Object Oriented Programming in Python 3

Objects

- Objects play a central role in the Python data model
- All the types we've seen until now are in-fact objects
  Numeric types, strings, lists, tuples, dictionaries, functions, modules etc.
- Several languages support the concept of objects
- The precise definition of "object" varies with language
- At the most general level. An object can be assigned to a variable name and can be used as an argument to a function
- The type of an object determines its behavior
  For example what happens when the object is used as an operand to the + operator

Python 3

User-defined Objects

- Python lets us create our own data types
  And define their behavior
- We create a new type by defining a template for the type called a *class*
- The template can then be used to create objects of that type

Defining a class

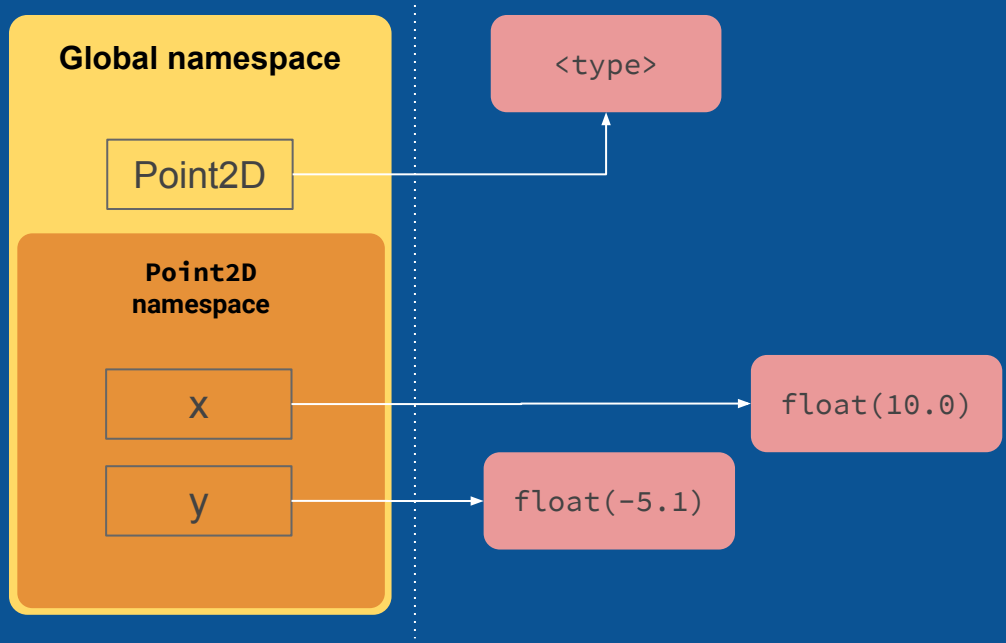
Class Syntax

```
class ClassName:
    statement-1
    statement-2
    ...
```

- `class` keyword followed by a name
- Naming convention for classes is usually `CapWords`

Class Syntax

```
>>>> class Point2D:
....      x = 10.0
....      y = -5.1
....
```



Class Syntax

Defining a class

```
>>>> class Point2D:
....      x = 10.0
....      y = -5.1
....
```

- Defining a class creates a new *namespace*
  - Namespace is a mapping between names and objects
  - The same variable names can exist in different namespaces
- Like function namespaces, an assignment inside a class definition creates a variable inside the class' local namespace

Class Syntax

```
>>>> class Point2D:
....      x = 10.0
....      y = -5.1
....

>>>> print(Point2D.x)
10.0
>>>> print(Point2D.y)
-5.1
```

- x and y are *attributes* of the class Point2D
- They can be accessed using the dot operator

Class Syntax

Instantiation

- Classes serve as templates for creating objects

```
>>>> ainst = Point2D()
>>>> type(ainst)
<class '__main__.Point2D'>
```

- An object of the class/type can be created by *calling* the class as though it were a function
- The newly *instantiated* object is returned and can be assigned to a variable name
- Above: variable `a` is called an *instance* of class/type `Point2D`

