Catapult Python Programming Session 2

Agenda for AM and early PM

Make sure that everyone has Python installed

More Python Basics

Review, Loops, exercises for you to do

Review lists, tuples

The Zellegraphics library

Install Python

For those of you who didn’t install Python yesterday, this is the time to do it.

There is a link to the installation instructions on the course webpage under the “Other” tab.

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

I am serious about this. Say it, don't just think it!

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

Stop me, ask a question! **I don't want anyone to get lost.**

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.

Questions from yesterday?

Yesterday's slides and transcript are on website

- Project Schedule → Session 1

---

1) numbers and arithmetic:
   - `8 / 3` `8 // 3` `8 % 3`

2) variables and assignment:
   - `width = 5` `width = 2 * height` `width = width + 2`

3) define a function:

```
def triangle_area(base, altitude):
    return base * altitude / 2
```

4) Use a function
   - `triangle_area(5, 8)` `abs(-3)` `math.sin(2.18)`

5) Strings of characters:
   - `my_name = 'claude'` `you = "your" + "mother's child"`
   - Pick out parts: `you[5]` `you[5:13]` `you[-5]`

Some Python data types - numeric

- **int**: whole numbers
  - Arbitrarily large whole numbers.
  - `6` `4438965490849890 2`
- **float**
  - Real numbers (there can be roundoff errors)
  - `4.72` `1.7e4` `1.0/3.0`
- **conversion**
  - `int (14.5 / 6.2)`
  - `float(a + 7)`
- **Complex (probably not used in Catapult)**
  - `3.7 + 2.8j`

Operations on numeric types

```
x + y
x - y
x * y
x / y
x // y      integer division
x % y       remainder (modulo)
x**y
-x
abs(x)
```

```
x == y      True if x equals y,
            False otherwise.
x != y      are x and y unequal?
x < y
x <= y
x > y
x >= y
```

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

More Python data types 1/2

Boolean

- values: True False
- >>> `3 < 5`
  True

String – a 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
```

Stretch Break

Then we will do more live coding together.

Make a folder where you will save your Python programs

When it is time to save files, save them there

(Windows: set "start in" for the IDLE shortcut)

General Structure of Programs

Steps are done in a certain order.

Some sets of steps may repeat.

We may have certain actions that we repeat with different things (e.g., when making a PB&J sandwich, we spread peanut butter on the bread, then spread jelly on the bread).

We usually need to use some things (inputs) and produce some things (outputs).

Python has several structures that support these concepts in code that you write!

Aspects of “Good” Code

Commented

- Can share with other people
- When you edit later, you can understand what you wrote
- Shows thought process, so easier to debug

Informative variable names

Encapsulation: each process is only in one place

- Instead of copying code over and over, you encapsulate it in a function that you call with different inputs over and over
- This makes it easier to change your code as you only change it in one place

Every line of code brings you closer to your goal—no useless code

Functions in Python

Functions help us when we need to repeat certain actions (like the PB&J example from the previous slide)

A function will always have:

- A name
- Inputs (called **parameters** or **arguments**)
- Output (called a **return value**)

Note that while there may be “nothing” in the inputs or output this still has to be indicated by the programmer.

In the case where no return is specified Python will return a special value of None by default.

Structure of a Function

General form:

```
def functionName(arguments):
        statements
        return expression (common but optional)
```

NB: blocks of code are indicated by indentation in Python. Indentation can include spaces, so be careful!

Examples:

```
def printHello():
    print("Hello")
```

No arguments, no return

```
def printThis(this):
    print(this)
```

One argument, no return

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

4 arguments, return

Practice

Do this yourself (with help from neighbors and us)

- Define and test a function that takes the x and y coordinates of a point and prints its distance from (0,0)
- Define and test a function that takes the side length of a square and returns its area.

Challenge

- Define a function that takes as input a day, month, and year and prints what season it is. For more fun try detecting if it is a holiday and printing out a special message in that case.

Logical Expressions

We often want to test whether a statement is true and change what we do based off of that.

To perform logical tests we use logical expressions.

The basic ones include:

and

or

not

<

>

Conditional Statements

We can change the behavior of the program based off the value of a logical test. Commonly this is done with if statements although you will see other ways we can change things soon.

An if statement will look like this—

```
if test:
    execute this stuff
```

You can also add other statements for behaviors that happen when the first test is not true--

```
if test1:
    execute this stuff
elif test2:
    test1 is False and test2 is True
    so do this stuff
else:
    test1 is False and test2 is False
    so do this stuff
```

Exercise to do together:
A function that takes a score as input and returns a letter grade, assuming a 90-80-70- … scale

Repeating Behavior in Python

We may want to repeat behavior over and over.

We can do this with “loops”. There are two types of loops: **for** loops and **while** loops. We start with **for** loops.

The code indented under the loop statement will be run repeatedly until you “break” out of the loop.

For loops will run (or iterate) a set number of times depending on the number you place where x is in this statement–

```
for i in range(x):
```

This number could also be a variable.

We did a few examples together. What about a loop inside a loop?

While Loops

While loops repeat the code inside of them until the conditional statement that controls them becomes False.

A while loop will have these things:

1) A controlling conditional statement

2) A change that occurs to factors involved in the conditional statement inside the loop

While loops look like this:

```
while(conditional):
    do this stuff
    update condition
```

Practice

Do this yourself (with help from neighbors and us)

- Define a function that takes an input number and determines whether it is between -5 and 5. If it is inside the range it should print “Good”, otherwise it prints “Bad”. Challenge: also print “Zero” if it is between -1 and 1. Try to make this happen with the fewest number of statements possible.
- Define a function that takes a number (itself) and prints out itself + 0, itself + 1, itself + 2, … , itself + itself.
- Define a function that takes a number (itself) and does the same thing as the previous function but this time stops when the number that would be printed is divisible by 7.

Challenge

- Create a function that calls another function within a loop.
- Create a loop that changes based on the output of a function.

A Simple Graphics Module

This module is not part of the standard Python distribution. We added it. Written by John Zelle, enhanced at Rose-Hulman

To use it on your personal computer, you will need to add **zellegraphics.py** to the Python34\Lib\site-packages folder.

Documentation is linked from the same place: **zellegraphics.pdf**).

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

- 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.

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. (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 graphics 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.

We'll have "show and tell" at the end.