Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2019
Week 10

Stavely, Chapter 11: Objects in Programs

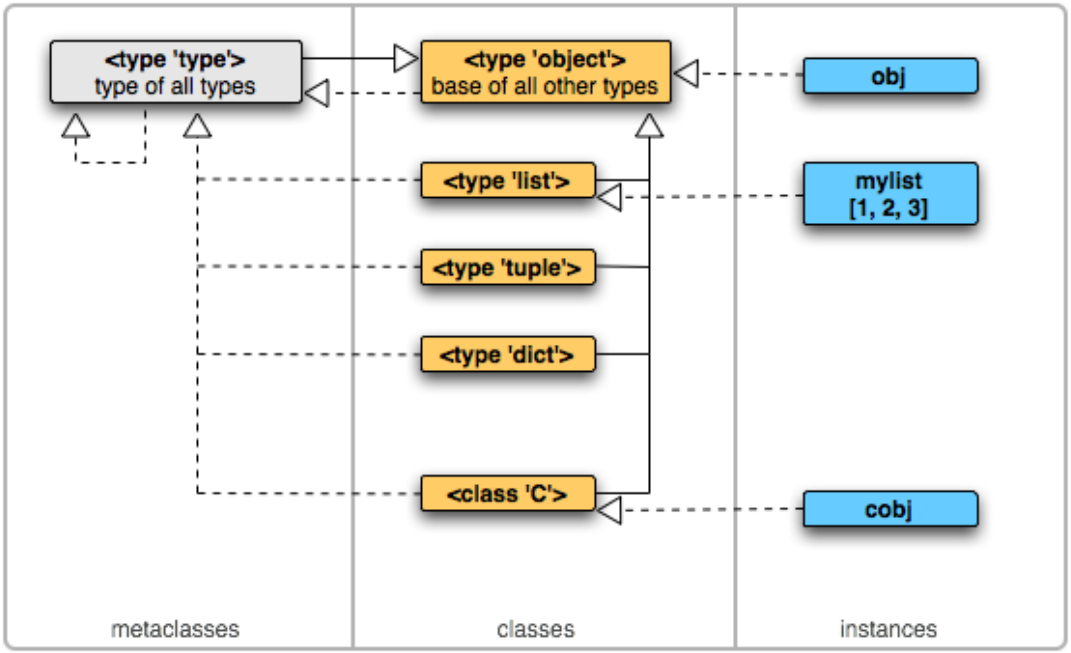
An object is a piece of data that typically has a number of attributes, identified by name, much like fields in a tuple in a database relation (Section 10.5). For example, an object representing a person might have attributes name, address, and department. Objects in a program often represents things in the real world, and then an object's attributes are properties of the thing.

All Objects Have Own ID

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

Interestingly...

```
A = set(['a','b','c'])
A #{'a', 'c', 'b'}
id(A) # 4354977128
B = A
id(B) # 4354977128
id(A) == id(B) #True
```



What are Objects in Python?

- Objects are containers for types of data
- Often an object is *mutable*: attributes for data can be *changed*
  - The collection of the values of all the data attributes of an object at any time is called, the *state* of the object
- Have specific attributes allow for interacting with the data (changing its state)
  - *Methods* are the functions for interacting with objects
  - Belong to the object: may have the same types of names as other functions in programming but all functions are specifically designed for interaction with the object

What are Objects in Python?

- Objects are of a particular type or belonging to a class of similar objects
- Object oriented programming is a way for programmers to design or adapt data-container (objects) for any possible task
- How works: Objects are links between (mappings) of types of data to variables that hold the data
- Objects allow us to easily access data using system-stored instructions

Classes

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

What are Classes?

- Almost everything in Python is an object, with own properties and methods
- A Class is like an object constructor, or a "blueprint" for creating objects.
- Inheritance: a derived class can override any methods of its base class

Basic Syntax of Classes

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

Classes

file: familyClass1.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

A Very Simple Class that Does Nothing

```
class Family(): #create class
    pass # class does nothing
#end of class Family
```

Iterations of Class; Create Variables for each

```
myPals = Family() #instance of object

myPals.first_name00 = "Alexander"
myPals.last_name00 = "Banhom-Certar"

myPals.first_name01 = "Daisy"
myPals.last_name01 = "Conham-Barter"

print("\t name: ", myPals.first_name00, myPals.last_name00)
print("\t name: ", myPals.first_name01, myPals.last_name01)
#print("\t ?: ", myPals.first_name02, myPals.last_name02) #Err
```

