CMPT 120
Lists and Strings

Summer 2012

Instructor: Hassan Khosravi

- All of the variables that we have used have held a single item
  - One integer, floating point value, or string

- often you find that you want to store a collection of values in your programs.
  - a list of values that have been entered by the user or a collection of values that are needed to draw a graph.
- In Python, lists can be used to store a collection of values.
- Python can hold values of any type; they are written as a comma-separated list enclosed in square brackets:
  - numlist = [23, 10, -100, 2]
  - words = [’zero’, ’one’, ’two’]
  - junk = [0, 1, ’two’, [1,1,1], 4.0]

- To get a particular value out of a list, it can be subscripted
  - testlist = [0, 10, 20, 30, 40, 50]
  - print testlist[2]
    - 20
  - print testlist[0]
    - 0
  - print testlist[10]
    - IndexError: list index out of range
  - Like strings, the first element in a list is element 0

- You can determine the length of a list with the len function:
  - print len(testlist)
    - 6

- for i in range(len(testlist)):
- print testlist[i],
    - 0 10 20 30 40 50

- Lists can be joined (concatenated) with the + operator:
  - testlist + [60, 70, 80]
    - [0, 10, 20, 30, 40, 50, 60, 70, 80]
  - [’one’, ’two’, ’three’] + [1, 2, 3]
    - [’one’, ’two’, ’three’, 1, 2, 3]

- It is also possible to delete an element from a list
- colours = [’red’, ’yellow’, ’blue’]
- del colours[1]
- print colours
  - [’red’, ’blue’]
- del colours[1]
- print colours
  - [’red’]

- You can also add a new element to the end of a list with the append
  - colours = [’red’, ’yellow’, ’blue’]
  - colours.append(’orange’)
  - colours.append(’green’)
  - print colours
    - [’red’, ’yellow’, ’blue’, ’orange’, ’green’]

- In order to do something similar with a string, a new string must be built with the + operator:
  - letters = ’abc’
  - letters = letters + ’d’
  - letters = letters + ’e’
  - print letters
    - abcde

- print "Enter some numbers, 0 to stop:"
  - numbers = []
  - x=1
  - while x!=0:
  - x = int(raw_input())
  - if x!=0:
  - numbers.append(x)
  - print "The numbers you entered are:"
  - print numbers

Lists and for loops

- range(10)
  - [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- range(1,17,3)
  - [1, 4, 7, 10, 13, 16]
- Range(13,1,2)
  - []
- Range(13,1,-2)
  - [13, 11, 9, 7, 5, 3]
- for i in range(10):
- # do something with I

- For loop in Python can iterate over any list
  - not just those produced by the range function.

- There, the for loop iterates over each element in the list words.

- words = ["up", "down", "green", "cabbage"]
- for word in words:
- print "Here's a word: " + word

  - Here's a word: up
  - Here's a word: down
  - Here's a word: green
  - Here's a word: cabbage

Slicing and Dicing

- colours = [’red’, ’yellow’, ’blue’]
- colours[1] = ’green’ # set an element with indexing
- Print colours[1]
  - green
- print colours[2] # index to retrieve an element
  - ’blue’

- colours = [’red’, ’yellow’, ’green’, ’blue’]
- print colours[1:3]
  - [’yellow’, ’green’]

- In general, the slice [a:b] extracts elements a to b-1
  - Same as range

Special Slice Positions

- Negative values count from the end of a list.
  - -1 refers to the last item in the list
  - -2 to the second-last, and so on

- colours = [’red’, ’yellow’, ’green’, ’blue’]
- print colours[0:-1]
  - [’red’, ’yellow’, ’green’]

- If you leave out one of the values in the slice, it will default to the start or end of the list.
  - [:num] refers to elements 0 to num-1.
  - [2:] gives elements from 2 to the end of the list.

