

# OBJECTS AND GRAPHICS

CSSE 120 – Rose-Hulman Institute of Technology

# Outline

---

- Eclipse
- The object of objects
- Graphics
- Creating and using objects
- Coordinate systems
- Interactive graphics
- In-class practice time

# Integrated Development Environments (IDEs)

Slide 3

- What are they?
- Why use one?
- Our IDE – Eclipse
  - ▣ Why we chose it
  - ▣ Basic concepts in Eclipse
    - Workspace, Workbench
    - Files, folders, projects
    - Views, editors, perspectives
    - <http://www.rose-hulman.edu/class/csse/resources/Eclipse/installation.htm>

The next slides address the listed points

# Verify that Eclipse Works for You

- Go to course Angel page:  
Resources → Course Resources section →  
Software Installation links →  
Configuring Python on Eclipse
- Scroll down to the section:  
**Configuring PyDev**
- Complete the instructions from there to the end of the document
- Get help as needed

# IDEs – What are they?

An IDE is an application that makes it easier to develop software.

They try to make it easy to:

The image shows a screenshot of the PyDev IDE interface. The main window displays a Python script named `test.py` with the following code:

```
import math

print "I am a newer Python module"
for i in range(10):
    print math.pow(i,2)
```

The interface includes several panels and callouts:

- Pydev Package Explorer:** Shows the project structure with folders like `test` and `src`, and files like `test.py`. Callout: "See the outline of the entire project".
- Main Editor:** The central area where code is written and edited. Callout: "Type and change code (editors)".
- Outline:** A panel on the right showing the structure of the code, with a callout: "See the outline of a chunk of code".
- Problems/Console:** A panel at the bottom showing the output of the code execution. Callout: "See output".
- Menu/Toolbar:** The top of the window contains menus (File, Edit, Source, Refactoring, Navigate, Search, Project, Run, Window, Help) and a toolbar with icons for various actions. Callout: "Compile, run, debug, document".

# IDEs – Why use one?

An IDE is an application that makes it easier to develop software.

They try to make it easy to:

The screenshot shows the Eclipse IDE interface with the Pydev plugin. The main window is titled 'Pydev - test.py - Eclipse SDK'. The menu bar includes File, Edit, Source, Refactoring, Navigate, Search, Project, Run, Window, and Help. The toolbar contains icons for file operations, running, and debugging. The Pydev Package Explorer on the left shows a project named 'test' with a sub-package 'src' containing 'test.py'. The central editor shows the following Python code:

```
import math

print "I am a newer Python module"
for i in range(10):
    print math.pow(i,2)
```

The Outline view on the right shows a tree structure with 'math' selected. The Console view at the bottom shows the output of the code:

```
<terminated> C:\Documents and Settings\...
16.0
25.0
36.0
49.0
64.0
81.0
```

Yellow callout boxes with arrows point to various parts of the IDE:

- 'Compile, run, debug, document' points to the toolbar.
- 'See the outline of a chunk of code' points to the Outline view.
- 'Type and change code (editors)' points to the code editor.
- 'See the outline of the entire project' points to the Pydev Package Explorer.
- 'See out' points to the Console view.

Eclipse is:

- **Powerful** -- everything here and more
- **Easy** to use
- **Free** and **open-source**
- An IDE for **any language**, not just Python
- **What our upper-class students told us to use!**

