Reusing Code:
Modules & Object-Oriented Programming

L435/L555

Dept. of Linguistics, Indiana University

Fall 2016

Reusing code

Functions were our first step in reusing code. We'll look at:

- Modules: packaging functions into libraries
- Classes: packaging new data types
  - http://greenteapress.com/thinkpython2/html/thinkpython2016.html

Modules

We've already seen modules, such as `math`, `fileinput`, `sys`, `random`, & `nltk`

Different ways to import:

1. `import math` ... and then use, e.g., `math.log(...)`
2. `from math import log` ... and then use `log(...)`
3. `from math import *` to import all functions

Making a module

Put a variety of function definitions into a program, e.g., contents of `nov10.py`:

```
def hello(name, greeting='howdy'):
    return str(greeting) + ', ' + str(name) + '!'
```

In a separate program (`example.py`):

```
import nov10
s = nov10.hello('benny','hejsan')
print(s)
```

Main code

It often helps to have everything within your code in functions, e.g.,

```
def hello(name, greeting='howdy'):
    return str(greeting) + ', ' + str(name) + '!'

def main():
    print(hello('bjorn','hej'))

main()
```

Running as main vs. module

But what if I want to sometimes run the main code and sometimes just use the available functions?

```
def hello(name, greeting='howdy'):
    return str(greeting) + ', ' + str(name) + '!'

def main():
    print(hello('bjorn','hej'))

if __name__ == "__main__":
    main()
```

Many more modules ...

Some modules I've used:

- `argparse`: flexible command-line options
- `os`: interaction with the operating system
- `codecs`: working with different character encodings
- `csv`: working with .csv files
- `bsddb`: one of the many database libraries
- `libxml2`/`libxslt`: working with XML files (external package)

For more modules, see:

- https://docs.python.org/3/library/
- https://docs.python.org/3/py-modindex.html

Programming paradigms

Three main types of programming paradigms (styles, genres, etc.)

imperative	Programs simply proceed one line at a time
functional	Programs consist mostly of functions
object-oriented	Programs are designed to mimic real-world objects

Objects and Classes

Definition

Classes are types and objects are tokens.

Example

- All cars have wheels and are self propelled.
  (class - describes car in general)
- Today I drove my car to work.
  (object - particular instance)

Methods and Attributes

Definition

method (1) Something you can do with or to an object.
(2) Function which is bound to a particular class.

attribute Property of a class.

Example

- An attribute of a car is its color, or its engine type

```
mycar = Car() # create a new car object
mycar.color = 'silver'
mycar.engine = '4-cylinder'
```

- A method of a car is to drive, or to open a door

```
mycar.drive(to='Florida')
mycar.open(door='front-driver-side')
```

A Lexicon Class

attributes

- lexicon (stores the data)
- headers (stores the headers)

methods

- readFile()
- lookup()
- sorted()
- printWord()
- printDict()

Write Your Own Class

```
class MyContact:
    def getPhone(self):
        self.phone = input('What is the phone number
                           of ' + self.first + ' '+ self.last + '? ')
    def enterNew(self, first_name, last_name):
        self.first = first_name
        self.last = last_name
        self.getPhone()
    def printInfo(self):
        print(self.first, self.last, ':', self.phone)

phonebook = MyContact()
phonebook.enterNew('Markus', 'Dickinson')
phonebook.printInfo()
```

Do's and Don't's

No

Do not refer to to class attributes from outside.

```
phonebook.name = 'Markus'
```

Yes

Write methods to initialize/access/change them!

```
phonebook.enterNew('Sandra', 'Kuebler')
```

Define a Superclass

```
class Person:
    def enterNew(self, first_name, last_name):
        self.first = first_name
        self.last = last_name
    def getAge(self):
        self.age = input('What is the age? ')
```

Use a Superclass

```
class Phone(Person):
    def getPhone(self):
        self.phone = input('What is the phone number of
    def enterNew(self, first_name, last_name):
        self.first = first_name
        self.last = last_name
        self.getPhone()
    def printInfo(self):
        print(self.first, self.last, ':', self.phone, se

phonebook = Phone()
phonebook.enterNew('Markus', 'Dickinson')
phonebook.getAge()
phonebook.printInfo()
```

Definition

- Attributes and methods of a particular object should not affect other objects. This is akin to the no-no of using global variables.
- All the inner workings of a class are said to be encapsulated.
- The rest of the world need only know how to use the various methods of the class, but not how they are implemented.
- This is similar to an API (Application Programming Interface).

Definition

- In the real world, there are classes and sub-classes (and sub-sub-classes etc.).
- For example, humans belong to the class of primates, which belongs to the class of mammals, which belongs to the class of vertebrates, which belong to the class of animals.
- Humans inherit attributes from these superclasses, such as the fact that humans have spines (inherited from vertebrates).

Theory

Polymorphism

Definition

- Flexibly defined classes can work with many different types of variables.
- This is refered to as polymorphism.
- For example, the + operator can add 2 integers, but can also concatenate strings.
- Polymorphism can be very powerful and handy, but it can also be tricky to implement.
- We won't worry about it too much.