Examples

- colours = [’red’, ’yellow’, ’green’, ’blue’, ’orange’]
- print colours[2:]
  - [’green’, ’blue’, ’orange’]
- print colours[:3]
  - [’red’, ’yellow’, ’green’]
- print colours[:-1]
  - [’red’, ’yellow’, ’green’, ’blue’]

- e [:-1] will always give you everything except the last element;
- [1:] will give everything but the first element.

Examples Fibonacci

- Store the first 20 values in the Fibonacci series in a list such that the index i of the list stores the ith element.
- Fibonacci series are the numbers in the following integer
- sequence:
- 0,1,1,2,3,5,…
- $F_n = F_{n-1} + F_{n-2}$
- $F_0 = 0$, $F_1 = 1$

Example Fibonacci

- fib = []
- fib.append('garbage')
- fib.append(0)
- fib.append(1)
- for i in range(3,20):
- `    `new = fib[i-1] + fib[i-2]
- `    `fib.append(new)
- print fib

Example Fibonacci

- Store the first 20 values in the Fibonacci series in a list such that the index i of the list stores the ith element.
- Fibonacci series are the numbers in the following integer
- sequence:
- 0,1,1,2,3,5,…
- $F_n = F_{n-1} + F_{n-2}$
- $F_0 = 0$, $F_1 = 1$

- For the odd elements of the series calculate their average
- For even elements of the series calcuate their sum

Example Fibonacci

```
def average(list):
    length = len(list)
    j=0
    sum=0
    for i in range(1,length,2):
        j=j+1
        sum = sum+list[i]
        print list[i]
    average = float(sum)/j
    return average
```

```
def sumforfib(list):
    length = len(list)
    sum=0
    for i in range(length,2):
        sum = sum+list[i]
        print list[i]
    return sum
```

Example Duplicate

- Store a list of 10 words and determine whether any word has been entered more than once.

```
words= []
for i in range(10):
    words.append( raw_input("enter word please: "))

flag = False
for i in range(10):
    for j in range(i+1,10):
        if words[i] == words[j]:
            print "duplicated"
            flag = True

if flag == False:
    print "No duplicates"
```

Words containing ‘s’

- Store a list of 10 words and determine whether each contains the letter s or not.

```
words= []
for i in range(10):
    words.append( raw_input("enter word please: "))
```

```
for w in words:
    flag = False
    for let in w:
        if let == 's':
            print w, "contains s"
            flag = True
    if flag == False:
        print w, "does not contain s"
```

- Store a list of 10 words and determine whether each contains the letter s or not. Remove all occurrences of letter s from all words

```
words= []
for i in range(4):
    words.append( raw_input("enter word please: "))

length = len(words)
for i in range(length):
    temp = ""
    for let in words[i]:
        if let != 's':
            temp = temp + let
    words[i]= temp
print words
```

Manipulating Slices

- You can actually do almost anything with list slices
  - colours = [’red’, ’yellow’, ’green’, ’blue’]
  - colours[1:3] = [’yellowish’, ’greenish’]
  - print colours
    - [’red’, ’yellowish’, ’greenish’, ’blue’]
  - colours[1:3] = [’pink’, ’purple’, ’ecru’]
  - print colours
    - [’red’, ’pink’, ’purple’, ’ecru’, ’blue’]
- we assigned a list of three elements to a slice of length two
  - The list expands to make room or the new elements
  - [’yellowish’, ’greenish] is replaced with [’pink’, ’purple’, ’ecru’].

Manipulating Slices

- If the list assigned is shorter than the slice, the list would shrink
- colours = ['red', 'yellow', 'green', 'blue']
- colours[1:3] = "red"
- print colours
  - ['red', 'r', 'e', 'd', 'blue']

- colours = ['red', 'yellow', 'green', 'blue']
- colours[1:3] = ['red']
- print colours
  - ['red', 'red', 'blue']

Deleting Slices

- You can also remove any slice from a list:
- colours = [’red’, ’yellow’, ’green’, ’blue’]
- del colours[1:3]
- print colours
  - [’red’, ’blue’]

Strings

- Strings are a sequence of characters; lists are a sequence of any combination of types.

