

Objectives

- A little more arithmetic
- Introduction to Object-Oriented Programming
- Introduction to APIs

Review

- What did you learn from yesterday's lab?
 - What are your takeaways?
 - How will you prepare for next week's lab?
- What is our development process?
 - What are good test cases?

Lab Retrospective

- Learning how to solve problems
 - Every week: new problems, new techniques to solve problems
- Note how I am explicit in directions/reminders early
 - Then stop reminding because you should know the process by then

Jan 16, 2019

Sprenkle - CSCI111

3

Two Division Operators

/ Float Division

- Result is a **float**
- Examples:
 - $6/3 \rightarrow 2.0$
 - $10/3 \rightarrow 3.3333333333333335$
 - $3.0/6.0 \rightarrow 0.5$
 - $19/10 \rightarrow 1.9$

// Integer Division

- Result is an **int**
- Examples:
 - $6//3 \rightarrow 2$
 - $10//3 \rightarrow 3$
 - $3.0//6.0 \rightarrow 0.0$
 - $19//10 \rightarrow 1$

Integer division is the default division used in most programming languages

Jan 16, 2019

Sprenkle - CSCI111

4

Division Practice

- $a = 12 // 4$
- $12 // 4 * 5.0$
- $b = 6/12$
- $6.0//12 * 5.0$
- $z = a / b$

Jan 16, 2019

Sprenkle - CSCI111

5

More on Arithmetic Operations

Symbol	Meaning	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder ("mod")	Left
**	Exponentiation (power)	Right

Precedence rules: P E - DM% AS
 ↖
 negation

Jan 16, 2019

Sprenkle - CSCI111

6

More on Arithmetic Operations

Symbol	Meaning	Associativity
+	Addition	Left
-	Subtraction	Left
*	Multiplication	Left
/	Division	Left
%	Remainder ("mod")	Left
**	Exponentiation (power)	

Precedence rules: P E - DM% AS

negation

Jan 16, 2019

Sprenkle - CSCI111

Associativity matters when you have the same operation multiple times. It tells you where you should start computing.

Python Math Practice

```
5 + 3 * 2
2 * 3 ** 2
-3 ** 2
2 ** 3 ** 3
```

How should we verify our answers?

Jan 16, 2019

Sprenkle - CSCI111

8

Modulo Operator: %

- Modular Arithmetic: Remainder from division
 - $x \% y$ means the remainder of $x//y$
 - Read as “x mod y”
- Example: $6 \% 4$
 - Read as “six mod four”
 - $6//4$ is 1 with a remainder of 2, so $6\%4$ evaluates to 2
- Works only with integers
 - Typically just positive numbers
- Precedence rules: P E - DM% AS

Jan 16, 2019

Sprenkle - CSCI111

9

Modulo Practice

- $7 \% 2$
- $3 \% 6$
- $6 \% 2$
- $7 \% 14$
- $14 \% 7$
- $6 \% 0$

Jan 16, 2019

Sprenkle - CSCI111

10

Brainstorm

- What useful thing does `% 10` do?
 - `3 % 10 =`
 - `51 % 10 =`
 - `40 % 10 =`
 - `678 % 10 =`
 - `12543 % 10 =`
- What useful thing does `// 10` do (integer division)?
 - `3 // 10 =`
 - `51 // 10 =`
 - `40 // 10 =`
 - `678 // 10 =`
 - `12543 // 10 =`
- What useful thing does `% 2` do?

