Functions, Data Structures, I/O

Review

- If statements
- For loops
- While loops
- Continue statement
- Break statement
- Questions? Homework problems?

Overview

- Functions
  - Definitions
  - Calling
- Common Data Structures
  - List
  - Tuple
  - Dictionary
- Input/Output
  - From the console
  - From file
  - Pygame events

Functions

- Help us structure code
- Allows us to define new behavior for our program
- We have already use quite a few of them (print, rect, blit)
- Now we get to define them:

```
def print_multiple_hellos(some_number):
  for n in range(some_number):
    print("Hello")
```

Functions (Follow Along!)

- Make an add 3 function!
- To define a function start with the keyword `def`
- Then add the name
- Add the arguments in parentheses
- Follow the arguments with a colon
- Indent the body
- And return the argument plus 3 using the `return` keyword

That add three function!

```
def add3 (x):
  return x + 3
```

Data Structures

- Allow us to store things so that we can use them later
-
- Python offers 3 common and powerful data structures: lists, tuples, and dictionaries.
- Added bonus: sets!

Lists

```
a_list = [ 2, 1, 'bacon', 'eggs', [] ]
```

- A collection of elements enclosed in [ ] and separated by commas
- Can contain elements of any type
- [ ] denotes an empty list
- Elements in a list can be accessed by their index (starting at 0), so that a_list[2] gives us 'bacon'
- Accessing an index outside the list will throw a nasty error at you

Lists (Follow Along!)

```
>>> a = [1, 2]

>>> a

>>> a.append(3)

>>> a.remove(2)

>>> len(a)

>>> a.index(3)

>>> help(list)
```

Tuples

```
a_tuple = ( 2, 3, 5, 6, 'eggs')
```

- Collection of elements enclosed in ( ) and separated by commas
- Can contain elements of any type
- You cannot have empty tuples
- A single element tuple looks like this: (1,)
- Can't access an index outside the tuple.
- The look and work very much like lists... BUT!
- You cannot modify the contents of a tuple!

Dictionaries

```
a_dict = {'a':1,'b':'eggs',2:'bacon'}
```

- A collection of key:value pairs enclosed in { } and separated by commas.
- Keys and values can be of any type.
- However, keys cannot be modifiable (so you cannot use a variable as a key)
- Values are accessed through their key (a_dict['b'] → 'eggs')

Input/Output

- I/O is the way you interact with your users
- Main two modes of I/O are the files and the shell
- The print function prints to the shell
- The function `raw_input` function gets input from the shell

Raw Input (Follow Along!)

```
>>> some_number = raw_input(“GIMMI YOUR NUMBER! “)

>>> some_number

>>> int(some_number)
```

File I/O

- To read or write to a file you have to open it using the `open` function
- To read use a for loop on the opened file
- To write to the file use the `write` function
- When done `close` the file using the close function
- The `with` statement is an awesome shortcut for opening and closing files

File I/O (Follow Along!)

```
>>> f = open('test.txt', 'w')
>>> f.write("Hello world!")
>>> f.close()
>>> with open('text.txt', 'r') as f2:
…     for line in f2:
…         print line
```

Pygame Events

- Events are how you deal with keyboard, mouse and joystick presses
- Events have types that specify what they are (keyboard vs mouse, etc)
- First get a list of the events using the `pygame.event.get` function
- Then you loop over the events handling them as they come

Pygame Events (In Code!)

```
>>> events = pygame.event.get()
>>> for e in events:
…     if e.type == QUIT:
…         return
…     elif e.type == KEYDOWN:
…         if e.key == K_a:
…             print("A was pressed!")
…         else:
…             print("Something else...")
…     else:
…         print("Not keydown")
```

Workshop and Demo code!

- Questions so far?
- Homework!
- Use GitHub!
- Email list!
- Problems?
- Suggestions?