

# AI Programming

Lecture 16

```
def countCharacter(inFp):  
    freq = {}  
  
    inList = inFp.readlines()  
  
    for line in inList:  
        for ch in line:  
  
            ch = ch.upper()  
            if ch == " ": ch = "Space"  
            elif ch == "\n": ch = "Enter"  
  
            if ch in freq:  
                freq[ch] += 1  
            else:  
                freq[ch] = 1  
  
    return freq
```

```
def writeCountFile(freq, outFp):  
  
    while len(freq) != 0:  
        key = findMaxKey(freq)  
        outStr = f"{key} --> {freq[key]}"  
        outFp.writelines(outStr + "\n")  
        freq.pop(key)
```

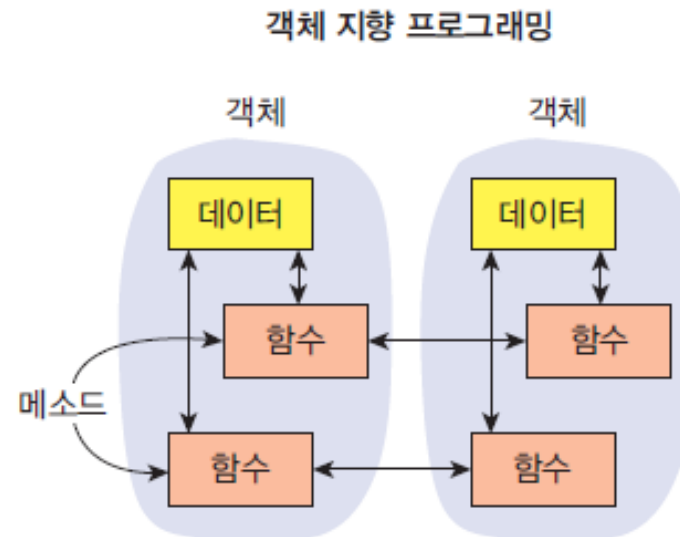
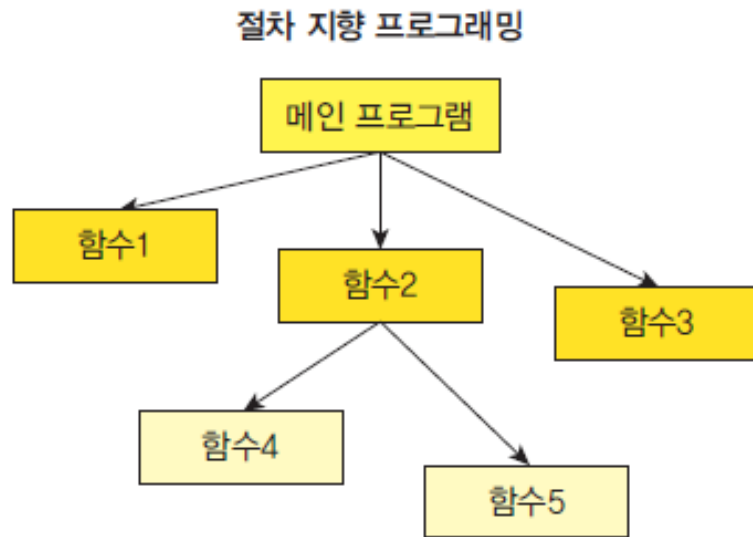
# Preview

- Ch. 12 객체지향 프로그래밍
  - 12.2 클래스
  - 12.3 생성자

# 객체 지향 프로그래밍

# Object-Oriented Programming

- 절차 지향 vs. 객체 지향 (Object-Oriented Programming)



# Object-Oriented Programming

- **Object-Oriented Programming (객체 지향 프로그래밍)**
  - 여러 사람이 프로그램을 개발할 때
    - 다른 사람이 작성한 코드를 효율적으로 사용하는 방법론
  - 함수: 특정 기능을 코드로 묶어 놓음
  - 객체 (object, instance): 함수, 변수 등을 모두 묶은 단일 프로그램
  - 클래스 (class): 객체를 만들어내는 설계도 코드