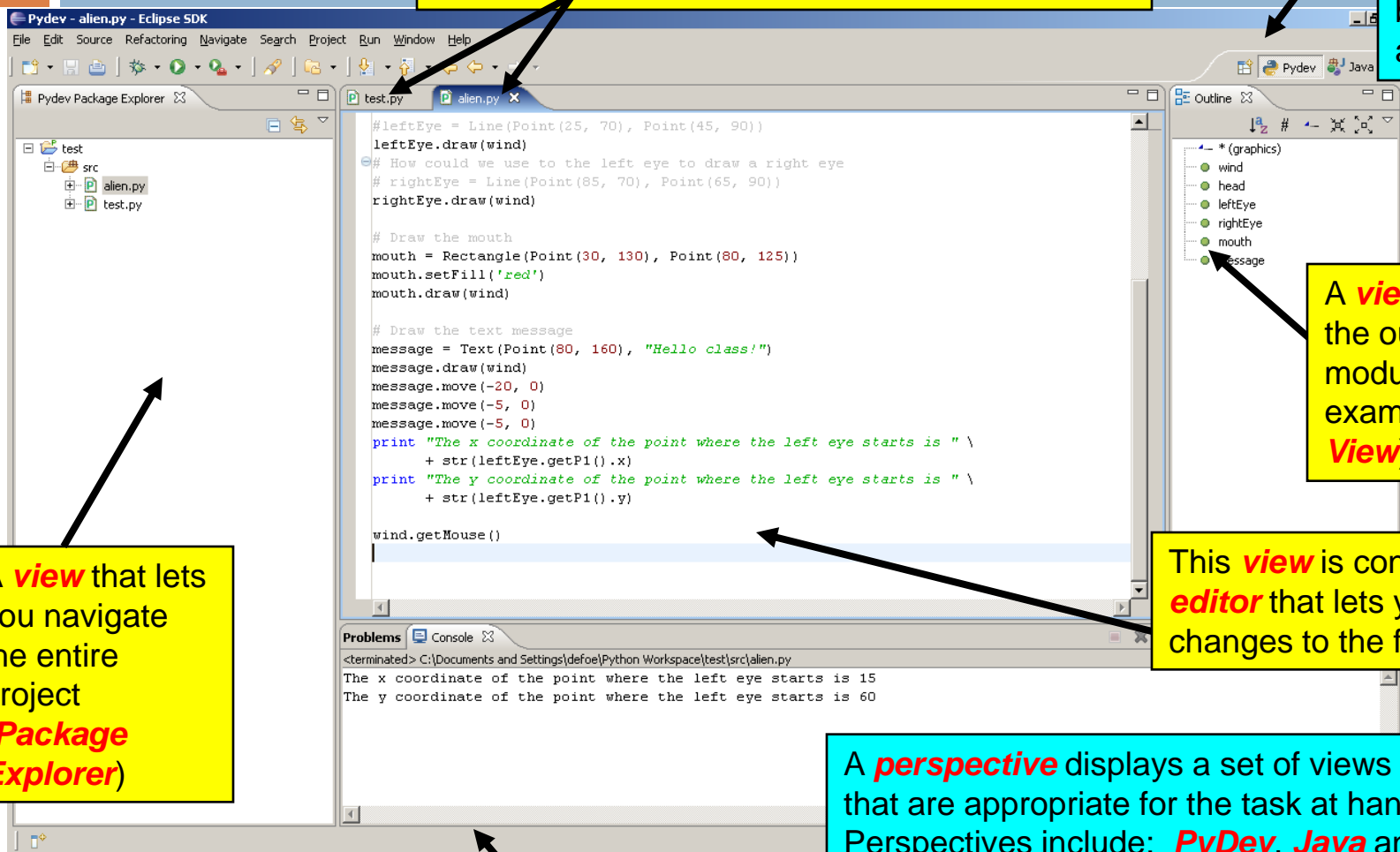
# Basic concepts in Eclipse

- **Workspace** – where your *projects* are stored on your computer
- **Project** – a collection of files, organized in folders, that includes:
  - **Source code** (the code that you write)
  - **Compiled code** (what your source code is translated into, for the machine to run)
  - **Design documents**
  - **Documentation**
  - **Tests**
    - And more that you will learn about over time
- **Workbench** – what we saw on the previous slide, that is, the tool in which you do your software development

# Views, editors, perspectives

Tabbed **views** of the source code of this project

This is the **PyDev perspective** but just a button click brings us to another



A **view** that lets you navigate the entire project (**Package Explorer**)

A **view** that shows the outline of the module being examined (**Outline View**)

This **view** is controlled by an **editor** that lets you make changes to the file

A **perspective** displays a set of views and editors that are appropriate for the task at hand. Perspectives include: **PyDev**, **Java** and lots more

Tabbed **views** (**Problems, Console**)

# Eclipse in a Nutshell

- **Workspace** – where your **projects** are stored on your computer
- **Project** – a collection of files, organized in folders, that includes:
  - **Source code** and **Compiled code** and more
- **Workbench** – the tool in which to work
  - It has **perspectives** which organize the **views** and **editors** that you use

# The object of objects

- Data types for strings and numbers are **passive**
  - ▣ Each represents set of values
    - Passive
  - ▣ Each has set of operations
    - Active
- Most modern computer programs built using Object-Oriented (OO) approach
  - ▣ Objects regarded as **active data type**
    - Know stuff
    - Can do stuff

