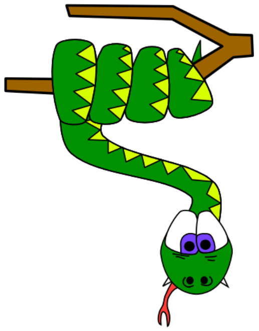


List, Tuple & Set

Session-V

What is a List?

- An ordered set of values:
  - Ordered: 1st, 2nd, 3rd, …
  - Values: can be anything, integers, strings, other lists
- List values are called *elements*.
- A *string* is an ordered set of characters so it is “like” a list but not exactly the same thing.

The Empty List

```
x = [ ]
```

- The empty list is usually used to initialize a list variable but not give it any useful elements.

Accessing Elements:

- List elements are accessed via integer indexes starting at 0 and working up.

```
numbers = [ 3, 87, 43]
print(numbers[1], numbers[2], numbers[0])
87 43 3

x = 3
print(numbers[x-2])
87

print (numbers[1.0])
TypeError: sequence index must be integer

print (numbers[3])
TypeError: list index out of range

print (numbers[-1])     # a negative index counts
back
3                      # from the end of the list
                       # index -1 is the last element

print (numbers[-3])
```

Accessing Many Elements:

- By index value, one at a time (called *list traversal*)

```
# list of a known size
horsemen = ['war', 'famine', 'pestilence', 'death']
i = 0
while i < 4:
    print (horsemen[i])
    i = i + 1

# or if you don't know how long the list is
i = 0
length = len(horsemen)
while i < length:
    print( horsemen[i])
    i = i + 1

war
famine
pestilence
death
```

always safer to use *len* as an upper bound

always I < length; never I <= length

List Membership:

- You simply ask if a value is *“in”* or *“not in”* a list.
- This is always a *True/False* question.

```
horsemen = [‘war’, ‘famine’, ‘pestilence’, ‘death’]
if ‘debauchery’ in horseman:
    print( ‘There are more than 4 horsemen of the apocolipse.)

print( ‘debauchery’ not in horsemen)
1
```

List Operations:

- Add two lists:

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

- Repeat a list many times:

```
a = [1, 2, 3]
print (a*3)
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- Exercise: Create a list of 20 zeros.

```
zeros = [0]*20
```

List Slices:

- Sometimes you just want a sub-list (*slice*) of a list.

```
list[a:b] means

   list[a], list[a+1], …, list[b-1]

# all list elements with indexes from a to b;
# including a and excluding b
                              vowels = [‘a’, ‘e’, ‘i’, ‘o’, ‘u’]
                              print (vowels[2:4])
                              [‘i’, ‘o’]

                              # how do you print out the last element?
                              print (vowels[2:])
                              [‘i’, ‘o’, ‘u’]
```

- Exercise: What does *vowels[:3]* mean?

[‘a’, ‘e’, ‘i’]

Lists are *Mutable*

```
fruit = [‘apple’, ‘orange’, ‘pear’]
fruit[1] = ‘fig’
print fruit
[‘apple’, ‘fig’, ‘pear’]
```

List Slices Used to Modify a List:

- Suppose you are keeping an ordered list:

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
```

- And you want to add *kate*. Assignment doesn’t work!

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[2] = ‘kate’

print (names)
[‘adam’, ‘carol’, ‘kate’, ‘margot’, ‘phil’]
```

- You can add an element by squeezing it into an empty slice between two list elements:

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[2:2] = ‘kate’

print (names)
[‘adam’, ‘carol’, ‘henry’,  ‘kate’, ‘margot’, ‘phil’]
```

Starting at index 2
but not including 2;
ie, empty

List Deletion:

- Using the *del* operator

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
del names[3]

print (names)
 [‘adam’, ‘carol’, ‘henry’, ‘phil’]
```

- Replacing an element with an empty list

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
names[3:4] = [ ]

print (names)
 [‘adam’, ‘carol’, ‘henry’, ‘phil’]
```

- Deleting slices

```
names = [‘adam’, ‘carol’, ‘henry’, ‘margot’, ‘phil’]
del names[1:4]

print (names)
 [‘adam’, ‘phil’]
```

