Reminder: You can always leave at 4pm CT! I tend to go over, sorry about that. We'll cover some problem set specific items towards the end if you want to stay.

Warm-up (We'll start at 2:08 CT; As usual, if you haven't been to my section before, send me a message with your usual TF's name so I can count your attendance ☺)

In warmup.py, do the following:

- Create a variable times and set it equal to 5. Create another variable eat and set it equal to "nom"
- Print eat * times. What do you get?
- Write arr = [x for x in range(3)]
- Print arr. What do you get?
- Create a variable str and set it equal to "hungry."
- Print ([x+"nom" for x in str]). What do you get?
- Write a program that prompts the user for two integers (using get_int("blah")) and prints the sum. Import cs50 at top!

CS50 Section 6

DESIGN

Python files are written using the .py file extension, and are run via the Python interpreter. Run the following line in the terminal to run a file called dna.py.

```
$ python dna.py
```

In CS50, we teach Python 3, though Python 2 is still popular. Be careful when finding examples online to restrict your search to Python 3.

- Variables in Python do not require a type specifier, and do not need to be declared in advance.

- Conditionals look similar to C, but have slightly different keywords and are **governed by indentation level**, rather than curly braces.
- *INDENT CORRECTLY OR YOU WILL DIE.*
- Create a program that does the following:
  - Get two inputs from the user, x and y.
  - If x > y and x is even, print “x > y and x is even”
  - If x > y and x is odd, print “x > y and x is odd”
  - If x = y, print “x = y”
  - If x < y, print “x < y”

- Loops primarily exist in two varieties only: for and while.
- *INDENT CORRECTLY OR YOU WILL DIE. again.*
- Create a program that outputs numbers from 1 to 100 using a for loop and a while loop.
    - Note that x++ is not valid in Python! Use x += 1 instead.

- Create a program that outputs even numbers from 1 to 100 using a for loop and a while loop.

- Arrays in Python are more formally referred to as **lists**. And they are way more flexible than C's. You can mix types, and they grow and shrink dynamically!
    - nums = []
    - nums = [1, 2, 3, 4]
        - To add a number to a list, do nums.append(5).
    - nums = [x for x in range(100)]
        - This is called list comprehension!

Strings are just like arrays!

str = “doggo”

str[1] gives you “o”

len(str) gives you 5

When printing, you can specify print(“blah”, end=“”) to prevent a newline from being printed.

Exercise: Write a program reverse.py that reverses a string. Get the string from the user using get_string(“text:”). Import cs50 up top!

- *Tuples* are a new concept in Python; they represent ordered data. Sort of like x-y coordinates. Use parentheses to denote them!

- A list of tuples might look like this:

  presidents = [("George Washington", 1789), ("John Adams", 1797), ("Thomas Jefferson", 1801)]

- We can iterate through tuples via a for loop! (For loops are LITERALLY the goat. Iterate through lists and even strings!

```
for pres, year in presidents:

        print(f”In {year}, {prez} took office.”)
```

- *Dictionaries* in Python are similar to hashtables from C. They map key-value pairs. But the values don’t have to just be strings!

- General structure:

```
pizzas = {
        "cheese": 9,
        "pepperoni": 10,
        "vegetable": 11,
        "buffalo chicken": 12
}
```

- Adding to a dictionary:

```
pizzas[“bacon”] = 14
```

- Indexing in a dictionary:

```
if pizzas["vegetables"] < 12:
        # do something
```

```
pizzas = {
        "cheese": 9,
        "pepperoni": 10,
        "vegetable": 11,
        "buffalo chicken": 12
}
```

What would these output? In a dictionary, a for loop iterates through the *keys* of a dictionary.

```
for pie in pizzas:
        print(pie)
```

```
for pie, price in pizzas.items():
        print(price)
```

What can we write instead of print(price) to print something like “A whole cheese pizza costs $9.”?

- Functions behave nearly identically to C, and just have a different syntax.

```
def square(x):
    return x * x

def better_square(x):
    return x ** 2

print(square(5))
```

- Python is ***object-oriented***.

- Think of an object like a C structure. They contain a number of fields which we'll now start calling *properties*, but they also contain **functions** that might apply only to those objects. We call those *methods*.

- You've seen us use several methods already!

```
pizzas = {
        "cheese": 9,
        "pepperoni": 10,
        "vegetable": 11,
        "buffalo chicken": 12
}

for pie, price in pizzas.items():
        print(f"A whole {pie} pizza costs ${price}.")
```

```
nums = [1, 2, 3, 4]
nums.append(5)
```

Lists (and indeed most native things in Python) are already objects, though it is also possible to define your own objects.

To create a new type of object you define a Python *class*. The only method required of a class is the method one uses to create an object of that type, which we normally call a *constructor*.

```
class Student():

        # constructor, and this is two underscores on each side
        def __init__(self, name, id):
                self.name = name
                self.id = id

        # method to change a student's ID
        def changeID(self, id):
                self.id = id

        # method to print the object. No parameters but still need self
        def print(self):
                print(f"{self.name} has ID {self.id}")

phyllis = Student("Phyllis", 10)
phyllis.print()
phyllis.changeID(11)
phyllis.print()
```

- To include files similar to what we did in C, use Python's import! At the top of your file, write:

```
import cs50
```

- Then you can use the functions inside of CS50's ***module***.
- You can also import specific functions inside of files.

```
from cs50 import function
```

FILES FILES FILES FILES FILES

```
#at top in order to read arguments from the user
import sys;

#check correct number of arguments
if len(sys.argv) != (some number):
        sys.exit("Usage: wrong input ):<")

#open a file
f = open(sys.argv[some number])

#reader for a text file
contents = f.read()
```

```
#open another file that we write to
outfile = open(sys.argv[some number], "w")

#write contents to that file
outfile.write(contents)

#close a file
f.close()
```

Exercise: Write a program copy.py that copies a text file, where the original and the copied file are specified as command line arguments.

FILES FILES FILES FILES FILES

```
import csv; #at top in order to read csv files

reader = csv.DictReader(f) #reader for a csv file

fields = reader.fieldnames #a list of fieldnames
#check if your required fields are in the list



for row in reader: #for csv files
        #code
f.close() #close our file
```

Exercise:

Write a program phonebook.py that reads from a CSV file provided as a command-line argument and prints in the format “Phyllis’s phone number is 806-392-3098”. The file contains columns *name* and *number* representing each person’s name and phone number.

Hint:

Once you have a row in your reader, you can do

```
name = row[“name”]
number = row[“number”]
```

Lab

Problem Set

Splicing a list:

Exiting a program:

File I/O from the previous slides!

Count occurrences:

CS50 Section 6