# The object of objects

---

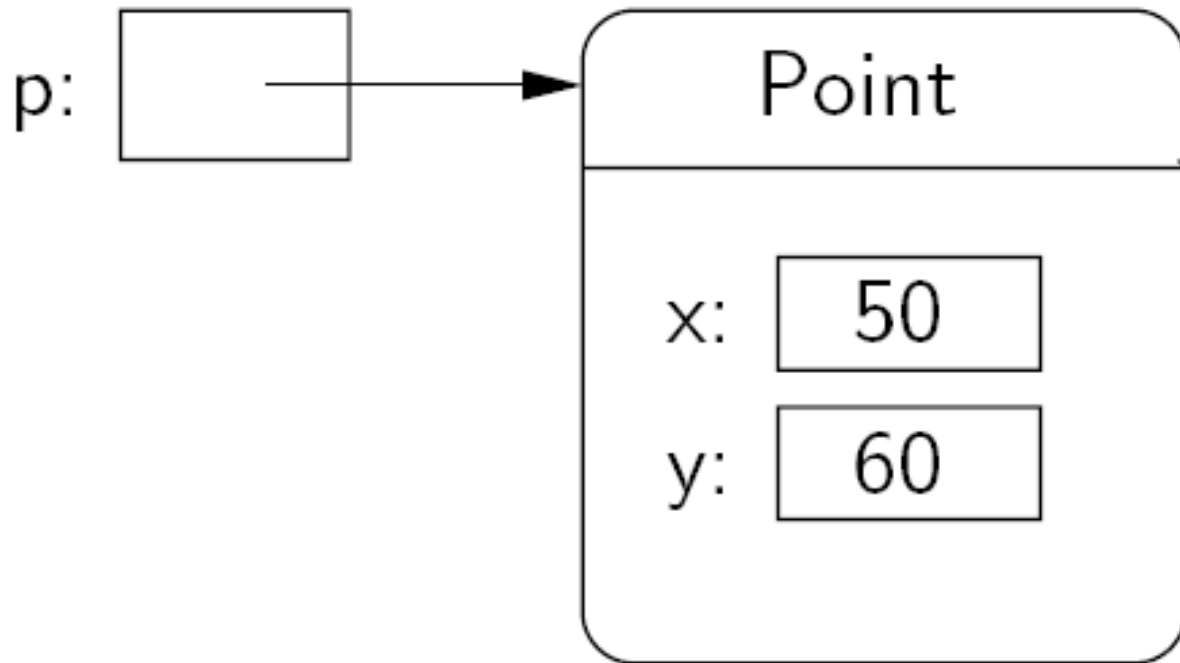
- Basic Idea of OO development
  - ▣ View complex system as interaction of simple objects
  - ▣ Example: the human body is a complex system

# How do objects interact?

- Objects interact by sending each other **messages**
  - ▣ Message: request for object to perform one of its operations
  - ▣ Example: the brain can ask the feet to walk
  - ▣ In Python, messages happen *via* **method calls**.
- `>>> win = zellegraphics.GraphWin()`
- `>>> p = Point(50, 60)`
- `>>> p.getX()` # accessor method
- `>>> p.getY()`
- `>>> p.draw(win)`

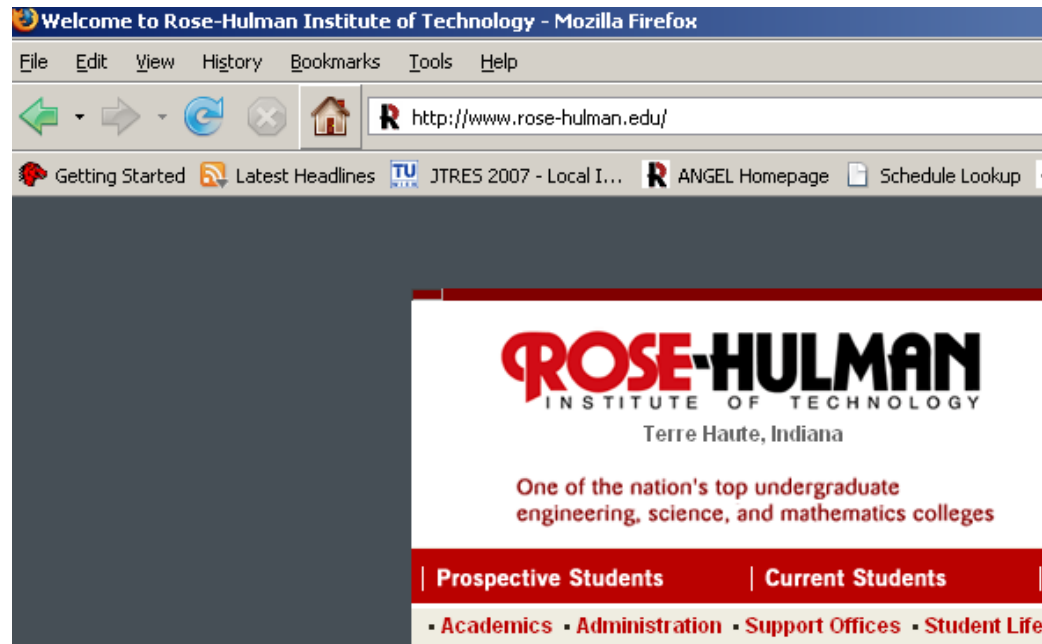
# How do objects interact? Point

```
p = Point(50, 60)
```



