SITARE University

Module 21

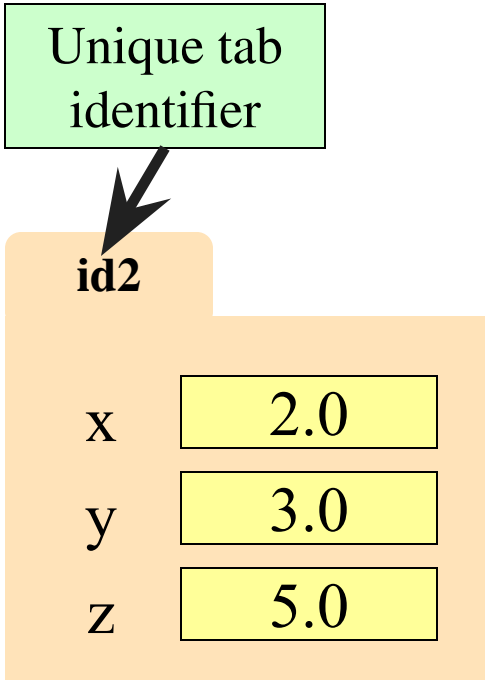
Classes

Acknowledgement

Walker M. White
Cornell University

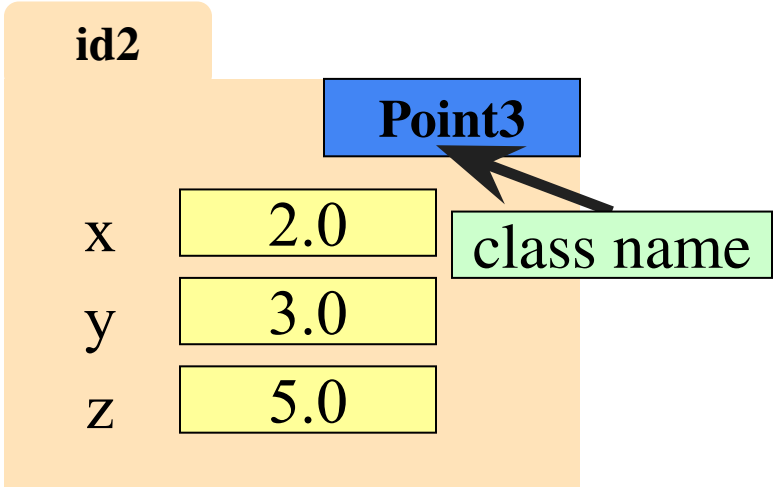
Recall: Objects as Data in Folders

- An object is like a **manila folder**
- It contains other variables
  - Variables are called **attributes**
  - Can change values of an attribute (with assignment statements)
- It has a "tab" that identifies it
  - Unique number assigned by Python
  - Fixed for lifetime of the object

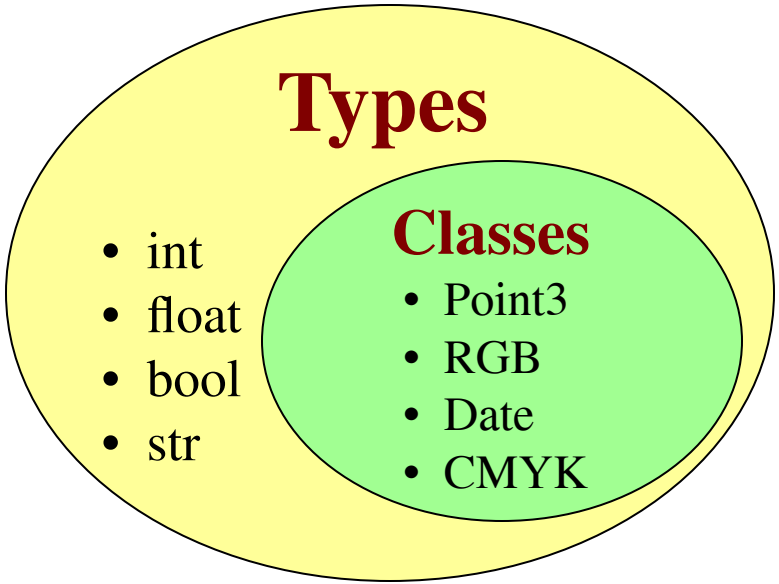


Recall: Classes are Types for Objects

- Values must have a type
  - An **object** is a **value**
  - A **class** is its **type**

