ECS 10

3/18

Announcements

- Final - Wds 20, 1-3pm, this room.
- Similar to Midterm 2; two programs.
- Bring a Scantron.
- Open book, notes, programs. No computers, phones.

What to review

- Using a dictionary
  - Creating
  - Putting data in
  - Getting data out
  - Changing values for an item
- Using a list
  - Putting data in
  - Getting data out
  - Changing items
  - Sorting

- Combined data structures
  - List of tuples
  - Dictionary with lists as values
- Functions
  - Calling functions
  - Passing data in
  - Returning data out
  - Local variables
  - Global variables

Materials

- You're responsible for what was covered in lecture. Review slides.
- Emphasis on things in programming assignments.
- No tkinter; may be some MC on object-oriented programming.
- Two sets of examples in Python Tutor. These are really helpful. On page with slides.
- Practice final is on SmartSite, and several practice programming problems.

Functions

```
def initX():
    x = 10
def main():
    print(x)
main()
```

What does this do?

Functions

```
def  initX():
    x = 10
def main():
    print(x)
main()
```

Crashes. The variable x belongs to function initX, not main.

Also, initX is never called!

Functions, strings

```
def twoChars(myStr, i):
        if i < 0 or i > len(myStr)-2:
                return ""
        one = myStr[i]
        two = myStr[i+1]
        return(one+two)

def main():
        s = twoChars("Edith",2)
        print(s)
main()
```

What does this print?

Functions, strings

```
def twoChars(myStr, i):
        if i < 0 or i > len(myStr)-2:
                return ""
        one = myStr[i]
        two = myStr[i+1]
        return(one+two)

def main():
        s = twoChars("Edith",2)
        print(s)
main()
```

Prints "it". The function takes two inputs, a string and a start position, and returns the two characters in positions i and i+1.

Functions, lists

```
def setUpList():
    L = []
    for i in range(4):
        L.append(i)
    return L
def main():
    numbers = setUpList()
    print(numbers)
main()
```

- What does it print?

Functions, lists

```
def setUpList():
    L = []
    for i in range(4):
        L.append(i)
    return L
def main():
    numbers = setUpList()
    print(numbers)
main()
```

- Prints the list [0,1,2,3]
- What about…

Functions, lists

```
def setUpList():
    L = []
    for i in range(4):
        L.append(i)
def main():
    setUpList()
    print(L)
main()
```

- What does it print?

Functions, lists

```
def setUpList():
    L = []
    for i in range(4):
        L.append(i)
def main():
    setUpList()
    print(L)
main()
```

- Crashes. L is not defined in main.

Functions, lists

```
def setUpList(L1):
    for i in range(4):
        L1.append(i)
def main():
    L = []
    setUpList(L)
    print(L)
main()
```

- What does it print?

Functions, lists

```
def setUpList(L1):
    for i in range(4):
        L1.append(i)
def main():
    L = []
    setUpList(L)
    print(L)
main()
```

- Also [0,1,2,3].
- The list is defined in main. It gets passed to setUpList() as L1 and modified there. Since lists are mutable the modifications are visible in main().
- What if we tried this with a string?

Functions, strings

```
def setUpString(S1):
    for i in range(4):
        S1=S1+ str(i)
def main():
    S = ""
    setUpString(S)
    print(S)
main()
```

- What does it do?

Functions, strings

```
def setUpString(S1):
    for i in range(4):
        S1=S1+ str(i)
def main():
    S = ""
    setUpString(S)
    print(S)
main()
```

- Prints the empty string. The string gets passed into setUpString, and changed there, but changes to S1 do not affect S.

A right way to do it

```
def setUpString(S1):
    for i in range(4):
        S1=S1+ str(i)
    return S1
def main():
    S = ""
    S = setUpString(S)
    print(S)
main()
```

List indexing

```
L = [5,7,2,6,3]
averages = []
_____:
    avg = (L[i]+L[i+1])/2
    averages.append(avg)
print(averages)
```

Prints [6, 4.5, 4, 4.5]

What could fill in the blank?

