Practical Programming
in Python

Inspired by 'Practical Programming' by Paul Gries, Jennifer Campbell, Jason Montojo

Lecture 14
Object Oriented Programming

User Defined Types, Encapsulation, Polymorphism, Inheritance

Kurt Rinnert, Kate Shaw

Physics Without Frontiers

ICTP The Abdus Salam International Centre for Theoretical Physics

Abstract

> *"You wanted a banana but what you got was a gorilla holding the banana and the entire jungle."*
>
> *– Joe Armstrong*

We have worked with objects from the very beginning of this course. Using objects is not the same as Object Oriented programming. Object Oriented Programming is a programming *paradigm* that is quite fashionable. Practical programmers need to know at least a little bit about it. Good programmers do not blindly follow paradigms. For Python programmers, it is most important to understand *user defined types*.

Overview

- User defined types
- The `Book` type
- Encapsulation
- Polymorphism
- Inheritance
- A case study

We will focus on user defined types. OO design is beyond the scope of this course.

defining your Own Types

- Let’s say we need to keep track of books
- Every record of a book contains things like author, title, publisher price and ISBN

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

```
survival_book = Book(
    "New Programmer's Survival Manual",
    ['Carter'],
    'Pragmatic Bookshelf',
    '978-1-93435-681-4',
    19.0)
```

```
print('{0}\nwas written by {1} authors.'.format(
    python_book.title,
    python_book.num_authors()))
```

```
print('{0}\nwas written by {1} authors.'.format(
    survival_book.title,
    survival_book.num_authors()))
```

- There is a problem: this code does not run

What does this code do?

Defining Our Own Types

- You might have guessed that we created two objects of type `Book`
- You probably also guess the output:

```
Practical Programming
was written by 3 authors.
```

```
New Programmer's Survival Manual
was written by 3 authors.
```

Python doesn’t know about the `Book` type yet.

Understanding the Problem Domain

- The code expresses what we *want* to do with books
- The idea of the `Book` type comes from the *problem domain*
- The problem domain determines the features we need
- We *decided* what information we need to keep track of
- Often we need multiple related types to reflect the problem domain

OO Design Steps

1. *Understand the problem domain*: you need to know what your users want
2. *Figure out what types you need*: start with the nouns in the problem domain
3. *Write the classes for the types*: you need to tell Python what your types are and what they can do
4. *Test your code*

It is important to think things through *before* you write code.

Another Way to Check for Types

- We have used the function `type` before
- Python also provides the function `isinstance`:

```
>>> isinstance('abc', str)
True
>>> isinstance(51.3, str)
False
```

- Python also has a class `object`
- Every other class is *based on* `object`:

```
>>> isinstance('abc', object)
True
>>> isinstance(51.3, object)
True
```

We say every class is *derived* from class `object`.

The Book Class

- This is the simplest class we can write:

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

- Much like `str`, `Book` is a type:

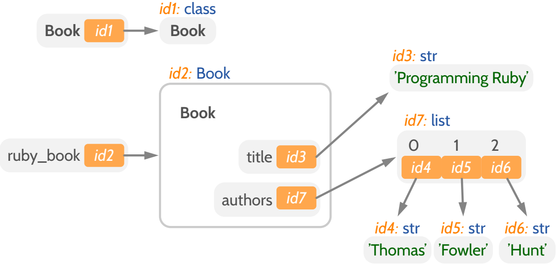
```
>>> type(str)
<class 'str'>
>>> type(Book)
<class 'Book'>
```

The keyword `class` tells Python we are defining a type.

The Book Class

- Let's use our new type:

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



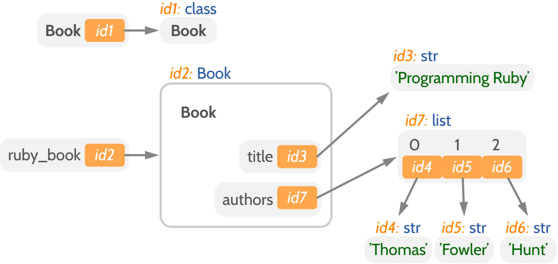
- The first assignment creates a `Book` object
- The second assignment creates `title` variable *inside* the `Book` object
- The third assignment creates another variable *inside* the `Book` object

The variables inside the Book object are called *instance variables*.

