ECE 20875
Python for Data Science
Milind Kulkarni and Chris Brinton

objects and classes
everything is an object

• Everything in Python is an **object**
  
  • Think of this as a bundle of data and operations you can perform on this data

• All objects in Python have:
  
  • An **id** (think of this as a unique “name” for an object — like its address in memory). You can access this by calling `id()`
  
  • A **type** (this defines the operations that you can perform on an object: asking for its length, adding to it, etc.). You can get this by calling `type()`

  • There is not strict type checking like in a language like Java — the operations you can perform on an object are determined by the methods it supports

• A **value** (this defines the data associated with the object: think the contents of an array, or the value of an integer)

  • Some objects are *immutable* and their values cannot change, other objects are *mutable* and their values can change
what is an object

• Intuition: an object is defined by
  • Where it is (what box of memory contains its information)
  • What it can do (what operations you can perform on it)
  • What it has (what data those operations will operate on)
  • Formally, an object is defined as an instance of a class
defining an object

• We define what an object has and what it can do by creating a **class** for that object

• Think of this as a template for an object that specifies what information and actions this object has

• We define **attributes** for an object
  • Variables: these are data we want associated with an object
  • Methods: these are the functions we want to be able to invoke on an object
  • **__init__()**: Special method automatically invoked for each new class instance

```python
class Foo:
    x = 7  # this will be accessible to all Foos
    def __init__(self, i):
        self.y = i  # this is specific to each Foo
    def bar(self):
        return self.x + self.y

a = Foo(1)  # a.x = 7, a.y = 1
b = Foo(2)  # b.x = 7, b.y = 2

print(a.bar())  # prints 8
print(b.bar())  # prints 9
```
manipulating objects

- Manipulating an object involves *invoking operations on it*
  - Including things you might not think of!
    - $x = a + b$ is invoking the `__add__()` method on object `a`
    - `len(s)` is invoking the `__len__()` method on object `s`
  - Think of this as “sending a message” to an object
    - Sending a method to an object is basically asking an object to handle an action
  - Accessing variables in objects is easy: can just use “.” notation: `foo.x`
    (Under the hood, this is also invoking methods!)
calling methods on objects

• When you call a method on an object, the object itself is always passed as the first argument of the method, called `self`

• Think of this like the `this` parameter in Java or C++ (except that it shows up explicitly in the argument list)

• Doesn’t have to be called `self` — Python will just set the first parameter to the object (but please don’t call it anything else!)

• By accessing `self.x`, we can create or access variables that are specific to this object