

# 객체와 클래스

## 학습 내용

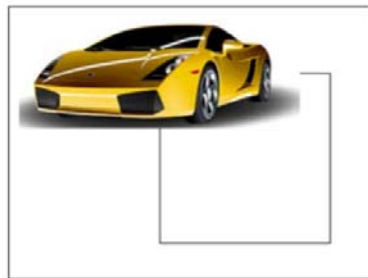
- 객체 지향 프로그래밍을 간단히 이해합니다.
- 객체의 개념을 이해합니다.
- 객체와 클래스의 관계를 이해합니다.
- 객체를 활용하여 프로그램을 작성해 봅니다.

## 참고자료

- 두근두근파이썬 - 13장
- 파이썬 7장

## 이번 장에서 만들 프로그램

- 자동차를 나타내는 클래스를 정의하고 사용해본다.



## 객체 지향 프로그래밍

### ●객체(object)

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



## 객체들 사이의 상호작용

- 텔레비전과 리모콘은 모두 특정한 기능을 수행하는 객체
- 텔레비전과 리모콘은 메시지를 통하여 서로 상호 작용하고 있다.



## 객체란?

- 객체는 하나의 물건이라고 생각하면 된다.
- 객체는 속성(attribute)과 동작(action)을 가지고 있다.
  - 객체의 속성 : 객체 안의 변수에 저장 (인스턴스 변수, 데이터 필드)
  - 객체의 동작 : 메소드라고 부른다.
  - ◆객체 안에 정의된 함수

object.make  
object.model  
object.color  
object.year  
object.price  
object.speed  
object.gasLevel

속성
메이커
모델
색상
연식
가격



동작
주행하기
방향바꾸기
주차하기

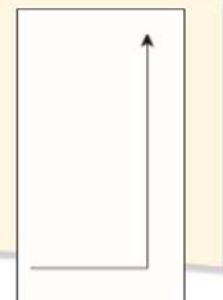
object.drive()  
object.park()  
object.accel()  
object.brake()

## 거북이도 객체

- 터틀 그래픽에서 거북이가 바로 객체

```
from turtle import * # turtle 모듈에서 모든 것을 불러올 것
alex = Turtle()      # 거북이 객체를 생성한다.

alex.forward(100)     # forward()는 거북이 객체의 메소드이다.
alex.left(90)         # left()는 거북이 객체의 메소드이다.
alex.forward(200)     # forward()는 거북이 객체의 메소드이다.
```



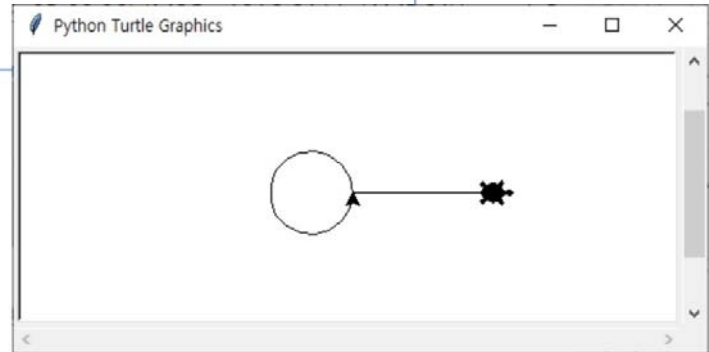
## 터클 객체 생성하기 (실습 - P01)

- 거북이 객체를 2개 생성하고, 한 거북이(t1)는 현재 위치에서 오른쪽으로 100 만큼 선을 그리고, 다른 거북이(t2)는 위쪽으로 반지름이 30만큼 원을 그린다.

```
import turtle
t1 = turtle.Turtle()
t2 = turtle.Turtle()

t1.shape("turtle")
t1.forward(100)

t2.left(90)
t2.circle(30)
```



## 터틀이 가지고 있는 메소드(참고)

### ●URL

- <https://docs.python.org/3.7/library/turtle.html>

#### Turtle methods

##### Turtle motion

###### Move and draw

```
forward() | fd()
backward() | bk() | back()
right() | rt()
left() | lt()
goto() | setpos() | setposition()
setx()
sety()
setheading() | seth()
home()
circle()
dot()
stamp()
clearstamp()
clearstamps()
undo()
speed()
```

##### Tell Turtle's state

```
position() | pos()
towards()
xcor()
ycor()
heading()
distance()
```

##### Setting and measurement

```
degrees()
radians()
```

## 터틀이 가지고 있는 메소드(참고)

### Pen control

#### Drawing state

pendown() | pd() | down()  
penup() | pu() | up()  
pensize() | width()  
pen()  
isdown()

