Objectives

- Vocabulary Check
- Intro to Design Patterns
- Introduction to Object-Oriented Programming
- Introduction to APIs

Identify the Parts of a Program

```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input("What is your favorite color? " )
print("Cool!  My favorite color is _light_", color, "!")

rating = eval(input("On a scale of 1 to 10, how much do
you like Ryan Gosling? "))
print("Cool!  I like him", rating*1.8, "much!")
```

Identify the comments, variables, functions, expressions, assignments, literals

input_demo.py

Identify the Parts of a Program

```
# Demonstrate numeric and string input
# by Sara Sprenkle for CS111
#

color = input( “What is your favorite color? " )
print("Cool!  My favorite color is _light_" , color, "!")

rating = eval(input( "On a scale of 1 to 10, how much do
you like Ryan Gosling? ” )
print("Cool!  I like him” , rating*1.8, "much!")
                                  expression
```

Identify the comments, variables, functions, expressions, assignments, literals

Improving average2.py

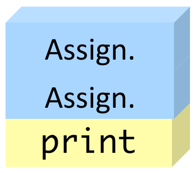
- With what we just learned, how could we improve `average2.py`?
- Example of suggested approach to development
  - Input is going to become fairly routine.
  - Wait on input until you have figured out the rest of the program/problem.

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution
- Example (Standard Algorithm)
  - Get input from user
  - Do some computation
  - Display output



```
x = input("…")
ans = …
print(ans)
```

Programming Paradigm: Imperative

- Most modern programming languages are imperative
- Have data (numbers and strings in variables)
- Perform operations on data using operations, such as + (addition and concatenation)
- Data and operations are separate

- Add to imperative:
  object-oriented programming

Super Power: Psychokinesis

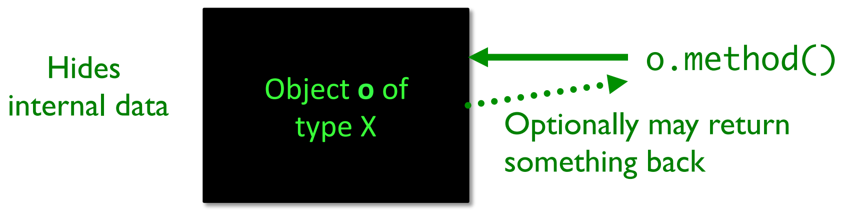
OBJECT-ORIENTED PROGRAMMING

Object-Oriented Programming

- Program is a collection of **objects**
- Objects **combine** data and methods together
- Objects interact by invoking **methods** on other objects
  - Methods perform some operation on object

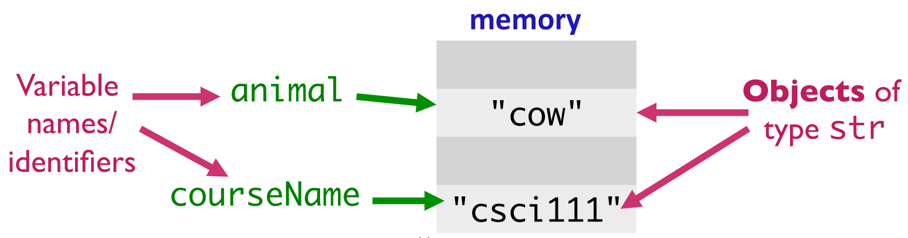
Object-Oriented Programming

- Program is a collection of **objects**
- Objects **combine** data and methods together
- Objects interact by invoking **methods** on other objects
  - Methods perform some operation on object



Object-Oriented Programming

- We've been using objects
  - Just didn't call them objects
- For example: `str` is a data type (or **class**)
  - We created objects of type (*class*) `string`
    - `animal = "cow"`
    - `coursename = "csci111"`



Example of OO Programming Abstraction

- Think of a car– It's an ***object***
- What can you do to a car?

Example of OO Programming Abstraction

- Think of a Car– it's an ***object***
- What can you do to a car?
  - Turn it on/off
  - Change gears
  - Press gas
  - Brake
  - Check fuel left
  - …

  methods
- You don't know ***how*** that operation is being done (i.e., implemented)
  - Just know ***what it does*** and that it ***works***

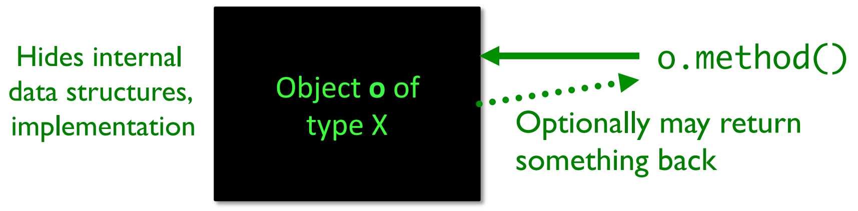
Example of OO Programming Abstraction

- A car is an ***object***
- ***Methods*** you can call on your car:
  - Turn on/off
  - Change gears
  - Press gas pedal
  - Brake
  - Check speed
  - …
- `Car` is a ***class***, a.k.a., a data ***type***
  - Public safety's car (identified by `psCar`) is an object of type `Car`
  - You can call the above methods on any object of type Car

Object-Oriented Programming

- Objects combine data *and* methods together

Provides **interface** (*methods*) that users interact with


Hides internal data structures, implementation
Object **o** of type X
o.method()
Optionally may return something back

Use an Application Programming Interface (**API**) to interact with a set of classes.