Jan 16, 2019

Sprenkle - CSCI111

11

Trick: Arithmetic Shorthands

- Called **extended assignment operators**
- Increment Operator
 - `x = x + 1` can be written as `x += 1`
- Decrement Operator
 - `x = x - 1` can be written as `x -= 1`
- Shorthands are similar for `*`, `/`, `//` :
 - `amount *= 1.055`
 - `x /= 2`

Jan 16, 2019

Sprenkle - CSCI111

12

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

Jan 16, 2019

Sprenkle - CSCI111

13

OBJECT-ORIENTED PROGRAMMING

Jan 16, 2019

Sprenkle - CSCI111

14

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

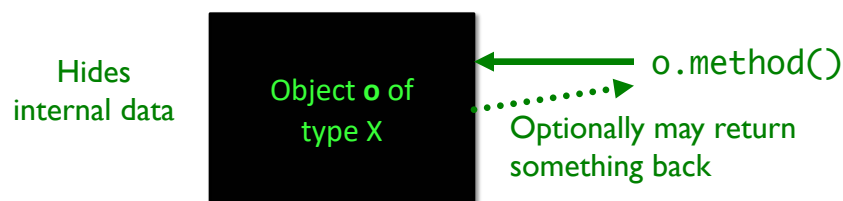
Jan 16, 2019

Sprenkle - CSCI111

15

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



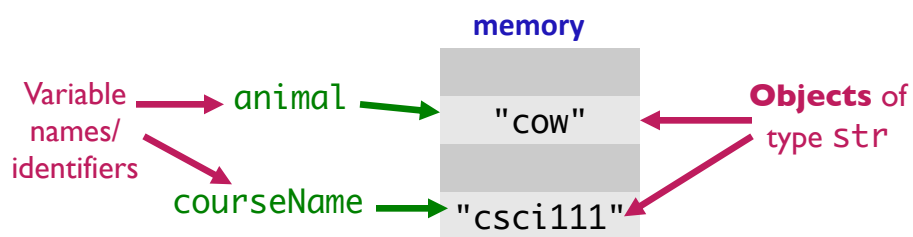
Jan 16, 2019

Sprenkle - CSCI111

16

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"`



Jan 16, 2019

Sprenkle - CSCI111

17

Example of OO Programming Abstraction

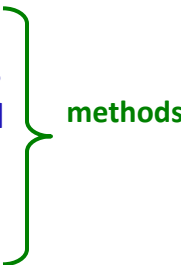
- Think of a smart phone— It's an **object**
- What can you do to a phone?

Jan 16, 2019

Sprenkle - CSCI111

18

Example of OO Programming Abstraction

- Think of a phone— it's an **object**
- What can you do to a phone?
 - 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**

Jan 16, 2019

Sprenkle - CSCI111

19

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
 - You can call the above methods on any object of type SmartPhone

Jan 16, 2019

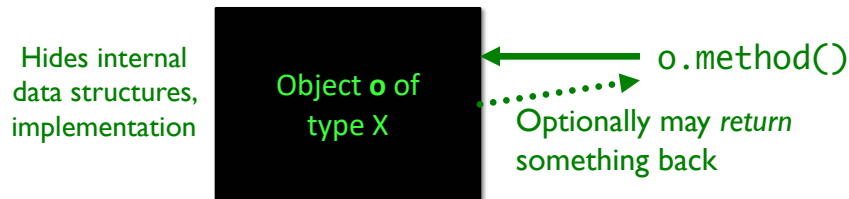
Sprenkle - CSCI111

20

Object-Oriented Programming

- Objects combine **data and methods** together

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



Use an **Application Programming Interface (API)** to interact with a set of classes.

