Week 7

Lists

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Lists

- **list**: Python's equivalent to Java's array (but cooler)
  - Declaring:

    name = [**value**, **value**, **...**, **value**]   or,

    name = [**value**] * **length**

  - Accessing/modifying elements: (same as Java)

    name[**index**] = **value**

```
>>> scores = [9, 14, 18, 19, 16]
[9, 14, 18, 19, 16]
>>> counts = [0] * 4
[0, 0, 0, 0]
>>> scores[0] + scores[4]
25
```

Indexing

- Lists can be indexed using positive or negative numbers:

```
>>> scores = [9, 14, 12, 19, 16, 18, 24, 15]
>>> scores[3]
19
>>> scores[-3]
18
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	9	14	12	19	16	18	24	15
index	*-8*	*-7*	*-6*	*-5*	*-4*	*-3*	*-2*	*-1*

Slicing

- **slice**: A sub-list created by specifying start/end indexes

```
name[start:end]          # end is exclusive
name[start:]             # to end of list
name[:end]               # from start of list
name[start:end:step]     # every step'th value
```

```
>>> scores = [9, 14, 12, 19, 16, 18, 24, 15]
>>> scores[2:5]
[12, 19, 16]
>>> scores[3:]
[19, 16, 18, 24, 15]
>>> scores[:3]
[9, 14, 12]
>>> scores[-3:]
[18, 24, 15]
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	9	14	12	19	16	18	24	15
index	*-8*	*-7*	*-6*	*-5*	*-4*	*-3*	*-2*	*-1*

Other List Abilities

- Lists can be printed (or converted to string with `str()`).
- Find out a list's length by passing it to the `len` function.
- Loop over the elements of a list using a `for` ... `in` loop.

```
>>> scores = [9, 14, 18, 19]
>>> print "My scores are", scores
My scores are [9, 14, 18, 19]
>>> len(scores)
4
>>> total = 0
>>> for score in scores:
...     print "next score:", score
...     total += score
next score: 9
next score: 14
next score: 18
next score: 19
>>> total
60
```

Exercise

- Recall the midterm `scores.txt` data:

```
76
89
76
72
68
```

- Recreate the `Midterm` histogram from lecture in Python:

```
75: *
76: *****
79: **
81: ********
82: ******
84: ***********
```

Ranges, Strings, and Lists

- The `range` function returns a list.

```
>>> nums = range(5)
>>> nums
[0, 1, 2, 3, 4]
>>> nums[-2:]
[3, 4]
>>> len(nums)
5
```

- Strings behave like lists of characters:
  - `len`
  - indexing and slicing
  - `for` ... `in` loops

String Splitting

- `split` breaks a string into a list of tokens.

  name`.split()`   `# break by whitespace`

  name`.split(`**delimiter**`)`   `# break by delimiter`

- `join` performs the opposite of a `split`

  delimiter`.join(`**list**`)`

```
>>> name = "Brave Sir Robin"
>>> name[-5:]
'Robin'
>>> tokens = name.split()
['Brave', 'Sir', 'Robin']
>>> name.split("r")
['B', 'ave Si', ' Robin']
>>> "||".join(tokens)
'Brave||Sir||Robin'
```

Tokenizing File Input

- Use `split` to tokenize line contents when reading files.
  - You may want to type-cast tokens: **type**(**value**)

```
>>> f = open("example.txt")
>>> line = f.readline()
>>> line
'hello world 42 3.14\n'

>>> tokens = line.split()
>>> tokens
['hello', 'world', '42', '3.14']

>>> word = tokens[0]
'hello'
>>> answer = int(tokens[2])
42
>>> pi = float(tokens[3])
3.14
```

Exercise

- Recall the `hours.txt` data:

```
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

- Recreate the `Hours` program from lecture in Python:

```
Susan worked 31.4 hours, 7.85 / day, 2 days above average
Brad worked 36.8 hours, 7.36 / day, 2 days above average
Jenn worked 39.5 hours, 7.9 / day, 4 days above average
```

Exercise Answer

hours.py

```
file = open("hours.txt")
for line in file:
    tokens = line.split()
    id = tokens[0]
    name = tokens[1]

    hours = 0.0   # cumulative sum of employee's hours
    days = 0
    for token in tokens[2:]:
        hours += float(token)
        days += 1

    average = hours / days
    above = 0     # compute number of days above average
    for token in tokens[2:]:
        if float(token) > average:
            above += 1

    print name, "worked", hours, "hours (", average,  \
          "/ day," above, "days above average"
```