파이썬 객체

Charles Severance

Python for Everybody
www.py4e.com

경고

- 이 강의는 객체의 정의와 기능들에 중점을 두고 있음
- 이 강의는 “어떻게 사용하는가”보다는 “어떤 식으로 사용되는가”에 가까움
- 실제 문제 안에서 이것들을 살펴보기 전까지는 전체 그림을 보기 어려울 수 있음
- 불신을 거두고 뒤의 40여장의 슬라이드에서 많은 것을 배우길 당부

5. Data Structures

This chapter describes some things you've learned about already in more detail, and adds some new things as well.

5.1. More on Lists

The list data type has some more methods. Here are all of the methods of list objects:

`list.`**`append`**`(x)`
Add an item to the end of the list. Equivalent to `a[len(a):] = [x]`.

`list.`**`extend`**`(L)`
Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`.

`list.`**`insert`**`(i, x)`
Insert an item at a given position. The first argument is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list, and `a.insert(len(a), x)` is equivalent to `a.append(x)`.

`list.`**`remove`**`(x)`
Remove the first item from the list whose value is *x*. It is an error if there is no such item.

`list.`**`pop`**`([i])`
Remove the item at the given position in the list, and return it. If no index is specified, `a.pop()` removes and returns the last item in the list. (The square brackets around the *i* in the method signature denote that the parameter is optional, not that you should type square brackets at that position. You will see this notation frequently in the Python Library Reference.)

https://docs.python.org/3/tutorial/datastructures.html

12.6. `sqlite3` — DB-API 2.0 interface for SQLite databases

Source code: Lib/sqlite3/

SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by PEP 249.

To use the module, you must first create a `Connection` object that represents the database. Here the data will be stored in the `example.db` file:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

You can also supply the special name `:memory:` to create a database in RAM.

Once you have a `Connection`, you can create a `Cursor` object and call its `execute()` method to perform SQL commands:

```
c = conn.cursor()

# Create table
c.execute('''CREATE TABLE stocks
             (date text, trans text, symbol text, qty real, price real)''')
```

https://docs.python.org/3/library/sqlite3.html

프로그램을 보며
시작해봅시다

```
inp = input('Europe floor?')
usf = int(inp) + 1
print('US floor', usf)
```

