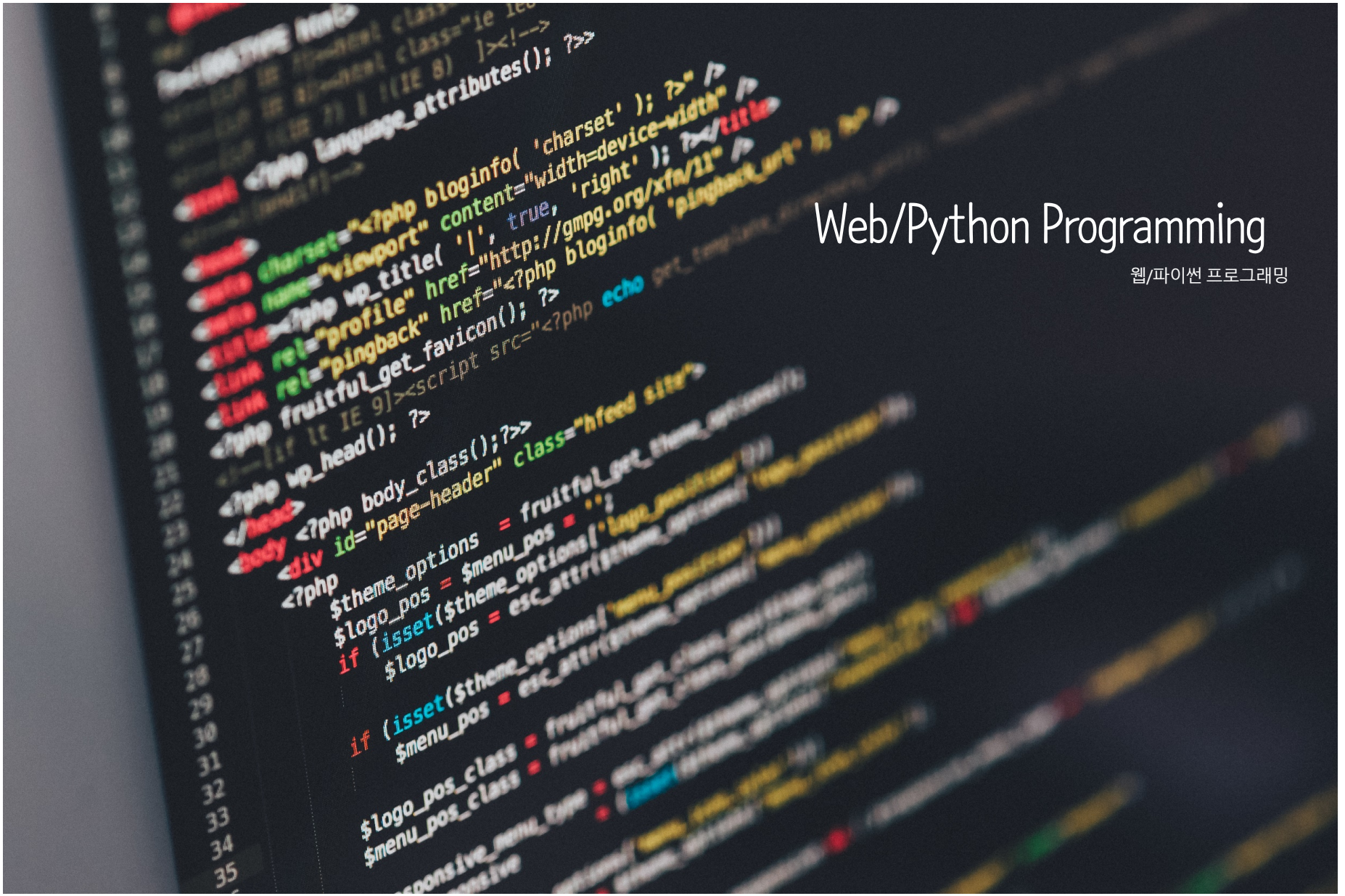


Web/Python Programming

웹/파이썬 프로그래밍

Today

- Object-oriented Programming
  - Object (= Instance, or object – mother of all classes in Python)
  - Class (= Type)
  - Method (= Member Function)
  - Inheritance

Types

- Integer
- Float
- String
- Boolean
- List
- Set
- Tuple
- Dictionary

```
>>> type(17)
<class 'int'>
```

- Type is a class
- How to make your own type

Object and variable

- Every location in the computer's memory has a **memory address**
- **Object**: a value (or thing) at a memory address with a type

`26.0` `id1` `float`

- **Variable** contains the memory address of the object

`degrees_celsius`

Function isinstance

- Function `isinstance` reports whether an object is an instance of a class – whether an object has a particular type:

```
>>> isinstance('abc',str)
True
>>> isinstance(55.2,str)
False
>>> isinstance(55.2,float)
True
```

- ‘abc’ is an instance of `str`, but 55.2 is not.
- 55.2 is an instance of `float`.