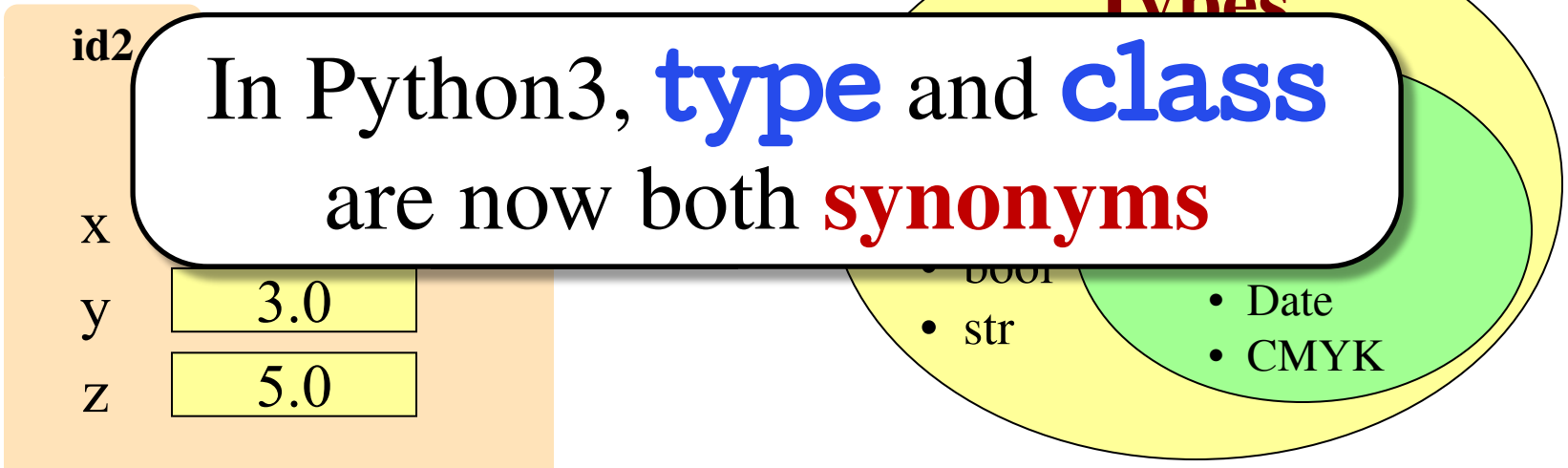


- Classes are how we add new types to Python



Recall: Classes are Types for Objects

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



In Python3, **type** and **class** are now both **synonyms**

It is Time to Define Classes

- Remember how we learned about functions
  - Learned to use (**call**) them first
  - Then we learned how to define them
- Now going to do the same for classes
  - Learned how to use (**instantiate**) them first
  - Will now learn how to define them
- First, let's look at the **syntax**
  - Will look at what it means later

The Class Definition

class *<class-name>*(object):

"""Class specification"""

<function definitions>

<assignment statements>

<any other statements also allowed>

Goes inside a module, just like a function definition.

The Class Definition

Keyword **class**
Beginning of a class definition

```
class <class-name>(object):
    """Class specification"""
    <function definitions>
    <assignment statements>
    <any other statements also allowed>
```

Goes inside a module, just like a function definition.

The Class Definition

Keyword **class**
Beginning of a class definition

class *<class-name>*(object):
"""Class specification"""
<function definitions>
<assignment statements>
<any other statements also allowed>

more on this later

Goes inside a module, just like a function definition.

The Class Definition

Keyword **class**
Beginning of a class definition

class *<class-name>*(object):

"""Class specification"""

<function definitions>

<assignment statements>

<any other statements also allowed>

Do not forget the colon!

more on this later

Goes inside a module, just like a function definition.

The Class Definition

Keyword **class**
Beginning of a class definition

class *<class-name>*(object):

"""Class specification"""

<function definitions>

<assignment statements>

<any other statements also allowed>

Specification (similar to one for a function)

Do not forget the colon!

more on this later

Goes inside a module, just like a function definition.

The Class 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>

<assignment statements>

<any other statements also allowed>

to define **attributes**

Goes inside a module, just like a function definition.

The Class Definition

Keyword **class**
Beginning of a class definition

class *<class-name>*(object):

Do not forget the colon!

more on this later

Specification (similar to one for a function)

"""Class specification"""

<function definitions>

to define **methods**

<assignment statements>

to define **attributes**

<any other statements also allowed>

Goes inside a module, just like a function definition.

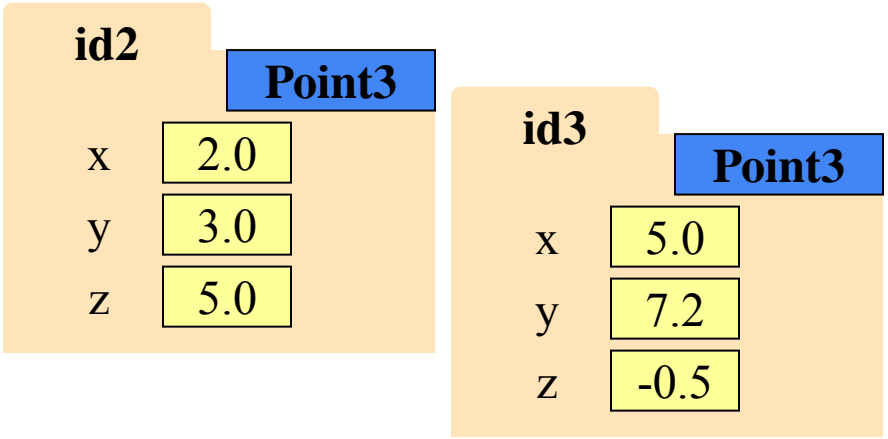
The Simplest Class

```
"""
A module with a simplest class possible
Author: XYZ
Date: Feb 16, 2023
"""
class Example(object):
    """
    instances of this class do nothing
    """
    pass
```

Classes Have Folders Too

Object Folders

- Separate for each *instance*



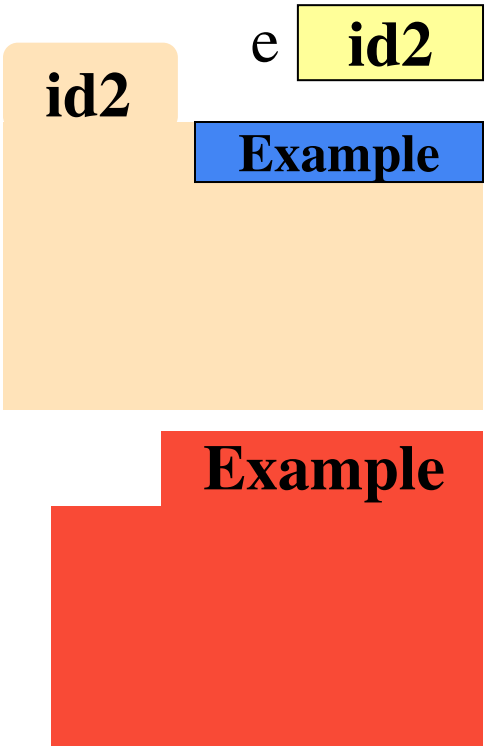
Class Folders

- Data common to all instances

Created by class definition

Recall: Constructors

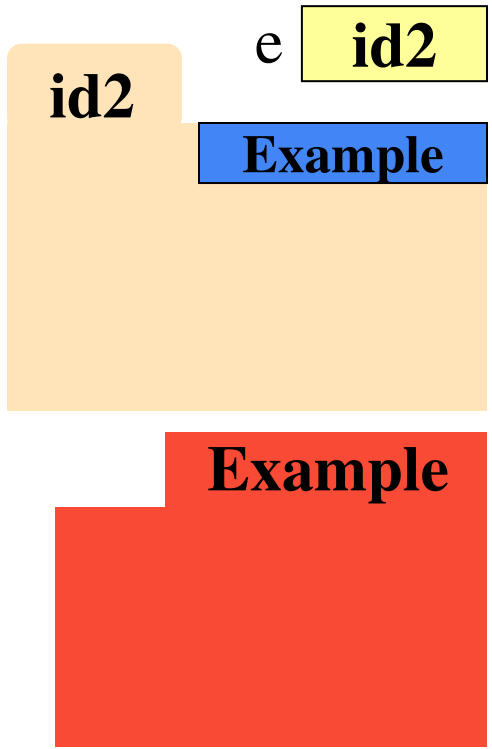
- Function to create new instances
  - Function name == class name
  - Created for you automatically
- Calling the constructor:
  - Makes a new object folder
  - Initializes attributes
  - Returns the id of the folder
- By default, takes no arguments
  - e = Example()