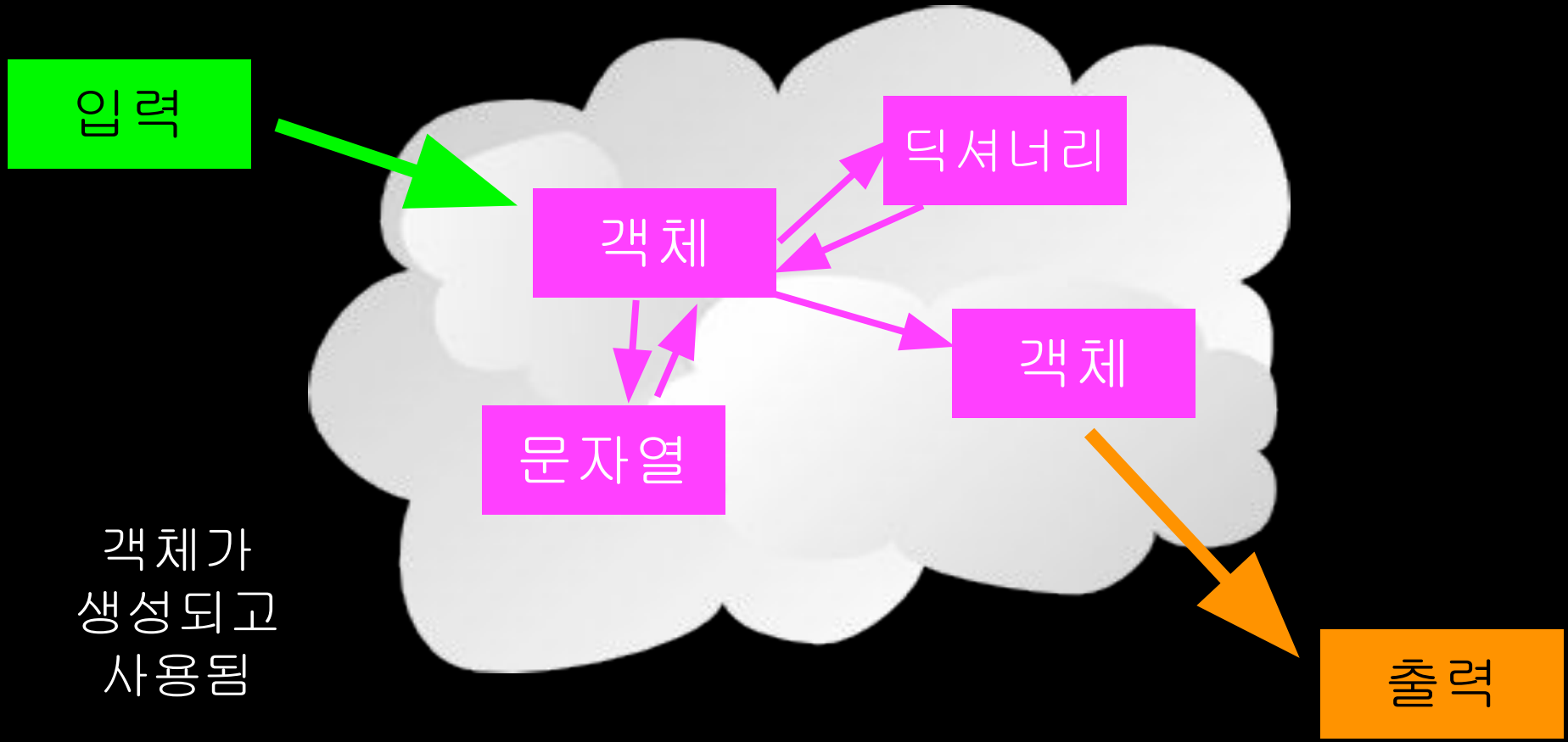
Europe floor? 0
US floor 1

객체 지향

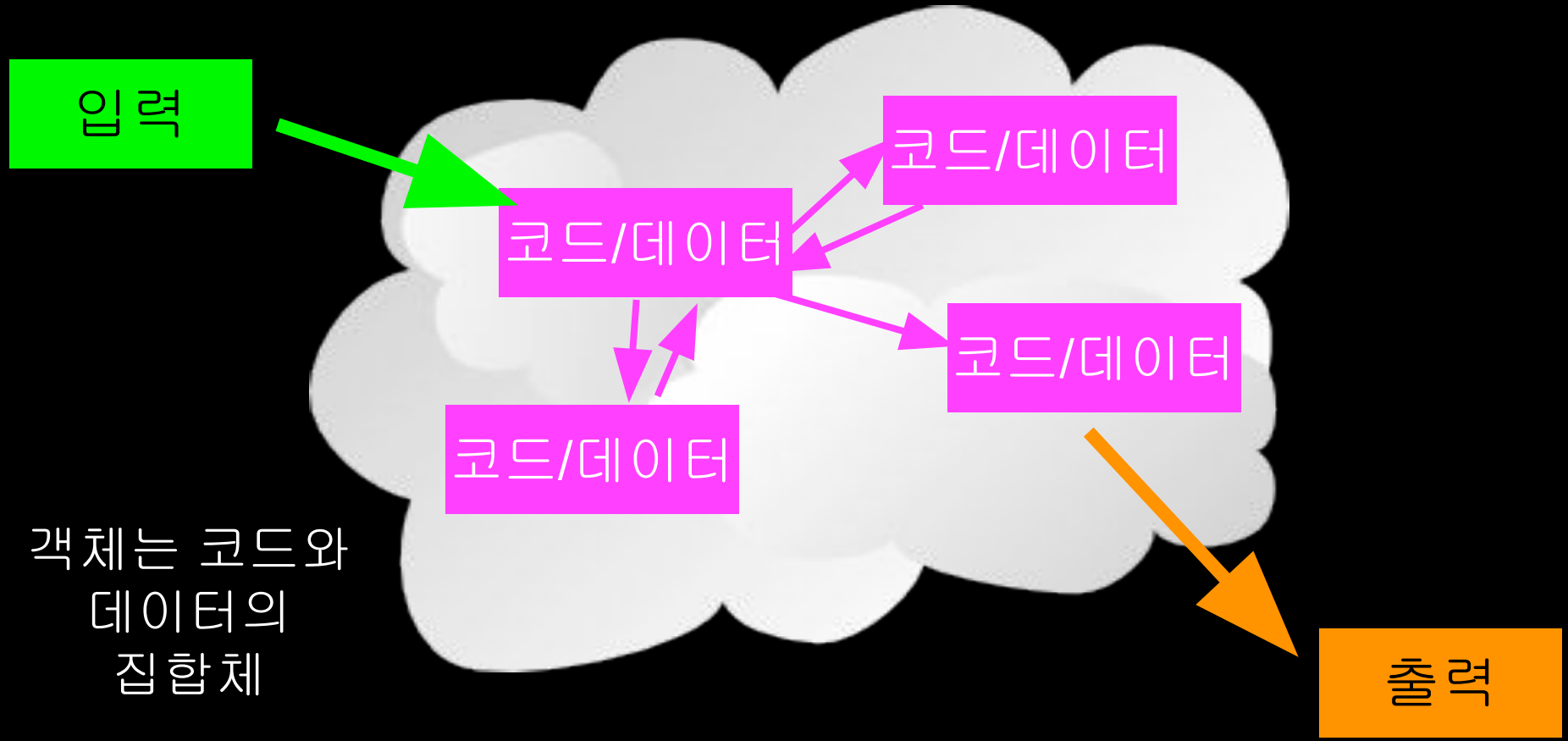
- 프로그램은 서로 협력하는 여러 개의 객체로 구성
- “전체 프로그램”이 아닌 각각의 객체가 마치 프로그램 안의 “섬”같이 서로 협력하여 작동
- 프로그램은 함께 실행되는 여러 개의 객체로 구성 - 객체는 서로의 기능들을 활용