#### Color control

color()  
pencolor()  
fillcolor()

#### Filling

filling()  
begin\_fill()  
end\_fill()

#### More drawing control

reset()  
clear()  
write()

### Turtle state

#### Visibility

showturtle() | st()  
hideturtle() | ht()  
isvisible()

#### Appearance

shape()  
resizemode()  
shapeseize() | turtlesize()  
shearfactor()  
settiltangle()  
tiltangle()  
tilt()  
shapetransform()  
get\_shapepoly()

### Using events

onclick()  
onrelease()  
ondrag()

#### Special Turtle methods

begin\_poly()  
end\_poly()  
get\_poly()  
clone()  
getturtle() | getpen()  
getscreen()  
setundobuffer()  
undobufferentries()

## 터틀이 가지고 있는 메소드(참고)

### ●거북이 숨기기/보이기

메소드명	내용	비고
showturtle( )	거북이 보이기	st( )
hideturtle( )	거북이 숨기기	ht( )
isvisible( )	거북이가 보이는지 확인	

### ●거북이 모양

메소드명	내용	비고
shape( )	거북이 (펜) 모양	
resizemode( )	거북이(펜) 크기 변경 모드 설정	
shapeseize( )	거북이(펜) 크기 변경	turtlesize( )
settiltangle( )	거북이(펜) 방향 변경	tiltangle( ), tilt( )



## 터틀이 가지고 있는 메소드(참고)

### ●거북이 모양

#### ●shape( )

- ◆arrow, turtle, circle, square, triangle, classic

#### ●resizemode( )

- ◆shapsize( )와 함께 사용
- ◆거북이의 크기를 변경할 때 어떻게 변경할지 선택한다.
- ◆옵션 : auto, user, noresize(기본)

#### ●shapsize( ), turtlesize( )

- ◆거북이의 크기를 변경
- ◆shapsize(너비, 길이, 외곽선)
- ◆거북이의 크기를 바꾸려면, resizingmode( "user" ) 로 한후 바꾼다.

```
turtle.resizingmode( "user" )  
turtle.shapsize(3, 1, 1)
```



## 터틀이 가지고 있는 메소드(참고)

### ●거북이 모양

#### ●settiltangle( ), tiltangle( ), tile( )

- ◆거북이의 각도를 변경한다.

```
turtle.tiltangle( 56 )
```

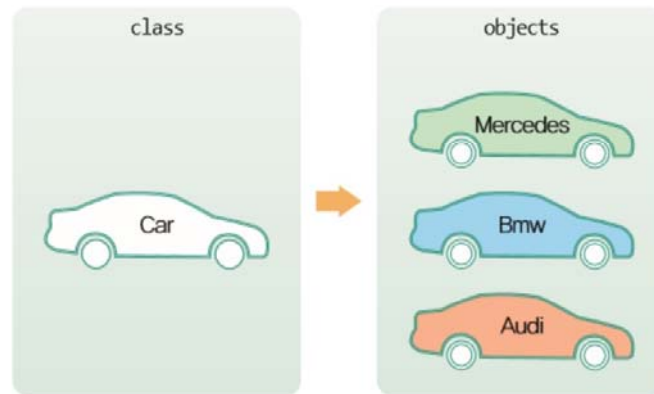


- ◆현재의 각도를 알기 위하여 각도를 넣지 않는다.

```
turtle.tiltangle( )
```

## 클래스를 작성 → 객체 생성하기

- 객체의 설계도(클래스)를 작성한다.
  - 객체가 가지고 있는 속성 : 변수로 표현 - 인스턴스 변수, 데이터 필드
  - 객체의 동작 : 메소드로 정의
- 클래스로부터 객체를 생성한다.



## 클래스를 작성 → 객체 생성하기

### ●자동차 클래스 작성하기

```
class Car:
    def drive(self):
        self.speed = 10
```

- 속성, 인스턴스 변수, 데이터 필드
  - ◆speed
  - ◆클래스의 속성은 self. 을 붙여야 한다.
  - ◆속성이 아닌 일반 변수는 self. 를 붙이지 않는다.
- 메소드
  - ◆drive( )
    - ※모든 메소드의 첫번째 매개변수는 항상 self 로 현재 객체를 가리킨다.

## 클래스를 작성 → 객체 생성하기

### ●객체 멤버에 접근하기

#### ●객체의 멤버

##### ◆데이터 필드

❖객체(인스턴스)는 데이터 필드에 특정 값을 가지고 있기 때문에, 필드를 인스턴스 변수라고도 한다

##### ◆메소드

❖객체의 데이터 필드 값을 변경하고, 객체에 대한 행동을 수행한다.