Recall: Constructors

- Function to create new instances
  - Function name == class name
  - Created for you automatically
- Calling the constructor:
  - Makes a new object folder
  - Initializes attributes
  - Returns the id of the folder
- By default, takes no arguments
  - e = Example()

Will come back to this

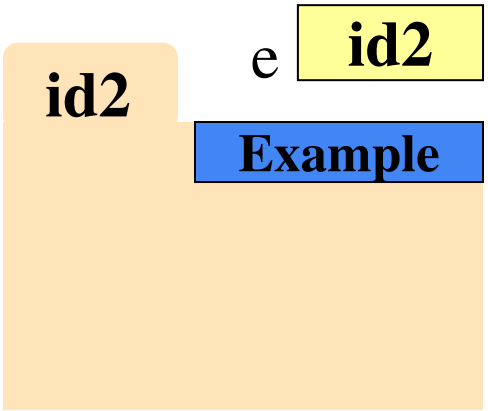


Folder Observations

- By default, the folders are empty
  - Nothing inside of the class folder
  - Nothing inside each object folder either
- We have to write code to put stuff there
  - Empty definition = empty folders
- Code must provide the features objects have
  - **Attributes**, or variables inside of folder
  - **Methods**, or functions inside of folder

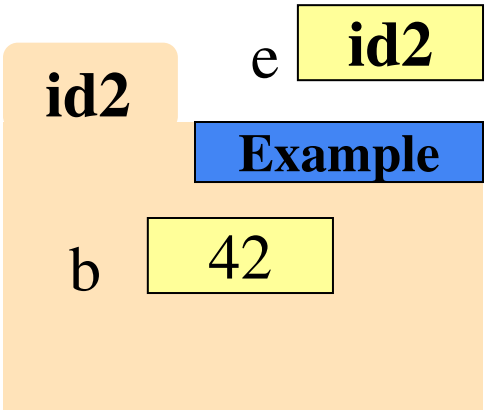
Instances and Attributes

- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: `e.b = 42`



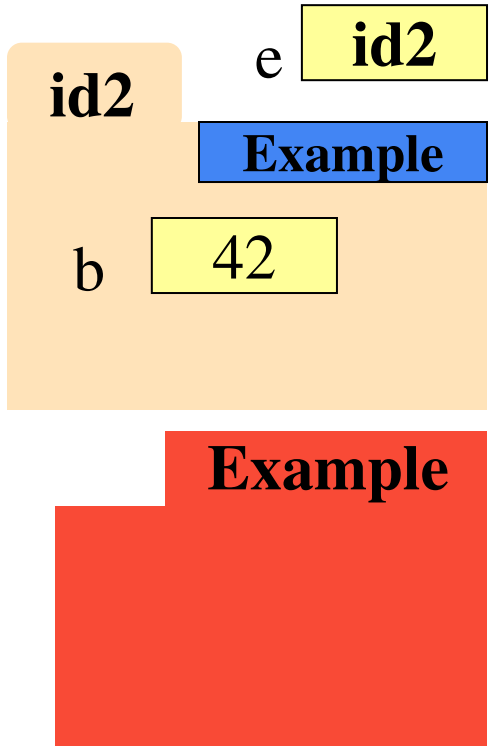
Instances and Attributes

- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: e.b = 42



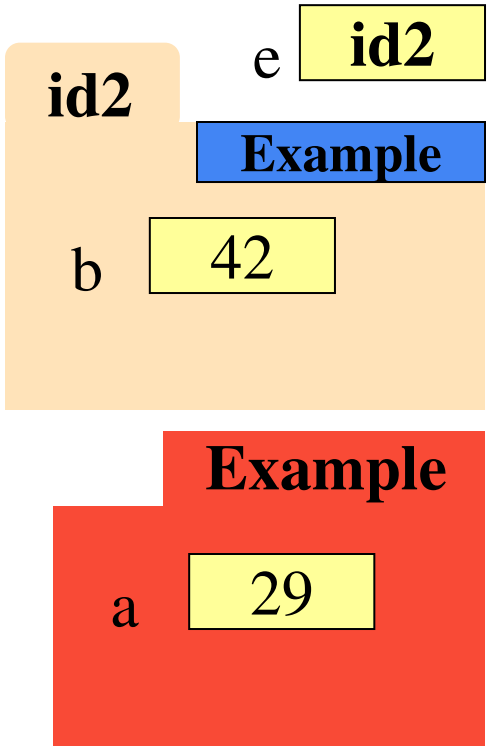
Instances and Attributes

- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: e.b = 42
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: Example.a = 29



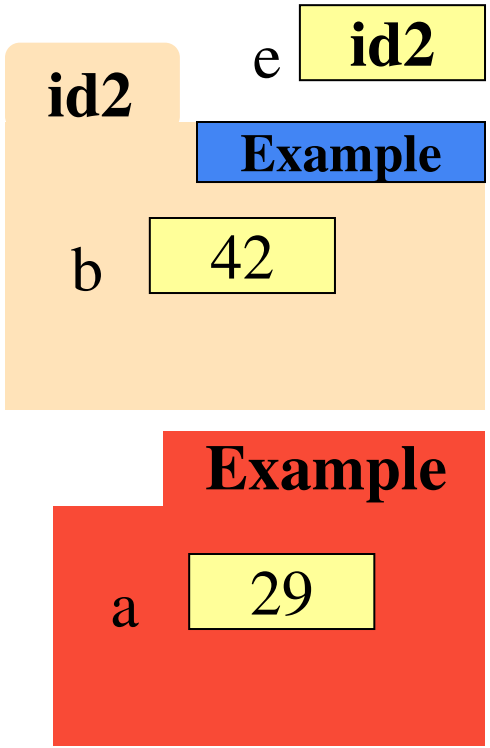
Instances and Attributes

- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: e.b = 42
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: Example.a = 29