What is Class?

- https://en.wikipedia.org/wiki/Class_(computer_programming)

Class Student

- student.py

```
class Student:
    name = ""
    id = 0
    gender = "female"

a = Student()
b = Student()
c = Student()

a.name = "Harry Potter"
a.id = 2017103701
a.gender = "male"

b.name = "Hermione Granger"
b.id = 2018103722
b.birthyear = 1999

c.name = "Ron Weasley"
```

- Student is a class.
- a, b, and c are the instances of the class Student.
- name, id, gender, birthyear are instance variables.
- a = Student() creates a Student object and then assigns that object to variable a.

```
>>> isinstance('abc',str)
True
>>> isinstance(55.2,str)
False
>>> isinstance(55.2,float)
True
```

Method

```
>>> 'browning'.capitalize()
'Browning'
>>> str.capitalize('browning')
'Browning'

>>> a.num_courses()
2
>>> Student.num_courses(a)
2
```

```
class Student:
    name = ""
    id = 0
    gender = "female"
    course = []

    def num_courses(self):
        return len(self.course)

a = Student()
b = Student()
c = Student()


a.name = "Harry Potter"
a.id = 2017103701
a.gender = "male"

b.name = "Hermione Granger"
b.id = 2018103722
b.birthyear = 1999

c.name = "Ron Weasley"

a.course = ["English","Programming"]
b.course = ["Writing","Physics","Programming"]
c.course = ["Programming"]
```

Inheritance

```
>>> isinstance('abc',str)
True
>>> isinstance(55.2,str)
False
>>> isinstance(55.2,float)
True

>>> help(object)
Help on class object in module builtins:

class object
 |  The most base type
>>> isinstance(55.2,object)
True
>>> isinstance('abc',object)
True
>>> isinstance(str, object)
True
>>> isinstance(max, object)
True
```

- 'abc' is an instance of class `str`
- 55.2 is an instance of class `float`
- 'abc' and 55.2 are instances of class `object`
- Classes and functions are instances of `object`
- Every class in Python is derived from class `object`, so every instance of every class is an `object`.
- Class `object` is the superclass of class `str`, and class `str` is a subclass of class `object`.

Inheritance

```
>>> dir(object)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge
__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '
__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '_
_sizeof__', '__str__', '__subclasshook__']
```

- `dir` shows a list of attributes (attributes are variables inside a class that refer to methods, functions, variables, or other classes)
- Every class in Python inherits these attributes from class `object`: they are automatically part of every class.
- Every subclass inherits the features of its superclass.
- It helps avoid a lot of duplicate code and makes interactions between related types consistent.

Inheritance

```
>>> dir(object)
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge
__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '
__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '_
_sizeof__', '__str__', '__subclasshook__']
>>> class Book:
        """Information about a book"""


>>> type(str)
<class 'type'>
>>> type(Book)
<class 'type'>
>>> dir(Book)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__form
at__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__',
 '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__
repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref_
_']
```

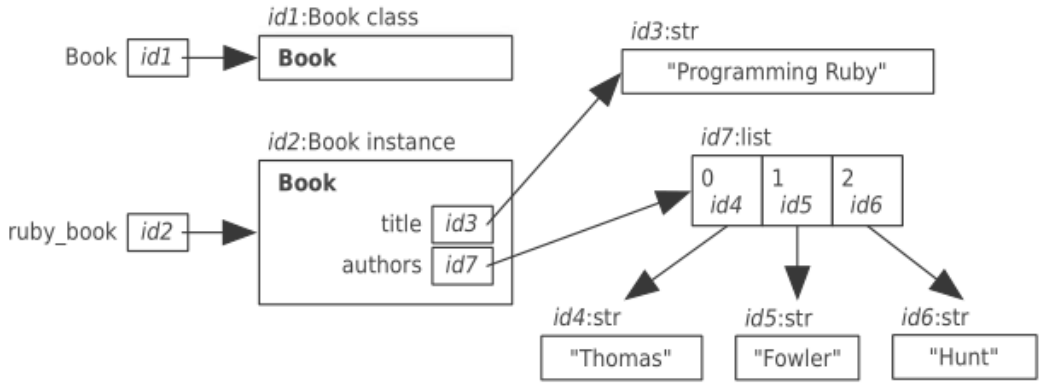
‘__dict__’, ‘__module__’, ‘__qualname__’, ‘__weakref__’

