Objectives

- Improving program readability
- Introduction to Object-Oriented Programming
- Introduction to APIs
- Problem-solving using APIs

Review

- What are benefits of functions?
- How do we call functions?
- What do we get access to functions that are in a module?
  - How does using the imported functions change with each type of import statement?

Benefits of Functions

- Reuse, simplify code
- Functions written by others
  - Well-written, efficiency
  - Abstraction: it works, that's all that matters!
- Greatly increase code you can write by leveraging others' code

Review: VA Lottery: Pick 4

- To play: you pick 4 numbers between 0 and 9
- To win: select the numbers that are selected by the magic ping-pong ball machine
- Your job: Simulate the magic ping-pong ball machines

`pick4.py`

VA Lottery: Mega Millions

- To play: you pick 5 numbers between 1 and 56
  - Ignoring rule: 1 Mega Ball number between 1 and 46
- Your job: Simulate the result of the magic ping-pong ball machines
  - How difficult to modify the last program?
  - What could we do to make easier?

`megamillions.py`

Constants

- Special variables whose values are defined once and never changed
  - By convention, not enforced by interpreter
- By convention
  - A constant's name is all caps
  - Typically defined at top of program → easy to find, change
- Examples:

```
NUM_CHOICES = 4
MIN_VALUE = 0
```

Never assigned values in remainder of program

Improving Code Readability

- Comments
  - Describe blocks of code at a high level
- Output/Display
  - Descriptive, explains what program outputs
- Constants
  - Change one value (at top of program) to change value everywhere in program
  - Flexible programs
  - Gets rid of "magic numbers"
    - Give a clear name and purpose to values

Improving Code Readability/Usability

- What does this program do?
  - How would you figure it out?
- What would you do to improve the program's readability and usability?

`program_before.py`
`program_after.py`

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

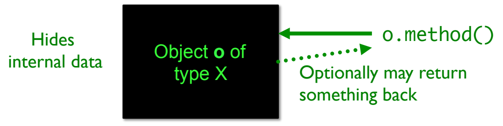
OBJECT-ORIENTED PROGRAMMING

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

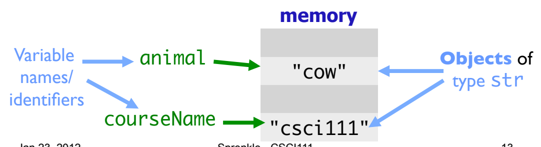
Object-Oriented Programming

- 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 TV – It's an ***object***
- What can you do to your TV using one of two ***interfaces***: the remote or the buttons on the TV?

Example of OO Programming Abstraction

- Think of a TV – it's an ***object***
- What can you do to your TV using one of two ***interfaces***: the remote or the buttons on the TV?
  - Turn on/off
  - Change channel
  - Change volume
  - …

  $\left.\begin{array}{l}\text{Turn on/off}\\\text{Change channel}\\\text{Change volume}\end{array}\right\}$ **methods**
- You don't know ***how*** that operation is being done (i.e., implemented)
  - Just know ***what it does*** and that it ***works***

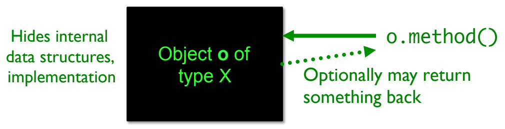
Example of OO Programming Abstraction

- Your TV is an ***object***
- ***Methods*** you can call on your TV:
  - Turn on/off
  - Change channel
  - Change volume
  - …
- TV is a ***class***, a.k.a., a data ***type***
  - Your TV (identified by `myTV`) is an object of type TV
  - You can call the above methods on any object of type TV

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

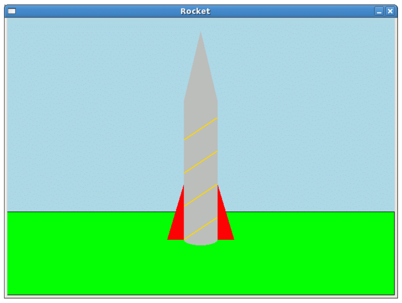
- Allows us to handle graphical input and output
  - Example output: Pictures
  - Example input: Mouse clicks
- Defines a collection of related graphics **classes**
- Not part of a standard Python distribution
  - Need to import from `graphics.py`
- Use the library to help us learn OO programming

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()
```

What Does This Code Do?

- Use OO terminology previously defined

```
from graphics import *

win = GraphWin("My Circle", 100, 100)
point = Point(50,50)
c = Circle(point, 10)
c.draw(win)
win.getMouse()
```

What Does This Code Do?

- Use OO terminology previously defined

```
from graphics import *

win = GraphWin("My Circle", 100, 100)
point = Point(50,50)
c = Circle(point, 10)
c.draw(win)
win.getMouse()
```

Constructor

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

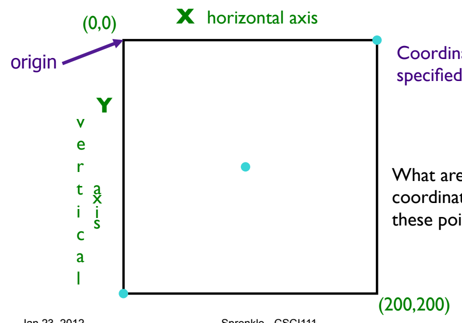
Method called on `GraphWin` object

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

A `GraphWin` Object's Canvas



Coordinates are specified as (x,y)

What are the coordinates for these points?

Reading Code

- After this program executes, what does the window look like?

```
from graphics import *

win = GraphWin("My Circle", 100, 100)
c = Circle(Point(50,50), 10)
c.draw(win)
win.getMouse()
```

graphics_test.py

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**

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`

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()`

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	string, 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

- Broader Issue
  - Read 2 short articles for Friday
- Lab tomorrow
  - Practice for loops, functions, and OO programming
- Next Monday
  - 11:15 a.m. – Andy Danner's GIS/CS talk
  - 10 points extra credit for answering questions on Sakai