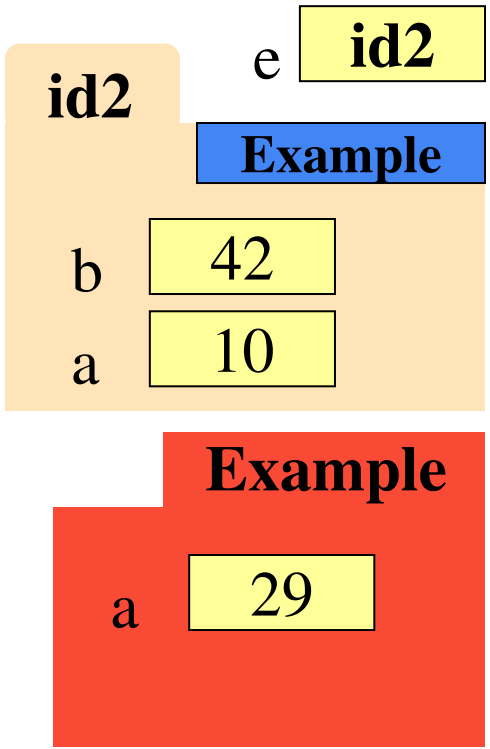
Instances and Attributes

- 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`



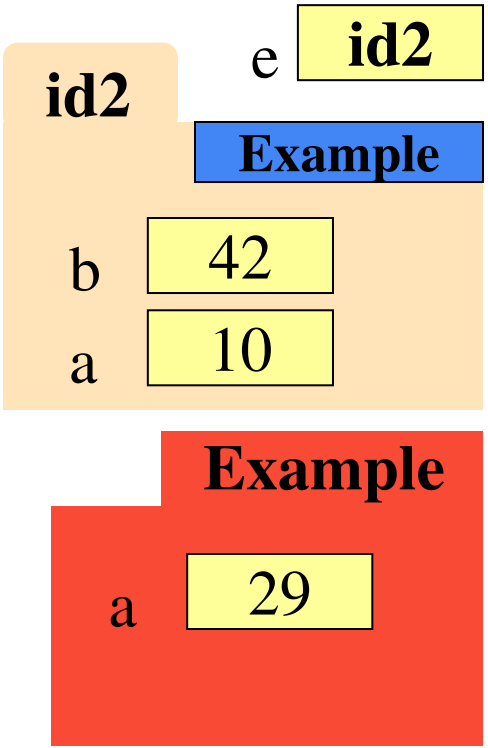
Instances and Attributes

- 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`



Instances and Attributes

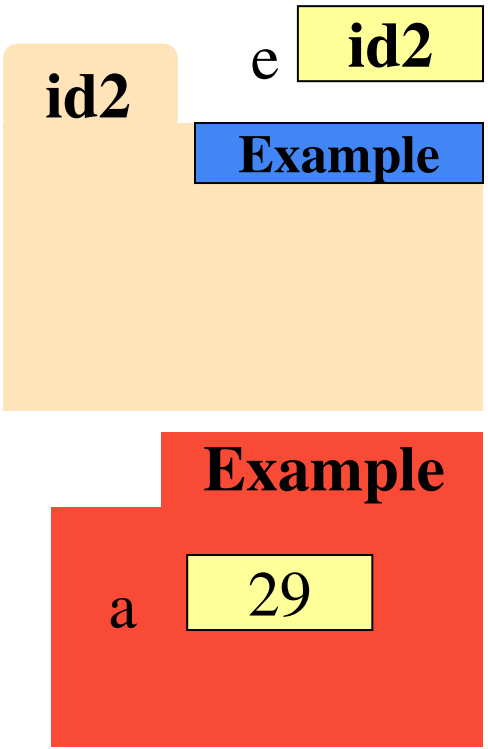
- Assignments add object attributes
  - <object>.<att> = <expression>
  - **Example**: `e.b = 42`
- Assignments can add class attributes
  - <class>.<att> = <expression>
  - **Example**: `Example.a = 29`
- Objects can access class attributes
  - **Example**: `print e.a`
  - But assigning it creates object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



How it Fits in a Definition

```
class Example(object):
    """
    The simplest possible class.
    """
    # A class attribute
    a = 29
```

Puts variable in class folder, not object folder



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**:
  - Point3 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):
    """A class representing a worker in a certain organization

    Instance has basic worker info, but no salary information.

    Attribute lname: The worker last name
    Invariant: lname is a string

    Attribute ssn: The Social Security number
    Invariant: ssn is an int in the range 0..999999999

    Attribute boss: The worker's boss
    Invariant: boss is an instance of Worker, or None if no boss"""
```

The Class Specification

```
class Worker(object):
    """A class representing a worker in a certain organization

    Instance has basic worker info, but no salary information.

    Attribute lname: The worker last name
    Invariant: lname is a string

    Attribute ssn: The Social Security number
    Invariant: ssn is an int in the range 0..999999999

    Attribute boss: The worker's boss
    Invariant: boss is an instance of Worker, or None if no boss"""
```

Short summary

The Class Specification

```
class Worker(object):
    """A class representing a worker in a certain organization

    Instance has basic worker info, but no salary information.

    Attribute lname: The worker last name
    Invariant: lname is a string

    Attribute ssn: The Social Security number
    Invariant: ssn is an int in the range 0..999999999

    Attribute boss: The worker's boss
    Invariant: boss is an instance of Worker, or None if no boss"""
```

Short summary

More detail

The Class Specification

```
class Worker(object):
    """A class representing a worker in a certain organization

    Instance has basic worker info, but no salary information.

