Objectives

- Design Patterns
- Introduction to Object-Oriented Programming
- Introduction to APIs
- Broader Issue: Algorithms

Review

- How do we get input from a user?
  - Give example of getting input from a user, one where we want a string and one where we want a number
- What is the testing process? What is our goal in testing?
- Problem: Averaging two numbers
  - What are good test cases?
  - What is your algorithm?

Review: Getting Input From User

- `input` is a ***function***
  - **Function**: A command to do something
    - A "subroutine"
- Syntax:
  - `input(<string_prompt>)`
- Semantics:
  - Display the prompt `<string_prompt>` in the terminal
  - Read in the user's input and *return* it as a string/text

Review: Getting Input From a User

- Save the result of calling input in a variable
  - Ex:

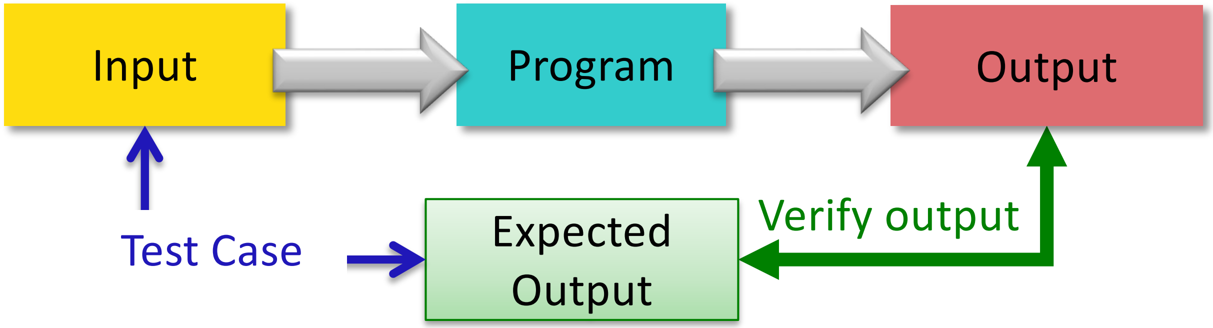
```
color = input("What is your favorite color? " )
```

- If you want the assigned variable to be of type int or float, we need to convert the result of calling input
  - Ex:

```
height = eval(input("Enter the height: " ))
width = float(input("Enter the width: "))
```

Tradeoffs about which function to use to wrap the input.
For this class, either will be correct to use.

Review: Testing Process



- Test case:
  - Input used to test the program
  - Expected output given that input
- Verify if output is what you expected
- Goal: create *good* test cases that will reveal if there is a problem in your code

If output is not what you expect, debug!

Our Development Process

1. Determine algorithm
   a) Calculate average: add two numbers together, divide by 2
   b) Display average
2. Implement algorithm
   a) “Hard-code” two numbers
      - Later: get the two numbers as input from user
   b) Calculate average
   c) Print average

`average2.py`

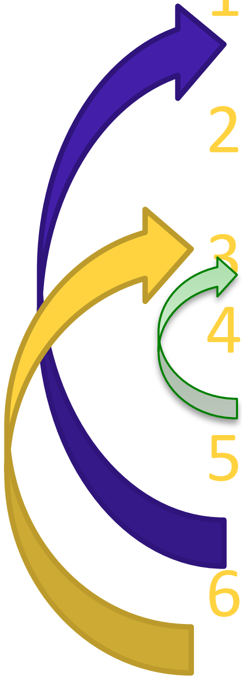
Suggested Approach to Development

- Input is going to become fairly routine.
- Wait to get user input until you have figured out the rest of the program/problem.
- Consider problem 1 in Lab 1
  - You “hard coded” the values of i and j
  - You can (and will) modify the program to get user input for those variables in Lab 2.

