Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2018
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.
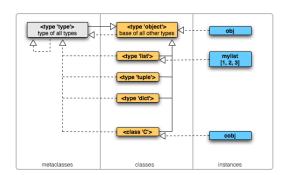
Interestingly…

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

Objects in
Python

All the ID's
Classes
Classes
Class Features
Docstrings
Add New
Method
Add Dynamic
Dates to
Method

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 is of a particular type or belongs 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

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>
```

A Very Simple Class that Does Nothing

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

Access the Class

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

myPals.f_name00 = "Alexander"
myPals.l_name00 = "Banhom-Certar"
myPals.f_name01 = "Daisy"
myPals.l_name01 = "Conham-Barter"

print(" name: ", myPals.f_name00, myPals.l_name00)
print(" name: ", myPals.f_name01, myPals.l_name01)
```

- 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
Create simple class to store data

- 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

```
f_name00 = "Johnny"
l_name00 = "Appleseed"

print(" name: ", f_name00, l_name00)

print(" name: ", myPals.f_name00, myPals.l_name00)
print(" name: ", myPals.f_name01, myPals.l_name01)
```

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

Using our previously defined class

```python
class Family(): #create class
    pass # class does nothing

myPals = Family() #instance of object
myPals.f_name00 = "Alexander"
myPals.film00 = "Frozen"

myPals.f_name01 = "Daisy"
myPals.hates_in_winter01 = "Snow"

print(myPals.f_name00,"and", myPals.film00)
print(myPals.f_name01,"and", myPals.film01) #attribErr!
print(myPals.f_name01,"and", myPals.hates_in_winter01)
```

```
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__`)
- Add a (*docstring*) for documentation

Defining a class with an initiation method

```
class User:
    def __init__(self, full_name, birthday):
        self.name = full_name
        self.birthday = birthday #yyyymmdd
        # Extract the first and last names
        name_pieces = full_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("  ",user.name)
print("  ",user.first_name)
print("  ",user.last_name)
print("  ",user.birthday)
```

Add a doc string to get some information about the class

```python
class User:
    """This is a class to create a user object.
    Used to store name aband birthday."""
    def __init__(self, full_name, birthday):
        self.name = full_name
        self.birthday = birthday #yyyymmdd
        # Extract the first and last names
        name_pieces = full_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

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

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.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

Add Another Method

Add new method to the class

```python
def age(self):
    """Return the age of the person in years.
    Convert birthday to get these years."""

    import datetime # library
    today = datetime.date(2018, 10, 29)
    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 age()
```

Add Method to get today's date

Add new method to the class

```python
def getToday(self):
    """returns today's data in yyyy-mm-dd format"""
    import datetime #library
    today = datetime.datetime.today().strftime('%Y-%m-%
    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()
```

Add new method to the class

```python
def age(self):
    """Return the age of the person in years.
    Convert birthday to get these years."""
    import datetime #library
    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 age()
```

Add new method to the class

```
user = User("Frank Wright","18670608") #June 8, 1867
print("  FullName:",user.name)
print("     First: ",user.first_name)
print("     Last :",user.last_name)
print("  Birthday:",user.birthday)
#print("       Age:",user.age()) #dynamic date getting
print("       Age:",user.age_former()) # old technique
#help(User)
```

```
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.
 |
 |  age(self)
 |      Return the age of the person in years. Convert birthday to get these years.
 |
 |  age_former(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 spend some time to build our own class that keeps track of three people's names and the type of pet companion that they have

- The name and the type of pet are to be outputted by a method for this purpose