    Attribute lname: The worker last name
    Invariant: lname is a string

    Attribute ssn: The Social Security number
    Invariant: ssn is an int in the range 0..999999999

    Attribute boss: The worker's boss
    Invariant: boss is an instance of Worker, or None if no boss"""
```

Short summary

More detail

Description

Invariant

Recall: Objects can have Methods

- Object before the name is an *implicit* argument
- **Example**: distance

```
>>> p = Point3(0,0,0)   # First point
>>> q = Point3(1,0,0)   # Second point
>>> r = Point3(0,0,1)   # Third point
>>> p.distance(r)       # Distance between p, r
1.0
>>> q.distance(r)       # Distance between q, r
1.4142135623730951
```

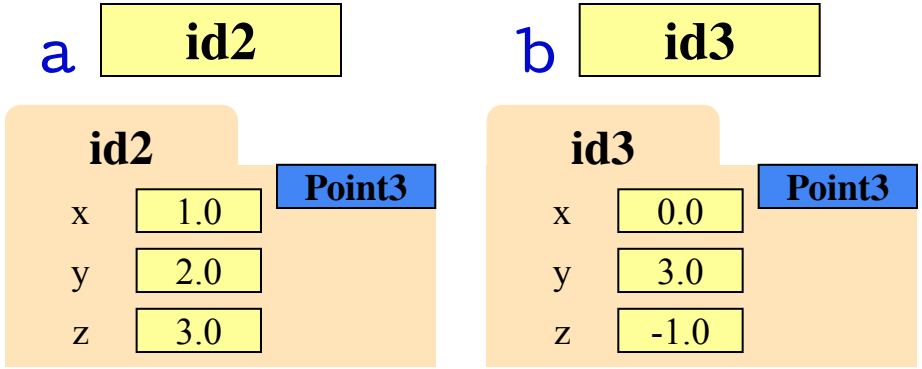
Method Definitions

- Looks like a function def
  - Indented *inside* class
  - First param is always self
  - But otherwise the same
- In a **method call**:
  - One less argument in ()
  - Obj in front goes to self
- **Example**: a.distance(b)

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
7.          """Returns dist from self to q
8.          Precondition: q a Point3"""
9.          assert type(q) == Point3
10.         sqrdst = ((self.x-q.x)**2 +
11.                   (self.y-q.y)**2 +
12.                   (self.z-q.z)**2)
13.         return math.sqrt(sqrdst)
```

Methods Calls

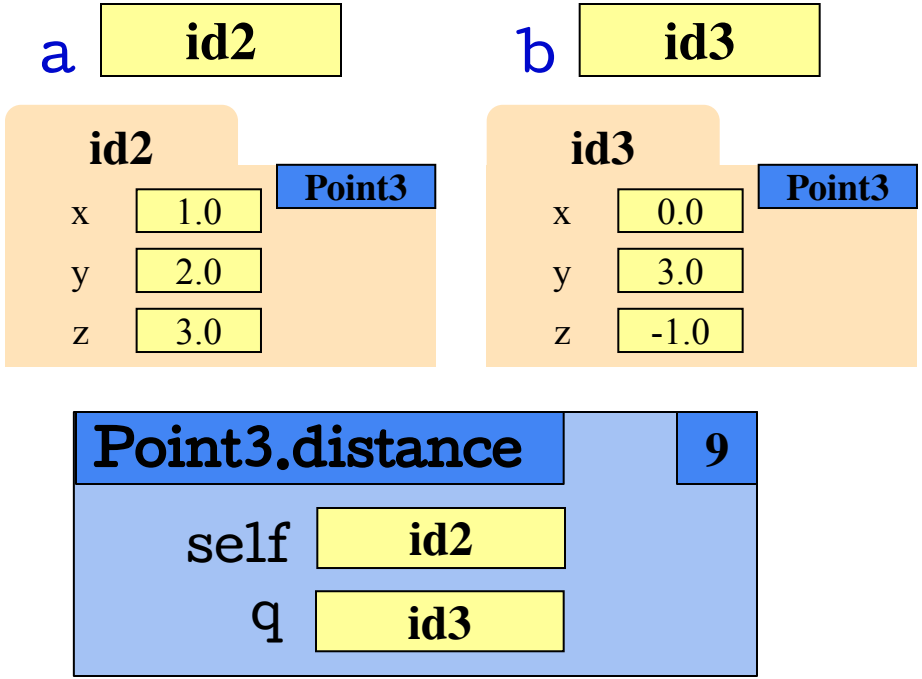
- **Example**: a.distance(b)




```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
7.          """Returns dist from self to q
8.          Precondition: q a Point3"""
9.          assert type(q) == Point3
10.         sqrdst = ((self.x-q.x)**2 +
11.                   (self.y-q.y)**2 +
12.                   (self.z-q.z)**2)
13.         return math.sqrt(sqrdst)
```

Methods Calls

- **Example**: a.distance(b)




```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.      def distance(self,q):
7.          """Returns dist from self to q
8.          Precondition: q a Point3"""
9.          assert type(q) == Point3
10.         sqrdst = ((self.x-q.x)**2 +
11.                   (self.y-q.y)**2 +
12.                   (self.z-q.z)**2)
13.         return math.sqrt(sqrdst)
```

Methods and Folders

- Function definitions…
  - make a folder in heap
  - assign name as variable
  - variable in global space