❖객체(인스턴스)에 의해 호출되기 때문에 인스턴스 메소드(Instance Method)라고도 한다.

## 클래스를 작성 → 객체 생성하기

### ●객체 멤버에 접근하기

#### ●객체 생성하기

◆객체를 생성하여 변수에 할당한다.

객체참조변수 = 클래스이름(인자)

myCar = Car( )



## 클래스를 작성 → 객체 생성하기

### ●객체 멤버에 접근하기

#### ●객체 멤버에 접근하기

##### ◆데이터 필드 접근 / 추가하기

객체참조변수.데이터필드

```
myCar.color = "blue"  
myCar.model = "E-Class"  
print(myCar.speed)
```

} 객체에 속성 추가

##### ◆메소드 호출

객체참조변수.메소드(인자)

```
myCar.drive( )
```

← 객체안의 drive( ) 메소드 호출하기

## 클래스를 작성 → 객체 생성하기 (실습 - P02)

### ●자동차 클래스 작성과 객체 생성

```
class Car:  
    def drive(self):  
        self.speed = 60  
        ...  
  
myCar = Car()  
myCar.speed = 0  
myCar.model = "E-Class"  
myCar.color = "blue"  
myCar.year = "2017"  
  
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

## 객체를 생성하면서 초기화하기 (실습 - P03)

### ●초기자 (Initializer)

- 객체를 생성시 속성을 초기화한다.
- 클래스 안에 `__init__()`를 정의한다.
- ◆외부에서 전달되는 초기값을 받아 들어 속성의 값으로 설정한다.

```
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

## 인스턴스 변수의 사용 영역

### ●인스턴스 변수의 사용 영역은 전체 클래스이다.

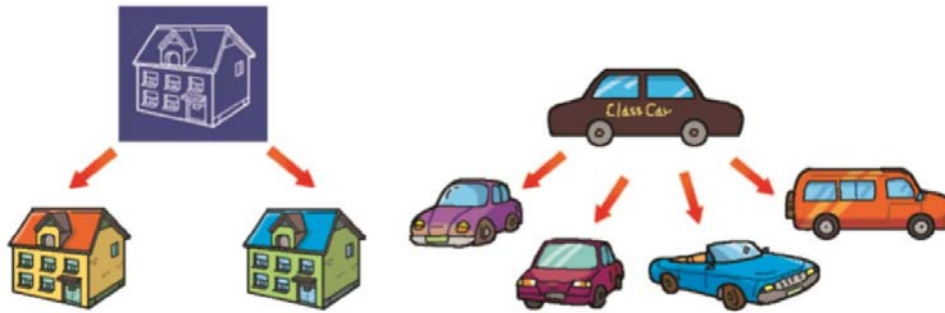
```
class ClassName:
    def __init__(self, ...):
        self.x = 1 # X 생성 및 수정
        ...

    def m1(self, ...):
        self.y = 2 # Y 생성 및 수정
        ...
        z = 5 # Z 생성 및 수정
        ...
        # z의 스코프

    def m2(self, ...):
        self.y = 3 # Y 생성 및 수정
        ...
        u = self.x + 1 # U 생성 및 수정
        self.m1(...) # m1 호출
        # self.x와 self.y의 스코프
```

## 하나의 클래스로 많은 객체 만들기

- 같은 사양을 가진 차를 여러 개 만드는 것과 같이, 클래스로부터 객체를 무수히 만들 수 있다.



## 하나의 클래스로 많은 객체 만들기 (실습 - P04)

- Car 클래스로부터 3개의 자동차를 만든다.

```
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")
```

- 객체마다 다른 속성을 가지게 된다.

## \_\_str\_\_() 메소드 (실습 - P04)

### ●객체를 출력

```
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")

print(myCar)
```

```
<__main__.Car object at 0x000001F56A440BE0>
```

## \_\_str\_\_() 메소드 (실습 - P05)

### ●객체 출력시 객체 안의 정보 출력하기

- \_\_str\_\_() 메소드는 객체를 출력하면 자동으로 호출된다.

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

    def __str__(self):
        msg = "속도:" + str(self.speed) + " 색상:" + self.color + " 모델:" + self.model
        return msg

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

```
속도:0 색상:blue 모델:E-class
```

- P05의 momCar 와 dadCar 객체를 생성하여 \_\_str\_\_() 메소드를 사용하여 객체의 내용을 출력하는 코드를 추가한다.



