Lists – more versatile sequences

- Lists are another sequential data type
- But unlike strings, lists …
  - – can hold any type of data (not just characters)
  - – *are* mutable – legal to change list elements
- Use square brackets, `[ ]` to define a list

```
fruit = ['apple', 'pear', 'orange']
```

- And use `[ ]` to access elements too

```
fruit[2]    >>> 'orange'
```

  - – Index slicing works the same as strings too

More operations involving lists

- Built-in functions like `len` (same as strings)
  - Use `max` and `min` for extremes (work for strings too)
  - And `sum` (only if all elements are number types)
- Test membership like with strings: `in`, `not in`
- But unlike strings, can use built-in `del` operator:

```
fruit >>> ['apple', 'pear', 'orange']
del fruit[1]
fruit >>> ['apple', 'orange']
```

- Also can use `[ ]` with `=` to change elements too

```
fruit[0] = 'tangerine'
fruit >>> ['tangerine', 'orange']
```

List + and * operations

- `+` concatentates (but both operands must be lists)

```
nums = [20, -92, 4]
nums + 9 >>> TypeError
nums + [9] >>> [20, -92, 4, 9]
```

- `*` repeats (one operand is a list, other is an int)

```
nums * [2] >>> TypeError
nums * 2   >>> [20, -92, 4, 20, -92, 4]
```

- Note: can make a list of lists, but still just 1 `nums`

```
[nums] * 2   >>> [[20, -92, 4], [20, -92, 4]]
```

  - Explained next slide

Actually, lists hold references

- Look at prior example a different way to see this

```
[nums, nums] == [nums] * 2 >>> True
```

- Now give a name for the list of list references

```
numList = [nums, nums]
numList >>> [[20, -92, 4], [20, -92, 4]]
```

- Delete an item from original list – see result!

```
del(nums[0])
numList >>> [[-92, 4], [-92, 4]]
```

- To understand: study p. 124 (especially Fig. 4.4)

Finding extreme values

- Usually able to use built-in functions `max`, `min`
  - But what if we didn't have such functions?
  - Or what if they don't fit our problem (e.g. max odd)?
- Basic algorithm applies to any extreme

```
Store value (or index) of first list item
Loop through remaining items:
    If current more extreme than stored item:
        Replace stored extreme item (or index)
```

  - Assumes there is at least one item in the list

Another way to create: `list()`

- With no arguments, creates an empty list

```
list() >>> []
```

- Or pass any sequence as an argument

```
list(range(3)) >>> [0, 1, 2]
list('cat') >>> ['c', 'a', 't']
```

- Makes a copy of another list

```
nums = [-92, 4]
numsCopy = list(nums)
nums[0] = 7
nums >>> [7, 4]
numsCopy >>> [-92, 4]
```

Try it!

Methods to add/remove list items

- `alist.append(item)` – similar but not same as `alist = alist + [item]` – append does not make a new list, just adds an item to old list
- `alist.insert(i,item)` – inserts item at i^{th} index; later items moved down one (toward end)
- `alist.remove(item)` – removes first occurrence of item; later items moved up by one
  - `ValueError` if item not in the list
- `alist.pop()` – removes *and returns* last item
  - `alist.pop(i)` – removes and returns i^{th} (index) item
  - `IndexError` if empty list or `i` not valid for the list

Try it!

Some other list methods

- `alist.index(item)` – returns index of first occurrence of item
  - `ValueError` if item not in the list
- `alist.count(item)` – returns number of occurrences of item in the list
- `alist.sort()` – sorts list items by value into ascending order (error if items not comparable)
- `alist.reverse()` – reverses the order of all items in the list
- Q. How to sort items into descending order?

