Graphics 1.0

CS106AP Lecture 17
Today’s questions

How can we benefit from others’ code and allow others to use our code?

How can we create graphical programs?
Today’s topics

1. Review
2. Modules
3. Graphics (the Tkinter module)
4. What’s next?
Congrats on finishing the midterm!
Congrats on finishing the midterm!

You’ve learned so much!
You’ve learned so much!

- Karel
- Control flow
  - Conditionals: if, if-else, if-elif-else statements
  - Loops: while, for-each, for-range
- Functions and decomposition
- Variables and constants
- Basic data types (integers, floats, booleans)

- Strings
- Console programs
- Images
- Parsing
- File reading
- Data structures
  - Lists
  - Dictionaries
  - Nested data structures
- Style best practices
Review
Accessing and manipulating nested data structures

1. What is the outermost data structure? ➔ list/dictionary
   ○ Access its element as normal

[1.5 Store the element in a variable (optional)]

2. What is the innermost data structure?
   ○ Work with the element/element variable like you would with that data structure type!
Accessing and manipulating nested data structures

```python
contact = phone_book['Kylie']
contact['phone'] = 6500123445
# This is the same as
# phone_book['Kylie']['phone'] = 6500123445
```
A note on mutability/immutability

Mutable data structures are modified if you use a variable to modify the data structure object itself...but not if you reassign that variable to a new data structure object!
A note on mutability/immutability

Mutable data structures are modified if you use a variable to modify the data structure object itself...but not if you reassign that variable to a new data structure object!

This means that if you pass a data structure into a function, modifying the data structure within the function modifies the original data structure!
Three final functions for parsing

- `f.readlines()`  # files
- `lst.reverse()`  # lists
- `d.get()`  # dictionaries
Data structures in real life

1. What questions do we want to answer with the data?

2. Based on what we want to do, what data structures should we use to store it?
Assignment 4: BabyNames
How can we benefit from others’ code and allow others to use our code?
How can we benefit from others’ code and allow others to use our code?

Modules!
modules

*.py* files containing working code for reuse across programs; sometimes also called “libraries”
Style note

modules
Files should “have short, all-lowercase names. Underscores can be used in the module name if it improves readability.”
modules
Files should “have short, all-lowercase names. Underscores can be used in the module name if it improves readability.”

e.g. simpleimage.py
Definition

package
A directory of Python module(s)
Using modules

1. Import the module

```python
import module
```
Using modules

1. Import the module

    import module

2. Use the predefined functions!

    module.function()
Using modules

1. Import the module

   ```python
   import module
   ```

2. Use the predefined functions!

   ```python
   module.function()  
   ```

   *We call this building your code “on top of” the module.*
Modules you’ve already used!

- karel
- math
- simpleimage
- sys
- random
How can we create graphical programs?
How can we create graphical programs?

GUI libraries (modules)!
Graphical User Interface (GUI)
Visual elements that you interact with within applications (windows, buttons, scroll bars, etc.)
No GUI! All text-based
(Old computers used to only have a command line!)
GUI libraries

- Windows, Mac OS, Linux each have their own GUI systems
GUI libraries

- Windows, Mac OS, Linux each have their own GUI systems

- **GUI libraries** are modules that support the GUI (i.e. their functions allow you to interact with your computer’s GUI system)
GUI libraries

- Windows, Mac OS, Linux each have their own GUI systems
- GUI libraries are modules that support the GUI (i.e. their functions allow you to interact with your computer’s GUI system)

- `tkinter` is Python’s standard GUI package
  - Many other GUI libraries are built on top of `tkinter`!
    - (more on this later this week)
GUI libraries

- Windows, Mac OS, Linux each have their own GUI systems

- **GUI libraries** are modules that support the GUI (i.e. their functions allow you to interact with your computer’s GUI system)

- **tkinter** is Python’s standard GUI package
  - Many other GUI libraries are built on top of **tkinter**!
    (more on this later this week)
tkinter
How Tkinter works

- `import tkinter`
How Tkinter works

- `import tkinter`

- Everything gets drawn on a `canvas`
  - In today’s examples and Assignment 4, we’ll provide the code for creating the canvas.
How Tkinter works

- **import tkinter**

- Everything gets drawn on a **canvas**
  - In today’s examples and Assignment 4, we’ll provide the code for creating the canvas.

- When dealing with x, y coordinates on the canvas, Tk converts floats to integers internally to address pixels
How Tkinter works

- **import tkinter**

- Everything gets drawn on a **canvas**
  - In today’s examples and Assignment 4, we’ll provide the code for creating the canvas.

