

자료구조



파이썬의 사용자 클래스



한국기술교육대학교
온라인평생교육원

학습내용

- 파이썬의 객체지향 프로그래밍
- 연산자 중복
- 클래스의 상속

학습목표

- 클래스와 객체의 개념을 설명하고 파이썬에서 클래스 기본 문법을 사용할 수 있다.
- 연산자 중복에 대해 설명하고 활용할 수 있다.
- 파이썬에서 사용자 클래스를 만들 수 있다.

파이썬의 객체지향 프로그래밍



파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

● 추상 자료형의 구현 방법

▶ 프로그래밍 기법 측면의 분류

함수 기반 프로그래밍(Functional Programming)

객체 지향 프로그래밍(Object Oriented Programming)

객체 지향 프로그래밍

- 데이터와 연산들을 하나로 묶은 클래스를 구현
 - 데이터: 클래스의 멤버 변수
 - 연산: 클래스의 멤버 함수
- 캡슐화(encapsulation), 정보 은닉(information hiding), 상속 등의 다양한 장점 활용 가능

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

● 함수 기반 프로그래밍의 문제점

방법 1

데이터는 전역 변수 + 연산은 일반 함수

```
items = [] # 객체(데이터) : 전역변수로 선언
def insert(pos, elem) : items.insert(pos, elem)
def delete(pos) : items.pop(pos)
...

```

여러 개의 리스트를
만들지 못함

방법 2

데이터는 지역 변수 + 연산은 일반 함수

```
# 객체(데이터) : 지역 변수로 리스트를 사용하는 위치에서 선언.
# 연산: 함수로 구현
def insert(items, pos, elem) : items.insert(pos, elem)
def delete(items, pos) : items.pop(pos)
...

```

다양한 자료구조에서
같은 연산이 있다면?
매우 혼란



파이썬의 객체지향 프로그래밍

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

● 함수 기반 프로그래밍의 문제점

일반 함수 구현의 문제점

- 같은 이름의 함수가 여러 자료구조에서 필요
- 예 리스트, 트리, 그래프, 힙 등에서 삽입(insert)과 삭제(delete)
- 해결 방안

• `insert_list()`, `insert_tree()`, `insert_graph()`

완전한 해결 방법 → 객체 지향 프로그래밍

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

● FP와 OOP의 사용 방법 비교

» 예 리스트 L, 트리 T, 그래프 G에 각각 새로운 노드 n을 삽입하는 코드 비교

함수 기반

```
insert_list(L, n)
insert_tree(T, n)
insert_graph(G, n)
```

객체 지향

```
L.insert(n)
T.insert(n)
G.insert(n)
```



파이썬의 객체지향 프로그래밍

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

파이썬의 객체 지향 프로그래밍

클래스(class)	객체(object)	인스턴스(instance, 사례)
<ul style="list-style-type: none"> 객체를 만들어 낼 수 있는 틀 예 붕어빵 틀 	<ul style="list-style-type: none"> 클래스를 이용해 만들어 낸 사례 각 객체마다 고유한 속성을 가짐 예 만들어진 붕어빵들 	<ul style="list-style-type: none"> 객체와 유사한 의미 클래스의 인스턴스가 객체

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

1 클래스와 객체

클래스와 객체의 예

배열 구조 리스트 클래스

ArrayList

```
a = ArrayList(10)
```

```
b = ArrayList(20)
```

- ArrayList는 배열 구조로 구현한 리스트 클래스
- a와 b는 객체
- a와 b는 ArrayList의 인스턴스

파이썬의 객체지향 프로그래밍



파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

○ 클래스의 개념

클래스: 객체를 정의하는 틀 또는 설계도

- 속성을 나타내는 멤버 변수
 - 자료구조 ADT의 데이터에 해당
- 동작을 나타내는 멤버 함수 또는 메소드(method)
 - 자료구조 ADT의 연산에 해당

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

○ 예 자동차 클래스(Car)

자동차의 속성(멤버 변수)

- 자동차의 색상(color)
- 자동차의 현재 속도(speed)

자동차의 동작(멤버 함수)

- 가속하기: speedUP()
- 감속하기: speedDown()

동작

- speedup()
- speedDown()



Car 클래스



파이썬의 객체지향 프로그래밍

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

● 자동차 클래스와 객체

속성	동작
색상: color 속도: speed	가속하기() 감속하기()



