Classes

CS106AP Lecture 19
Today’s questions

How do we animate our graphics programs?

How and why would we define our own data types while programming?
Today’s topics

1. Review
2. Event loops
3. Classes
4. What’s next?
Review
campy
New graphics library: campy

- Python version of Stanford’s ACM Graphics Libraries
- Allows us to easily create graphical objects and add them to a canvas
- Built on top of Tkinter
New graphics library: campy

- Python version of Stanford’s ACM Graphics Libraries
- Allows us to easily create graphical objects and add them to a canvas
- Built on top of Tkinter

*Do not use campy on Assignment 4!*
GWInow
GWindow

- The GWindow stores all the information about the objects in the window!
- GObjects are added in order (most recently added are on top)

```python
window = GWindow(height, width, title)
window.add(obj, x, y)  # x, y are optional
window.remove(obj)
window.clear()  # remove all objects
window.get_object_at(x, y)  # returns topmost object
```
GObjects
Types of GObjects

- GRect
- GOval
- GLine
- GLLabel
Getting information from (any) GObject

obj.width

obj.height

obj.x

obj.y

obj.filled

obj.fill_color
def get_rect():
    rect = GRect(width=100, height=50, x=25, y=25)
    rect.filled = True
    rect.fill_color = 'blue'
    return rect
def get_oval():
    oval = GOval(width=100, height=50, x=0, y=0)
    oval.filled = True
    oval.fill_color = 'green'
    return oval
GLine

\[
\text{line} = \text{GLine}(x_0, y_0, x_1, y_1)
\]

# similar to tk’s create_line() function!
GLabel

label = GLabel(text, x, y)
# similar to tk’s create_text() function!
label = GLabel('hi', 0, 0)
Randomness
The `random` module

- `random` is a useful Python module

```python
import random
COLORS = ['red', 'orange', 'yellow', 'green', 'blue']
color = random.choice(COLORS)
```
The `random` module

- `random` is a useful Python module

```python
import random
COLORS = ['red', 'orange', 'yellow', 'green', 'blue']
color = random.choice(COLORS)
```

randomly selects from a list
The `random` module

- Generate a random float $0 \leq x < 1$
  
  \[ \text{random.random()} \]

- Generate a random int $a \leq x \leq b$
  
  \[ \text{random.randint(a, b)} \]

- Randomly select from a list
  
  \[ \text{random.choice(lst)} \]

note that it is inclusive!
Random bubbles:

draw_bubbles()

[demo]
How do we animate our graphics programs?
Random bubble: 

animate_bubble_pop() 

[demo]
Use an animation loop

```python
while True:
    if stop_condition:
        break

    # 'Animate' object
    pause(timestep)
```

Small movements help make the animation appear continuous!
Use an animation loop

```python
while True:
    if stop_condition:
        break

    # 'Animate' object
    pause(timestep)
```

Small movements help make the animation appear continuous!
Use an animation loop

```python
while True:
    if stop_condition:
        break

    # 'Animate' object
    pause(timestep)
```

How long to wait in milliseconds
Random bubble:

animate_bubble_pop()
Random bubbles:

animate_many_bubbles()
Random bubbles: `animate_many_bubbles()`

How could we keep track of multiple bubbles?
Module imports
(a review)
How can I leverage Campy in my programs?

- Import modules from the campy library
- New import syntax:

```python
from campy.graphics.gobjects import GOval
```

path to specific module

feature(s) we want to use
Using modules - two ways

1. Import the module
   ```python
   import module
   ```
2. Use the predefined features!
   ```python
   module.function()
   ```

   Method A

1. Import specific feature from module
   ```python
   from module import function
   ```
2. Use the feature!
   ```python
   function()
   ```

   Method B
Using modules - two ways

1. Import the module

   ```python
   import campy.graphics.gobjects
   ```

2. Use the predefined features!

   ```python
   campy.graphics.gobjects.GOval(width, height)
   ```

   *Method A*
Using modules - two ways

1. Import the feature from module

   ```python
   from campy.graphics.gobjects import GOval, GRect
   ```

2. Use the feature!

   ```python
   GOval(width, height)
   ```

   *Method B*
Using modules - two ways

1. Import the module
   ```python
   import module
   ```
2. Use the predefined features!
   ```python
   module.function()
   ```