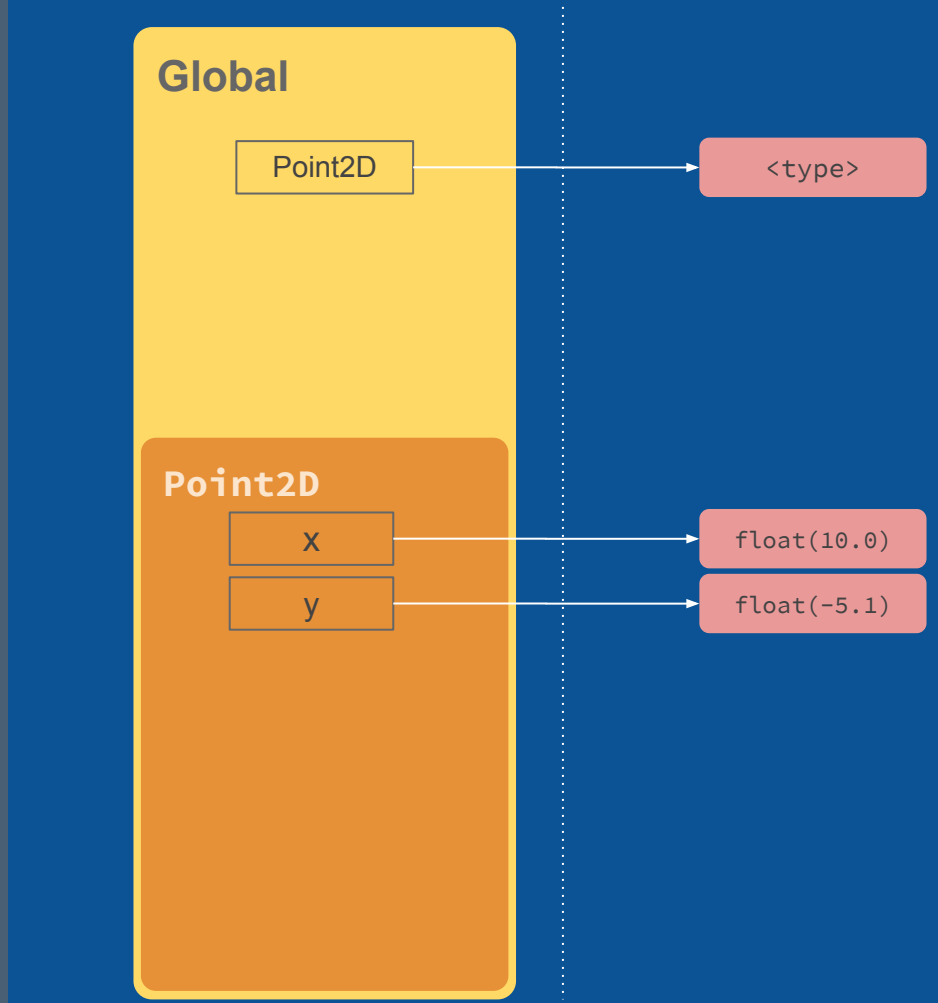
Class Syntax

- Classes serve as templates for creating objects

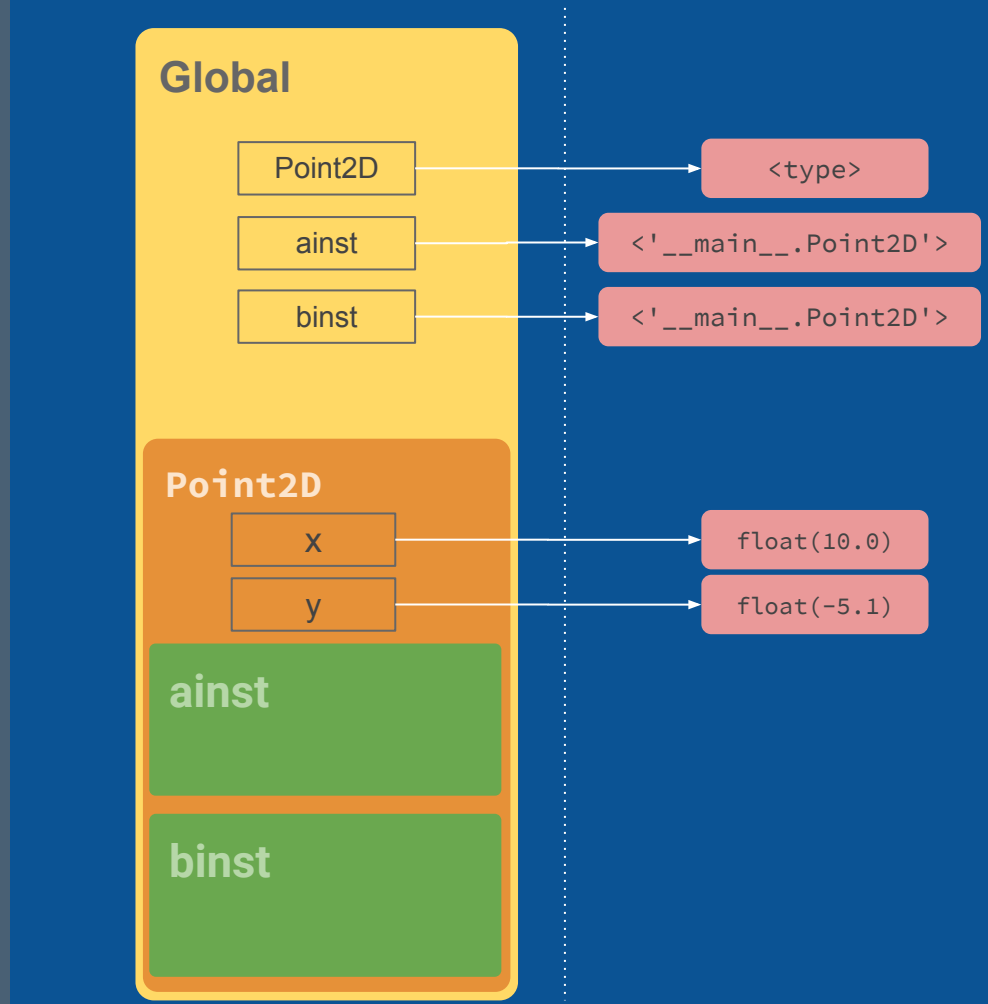
```
>>>> ainst = Point2D()
>>>> binst = Point2D()
>>>> print(ainst.x, ainst.y)
10.0 -5.1
>>>> print(binst.x, binst.y)
