Data Structure and Iteration

Lecture 3

Python List

1 variable store 1 value?

Most of our variables have one value in them - when we put a new value in the variable, the old value is overwritten

```
$ python
>>> x = 2
>>> x = 4
>>> print(x)
4
```

A List is a Kind of Collection

- A collection allows us to put many values in a single “variable”
- A collection is nice because we can carry all many values around in one convenient package.

```
friends = [ ‘Alice’, ‘Max', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

List

- List is surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

Get a single element

- We can get any single element in a list using an index specified in square brackets
- The index value must be an integer and starts at zero

Alice	Max	Sally
0	1	2

```
>>> friends = [ ‘Alice', ‘Max', 'Sally' ]
>>> print(friends[1])
Max
>>> print(friends[-1])
Sally
```

Get a sequence of elements

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: the second number is “up to but not including”

Lists are Mutable

- Lists are “mutable” - we can change an element of a list using the index operator

```
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

How Long is a List?

- The len() function takes a list as a argument and returns the number of elements in the list

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
```

Concatenating Lists Using +

We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Create a List

- We can create an empty list and then add elements using the append method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Is Something in a List?

- in not in
- These are logical operators that return True or False
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

Other List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

http://docs.python.org/tutorial/datastructures.html

A Collection is nice :

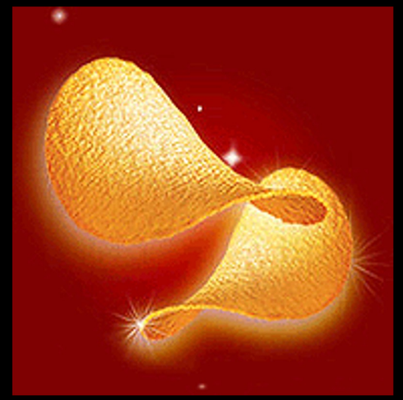
- A sequence of values in a single “variable”
- More than one place “in” the variable
- Access to the different places in the variable

Python Dictionary

A Story of Two Collections..

- List

  - A linear collection of values that stay in order



- Dictionary

  - A “bag” of values, each with its own label

Dictionaries

http://en.wikipedia.org/wiki/Associative_array

Dictionary

- A set of key : value pairs specified in curly brackets

```
>>> bag = { ‘pencil’ : 1, ‘calculator’ : 1 , ‘candy’: 4}
>>> print(bag)
{‘pencil’ : 1, ‘calculator' : 1 , ‘candy’: 4}
>>> print(bag[‘pencil’])
2
```

Lists & Dictionaries

Dictionaries are like lists except that they use keys instead of numbers to look up values

```
lst = [1,1,4]
```

Key	Value
[0]	1
[1]	1
[2]	4

```
print(lst[2])
```

?

```
dico = {‘pencil’: 1,
‘calculator’: 1 ,‘candy’: 4}
```

Key	Value
[‘candy']	4
[‘calculator']	1
[‘pencil']	1

```
print(dico[“candy”])
```

?

Add new elements

`lists`: append( )

```
>>> l = list()
>>> l.append(21)
>>> l.append(4)
>>> print(l)
[21, 4]
>>> l[0] = 23
>>> print(l)
[23, 4]
```

`dict`: assign to a new key

```
>>> d = dict()
>>> d['age'] = 21
>>> d['course'] = 4
>>> print(d)
{'course': 4, 'age': 21}
>>> d['age'] = 23
>>> print(d)
{'course': 4, 'age': 23}
```

len() function

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>> d = {‘pencil’: 1,‘calculator’: 1 ,‘candy’: 4}
>>> print(len(d))
3
```

Operators: `in` & `not in`

`Lists:` list element?

```
>>> some = [1, 9, 21, 10]
>>> 9 in some
True
>>> 20 not in some
True
```

Dictionaries: dict key?

```
>>> dd = {'pencil': 1,
'calculator': 1 ,'candy': 4}
>>> 9 in dd
False
>>> 'candy' not in dd
False
```

Retrieving Lists of Keys and Values

You can get a list of keys, values, or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
['jan', 'chuck', 'fred']
>>> print(list(jjj.keys()))
['jan', 'chuck', 'fred']
>>> print(list(jjj.values()))
[100, 1, 42]
>>> print(list(jjj.items()))
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Loops and Iteration

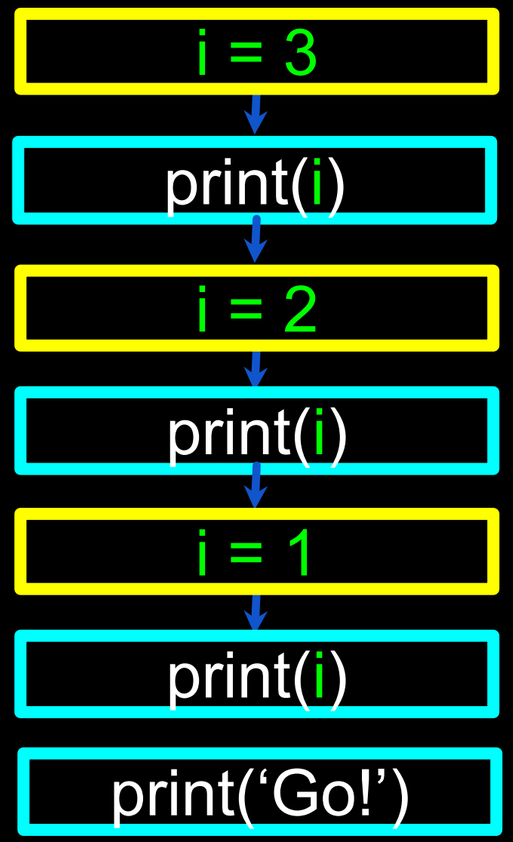
`for` structure: Iterating over a set of items…

