Lect 4 – Strings, Lists, Tuples

Rob Capra

INLS 490-172

Strings

- Concatenation operator: +

```
fruit = "banana"
bakedGood = " nut bread"
print(fruit + bakedGood)
```

Strings

- Index operator: []

```
fruit = "banana"
print(fruit[2])
print(fruit[-2])
```

0	1	2	3	4	5
b	a	n	a	n	a
-6	-5	-4	-3	-2	-1

Strings

- Index operator: []

```
fruit = "banana"
print(fruit[2])
print(fruit[-2])
```

0	1	2	3	4	5
b	a	n	a	n	a
-6	-5	-4	-3	-2	-1

Strings

- String Methods

```
ss = "Hello, World"
print(ss.upper())

tt = ss.lower()
print(tt)
print(ss)
```

String Methods

Method	Parameters	Description
upper	none	Returns a string in all uppercase
lower	none	Returns a string in all lowercase
capitalize	none	Returns a string with first character capitalized, the rest lower
strip	none	Returns a string with the leading &trailing whitespace removed
lstrip	none	Returns a string with the leading whitespace removed
rstrip	none	Returns a string with the trailing whitespace removed
count	item	Returns the number of occurrences of item
replace	old, new	Replaces all occurrences of old substring with new
center	width	Returns a string centered in a field of width spaces
ljust	width	Returns a string left justified in a field of width spaces
rjust	width	Returns a string right justified in a field of width spaces
find	item	Returns the leftmost index where the substring item is found
rfind	item	Returns the rightmost index where the substring item is found
index	item	Like find except causes a runtime error if item is not found
rindex	item	Like rfind except causes a runtime error if item is not found

String Length

- Watch out!

```
fruit = "Banana"
sz = len(fruit)
# Which of the next two lines?
lastch = fruit[sz]
lastch = fruit[sz-1]
print(lastch)
```

Slice Operator

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
h	t	t	p	:	/	/	w	w	w	.	u	n	c	.	e	d	u	/	

- Returns a substring

```
url = "http://www.unc.edu/"
print url[0:4]
print url[7:]     # [7:19]
print url[:4]     # [0:4]
print url[7:-1]
```

Strings are Immutable

- [] cannot be used with a string on LHS of =

```
s = "abc"
s[2] = "z"       # ERROR!
print s
```

String Traversal

- Example using `for..in`

```
s = "UNC Tar Heels"
for achar in s:
    print achar
```

String Traversal by Index

- Example using `for..range`

```
s = "UNC Tar Heels"
for i in range(len(s)):
    print s[i]
```

String Traversal using While

- Example using `while`

```
s = "UNC Tar Heels"
i = 0
while i < len(s):
    print s[i]
    i = i + 1
```

When would you use each method?

a) for..in?
b) for..range?
c) while?

`in` and `not in` operators

- Does one string occur in another string?

```
print 'i' in 'teamwork'
print 'unc' in 'uncle'
print 'duke' in 'unc'

print 'z' not in 'unc'
```

Accumulator Pattern: Strings

```
def removeVowels(s):
    vowels = "aeiouAEIOU"
    snov = ""
    for curchar in s:
        if curchar not in vowels:
            snov= snov + curchar
    return snov

print(removeVowels("compsci"))
print(removeVowels("aAbEefIijOopUus"))
```

Lists

- List is a sequential collection of data
- Each value is identified by an index
- Values are called the elements of the list
- Elements can be any data type
- Elements in a list can be of different data types

```
a = [10, 20, 30]
b = ["unc", "duke", "ncstate"]
c = ["luke", 3.14, [10, 20], 50]
```

List Length

- Len returns length of top-most list

```
a = [10, 20, 30]
b = ["unc", "duke", "ncstate"]
c = ["luke", 3.14, [10, 20], 50]
print len(a)
print len(b)
print len(c)
```

Accessing Elements

- Use the index operator: []
- Indices start a zero

```
a = [10, 20, 30]
b = ["unc", "duke", "ncstate"]
c = ["luke", 3.14, [10, 20], 50]
print a
print a[1]
print c[1]
print c[2][1]
```

in and **not in** with lists

- List membership test

```
accc = ["duke", "gatech", "miami",
"unc", "pitt", "uva", "vatech"]

print "unc" in accc
print "texas" in accc
print "uva" not in accc
```