객체

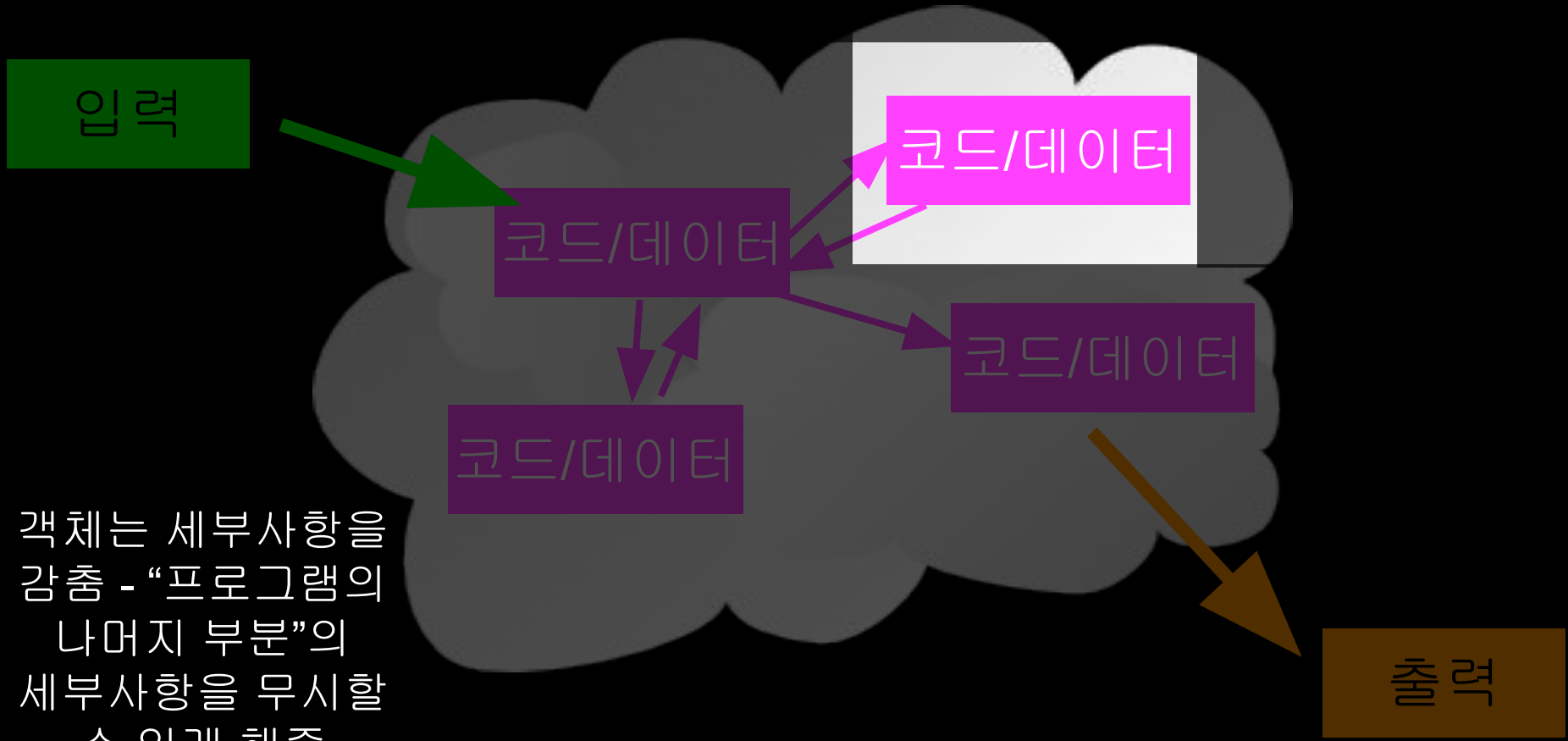
- 객체는 하나의 자족적인 코드와 데이터
- 객체 지향 접근의 요점은 문제를 이해가능한 작은 문제로 분할하여 접근(분할 정복 divide-and-conquer)
- 객체는 우리가 필요없는 세부사항들을 무시할 수 있도록 경계를 제공
- 우리는 객체를 계속 사용해 왔음: 문자열 객체, 정수형 객체, 딕셔너리 객체, 리스트 객체...