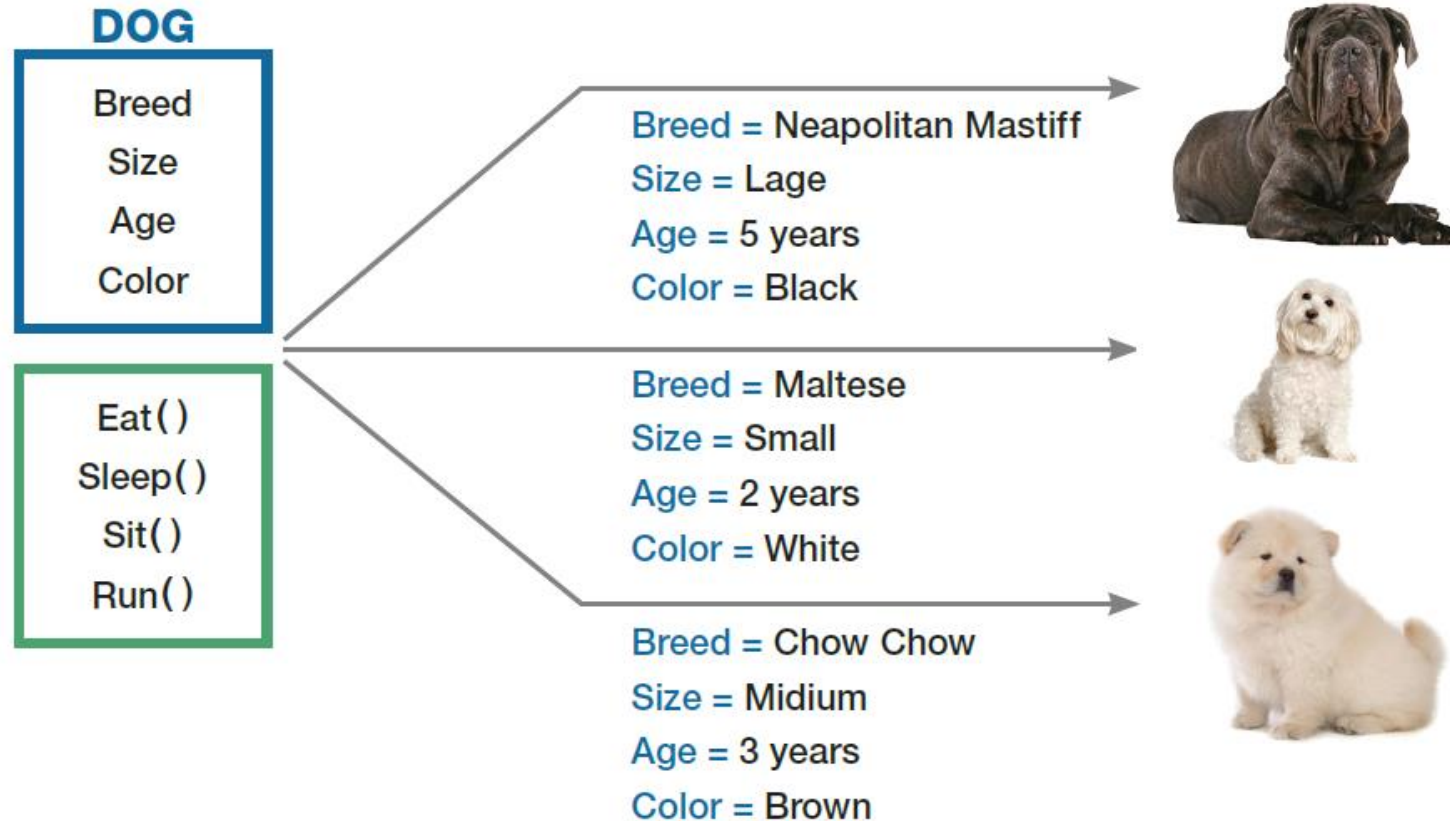
# Object-Oriented Programming

- **Example: 야구 게임**

- player (객체)
  - 속성 (변수): name, age, team, position
  - 동작 (method): pitch(), hit()
- referee (객체)
  - 속성 (변수): name
  - 동작 (method): ball(), strike()
- coach (객체)
  - 속성 (변수): name, team
  - 동작 (method): change(), shift()

# Object-Oriented Programming

- Example: “Dog” class





# Object-Oriented Programming

- Examples

- List, dictionary, string, int,...

```
>>> aa = [1, 2, 3]
>>> type(aa)
<class 'list'>
>>> aa.append(4)
>>> print(aa)
[1, 2, 3, 4]
```

```
>>> aa = {1: 'a', 2: 'b', 3: 'c'}
>>> type(aa)
<class 'dict'>
>>> aa.keys()
dict_keys([1, 2, 3])
```

```
>>> a = 10
>>> type(a)
<class 'int'>
```

```
>>> aa = '123'
>>> type(aa)
<class 'str'>
>>> aa.find('1')
0
```

## 12.2 클래스

# Class

## • Class

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



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

# Class

- **Example: “Car” class**

- Attributes (field)

- color
- speed

- Methods

- upSpeed( )
- downSpeed( )

Code12-01.py

```
1 class Car :  
2     color = ""  
3     speed = 0  
4  
5     def upSpeed(self, value) :  
6         self.speed += value  
7  
8     def downSpeed(self, value) :  
9         self.speed -= value
```

