Part 1:

Object Oriented Programming

IN PYTHON

Good Text Books

- **Data Structures and Algorithms in Python**
  Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser
  July 2013, ©2013
  [Exists in Braude Library]
- **Python 3 Object Oriented Programming**
  Dusty Philips
- **Introduction to Computation and Programming Using Python, revised and expanded edition**
  John V. Guttag

Objects and Classes

- Look around in classroom and make a list of all objects you see? Try to be creative …
- Did you count objects or classes?
- Expand your view to the college area, which other objects/classes do you see?
- Further categorize your classes according to is-a relationships (base classes, super classes)

Quiz 1: An Object or a Class?

Animal

Dog

Sylvester

Cat

Scooby-doo

Parrot

Tom

Tweety

Bird

Quiz 2: An Object or a Class?

Integer

Number

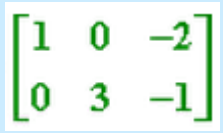
157

3.14159

Float

Vector

Matrix



(3.1, 5, 420)

What is an Object?

- A “stand alone” entity that belongs to some class
- An instance of a class
- It must be something that can be identified by a private name or identity number (or memory address!)

3.14159

157

(3.1, 5, 420)

$$\begin{bmatrix} 1 & 0 & -2 \\ 0 & 3 & -1 \end{bmatrix}$$

Software Objects (C Struct)

- In software engineering, objects are **abstract models** of real life objects

avi: Student

```
first_name = Avi
last_name = Levy
age = 25
id = 9108253
phone = 07-5432198
email = "avi@gmail.com"
address = "Dan 12, Haifa"
```

twit: Bird

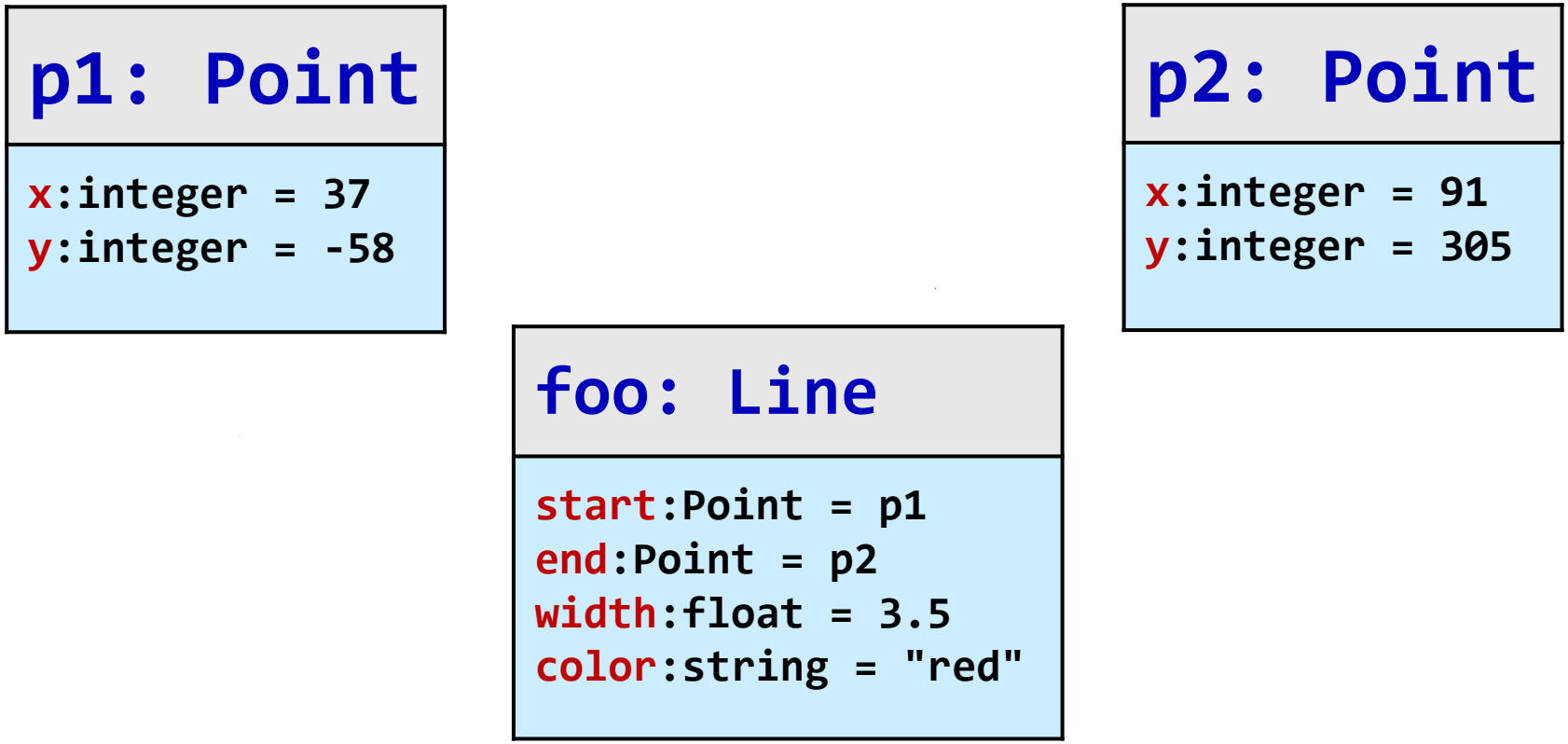
```
name = "Twitty"
color = "yellow"
weight = 12
owner = avi
friends = [sylv, spike, granny]
```