Book class

book.py - C:/Users/jiyoung/AppData/Local/Programs/Python/Python35/Scripts/class/book.py (3.5.3)

File Edit Format Run Options Window Help

```
class Book:
    """Information about a book"""

ruby_book = Book()
ruby_book.title = "Programming Ruby"
ruby_book.authors = ['Thomas', 'Fowler', 'Hunt']
```




```
>>> ruby_book.title
'Programming Ruby'
>>> ruby_book.authors
['Thomas', 'Fowler', 'Hunt']
>>> help(Book)
Help on class Book in module __main__:

class Book(builtins.object)
 |  Information about a book
 |
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Book class - method

```
class Book:
    """Information about a book"""

    def num_authors(self):
        """(Book) -> int

        Return the number of authors of this book.
        """
        return len(self.authors)
```

```
>>> import book
>>> ruby_book = book.Book()
>>> ruby_book.title = "Programming Ruby"
>>> ruby_book.authors = ['Thomas','Fowler','Hunt']
>>> book.Book.num_authors(ruby_book)
3
>>> ruby_book.num_authors()
3
```

- `book` is the imported module
- In that module, there is class `Book`
- Inside `Book`, there is method `num_authors`.
- The argument to the call, `ruby_book`, is passed to parameter `self`.

```
>>> 'browning'.capitalize()
'Browning'
>>> str.capitalize('browning')
'Browning'
```

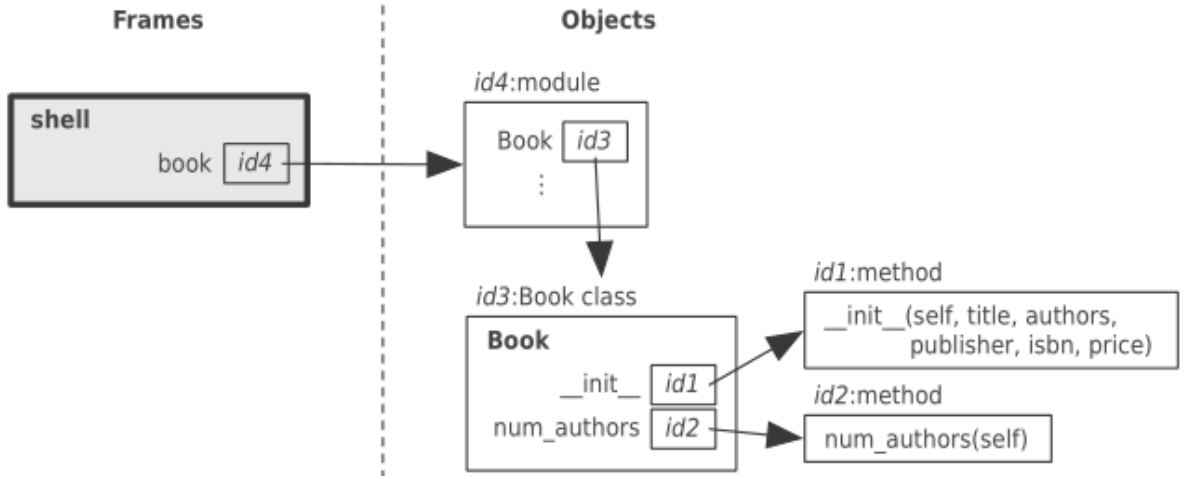
__init__ method

```
class Book:
    """Information about a book"""

    def __init__(self, title, authors, publisher, isbn, price):
        """(Book, str, list of str, str, str, number) -> NoneType
        Create a new book entitled title, written by the people in authors,
        published by publisher, with ISBN isbn and costing price dollars.
        """
        self.title = title
        self.authors = authors[:]
        self.publisher = publisher
        self.ISBN = isbn
        self.price = price


    def num_authors(self):
        """(Book) -> int

        Return the number of authors of this book.
        """
        return len(self.authors)
```