## self는 무엇인가?

### ●self 매개 변수

- 어떤 객체가 메소드를 호출했는지 알려준다.

```
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")
yourCar = Car(0, "white", "S-class")

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

## 객체와 관련된 실습

### ●클래스는 별도의 파일로 작성한다.

- 파일에는 하나의 클래스만 정의한다.

### ●클래스와 파일 이름을 동일하게 만든다.

- 클래스 명 : Circle
- 파일명 : Circle.py

### ●클래스를 만든 후, 클래스에서 정의된 메소드를 테스트하는 코드를 클래스 파일에 반드시 넣는다.

### ●실습명

- 클래스명과 동일하게 만든다.
- 클래스의 기능 연습을 위한 코드는 표시된 실습 번호를 따른다.



## 클래스선언과 테스트하기 (실습 - Circle)

### ●Circle 클래스 (Circle.py)

#### ●데이터 필드 : radius

#### ●생성자 ( \_\_init\_\_ )

- ◆매개 변수로 받은 값을 데이터 필드에 저장한다.
- ◆매개 변수가 없는 경우에는 radius를 1로 설정한다.

#### ●메소드

#### ◆getPerimeter( self ) : 원의 둘레를 계산하여 반환

- ❖매개 변수가 없다.
- ❖데이터 필드 radius 를 사용하여 계산한다.

#### ◆getArea( self ) : 원의 넓이를 계산하여 반환

- ❖매개 변수가 없다.
- ❖데이터 필드 radius 를 사용하여 계산한다.

#### ◆setRadius( self, radius )

- ❖매개 변수 radius 로 전달 받은 값을 데이터 필드 radius 에 저장한다.

## 클래스선언과 테스트하기 (실습 - Circle)

### ●Circle 클래스 (Circle.py)

#### ●데이터 필드 : radius

#### ●생성자 ( \_\_init\_\_ )

- ◆매개 변수로 받은 값을 데이터 필드에 저장한다.
- ◆매개 변수가 없는 경우에는 radius를 1로 설정한다.

```
class Circle :  
    def __init__(self, radius=1):  
        self.radius = radius
```

## 클래스선언과 테스트하기 (실습 - Circle)

### ●Circle 클래스 (Circle.py)

#### ●메소드

◆getPerimeter( self ) : 원의 둘레를 계산하여 반환

❖매개 변수가 없다.

❖데이터 필드 radius 를 사용하여 계산한다.

```
import math

class Circle :
    def __init__(self, radius=1):
        self.radius = radius

    def getPerimeter( self ) :
        return 2 * self.radius * math.pi
```

## 클래스선언과 테스트하기 (실습 - Circle)

### ●Circle 클래스 (Circle.py)

#### ●메소드

◆getArea( self ) : 원의 넓이를 계산하여 반환

❖매개 변수가 없다.

❖데이터 필드 radius 를 사용하여 계산한다.

```
import math

class Circle :
    def __init__(self, radius=1):
        self.radius = radius

    def getPerimeter( self ) :
        return 2 * self.radius * math.pi

    def getArea( self ) :
        return self.radius * self.radius * math.pi
```

## 클래스선언과 테스트하기 (실습 - Circle)

### ● Circle 클래스 (Circle.py)

#### ● 메소드

#### ◆ setRadius( self, radius )

❖ 매개 변수 radius 로 전달 받은 값을 데이터 필드 radius 에 저장한다.

```
import math

class Circle :
    def __init__(self, radius=1):
        self.radius = radius

    def getPerimeter( self ) :
        return 2 * self.radius * math.pi

    def getArea( self ) :
        return self.radius * self.radius * math.pi

    def setRadius( self, radius ) :
        self.radius = radius
```

## 클래스선언과 테스트하기 (실습 - Circle)

### ● Circle 클래스 (Circle.py)

```
import math

class Circle:
    # Circle 객체를 생성한다.
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

#### ● 원주율 (=3.141592653....)

◆ math 모듈의 pi 에 저장되어 있다.

## 클래스선언과 테스트하기 (실습 - Circle)

### ●작성한 모듈 테스트하기

- 모듈이 직접 실행하는지 확인한다.

```
if __name__ == '__main__':
```

- ◆직접 실행될 경우 참이 반환된다.

- 실행한 문을 if 문의 실행 블록에 넣는다.