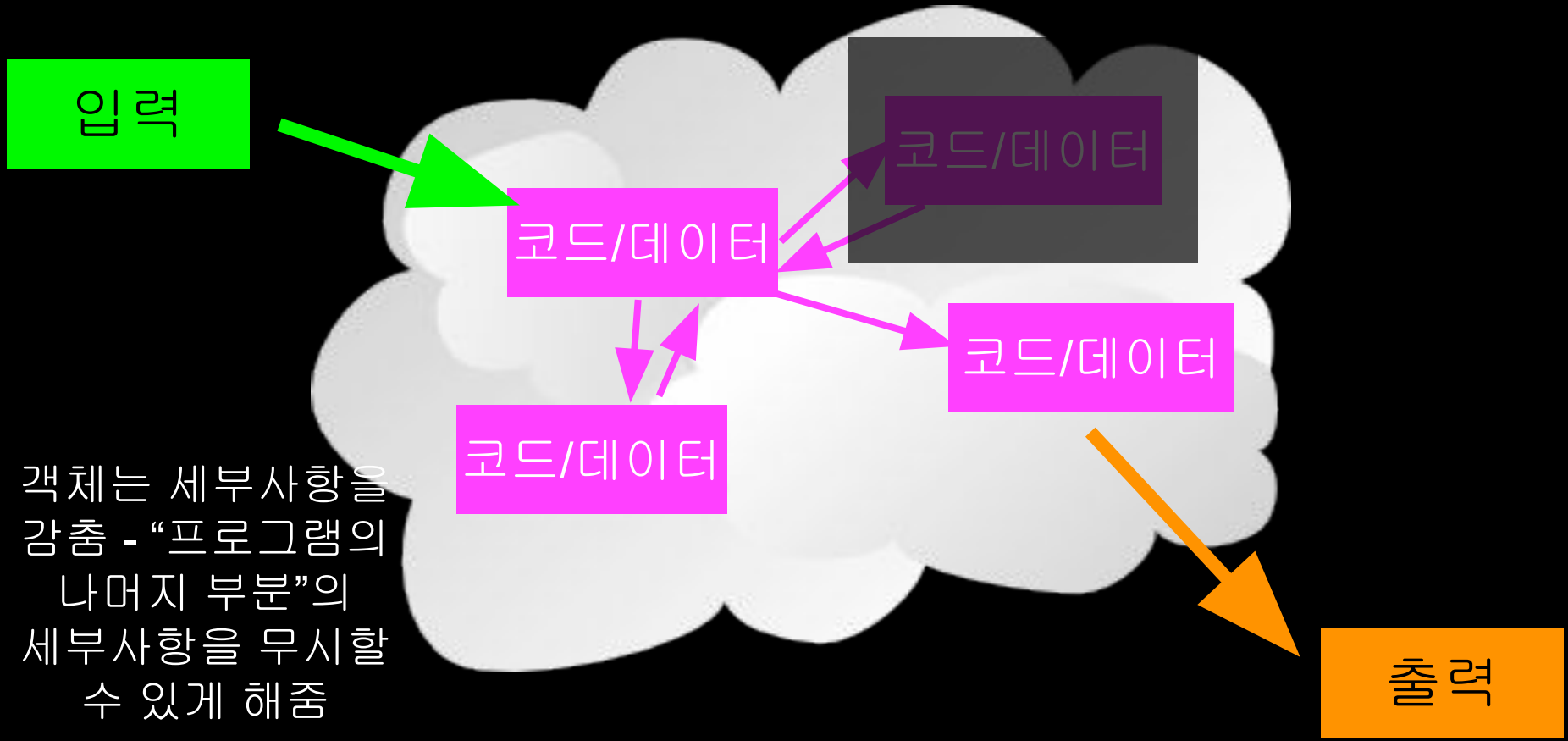
객체가
생성되고
사용됨



객체는 코드와
데이터의
집합체



객체는 세부사항을 감춤 - “프로그램의 나머지 부분”의 세부사항을 무시할 수 있게 해줌



객체는 세부사항을 감춤 - “프로그램의 나머지 부분”의 세부사항을 무시할 수 있게 해줌

정의

- 클래스 Class - 하나의 형식, 템플릿
- 메소드 Method or Message - 클래스 내에 정의된 기능
- 필드/속성 Field or attribute- 클래스 내의 데이터
- 객체/인스턴스 Object or Instance - 클래스의 한 인스턴스

용어: 클래스 Class

어떤 물체(객체)의 특징(필드 또는 속성)과 행동(메소드, 연산 등의 기능) 등 추상적인 특성을 정의. 클래스를 어떤 것의 특성을 설명하는 설계도 혹은 공장이라고도 이야기함. 예를 들어, 개 라는 클래스는 품종 또는 털색깔(특성), 혹은 짖거나 앉는 행위(행동) 등 개들이 가지는 특성을 가짐

http://en.wikipedia.org/wiki/Object-oriented_programming

용어: 인스턴스 Instance

클래스 안에서 인스턴스 혹은 특정 객체를 가질 수 있음.
인스턴스란 실행 중 실제로 생성된 객체를 의미.
프로그래머의 용어를 따르면, “래씨"라는 객체는 “개”라는 클래스의 한 인스턴스임. 특정 객체의 특성들을 모아놓은 것을 상태라고 함. 객체는 클래스 안에서 정의된 상태와 행동으로 구성됨.

객체와 인스턴스는 자유롭게 바꾸어 부를 수 있다.

http://en.wikipedia.org/wiki/Object-oriented_programming

용어: 메소드 Method

객체의 능력. 언어로 따지만 메소드는 동사임. 래시는 개로써 짖을 수 있는 능력이 있음. 그래서 bark() “래시”의 한 메소드가 됨. 이것 외에 다른 메소드들도 가질 수 있음. 예를 들면 sit(), eat(), walk() 또는 save_timmy() 등. 프로그램 내에서는 메소드는 하나의 객체에만 영향을 줄 수 있음. 모든 개는 짖을 수 있지만, 실제로 짖는 행위는 수행하는 개는 하나 밖에 없음.