10.0 -5.1

>>>> print(id(Point2D), id(a), id(b))
28256992 140046856704976 140046856822984
```

- Each instance is a new object
- And a new (empty) namespace

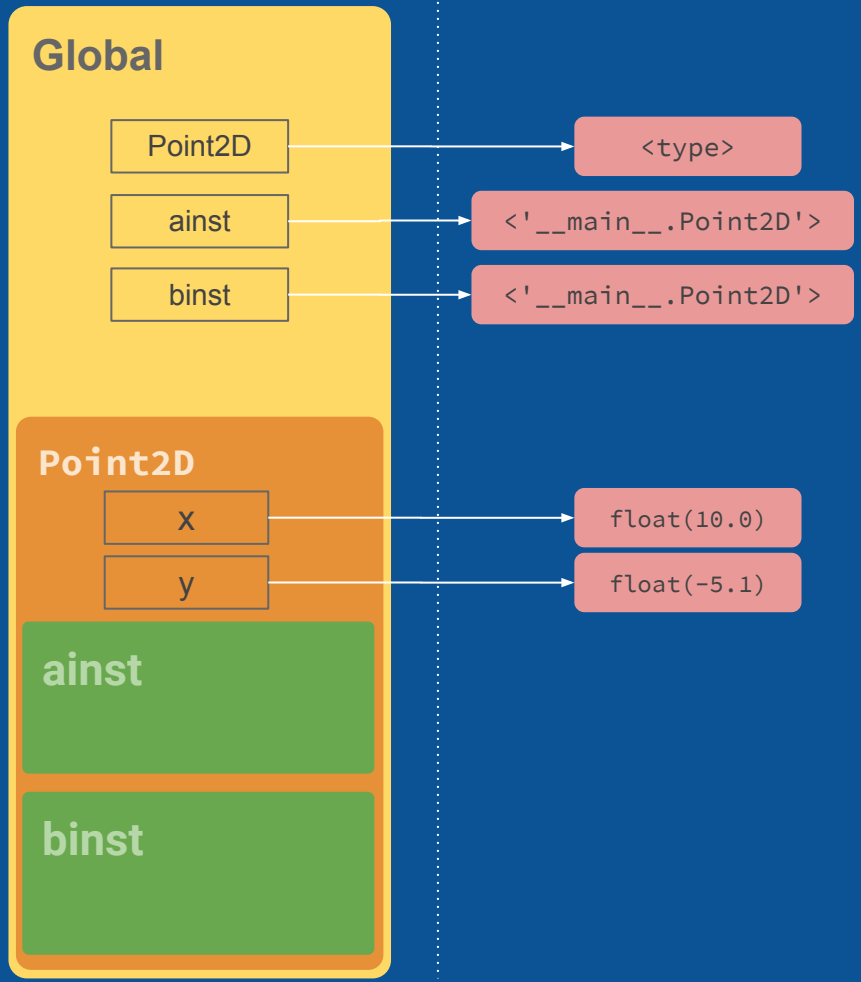
```
>>>> class Point2D:
....     x = 10.0
....     y = -5.1
....
```