# Simple graphics programming

- Graphics is fun and provides a great vehicle for learning about objects
- Computer graphics: study of graphics programming
- Graphical User Interface (GUI)



# Simple graphics programming

- Must import graphics library before accessing it
  - `>>> import zellegraphics`
  - `>>> win = zellegraphics.GraphWin()`
- Another way to import graphics library
  - `>>> from zellegraphics import *`
  - `win = GraphWin()`
- What is the difference between these two approaches?

# Graphics window

- Collection of tiny points called *pixels*
  - ▣ Pixel: **picture element**
- Has a title, length, and width
- E.g. height = 200 pixels, width = 200 pixels
  - ▣ How many pixels?
- Computer monitor
  - ▣ # pixels wide
  - ▣ # pixels tall

# Using graphical objects

- Using different types of objects from the graphics library, draw the following **alien face** and message



# Paige clearly isn't working on homework for CSSE1 20

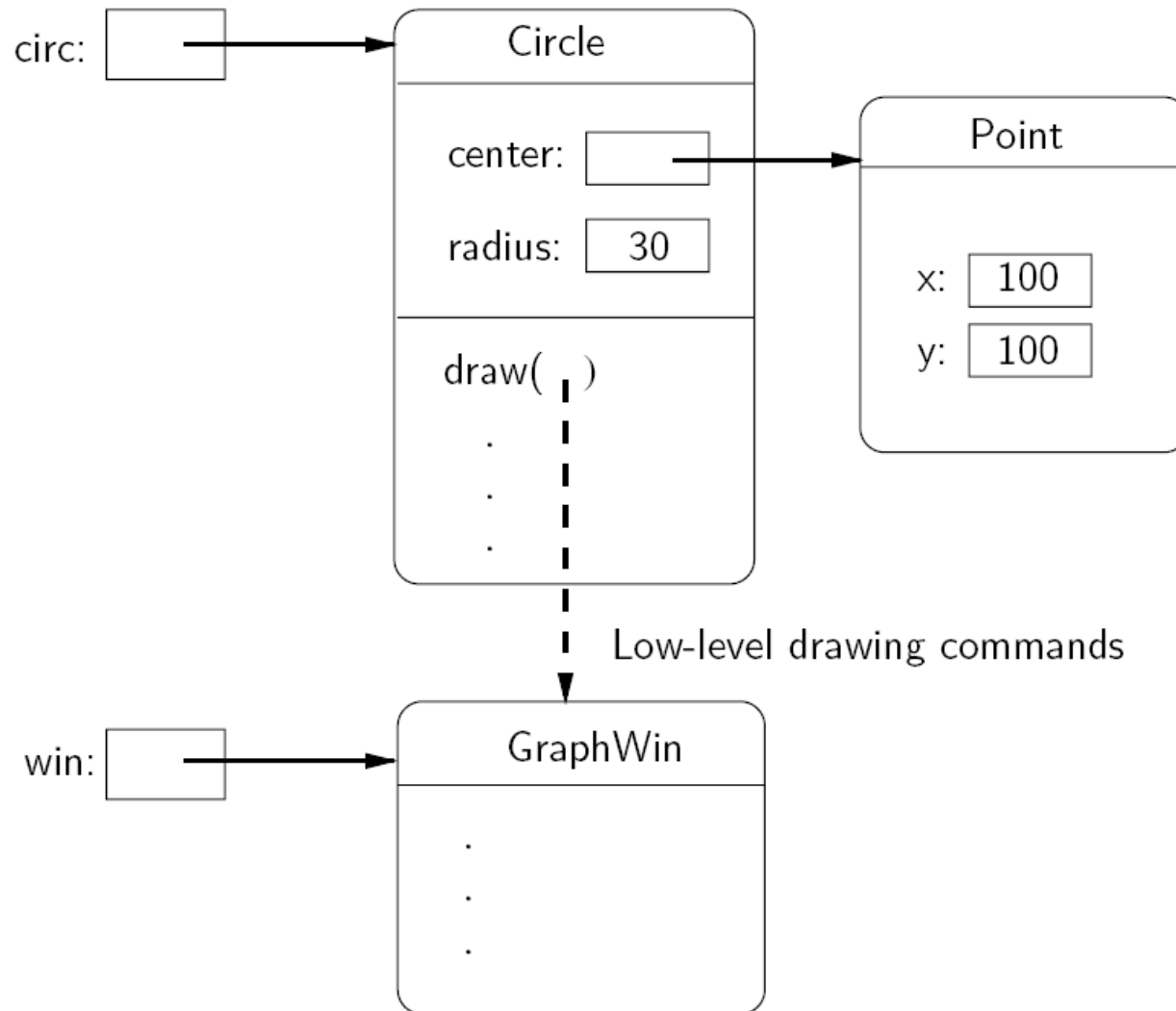


- Preview of tonight's homework:
  1. Read in and draw cool plots from the points in the files you generated in HW5
  2. Create a slideshow viewer for photos

# Classes and objects

- Different types of objects
  - ▣ Point, Line, Rectangle, Oval, Text
  - ▣ These are examples of *classes*
- Different objects
  - ▣ head, leftEye, rightEye, mouth, message
  - ▣ Each is an *instance* of a class
  - ▣ Created using a *constructor*
  - ▣ Objects have *instance variables*