- ◆예 : Circle.py 를 테스트하기 위한 코드

```
if __name__ == '__main__':  
    c = Circle(5)  
    print("Radius=", c.radius)  
    print("Perimeter=", c.getPerimeter())  
    print("Area=", c.getArea())
```

```
import math
```

```
class Circle:
```

```
    # Circle 객체를 생성한다.
```

```
    def __init__(self, radius = 1):  
        self.radius = radius
```

```
    def getPerimeter(self):  
        return 2 * self.radius * math.pi
```

```
    def getArea(self):  
        return self.radius * self.radius * math.pi
```

```
    def setRadius(self, radius):  
        self.radius = radius
```

```
if __name__ == '__main__':  
    c = Circle(5)  
    print( "Radius=", c.radius )  
    print( "Perimeter=", c.getPerimeter() )  
    print( "Area=", c.getArea() )
```

```
Radius= 5  
Perimeter= 31.41592653589793  
Area= 78.53981633974483
```

## 선언된 클래스 사용하기 (실습 - TestCircle)

- 선언된 클래스(Circle.py) 를  
import 하여 사용하기

- 파일명 : TestCircle.py

```
from Circle import Circle
```

```
def main():
```

```
    # Create a circle with radius 1
```

```
    circle1 = Circle()
```

```
    print("반지름이 ", circle1.radius,  
          "인 원의 넓이는 ", circle1.getArea(), "입니다.")
```

```
    # Create a circle with radius 25
```

```
    circle2 = Circle(25)
```

```
    print("반지름이 ", circle2.radius,  
          "인 원의 넓이는 ", circle2.getArea(), "입니다.")
```

```
    # Create a circle with radius 125
```

```
    circle3 = Circle(125)
```

```
    print("반지름이 ", circle3.radius,  
          "인 원의 넓이는 ", circle3.getArea(), "입니다.")
```

```
    # Modify circle radius
```

```
    circle2.radius = 100
```

```
    print("반지름이 ", circle2.radius,  
          "인 원의 넓이는 ", circle2.getArea(), "입니다.")
```

```
main() # Call the main function
```



## 선언된 클래스 사용하기 (실습 - TestCircle)

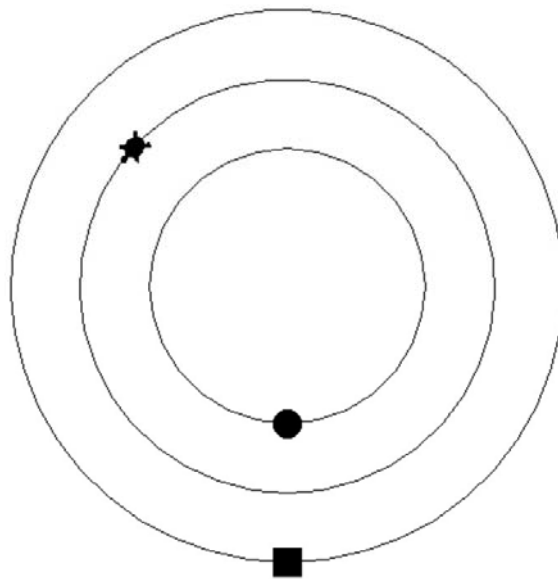
●선언된 클래스(Circle.py) 를 import 하여 사용하기

●TestCircle.py

반지름이 1 인 원의 넓이는 3.141592653589793 입니다.  
반지름이 25 인 원의 넓이는 1963.4954084936207 입니다.  
반지름이 125 인 원의 넓이는 49087.385212340516 입니다.  
반지름이 100 인 원의 넓이는 31415.926535897932 입니다.

## 실습 - P06 : 터틀 그래픽을 다시 보자

●터틀 객체를 여러 개 생성해서 다른 원을 그려보자





## 실습 - P06 : 터틀 그래픽을 다시 보자

```
from turtle import * # turtle 모듈에서 모든 것을 불러온다.

t1 = Turtle()        # 거북이 객체를 생성한다.
t1.shape("circle")

t2 = Turtle()        # 거북이 객체를 생성한다.
t2.shape("turtle")

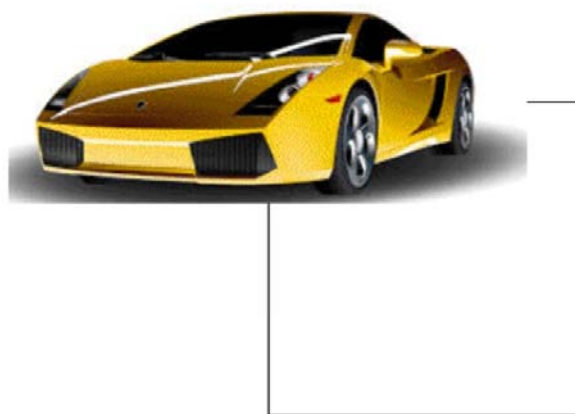
t3 = Turtle()        # 거북이 객체를 생성한다.
t3.shape("square")

t1.penup()           # 펜을 든다.
t2.penup()
t1.goto(0, 100)      # 거북이를 이동한다.
t2.goto(0, 50)
t1.pendown()         # 펜을 내린다.
t2.pendown()

while True:
    t1.circle(100)    # 원을 그린다.
    t2.circle(150)
    t3.circle(200)
```

## 실습 - P07 : Car 클래스 + Turtle 클래스

- 터틀 그래픽을 사용하여서 화면에 자동차를 그리고 움직여 보자.



