Game Design

11/13/12: Python

Rensselaer

Python

Rensselaer

interpreted

Rensselaer

dynamically typed

Rensselaer

whitespace matters

Rensselaer

```
#code will look like this
#comments start with
#a pound sign
```

Rensselaer

Python

- Programs are composed of *modules*
- Modules contain *statements*
- Statements contain *expressions*
- Expressions contain *objects* and *operators*
  - create and manipulate data

Rensselaer

Numbers

- Work pretty much the way you think they should

```
>>>1+1
2
>>>5/2
2
>>>5%2
1
```

Rensselaer

Strings

- Ordered sequences of characters (e.g. text)
- Surround with double or single quotes

```
q1 = “Hello there, I’m Dr. Marc”

q2 = ‘Quoth the raven: “Nevermore”’
```

Rensselaer

String Formatting

- uses the % operator, works a lot like printf()

```
print "Joe says %s"%q1
Joe says Hello there, I'm Dr. Marc
```

- `%s` for string, `%d` for integer, `%f` for floating-point number
- Could also use `print "Joe says "+ q1`
  - but that's slower

Rensselaer

String Methods

- `q1.split(' ')`
  - returns ['Hello', 'there', "I'm", 'Dr.', 'Marc']
- `q1.split('e')`
  - returns ['H', 'llo th', 'r', " I'm Dr. Marc"]
- `a_list = ["these", "are", "words"]`
- `' '.join(a_list)`
  - returns 'these are words'

Rensselaer

Lists and Dictionaries

- Collections of objects - main workhorse in most Python programs
- Can contain and be contained by anything

Rensselaer

Lists

- created using square brackets
- thislist = [‘first item’, 23, “b”, [255, 255], 10]
- accessed by offset, starting with 0
  >>>thislist[2]
  ‘b’

Rensselaer

List Methods

length:	`len(thelist)`	returns 5
sort:	`thelist.sort()`	no return value
add:	`thelist.append(37)`	
return & remove:	`thelist.pop()`	returns 37
	`thelist.pop(1)`	returns 5
insert:	`thelist.insert(2, 'z')`	
remove:	`thelist.remove('e')`	
	`del thelist[0]`	
concatenation:	`thelist + [0]`	returns ['z',9,'p',0]
finding:	`9 in thelist`	returns True

Rensselaer

List Slicing

`thelist[1:3]`	returns [3, ‘p’]
`thelist[2:]`	returns [‘p’, 9, ‘e’]
`thelist[:2]`	returns [5, 3]
`thelist[2:-1]`	returns [‘p’, 9]

Rensselaer

List Comprehension

```
>>>[x*5 for x in range(5)]
[0, 5, 10, 15, 20]

>>>[x for x in range(5) if x%2 == 0]
[0, 2, 4]
```

Rensselaer

Dictionaries

- Also a collection of objects, but unordered
- Accessing by offset is nonsensical
- Items are stored and retrieved by arbitrary values called *keys*
- Finding items in a dictionary is *really fast*

Rensselaer

Dictionaries

- Created using curly braces

```
thisdict = {'spam':42, 23:"twenty-three"}
```

- Accessing:

```
>>>thisdict['spam']
42
```

Rensselaer

Tuples

- Like lists, but *immutable* - can't be changed once created
- Created using parenthesis
- thistuple = (25, "hello", 'Q')

Rensselaer

Conditional Statements

```
if test:
  #do stuff if test is true
elif test 2:
  #do stuff if test2 is true
else:
  #do stuff if both tests are false
```

Rensselaer

Loops

```
while test:
  #keep doing stuff until
  #test is false


for x in aSequence:
  #do stuff for each member of aSequence
  #for example, each item in a list, each
  #character in a string, etc.
```

Rensselaer

Loops

```
for x in range(10):
    #do stuff 10 times (0 through 9)

for x in range(5,10):
    #do stuff 5 times (5 through 9)

l = [25, 19, “samIam”, 2, ‘s’]

for snarf in l:

    print snarf
```

Rensselaer

Functions

```
def myFunc(param1, param2):
        """By putting this initial sentence in
        triple quotes, you can access it
        by calling myFunc.__doc___"""
        #indented code block goes here
        spam = param1 + param2
        return spam
```

Rensselaer

Files

open:

```
thisfile = open("datadirectory/file.txt")
```

(defaults to read-only)

accessing:

`thisfile.read()`	reads entire file into one string
`thisfile.readline()`	reads one line of a file
`thisfile.readlines()`	reads entire file into a list of strings, one per line

`for eachline in thisfile:` steps through lines in a file

Rensselaer

Classes

- Code blocks that define new objects
- object.attribute
- All functions defined within a class are called methods, take a special self argument (used to differentiate instances)

Rensselaer

Classes

```
class Eggs(Offspring):
            def __init__(self):
              Offspring.__init__(self)
              #initialization code goes here
              self.cookingStyle = 'scrambled'
            def anotherFunction(self, argument):
              if argument == "just contradiction":
                 return False
              else:
                 return True

theseEggsInMyProgram = Eggs()
```

