Objects

“say something to express one's disapproval of or disagreement with something.”

```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
```

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
```

Fields

Constructor

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
```

Constructor

```
def __init__(self, name, age):
  self.name = name
  self.age = age
```

- Must be named __init__
- Not necessary (by default do nothing)
- Always called when object created

`self` argument

```python
def __init__(self, name, age):
  self.name = name
  self.age = age
```

- Gives access to **receiving** object
  - A method is **always** called “on” an object
- Every method takes at least one parameter
  - Can be named anything, self is convention

Do not do this

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def foo(): return self.age

p1 = Person("John", 36)
```

foo expects at least one parameter

An **object** is a collection of:
- properties (fields)
- methods

A **class** is like a **blueprint** for making objects

[Draw runtime representation on board..]

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)
```

Definitions..

- Dynamic: relating to the runtime execution of the program
- Static: relating to the source of the program alone
    - I.e., not at runtime

Objects are the **dynamic** representation

Classes are the **static** representation

Example: Pair

- Design a “Pair” class
- Should have two properties: left and right
    - Build these in constructor
- Two “accessor” methods:
    - getLeft()
    - getRight()

Message Passing

- An object’s methods respond to **messages**
- Calling an object method analogous to sending message
- Messages can **change object’s state**

Message Passing Qs

- In example, which messages could the object receive?
- [Draw example on board of where object is represented]

Example: Rectangle

- Build a class with the following properties / fields:
    - Width
    - Height

- And the following methods:
    - __init__(self,width,height)
    - calculateArea(self)
    - setHeight(self,height)
    - setWidth(self,width)
    - getWidth(self)
    - getHeight(self)

Example: Using Rectangle

- Construct 2 rectangles:
  - 8 x 12
  - 4 x 4

- Calculate their areas

Example: Caching Area

- Might not want to recompute area every time
- Add another field (in __init__) called cachedArea
    - Set it to None initially
- When area() called, check if cachedArea == None
    - If so, calculate area and set cachedArea
    - If not, return cachedArea

Information Hiding

- The principle that program components should hide their underlying representations
- OO **enables** information hiding in many ways:
    - One is accessors / getters / setters
- Nothing in **Python** prevents you from accessing fields outside of object
    - But—by convention—it is often a bit faux pas
    - Other languages **do** forbid this (e.g., private fields in Java)

Types for Objects

- Basically: Python has no real concept of an object’s type
- Simply regarded as the collection of fields / methods
  - Equivalently: the set of messages to which it responds
- This concept called “duck typing”

Types for Objects

- Basically: Python has no real concept of an object’s type
- Simply regarded as the collection of fields / methods
  - Equivalently: the set of messages to which it responds
- This concept called “duck typing”

"If it walks like a **duck** and it quacks like
a **duck**, then it must be a **duck**"

Example: Circle Object

- Create a “circle” object
  - Needs a “center”
  - Can either have a radius or a diameter (you pick)
  - Must support “area” message

Example: ShapeList

- Create an object ShapeList:
    - One field: underlying list (call this list)
    - __init__(self):
        - Initialize list (to empty list)
    - length(self): calculates the length of the list
    - add(self,shape):
        - Adds a shape to the underlying list
    - sumOfAreas(self):
        - Sum of the areas of all of the shapes

Testing ShapeList

- Create empty ShapeList
- Add a 8 x 12 rectangle
- Add an 4 x 5 CachedRectangle
- Add a circle centered at (1,3) whose radius is 2
- Call sumOfAreas

Things to know…

- Static vs. dynamic property
- Class vs. Object
- What are fields
- What is a constructor
- What is duck typing
    - Concept of treating object’s type as set of methods to which it responds (and their behaviors)