Formalizing Process of Developing Computational Solutions

1. Think about expectations/test cases
   - "When user enters these values, this should happen."
2. Create a sketch of how to solve the problem
   (the algorithm)
3. Fill in the details in Python
4. Execute the program ***with good, varied test cases*** *to try to* ***reveal errors***
5. If output doesn't match your expectation, debug the program
   - (Where is the problem? How do I fix it?)
6. Iterate to improve your program
   - Better variable names, better input/output, more efficient, …

Formalizing Process of Developing Computational Solutions



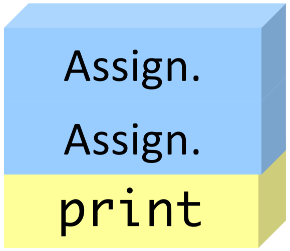
1. Think about expectations/test cases
   - “When user enters these values, this should happen.”
2. Create a sketch of how to solve the problem (the algorithm)
3. Fill in the details in Python
4. Execute the program ***with good, varied test cases*** *to try to* ***reveal errors***
5. If output doesn’t match your expectation, debug the program
   - (Where is the problem? How do I fix it?)
6. Iterate to improve your program
   - Better variable names, better input/output, more efficient, …

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



Assign.

Assign.

print

```
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***

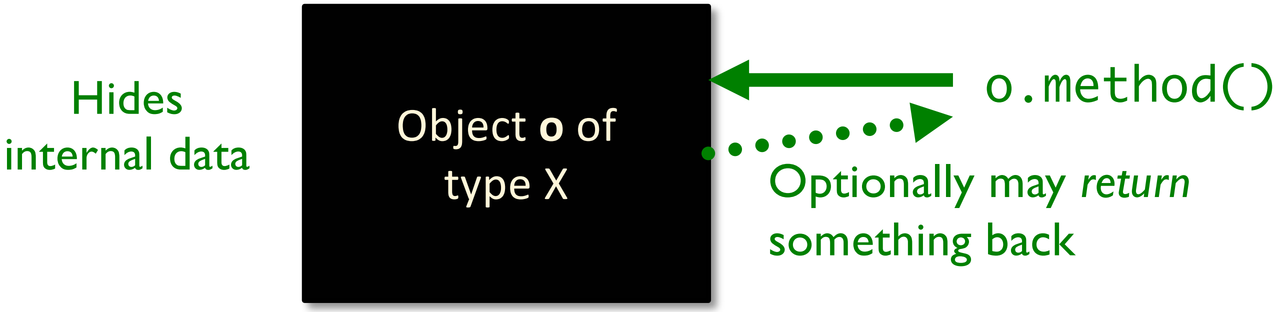
OBJECT-ORIENTED PROGRAMMING

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

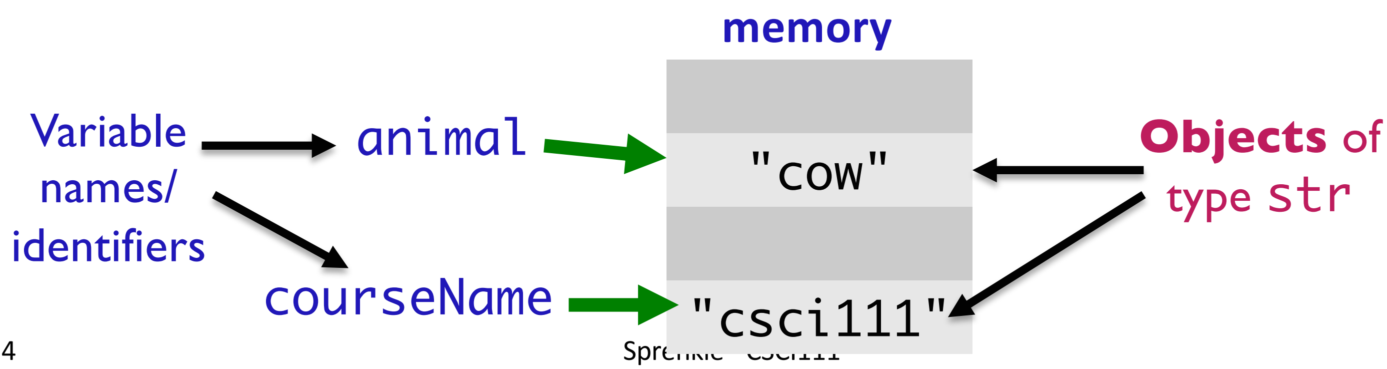
Object-Oriented Programming

- 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"`



memory

Variable names/ identifiers

animal

courseName

"cow"

"csci111"

Objects of type `str`

Example of OO Programming Abstraction

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

Example of OO Programming Abstraction

- Think of a smart phone– it’s an ***object***
- What can you do to a phone? Those are ***methods***
  - Turn it on/off
  - Open applications
  - Make a phone call
  - Mute it
  - Update settings
  - …

  methods

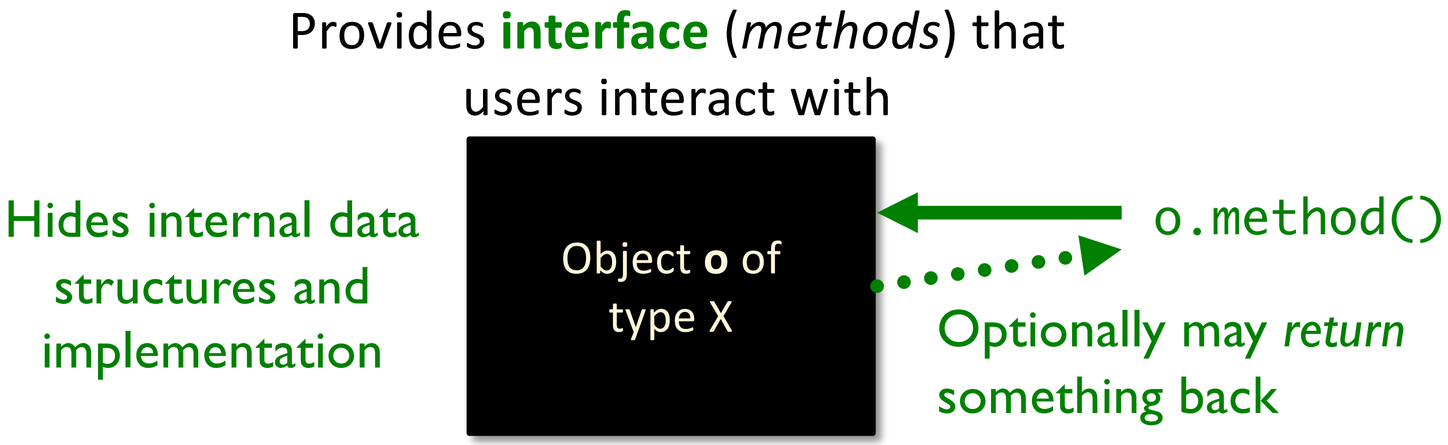
- You don't know ***how*** that operation is being done (i.e., implemented)
  - Just know ***what it does*** and that it ***works***

