

### Recall: Classes are Types for Objects

- Values must have a type
  - An object is a **value**
  - Object type is a **class**
- Classes are how we add new types to Python

The diagram shows a yellow box labeled 'id2' containing a 'Point' class with attributes x=2.0, y=3.0, and z=5.0. A label 'class name' points to the 'Point' box. To the right, a large yellow circle labeled 'Types' contains a smaller green circle labeled 'Classes' which lists: Point, RGB, Turtle, and Window. Below the 'Types' circle are listed: int, float, bool, and str.

### Recall: Objects can have Methods

- Method:** function tied to object
  - Function call: `<function-name>(<arguments>)`
  - Method call: `<object-variable>.<function-call>`
  - Use of a method is a **method call**
- Example:** `p.distanceTo(q)`
  - Both `p` and `q` act as arguments
  - Very much like `distanceTo(p, q)`
- Methods (often) in **class folders**

The diagram shows two objects, 'p' (id3) and 'q' (id4), both of type 'Point'. Object 'p' has attributes x=5.0, y=2.0, z=3.0. Object 'q' has attributes x=7.4, y=0.0, z=0.0. Below them is a red box representing the 'Point' class folder containing methods: `__init__(x, y, z)`, `distanceTo(other)`, and `abs()`.

### Name Resolution for Objects

- `<object>.<name>` means
  - Go the folder for *object*
  - Find attribute/method *name*
  - If missing, check **class folder**
  - If not in either, raise error
- For most Python objects
  - Attributes** are in **object** folder
  - Methods** are in **class** folder
- Rules can be broken... (but not in this class)

The diagram shows objects 'p' (id3) and 'q' (id4) with their attributes. A red box below shows the 'Point' class folder with methods `__init__(x, y, z)`, `distanceTo(other)`, and `abs()`. Arrows indicate that for object 'p', attributes x, y, z are found in its folder, while for 'p.distanceTo', the method is found in the 'Point' class folder.

### The Class Definition

Goes inside a module, just like a function definition.

keyword `class`  
Beginning of a class definition

`class <class-name>(object):` Do not forget the colon!

Specification (similar to one for a function)  
"""Class specification""" more on this later

`<function definitions>`

to define **methods**

`<assignment statements>` ...but not often used

to define **variables**

`<any other statements also allowed>`

**Example**  
Python creates after reading the class definition

```
class Example(object):
    """The simplest possible class."""
    pass
```

### Instances and Attributes

- Assignments add object attributes
  - `<object>.<att> = <expression>`
  - Example:** `e.b = 42`
- Assignments can add class attributes
  - `<class>.<att> = <expression>`
  - Example:** `Example.a = 29`
- Objects can access class attributes
  - Example:** `print e.a`
  - But assigning it creates object attribute
  - Example:** `e.a = 10`
- Rule:** check object first, then class

The diagram shows an object 'e' (id2) with attribute 'b' = 42. A class 'Example' has attribute 'a' = 29. A red box shows object 'e' with attribute 'a' = 29, illustrating that the object attribute takes precedence over the class attribute.

### Invariants

- Properties of an attribute that must be true
- Works like a precondition:
  - If invariant satisfied, object works properly
  - If not satisfied, object is "corrupted"
- Examples:**
  - Point** class: all attributes must be floats
  - RGB** class: all attributes must be ints in 0..255
- Purpose of the **class specification**

### The Class Specification

```
class Worker(object):
    """An instance is a worker in an organization.
    Instance has basic worker info, but no salary information.
    """
    ATTRIBUTES:
    lname: Worker's last name. [str]
    ssn: Social security no. [int in 0..999999999]
    boss: Worker's boss. [Worker, or None if no boss]
```

**Annotations:**

- Short summary:** """An instance is a worker in an organization.
- More detail:** Instance has basic worker info, but no salary information.
- Attribute list:** ATTRIBUTES:
- Description:** lname: Worker's last name. [str]
- Invariant:** ssn: Social security no. [int in 0..999999999]
- Attribute Name:** boss: Worker's boss. [Worker, or None if no boss]

### Method Definitions

- Looks like a function def
  - But indented *inside* class
  - The first parameter is always called **self**
- In a method call:
  - Parentheses have one less argument than parameters
  - The object in front is passed to parameter self
- Example:** a.distanceTo(b)
  - self
  - q

```
class Point(object):
    """Instances are points in 3d space
    x: x coord [float]
    y: y coord [float]
    z: z coord [float]
    """
    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

### Methods Calls

**Example:** a.distanceTo(b)

a: id2 (x: 1.0, y: 2.0, z: 3.0)    b: id3 (x: 0.0, y: 3.0, z: -1.0)

```
class Point(object):
    """Instances are points in 3d space
    x: x coord [float]
    y: y coord [float]
    z: z coord [float]
    """
    def distanceTo(self,q):
        """Returns: dist from self to q
        Precondition: q a Point"""
        assert type(q) == Point
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Call: distanceTo(self=id2, q=id3) returns 1

### Special Method: \_\_init\_\_

w = Worker('Obama', 1234, None)

```
def __init__(self, n, s, b):
    """Initializer: creates a Worker
    Has last name n, SSN s, and boss b
    Precondition: n a string, s an int in range 0..999999999, and b either a Worker or None.
    self.lname = n
    self.ssn = s
    self.boss = b
    """
```

Called by the constructor

id8: Worker (lname: 'Obama', ssn: 1234, boss: None)

### Aside: The Value None

- The boss field is a problem.
  - boss refers to a Worker object
  - Some workers have no boss
  - Or maybe not assigned yet (the buck stops there)
- Solution:** use value None
  - None:** Lack of (folder) name
  - Will reassign the field later!
- Be careful with None values
  - var3.x gives error!
  - There is no name in var3
  - Which Point to use?

var1: id5 (x: 2.2, y: 5.4, z: 6.7)    var2: id6 (x: 3.5, y: -2.0, z: 0.0)    var3: None

### Making Arguments Optional

- We can assign default values to \_\_init\_\_ arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional
- Examples:**
  - p = Point() # (0,0,0)
  - p = Point(1,2,3) # (1,2,3)
  - p = Point(1,2) # (1,2,0)
  - p = Point(y=3) # (0,3,0)
  - p = Point(1,z=2) # (1,0,2)

```
class Point(object):
    """Instances are points in 3d space"""
    x = 0.0 # x coord, float
    y = 0.0 # y coord, float
    z = 0.0 # z coord, float
    def __init__(self,x=0,y=0,z=0):
        """Initializer: makes a new Point
        Precondition: x,y,z are numbers"""
        self.x = x
        self.y = y
        self.z = z
        ...
```