List slices and concatentation

- List membership test

```
accc1 = ["duke", "gatech", "miami"]
accc2 = ["unc", "pitt", "uva", "vatech"]
print accc1 + accc2
print accc1[1:3]
print (accc1 + accc2)[2:6]
```

List are Mutable

- Can change a list as part of an LHS assignment

```
accc = ["duke", "gatech", "miami", "unc",
"pitt", "uva", "vatech"]
accc[6] = "vt"
```

List Deletion

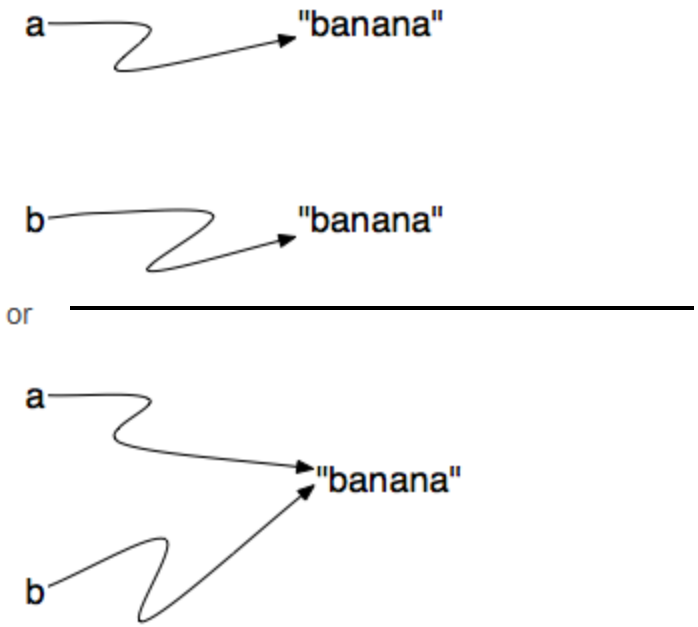
- Del statement

```
accc = ["duke", "gatech", "miami", "unc",
"pitt", "uva", "vatech"]
del accc[0]
print accc
```

Objects and References

- Do a and b below point to the *same* **string**?

```
a = "banana"
b = "banana"

print (a is b)

print (a == b)
```

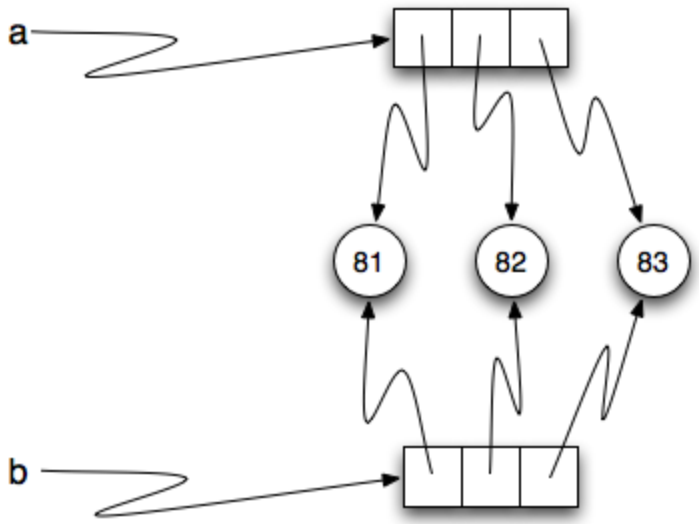


Recall: strings are immutable

Objects and References

- Do a and b below point to the *same* **list**?

```
a = [81,82,83]
b = [81,82,83]

print(a is b)

print(a == b)
```

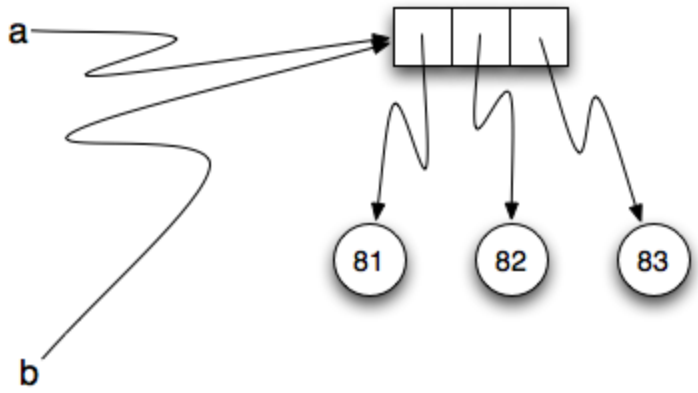


Recall: lists are mutable

Aliasing

- What if we assign a variable to another?

```
a = [81, 82, 83]
b = a
print(a is b)

a[0] = 71
print b

b[-1] = 73
print a
print b
```