Method A

1. Import specific feature from module
   ```python
   from module import function
   ```
2. Use the feature!
   ```python
   function()
   ```

Method B

more concise, especially if you're only importing 1-2 things
What are we importing from the module?

```
from random import randint

from campy.graphics.gobjects import GOval, GRect
```
What are we importing from the module?

```
from random import randint
```

```
from campy.graphics.gobjects import GOval, GRect
```

A function!
What are we importing from the module?

```python
from random import randint
from campy.graphics.gobjects import GOval, GRect
```

Object types defined in the module
What are we importing from the module?

from random import randint

from campy.graphics.gobjects import GOval, GRect

i.e. “classes”
How and why would we define our own data types?
How and why would we define our own data types?
How and why would we define our own data types?

Classes!
A Python class defines a new data type for our programs to use.
A Python class defines a new data type for our programs to use.

ints, strings, booleans, lists, floats, dictionaries, etc. are all built-in Python data types
Definition

class
A Python class defines a new data type for our programs to use.

Classes allow us to define our own data types!
Classes we’ve already used that others have defined

- SimpleImage
- Pixel
- GObjects! (GRect, GOval, GLabel, etc.)
What is a class?

- A blueprint for a new type of Python object!
What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.
What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

**Definition**

**instance**

When we create an object that is our new type, we call this creating an instance of our class.
What is a class?

- A blueprint for a new type of Python object!
  - The blueprint describes a general structure, and we can create specific instances of our class using this structure.

```python
image = SimpleImage(width, height)
```

Creates an instance of the SimpleImage class (i.e. an object of the type SimpleImage)
What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

```python
oval = GOval(width=50, height=50)
```

*Creates an **instance** of the GOval class (i.e. an object of the type GOval)*
What is a class?

- A blueprint for a new type of Python object!
  - The blueprint describes a general structure, and we can create specific instances of our class using this structure.

```
oval = GOval(width=50, height=50)
```

Note that each instance has its own attributes (i.e. each GOval object has its own width, height, x, y, color)
What is a class?

● A blueprint for a new type of Python object!
  ○ The blueprint describes a general structure, and we can create specific instances of our class using this structure.

● 3 main parts
  ○ Attributes
  ○ Methods
  ○ Constructor
What is a class?

- A blueprint for a new type of Python object!
  - The blueprint describes a general structure, and we can create specific instances of our class.

- 3 main parts
  - Attributes (e.g. `oval.fill_color`, `oval.width`, etc.)
  - Methods
  - Constructor

Variables stored inside the class
What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

- 3 main parts
  - Attributes (e.g. `oval.fill_color`, `oval.width`, etc.)
  - Methods (e.g. `oval.move()`)
  - Constructor
What is a class?

- A blueprint for a new type of Python **object**!
  - The blueprint describes a general structure, and we can create specific **instances** of our class using this structure.

- 3 main parts
  - Attributes (e.g. `oval.fill_color`, `oval.width`, etc.)
  - Methods (e.g. `oval.move()`)
  - Constructor (e.g. `GOval(width, height)`)
How do we design a class?

We must specify the 3 parts:
How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*
How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

**Definition**

**instance attributes/instance variables**
These variables belong to a specific instance of our class, and every new instance of our class can have its own values for each of them.
How do we design a class?

We must specify the 3 parts:

1. Attributes: What subvariables make up this new variable type?

   instance.attribute
How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

   \[
   \text{image.width}
   \]
How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*
How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

**Definition**

*methods*

Methods are functions that belong to a class and can be called on objects that are of the type the class defines.
How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

   ```python
   instance.method(args)
   ```
How do we design a class?

We must specify the 3 parts:

2. Methods: *What functions can you call on a variable of this type?*

\`
image.get_pixel(x, y)
\`
How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

**Definition**

**constructor**

A special kind of method that *instantiates* an object of your data type (i.e. creates an instance of your class)
How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

    \[
    \text{instance} = \text{ClassName}(\text{args})
    \]
How do we design a class?

We must specify the 3 parts:

3. Constructor: *What happens when you make a new instance of this type?*

```python
image = SimpleImage(width, height)
```
How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

2. Methods: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*
How do we design a class?

We must specify the 3 parts:

1. Attributes: *What subvariables make up this new variable type?*

2. Methods: *What functions can you call on a variable of this type?*

3. Constructor: *What happens when you make a new instance of this type?*

   *In general, classes are useful in helping us with complex programs where information can be grouped into objects.*
Let’s create a social network for Python users!
Let’s create a social network for Python users!