- 색상: black / 속도: 0
- 색상: red / 속도: 120
- 색상: yellow / 속도: 30
- 색상: blue / 속도: 0
- 색상: green / 속도: 0

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

● 클래스 정의와 생성자

▶ 클래스 정의

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

생성자

- 클래스의 특별한 멤버 함수: `__init__()`
- 객체가 생성될 때마다 자동으로 호출되는 함수
- 속성(데이터)을 정의하고 초기화



파이썬의 객체지향 프로그래밍

파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

- 생성자: `_init_(self, ...)`

```
class Car :
    def __init__(self, color, speed = 0) : # 생성자
        self.color = color # 자동차의 속성 color를 정의하고 초기화
        self.speed = speed # 자동차의 속성 speed를 정의하고 초기화
```



파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

- 객체 생성 코드

```
black = Car('black', 0)          # 검정색, 속도=0
red = Car('red', 120)           # 빨간색, 속도=120
yellow = Car('yellow', 30)       # 노란색, 속도=30
blue = Car('blue')              # 파란색, 속도=0 (디폴트 인수 사용)
green = Car('green')            # 초록색, 속도=0 (디폴트 인수 사용)
```

```
class Car :
    def __init__(self, color, speed = 0) : # 생성자
        self.color = color # 자동차의 속성 color를 정의하고 초기화
        self.speed = speed # 자동차의 속성 speed를 정의하고 초기화
```



파이썬의 객체지향 프로그래밍



파이썬의 사용자 클래스

파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

● 멤버 함수 구현

들여쓰기와 self에 유의

```
class Car :
    ...
    def speedUp(self) :          # 가속 연산
        self.speed += 10         # 속성(속도)을 변경

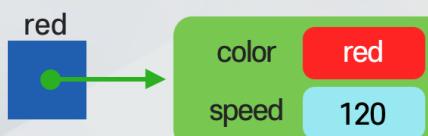
    def speedDown(self) :        # 감속 연산
        self.speed -= 10         # 속성(속도)을 변경
```

파이썬의 사용자 클래스

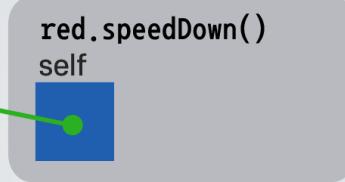
파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

● 멤버 함수의 활용



`red.speedDown()` # 감속시킴



```
class Car :
    ...
    def speedUp(self) :          # 가속 연산
        self.speed += 10         # 속성(속도)을 변경

    def speedDown(self) :        # 감속 연산
        self.speed -= 10         # 속성(속도)을 변경
```