Jan 16, 2019

Sprenkle - CSCI111

21

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

Jan 16, 2019

Sprenkle - CSCI111

22

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

Jan 16, 2019

Sprenkle - CSCI111

23

USING A GRAPHICS MODULE

Jan 16, 2019

Sprenkle - CSCI111

24

Using a Graphics Module/Library

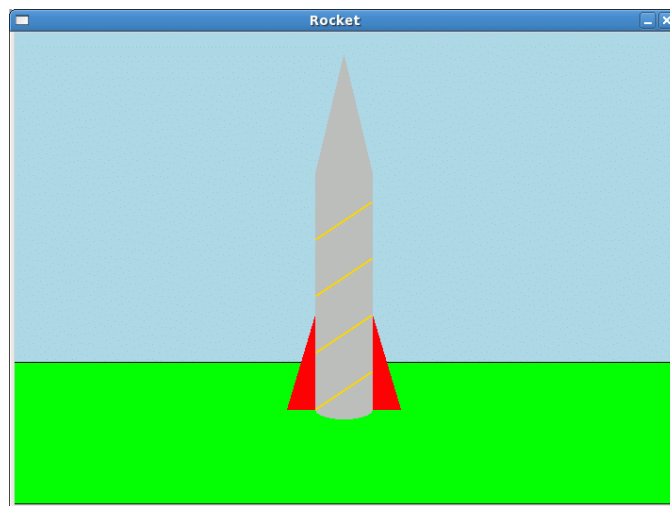
- Handout lists the various classes
 - **Constructor** is in bold
 - Creates an object of that type
 - For each class, lists *some* of their methods and parameters
 - Drawn objects have some common methods
 - Listed at end of handout
- Known as an **API**
 - **Application Programming Interface**

Jan 16, 2019

Sprenkle - CSCI111

25

Example of Output



Jan 16, 2019

Sprenkle - CSCI111

26

Using the Graphics Library

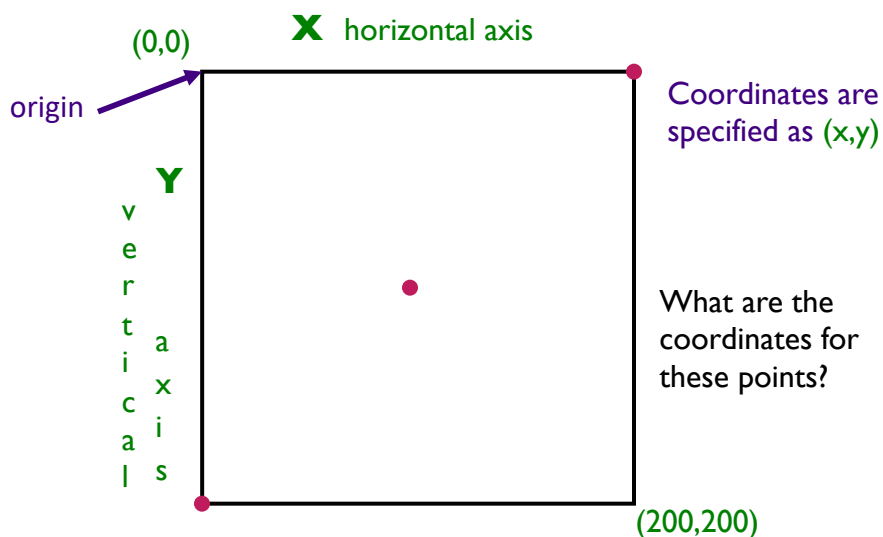
- In general, graphics are drawn on a canvas
 - A canvas is a 2-dimensional grid of pixels
- For our Graphics library, our canvas is a *window*
 - Specifically an **instance** of the **GraphWin** class
 - By default, a GraphWin object is 200x200 pixels