Changes to one affect the other

Cloning (copying)

- We can clone a list

```
a = [81, 82, 83]
b = a[:]   # clone by slice
print (a == b)
print(a is b)

a[0] = 71
print b

b[-1] = 73
print a
print b
```

Changes to one DO NOT affect the other

List Methods

Method	Parameters	Result	Description
`append`	item	mutator	Adds a new item to the end of a list
`insert`	position, item	mutator	Inserts a new item at the position given
`pop`	none	hybrid	Removes and returns the last item
`pop`	position	hybrid	Removes and returns the item at position
`sort`	none	mutator	Modifies a list to be sorted
`reverse`	none	mutator	Modifies a list to be in reverse order
`index`	item	return idx	Returns the position of first occurrence of item
`count`	item	return ct	Returns the number of occurrences of item
`remove`	item	mutator	Removes the first occurrence of item

List Methods

- Examples

```
mylist = []
mylist.append("duke")
mylist.append("unc")
mylist.append("ncstate")
print(mylist)

mylist.insert(1, "vt")
print(mylist)
print(mylist.count(12))

print(mylist.index("unc"))
print(mylist.count(5))
```

```
mylist.reverse()
print(mylist)

mylist.sort()
print(mylist)

mylist.remove("duke")
print(mylist)

lastitem = mylist.pop()
print(lastitem)
print(mylist)
```

List method return values

- Return **`None`**: append, sort, reverse

```
#probably an error
mylist = mylist.sort()
```

List traversal

- Similar options to string traversal

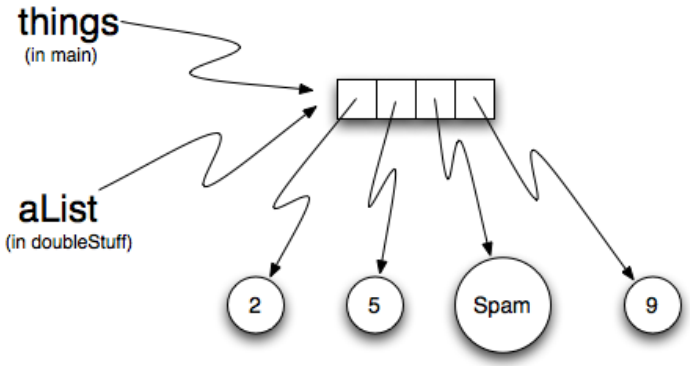
```
schools = ["unc", "duke", "ncstate"]
for school in schools:
    print school

for i in range(len(schools)):
    print schools[i]

i = 0
while i < len(schools):
    print schools[i]
    i = i + 1
```

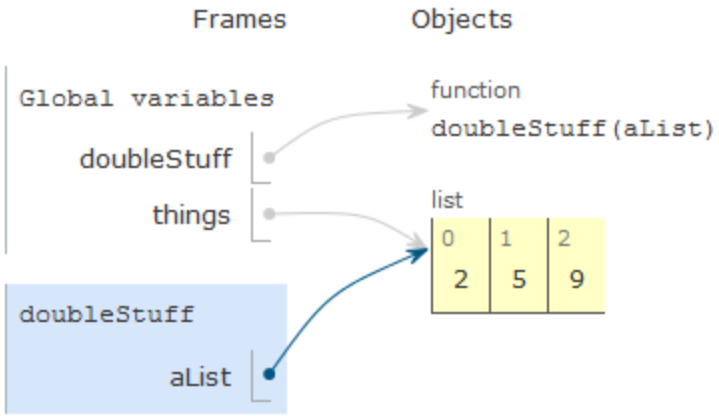
List Parameters

- When a list is used as an argument to a function, Python passes a **reference** to the list, not a copy.

```
def doubleStuff(a):
    for position in range(len(a)):
        a[position] = 2 * a[position]
```

```
b= [2, 5, 9]
print(b)
doubleStuff(b)
print(b)
```