파이썬의 객체지향 프로그래밍

파이썬의 사용자 클래스

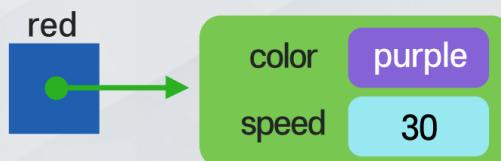
파이썬의 객체지향 프로그래밍

2 클래스 기본 문법

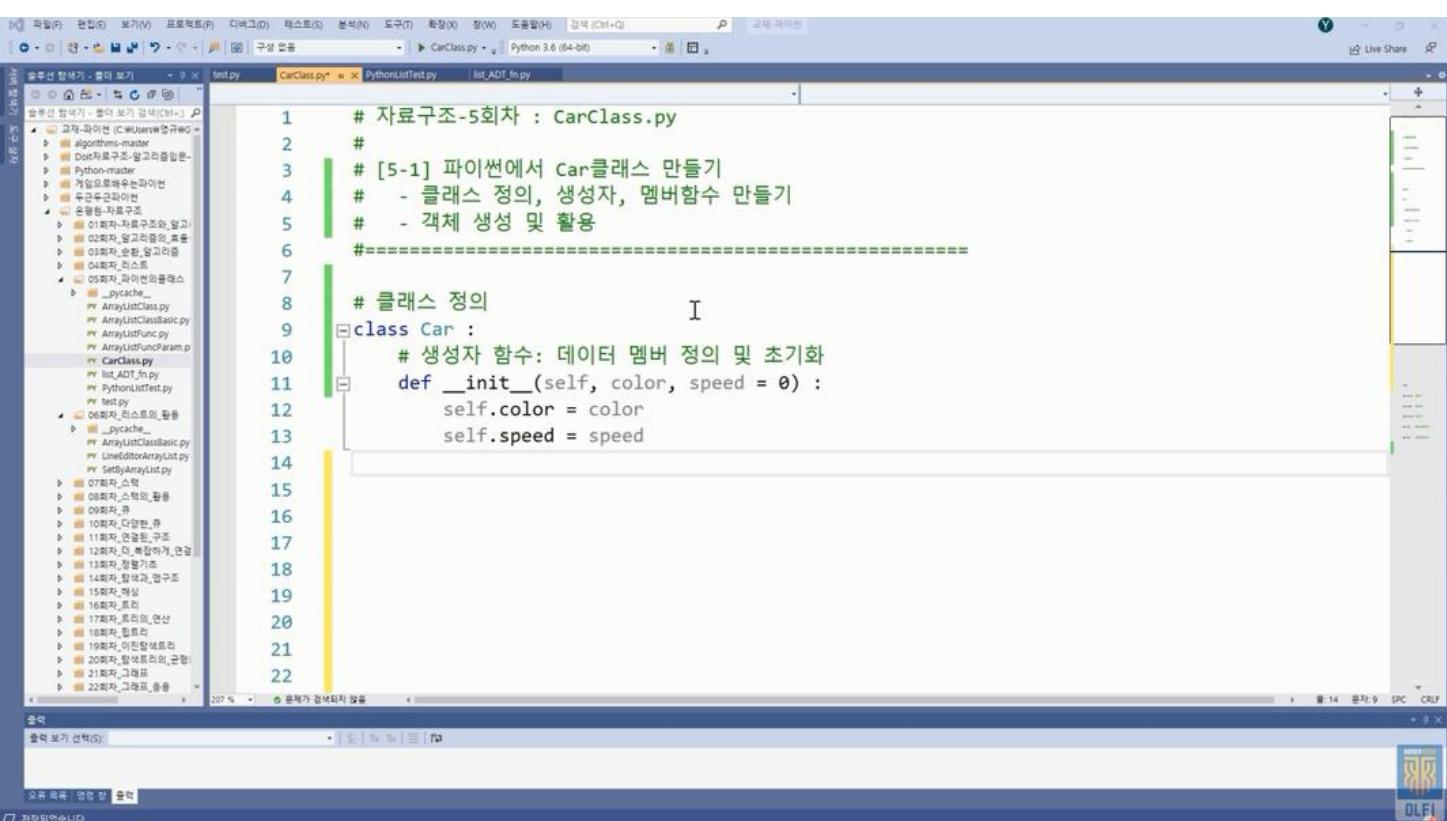
- 데이터 멤버의 직접 변경

데이터 멤버의 직접 변경은 가능하지만 바람직하지 않음

```
red.color = 'purple'      # 색상 변경  
red.speed = 30            # 속도 변경
```



파이썬의 객체지향 프로그래밍



The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- Title Bar:** 파일(F), 관리(S), 보기(V), 프로젝트(P), 디버그(D), 테스트(S), 브레이크(B), 도구(T), 확장(X), 창(W), 도움말(H), 검색(Ctrl+Q)
- Code Editor:** 파일에 있는 코드는 `CarClass.py`입니다.
- Code Content:**

```

1  # 자료구조-5회차 : CarClass.py
2  #
3  # [5-1] 파이썬에서 Car클래스 만들기
4  #   - 클래스 정의, 생성자, 멤버함수 만들기
5  #   - 객체 생성 및 활용
6  =====
7
8  # 클래스 정의
9  class Car :
10    # 생성자 함수: 데이터 멤버 정의 및 초기화
11    def __init__(self, color, speed = 0) :
12      self.color = color
13      self.speed = speed
14
15
16
17
18
19
20
21
22

```
- Left Sidebar:** 파일 목록과 편집기 설정을 보여주는 패널입니다.
- Bottom Status Bar:** 출처 보기 선택(S), 확장되었습니다.

실습 단계

클래스 정의: `class car :`

자동차의 색상 속성, 속도 속성에 대한 멤버 변수 정의와 초기화

클래스의 모든 멤버 함수는 첫번째 매개변수 `self` 필요

클래스의 생성자를 이용하여 객체 생성

`display` 함수: 자동차의 현재 상태를 출력하는 함수

`Car class`는 속도 조정 가능

연산자 중복



파이썬의 사용자 클래스

연산자 중복

1 연산자 중복이란?