```
>>> import book
>>> ruby_book = book.Book()
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    ruby_book = book.Book()
TypeError: __init__() missing 5 required positional
arguments: 'title', 'authors', 'publisher', 'isbn',
and 'price'
>>> python_book = book.Book('Practical Programming',
['Campbell','Gries','Montojo'],'Pragmatic Bookshelf'
,'978-1-93778-545-1',25.0)
>>> python_book.title
'Practical Programming'
>>> python_book.authors
['Campbell', 'Gries', 'Montojo']
>>> python_book.price
25.0
>>> python_book.num_authors()
3
```

Constructor

- The module `book` contains a single statement: the class definition.



- Method `__init__` is called whenever a Book object is created, to initialize the new object. (=a constructor)

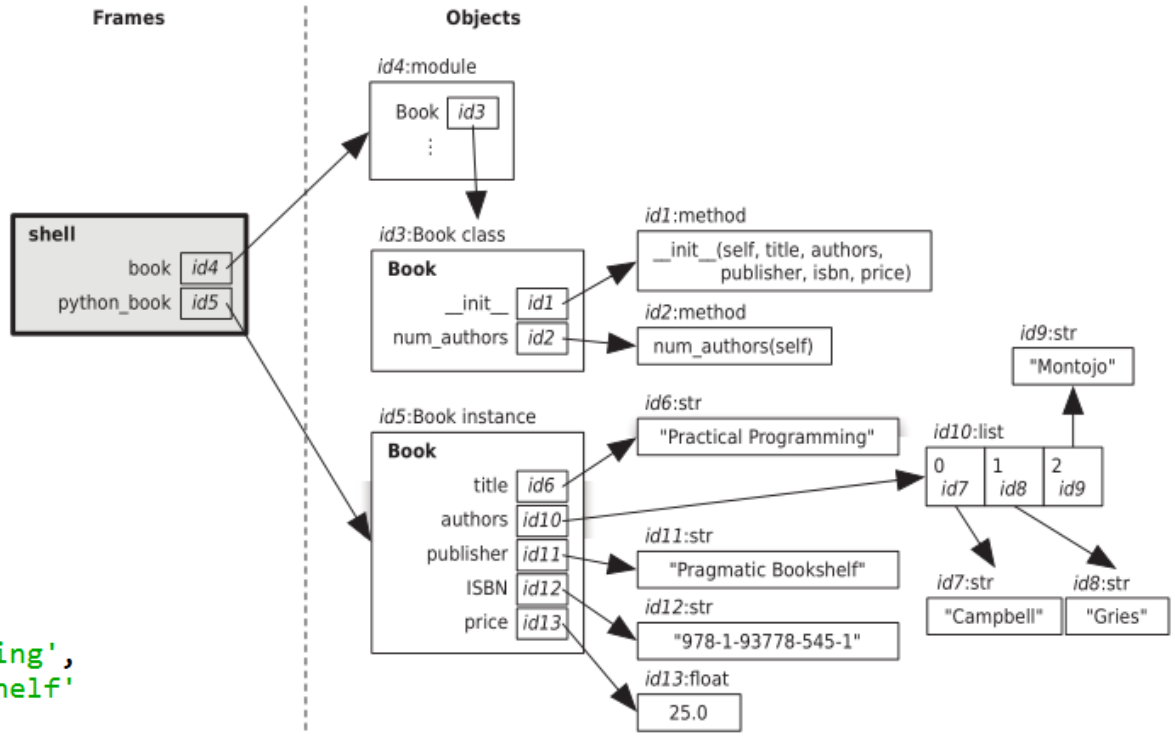
Constructor

- The steps that Python follows when creating an object:
  - It creates an object at a particular memory address.
  - It calls method `__init__`, passing in the new object into the parameter `self`.
  - It produces that object's memory address.

book.py

```
class Book:
    """Information about a book"""

    def __init__(self, title, authors, publisher, isbn, price):
        """(Book, str, list of str, str, str, number) -> NoneType
        Create a new book entitled title, written by the people in au
        published by publisher, with ISBN isbn and costing price doll
        """
        self.title = title
        self.authors = authors[:]
        self.publisher = publisher
        self.ISBN = isbn
        self.price = price


    def num_authors(self):
        """(Book) -> int

        Return the number of authors of this book.
        """
        return len(self.authors)
```