Attribute

# Class

- self
  - 자기 자신 객체
  - Class 코드 내부에서 속성변수에 접근할 때 필요
  - Method를 호출할 때 필요

```
def stop(self):  
    self.downSpeed(self.speed)
```

Code12-01.py

```
1 class Car :  
2     color = ""  
3     speed = 0  
4  
5     def upSpeed(self, value) :  
6         self.speed += value  
7  
8     def downSpeed(self, value) :  
9         self.speed -= value
```

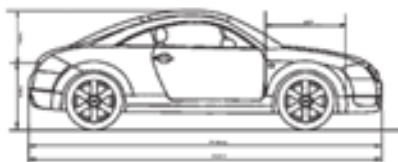
```
def printMessage() :  
    print("시험 출력이다.")
```

# Class

- **Instance**

- Class: 설계도
- Instance, object: 자동차

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()
```

# Class

- Instance

Code12-01.py

```
1 class Car :  
2     color = ""  
3     speed = 0  
4  
5     def upSpeed(self, value) :  
6         self.speed += value  
7  
8     def downSpeed(self, value) :  
9         self.speed -= value
```

```
myCar1 = Car()  
myCar2 = Car()  
myCar3 = Car()
```

# Class

- Attributes



색상  
속도



빨강  
0km



색상  
속도



파랑  
0km



색상  
속도



노랑  
0km

```
myCar1.color = "빨강"  
myCar1.speed = 0  
myCar2.color = "파랑"  
myCar2.speed = 0  
myCar3.color = "노랑"  
myCar3.speed = 0
```

- Methods

```
myCar1.upSpeed(30)  
myCar2.downSpeed(60)
```



# Class

- OOP via class

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)

# Class

- Exercise

Code12-02.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      color = ""
4      speed = 0
5
6      def upSpeed(self, value) :
7          self.speed += value
8
9      def downSpeed(self, value) :
10         self.speed -= value
```

```
11
12  ## 메인 코드 부분 ##
13  myCar1 = Car()
14  myCar1.color = "빨강"
15  myCar1.speed = 0
16
17  myCar2 = Car()
18  myCar2.color = "파랑"
19  myCar2.speed = 0
20
```

# Class

- (Cont'd)

```
21 myCar3 = Car()
22 myCar3.color = "노랑"
23 myCar3.speed = 0
24
25 myCar1.upSpeed(30)
26 print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
27
28 myCar2.upSpeed(60)
29 print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
30
31 myCar3.upSpeed(0)
32 print("자동차3의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar3.color, myCar3.speed))
```

## 출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.  
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.  
자동차3의 색상은 노랑이며, 현재 속도는 0km입니다.

## 12.3 생성자

# Constructor

- **Constructor (생성자)**

- 메인 코드에서 객체를 생성할 때 자동으로 호출되는 method
- 속성변수를 초기화 할 때 사용

- `def __init__()`:

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

# Constructor

- Example

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

# Constructor

- Example

```
## 메인 코드 부분 ##
```

```
myCar1 = Car()
```

```
myCar2 = Car()
```

```
print("자동차1의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar1.color, myCar1.speed))
```

```
print("자동차2의 색상은 %s이며, 현재 속도는 %dkm입니다." % (myCar2.color, myCar2.speed))
```

## 출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 0km입니다.

자동차2의 색상은 빨강이며, 현재 속도는 0km입니다.

# Constructor

- **Types of constructors**

- Default constructor

- A simple constructor which accepts no arguments except for `self`

- `def __init__(self):`

- Parameterized constructor

- A constructor with input parameters

- `def __init__(self, value1, value2):`



# Constructor

- Example

Code12-04.py

```
1  ## 클래스 선언 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     # Code12-03.py의 upSpeed(), downSpeed() 함수와 동일
11
12  ## 메인 코드 부분 ##
13  myCar1 = Car("빨강", 30)
14  myCar2 = Car("파랑", 60)
15
16  # Code12-03.py의 20~21행과 동일
```

## 출력 결과

자동차1의 색상은 빨강이며, 현재 속도는 30km입니다.  
자동차2의 색상은 파랑이며, 현재 속도는 60km입니다.