Making a list by `split`ting a string

- A handy string method named `split` returns a list of substrings

```
def countWords(string):
    substrings = string.split()
    return len(substrings)
```

- Default delimiter is white space – consecutive spaces, tabs, and/or newline characters
- Can specify a different delimiter

```
>>> 'dog/cat/wolf/ /panther'.split('/')
['dog', 'cat', 'wolf', ' ', 'panther']
```

Calculating average values

- What do we mean by average (a.k.a., central tendency)?
  - Usually "mean" but sometimes "median" or "mode"
- Easy to calculate mean of list x in Python

```
xmean = sum(x) / len(x)
```

- A little bit harder to find median

```
xs = sorted(x) # need a sorted copy (sorted is built-in)
n = len(x)
if n % 2 == 1: # odd number of values: middle one is it
    xmedian = xs[n//2]
else: # even number of values: find average of middle two
    xmedian = ( xs[n//2] + xs[n//2-1] ) / 2
```

- Harder yet to find mode, but not too bad with a dictionary

Dictionaries – key/value pairs

- Unordered *associative* collections
  - Basically lists, but access each value by a key instead of an index position
- Use curly braces, `{ }` to define a dictionary

```
ages = { 'sam':19, 'alice':20 }
```

- Use familiar `[ ]` to access, set or delete *by key*

```
ages['alice']    >>>  20
ages['pete'] = 24 #  adds new item in this case
del(ages['pete']) #  bye bye pete
```

  - Index slicing doesn't make sense though, because values not stored in discernible order

Some dictionary methods

- Get lists of all keys, all values, or all pairs

```
list(ages.keys()) >>> ['alice', 'sam']
list(ages.values()) >>> [20, 19]
list(ages.items()) >>>
  [('alice', 20), ('sam', 19)] # each is a tuple
```

  - Note: a tuple is immutable, but otherwise same as a list

- Or use get method (without or with default)

```
ages.get('harry') >>> None
ages.get('harry', 0) >>> 0
```

Try it!

Finding the mode of a list

- First note: might be more than one mode

```
def mode(alist):   # Listing 4.6 (and start of 4.7)
    countdict = {}

    for item in alist:
        if item in countdict:
            countdict[item] = countdict[item]+1
        else:
            countdict[item] = 1
```

  - Continued next slide

Finding mode (cont.)

- Rest of Listing 4.7:

```
countlist = countdict.values()
maxcount = max(countlist)

modelist = [ ]   # in case there is more than one
for item in countdict:
    if countdict[item] == maxcount:
        modelist.append(item)

return modelist
```

Printing a frequency table I

- Easiest with a dictionary (rev. Listing 4.8):

```
countdict = {}
for item in alist:
    if item in countdict:
        countdict[item] = countdict[item] + 1
    else:
        countdict[item] = 1
itemlist = list(countdict.keys())
for item in sorted(itemlist):
    print(item, "\t", countdict[item])
```

Printing a frequency table II

- A bit more to do by yourself without a dictionary (rev. Listing 4.9):

```
slist = sorted(alist)
previous = slist[0]
groupCount = 0
for current in slist:
    if current == previous:
        groupCount = groupCount + 1
    else:
        print(previous, "\t", groupCount)
        previous = current
        groupCount = 1
print(current, "\t", groupCount)
```

Measuring dispersion

- How much do values *vary* from the average?
- Differences from mean: `x[i] - mean(x)`
  - Includes positive and negative differences
  - So usually square difference: `(x[i] - mean(x))**2`
- Variance = sum of squared differences (for all `i`), divided by `n - 1` (ask me why `n - 1`, not `n`)
- Standard deviation = square root of variance
  - See Listing 4.11

$$sd = \sqrt{\frac{\sum_{i=0}^{n-1} \left(x[i] - mean(x)\right)^2}{n-1}}$$

About redundant calculations

- Why not `x[i]-mean(x)` inside loop (in Listing 4.11)?
  - Because no need to recalculate the mean `n` times!
- Related question: why loop twice then – once for the mean, again for standard deviation?
  - Summation algebra → "computational formula"
- Calculate sum and sum of squares in same loop
  - Will see in comp.py (uses tuple to return both mean and s.d.) and regress.py (for fun?) – *after* learning file basics

$$sd = \sqrt{\frac{n * \sum x[i]^2 - \left(\sum x\right)^2}{n * (n-1)}}$$

Next

Reading and writing text files