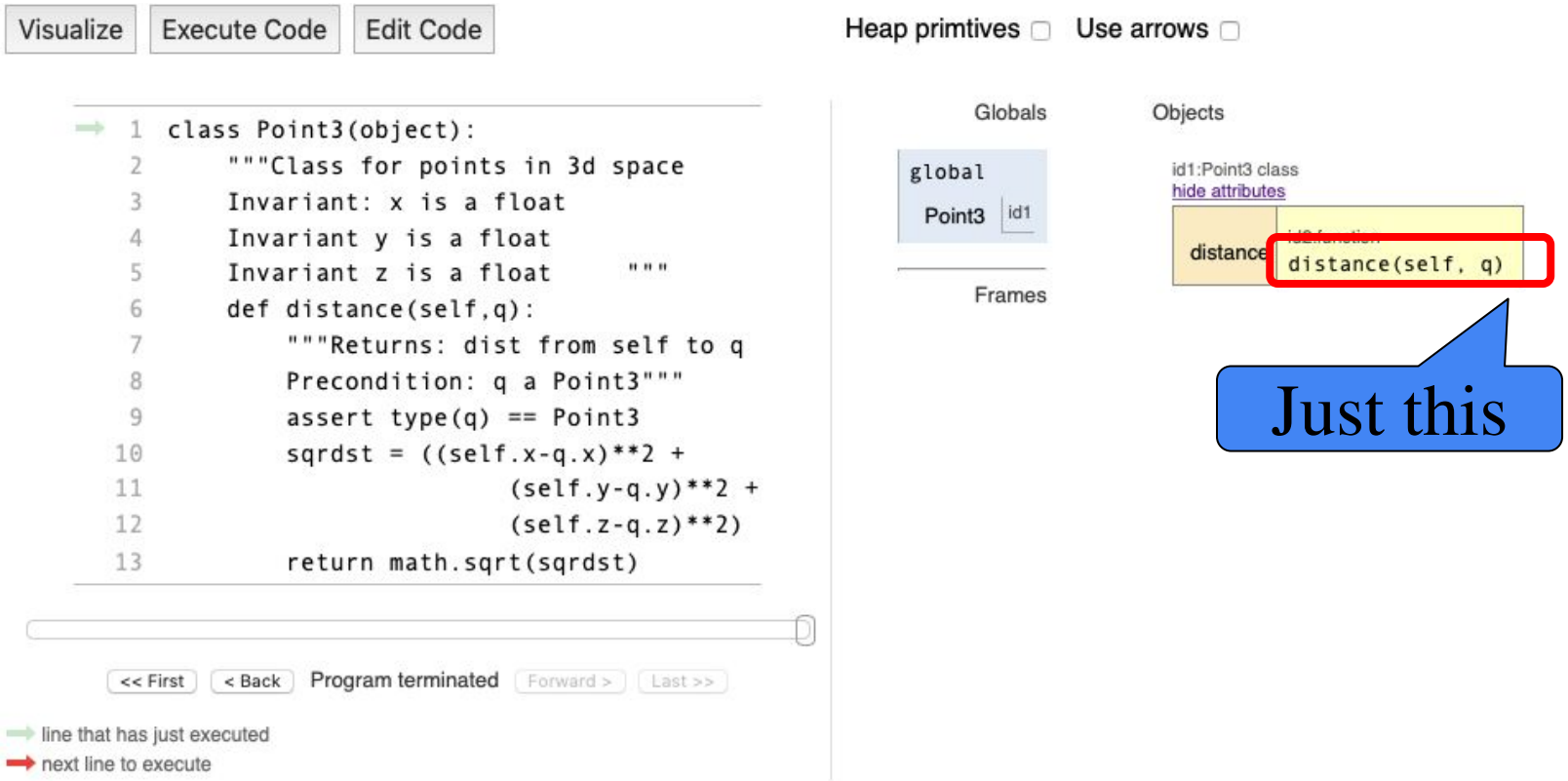
```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float """
6.      def distance(self,q):
```

Methods and Folders

- Function definitions…
  - make a folder in heap
  - assign name as variable
  - variable in global space
- Methods are similar...
  - Variable in **class folder**
  - But otherwise the same
- **Rule of this course**
  - Put header in class folder
  - Nothing else!

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float """
6.      def distance(self,q):
```

Methods and Folders



Initializing the Attributes of an Object (Folder)

- Creating a new Worker is a multi-step process:
  - `w = Worker()`
  - `w.lname = 'University'`

Initializing the Attributes of an Object (Folder)

- Creating a new Worker is a multi-step process:
  - `w = Worker()`
  - `w.lname = 'University'`

- Want to use something like

  `w = Worker(University, 123, None)`
  - Create a new Worker **and** assign attributes
  - `lname` to University, `ssn` to 123, and `boss` to None

Initializing the Attributes of an Object (Folder)

- Creating a new Worker is a multi-step process:
  - `w = Worker()`
  - `w.lname = 'University'`

- Want to use something like

  `w = Worker(University, 123, None)`
  - Create a new Worker **and** assign attributes
  - `lname` to University, `ssn` to 123, and `boss` to None

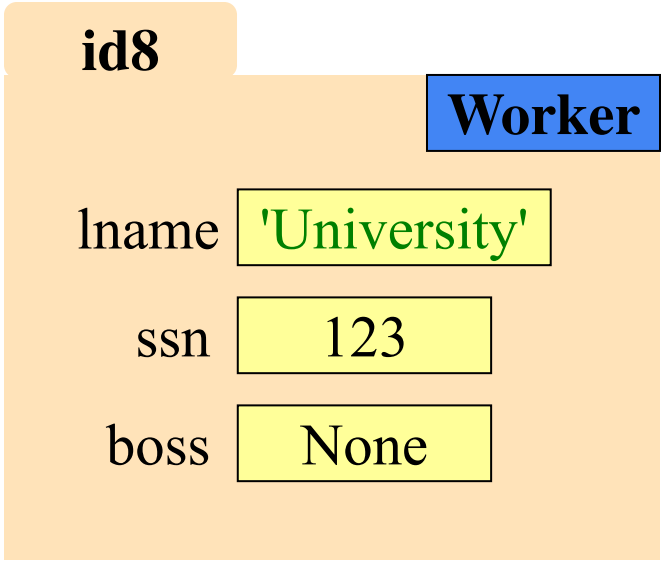
- Need a **custom constructor**

Special Method: __init__