Rensselaer

Style

- Readability counts
- 4 spaces, never tabs
- ClassName, function_name
- self._private_attr

Rensselaer

More Information

- http://docs.python.org
- http://diveintopython.org

Rensselaer

Game Architecture

Rensselaer

Render Loop

- Game Logic
  - Player update
    - Sense player input
    - Computer restrictions
    - Update player state
  - World update
    - Passive elements
      - Pre-select active zone for engine use
    - Logic-based elements
      - Sort according to relevance
      - Execute control mechanism
      - Update state
    - AI based elements
      - Sort according to relevance
      - Sense internal state and goals
      - Sense restrictions
      - Decision engine
      - Update world
- End

- Presentation
  - World presentation
    - Select visible subset
      - Clip
      - Cull
      - Occlude
    - Select Resolution
    - Pack geometry
    - Render world geometry
    - Select audible sound sources
    - Pack audio data
    - Send to audio hardware
  - NPC presentation
    - Select visible subset
    - Animate
    - Pack
    - Render NPC data
  - Player presentation
    - Animate
    - Pack
    - Render
- End

Rensselaer

- import statements
- class definitions
- create initial objects
- main while loop
  - tick clock
  - process events (keypresses, etc.)
  - update objects (collision detection, etc.)
  - draw objects
  - display screen