Variables, Class Terms and Conventions

- Fields store data in classes
- After creating an instance of a class, access its methods and data using a *dot*: class_object.field_variable
- If you use more than one word in the field name, separate words using underscores ($first\_name$)
- No capital letters in variable names for classes

`joined_lower`
– for functions, variables, attributes

`joined_lower` or `ALL_CAPS`
– for constants

`StudlyCaps`
– for classes

```
#variables
my_variable = 12
my_second_variable = 'Hello!'

#functions
my_function(my_variable)
my_print(my_second_variable)
```

Classes

file: familyClass2.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

- Let's see how variables that have been attached to classes compare with unattached variables
- Does the output change? Has the data of the class been *preserved* somehow?

Using our previously defined class

```
myPals.first_name02 = "Johnny"
myPals.last_name02 = "Appleseed"

print("\t name: ", first_name02, last_name02) # err
print("\t name: ", myPals.first_name02, myPals.last_name02)
```

Classes

file: familyClass3.py

- Can we have two instances of an with different fields attached?

Using our previously defined class

```
#create class, remove former Family class's data
class Family():
    pass # class does nothing

myPals = Family() #instance of object
myPals.first_name00 = "Alexander"
myPals.film00 = "Frozen"

myPals.first_name01 = "Daisy"
myPals.likes01 = "Snow"

print("\t ",myPals.first_name00,"and", myPals.film00)
print("\t ",myPals.first_name01,"and", myPals.likes01)
#print("\t ",myPals.first_name01,"and", myPals.film01) #Err!?
```

Features of Classes

```
In [122]: class Bill():
              def __init__(self, description):
                  self.description = description
          class Tail():
              def __init__(self, length):
                  self.length = length
          class Duck():
              def __init__(self, bill, tail):
                  self.bill = bill
                  self.tail = tail
              def about(self):
                  print('This duck has a', bill.description, 'bill and a', tail.length, 'tail')
          tail = Tail('long')
          bill = Bill('wide orange')
          duck = Duck(bill, tail)
          duck.about()

          This duck has a wide orange bill and a long tail
```

- Methods are *functions* inside of classes
- Classes are initialized (constructed) by an `init method`
  - Define the class with `__init__`
- *self* variables are used by the object, non-self variables are used to build the object (i.e., used by `__init__`)

Combining Classes (Basic Inheritance)

file:myClass.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

```
class Bill():
    def __init__(self, description):
        self.description = description

class Tail():
    def __init__(self, length):
        self.length = length
class Duck():
    def __init__(self, bill, tail):
        self.bill = bill
        self.tail = tail
    def about(self):
        print("\t Duck's description: ",bill.description)
        print("\t Tail's length: ",tail.length)

tail = Tail("long")
bill = Bill("wide orange")
duck = Duck(bill,tail)
duck.about()
print("\t Object memory address: ",duck)
```

Docstrings

file: birthday1.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

```
class User:
    def __init__(self, full_last_name, birthday):
        self.name = full_last_name
        self.birthday = birthday #yyyymmdd
        # Extract the first and last names
        name_pieces = full_last_name.split(" ") #ret a list
        self.first_name = name_pieces[0] # first element
        self.last_name = name_pieces[1] # second element
    #end of __init__()
#end of class
user = User("Frank Wright","18670608")#June 8, 1867
print("\t Name: ",user.name)
print("\t First: ",user.first_name)
print("\t Last: ",user.last_name)
print("\t Birthday: ",user.birthday)
```

Docstrings

file: docString.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

Add a *docstring* to get some information about the class

```
class User:
    """MY COOL CLASS!! This is a class to create a user object.
Used to store name and birthday."""
    def __init__(self, fullast_name, birthday):
        self.name = fullast_name
        self.birthday = birthday #yyyymmdd
        # Extract the first and last names
        name_pieces = fullast_name.split(" ") #ret a list
        self.first_name = name_pieces[0] # first element
        self.last_name = name_pieces[1] # second element
    #end of __init__()
#end of class User

help(User) #get information about class.
```