```
w = Worker('University', 123, None)
```

Called by the constructor

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

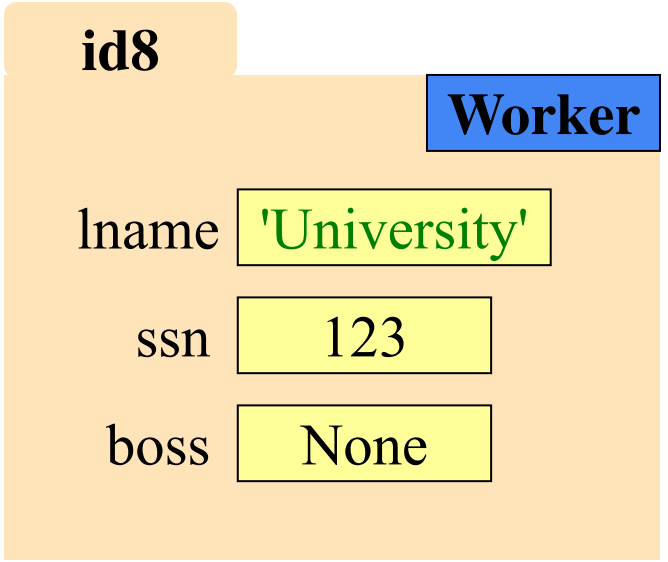


Special Method: __init__

w = two underscores ersity', 123, None)

Called by the constructor

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



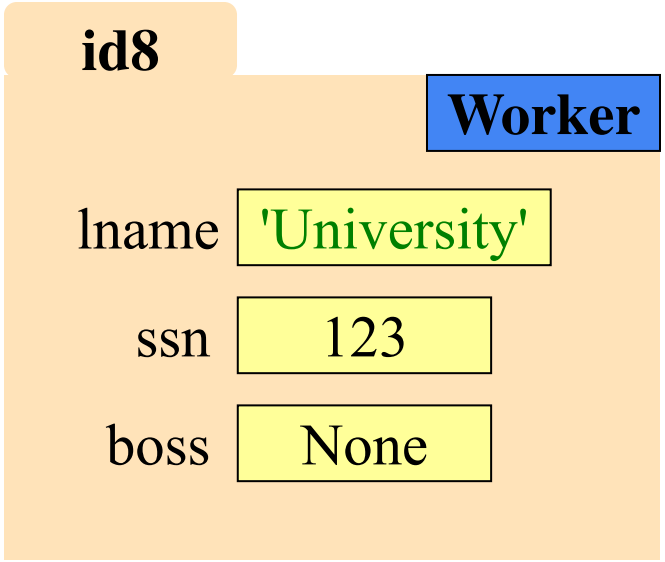
Special Method: __init__

```
w = two underscores ersity', 123, None)
```

Called by the constructor

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

don't forget self



Special Method: __init__

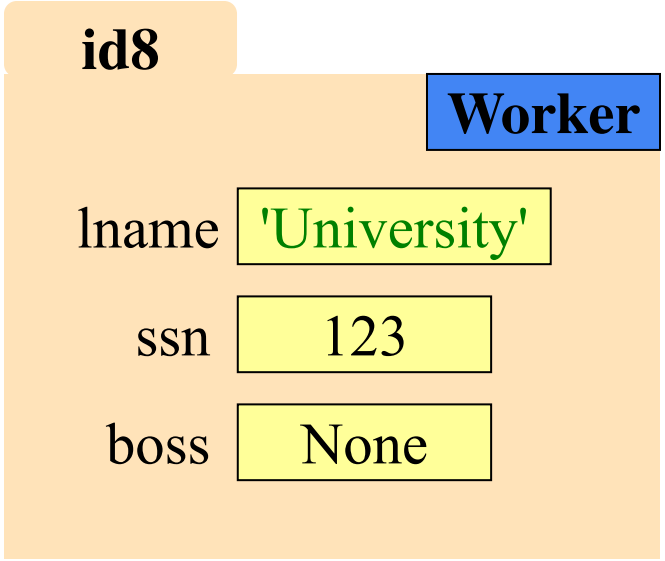
w = **two** underscores ersity', 123, None)

Called by the constructor

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

don't forget self

use self to assign attributes

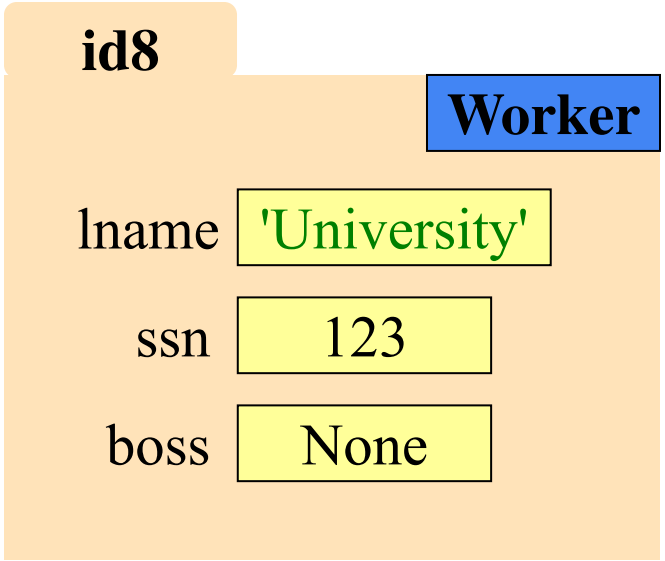


Evaluating a Constructor Expression

SITARE University

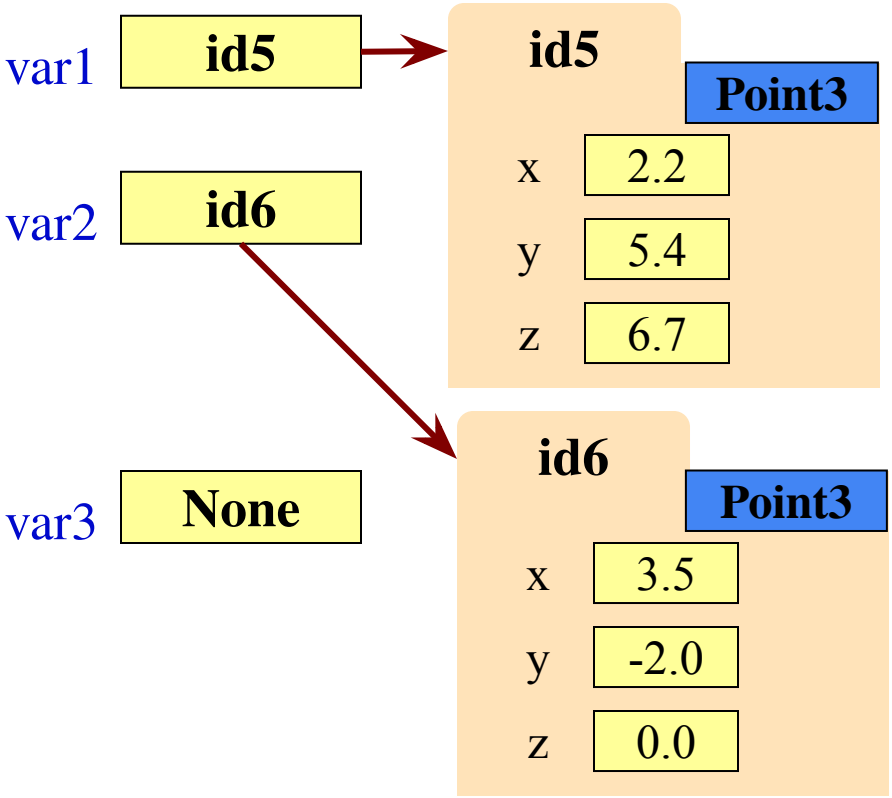
`Worker('University', 123, None)`

1. Creates a new object (folder) of the class Worker
   - Instance is initially empty
2. Puts the folder into heap space
3. Executes the method `__init__`
   - Passes folder name to self
   - Passes other arguments in order
   - Executes the (assignment) commands in initializer body
4. Returns the object (folder) name



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
- **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 Point3 to use?



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 = Point3() # (0,0,0)
  - p = Point3(1,2,3) # (1,2,3)
  - p = Point3(1,2) # (1,2,0)
  - p = Point3(y=3) # (0,3,0)
  - p = Point3(1,z=2) # (1,0,2)

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float   """
6.
7.      def __init__(self,x=0,y=0,z=0):
8.          """Initializes a new Point3
9.          Precond: x,y,z are numbers"""
10.         self.x = x
11.         self.y = y
12.         self.z = z
```