Jan 16, 2019

Sprenkle - CSCI111

27

A GraphWin Object's Canvas

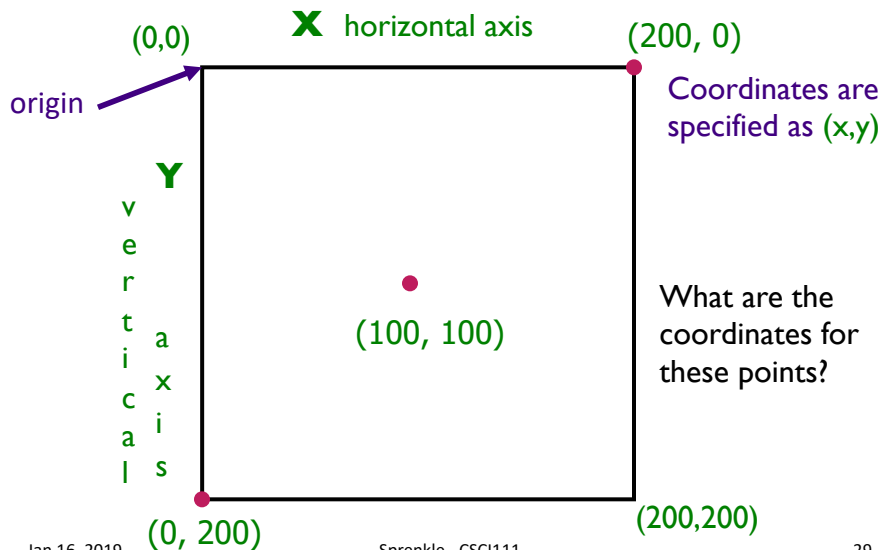


Jan 16, 2019

Sprenkle - CSCI111

28

A GraphWin Object's Canvas



Jan 16, 2019

Sprenkle - CSCI111

29

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)
```

Jan 16, 2019

Sprenkle - CSCI111

30

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>)	

Jan 16, 2019

Sprenkle - CSCI111

31

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")
```

Jan 16, 2019

Sprenkle - CSCI111

32

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()
```

Jan 16, 2019

Sprenkle - CSCI111

33

The GraphWin API

- **Accessor** methods for GraphWin
 - Return some information about the GraphWin
- Example methods:
 - <GraphWinObj>.getWidth()
 - <GraphWinObj>.getHeight()

Jan 16, 2019

Sprenkle - CSCI111

34

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**

Jan 16, 2019

Sprenkle - CSCI111

35

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()`

Jan 16, 2019

Sprenkle - CSCI111

36

What Does This Code Do?

- Use OO terminology previously defined

```
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

Jan 16, 2019

Sprenkle - CSCI111

37

What Does This Code Do?

- Use OO terminology previously defined

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()
```

GraphWin object
Also known as an instance of the GraphWin class

Constructor

Method called on GraphWin object

Note: Class names start with capital letters,
Method names start with lowercase letters

Jan 16, 2019

Benefits of Object-Oriented Programming

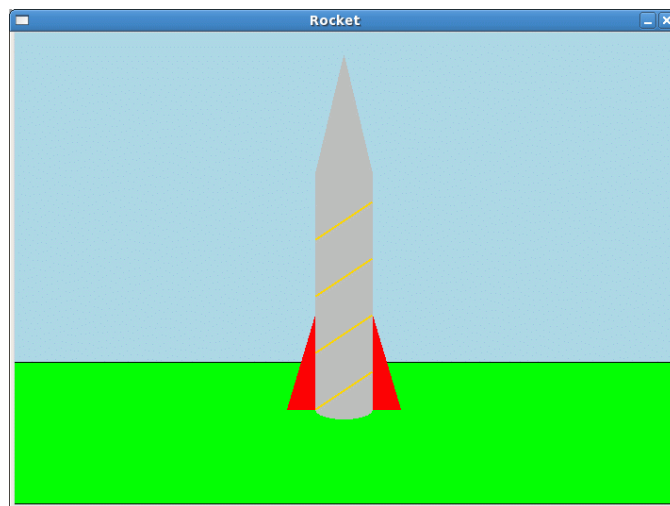
- **Abstraction**
 - Hides details of underlying implementation
 - Easier to change implementation
- Easy reuse of code
 - Can import the library in multiple files
- Collects related data/methods together
 - Easier to reason about data
- Less code in main program
 - Our program code is relatively simple

Jan 16, 2019

Sprenkle - CSCI111

39

What objects make up this image?



Jan 16, 2019

Sprenkle - CSCI111

40

Looking Ahead

- Lab 1 due Friday
- Broader Issue write up due Friday