```
>>> import book
>>> python_book = book.Book('Practical Programming',
['Campbell','Gries','Montojo'],'Pragmatic Bookshelf'
,'978-1-93778-545-1',25.0)
```



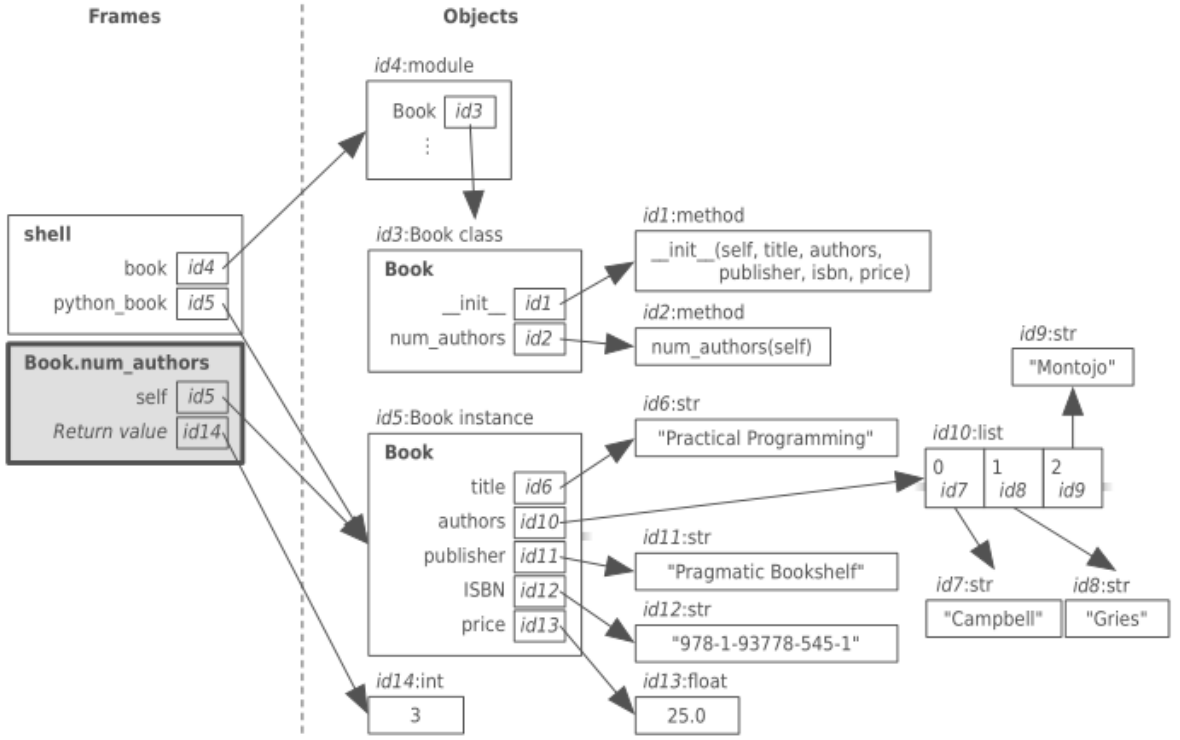
__init__ method - refined

```
class Book:
    """Information about a book"""

    def __init__(self, title="", authors=[], publisher="", isbn="0", price=10.0):
        """(Book, str, list of str, str, str, number) -> NoneType
        Create a new book entitled title, written by the people in authors,
        published by publisher, with ISBN isbn and costing price dollars.
        """
        self.title = title
        self.authors = authors[:]
        self.publisher = publisher
        self.ISBN = isbn
        self.price = price


    def num_authors(self):
        """(Book) -> int

        Return the number of authors of this book.
        """
        return len(self.authors)
```

```
>>> import imp
>>> imp.reload(book)
<module 'book' from 'C:\\Users\\jiyoung\\AppData\\Lo
cal\\Programs\\Python\\Python35\\Scripts\\class\\boo
k.py'>
>>> ruby_book = book.Book()
>>> ruby_book.title
''
>>> python_book = book.Book('Practical Programming',
['Campbell','Gries','Montojo'],'Pragmatic Bookshelf'
,'978-1-93778-545-1',25.0)
>>> python_book.title
'Practical Programming'
```

Calling a method