Class Libraries

- Python provides libraries of classes
  - ➢ Defines methods that you can call on objects from those classes
  - ➢ `str` class provides a bunch of useful methods
    - More on that later
- Third-party libraries
  - ➢ Written by non-Python people
  - ➢ Can write programs using these libraries too

Benefits of Object-Oriented Programming

- **Abstraction**
  - Hides details of underlying implementation
  - Easier to change implementation
- Easy reuse of code
- Collects related data/methods together
  - Easier to reason about data
- Less code in main program

Using a Graphics Module/Library

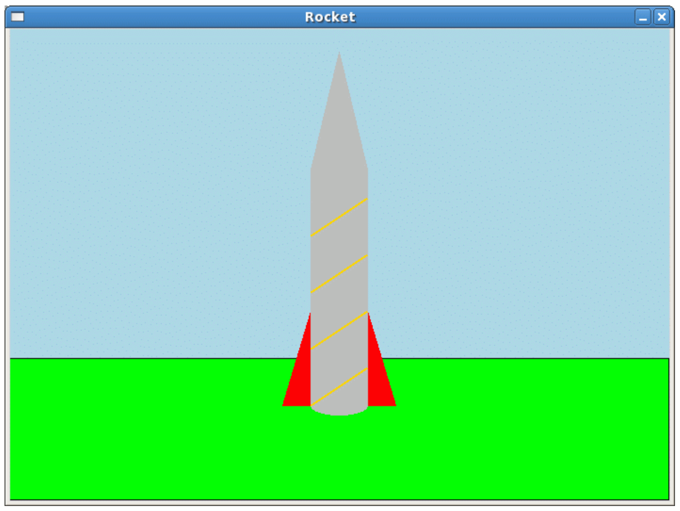
- Allows us to handle graphical input and output
  - Example output: Pictures
  - Example input: Mouse clicks
- Defines a collection of related graphics **classes**
- Not part of a standard Python distribution
  - Need to **import** from `graphics.py`
- Use the library to help us learn OO programming

USING A GRAPHICS MODULE

Using a Graphics Module/Library

- Handout lists the various classes
  - **Constructor** is in bold
    - Creates an object of that type
  - For each class, lists *some* of their methods and parameters
  - Drawn objects have some common methods
    - Listed at end of handout
- Known as an **API**
  - **Application Programming Interface**

Example of Output



Using the API: **Constructors**

- To create an object of a certain type/class, use the **constructor** for that type/class
  - Syntax:
    ```
    objName = ClassName([parameters])
    ```
  - Note:
    - Class names typically begin with capital letter
    - Object names begin with lowercase letter
  - `objname` is known as an **instance** of the class
- Example: To create a `GraphWin` object that's identified by `window`
  ```
  window = GraphWin("My Window",200,200)
  ```

Using the API: Methods

- To call a **method** on an object,
  - Syntax:

```
objName.methodName([parameters])
```

  - Method names typically begin with lowercase letter
  - Similar to calling *functions*
- Example: To change the background color of a `GraphWin` object named `window`

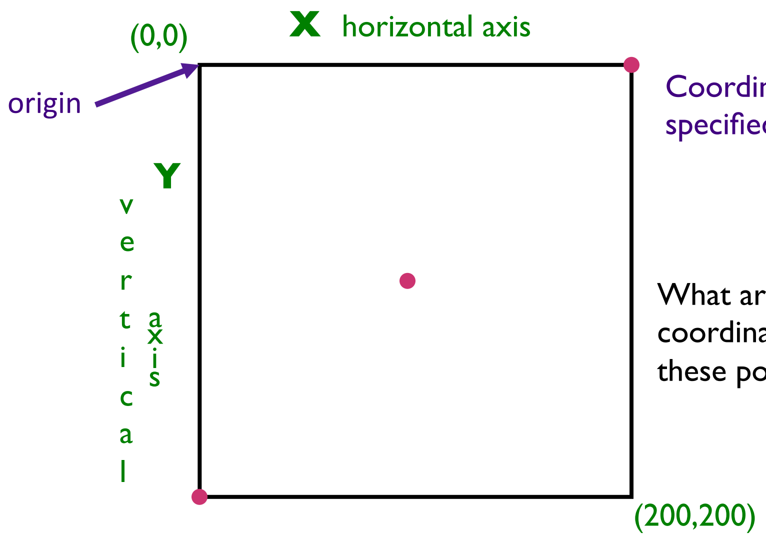
```
window.setBackground("blue")
```

Using the API: Methods

- A method sometimes **returns** output, which you may want to save in a variable
  - Class's API should say if method returns output
- Example: if you want to know the *width* of a `GraphWin` object named `window`

```
width = window.getWidth()
```

