Lecture 18

Classes
Announcements for This Lecture

Assignments

- **A4 Friday** at midnight
  - Hopefully you are on Task 4
  - That and task 5 are hardest
- Will post **A5** on Saturday
  - Written assignment like A2
  - Needs material from next Tues
- Will post **A6** on Monday.
  - Not due until November 14
  - But it is much longer!

Optional Videos

- **Videos 20.1-20.8** today
- **Videos 20.9-20.10** next time
- Also **Lesson 21** next time

Exams

- Last week for regrades
  - Limit them to valid issues
- We will do them **eventually**
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

| id2 | Point3
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>2.0</td>
</tr>
<tr>
<td>y</td>
<td>3.0</td>
</tr>
<tr>
<td>z</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Types

- int
- float
- bool
- str

Classes

- Point3
- RGB
- Turtle
- Window
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**
Classes Have Folders Too

Object Folders

- Separate for each *instance*

Class Folders

- Data common to all instances
The Class Definition

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

Example

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

Do not forget the colon!

more on this later

…but not often used

Example

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

Python creates after reading the 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()
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
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
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**: \(\text{print}(e.a)\)
  ▪ But assigning it creates object attribute
  ▪ **Example**: \(e.a = 10\)

• **Rule**: check object first, then class
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**
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"""
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"")

Warning: New format since 2019. 2018 exams will be very different.
Recall: Objects can have Methods

- Object before the name is an *implicit* argument

- **Example**: distance

  ```python
  >>> 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)`

```
class Point3(object):
    """Class for points in 3d space
    Invariant: x is a float
    Invariant y is a float
    Invariant z is a float"
    def distance(self,q):
        """Returns dist from self to q
        Precondition: q a Point3"
        assert type(q) == Point3
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        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)`

```python
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

- 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

```
class Point3(object):
    '''Class for points in 3d space
    Invariant: x is a float
    Invariant y is a float
    Invariant z is a float
    '''
    def distance(self, q):
        '''Returns: dist from self to q
        Precondition: q a Point3
        '''
        assert type(q) == Point3
        sqrdst = ((self.x-q.x)**2 +
                  (self.y-q.y)**2 +
                  (self.z-q.z)**2)
        return math.sqrt(sqrdst)
```

Just this
Initializing the Attributes of an Object (Folder)

• Creating a new Worker is a multi-step process:
  ▪ \( w = \text{Worker}() \)
  ▪ \( w.\text{name} = \text{'White'} \)
  ▪ ...

• Want to use something like
  \[ w = \text{Worker('White', 1234, None)} \]
  ▪ Create a new Worker and assign attributes
    ▪ \text{lname} to 'White', \text{ssn} to 1234, and \text{boss} to None

• Need a \textbf{custom constructor}
Special Method: `__init__`

```python
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
```

```
w = Worker('White', 1234, None)
```

Called by the constructor

<table>
<thead>
<tr>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>lname</td>
</tr>
<tr>
<td>ssn</td>
</tr>
<tr>
<td>boss</td>
</tr>
</tbody>
</table>
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

10/24/23
Evaluating a Constructor Expression

Worker('White', 1234, 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 (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 Point3 to use?
A Class Definition

```python
class Example(object):
    def __init__(self, x):
        self.x = x

    def foo(self, y):
        x = self.bar(y + 1)
        return x

    def bar(self, y):
        self.x = y - 1
        return self.x

>>> a = Example(3)
```

Ignoring the class folder, what does the call stack and the heap look like?
Which One is Closest to Your Answer?

A:

```
Ex.__init__  13
  self  id1  x  3
```

B:

```
Ex.__init__  13
  id1  Example
  self  id1  x  3
```

C:

```
id1  Example
  x  3
```

D:

```
id1  Example
  x  3
```
A Class Definition

```python
class Example(object):
    def __init__(self, x):
        self.x = x
    def foo(self, y):
        x = self.bar(y + 1)
        return x
    def bar(self, y):
        self.x = y - 1
        return self.x

>>> a = Example(3)
```

What is the next step?
Which One is Closest to Your Answer?

A: id1
   Example
   x 3

B: id1
   Ex.__init__ 13
   self id1 x 3

C: id1
   Example
   x 3

D: id1
   Ex.__init__ 13
   x 3
A Class Definition

class Example(object):

def __init__(self, x):
    self.x = x

def foo(self, y):
    x = self.bar(y + 1)
    return x

def bar(self, y):
    self.x = y - 1
    return self.x

>>> a = Example(3)

What is the next step?
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.     def __init__(self,x=0,y=0,z=0):
  7.         '''Initializes a new Point3
  8.         Precond: x,y,z are numbers'''
  9.         self.x = x
 10.         self.y = y
 11.         self.z = z
 12.     ...```

10/24/23  Classes  33
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(x=1,y=2)`
- `p = Point3(1,2)`    \# (1,2,0)
- `p = Point3(y=3)`    \# (0,3,0)
- `p = Point3(1,z=2)`    \# (1,0,2)

```python
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 __init__(self,x=0,y=0,z=0):
    7.         """Initializes a new Point3"
    8.         Precond: x,y,z are numbers"
    9.         self.x = x
    10.        self.y = y
    11.       self.z = z
    12.       ...
```
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)`
  - `p = Point3(y=3)`  # (0,3,0)
  - `p = Point3(1,z=2)`  # (1,0,2)

```python
class Point3(object):
    
    '''Class for points in 3d space
    Invariant: x is a float
    Invariant y is a float
    Invariant z is a float   '''

    def __init__(self,x=0,y=0,z=0):
        '''Initializes a new Point3
        Precond: x,y,z are numbers'''
        self.x = x
        self.y = y
        self.z = z
```

Not limited to methods. Can do with any function.