Pynstagram.py
Let’s create a class to define a PynstaUser!
PynstaUser: We must specify our 3 parts

- Attributes?
**PynstaUser**: We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)
**PynstaUser**: We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- **Methods?**
PynstaUser: We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- **Methods**
  - Post a status
  - Add a friend
**PynstaUser:** We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- **Methods**
  - Post a status
  - Add a friend

- **Constructor?**
**PynstaUser**: We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- **Methods**
  - Post a status
  - Add a friend

- **Constructor**: User should provide a username
**PynstaUser**: We must specify our 3 parts

- **Attributes**
  - Name (string)
  - Posts (list of strings)
  - Friends (list of other PynstaUsers)

- **Methods**
  - Post a status
  - Add a friend

- **Constructor:** `PynstaUser(name)`
Defining a class

class PynstaUser:
Defining a class

class PynstaUser:

Tells Python we’re creating a new class
Defining a class

class PynstaUser:

The name of our new class

**Style note**

class names
Uppercase the first letter of every word in class names
Defining a class

class PynstaUser:

def __init__(self):

  The constructor method for our class
Defining a class

class PynstaUser:

def __init__(self, username):

  Add the username as a parameter for our constructor
Defining a class

class PynstaUser:

    def __init__(self, username):

Whenever we create methods inside our class, we must pass in self
Defining a class

class PynstaUser:

    def __init__(self, username):

        self refers to this specific instance of our class.
        In other words, it makes sure that we’re calling the method on the correct instance of our class!
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

Create and initialize the attributes for this instance of the class
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie')
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie') # Calls the constructor
Defining a class

class PynstaUser:
    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie') # Assigns username to value 'Kylie'
Defining a class

class PynstaUser:
    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie')  # self is defined implicitly
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie')
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser(‘Kylie’)
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

user = PynstaUser('Kylie')
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

    def add_friend(self, user):
        ...

    def post(self, message):
        ...

We can define more methods for our class here!
Defining a class

class PynstaUser:

    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []

    def add_friend(self, user):
        ...

    def post(self, message):
        ...

They must all take in self as a parameter
Pynstagram.py
[demo]
class PynstaUser:
    def __init__(self, username):
        self.name = username
        self.friends = []
        self.posts = []
    def add_friend(self, user):
        ...
    def post(self, message):
        ...
How and **why** would we define our own data types?
Why do we use classes?

- For ourselves
Why do we use classes?

● For ourselves
  ○ Grouping related data and the functions that act on it
  ○ Modular code development (isolation of particular tasks)
Why do we use classes?

- For ourselves
  - Grouping related data and the functions that act on it
  - Modular code development (isolation of particular tasks)

Like top-down decomposition!
Why do we use classes?

- For ourselves
  - Grouping related data and the functions that act on it
  - Modular code development (isolation of particular tasks)

This is called **encapsulation**! (more in Lecture 20)
Why do we use classes?

- For ourselves

- For others
Why do we use classes?

- For ourselves
- For others
  - We hide the implementation details of our code so others don’t need to worry about them.
  - They can just use the class, like we do for SimpleImage.
Why do we use classes?

- For ourselves

- For others
  - We hide the implementation details of our code so others don’t need to worry about them.
  - They can just use the class, like we do for SimpleImage.

This is called abstraction! (more in Lecture 21)
Why do we use classes?

- For ourselves
  - Grouping related data and the functions that act on it
  - Modular code development (isolation of particular tasks)

- For others
  - We hide the implementation details of our code so others don’t need to worry about them.
  - They can just use the class, like we do for SimpleImage.
What’s next?
More on classes next week

- A better Pynstagram
  - Can friendships be bidirectional?

- Encapsulation
  - How classes help us
  - Classes + graphics programs

- Abstraction
  - How classes help people who use our code
Roadmap

Day 1!

Programming Basics

The Console

Images

Data structures

Midterm

Everyday Python

Graphics

Graphics 1.0

Graphics 2.0

Event-driven programming

Object-Oriented Programming

Life after CS106AP!