연산자 중복(Operator Overloading)

사용자 정의 클래스의 객체들에게 ==, +, - 등의 표준 연산자들을 적용할 수 있도록 하는 기능

클래스의 특별한 멤버 함수

```
car3 = car1 + car2          # 자동차 객체의 덧셈?  
if car1 == car2 : ...       # 자동차 객체의 비교?
```

정해진 피연산자의 수를 변경할 수는 없음

```
car3 = car1 - car2          # 이항 연산자 -  
car4 = - car2               # 단항 연산자 -
```

파이썬의 사용자 클래스

연산자 중복

1 연산자 중복이란?

● 연산자 중복의 예 : == 연산자

▶ 예 if car1 == car2

무엇이 같으면 같은 차인가?

- 색상? 속도? 모델? 배기량?

연산자 중복을 이용해 사용자가 두 자동차 객체를 '==' 연산자를 이용해 비교 가능

'==' 연산자의 의미는 사용자가 마음대로 정하는 것이 가능

'==' 연산자 중복 함수 이름

연산자 중복



파이썬의 사용자 클래스

연산자 중복

1 연산자 중복이란?

● 연산자 중복 정의 표(단항/이항)

Operation	Class Method	Operation	Class Method
<code>srt(obj)</code>	<code>__str__(self)</code>	<code>obj + rhs</code>	<code>__add__(self, rhs)</code>
<code>Len(obj)</code>	<code>__len__(self)</code>	<code>obj + rhs</code>	<code>__sub__(self, rhs)</code>
<code>item in obj</code>	<code>__contains__(self, item)</code>	<code>obj * rhs</code>	<code>__mul__(self, rhs)</code>
<code>Y=obj[idx]</code>	<code>__getitem__(self, idx)</code>	<code>obj / rhs</code>	<code>__truediv__(self, rhs)</code>
<code>obj[idx]=val</code>	<code>__setitem__(self, idx, val)</code>	<code>obj // rhs, obj % rhs</code>	<code>__floordiv__(self, rhs)</code>

파이썬의 사용자 클래스

연산자 중복

1 연산자 중복이란?

● 연산자 중복 정의 표(단항/이항)

Operation	Class Method	Operation	Class Method
<code>obj = rhs</code>	<code>__eq__(self, rhs)</code>	<code>obj ** rhs</code>	<code>__pow__(self, rhs)</code>
<code>obj < rhs</code>	<code>__lt__(self, rhs)</code>	<code>obj += rhs</code>	<code>__iadd__(self, rhs)</code>
<code>obj <= rhs</code>	<code>__le__(self, rhs)</code>	<code>obj -= rhs</code>	<code>__isub__(self, rhs)</code>
<code>obj != rhs</code>	<code>__ne__(self, rhs)</code>	<code>obj *= rhs</code>	<code>__imul__(self, rhs)</code>
<code>obj > rhs</code>	<code>__gt__(self, rhs)</code>	<code>obj /= rhs</code>	<code>__itruediv__(self, rhs)</code>
<code>obj >= rhs</code>	<code>__ge__(self, rhs)</code>	<code>obj // rhs</code>	<code>__ifloordiv__(self, rhs)</code>
			<code>__imod__(self, rhs)</code>
			<code>__ipow__(self, rhs)</code>

연산자 중복



파이썬의 사용자 클래스

연산자 중복

2 파이썬의 연산자 중복

◎ 예 비교 연산자 == 중복

```
def __eq__(self, carB): return self.color == carB.color
```

```
print("car2==car6 : ", car2==car6)
print("car3==car6 : ", car3==car6)
```

