An Introduction to Python

Part III: The Rest of Part II

Introduction

- **Mentors:** Mike & Corin
- **Goals**
  - Review Lecture 1 and 2
  - Lists
  - For Loops
  - Activities
- **Any Questions Before We Start?**

Lists and For Loops

Cats

```
cat0 = “Bob”

cat1 = “Alice”

cat2 = “Tom”

cat3 = “Boots”

cat4 = “Elthur”
```

- This is tedious and inefficient!
- Can we do something similar, but in a way that is less tedious and inefficient?

Lists

```
>>> cats = [“Bob”, “Alice”, “Tom”, “Boots”, “Elthur”]
>>> cats[0]
“Bob”
>>> cats[1] + “ and “ + cats[4] + “ are both cats”
“Alice and Elthur are both cats”
```

This is much better! Lists store any number of elements which are accessed like this:

- **`list[0]`** = first element
- **`list[1]`** = second element
- **`list[2]`** = third element

***Notice** that the counting starts from 0!

aside: Lists VS. Arrays

Lists

- Can grow and shrink in size
- Can hold multiple different data types!

```
list = [1, True, "three"]
```

- Also slower, but this doesn't really matter to us.

Arrays

- Size is fixed once created
- Can only hold one type
- More common in other programming languages
- We will generally only be using lists, so don't worry about these!

List Slicing

- What if we only want a certain part of a list?
- We can slice them just like with Strings via **`list[n:m]`** which will give us a list starting at the **`nth`** list element and ending just *before* the **`mth`** element.

```
>>> cats = [“Bob”, “Alice”, “Tom”, “Boots”, “Elthur”]
>>> cats[:2]
[“Bob”, “Alice”]

>>> cats[2:]
[“Tom”, “Boots”, “Elthur”]

>>> cats[1:3]
[“Alice”, “Tom”]
```

List Functions

- `sum(list)` - takes a list, returns the sum of all its members.
- `min(list)` - takes a list, returns the smallest of its members.
- `max(list)` - takes a list, returns the largest of its members.
- `len(list)` - takes a list, returns its length.
- `list.append(item)` - takes `item` and adds it to the end of `list`.
- `list.pop()` - removes the last item in a list and returns it.

```
>>> s = [1,3,4,6,2,9,0]
>>> sum(s)
25
>>> min(s)
0
>>> max(s)
9
>>> len(s)
7
>>> s.append(10)
>>> s
[1, 3, 4, 6, 2, 9, 0, 10]
>>> s.pop()
10
```

Looping with Lists

What if we want to do something to everything in a list?

```
cats = [“Bob”, “Alice”, “Tom”, “Boots”, “Elthur”]

# print all cats
print(“Here are your cats:”)
i = 0
while i < len(cats):
    print(cats[i])
    i = i + 1
```

We could use our old friend, the **`while`** loop.

For Loops

However, with lists, there's an even better way to loop through them: **`for`** loops. This code makes much more sense!

```
cats = [“Bob”, “Alice”, “Tom”, “Boots”,
“Elthur”]

# print all cats
print(“Here are your cats:”)

for cat in cats:
    print(cat)
```

Syntax: **`for x in list`**, where **`x`** is a variable name and **`list`** is any list.

For Loops - More Examples

When **`for`** loops are given a String, they iterate through each character in that String

```
str = “Hello”
reversed_str = “”

for ch in str:
    reversed_str = ch + reversed_str
```

For Loops - More Examples

```
names = [“Polly”, “Yvette”, “Tom”, “Howard”,
“Ornette”, “Nancy”]

first_letters = “”
for name in names:
    first_letters = first_letters + name[0]

# This will print “PYTHON”
print(first_letters)
```

Range

- How would you use a `for` loop if you want to loop through consecutive numbers?
- You could manually define a list of all the numbers you want.
- Or you could use **`range()`**
- **`range()`** takes a number `n`, and returns a list* of all the numbers from `0` to `n-1`.
- If **`range()`** is given two numbers `n` and `m`, it will return a list of all numbers `>= n` and `< m`

*something that acts like a list actually, but we don't need to worry about this.

```
# the bad way
nums = [0,1,2,3,4]
for i in nums:
    print(i)

# the better way
for i in range(5):
    print(i)

list = []
for i in range(10):
    list.append(2 ** i)
# this will produce [1, 2, … , 512]
```

Lists and For Loops Exercise

Try it yourself!

1. Create a new file called **`fib.py`** by typing:

   ```
   idle3 fib.py &
   ```

2. Write the following program in the file:

   a. `Make a function called fib that takes in a number n`

   b. `If the user only wants 1 or 2 numbers, just return those`

   c. `If not, use a` **`for`** `loop to keep appending numbers to a list in terms of the last two`

   d. `Return the list`

3. Run the file in terminal: **`python3 fib.py`**

Summary

- **Lists**
  - They are different from arrays in Python
  - A good data structure to store elements of various types
- **For Loops**
  - Does the same thing as a While loop, but it makes more sense to use in some contexts

Any Questions?

Activities!

Next Lecture: Introducing Object-Oriented Programming

Distant future plans

- Games & Cyber Security Hackathon