- Any type that represents a collection of values can be used as the “list” in a for loop.
  - Since a string represents a sequence of characters, it can be used.

- for char in "abc":
- print "A character:", char

  - A character: a
  - A character: b
  - A character: c

Slicing Strings

- Slices in strings are read only

- sentence = "Look, I’m a string!“
- print sentence[:5]
  - Look
- print sentence[6:11]
  - I’m a
- print sentence[-7:]
  - string!

- But, you can’t modify a string slice

- sentence = "Look, I’m a string!“
- sentence[:5] = "Wow"
  - TypeError: object doesn’t support slice assignment
- del sentence[6:10]
  - TypeError: object doesn’t support slice assignment

Mutability

- dots = dots + "." # statement #1
  - The right side of = is evaluated and put into dots. The old value of dots is lost in the assignment
- values = values + [n] # statement #2
  - Same with this method of modifying lists
- values.append(n) # statement #3
  - The output of this statement is the same as statement 2.
    - Statement 3 requires a lot less work.
    - The whole list in not rebuilt.

- Data structures that can be changed in-place like lists are called mutable.
- Strings and numbers are not mutable: they are immutable.
- Objects depends how they are written, but they have the capability to be mutable

References

- There are several cases where the contents of one variable are copied to another.

- # x copied into y:
- y = x

- You probably don't think of copying the contents of a variable as a difficult operation, but consider the case where x is a list with millions of elements.
- Python avoids making copies where possible

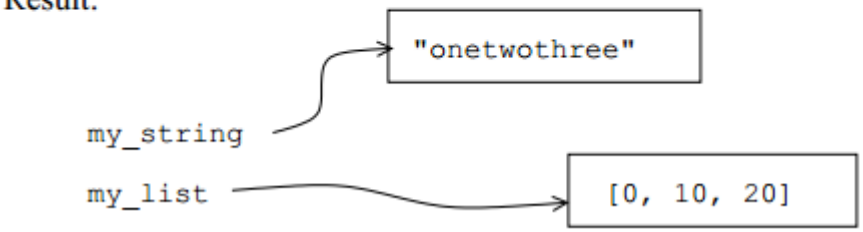
- To understand this, we need to understand references.

- Every variable in Python is actually a reference to the place in memory where its contents are stored.
- Conceptually, you should think of a variable referencing its contents like an arrow pointing to the contents in memory.

Statements:

```
my_string = "one" + "two" + "three"
my_list = [0, 10, 20]
```

Result:



- When you use a variable in an expression, Python follows the reference to find its contents.
- **Usually**, the expression on the right side of an assignment creates a new object in memory.
  - Total = a+b calculates a+b, stores this in memory, and sets total to reference that value.
- The exception to this is when the right side of an assignment is simply a variable reference
  - (like total=a).
  - In this case, the result is already in memory and the variable can just reference the existing contents

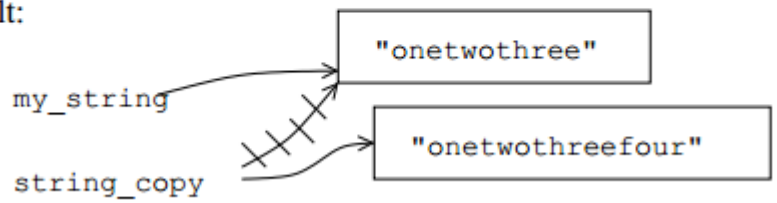
Aliases

- When two variables refer to the same contents, they are aliases of each other.
  - it's generally good since it doesn't require copying the contents to another location in memory.

Statements:

```
my_string = "one" + "two" + "three"
string_copy = my_string
string_copy = string_copy + "four"
```

Result:



- When you assign to a variable, you are changing it so the variable now references different contents.
  - (The old contents are thrown away since they are no longer being referenced.)

