Catapult Python Programming Session 2

Agenda for AM and early PM

Review the Zellegraphics library

Review, Loops, exercises for you to do

Review list, tuples

"I have no idea what you are talking about!"

Say it, don't just think it!

If I go too fast, or for any other reason you are not getting it, don’t let me go on.

Stop me, ask a question!

During "all together" demo times, if you are stuck, raise your hand and look at us; one of us will come to help you.

Or perhaps someone near you can help.

Ideas from yesterday

Assignment

- x = 5   x $\boxed{5}$
- x = x + 3   x $\boxed{8}$

function definition and use

-
  ```
  def f(x):
      return 2*x + 3
  ```
- print(f(4), f(6))   prints 11 15

Questions from yesterday?

Numbers and arithmetic

Importing the math module

(Character Strings)

for loops

if-else statements

Yesterday's slides and transcript are posted on website

A function that expects multiple arguments

General form:

```
def functionName(arguments):
    statements
    return expression
```

Example:

```
def slope(x1, y1, x2, y2):
    return (y2-y1)/(x2-x1)
```

Using this function:

```
slope(4.5, 5.3, 8.0, 12.9)
```

It is also possible to define a function with some optional arguments. More on that later.

A little more live coding

Interacting with the Python shell

vs

Writing a program in a file

- 3 vs, print (3).

Getting input from the user

Range and for loops

Practice

Do this yourself (with help from neighbors and us)

- Define a function that takes the radius of a circle and returns its area. (Don't forget to import math)
- Write a loop that calls this function repeatedly, so that it prints the areas of circles whose radii are 1, 2, 3, 4, …, 9, 10.

Then a function that asks the user how they are doing. If "good", it says “that’s nice”. If “bad”, it says “that’s too bad”.

A Simple Graphics Module 1/2

This module is not part of the standard Python distribution. We added it. It's called **zellegraphics.py**

To use it on your personal computer, you will need to add **zellegraphics.py** to the appropriate **site-packages** folder.

→ We have instructions for doing that on the computer before you.

It comes from http://mcsp.wartburg.edu/zelle/python/, but we enhanced it.

Documentation is in the file zellegraphics.pdf.

A Simple Graphics Module 2/2

We'll do an example, then you'll get a chance to play.

- Another example of writing a program in a separate file. (Let's do it now.)
- Draw lines and a circle, fill a rectangle and move it.
- Notice that a window, a line, and a circle, are all examples of objects. We'll see that as we write the code.
- Later we'll see how to create our own object types.

Your turn

Use the zellegraphics module to draw a picture of a house. Make it as simple or fancy as you want. Draw and/or fill rectangles, circles, lines, ovals, polygons, individual points.

See if you can use a loop to create some part of your drawing.

Perhaps you will want to make something move.

Try to be artistic, as well as to learn about programming.

Some Python data types

Numeric types

- **int**: whole numbers
  - Arbitrarily large whole numbers.
  - Examples   6, 44389654908498902
- **float**
  - Real numbers (there can be roundoff errors)
  - Examples 4.72 1.7e4   1.0/3.0
- **complex** : the imaginary number i is represented by 1j.

Operations on numeric types

`x + y`	
`x - y`	
`x * y`	
`x / y`	
`x // y`	forces integer division
`x % y`	remainder (modulo)
`x**y`	same as pow(x,y)
`-x`	
`abs(x)`	

`x == y`	True if x equals y, False otherwise.
`x != y`	not equal
`x < y`	
`x <= y`	
`x > y`	
`x >= y`	
`cmp(x,y)`	-1 if x < y 0 if x==y 1 if x > y

Other numeric operations, such as **sin**, **cos**, **tan**, **sqrt**, **log**, can be imported from the math module.

Decisions!

You can make your program do different things based on the result of comparisions:

```
x = input(“Enter a number”)

if x < 10:
    print(“small number”)
```

Decisions!