- 자동차를 그리는 부분을 Car 클래스에 추가하기
  - ◆Car 클래스의 생성자에서 터틀을 생성하고 변수에 저장한다.
  - ◆터틀의 모양을 자동차로 변경한다.
- 터틀 그래픽에 이미지를 등록하기 위하여 register\_shape( ) 메소드를 사용한다.

## 실습 - P07 : Car 클래스 + Turtle 클래스

```
from turtle import *
class Car:
    def __init__(self, speed, color, model):
        self.speed = speed
        self.color = color
        self.model = model
        self.turtle = Turtle()
        self.turtle.shape("car.gif")

    def drive(self):
        self.turtle.forward(self.speed)

    def left_turn(self):
        self.turtle.left(90)

register_shape("car.gif")
myCar = Car(200, "red", "E-class")
for i in range(20):
    myCar.drive()
    myCar.left_turn()
```

## 실습 - P08 : Car 클래스 + Turtle 클래스

### ●P07 실습의 코드를 다음과 같이 변형하시오.

- 코드가 직접 실행되는 경우에만 Car 클래스 이외의 부분이 실행되도록 한다.
- 테스트 코드를 다음과 같이 작성한다.
  - ◆Car 객체(myCar)를 만든다.
  - ◆원하는 위치(x, y)로 이동한다. = (100, 0)
  - ◆20번 반복하면서 운전한다.
  - ◆운전한 영역을 Car의 color로 칠한다.

```
if __name__ == '__main__':
    register_shape("car.gif")

    myCar = Car(100, "yellow", "E-class")
    myCar.turtle.penup()
    myCar.turtle.goto(100, 0)
    myCar.turtle.pendown()
    myCar.turtle.fillcolor(myCar.color)
    myCar.turtle.begin_fill()

    for i in range(20):
        myCar.drive()
        myCar.left_turn()

    myCar.turtle.end_fill()
```

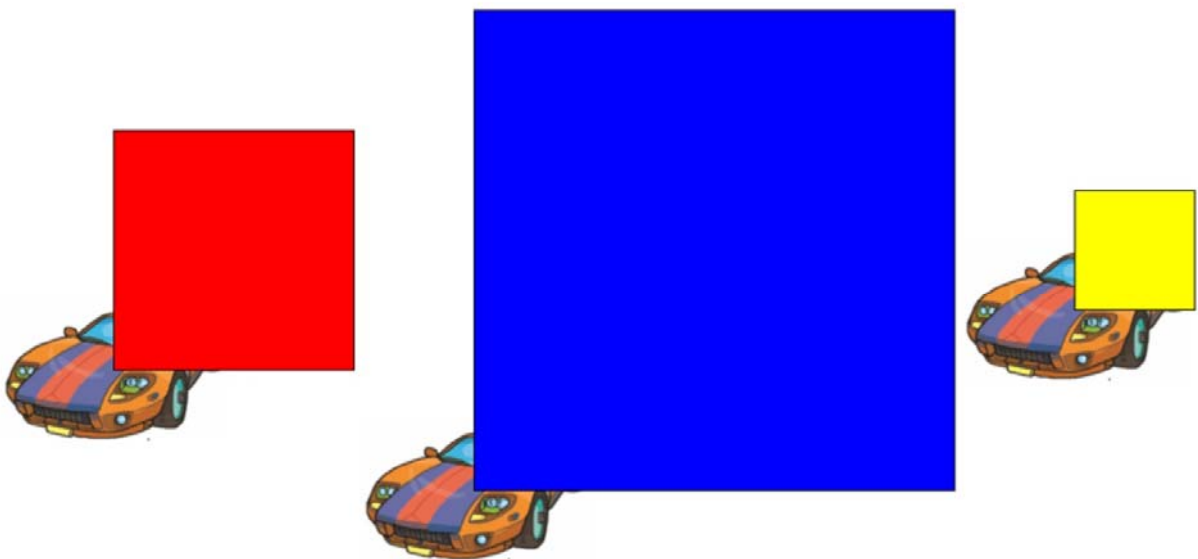
## 실습 - P09 : Car 클래스 + Turtle 클래스

●P08 실습의 코드를 다음과 같이 변형하시오.