메소드와 메시지는 자유롭게 바꾸어 부를 수 있음

http://en.wikipedia.org/wiki/Object-oriented_programming

파이썬 객체 예제

```
>>> x = 'abc'
>>> type(x)
<class 'str'>
>>> type(2.5)
<class 'float'>
>>> type(2)
<class 'int'>
>>> y = list()
>>> type(y)
<class 'list'>
>>> z = dict()
>>> type(z)
<class 'dict'>
```

```
>>> dir(x)
[ … 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find',
'format', … 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
>>> dir(y)
[… 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
>>> dir(z)
[…, 'clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

클래스 예시

클래스는 예약어

각각의 PartyAnimal 객체에는 일정량의 코드가 있음

객체에게 party() 를 실행하라고 알려줌

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

PartyAnimal 객체를 만드는 템플릿

각각의 PartyAnimal 객체에는 일정량의 데이터가 있음

PartyAnimal 객체를 생성하여 an 에 저장

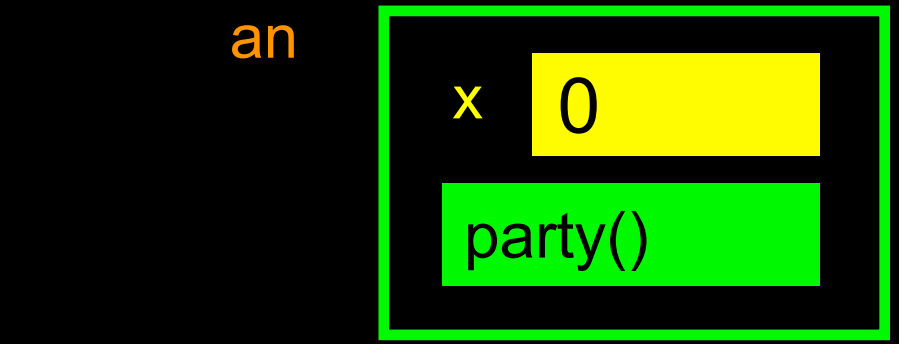
PartyAnimal.party(an)

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
```

```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

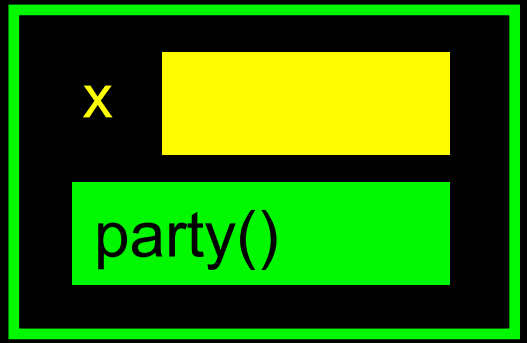
```
$ python party1.py
```



```
class PartyAnimal:
  x = 0

  def party(self) :
    self.x = self.x + 1
    print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

```
$ python party1.py
So far 1
So far 2
So far 3
```

an
self



PartyAnimal.party(an)

dir() 과 type() 다루기

기능을 찾는 한 방법

- dir() 명령은 여러 기능들을 나열한다
- __ 표시되어있는 것들은 무시해도 된다 - 파이썬이 자체적을 사용
- 나머지는 객체가 실제로 수행할 수 있는 작업이다
- type() 과 유사함 - 어떤 변수에 "대해서" 말해준다

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(x)
['__add__', '__class__',
'__contains__', '__delattr__',
'__delitem__', '__delslice__',
'__doc__', … '__setitem__',
'__setslice__', '__str__',
'append', 'clear', 'copy',
'count', 'extend', 'index',
'insert', 'pop', 'remove',
'reverse', 'sort']
>>>
```

```
class PartyAnimal:
    x = 0

    def party(self) :
      self.x = self.x + 1
      print("So far",self.x)

an = PartyAnimal()

print("Type", type(an))
print("Dir ", dir(an))
```

dir() 을 사용해서
새롭게 생성된
클래스의 “기능” 을
찾을 수 있음

```
$ python party3.py
Type <class '__main__.PartyAnimal'>
Dir  ['__class__', ... 'party', 'x']
```

dir() 를 문자열에 사용

```
>>> x = 'Hello there'
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__',
'__doc__', '__eq__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__getslice__', '__gt__',
'__hash__', '__init__', '__le__', '__len__', '__lt__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
'lower', 'lstrip', 'partition', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```

객체 생명주기

http://en.wikipedia.org/wiki/Constructor_(computer_science)