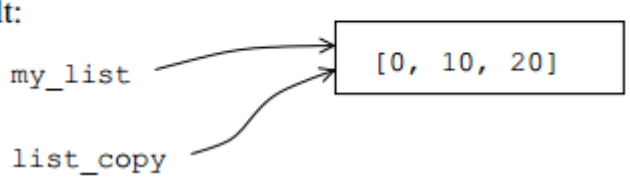
Mutable data structures and aliases

- Mutable data structures (lists and some objects), aliases complicate things.
  - Mutable data structures can be changed without totally rebuilding them, we can change the contents without moving the reference to a new object in memory
- It's possible to change a variable, and the changes will affect any other variables that reference the same contents.
  - my_list and list_copy are aliases of the same contents. When either one is changed, both are affected. I

Statements:

```
my_list = [0, 10, 20]
list_copy = my_list
```

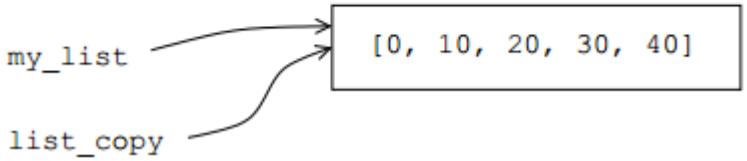
Result:



Statements:

```
my_list = [0, 10, 20]
list_copy = my_list
list_copy.append(30)
my_list.append(40)
```

Result:

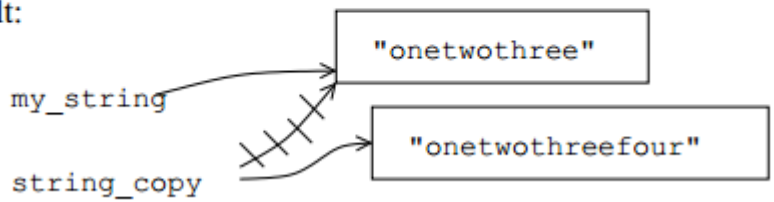


- Any expression (that's more complicated than a variable reference) will result in a new reference being created. If this is assigned to a variable, then there is no aliasing.

Statements:

```
my_string = "one" + "two" + "three"
string_copy = my_string
string_copy = string_copy + "four"
```

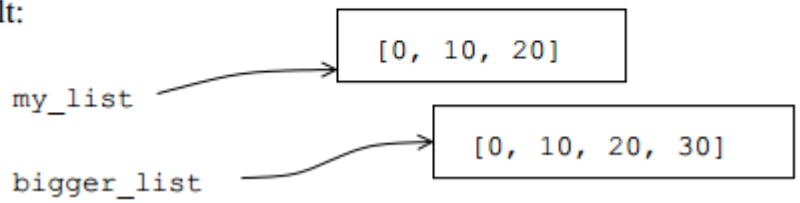
Result:



Statements:

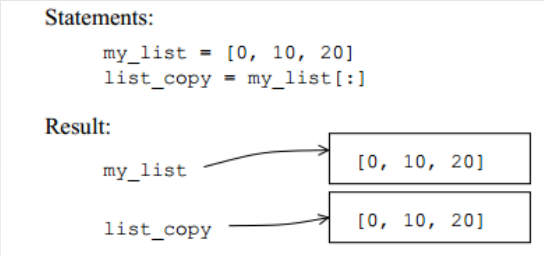
```
my_list = [0, 10, 20]
bigger_list = my_list + [30]
```

Result:



Really Copying

- If you want to make a copy of a variable that isn't a reference, it's necessary to force Python to actually copy its contents to a new place in memory. This is called cloning. Cloning is more expensive than aliasing,
- There are three methods for this
  - The slice operator can be used to create a clone.



  - You could also make a copy of a list with the list function that creates a new list (out of the old one). So, list(my_list) would give the same result as my_list[:].

  - the copy module contains a copy function. This function will take any Python object and clone its contents
    - import copy
    - new_obj = copy.copy(obj)