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

Types

- int
- float
- bool
- str

Classes
- Point3
- RGB
- Turtle
- Window

Classes Have Folders Too

Object Folders
- Separate for each instance

Class Folders
- Data common to all instances

Name Resolution for Objects

- \(\langle\text{object}\rangle.\langle\text{name}\rangle\) means
  - Go the folder for object
  - Find attribute/method name
  - If missing, check class folder
  - If not in either, raise error
- What is in the class folder?
  - Data common to all objects
  - First must understand the class definition

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.

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

Python creates instance after reading the class definition

Instances and Attributes

- Assignments add object attributes
  - \(\langle\text{object}\rangle.\langle\text{att}\rangle = \langle\text{expression}\rangle\)
  - Example: x = 42
- Assignments can add class attributes
  - \(\langle\text{class}\rangle.\langle\text{att}\rangle = \langle\text{expression}\rangle\)
  - 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 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]
```

More detail

Attribute: Name

Invariant
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)

```python
def Be careful with Solution The Example In a method call:
  # 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)
```

Methods Calls

```python
def
  # 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)
```

Special Method: `__init__`

- Two underscores __ before and after `init`.
- Called by the constructor
- Precondition: n a string, s an int in range 0-999999999, and b either a Worker or None
- self.name = n
- self.ssn = s
- self.boss = b

```python
def
  # Two underscores __ before and after `init`.
  # Called by the constructor
  # Precondition: n a string, s an int in range 0-999999999, and b either a Worker or None
  # self.name = n
  # self.ssn = s
  # self.boss = b
```

Evaluating a Constructor Expression

Worker('Obama', 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

```python
Worker('Obama', 1234, 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 back 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?

```python
# The `boss` field is a problem:
# `boss` refers to a Worker object
# Some workers have no boss
# Or maybe not assigned yet (the back 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?
```

Making Arguments Optional

- We can assign default values to `__init__` arguments
  - Write as assignments to parameters in definition
  - Parameters with default values are optional

```python
# We can assign default values to `__init__` arguments
# Write as assignments to parameters in definition
# Parameters with default values are optional
```

```python
def
  # 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(x=1, y=2)` # (1,2,0)
  # `p = Point3((x,y,z))` # (0,0,0)
  # `p = Point3(x=1, y=2, z=3)` # (1,2,3)
```

```python
# We can assign default values to `__init__` arguments
# Write as assignments to parameters in definition
# Parameters with default values are optional
```