You can make your program do different things based on the result of comparisions:

```
x = input(“Enter a number”)
if x < 10:
    print(“small number”)
else:
    print(“big number”)
```

Decisions!

You can make your program do different things based on the result of comparisions:

```
x = input("Enter a number")
if x < 10:
    print("small number")
elif x > 20:
    print("big number")
else:
    print("medium number")
```

More Python data types 1/2

Boolean

- values: True False

-
  ```
  >>> 3 < 5
  True
  ```

String – an immutable sequence of characters.

- Examples: ”abc ” ’abc ’ ’a\nc’ ’a\\c’
- print abc vs print ’abc’

More Python data types 2/2

String concatenation:

```
>>> s = "abc"
>>> s + "def"
'abcdef'
>>> 'abc' 'def'
'abcdef'
>>> s 'def'
SyntaxError: invalid syntax
```

String Operations (do live)

```
>>> alpha = 'abcdefghjklm'

>>> alpha[1]

'b'

>>> alpha [3:5]

'de'

>>> alpha[6:]

'ghjklm'

>>> len(alpha)

12

>>> alpha[0:12:2]

'acegjl'

>>> alpha[1] = 'X'
 # are strings immutable

Traceback (most recent call last):

  File "<pyshell#32>", line 1, in <module>

    alpha[1] = 'X'

TypeError: 'str' object does not support item
 assignment

>>> 'def' in alpha

True

>>> 'df' in alpha

False
```

```
>>> alpha.find('def')

3

>>> alpha.find('deg')

-1

>>> >>> alpha.upper()

'ABCDEFGHJKLM'

>>> alpha.islower()

True

>>> '***' + alpha.center(20) + '***'

'***    abcdefghjklm    ***'

>>> '***' + alpha.rjust(20) + '***'

'***        abcdefghjklm***'

>>> '***' + alpha.ljust(20) + '***'

'***abcdefghjklm        ***'

>>> sent = "This sentence has 5 words"

>>> sent.split()

['This', 'sentence', 'has', '5', 'words']

>>> sent.split('e')

['This s', 'nt', 'nc', ' has 5 words']

>>> sent.count('en')

2
```

Tuples and Lists

Tuples (immutable)

```
>>> tup1 = ('coke', 'sprite', 'dr pepper')
>>> tup1[1]
'sprite'
>>> tup1[1:]
('sprite', 'dr pepper')
>>> for drink in tup1:
  print 'I like ' + drink


I like coke
I like sprite
I like dr pepper
>>> tup1[2] = 'pepsi'


Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    tup1[2] = 'pepsi'
TypeError: 'tuple' object does not support item assignment
```

Notice the difference in the form of the use of the **sort** method and the **len** function. A list is an **Object**. Objects *know* things and *do* things.

lists (mutable)

```
>>> list2 = ['coke', 'sprite', 'dr pepper'] # all one line
>>> list2[1:]
['sprite', 'dr pepper']
>>> list2[2] = 'pepsi'
>>> list2
['coke', 'sprite', 'pepsi']
>>> numList = [2, 3, 6, 7, 3, 4, 7, 5, 3, 4, 2, 1, 8, 3];
>>> numList.count(3)
4

>>> numList.remove(4)
>>> numList
[2, 3, 6, 7, 3, 7, 5, 3, 4, 2, 1, 8, 3]
>>> numList.reverse()
>>> numList
[3, 8, 1, 2, 4, 3, 5, 7, 3, 7, 6, 3, 2]
>>> numList.sort()
>>> numList
[1, 2, 2, 3, 3, 3, 3, 4, 5, 6, 7, 7, 8]
>>> len(numList)
13
```

Getting help from IDLE

How could we find the names of all list operations?

Try

- dir(list)
- help(list)
- help(list.append)

Programs that help you write programs

IDLE: great for quick experimentation

Visual Code Studio: great for organizing bigger programs

Soon we'll install Visual Code Studio, pydev, and pygame on student-owned laptops.