Example of OO Programming Abstraction

- A smart phone is an ***object***
- ***Methods*** you can call on your smart phone:
  - Turn it on/off
  - Open applications
  - Make a phone call
  - Mute it
  - Update settings
  - …
- `SmartPhone` is a ***class***, a.k.a., a data ***type***
  - My smart phone (identified by `myPhone`) is an object of type `SmartPhone`
  - Call the above methods on any object of type SmartPhone

Object-Oriented Programming

- Objects combine data *and* methods together



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

Hides internal data structures and 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 useful methods
    - More on that later
- Third-party libraries
  - Written by non-Python people
  - Can write programs using these libraries too

Using a Graphics Module/Library

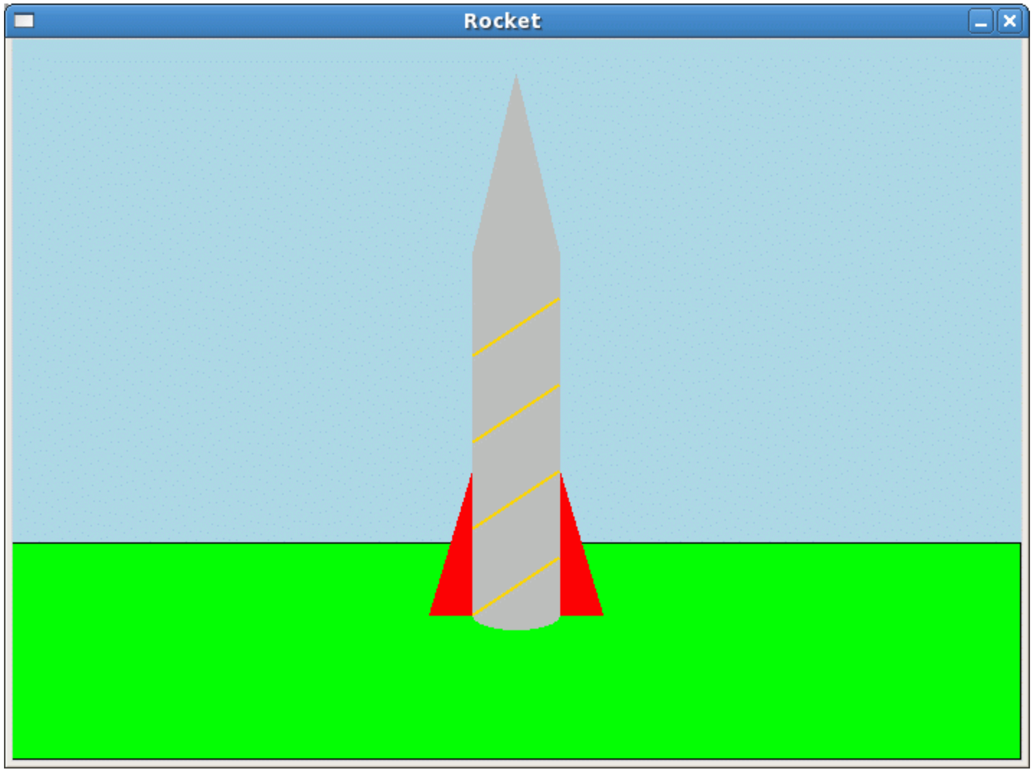
- Allows us to handle graphical input and output
  - Example output: Pictures
  - Example input: Mouse clicks
- Defines a collection of related graphics **classes**
- Not part of a standard Python distribution
  - Need to ***import*** from `graphics.py`