●테스트 코드를 다음과 같이 작성한다.

- ◆Car 객체 3개(dadCar, momCar, myCar)를 만든다.
  - ❖dadCar의 speed = 200, color = 'red', model='E-Class'
  - ❖momCar의 speed = 400, color = 'red', model='E-Class'
  - ❖myCar의 speed = 100, color = 'red', model='E-Class'
- ◆3개의 Car 객체는 다음과 같은 위치에서 drive 한다.
  - ❖dadCar 의 시작 좌표(x, y) = (-500, speed/2)
  - ❖momCar 의 시작 좌표(x, y) = (-200, speed/2)
  - ❖myCar 의 시작 좌표(x, y) = (300, speed/2)
- ◆운전한 영역을 Car 의 color 로 칠한다.

## 실습 - P09 : Car 클래스 + Turtle 클래스



## 실습 - P09 : Car 클래스 + Turtle 클래스

```
if __name__ == '__main__':
    register_shape("car.gif")

    dadCar = Car(200, "red", "E-class")
    dadCar.turtle.penup()
    dadCar.turtle.goto(-500, -200/2)
    dadCar.turtle.pendown()
    dadCar.turtle.fillcolor( dadCar.color )
    dadCar.turtle.begin_fill()
    for i in range(20):
        dadCar.drive()
        dadCar.left_turn()
    dadCar.turtle.end_fill()

    momCar = Car(400, "blue", "E-class")
    momCar.turtle.penup()
    momCar.turtle.goto(-200, -400/2)
    momCar.turtle.pendown()
    momCar.turtle.fillcolor( momCar.color )
    momCar.turtle.begin_fill()
    for i in range(20):
        momCar.drive()
        momCar.left_turn()
    momCar.turtle.end_fill()
```

```
myCar = Car(100, "yellow", "E-class")
myCar.turtle.penup()
myCar.turtle.goto(300, -100/2)
myCar.turtle.pendown()
myCar.turtle.fillcolor( myCar.color )
myCar.turtle.begin_fill()
for i in range(20):
    myCar.drive()
    myCar.left_turn()
myCar.turtle.end_fill()
```

## 실습 - P10 : Car 클래스 + Turtle 클래스

- 실습 - P09 의 반복적인 부분을 runCar( ) 함수를 만들어 실행하시오.
- runCar 의 매개변수
  - ◆ carName : 생성한 car 객체를 전달한다.
  - ◆ x : carName 객체가 운전을 시작할 원점의 x 좌표 (y좌표는 speed/2 이다)
- 실행 코드는 다음과 같다.


```
if __name__ == '__main__':
    register_shape("car.gif")

    dadCar = Car(200, "red", "E-class")
    runCar( dadCar, -500 )

    momCar = Car(400, "blue", "E-class")
    runCar( momCar, -200 )

    myCar = Car(100, "yellow", "E-class")
    runCar( myCar, 300 )
```

```
def runCar( carName, x ):
```

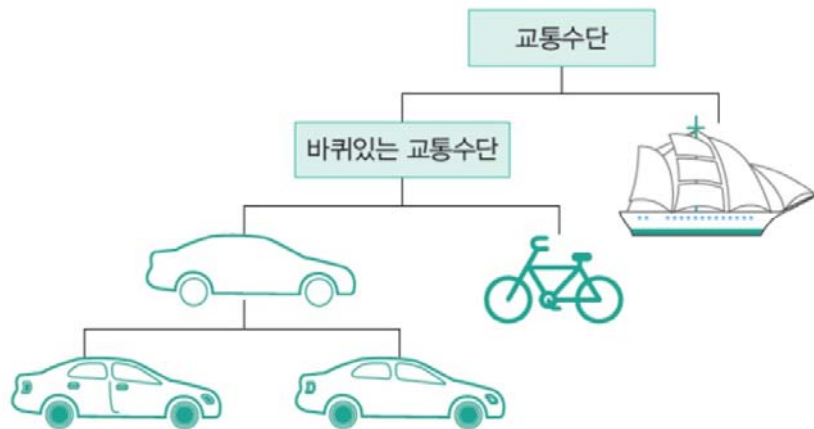




## 상속이란?

### ●상속

- 클래스를 정의할 때 부모 클래스를 지정하는 것이다.
- 자식 클래스는 부모 클래스의 메소드와 변수들을 사용할 수 있다.