C:\WINDOWS\system32\cmd.exe
car2==car6 : False
car3==car6 : True

파이썬의 사용자 클래스

연산자 중복

2 파이썬의 연산자 중복

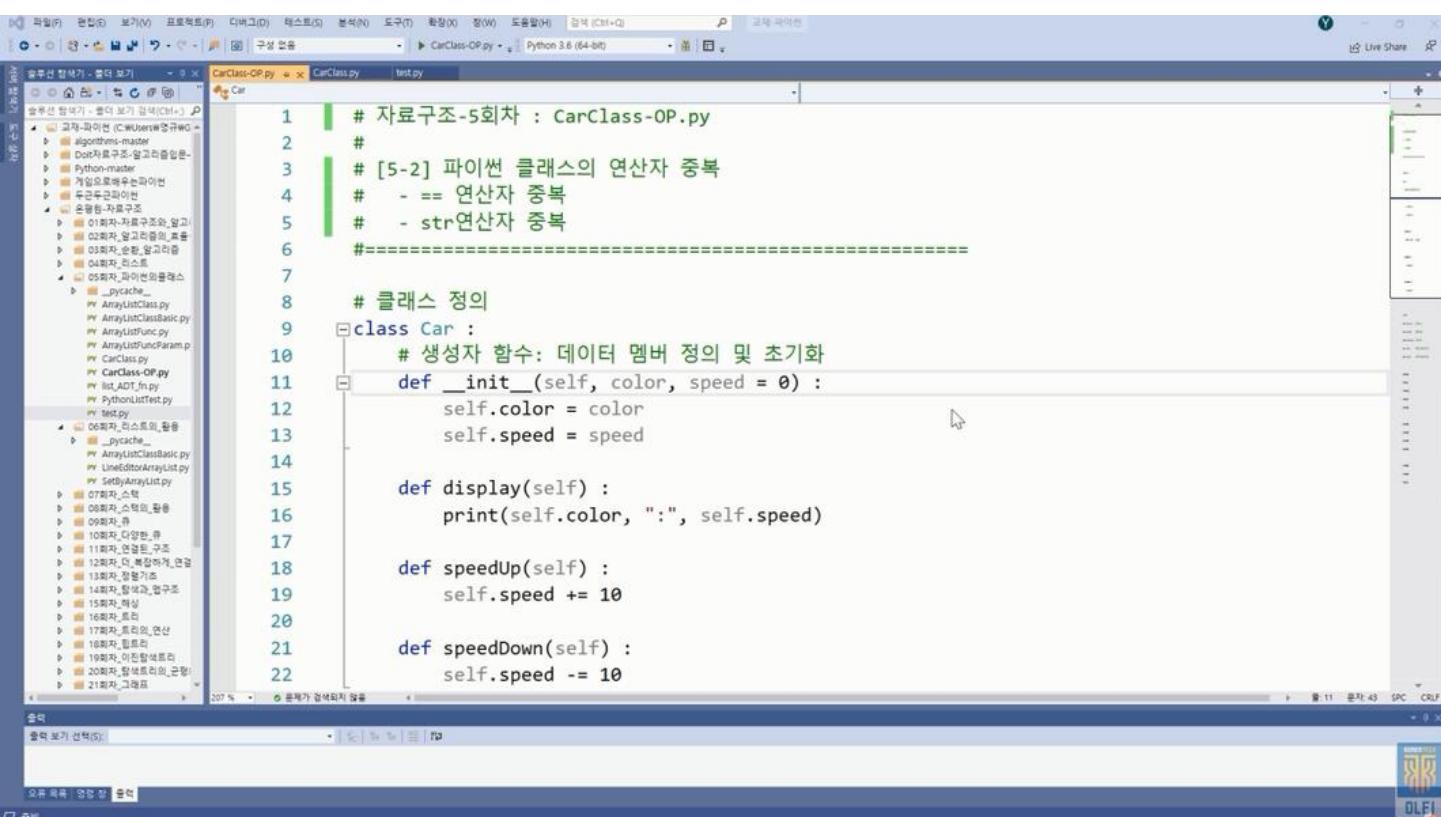
◎ 문자열로 변환 연산자: __str__

```
def __str__(self):
    return 'color = %s, speed = %d'%(self.color, self.speed)
```

```
print("[car3]", car3)
```

C:\WINDOWS\system32\cmd.exe
[car3] color = yellow, speed = 30

연산자 중복



```

파일(F) 관찰(O) 보기(V) 프로젝트(P) 디버그(D) 테스트(S) 브레이크(B) 도구(T) 확장(X) 설정(W) 도움말(H) 검색(Ctrl+Q) Python 3.6 (64-bit) | 자료구조
CarClass-OP.py CarClass.py test.py
Car
1 # 자료구조-5회차 : CarClass-OP.py
2 #
3 # [5-2] 파이썬 클래스의 연산자 중복
4 # - == 연산자 중복
5 # - str연산자 중복
6 =====
7
8 # 클래스 정의
9 class Car :
10     # 생성자 함수: 데이터 멤버 정의 및 초기화
11     def __init__(self, color, speed = 0) :
12         self.color = color
13         self.speed = speed
14
15     def display(self) :
16         print(self.color, ":", self.speed)
17
18     def speedUp(self) :
19         self.speed += 10
20
21     def speedDown(self) :
22         self.speed -= 10
  
```

The screenshot shows a Microsoft Visual Studio Code interface with a Python file named 'CarClass-OP.py' open. The code defines a 'Car' class with an __init__ method, a display method, and two additional methods, speedUp and speedDown. The code is annotated with comments explaining operator overriding: '# [5-2] 파이썬 클래스의 연산자 중복', '# - == 연산자 중복', and '# - str연산자 중복'. The code editor has syntax highlighting for Python, and the status bar at the bottom right shows the current file is 'CarClass-OP.py' with 43 lines of code.

