Reusing Code: Modules & Object-Oriented Programming

L555
Dept. of Linguistics, Indiana University
Fall 2012

Reusing code

Functions were our first step in reusing code. We’ll look at:
- Modules: packaging functions into libraries
- Classes: packaging new data types

I’ll also provide some slides (but we won’t cover) on:
- Classes: packaging new data types

Making a module

Put a variety of function definitions into a program, e.g.,

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

In a separate program (example.py):

```python
import dec01
s = dec01.hello('benny', 'hejsan')
print(s)
```

Running as main vs. module

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

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

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

main()
```

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

if __name__ == '__main__':
    main()
```
Programs consist mostly of functions
Programs are designed to mimic real-world

For more modules, see:
https://docs.python.org/3/library/
https://docs.python.org/3/py-modindex.html

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)

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?')

    def enterNew(self, first_name, last_name):
        self.first = first_name
        self.last = last_name

    def printInfo(self):
        print(self.first, self.last, self.phone)

phonebook = MyContact()
phonebook.enterNew('Markus', 'Dickinson')
phonebook.printInfo()
Do's and Don'ts

No
Do not refer to class attributes from outside.
phonebook.name = 'Markus'

Yes
Write methods to initialize/access/change them!
phonebook.enterNew('Sandra', 'Kuebler')

Use a Superclass

class Phone(Person):
    def getPhone(self):
        self.phone = input('What is the phone number? ')
    def enterNew(self, first_name, last_name):
        self.first = first_name
        self.last = last_name
        self.phone = input()
    def printInfo(self):
        print(self.first, self.last, ',', self.phone, 'phonebook = Phone()
phonebook.enterNew('Markus', 'Dickinson')
phonebook.getPhone()
phonebook.printInfo()

Theory

Inheritance

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).

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?

Theory

Encapsulation

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).

Theory

Polymorphism

Definition
- Flexibly defined classes can work with many different types of variables.
- This is referred 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.