴스 라고 함

자동차 설계도(클래스)



여러 번  
찍어 내기

자동차(인스턴스)



자동차 설계도(클래스)

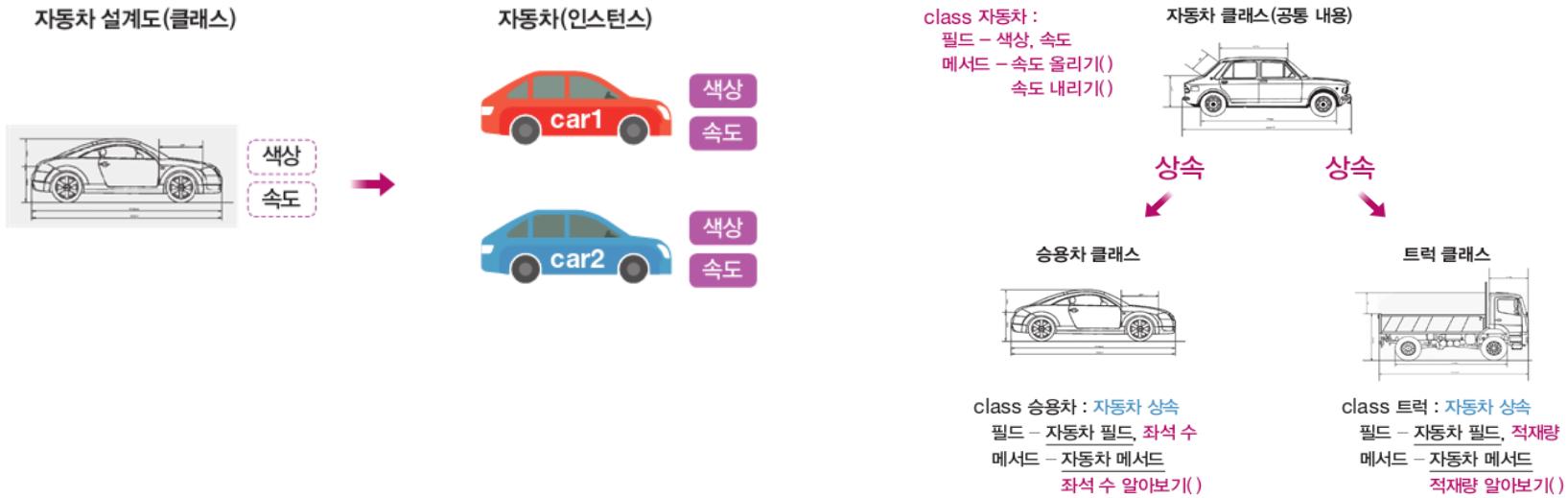
```
class 자동차 :  
    # 자동차의 속성  
    색상  
    속도  
    # 자동차의 기능  
    속도 올리기()  
    속도 내리기()
```

자동차(인스턴스)

```
자동차1 = 자동차()  
자동차2 = 자동차()  
자동차3 = 자동차()
```

# Chapter 12. 요약

- ✓ 생성자는 인스턴스를 생성하면 무조건 호출되는 메서드로 `_init_0` 이름을 갖음
- ✓ `_init_0`은 `self` 외에 별도의 매개변수를 사용하지 않으며, 이렇게 매개변수가 `self`만 있는 생성자를 기본 생성자라고 함(생성자도 다른 메서드처럼 매개변수 사용이 가능)
- ✓ 인스턴스 변수와 클래스 변수의 개념      ✓ 클래스 상속의 개념은 다음과 같음



## Chapter 12. 요약

- ✓ 클래스의 특별한 메서드로 `_init_()`, `_del_()`, `_repr_()`, `_add_()` 등이 있음
- ✓ 슈퍼 클래스에 본체가 없이 서브 클래스에서 반드시 오버라이딩을 통해 사용하도록 만든 메서드를 추상 메서드라고 함
- ✓ 이때는 구현하지 않더라도 문제가 없지만 향후 논리적 오류를 방지하기 위해 메서드 안에 `raise NotImplementedError()`를 추가해야 오류를 발생시킬 수 있음
- ✓ 스레드는 프로그램 하나에서 여러 처리를 동시에 할 수 있도록 제공하는 기능으로 동시에 여러 스레드가 작동하는 것을 멀티 스레드라고 함
- ✓ 스레드는 내부적으로 CPU 1개를 사용하지만, 멀티 프로세싱 기법은 여러 CPU를 사용

# 감사합니다.

우창우

[Dr.woo@chungbuk.ac.kr](mailto:Dr.woo@chungbuk.ac.kr)