```
>>> import book
>>> python_book = book.Book('Practical Programming',
['Campbell','Gries','Montojo'],'Pragmatic Bookshelf'
,'978-1-93778-545-1',25.0)
>>> python_book.title
'Practical Programming'
>>> python_book.authors
['Campbell', 'Gries', 'Montojo']
>>> python_book.price
25.0
>>> python_book.num_authors()
3
```



__str__ method

```
def __str__(self):
    """(Book) -> str
    Return a human-readable string representation of this book.
    """
    rep = " Title: {0}\n Authors: {1}\n Publisher: {2}\n ISBN: {3}\n Price: {4}".format(
        self.title, self.authors, self.publisher, self.ISBN, self.price)
    return rep
```

```
>>> print(python_book)
<book.Book object at 0x00000230AE876CC0>
>>> imp.reload(book)
<module 'book' from 'C:\\Users\\jiyoung\\AppData\\Lo
cal\\Programs\\Python\\Python35\\Scripts\\class\\boo
k.py'>
>>> python_book = book.Book('Practical Programming',
['Campbell','Gries','Montojo'],'Pragmatic Bookshelf'
,'978-1-93778-545-1',25.0)
>>> print(python_book)
 Title: Practical Programming
 Authors: ['Campbell', 'Gries', 'Montojo']
 Publisher: Pragmatic Bookshelf
 ISBN: 978-1-93778-545-1
 Price: 25.0
```

__eq__ method

```
>>> python_book_1 = book.Book(
...     'Practical Programming',
...     ['Campbell', 'Gries', 'Montojo'],
...     'Pragmatic Bookshelf',
...     '978-1-93778-545-1',
...     25.0)
>>> python_book_2 = book.Book(
...     'Practical Programming',
...     ['Campbell', 'Gries', 'Montojo'],
...     'Pragmatic Bookshelf',
...     '978-1-93778-545-1',
...     25.0)
>>> python_book_1 == python_book_2
False
>>> python_book_1 == python_book_1
True
>>> python_book_2 == python_book_2
True
```

```
def __eq__(self, other):
    """ (Book, Book) -> bool
    Return True iff this book and other have the same ISBN.
    """
    return self.ISBN == other.ISBN
```

```
>>> python_book_1 = book.Book(
...     'Practical Programming', ['Campbell', 'Gries', 'Montojo'],
...     'Pragmatic Bookshelf', '978-1-93778-545-1', 25.0)
>>> python_book_2 = book.Book(
...     'Practical Programming', ['Campbell', 'Gries', 'Montojo'],
...     'Pragmatic Bookshelf', '978-1-93778-545-1', 25.0)
>>> survival_book = book.Book(
...     "New Programmer's Survival Manual", ['Carter'],
...     'Pragmatic Bookshelf', '978-1-93435-681-4', 19.0)
>>> python_book_1 == python_book_2
True
>>> python_book_1 == survival_book
False
```

Override vs. overload

```
def __eq__(self, other):
    """ (Book, Book) -> bool
    Return True iff this book and other have the same ISBN.
    """
    return self.ISBN == other.ISBN
```

- We can override an inherited method by defining a new version in our subclass. This replaces the inherited method so that it is no longer used.
- Overloading occurs when two or more methods in one class have the same method name but different parameters.

```
class Book:
    """Information about a book"""