sylv: Cat

```
name = "Sylvester"
color = "black and white"
weight = 36
owner = avi
friends = [granny, spike]
```

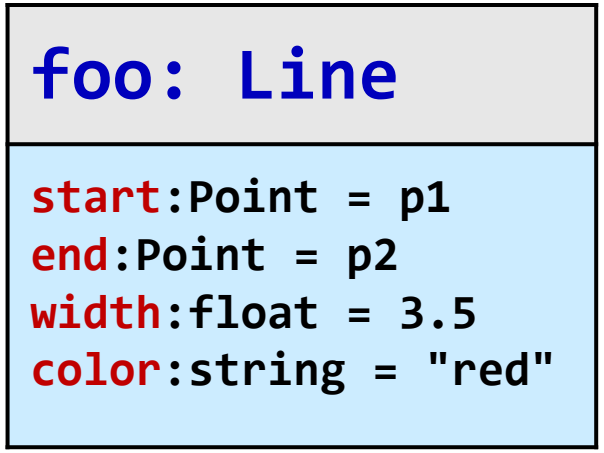
Software Objects

- Sometimes we would like to also indicate members Type (Class)
- **Class = Type !**



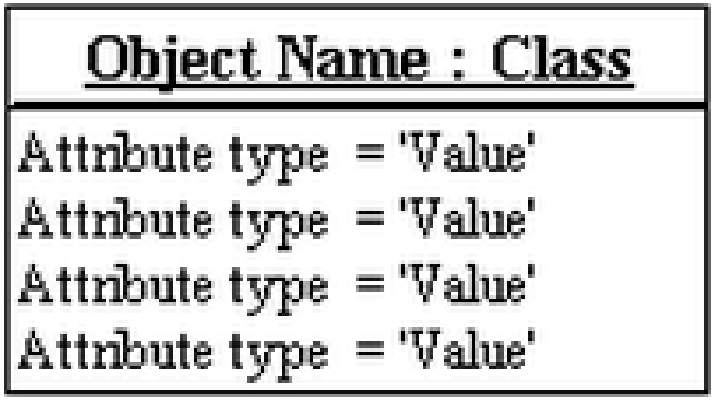
Attributes

- Object attributes are the list of properties that we decided to model for it
- Example: a Line object has 4 attributes:
  - Point [start]
  - Point [end]
  - Float [width]
  - String [color]



Object Diagram

- An object Diagram consists of the object name, object Class, and a list of attributes with their values



