Objectives

- Indefinite Loops
- Introduction to Object-Oriented Programming
- Introduction to APIs
- Problem-solving using APIs

Nondeterministic Decisions

- Sometimes, we don’t want to necessarily know that a specific decision is always made
- For example, games often use randomness to make decisions
  - Roll dice
  - Coin flips
  - Location and behavior of baddies

Flipping Coins

- Simulate by randomly selecting between 0 (heads) and 1 (tails)
- Program: coinFlip.py

Flipping Coins

- Problem: How many flips does it take to get 3 consecutive heads?

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

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
    - Hides internal data
    - Optionally may return something back

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?

- 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 (named "myTV") is an object of type TV
    - You can call the above methods on any object of type TV

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"`
    - `pick4Str = str(randnum) + "-"`

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 keys on the TV?
  - Turn on/off
  - Change channel
  - Change volume
  - ...
- You don’t know **how** that operation is being done (i.e., implemented)
- Just know **what it does** and that it **works**

Object-Oriented Programming

- Objects combine data and methods together
  - Provides **interface** (methods) that users interact with
  - Hides internal data structures, implementation
  - 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
  - Not written by Python folks
  - 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

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/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
- From Fall07 class

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 named “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

```python
from graphics import *
win = GraphWin("My Circle", 100, 100)
c = Circle(Point(50,50), 10)
c.draw(win)
win.getMouse()
```

What Does This Code Do?

- Use OO terminology previously defined

```python
from graphics import *
win = GraphWin("My Circle", 100, 100)
c = Circle(Point(50,50), 10)
c.draw(win)
win.getMouse()
```

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?
Snippet of Code

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

```python
from graphics import *

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

The GraphWin Class

- All parameters to the constructor are optional
- Could call constructor as
  - `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()`

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:
    - `win.setBackground(color_rgb(10, 100, 100))`
      - Background is a dark blue/green color
  - Example color codes:

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?

Midterm Results

- Avg: 84%
- Median: 83%
- Common mistakes
  - Budgeting time
  - Too-long answers
    - Only worth 3-5 points; looking for key words
  - Missing parts of questions
  - For loop (up to but not including)
  - Tracing through code (use control flow charts)
  - <= instead of ≤

Midterm Answers

- Difference between these two statements
  - print contestant, “won the”, prize
    - Has spaces between words
  - print contestant + “won the” + prize
    - No spaces between words (except “won the”)

- Truth Table
  - bool1 = False; bool2 = True
  - bool1 or bool2 = True
  - not(bool1 or bool2) = False

This Week

- Lab Tuesday
  - Due Friday
- Broader Issue: DARPA Urban Challenge
  - Write up in Sakai due Friday