Bigger Data:
Lists and Loops

Today's plan

- Talk about Lab 1
- Review
- Lists
- Sequences
- **for** loops
- Example programs, program structure

Lab 1

- Read the write-up
- Start early to take advantage of support
  - Labs, office hours, ninja sessions
- Run **update21** before you start
- Run **handin21** whenever you make progress, when you finish
- Remote access to lab computers: www.cs.swarthmore.edu/help/access.html

Review

- Four parts of a program
  - input, computation/algorithm, output, repetition
- Python programs vs Python shell
- Input and output for Python programs
  - **raw_input** function gathers text from user
  - **print** function displays text to user

New type: Lists

- Ordered, numbered group of values, usually all of the same type

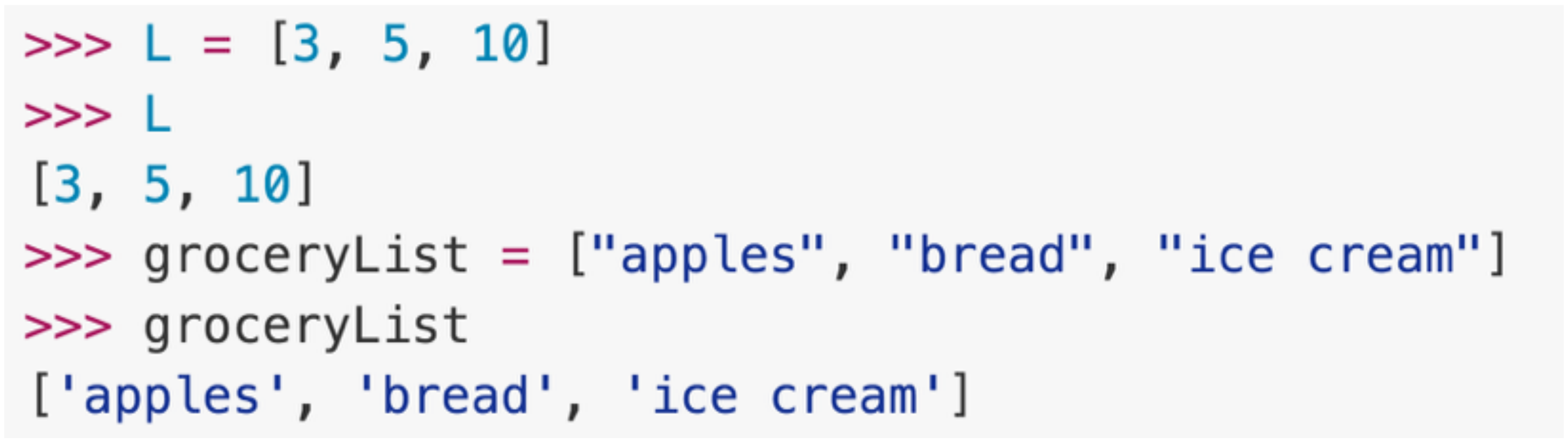
- Sometimes we want to treat the list as a single thing, sometime we want to access individual items in a list

- Values in a list are sometimes called **elements** or **items**

- Each item is numbered with an **index**, starting at 0.

Creating lists

- Square brackets, separated by commas



```
>>> L = [3, 5, 10]
>>> L
[3, 5, 10]
>>> groceryList = ["apples", "bread", "ice cream"]
>>> groceryList
['apples', 'bread', 'ice cream']
```