    def __init__(self, title="", authors=[], publisher="", isbn="0", price=10.0):
        """(Book, str, list of str, str, str, number) -> NoneType
        Create a new book entitled title, written by the people in authors,
        published by publisher, with ISBN isbn and costing price dollars.
        """
        self.title = title
        self.authors = authors[:]
        self.publisher = publisher
        self.ISBN = isbn
        self.price = price
```

Lookup rules for a method call

- Look in the current object's class. If we find a method with the right name, use it.
- If we didn't find it, look in the superclass. Continue up the class hierarchy until the method is found.

Classes and objects

- Classes and objects are two of programming's power tools!
- They let good programmers do a lot in very little time.
- But with them, bad programmers can create a real mess…

- Some concepts that will help you design reliable, reusable object-oriented software.

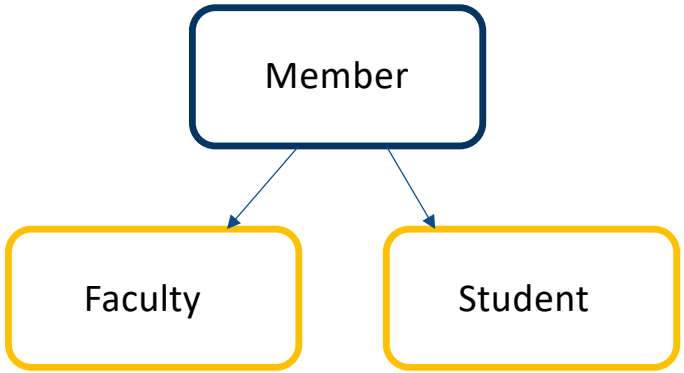
Inheritance (상속)

- Whenever you create a class, you are using inheritance: your new class automatically inherits all of the attributes of class object.
- Just like a child inherits attributes from his/her parents.
- You can also declare that your new class is a subclass of some other class.

- Managing people at a university.

Inheritance: example

```
class Member:
    """A member of a university"""

    def __init__(self, name, address, email):
        """(Member, str, str, str) -> NoneType
        Create a new member named name,
        with home address and email address
        """
        self.name = name
        self.address = address
        self.email = email
```




```
class Faculty(Member):
    """A faculty member at a university"""


    def __init__(self, name, address, email, faculty_num):
        """(Faculty, str, str, str, str) -> NoneType
        Create a new faculty named name,
        with home address and email address,
        faculty number faculty_num and an empty list of courses.
        """
        super().__init__(name, address, email)
        self.faculty_number = faculty_num
        self.courses_teaching = []
```

```
class Student(Member):
    """A Student member at a university"""


    def __init__(self, name, address, email, student_num):
        """(Faculty, str, str, str, str) -> NoneType
        Create a new student named name,
        with home address and email address,
        student number student_num and an empty list of courses taken,
        and an empty list of current courses.
        """
        super().__init__(name, address, email)
        self.student_number = student_num
        self.courses_taken = []
        self.courses_taking = []
```

Faculty and Student

```
>>> snape = Faculty('Severus Snape','Seoul','snape@k
hu.ac.kr','1234')
>>> snape.name
'Severus Snape'
>>> snape.email
'snape@khu.ac.kr'
>>> snape.faculty_number
'1234'

>>> harry = Student('Harry Potter','London','hpotter
@khu.ac.kr','4321')
>>> harry.name
'Harry Potter'
>>> harry.email
'hpotter@khu.ac.kr'
>>> harry.student_number
'4321'
```

Add features to the superclass

```
class Member:
    """A member of a university"""

    def __init__(self, name, address, email):
        """(Member, str, str, str) -> NoneType
        Create a new member named name,
        with home address and email address
        """
        self.name = name
        self.address = address
        self.email = email

    def __str__(self):
        """(Member) -> str
        Return a string representation of this Member
        """
        rep = "{}\n{}\n{}".format(self.name, self.address, self.email)
        return rep
```

```
>>> snape = Faculty('Severus Snape','Seoul','snape@k
hu.ac.kr','1234')
>>> harry = Student('Harry Potter','London','hpotter
@khu.ac.kr','4321')
>>> print(snape)
Severus Snape
Seoul
snape@khu.ac.kr
>>> print(harry)
Harry Potter
London
hpotter@khu.ac.kr
>>> str(harry)
'Harry Potter\nLondon\nhpotter@khu.ac.kr'
```

Add features to the subclass

```
>>> snape = Faculty('Severus Snape',
'Seoul','snape@khu.ac.kr','1234')
>>> print(snape)
Severus Snape
Seoul
snape@khu.ac.kr
1234
Courses:[]
```

```
class Faculty(Member):
    """A faculty member at a university"""


    def __init__(self, name, address, email, faculty_num):
        """(Faculty, str, str, str, str) -> NoneType
        Create a new faculty named name,
        with home address and email address,
        faculty number faculty_num and an empty list of courses.
        """
        super().__init__(name, address, email)
        self.faculty_number = faculty_num
        self.courses_teaching = []

    def __str__(self):
        """(Faculty) -> str
        Return a string representation of this Faculty
        """
        member_string = super().__str__()
        rep = "{}\n{}\nCourses:{}".format(
            member_string, self.faculty_number, self.courses_teaching)
        return rep
```