- Use the library to help us learn object-oriented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

USING A GRAPHICS MODULE

Using a Graphics Module/Library

- Handout describes how to use 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

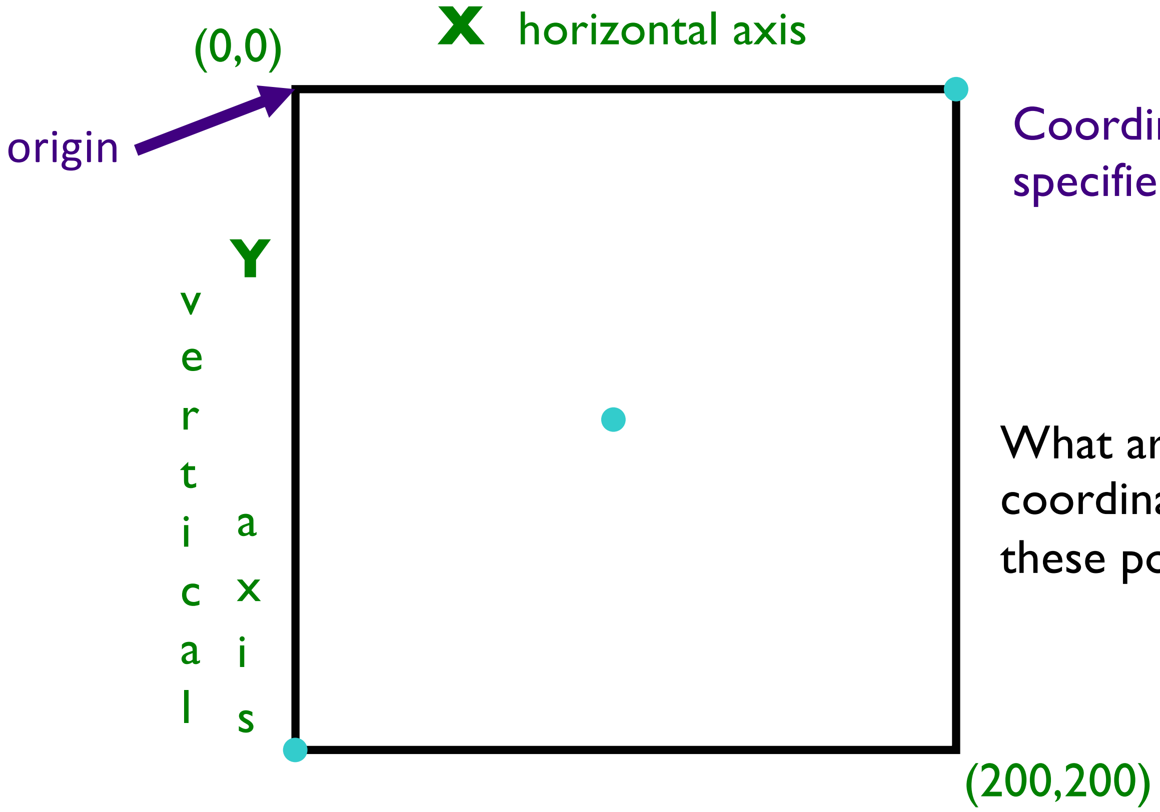


Using the Graphics Library

- 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

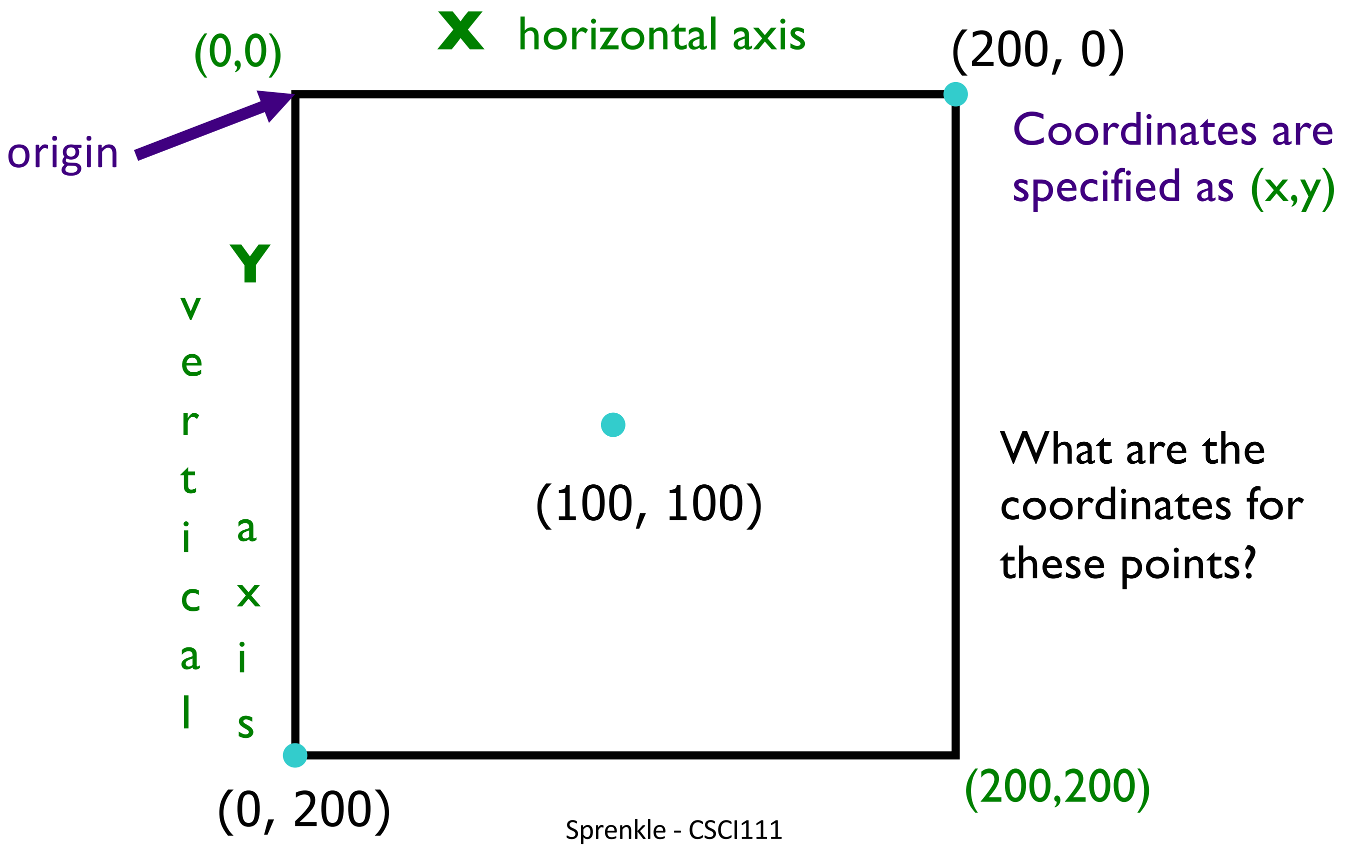
A GraphWin Object’s Canvas



Coordinates are specified as (x,y)

What are the coordinates for these points?

A GraphWin Object’s Canvas



Coordinates are specified as (x,y)

What are the coordinates for these points?

Using the API: **Constructors**

- To create an object of a certain type/**class**, use the **constructor** for that type/class
  - Syntax:

    ```
    objName = ClassName([parameters])
    ```

  - Semantics: create an object of type `ClassName` with the given parameters and save it in the variable `objName`
  - **`objname`** is as an ***instance** of the class `ClassName`*
- Example: To create a `GraphWin` object that's identified by `window`