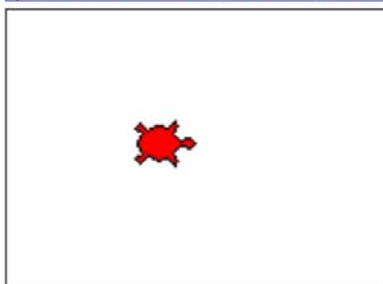
## 실습 - P11 : MyTurtle 클래스

- Turtle 클래스를 상속 받아서 MyTurtle 클래스를 작성하자.
- 터틀을 클릭하면 터틀의 색상이 빨간색으로 변경되도록 이벤트 처리함수를 추가한다.

```
from turtle import * # turtle 모듈에서 모든 것을 불러온다.

class MyTurtle(Turtle):
    def glow(self,x,y):
        self.fillcolor("red")

turtle = MyTurtle()
turtle.shape("turtle")
turtle.onclick(turtle.glow) # 거북이를 클릭하면 색상이 빨간색으로 변경된다.
```



### 도전 문제

- 리스트에 색상이 저장되어 있다.
- 거북이를 클릭할 때마다 거북이의 색을 리스트에 저장된 순서대로 변경한다.
- 리스트의 마지막 색상을 사용한 경우 다시 처음으로 돌아간다.



## 데이터 은닉 (데이터 필드 감추기)

- Circle 클래스의 radius 데이터필드 값은 외부에서 변경 가능하다.

```
import math

class Circle:
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

if __name__ == '__main__':
    c = Circle(5)
    print("Radius=", c.radius)
    print("Perimeter=", c.getPerimeter())
    print("Area=", c.getArea())

    c.radius = 5.4
    print("Radius=", c.radius)
```

1. 데이터가 부정하게 변경될 수 있다.
2. 클래스를 관리하기 어렵고 버그로부터 취약하게 만든다.

```
Radius= 5
Perimeter= 31.41592653589793
Area= 78.53981633974483
Radius= 5.4
```

## 데이터 은닉 (데이터 필드 감추기)

- 객체의 속성인 인스턴스변수(데이터필드)를 객체외부에서 직접 변경하지 못하게 하기
  - 객체의 속성에 접근하지 못하도록 해야한다.
    - ◆ private 으로 정의한다.
    - ◆ 인스턴스 변수 앞에 밑줄 두개를 붙인다.
  - private 메소드 정의
    - ◆ 메소드 이름 앞에 밑줄 두개를 붙인다.
  - private 데이터 필드와 메소드는 클래스 내부에서만 접근할 수 있다.
    - ◆ 클래스 외부에서는 접근이 불가능하다.

## 데이터 은닉 (데이터 필드 감추기)

- 객체의 속성인 인스턴스변수(데이터필드)를 객체외부에서 직접 변경하지 못하게 하기

- getter / setter

- ◆private 데이터 필드의 값을 읽기 위하여 get필드명( ) 메소드를 제공해야 한다.
- ◆private 데이터 필드의 값을 설정하기 위하여 set필드명( ) 메소드를 제공해야 한다.

```
def getPropertyName( self ):
def isPropertyName(self):                // 데이터필드가 부울인 경우
def setPropertyName(self, propertyValue):
```

## 실습 - P12 : 데이터 은닉 (데이터 필드 감추기)

- 데이터 은닉을 고려한 프로그램

```
import math

class CirclePrivate:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def getArea(self):
        return self.__radius * self.__radius * math.pi

    def setRadius(self, radius):
        if radius >= 0:
            self.__radius = radius
```

```
if __name__ == '__main__':
    c = CirclePrivate(5)
    print("Radius=", c.getRadius())
    print("Perimeter=", c.getPerimeter())
    print("Area=", c.getArea())

    c.setRadius(5.4)
    print("Radius=", c.getRadius())
    print("Perimeter=", c.getPerimeter())
    print("Area=", c.getArea())
```

```
Radius= 5
Perimeter= 31.41592653589793
Area= 78.53981633974483
Radius= 5.4
Perimeter= 33.929200658769766
Area= 91.60884177867838
```

## 이장에서 배운 것

---

- 클래스는 속성과 동작으로 이루어진다.
  - 속성은 인스턴스 변수로 표현되고 동작은 메소드로 표현된다.
- 객체를 생성하려면 생성자 메소드를 호출한다.
  - 생성자 메소드는 `__init__()` 이름의 메소드이다.
- 인스턴스 변수를 정의하려면 생성자 메소드 안에서 `self.변수이름` 과 같이 생성한다.