# Constructor

## • Exercise

Code12-05.py

```
1  ## 클래스 선언 부분 ##
2  class Car :
3      name = ""
4      speed = 0
5
6      def __init__(self, name, speed) :
7          self.name = name
8          self.speed = speed
9
10     def getName(self) :
11         return self.name
12
13     def getSpeed(self) :
14         return self.speed
```

```
16  ## 변수 선언 부분 ##
17  car1, car2 = None, None
18
19  ## 메인 코드 부분 ##
20  car1 = Car("아우디", 0)
21  car2 = Car("벤츠", 30)
22
23  print("%s의 현재 속도는 %d입니다." % (car1.getName(), car1.getSpeed()))
24  print("%s의 현재 속도는 %d입니다." % (car2.getName(), car2.getSpeed()))
```



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:\CookPython\Code12-05.py =====
아우디의 현재 속도는 0입니다.
벤츠의 현재 속도는 30입니다.
>>>
```

Ln: 12 Col: 4

# Constructor

- 정보 은닉

- 객체의 속성변수를 메인코드에서 직접 접근/변경하지 못하도록 은닉
  - Private 속성변수

```
class Student:  
    def __init__(self, name=None, age=0):  
        self.__name = name # __가 변수 앞에 붙으면 외부에서 변경 금지  
        self.__age = age    # __가 변수 앞에 붙으면 외부에서 변경 금지
```

```
obj=Student()  
print(obj.__age)
```

AttributeError: 'Student' object has no attribute '\_\_age'

# Constructure

- 접근자 (getters) & 설정자 (setters)
  - 은닉된 속성변수값을 접근 (get) & 변경 (set)

```
class Student:
    def __init__(self, name=None, age=0):
        self.__name = name
        self.__age = age

    def getAge(self):
        return self.__age

    def getName(self):
        return self.__name
```



```
def setAge(self, age):
    self.__age=age

def setName(self, name):
    self.__name=name
```

```
obj=Student("Hong", 20)
obj.getName()
```

# Review

- OOP via class

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

# Assignment 14

- “BankAccount” Class 구현

- Attributes

- name: 계좌 주인 이름
    - pin: PIN
    - balance: 잔액

- Methods

- checkPin(pin): PIN이 맞는지 확인
    - getPIN(): PIN 확인 (접근자)
    - getBalance(): 잔액 확인 (접근자)
    - setBalance(): 잔액 확인 (설정자)
    - deposit(amount): 입금
    - withdraw(amount): 출금

# Assignment 14

## • “BankAccount” Class 구현

### • 메인 코드 및 실행 결과

```
from bankaccount import BankAccount

user = BankAccount("홍길동", 1234, 30000) ## 이름, PIN, 금액

pin = int(input("ATM) PIN을 입력하시오: "))

while not user.checkPin(pin):
    pin = int(input("ATM) PIN이 다릅니다. 다시 입력하시오: "))

while True:
    ctr = int(input("ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> "))

    if ctr == 0:
        print("ATM) 감사합니다.")
        break
    elif ctr == 1:
        print("ATM) 계좌 잔액은 %d원 입니다." % user.getBalance())
    elif ctr == 2:
        amount = int(input("ATM) 입금할 금액을 입력하시오: "))
        user.deposit(amount)
    elif ctr == 3:
        amount = int(input("ATM) 출금할 금액을 입력하시오: "))
        user.withdraw(amount)
    else:
        print("ATM) 잘못된 입력입니다.")
```

```
ATM) PIN을 입력하시오: 5678
ATM) PIN이 다릅니다. 다시 입력하시오: 1234
ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> 1
ATM) 계좌 잔액은 30000원 입니다.
ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> 2
ATM) 입금할 금액을 입력하시오: 3000
ATM) 입금 후 잔액은 33000원 입니다.
ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> 3
ATM) 출금할 금액을 입력하시오: 50000
ATM) 잔액이 부족합니다.
ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> 3
ATM) 출금할 금액을 입력하시오: 30000
ATM) 출금 후 잔액은 3000원 입니다.
ATM) 0: 종료, 1: 잔액 확인, 2: 입금, 3: 출금 --> 0
ATM) 감사합니다.
```

# Assignment 14

- “BankAccount” Class 구현

- 파일 이름: bankaccount.py

- Method 구현

- `__init__`: 생성자
    - `getPIN()`: PIN 속성변수 return (접근자)
    - `getBalance()`: balance 속성변수 return (접근자)
    - `setBalance(balance)`: balance 속성변수를 balance로 변경 (설정자)

```
class BankAccount():  
  
    def __init__(self, name, pin, balance=10000):  
        self.__name = name  
        self.__pin = pin  
        self.__balance = balance  
  
    def getPIN(self):  
        ## 코드 작성 ##  
  
    def getBalance(self):  
        ## 코드 작성 ##  
  
    def setBalance(self, balance):  
        ## 코드 작성 ##  
  
    def checkPin(self, pin):  
        ## 코드 작성 ##  
  
    def deposit(self, amount):  
        ## 코드 작성 ##  
  
    def withdraw(self, amount):  
        ## 코드 작성 ##
```



# Assignment 14

- (Cont'd)

- `checkPin(pin)`: PIN이 맞는지 확인
  - Return: `True` or `False`
  - 속성변수 직접 접근 금지
- `deposit(amount)`: 입금
  - 속성변수 직접 접근 & 변경 금지
- `withdraw(amount)`: 출금
  - 속성변수 직접 접근 & 변경 금지