The Book Class

- We can access the instance variables like this:

```
>>> ruby_book.title
'Programming Ruby'
>>> ruby_book.authors
['Thomas', 'Fowler', 'Hunt']
```



This is similar to what we learned about modules.

Adding a Method to the Book Class

- We have used methods before:

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

- We would like to be able to do something like this:

```
>>> Book.num_authors(ruby_book)
3
>>> ruby_book.num_authors()
3
```

Methods define the *behaviour* of classes.

Adding a Method to the Book Class

- We can add a method to the `Book` class as follows:

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

    def num_authors(self):
        """Return the number of authors"""
        return len(self.authors)
```

- Assuming we have a module `book.py`, we can call the method like this:

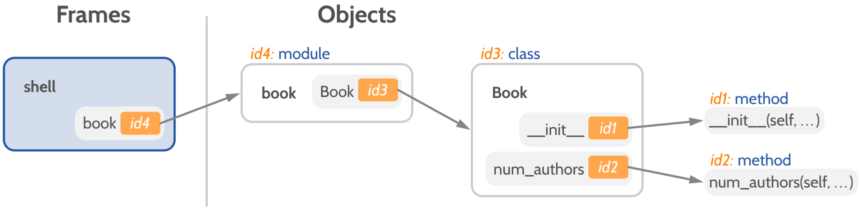
```
import book
>>> ruby_book = book.Book()
>>> ruby_book.title = 'Programming Ruby'
>>> ruby_book.authors = ['Thomas', 'Fowler', 'Hunt']
>>> book.Book.num_authors(ruby_book)
3
>>> ruby_book.num_authors()
3
```

All methods expect a class instance as first argument. The name `self` is a convention.

The book Module

- Let's assume the class is defined in `book.y`
- When we import `book` the definition gets executed

```
>>> import book
```



The class definition is just a complicated statement.

Object Initialization: Constructors

- Previously we have added variables to an object instance
- This is *not* what we normally do
- Instead we add the variables in the *constructor* method
- Constructor methods have a special name: `__init__`

```python
"""Information about a book."""

def __init__(self, title, authors, publisher, isbn, price):
    self.title = title
    self.authors = authors
    self.publisher = publisher
    self.isbn = isbn
    self.price = price
```

You can do anything in a constructor that you can do in other methods.

Object Creation

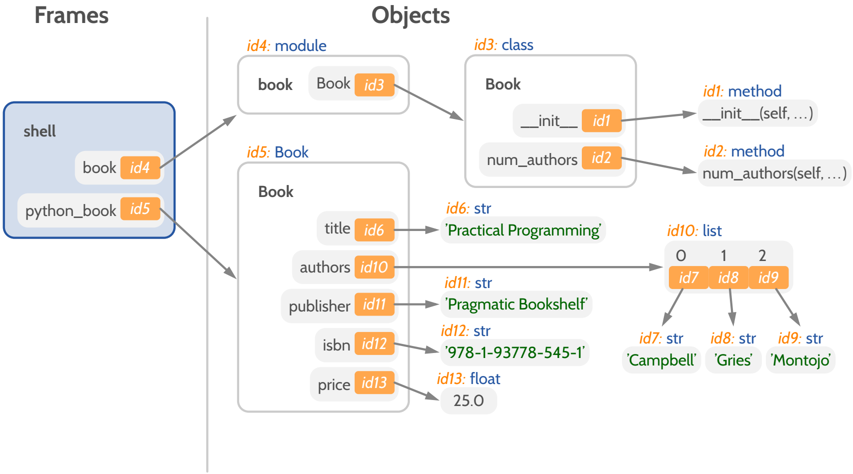
- Python executes the constructor when an object is created:

```
>>> python_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.publisher
'Pragmatic Bookshelf'
>>> python_book.ISBN
'978-1-93778-545-1'
>>> python_book.price
25.0
```

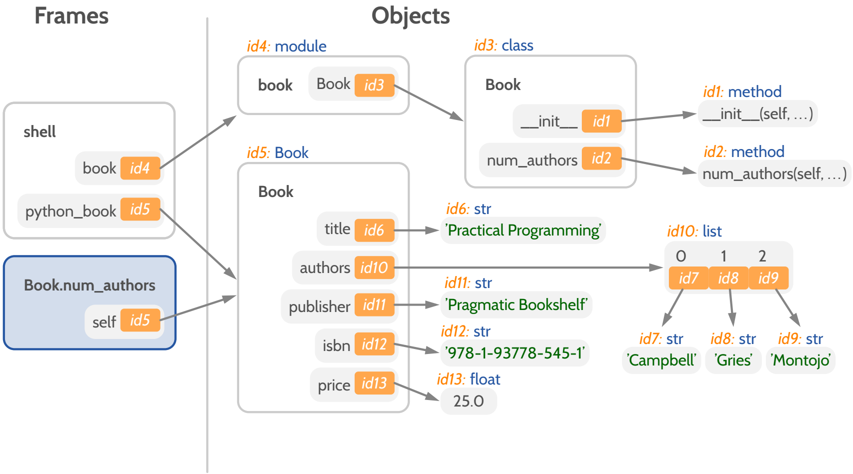
You almost always want to define `__init__` for your classes.

Memory Model After Creating Book Object



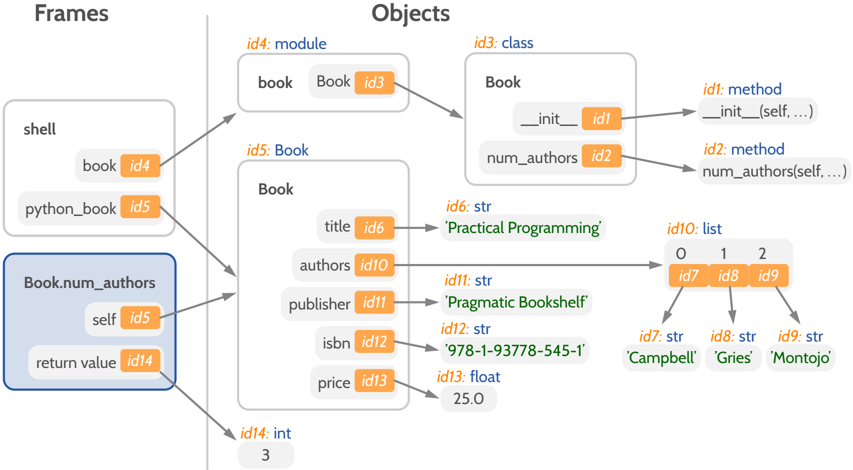
Tracing a Method Call

```
>>> python-book.num_authors()
```



Tracing a Method Call

```
3
```



More Methods Special to Python

- When we print an object the `__str__` is called
- We can write this method for book:

```
def __str__(self):
    return '"{}" ISBN: {}'.format(self.title, self.isbn)
```

- Is common to ask whether to object have equal values
- The operator **==** calls the `__eq__` method.

```
def __eq__(self, other):
    return self.isbn == other.isbn
```

This is an example of *polymorphism*.

Inheritance

- Sometimes classes share a lot of functionality
- Then it is useful to factor out the common things and use *inheritance*

```python
class InstituteMember:

    def __init__(self, name, id):
        self.name = name
        self.id = id
```

```python
class Student(InstituteMember):
    """Student information."""
```

```python
class Professor(InstituteMember):
    """Professor information."""
```

Use with great care.

Exercises Lecture 14