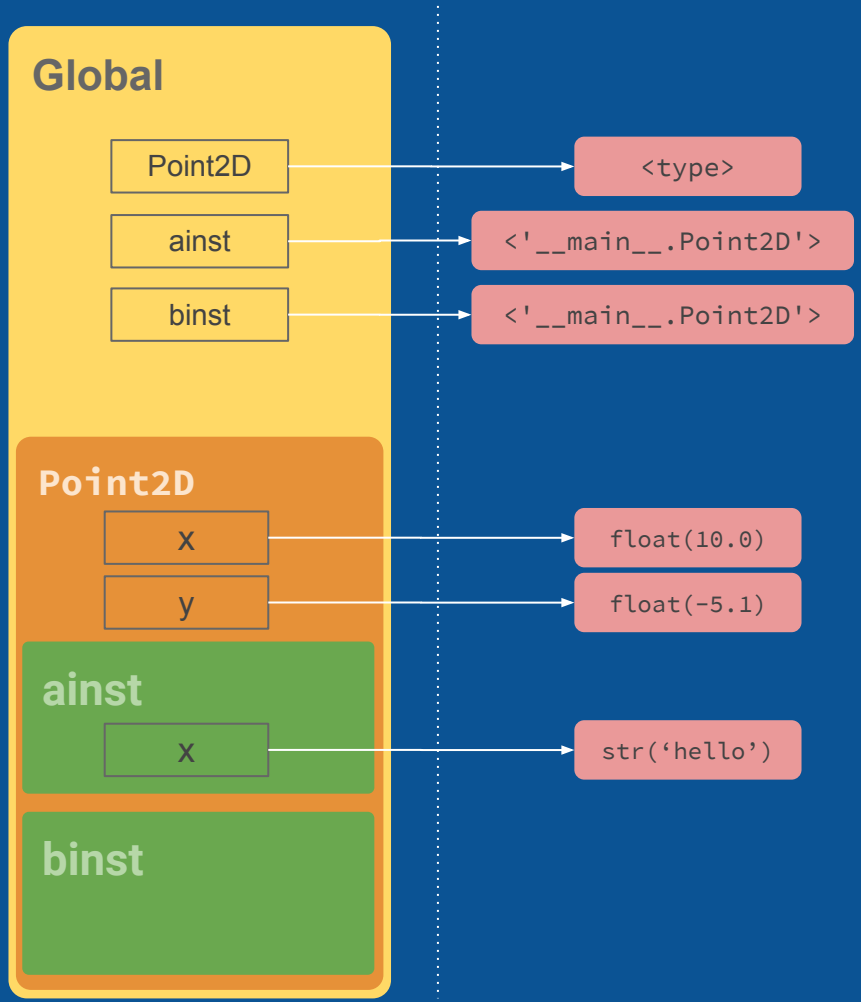
```
>>>> class Point2D:
....      x = 10.0
....      y = -5.1
....

>>>> ainst = Point2D()
>>>> binst = Point2D()
```



```
>>>> class Point2D:
....       x = 10.0
....       y = -5.1
....

>>>> ainst = Point2D()
>>>> binst = Point2D()

>>>> print(ainst.x, ainst.y)
10.0 -5.1
>>>> print(binst.x, binst.y)
10.0 -5.1
```



```
>>>> class Point2D:
....      x = 10.0
....      y = -5.1
....

>>>> ainst = Point2D()
>>>> binst = Point2D()

>>>> print(ainst.x, ainst.y)
10.0 -5.1
>>>> print(binst.x, binst.y)
10.0 -5.1

>>>> ainst.x = 'hello'
>>>> print(ainst.x, ainst.y)
hello -5.1
>>>> print(binst.x, binst.y)
10.0 -5.1
```



