Physics Without Frontiers

The Abdus Salam
International Centre
for Theoretical Physics

Practical Programming
in Python

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

Lecture 14: Summary & Exercises

Object Oriented Programming

User Defined Types, Encapsulation, Polymorphism, Inheritance

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

Copyright © 2018 Kurt Rinnert, Kate Shaw
Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved. This file is offered as-is, without any warranty.

Lecture 14: Summary

In this lecture you learned the following:

- In object-oriented languages, new types are defined by creating classes. Classes support encapsulation; in other words, they combine data and the operations on it so that other parts of the program can ignore implementation details.
- Classes also support polymorphism. If two classes have methods that work the same way, instances of those classes can replace one another without the rest of the program being affected. This enables "plug-and-play" programming, in which one piece of code can perform different operations depending on the objects it is operating on.
- Finally, new classes can be defined by inheriting features from existing ones. The new class can override the features of its parent and/or add entirely new features.
- When a method is defined in a class, its first argument must be a variable that represents the object the method is being called on. By convention, this argument is called `self`.
- Some methods have special predefined meanings in Python; to signal this, their names begin and end with two underscores. Some of these methods are called when constructing objects (`__init__`) or converting them to strings (`__str__` and `__repr__`); others, like `__add__` and `__sub__`, are used to imitate arithmetic.

Lecture 14: Exercises

When writing code, only use Python concepts that have been introduced in the lectures already.

Exercise 1:

In this exercise, you will implement class Country, which represents a country with a name, a population, and an area.

a. Here is a sample interaction from the Python shell:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada.name
'Canada'
>>> canada.population
34482779
>>> canada.area
9984670
```

The code above cannot be executed yet because class `Country` does not exist. Define Country with a constructor (method `__init__`) that has four parameters: a country, its name, its population, and its area.

b. Consider this code:

```
>>> canada = Country('Canada',
    34482779,
    9984670)
>>> usa = Country('United States of America',
    313914040,
    9826675)
>>> canada.is_larger(usa)
True
```

In class `Country`, define a method named `is_larger` that takes two `Country` objects and returns **`True`** if and only if the first has a larger area than the second.

c. Consider this code:

```
>>> canada.population_density()
3.4535722262227995
```

In class `Country`, define a method named `population_density` that returns the population density of the country (people per square km).

d. Consider this code:

```
>>> usa = Country('United States of America',
313914040,
9826675)
>>> print(usa)
United States of America population 313914040
and is 9826675 aquare km.
```

In class `Country`, define a method named `__str__` that returns a string representation of a country in the format above.

e. After you have written `__str__`, this session shows that a `__repr__` method would be useful:

```
>>> canada = Country('Canada',
34482779,
9984670)
>>> canada
<exercise_country.Country object at 0x7f2aba30b550>
>>> print(canada)
Canada has population 34482779
and is 9984670 square km.
>>> [canada]
[<exercise_country.Country object at 0x7f2aba30b550>]
>>> print([canada])
[<exercise_country.Country object at 0x7f2aba30b550>]
```

Define the `__repr__` method in `Country` to produce a string that behaves like this:

```
>>> canada = Country('Canada', 34482779, 9984670)
>>> canada
Country('Canada', 34482779, 9984670)
>>> [canada]
[Country('Canada', 34482779, 9984670)]
```

Exercise 2:

n this exercise, you will implement a `Continent` class, which represents a continent with a name and a list of countries. Class `Continent` will use class `Country` from the previous exercise. If `Country` is defined in another module, you'll need to import it.

a. Here is a sample interaction from the Python shell:

```
>>> canada = country.Country('Canada', 34482779, 9984670)
>>> usa = country.Country('United States of America', 313914040,
...                       9826675)
>>> mexico = country.Country('Mexico', 112336538, 1943950)
>>> countries = [canada, usa, mexico]
>>> north_america = Continent('North America', countries)
>>> north_america.name
'North America'
>>> for country in north_america.countries:
...     print(country)
...
```

The code above cannot be executed yet, because class `Continent` does not exist. Define `Continent` with a constructor (method `__init__`) that has three parameters: a continent, its name, and its list of `Country` objects.

b. Consider this code:

```
>>> north_america.total_population()
460733357
```

In class Continent, define a method named `total_population` that returns the sum of the populations of the countries on this continent.

c. Consider this code:

```
>>> print(north_america)
North America
Canada has population 34482779
and is 9984670 square km.
United States of America has population 313914040
and is 9826675 square km.
Mexico has population 112336538
and is 1943950 square km.
```

In class `Continent`, define a method named `__str__` that returns a string representation of the continent in the format shown above.

Exercise 3:

Write a class called `Nematode` to keep track of information about C. elegans, including a variable for the body length (in millimeters; they are about 1 mm in length), gender (either hermaphrodite or male), and age (in days). Include methods `__init__`, `__repr__`, and `__str__`.