A Simple `for` Loop

Iteration variable

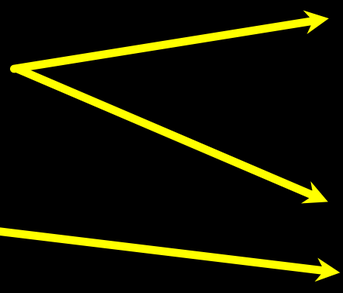
```
for i in [3, 2, 1] :
    print(i)
print('Go!')
```

```
3
2
1
Go!
```



Lists and `for` Loops - Best Pals

```
friends = [‘Alice', ‘Max', 'Sally' ]
for friend in friends :
    print('Happy New Year:',  friend)
print('Done!')
```



Happy New Year: Alice
Happy New Year: Max
Happy New Year: Sally
Done!

```
friends = [‘Alice', ‘Max', 'Sally' ]
for x in friends:
    print('Happy New Year:',  x)
print('Done!')
```

Using the range() Function

- Returns a list of numbers that range from zero to one less than the argument
- We can construct an index loop using `for`

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = [‘Alice', ‘Max', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

A Tale of Two Loops...

```
friends = [‘Alice', ‘Max', 'Sally']

for friend in friends :
    print('Happy New Year:',  friend)

for i in range(len(friends)) :
    f = friends[i]
    print('Happy New Year:',  f)
```

```
>>> friends = [‘Alice', ‘Max', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
>>>
```

Happy New Year: Alice

Happy New Year: Max

Happy New Year: Sally

Loop Idioms:
What We Do in Loops

Note: Even though these examples are simple, the patterns apply to all kinds of loops

Making “smart” loops

Set some variables to initial values

for item in data:

1. Do sth to each item separately
2. Updating variables

Look at the variables

Looping Through a Set

```
print('Before')
for item in [9, 41, 12, 3, 74, 15] :
     print(item)
print('After')
```

```
$ python basicloop.py
Before
9
41
12
3
74
15
After
```

What is the Largest Number?

What is the Largest Number?

3

What is the Largest Number?

41

What is the Largest Number?

12

What is the Largest Number?

9

What is the Largest Number?

74

What is the Largest Number?

15

What is the Largest Number?

What is the Largest Number?

3 41 12 9 74 15

What is the Largest Number?

largest_so_far -1

What is the Largest Number?

3

largest_so_far 3

What is the Largest Number?

41

largest_so_far 41

What is the Largest Number?

12

largest_so_far 41

What is the Largest Number?

9

largest_so_far 41

What is the Largest Number?

74

largest_so_far 74

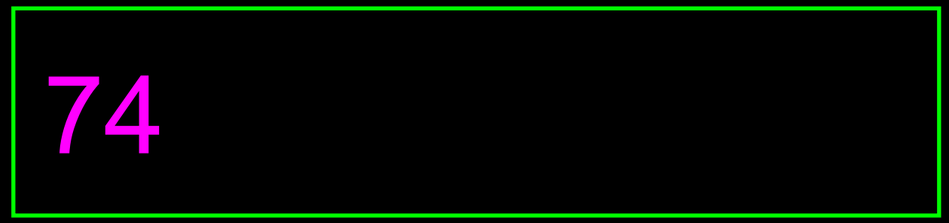
What is the Largest Number?

15

74

What is the Largest Number?

3 41 12 9 74 15



Finding the Largest Value

```
largest_so_far = -1
print('Before', largest_so_far)
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print(largest_so_far, the_num)

print('After', largest_so_far)
```

```
$ python largest.py
Before -1
9  9
41  41
41 12
41  3
74  74
74  15
After 74
```

More Loop Patterns…

Counting in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + 1
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
1 9
2 41
3 12
4 3
5 74
6 15
After 6
```

Summing in a Loop

```
zork = 0
print('Before', zork)
for thing in [9, 41, 12, 3, 74, 15] :
    zork = zork + thing
    print(zork, thing)
print('After', zork)
```

```
$ python countloop.py
Before 0
9 9
50 41
62 12
65 3
139 74
154 15
After 154
```

To add up a value we encounter in a loop, we introduce a sum variable that starts at 0 and we add the value to the sum each time through the loop.

Filtering in a Loop

```
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
        print('Large number',value)
print('After')
```

```
$ python search1.py
Before
Large number 41
Large number 74
After
```

We use an if statement in the loop to catch / filter the values we are looking for.

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...