객체 생명주기

- 객체는 생성되고, 사용되어지고, 없어짐
- 객체를 불러올 때 특별한 코드 (메소드) 가 존재
  - 생성되어 질 때 (생성자)
  - 소멸되어 질 때 (소멸자)
- 생성자는 자주 사용
- 소멸자는 거의 사용되지 않음

생성자

생성자의 주된 목적은 인스턴스 변수가 객체가 생성될 때 적절한 초기값을 가지게 하는 것

```
class PartyAnimal:
   x = 0

   def __init__(self):
     print('I am constructed')

   def party(self) :
     self.x = self.x + 1
     print('So far',self.x)

   def __del__(self):
     print('I am destructed', self.x)

an = PartyAnimal()
an.party()
an.party()
an = 42
print('an contains',an)
```

```
$ python party4.py
I am constructed
So far 1
So far 2
I am destructed 2
an contains 42
```

생성자와 소멸자는 필수가 아님.
생성자는 흔히 초기화에 사용되고.
소멸자는 거의 사용되지 않음

생성자

객체 지향 프로그래밍에서, 클래스의 생성자는 객체가 생성되어질 때 불러오는 문장

http://en.wikipedia.org/wiki/Constructor_(computer_science)

여러 인스턴스

- 다수의 객체를 만들 수 있음 - 클래스가 그 형식의 틀을 제공
- 별개의 객체들을 각자의 변수에 저장할 수 있음
- 이것을 동일 클래스의 다중 인스턴스 라고 함
- 각각의 인스턴스는 자신만의 인스턴스 변수를 가지고 있음

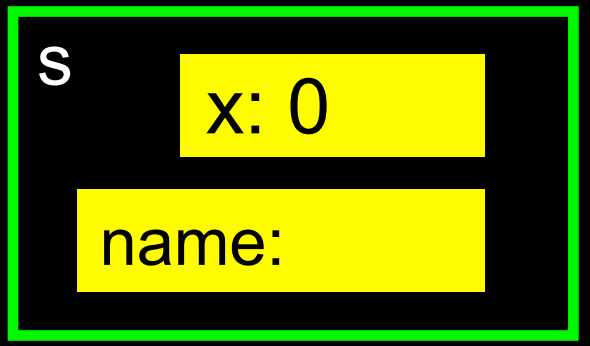
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

생성자는 추가적인 매개 변수를 가질 수 있음. 이것이 클래스의 특정 인스턴스의 인스턴스 변수를 설정하는데 사용.

party5.py

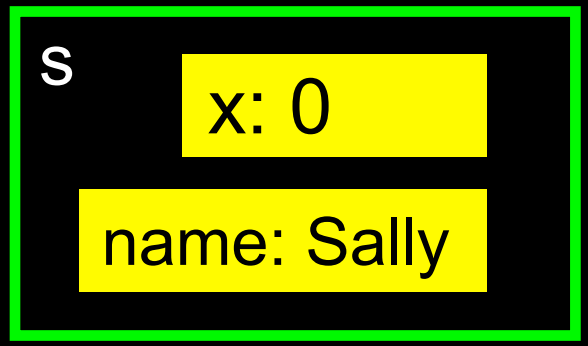
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

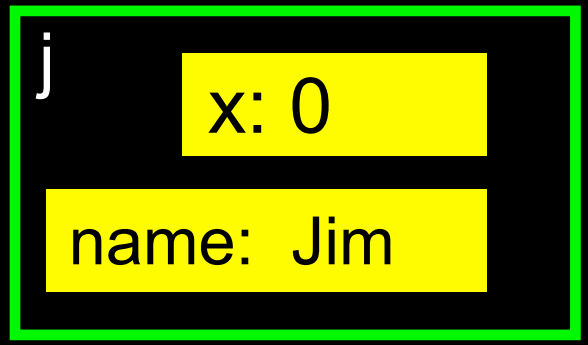
```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```



```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

두 개의 구별된
인스턴스를
가지게 된다





```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, z):
      self.name = z
      print(self.name,"constructed")

    def party(self) :
      self.x = self.x + 1
      print(self.name,"party count",self.x)

s = PartyAnimal("Sally")
j = PartyAnimal("Jim")

s.party()
j.party()
s.party()
```

```
Sally constructed
Jim constructed
Sally party count 1
Jim party count 1
Sally party count 2
```