Object with attributes

OOP, OOD, and OOA

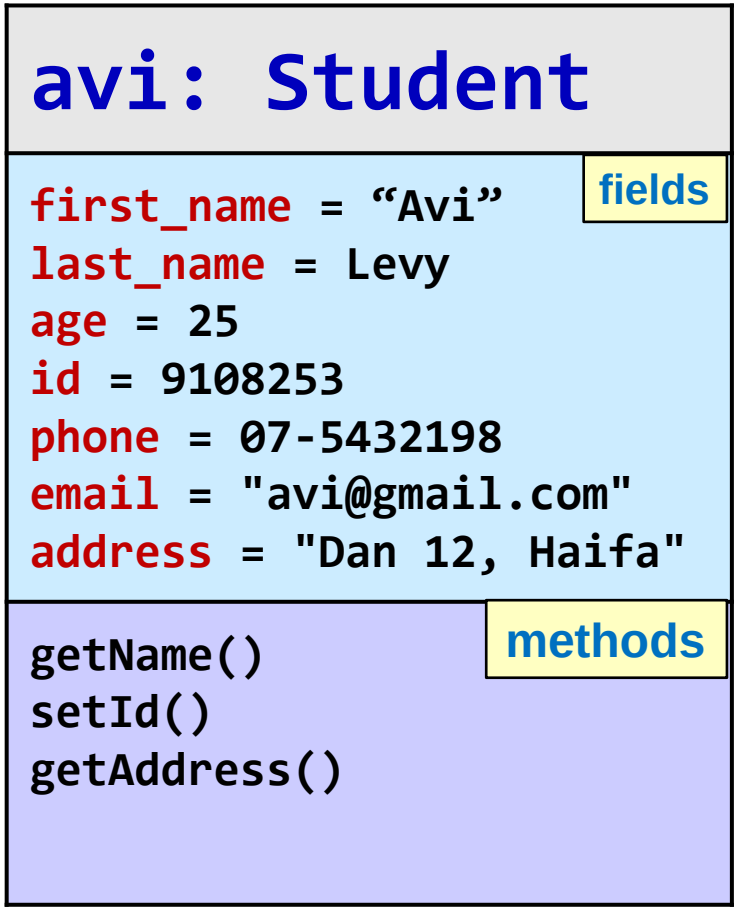
- **OOP – Object Oriented Programming**
- **OOD – Object Oriented Design**
- **OOA – Object Oriented Analysis**

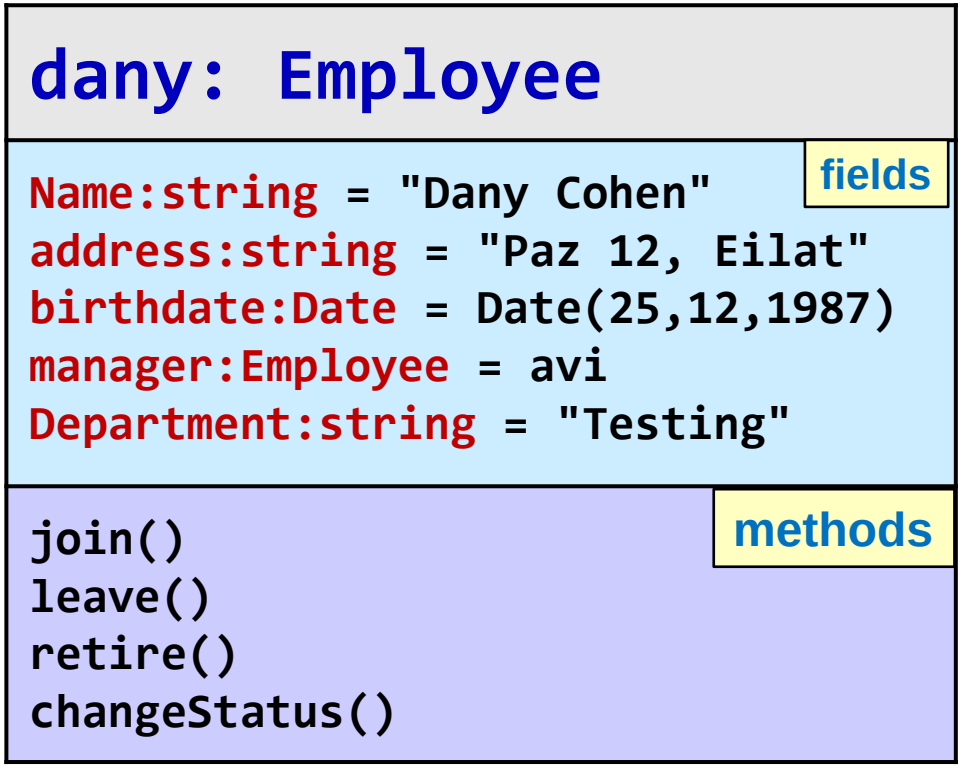
All these three topics are related but are not the same!

- **OOA**
  building and analyzing an object model of the application domain
- **OOD**
  Implementing the object model (using abstract data types, charts, and specifications. UML – Unified Modelling Language
- **OOP**
  Realizing the OOD in a specific programming language (like C++, Java, or Python)

From C Struct to Class

- In contrast to C struct, a Class contains **methods**





Object Instantiation

- The process of creating a new instance of a class is known as **instantiation**
- To instantiate an object we usually invoke the constructor of a class:

```
u = Vector()
```

  - This is assuming that the constructor does not require any parameters.
- A constructor may require parameters, such as

```
v = Vector(10, 20, 30)
```

- Many of Python's built-in classes a **literal form** for designating new instances. For example, the command

```
temperature = 98.6
```

results in the creation of a new instance of the float class.