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 = Point3()` `# (0,0,0)`
  - `p = Point3(1,2,3)` `# (1,2,3)`
  - `p = Point3(1,2)` `# (1,2,0)`
  - `p = Point3(y=3)` `# (0,3,0)`
  - `p = Point3(1,z=2)` `# (1,0,2)`

Assigns in order

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.
7.      def __init__(self,x=0,y=0,z=0):
8.          """Initializes a new Point3
9.          Precond: x,y,z are numbers"""
10.         self.x = x
11.         self.y = y
12.         self.z = z
```

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 = Point3() # (0,0,0)
  - p = Point3(1,2,3)
  - p = Point3(1,2)
  - p = Point3(y=3) # (0,3,0)
  - p = Point3(1,z=2) # (1,0,2)

Assigns in order

Use parameter name when out of order

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float   """
6.
7.      def __init__(self,x=0,y=0,z=0):
8.          """Initializes a new Point3
9.          Precond: x,y,z are numbers"""
10.         self.x = x
11.         self.y = y
12.         self.z = z
```

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 = Point3()` `# (0,0,0)`
  - `p = Point3(1,2,3)`
  - `p = Point3(1,2)`
  - `p = Point3(y=3)` `# (0,3,0)`
  - `p = Point3(1,z=2)` `# (1,0,2)`

Assigns in order

Use parameter name when out of order

Mix two approaches

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Invariant z is a float    """
6.
7.      def __init__(self,x=0,y=0,z=0):
8.          """Initializes a new Point3
            Precond: x,y,z are numbers"""
            self.x = x
11.         self.y = y
12.         self.z = z
```

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 = Point3()` `# (0,0,0)`
  - `p = Point3(1,2,3)`
  - `p = Point3(1,2)`
  - `p = Point3(y=3)` `# (0,3,0)`
  - `p = Point3(1,z=2)` `# (1,0,2)`

Assigns in order

Use parameter name when out of order

Mix two approaches

```
1.  class Point3(object):
2.      """Class for points in 3d space
3.      Invariant: x is a float
4.      Invariant y is a float
5.      Inva
6.
7.                 ,x=0,y=0,z=0):
8.          Initializes a new Point3
            Precond: x,y,z are numbers"""
            self.x = x
11.         self.y = y
12.         self.z = z
```

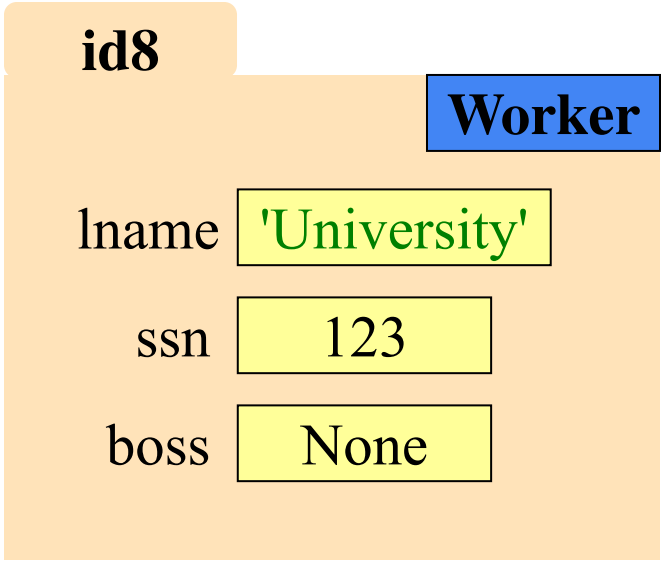
Not limited to methods. Can do with any function.

Special Method: __init__

w = two underscores ersity', 123, None)

Called by the constructor

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



Special Method: __init__

w = two underscores ersity', 123, None)

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

Are there other special methods that we can use?

Example: Converting Values to Strings

SITARE University

str() Function

- **Usage**: str(<expression>)
  - Evaluates the expression
  - Converts it into a string
- How does it convert?
  - str(2) → '2'
  - str(True) → 'True'
  - str('True') → 'True'

repr() Function

- **Usage**: repr(<expression>)
  - Evaluates the expression
  - Converts it into a string
- How does it convert?
  - repr(2) → '2'
  - repr(True) → 'True'
  - repr('True') → "'True'"

What Does str() Do On Objects?

- Does **NOT** display contents

```
>>> p = Point3(1,2,3)
>>> str(p)
'<Point3 object at 0x1007a90>'
```

- Must add a special method
  - __str__ for str()
  - __repr__ for repr()
- Could get away with just one
  - repr() requires __repr__
  - str() can use __repr__ (if __str__ is not there)

```
class Point3(object):
    """Class for points in 3d space"""

    def __str__(self):
        """Returns: string with contents"""
        return '('+str(self.x) + ',' +
                   str(self.y) + ',' +
                   str(self.z) + ')'

    def __repr__(self):
        """Returns: unambiguous string"""
        return str(self.__class__)+
               str(self)
```