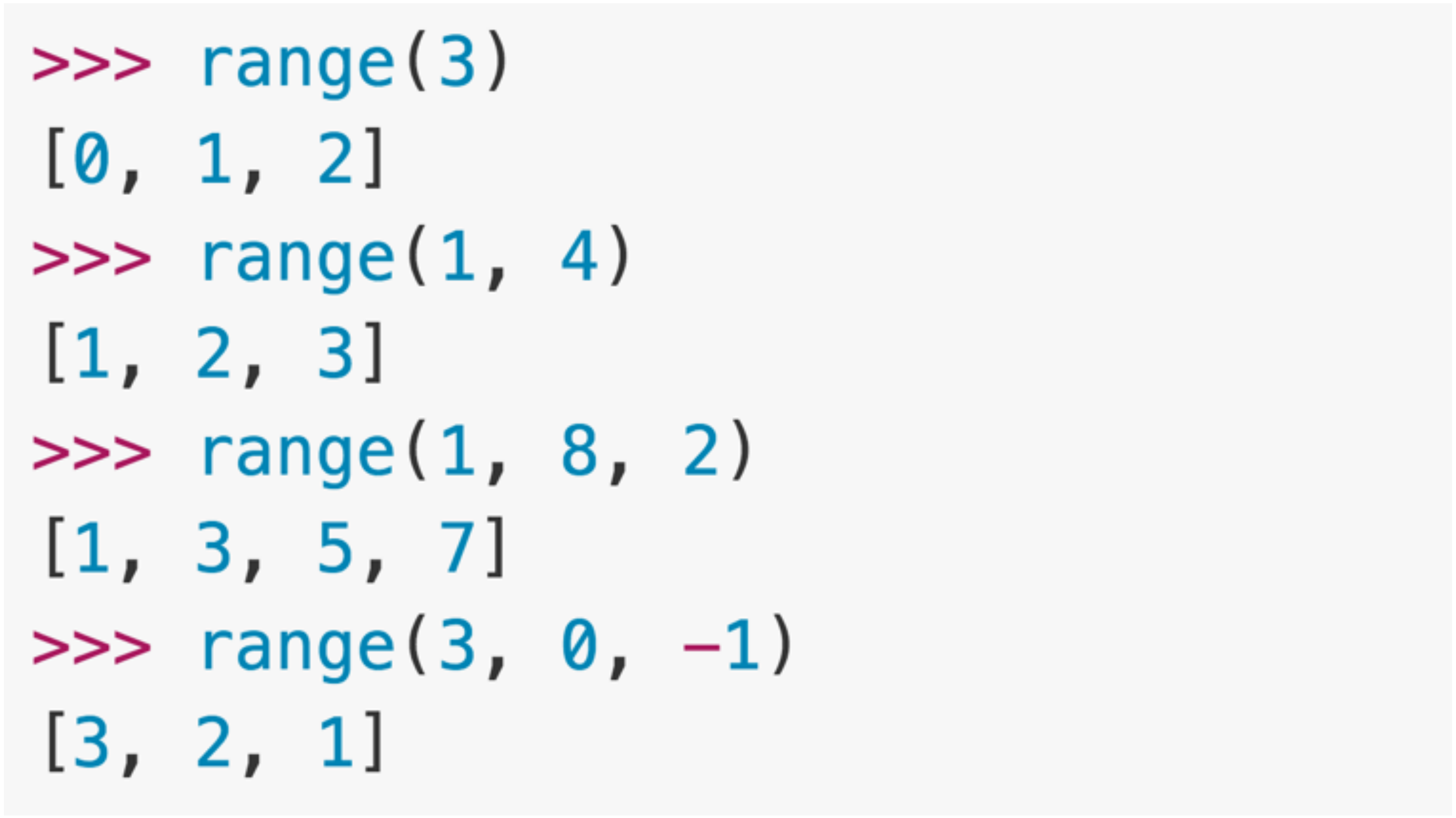
Creating lists

- Expressions in list are evaluated first
- Lists can be empty

```
>>> x = "hello"
>>> L = [x, len(x) + 2.09, 1/2, x + ", you"]
>>> L
['hello', 7.09, 0, 'hello, you']
>>> L = []
>>> L
[]
```

Creating lists with range

- Lists of evenly spaced integers
- range function takes up to three arguments



```
>>> range(3)
[0, 1, 2]
>>> range(1, 4)
[1, 2, 3]
>>> range(1, 8, 2)
[1, 3, 5, 7]
>>> range(3, 0, -1)
[3, 2, 1]
```

range function

- Three arguments: **start**, **stop**, **step**
- Side effects: none
- Returns: a list with the integers from **start** to **stop**, skipping by **step**. **start** is included, but **stop** is not.

More on range

- range(start, stop, step)
- range(start, stop)
  - step is assumed to be 1
- range(stop)
  - step is assumed to be 1
  - start is assumed to be 0

Sequences

- Strings and lists are both **compound** or **composite** data types
- A whole made up of pieces
- Collectively, we call such data types **sequences**

for loops

- Python **control structure** that **traverses** a sequence
- A for loop looks like:

```
for <variable> in <sequence>:
    <body>
```

- The instructions in the `<body>` will happen once for each value in the sequence. The `<variable>` will be assigned to each value in turn.

for loops

- Before: do all the instructions in order

- Now: selectively repeat certain instructions, once for each item in a sequence

Programs that use for loops

Unrolling the loop

- This:

```
for i in [3, 5, 10]:
    squared = i**2
    print(squared)
```

- Is short for:

```
i = 3
squared = i**2
print(squared)

i = 5
squared = i**2
print(squared)

i = 10
squared = i**2
print(squared)
```

Different types of for loops

- `for` **`item`** `in` **`L:`**
  - do something with each value in the list **L**
- `for` **`char`** `in` **`S:`**
  - do something with each character in the string **S**

Different types of for loops

- `for i in range(start, stop):`
  - do something with the ints from **start** up to, but not including, **stop**
- `for i in range(n):`
  - do something **n** times

Recap

- New type: lists
- New function: range
- Lists and strings (and some other types) are sequences

Recap

- New control structure: for loop
- `def main():`
- block comment
- New arithmetic operator: `**`

Good luck on Lab 1!