Rensselaer

Pygame

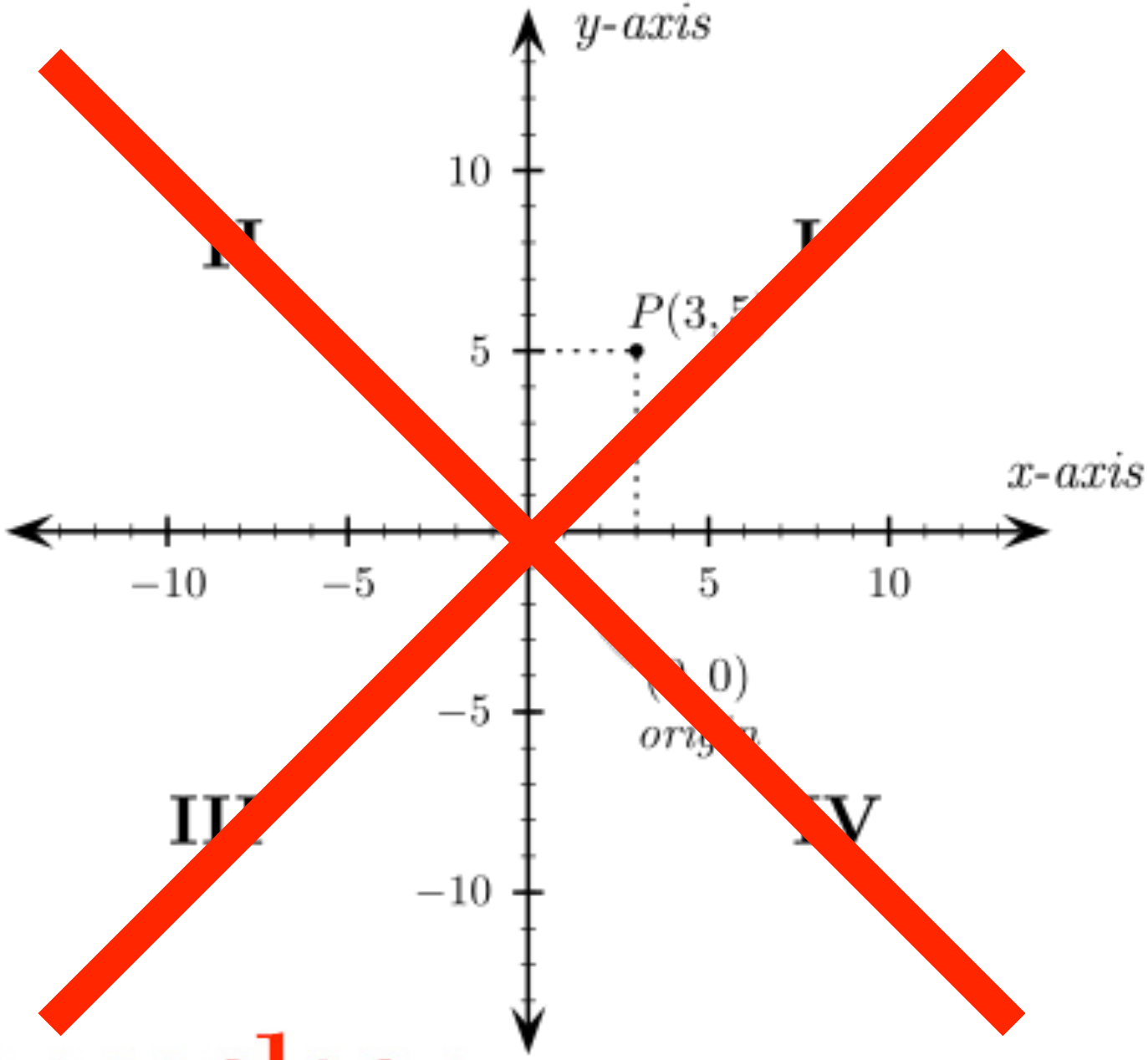
Rensselaer

Initialization

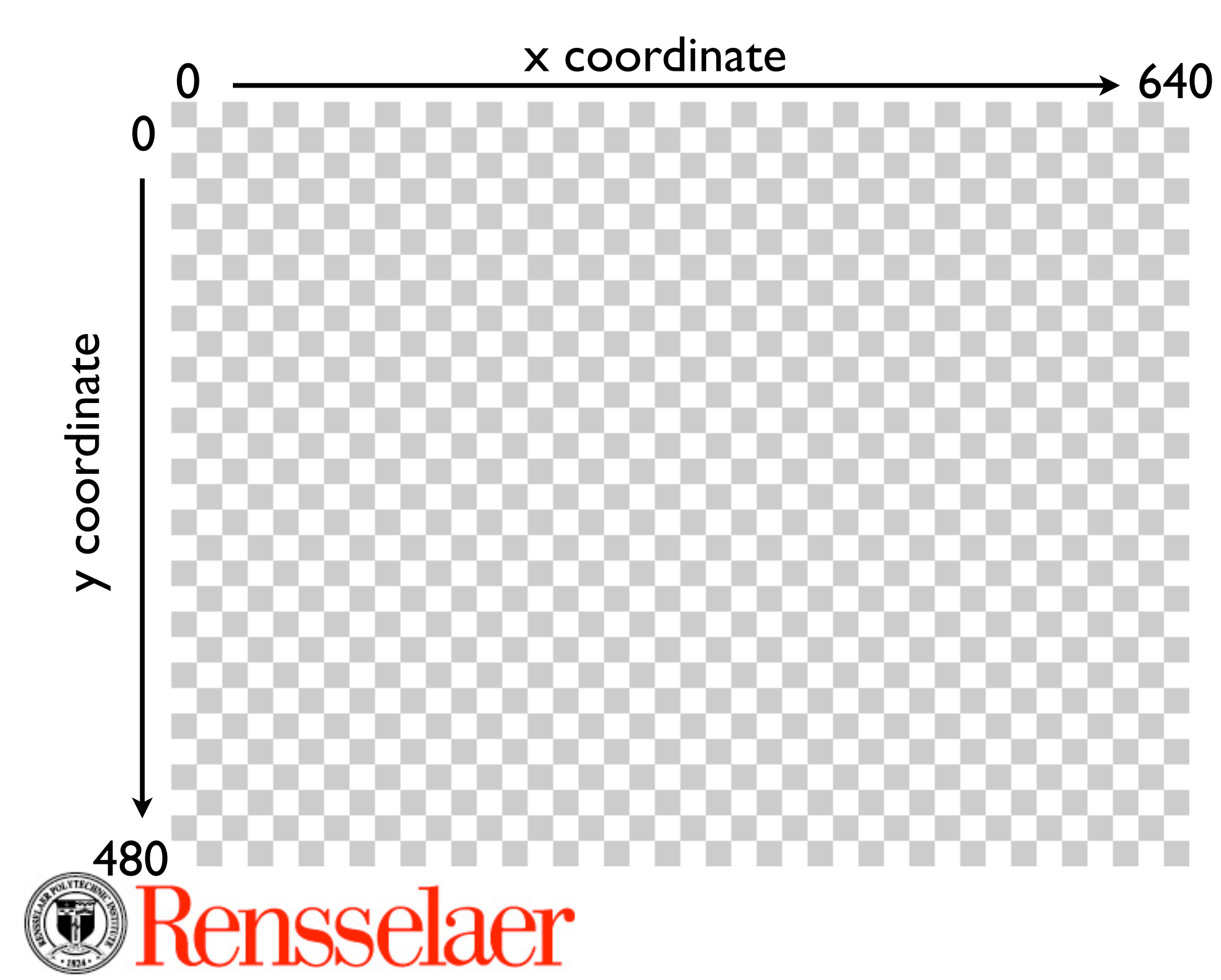
```
import pygame

pygame.init()

screen = pygame.display.set_mode(size)
```

- size is a tuple, common sizes are (640, 480) and (800, 600)

Rensselaer



Rensselaer



Rensselaer

Surfaces

- an image, stored in memory
- surf = pygame.Surface(size)
- surf = pygame.image.load(filename)

Rensselaer

Blit

- From "BitBLT", for "Bit Block Transfer"
- *copies* pixels from one surface to another

```
surface.blit(sourceSurface,
destPos, [sourcePos])
```

Rensselaer