Python Built-In Classes

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of **set** class	✓
dict	associative mapping (aka dictionary)	

- A class is *immutable* if each object of that class has a fixed value upon instantiation that cannot subsequently be changed. For example, the **float** class is immutable.

Mutable and Immutable Classes

```
>>> a = [2,4,6,8]
>>> a[0]
2
>>> a[0] = 5
>>> a
[5, 4, 6, 8]
>>> b = (2,4,6,8)
>>> b[0]
2
>>> b[0] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

The bool Class

- The **bool** class is used for logical (Boolean) values
- It has only two objects: `True, False`
- The default constructor, bool( ), returns False:

```
>>> b = bool()
>>> b
False
```

The bool Class

- Python allows the creation of a Boolean value from a nonboolean type using the syntax bool(foo) for value foo. The interpretation depends upon the type of the parameter.
  - Numbers evaluate to False if zero, and True if nonzero.
  - Sequences and other container types, such as strings and lists, evaluate to False if empty and True if nonempty.

```
>>> A = [1,2,3]
>>> a = bool(A)
>>> a
True
>>> B = []
>>> b = bool(b)
>>> b
False
```

```
>>> c = bool(635)
>>> c
True
>>> bool(0)
False
>>> bool("Hello")
True
>>> bool("")
False
```

Boolean expressions

```
s = "Hello World"
A = [1,2,3]

if A:
    print "The list A is not empty!"

if s:
    print "The string s is not empty"

B = []
if not B:
    print "B is empty list"
```

The int Class

- The **int** class is designed to represent integer values with arbitrary magnitude.
  - Python automatically chooses the internal representation for an integer based upon the magnitude of its value.
- The integer constructor, `int()`, returns 0 by default.
- This constructor can also construct an integer value based upon an existing value of another type.
  - For example, if f represents a floating-point value, the syntax `int(f)` produces the truncated value of f. For example, `int(3.14)` produces the value 3, while `int(-3.9)` produces the value `-3`.
  - The constructor can also be used to parse a string that represents an integer. For example, the expression `int(137)` produces the integer value `137`.

The int constructor

```
>>> int()
0
>>> int(15.32)
15
>>> int("4207")
4207
>>> int("105", 16)
261
>>>
>>> int()
0
>>> int(3.14)
3
>>> int("101", 16)
257
>>> int("101", 2)
5
```

The float Class

- The **float** class is the floating-point type in Python.
  - The floating-point equivalent of an integral number, 2, can be expressed directly as 2.0.
  - One other form of literal for floating-point values uses scientific notation. For example, the literal `6.022e23` represents the mathematical value `6.022×1023`.
- The constructor `float( )` returns `0.0`.
- When given a parameter, the constructor, float, returns the equivalent floating-point value.
  - `float(2)` returns the floating-point value 2.0
  - `float('3.14')` returns 3.14

The float constructor

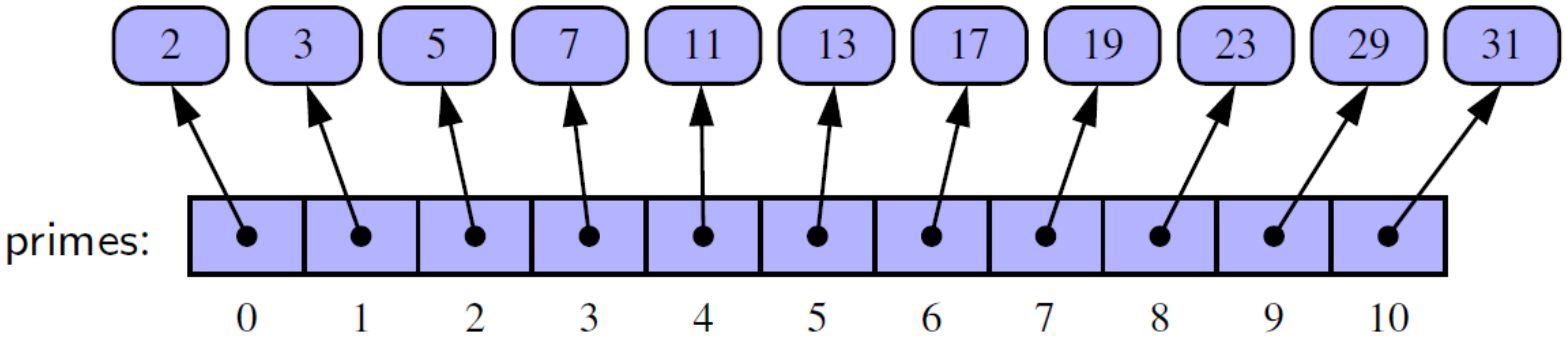
```
>>> float()
0.0
>>> float(5)
5.0
>>> float(" 7.310 ")
7.31
>>> s = "3.14 is a cool number"
>>> float(s[0:4])
3.14
```

The infinite float