  - ▣ Objects use *methods* to operate on instance variables

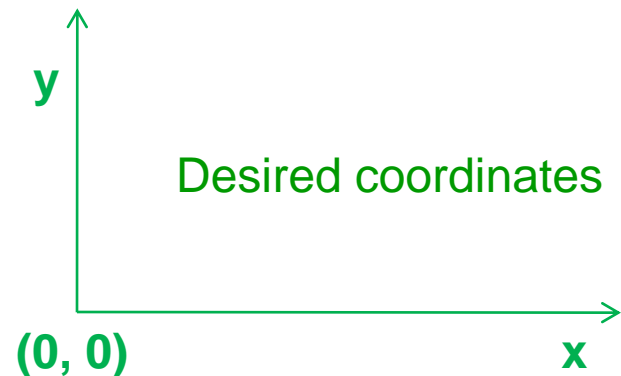
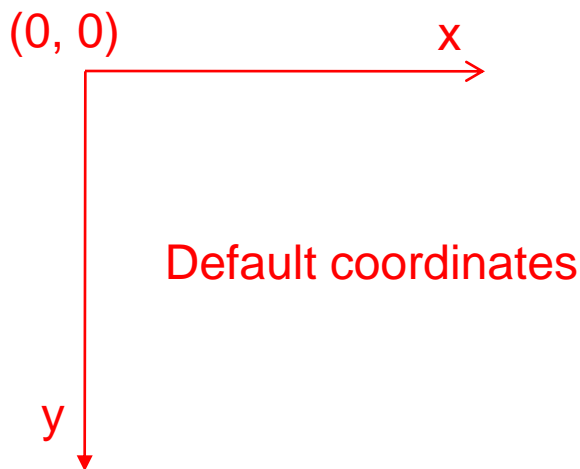
# Object interaction to draw a circle





# Coordinate systems

- An important use of graphics is to represent **data** visually
  - ▣ Example: a bar chart
- We really want  $(0,0)$  to be in the lower-left corner



# Desired coordinate system



- `setCoords(x1, y1, x2, y2)` method from `GraphWin` class
  - ▣ Sets the coordinates to run from  $(x_1, y_1)$  in the lower-left corner to  $(x_2, y_2)$  in the upper-right corner.

# Interactive graphics

- *GUI*—Graphical User Interface
  - Accepts input
    - Keyboard, mouse clicks, menu, text box
  - Displays output
    - In graphical format
    - On-the-fly
- Developed using *Event-Driven Programming*
  - Program draws interface elements (*widgets*) and waits
  - Program responds when user does something

# getMouse

- `win.getMouse()`
  - ▣ Causes the program to halt, waiting for the user to click the mouse somewhere in the window
  - ▣ To find out where it was clicked, assign it to a variable:
    - `p = win.getMouse()`

# Mouse Event Exercise

Together, let's solve the following problem:

Create a program, `clickme.py`, with a window labeled "Click Me!" that displays the message *You clicked (x, y)* the first 10 times the user clicks in the window.

The program also draws a red-filled circle, with blue outline, for each of these first 10 clicks.

The program closes the window on the 11<sup>th</sup> click