List indexing

```
L = [5,7,2,6,3]
averages = []
for i in range(0,len(averages):
    avg = (L[i]+L[i+1])/2
    averages.append(avg)
print(averages)
```

List of tuples

```
L = [ (354,"Yolo"), (175,"Napa")]
for  i in range(0,2):
    _____
    _____
```

- Prints:

```
Yolo score: 354
Napa score: 175
```

- Fill in the blank.

List of tuples

```
L = [ (354,"Yolo"), (175,"Napa")]
for  i in range(0,2):
    tupe = L[i]
    print(tupe[1], "score:", tupe[0])
```

List of tuples

```
L = [ (354,"Yolo"), (175,"Napa")]
for  i in range(0,2):
    print(L[i][1], "score:", L[i][0])
```

Putting things into a dictionary

12. This program makes a dictionary giving the frequency of each letter of the alphabet; `e` is the most frequent letter, `t` the second most frequent, and so on.

```
frequencyDict = {}
alphabet = "etaoinshrdlucmfwypvbgkqjxz"
for i in range(len(alphabet)):
    ----------------------
for char in "pet":
    print (frequencyDict[char])
```

This program prints `17`, then `0`, then `1`. What should go in the missing line?

For on a dictionary

```
Band = {}
Band["Anders"] = "tuba"
Band["Ho"] = "flute"
Band["Moon"] = "tuba"
L = []
_____________________
    _____________________
    L.append( (instrument, name) )
L.sort()
```

What could fill in the blanks?

PS why the double parens in the append?

For on a dictionary

```
Band = {}
Band["Anders"] = "tuba"
Band["Ho"] = "flute"
Band["Moon"] = "tuba"
L = []
for name in Band:
    instrument = Band[name]
    L.append( (instrument, name) )
L.sort()
```

The inside set of parens makes a tuple; the outside set are the ones containing the arguments to the append method.

Programming - input

```
Lincecum, Tim  $9,000,000 SP
Posey, Buster $400,000 C 1B
Burriss, Emmanuel  $410,000 2B, SS
DeRosa, Mark $6,000,000  LF, 2B
Ross, Cody  $4,450,000 CF, RF, LF
Ishikawa, Travis  $417,000 1B
......
```

Output

- Lowest-paid player for each position

```
SP - Bumgarner, Madison $400,000
C - Posey, Buster $400,000
1B - Posey, Buster $400,000
2B - Burriss, Emmanuel  $410,000
.....
```

- What data structure do you want?

Data structure

- Dictionary – keys are positions, values are tuples of names and salaries.
- Produce output with for loop on dictionary.
- How to construct it?

Construction

- Extract name, salary. Convert salary to integer. Extract positions as a list

```
for position in posList:
        if position in posDict:
                tupe = posDict[position]
                if tupe[0] < salary:
                        continue
        # either not there or stored salary is larger
        newTupe = (salary,name)
        posDict[position] = newTupe
```

Input

```
1,Norris arrival
1,Arestide departure
1,Alvarez arrival
1,Tang arrival
2,Tang departure
2,Bioletti arrival
3,Norris departure
3,Green arrival
3,Bioletti departure
```

Output

```
Enter the name of a guest: Tang
Tang stayed for 1 nights.
Enter the name of a guest: Norris
Norris stayed for 2 nights.
Enter the name of a guest: Marz
Marz was not here during this period.
Enter the name of a guest: Green
Green stayed past the end of the period.
```

- Data structure?

Data structure

- Dictionary, keys are names, values are tuples with an integer and a code that is either "arrival", "departure" or "length", if the guest had both an arrival and a departure in the period.
- How to construct? One solution…
  - For each line, extract name, day, event (arrival or departure)
  - Change dictionary as needed (see next slide)
  - Then use dictionary to answer questions

Adding and changing dictionary

```
if event == "arrival":
        stayDict[name] = (day, "arrival")
else:  # departure
        if name in stayDict:
                tupe = stayDict[name]
                length = day - tupe[0]
                stayDict[name] = (length, "length")
        else:
                stayDict[name] = (day, "departure")
```

Thanks!

- Thank you all, and see you Monday!