실습 단계

black, red 객체 생성 → black, red display → red2 자동차 생성

black isEqual(red2) / red isEqual(red2)의 출력

색상이 같을 경우: True, 색상이 다를 경우: False

자동 문자열 변환 str() 함수(연산자 중복 함수): 자동차 객체를 문자열로 바꿀 시 사용



클래스의 상속

파이썬의 사용자 클래스

클래스의 상속

1 클래스 상속의 개념

클래스의 상속

- Is-a 관계(is a kind of)
- 예 SuperCar is a Car

```
Class SuperCar(Car) :
    def __init__(self, color, speed = 0, bTurbo = True) :
        super().__init__(color, speed)           # 부모(Car)클래스의 생성자 호출
        self.bTurbo = bTurbo                   # 터보모드를 위한 변수 생성 및 초기화
```

Car 부모 클래스, 기반 클래스

SuperCar 자식 클래스, 서브 클래스, 파생 클래스

파이썬의 사용자 클래스

클래스의 상속

1 클래스 상속의 개념

자식 객체의 생성

```
s1 = SuperCar( " GOLD ", 0, True)
s2 = SuperCar( " white ", 0, False)
```



클래스의 상속



파이썬의 사용자 클래스

클래스의 상속

1 클래스 상속의 개념

부모 멤버의 활용

```
s1.speedup( )
s2.speedup( )
print("슈퍼카1:", s1)
print("슈퍼카2:", s2)
```

C:\WINDOWS\system32\cmd.exe

```
슈퍼카1: color = Gold, speed = 10
슈퍼카2: color = White, speed = 10
```

파이썬의 사용자 클래스

클래스의 상속

1 클래스 상속의 개념

◎ 재정의(Overriding)

▶ 메소드의 재정의

```
def setTurbo(self, bTurbo = True) : # 터보 모드를 on/off 하는 메소드
    self.bTurbo = bTurbo
def speedUp(self) : # SuperCar의 SpeedUp. 메소드의 재정의
    if self.bTurbo :
        self.speed += 50 # 속도가 급속히(50) 증가됨
    else :
        super().speedup() # 일반 자동
def __str__(self) :
    if self.bTurbo :
        return "[%s] [speed = %d] 터보모드" %(self.color, self.speed)
    else :
        return "[%s] [speed = %d] 일반모드" %(self.color, self.speed)
```

```
s1.speedup( )
s2.speedup( )
print("슈퍼카1:", s1)
print("슈퍼카2:", s2)
```

C:\WINDOWS\system32\cmd.exe

```
슈퍼카1: [Gold] [speed = 50] 터보모드
슈퍼카2: [White] [speed = 10] 일반모드
```

클래스의 상속

```

24     def isEqual(self, carB) : [...]
25
26     def __eq__(self, carB) : [...]
27
28     # 자동 문자열 변환 str() 함수
29     def __str__(self) :
30         return "color = %s, speed = %d" % (self.color, self.speed)
31
32
33     # Car를 상속한 SuperCar 클래스 만들기
34
35     class SuperCar(Car) :
36         def __init__(self, color, speed = 0, bTurbo = True) :
37             super().__init__(color, speed)
38             self.bTurbo = bTurbo
39
40
41
42
43
44
45
46
47

```

실습 단계

파이썬의 클래스 상속

상속한 SuperCar 생성: `class SuperCar(car)` :

생성자 새롭게 작성

Display, speedUp, speedDown, isEqual, __eq__ 연산은 상속 시 동일. 즉, 바로 사용 가능

SuperCar의 정의: 자동차의 속성 (color, speed)에 추가적 멤버함수 '터보' 추가

`bTuebo = True` → 터보 상태 켜짐

터보 상태가 꺼짐 = 일반적 자동차 동작

상속받은 클래스 = 자식 클래스

부모에 대한 영역 초기화 필요

부모 생성자 이용하여 부모 부분 초기화

부모 클래스의 객체: 함수 `super`