Methods

- In addition to data attributes
- Classes can have functions defined inside it

  These functions are called *Methods*

Classes

Defining Methods

```
class Point2D():
     x = 10.0
     y = -5.1

     def printxy(self):
         print('x coord:',self.x)
         print('y coord:',self.y)
```

Methods

```
class Point2D():
     x = 10.0
     y = -5.1

     def printxy(self):
         print('x coord:',self.x)
         print('y coord:',self.y)

ainst = Point2D()
ainst.printxy()
```

```
$ python3 point2d.py
x coord: 10.0
y coord: -5.1
```

```
class Point2D():
     x = 10.0
     y = -5.1

     def printxy(self):
         print('x coord:',self.x)
         print('y coord:',self.y)

ainst = Point2D()
ainst.printxy()
```

```
$ python3 point2d.py
x coord: 10.0
y coord: -5.1
```

- Whenever a method is called from an instance. It is given an implicit first argument
  - Which is the instance itself
- Which is why methods generally are defined with at least one parameter Conventionally called `self`
- Attributes of the instance are accessed from the method using `self.<attribute>`

Special Method

- In our example, every instance got the same initial value for `x` and `y`
- The special method `__init__()` can be used to perform initialization of an instance
- The `__init__()` method is special in that, if a method with that name is defined, it is automatically called whenever an object is instantiated

__init__()

```
class Point2D():
     x = 10.0
     y = -5.1

     def __init__(self):                # must use this exact name
         print('__init__() called ')    # so Python can call it

     def printxy(self):
         print('x coord:',self.x)
         print('y coord:',self.y)

ainst = Point2D()          # will cause ainst.__init__() to be called
binst = Point2D()          # binst.__init__() here
mynameinst = Point2D()     # again for mynameinsta.__init__()
```

```
$ python3 point2d_init.py
__init__() called
__init__() called
__init__() called
```

Special Method

- `__init__()` called automatically
- Can have parameters other than `self`
- Can create variables in the instance namespace using self and attribute syntax

  `self.<attribute> = value`

__init__()
arguments

```
class Point2D():
                                                # x and y not assigned
    def __init__(self, strcoord='0.0,0.0'):    # default argument
        self.x, self.y = map(float, strcoord.split(','))

    def printxy(self):
        print('x coord:',self.x)
        print('y coord:',self.y)

ainst = Point2D('-200,900')
binst = Point2D()

ainst.printxy()
binst.printxy()
```

```
$ python3 point2d_initparam.py
x coord: -200
y coord: 900
x coord: 0.0
y coord: 0.0
```

```
class Point2D():

      def __init__(self, strcoord='0.0,0.0'):
          self.x, self.y = map(float, strcoord.split(','))

      def printxy(self):
          print('x coord:',self.x)
          print('y coord:',self.y)

      def isleftof(self, other):
          return self.x < other.x

      def isrightof(self, other):
          return self.x > other.x

      def isabove(self, other):
          return self.y > other.y

      def isbelow(self, other):
          return self.y < other.y

ainst = Point2D('-200,900')
binst = Point2D()# 0.0,0.0

print(ainst.isleftof(binst))
print(ainst.isrightof(binst))
print(binst.isabove(ainst))
print(binst.isbelow(ainst))
```

```
$ python point2d_methods.py
True
False
False
True
```

```
git clone git@github.com:ebasheer/math5061
```

Homework