Docstrings

file: docString.py

Objects in Python
All the ID's
Classes
Class Features
Another Class
Docstrings
Using Classes

```
help(User) #get overview of the class
```

```
Help on class User in module __main__:

class User(builtins.object)
 |  This is a class to create a user object. Used to store name aband birthday.
 |
 |  Methods defined here:
 |
 |  __init__(self, full_name, birthday)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Combining Classes (Basic Inheritance)

file: birthday2.py

Defining a class with an initiation method

```
class User:
    def __init__(self, full_last_name, birthday):
        self.name = full_last_name
        self.birthday = birthday #yyyymmdd
        # Extract the first and last names
        name_pieces = full_last_name.split(" ") #ret a list
        self.first_name = name_pieces[0] # first element
        self.last_name = name_pieces[1] # second element
    #end of __init__()
#end of class
user = User("Frank Wright","18670608")#June 8, 1867
print("\t Name: ",user.name)
print("\t First: ",user.first_name)
print("\t Last: ",user.last_name)
print("\t Birthday: ",user.birthday)
```

Add Method: ageMethod1

file: birthday2.py

Add new method to class User

```
def ageMethod1(self):
      """Return the age of the person in years.
Convert birthday to get these years."""
      today = datetime.date(2019, 1, 1) #1 Jan 2019
      yyyy = int(self.birthday[0:4])
      mm = int(self.birthday[4:6])
      dd = int(self.birthday[6:8])
      dob = datetime.date(yyyy,mm,dd) #date of birth
      age_in_days = (today - dob).days
      age_in_years = age_in_days/365
      return int(age_in_years)
      #end of ageMethod1()
```

Driver Code

file: birthday2.py

Run the class with ageMethod1

```
import datetime
user = User("Frank Wright","18670608") #June 8, 1867
print("\t FullName: ",user.name)
print("\t First: ",user.first_name)
print("\t Last: ",user.last_name)
print("\t Birthday: ",user.birthday)
print("\t AgeMethod1: ",user.ageMethod1())
```

```
FullName: Frank Wright
   First:  Frank
   Last : Wright
Birthday: 18670608
     Age: 151
```

Add Method: getToday

file: birthday3.py

Method works with getToday()

```
def getToday(self):
      """returns today's date in yyyy-mm-dd format"""
      import datetime #library
      today = datetime.datetime.today()
 .strftime('%Y-%m-%d') # On one line. Is a string
      yyyy = int(today[0:4])
      mm = int(today[5:7])
      dd = int(today[8:10])
      today = datetime.date(yyyy,mm,dd) #date of birth
      return today
      #end of getToday()
```

Modify the ageMethod1() Method

file: birthday3.py

Add new method: ageMethod2()

```
def ageMethod2(self):
      """Return the age of the person in years.
Convert birthday to get these years."""
      yyyy_b = int(self.birthday[0:4])
      mm_b = int(self.birthday[4:6])
      dd_b = int(self.birthday[6:8])
      #date of birth
      dob = datetime.date(yyyy_b,mm_b,dd_b)
      today = self.getToday()
      age_in_days = (today - dob).days
      age_in_years = age_in_days/365
      return int(age_in_years)
      #end of ageMethod2()
```

Run the ageMethod1() Method

Run the class with ageMethod2

```
user = User("Frank Wright","18670608") #June 8, 1867
print("\t FullName: ",user.name)
print("\t First: ",user.first_name)
print("\t Last: ",user.last_name)
print("\t Birthday: ",user.birthday)
#print("\t AgeMethod1: ",user.ageMethod1()) # old technique
print("\t Today's date:", user.getToday())
print("\t AgeMethod2: age from today is ",user.ageMethod2())
#help(User) # see the docstrings! yey!
```

```
FullName: Frank Wright
   First:  Frank
   Last : Wright
Birthday: 18670608
     Age: 151
```

Updated Docstrings

```
help(User) #get overview of the class
```

```
Help on class User in module __main__:

class User(builtins.object)
 |  This is a class to create a user object.
 |  Used to store name aband birthday.
 |
 |  Methods defined here:
 |
 |  __init__(self, full_name, birthday)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ageMethod1(self)
 |      Return the age of the person in years. Convert birthday to get these years.
 |
 |  ageMethod2(self)
 |      Return the age of the person in years. Convert birthday to get these years.
 |
 |  getToday(self)
 |      returns today's data in yyyy-mm-dd format
 |
```

Let's Code...

- Let's build our own class that greets people while keeping track of names and ages.