  ```
  window = GraphWin("My Window",200,200)
  ```

The `GraphWin` API: Constructor

- All parameters to the ***constructor*** are optional
  - Marked by [ ]
- 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>)`	

Using the API: **Methods**

- To call a ***method*** on an object,
  - Syntax:
    ```
    objName.methodName([parameters])
    ```
  - Semantics: call `methodName` with the given parameters on the object identified by the name `objName`
  - Similar to calling *functions*
- Method names typically begin with lowercase letter
- Example: To change the background color of a `GraphWin` object named `window`

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

Using the API: Accessor Methods

- A method sometimes ***returns*** output, which you may want to save in a variable
  - Class's API should say if method returns output
    - Good rule of thumb: if you call a method that returns something, save it in a variable.
  - Referred to as an ***accessor* method**
- Example: if you want to know the *width* of a `GraphWin` object named `window`

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

The GraphWin API: Accessor Methods

Accessor methods for GraphWin

- Return some information about the GraphWin

Example methods:

- `<GraphWinObj>.getWidth()`
- `<GraphWinObj>.getHeight()`

The `GraphWin` API: Mutator Methods

- ***Mutator*** methods: methods that change or *mutate* an object/its state but don’t return anything
- Example: `<GraphWinObj>.setBackground(<color>)`
  - Colors are strings, such as "red" or "purple" (more later…)

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

  - Changes **`win`**'s state but does *not return* anything
    - Don’t save method call in a variable

Summary: General Categories of Methods

Accessor

- Returns information about the object
- Example use – save method call's output in a variable:
  `windowWidth = win.getWidth()`

Mutator

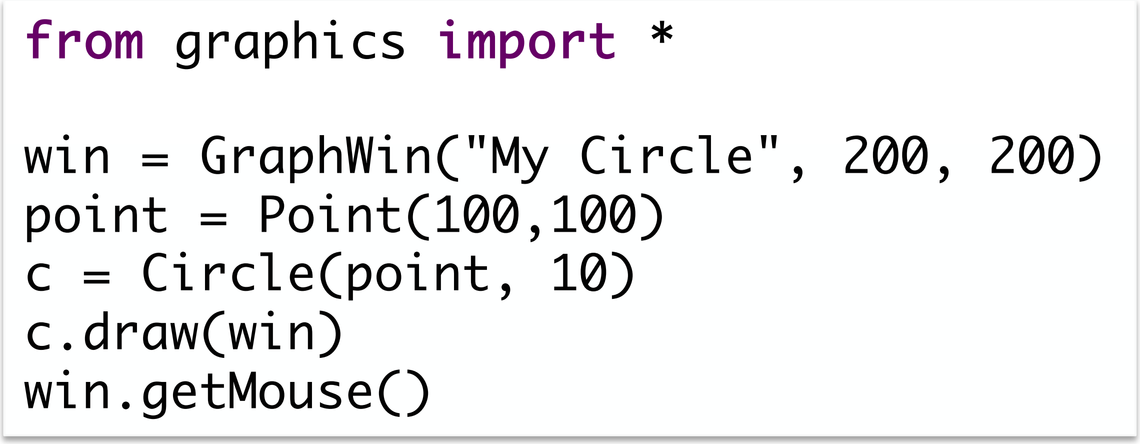
- Changes the state of the object
  - i.e., changes something about the object
- Example use:
  `win.setBackground("blue")`

Python Naming Conventions

- Object names begin with a lowercase letter
- Class names typically begin with a *capital* letter
- Method names begin with a lowercase letter

What Does This Code Do?

1. Identify examples of the OO terminology in this code: *class, objects, methods, constructors*
2. Describe the output from this code

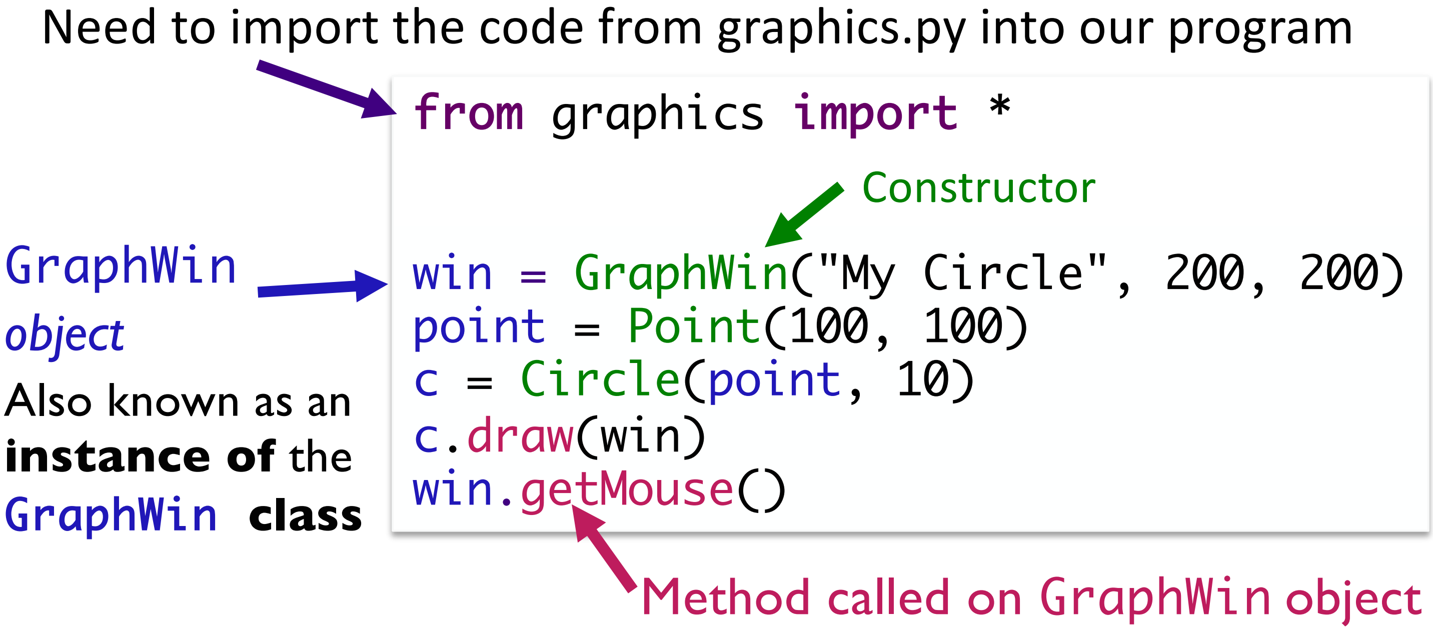


```
from graphics import *

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?



Need to import the code from graphics.py into our program

```
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

Typical OOP Programming Process:

1. Create an instance of an class
2. Call methods on that object

Looking Ahead

- Continue reading in the interactive textbook
- Pre Lab 2 due Tuesday before lab
    - You're going to make "something significant" using the graphics library