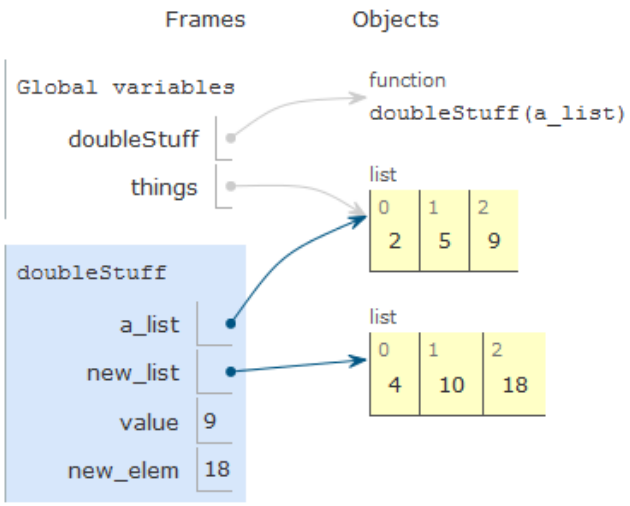




Pure Functions

- Do not produce *side effects*
- Parameters are not modified

```
def doubleStuff(a_list):
    new_list = []
    for value in a_list:
        new_elem = 2 * value
        new_list.append(new_elem)
    return new_list
```



Pure vs. Mutator/Modifier

- Which is better?

List creation pattern

- Another useful pattern:

```
init a result var to be an empty list
loop
    create a new element
    append it to result
return the result
```

Map

- Map a function onto the elements of a sequence

```
def capitalize_all(t):
    result = []
    for s in t:
        result.append(s.capitalize())
    return result

a = ["mary", "john", "sally"]
print capitalize_all(a)
```

Filter

- Filter selects some elements and filters out others

```
def only_small(t):
    result = []
    for s in t:
        if s < 50:
            result.append(s)
    return result

a = [10, 20, 80, 90]
print only_small(a)
```

Reduce

- Combine a sequence of elements into a single value

```
def add_all(t):
    result = 0
    for s in t:
        result += s
    return result

a = [10, 20, 80, 90]
print add_all(a)
```

List Comprehensions

- Create a list from a sequence based on a condition

```
[<expr> for <item> in <seq> if <cond>]
```

```
a = [10, 20, 80, 90]
b = [n*2 for n in a]
print b
```

```
a = [10, 20, 80, 90]
b = [n*2 for n in a if n>50]
print b
```

very common in Python

Tuples

- Tuples are similar to lists, but are **immutable**

```
t = 'a', 'b', 'c'    #ok
t = ('a', 'b', 'c') #typical
print type(t)
t = 'a' #str
print type(t)
t = ('a')    #str
t = tuple()
t = tuple('unc')
print t
```

Tuple operators

- Brackets and slices work similar to lists
- But tuples are immutable

```
t = ('a', 'b', 'c', 'd', 'e')
print t[2]
print t[1:3]
t[2] = 'z'   #error
```

Tuple assignment

- Use tuples to swap variables without a temp var

```
temp = a
a = b
b = temp

a, b = b, a        #tuple assignment

addr = 'santa@northpole.org'
uname, domain = addr.split('@')
```

Tuple as return values

- Functions can use tuples to return multiple values

```
t = divmod(7, 3)
print t
print type(t)

quot, rem = divmod(7, 3)
print quot
print rem

def min_max(t):
    return min(t), max(t)
print min_max([1, 3, 5, 7])
```

Variable-length arguments

- Prefix param name with a * to gather args into a tuple
- Gather & scatter

```
def printall(*args):  #gather
    print args
printall(1, 2, 3)
t = (7, 3)
print divmod(t)  #error
print divmod(*t)  #scatter
```

Lists of tuples & zip

- Zip takes 2 or more sequences and “zips” them into a list of tuples

```
s = 'abc'  #string
t = [0, 1, 2]  #list
z = zip(s, t)
print z
for aletter, anumber in z:  #tuple assignment
    print aletter, anumber
```