# 팀프로젝트를 위한 조원 명단

조 번호	프로젝트 주제	개발내용	조장	조원
1조	(게임) 슈팅게임 (like 슈퍼마리오)	다음 페이지 양식에 맞추어 이메일로 작성/제출해 주세요.	조형준	고태경, 김다민
2조	(게임) 카드뒤집기 게임		김민혁	전영우, 김정민
3조	(게임) 텍스트 방탈출 게임		홍성진	김태영, 정새연
4조	(앱) 식당맛집		이규민	우태현, 전수혁
5조	(모바일_앱) 캘린더 앱		배정민	박상인, 서범교, 송설희
6조	(웹) 교내 학생 간 개인 중고 거래 사이트		박조현	김건우, 오다영
7조	(모바일_앱) 가계부 앱		박주현	권정욱, 정현준
8조	(게임) 보드게임 (like 클루)		김규현	김준후, 조윤정
9조	(모바일_앱) 학교 주변 식당 맛집 앱		신종환	신승우, 한강민
11조	(모바일_앱) 학교 졸업사정 Q&A 챗봇		한준영	고태영, 이관학, 육광민
12조	(앱) 수업 참여도 분석		윤시훈	전준석, 김민경
13조	(모바일_앱) 화장품 추천 앱		김준호	황지연, 이용희
14조	(게임) 슈팅게임 (like baba is you)		배수환	이한결, 신혜원
15조	(모바일_앱) 택시 함께 탈 사람 매칭 앱		박성범	이태정, 김민석

# 실습&과제 : (제출처) Dr.woo@chungbuk.ac.kr, (기한) 4.20(토) 까지

## 첨임1 팀프로젝트 과제 제출양식

과제명	00000 수준 달성을 위한 0000 00000 기술개발 00000 문제를 해결하기 위한 0000 00000 기술개발 00000 를 위한 0000 00000 기술 국제공동연구 00000 를 위한 0000 표준 개발 00000 를 위한 0000 00000 기반구축 등				
	기술분야 (ICT연구개발 기술분류체계)	대분류	중분류	소분류	세분류
	* ICT R&D 기술분류체계 참조				
개발목표 및 배경	<ul style="list-style-type: none"> <li>■(목표) *기술개발의 목표 제시           <ul style="list-style-type: none"> <li>○</li> <li>-</li> <li>○</li> <li>-</li> </ul> </li> </ul>				
	<ul style="list-style-type: none"> <li>■(추진배경) *기술개발 배경, 관련 연구개발 동향, 시장동향 및 규모 등 작성           <ul style="list-style-type: none"> <li>○</li> <li>-</li> <li>○</li> <li>-</li> </ul> </li> </ul>				
	<ul style="list-style-type: none"> <li>■(필요성) *기존 기술의 문제점, 개선 필요사항, 필요성 등           <ul style="list-style-type: none"> <li>○</li> <li>-</li> <li>○</li> <li>-</li> </ul> </li> </ul>				
개발내용/ 파급효과	<ul style="list-style-type: none"> <li>■(개발내용) *요약형태로 기술하되, 정량적/정성적 내용이 드러나도록 작성           <ul style="list-style-type: none"> <li>○</li> <li>-</li> <li>○</li> <li>-</li> </ul> </li> </ul>				
	<ul style="list-style-type: none"> <li>■(기대효과) *기술개발이 완료되었을 시 예상되는 기대 및 파급효과           <ul style="list-style-type: none"> <li>○</li> <li>-</li> <li>○</li> <li>-</li> </ul> </li> </ul>				

## 첨임2 ICT R&D 기술분류체계

대분류	중분류	소분류	세분류
SW-AI	인공지능	학습지능	머신러닝 추론/지식표현
		단일지능	언어지능 시각지능 청각지능
		복합지능	행동/소셜지능 상황/감정이해 지능형 메이전트 법률 인공지능(AGI)
		빅데이터	빅데이터 처리·유통 빅데이터 저장/처리/관리 기술 빅데이터 분석/예측 기술 빅데이터 활용·시각화
		응용SW	가상 시뮬레이션 사이버 물리 시스템(CPS) 디지털 신호처리 기술 UI/UX 기술 인터넷 서비스 SW 기술
		응용특화 SW	공공용SW 기업용SW 법용SW
	시스템SW	운영체제	초경량·저전력 운영체제 실시간 운영체제 모바일 운영체제 PC/서버 운영체제
		미들웨어	분산 시스템 SW 서비스 플랫폼 프로그래밍 언어 처리 SW 데이터 관리 기술 SW공학도구
		클라우드 컴퓨팅	SaaS(Software as a Service) 기술 *aaS 플랫폼 기술 가상 실행환경 기술 차세대 클라우드 컴퓨팅
		컴퓨팅 시스템	HPC/병렬 컴퓨팅 기술 뉴로컴퓨팅 차세대 컴퓨팅 기반 컴퓨팅
			서버 시스템 기술 스토리지 시스템 기술 IDC 인프라 기술