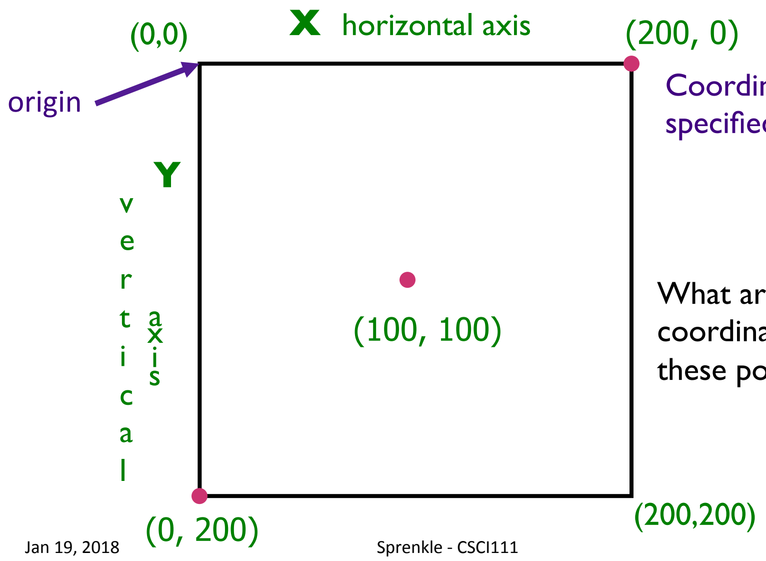
A `GraphWin` Object's Canvas



Coordinates are specified as (x,y)

What are the coordinates for these points?

A `GraphWin` Object's Canvas



Coordinates are specified as (x,y)

What are the coordinates for these points?

The `GraphWin` Class

- All parameters to the constructor are optional
- Could call constructor as

Call	Meaning
`GraphWin()`	Title, width, height to defaults (“Graphics Window”, 200, 200)
`GraphWin(<title>)`	Width, height to defaults
`GraphWin(<title>,<width>)`	Height to default
`GraphWin(<title>, <width>, <height>)`	

The `GraphWin` API

- **Accessor** methods for `GraphWin`
  - Return some information about the `GraphWin`
- Example methods:
  - `<GraphWinObj>.getWidth()`
  - `<GraphWinObj>.getHeight()`

The `GraphWin` API

- `<GraphWinObj>.setBackground(<color>)`
  - Colors are strings, such as “red” or “purple”
    - Can add numbers to end of string for darker colors, e.g., "red2", "red3", "red4"

```
win = GraphWin()
win.setBackground("purple")
```

  - Does *not return* anything to shell
  - Called for change in `win`'s state, i.e., this method is a **mutator**

General Categories of Methods

- Accessor
  - Returns information about the object
  - Example: `getWidth()`
- Mutator
  - Changes the state of the object
    - i.e., changes something about the object
  - Example: `setBackground()`

What Does This Code Do?

- Use OO terminology previously defined

```
from graphics import *

win = GraphWin("My Circle", 200, 200)
point = Point(100,100)
c = Circle(point, 10)
c.draw(win)
win.getMouse()
```

graphics_test.py

What Does This Code Do?

- Use OO terminology previously defined

```
from graphics import *


win = GraphWin("My Circle", 200, 200)
point = Point(100, 100)
c = Circle(point, 10)
c.draw(win)
win.getMouse()
```

Constructor

GraphWin object

Also known as an **instance of** the GraphWin **class**

Method called on GraphWin object

Note: Class names start with capital letters,
Method names start with lowercase letters

Using the Graphics Library

- In general, graphics are drawn on a canvas
  - A canvas is a 2-dimensional grid of pixels

- For our Graphics library, our canvas is a *window*
  - Specifically an **instance of** the `GraphWin` class
  - By default, a `GraphWin` object is 200x200 pixels

Colors

- Strings, such as "blue4"
- Can also create colors using the *function* `color_rgb(<red>,<green>,<blue>)`
  - Parameters in the range [0,255]
  - Example use:

```
darkBlueGreen = color_rgb(10, 100, 100)
win.setBackground(darkBlueGreen)
```

    - Background is a dark blue/green color
  - Example color codes:
    - `http://en.wikipedia.org/wiki/List_of_colors`

Using the Graphics Library

- How do we create an instance of a Rectangle?
- Draw the rectangle?
- Shift the instance of the Rectangle class to the **right** 10 pixels
- What are the x- and y- coordinates of the upper-left corner of the Rectangle now?

`rectangle.py`

OO Terminology Summary

Term	Definition	Examples
Class	A data type. Defines the data and operations for members of the class	`str`, `TV`, `GraphWin`
Object	An *instance* of a specific class	`animal`, `myTV`, `window`
Method	Operations you can call on an object	`setBackground(<color>)`, `getWidth()`
Constructor	Special method to create an object of a certain type/class	`GraphWin()`, `str(1234)`

Looking Ahead

- Lab 2 coming up
  - What picture do you want to draw with the library?
- Fix Broader Issue write up due