- When dealing with x, y coordinates on the canvas, Tk converts floats to integers internally to address pixels

  *A quick note about Python floats...* [demo]
How Tkinter works

● `import tkinter`

● Everything gets drawn on a `canvas`
  ○ In today’s examples and Assignment 4, we’ll provide the code for creating the canvas.

● When dealing with x, y coordinates on the canvas, Tk converts floats to integers internally to address pixels
  ○ **Note**: Python floats aren’t exact!
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text()`
Tkinter canvas functions

- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`
Tkinter canvas functions

- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`

The first point of the line is $(x_1, y_1)$
Tkinter canvas functions

- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`

The second point of the line is \((x2, y2)\)
Tkinter canvas functions

- `canvas.create_line(x1, y1, x2, y2)`
- `canvas.create_oval()`
- `canvas.create_text()`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval(x1, y1, x2, y2)`
- `canvas.create_text()`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval(x1, y1, x2, y2)`
- `canvas.create_text()`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi')`
Tkinter canvas functions

- `canvas.create_line()``

- `canvas.create_oval()``

- `canvas.create_text(x, y, text='hi')`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.NW)`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

The anchor argument determines where \((x, y)\) is. It defaults to `tkinter.CENTER`.
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

It can be any of `[tkinter.]N, S, E, W, NW, SW, NE, SW, or CENTER.`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

It can be any of `[tkinter.]N, S, E, W, NW, SW, NE, SW, or CENTER.`

Where do these come from?
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

It can be any of `[tkinter.]N, S, E, W, NW, SW, NE, SW, or CENTER.`

They are constants inside the `tkinter` module!
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

Why do these have names?
Tkinter canvas functions

- `canvas.create_line()`
- `canvas.create_oval()`
- `canvas.create_text(x, y, text='hi', anchor=tkinter.S)`

kwargs!
Keyword arguments
**Definition**

**keyword arguments (kwargs)**
Arguments whose names matter but whose positions do not
Keyword (named) arguments: `kwargs`

- Come *after* the unnamed *positional* arguments whose order matter
Keyword (named) arguments: **kwargs

- Come *after* the unnamed **positional** arguments whose order matter

---

**Definition**

**positional arguments**

Unnamed arguments whose order matter
Keyword (named) arguments: \texttt{kwargs}

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
Keyword (named) arguments: **kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
Keyword (named) arguments: **kwargs

- Come after the unnamed positional arguments whose order matter

- Name matters, but order doesn’t matter

- Help make function calls clearer
  - What if you didn’t name the arguments for `create_text()`?
Keyword (named) arguments: *kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
  - What if you didn’t name the arguments for `create_text()`?

```python
canvas.create_text(50, 100, 'hi', tkinter.NW, tkinter.left, 20)
```
Keyword (named) arguments: *kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
  - What if you didn’t name the arguments for `create_text()`?

```
canvas.create_text(50, 100, 'hi', tkinter.NW, tkinter.left, 20)
```

There are even more possible args!
Keyword (named) arguments: **kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
- Enable default values for parameters (make arguments optional)
Keyword (named) arguments: **kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
- Enable default values for parameters (make arguments optional)

```python
print('Kylie', 'Nick', 'Sonja', sep=', ', end='')
```
Keyword (named) arguments: *kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
- Enable default values for parameters (make arguments optional)

```python
print('Kylie', 'Nick', 'Sonja', sep=', ', end='')
```

*These are optional!*
Keyword (named) arguments: **kwargs

- Come after the unnamed positional arguments whose order matter
- Name matters, but order doesn’t matter
- Help make function calls clearer
- Enable default values for parameters (make arguments optional)
Proportional math: draw_pyramid()
draw_pyramid()
You Try It!

What line of code would you add to `draw_pyramid()` to draw n lines from the top edge to the lower left corner?
Proportional math:
\texttt{draw\_nest()}
Assignment 4: BabyNames
[demo]
Interactors
Interactors
Interactors

- The text fields that allow the user to *interact with* the GUI are called **interactors**
  - In particular, the text field allows the user to input information

- There are different types of interactors: text fields, buttons, etc.

- For Assignment 4, we handle the interactors for you
  - You’ll create them yourself in future assignments
Bluescreen results!
Most Artistic
Most Artistic
Most Artistic
Most Humorous
Most Humorous
Best Picture of You Hanging out with Someone Famous
Best Picture of You Hanging out with Someone Famous
Best Use of Background
Best Use of Background
People’s Choice
People’s Choice
What’s next?