```
>>> inf = float("Infinity")
>>> inf + 1000
inf
>>> inf * inf * 12 + 75
inf
>>> 1.0 / inf
0.0
>>> inf / (2**1000)
inf
```

Python list Class

- A **list** instance stores a sequence of objects, that is, a sequence of references (or pointers) to objects in the list.
- Elements of a list may be arbitrary objects (including the **None** object).
- Lists are array-based sequences and a list of length n has elements indexed from 0 to n−1 inclusive.



Python list Class

- Lists have the ability to dynamically expand and contract their capacities as needed.
- Python uses the characters `[ ]` as delimiters for a list literal.
  - `[]` is an empty list.
  - `['red', 'green', 'blue']` is a list containing three string instances.
- The `list( )` constructor produces an empty list by default.
- The list constructor will accept any iterable parameter.
  - list('hello') produces a list of individual characters, `['h', 'e', 'l', 'l', 'o']`.

```
>>> A = [2,3,5,7,11]
>>> B = ['red', 'green', 'blue']
>>> C = list('hello')
>>> C
['h', 'e', 'l', 'l', 'o']
```

Python tuple Class

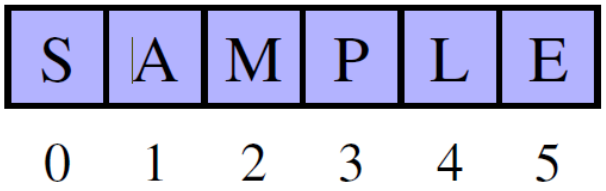
- The **tuple** class provides an immutable (unchangeable) version of a sequence
- Parentheses delimit a tuple
  - The empty tuple is `()`
- To express a tuple of length one as a literal, a comma must be placed after the element, but within the parentheses.
  - For example, `(17,)` is a one-element tuple.

The str Class

- String literals can be enclosed in single quotes, as in ‘Hello OOP’, or double quotes, as in “Hello OOP".
- A string can also begin and end with three single or double quotes, if it contains newlines in it:

```
s1 = "Hello World"

s2 = """add a new element e at the end of "children" of p
        Return the 'Position' of new node.
        Raise ValueError if Position p is invalid
        """
s3 = "SAMPLE"
```



The set Class

- Python's **set** class represents a **set**, namely a collection of unique elements:
  - without duplicates
  - (if you add element which is in the set, nothing will happen)
  - without an inherent order to those elements
- Only instances of immutable types can be added to a Python set!
- Therefore, objects such as integers, floating-point numbers, and character strings are eligible to be elements of a set.
  - The frozenset class is an immutable form of the set type, itself.

The set Class

- Python uses curly braces **{** and **}** as delimiters for a set
- For example, as **{17}** or **{'red', 'green', 'blue'}**
- The exception to this rule is that { } does not represent an empty set
- Instead, the constructor set( ) returns an empty set.

```
>>> s = {1, 2, 3, 1, 2, 1, 3}
>>> s
set([1, 2, 3])
>>> s = set("hello")
>>> s
set(['h', 'e', 'l', 'o'])
>>> s = {'red', 'green', 'blue'}
```

The dict Class

- Python's **`dict`** class represents a dictionary, or mapping, from a set of distinct keys to associated values
- Python implements a **`dict`** using an almost identical approach to that of a set, but with storage of the associated values
- The literal form **`{ }`** produces an empty dictionary
- A nonempty dictionary is expressed using a comma-separated series of **`key:value`** pairs:

```
{key1: value1, key2: value2,  … }
```

- Alternatively, the constructor accepts a sequence of key-value pairs as a parameter, as in dict(pairs) with **`pairs = [('ga', 'Irish'), ('de', 'German')]`**.

The dict Class

```
d = dict() # creates an empty dictionary
d['name'] = 'Avi Cohen'
d['age'] = 32
d['id'] = 5802231
d['address'] = 'Hayarden 43, Gedera'

# Another way to do the same thing:
d = { 'name': 'Avi Cohen', 'age': 32, 'id': 5802231,
      'address': 'Hayarden 43, Gedera' }

# Another way to do the same thing:
d = dict(name='Avi Cohen', age=32, id=5802231, address='Hayarden 43, Gedera')

# Another way to do the same thing:
pairs = [ ('name', 'Avi Cohen'), ('age', 32), ('id', 5802231),
          ('address', 'Hayarden 43, Gedera') ]
d = dict(pairs)
```