Lists

We have seen lists already. [3, 2, 27, 54] is a list of 4 numbers. ["Eric", "Ginger", "Jack"] is a list of 3 strings. There is a lot more that can be done with lists. After strings, lists are the most important way to structure data.

In many situations where we use lists to store data we start with an empty list and build up the data one entry at a time.

```
L = [ ]
```

makes variable L into an empty list

```
L.append(x)
```

adds x as a new entry onto the end of L.

For example,

```
        L = [ ]
        L.append(23)
        L.append(14)
        L.append( 5)
```

makes L into the list [23, 14, 5]

Here is a program that reads a bunch of data, then prints it:

```
def main():
    L = [ ]
    done = False
    while not done:
        x = eval(input( ">>> " ))
        if x == 0:
            done = True
        else:
            L.append(x)
    for data in L:
        print( data )
```

What will this print?

```
def main():
        L = [ ]
        L.append(2)
        L.append(4)
        L[1] = L[0]+L[1]
        for x in L:
                print(x)
main()
```

A:
```
2
4
```

B:
```
2
4
6
```

C:
```
2
6
```

D: It crashes

What will this print?

```
def main():
        L = [ ]
        L.append(2)
        L.append(4)
        for x in L:
                x = L[0]+L[1]
        for x in L:
                print(x)
main()
```

A:
```
2
4
```

B:
```
2
4
6
```

C:
```
6
6
```

D: It crashes

Lists are usually easy to work with. The problems that come up tend to be with passing lists as arguments to functions.

The tricky places are generally due to confusions between the value of a list and the data the list contains. The *value* of a list is the location in memory where it is stored. Of course, you can't know where this is.

Lists are *mutable* structures because you can change the data that is stored in a list. If you start

```
        L = [1, 2, 3]
        L[1] = 45
```

then L becomes the list [1, 45, 3]

Strings are *immutable*; if S is "Bob Geitz" you can't say S[1] = 'u' . You could say S = "Doofus" + S[3:] but that makes a new string and assigns it to S.

So strings are immutable and lists are mutable.

This means the following program works as you would expect:

```
def add23ToList( L ):
        L.append(23)

def main():
        myList = [1, 2, 3]
        add23ToList( myList)
        print( myList )

main() # prints [1, 2, 3, 23]
```

This doesn't work the way you might expect:

```
def changeList( L ):
    L = [ 23]
def main():
    myList = [5 ]
    changeList( myList )
    print( myList )

main()  # prints [5 ]
```

Remember that changeList's L and main's L are different variables. Once changeList( ) assigns a new list to L, changeList's L and main's L refer to different lists.

What will this print?

```
def change(L):
        for i in range(0, len(L)):
                L[i] = 0

def main():
        L == [1,2,3]
        change(L)
        for x in L:
                print(x, end = " ")
        print( )
main( )
```

A: 0 0 0

B: 1 2 3

C: 0 2 3

D: It crashes

What will this print?

```
def change(L):
        L = [0,0,0]

def main():
        L == [1,2,3]
        change(L)
        for x in L:
                print(x, end = " ")
        print( )
main( )
```

A: 0 0 0

B: 1 2 3

C: 0 2 3

D: It crashes