상속

http://www.ibiblio.org/g2swap/byteofpython/read/inheritance.html

상속

- 새로운 클래스를 만들 때 - 기존에 있던 클래스를 재사용하여 그 모든 기능을 상속받고 조금 추가하여 새로운 클래스를 만들 수 있음
- 저장하고 재사용하는 또 다른 형태
- 한 번 작성하고 - 여러번 재사용
- 새로운 클래스는 (child) 이전 클래스 (parent) 의 모든 기능을 가지고 있음 - 그리고 더 추가할 수 있음

용어: 상속

‘하위 클래스’란 클래스의 특수한 버전이며,
상위 클래스의 속성과 행동을 상속하고, 자신의 것도 추가

http://en.wikipedia.org/wiki/Object-oriented_programming

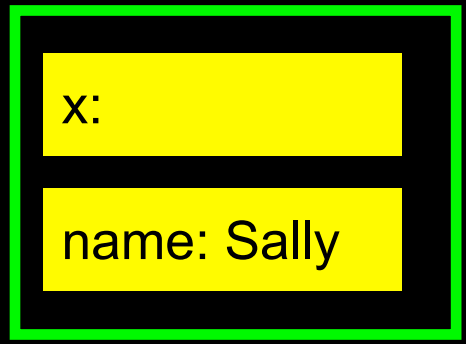
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```

FootballFan은 PartyAnimal을 확장한 클래스. PartyAnimal의 모든 것을 담고 있고 추가적인 내용을 또 가짐.

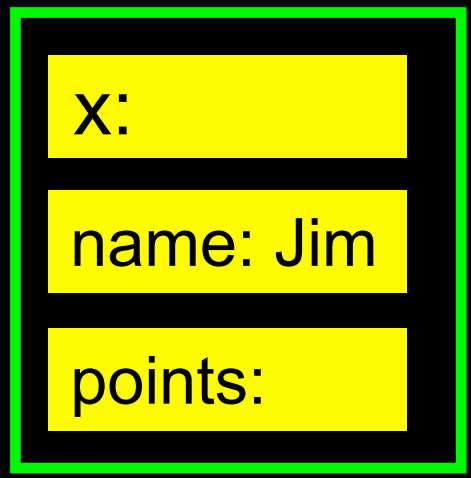
```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



```
class PartyAnimal:
   x = 0
   name = ""
   def __init__(self, nam):
     self.name = nam
     print(self.name,"constructed")

   def party(self) :
     self.x = self.x + 1
     print(self.name,"party count",self.x)

class FootballFan(PartyAnimal):
   points = 0
   def touchdown(self):
      self.points = self.points + 7
      self.party()
      print(self.name,"points",self.points)
```

```
s = PartyAnimal("Sally")
s.party()

j = FootballFan("Jim")
j.party()
j.touchdown()
```



정의

- 클래스 Class - 형식 또는 템플릿
- 속성 Attribute – 클래스 내의 변수
- 메소드 Method - 클래스 내의 함수
- 객체 Object - 클래스의 특정 인스턴스
- 생성자 Constructor – 객체가 생성될 때 실행되는 코드
- 계승 Inheritance - 기존의 클래스를 확장하여 새로운 클래스를 만드는 것

요약

- 객체 지향 프로그래밍은 코드 재사용에 매우 구조화된 접근
- 데이터와 기능들을 함께 묶어서 클래스 내에 여러 독립적인 인스턴스를 만들 수 있음

Acknowledgements / Contributions

Thes slide are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

...

Initial Development: Charles Severance, University of Michigan School of Information

Contributor:

- Seung-June Lee (plusjune@gmail.com)
- Connect Foundation

Translator:

- Jo Ha Nul
- Jeungmin Oh (tangza@gmail.com)

Additional Source Information

- Snowman Cookie Cutter" by Didriks is licensed under CC https://www.flickr.com/photos/dinnerseries/23570475099

- Photo from the television program *Lassie*. Lassie watches as Jeff (Tommy Rettig) works on his bike is

  https://en.wikipedia.org/wiki/Lassie#/media/File:Lassie_and_Tommy_Rettig_1956.JPG