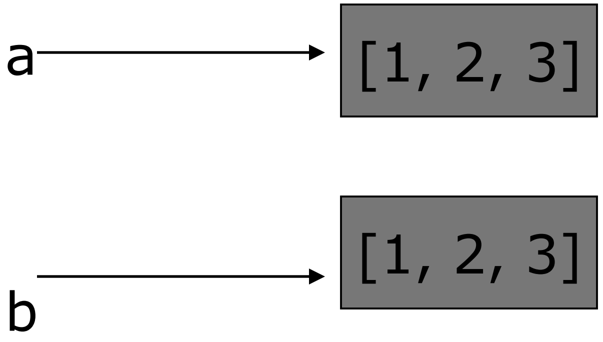
Lists, Objects and Values

- Lists are different:

```
a = [1, 2, 3]
b = [1, 2, 3]

print( id(a), id(b))
135023431   135024732
```

- So this time the memory state picture is:



Aliasing

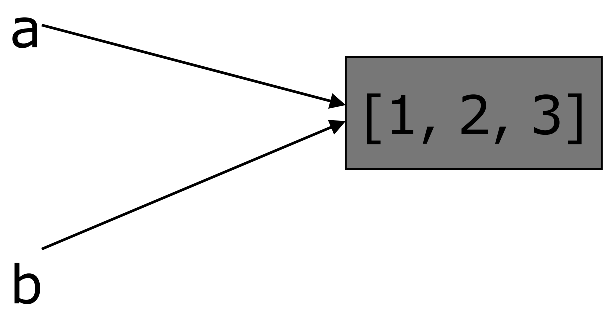
- However, if we assign one variable to another:

```
a = [1, 2, 3]
b = a

print( id(a), id(b))

135023431 135023431
```

- So this time the memory state picture is:



- More importantly, changing *b* also changes *a*

```
b[0] = 0
print a
[0, 2, 3]
```

Cloning a List:

- Cloning means making an exact but separate copy:
- Not Cloning:

```
a = [1, 2, 3]
b = a

print (id(a), id(b))
135023431 135023431
```

- Cloning:

```
a = [1, 2, 3]
b = a[:]              # slices are always separate lists

print (id(a), id(b))
135023431 13502652
```

List Methods:

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort (order) the list. A comparison function may be passed as a parameter.
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the ith element of the list and returns its value.

Adding Elements

```
"""List are mutable objects represented
 by []list is an ordered collection"""
l=[]
print(l)
"""append adds an item at the end of
list"""
l.append(9)
l.append("hello")
l.append(75.2)
print(l)
"""insert adds item at a given
 index-insert(index,object)"""
l.insert(0,90)
print(l)
```

Adding Elements

```
l1=[78,23]
"""extend list by appending
elements from object"""
l.extend(l1)
print(l)
l.extend("abc")
print(l)
l.extend([32,"abc",35.5])
print(l)
"""append can be used to create
 nested list"""
l1=[34,56]
l1.append([23,34])
print(l1)
```

More Operations..

```
l=[92,56,12,78,1]
print(l.count(92))
#sort() the list in increasing order of elements"""
l.sort()#doesn’t return object -inplace sorting
print(l)
#sort(reverse=True sort the list in non increasing order"""
l.sort(reverse=True)
print(l)
l.reverse()
print(l)
```

Remove/del

```
l=[23,"bill",67, 89, 90,"abc", "xyz"]
"""remove searches for an element in list and
deletes it"""
l.remove("abc")
print(l)
del(l[2])
print(l)
del(l[2:])
print(l)
l.extend([34,4,67,90])
print(l)
#del l[2:4]
del(l[:-1])
print(l)
del l[:]#removes all elements but not list
print(l)
"""delete list"""
del l
print(l)#NameError: name 'l' not defined
```

Index/pop

```
l=["abc",30,50]
x=l.index(30)
print(x)
"""pop function by default removes last element
--pop(index)"""
numbers.pop()
print(numbers)
x=numbers.pop(0)
print(numbers)
x=len(numbers)
print(x)
x=max(numbers)
print(x)
x=min(numbers)
print(x)
```

Sorted

- `l=[5,4,3,2,1]`
- `p=sorted(l)`
- `print(p)`

Array

```
l=array('i')
l.append(90)
l.append(40)
print(l)
#l.append(53.25)
print(l)
l[0]=100
print(l)
```

More…

```
s=array('u',"hello")
print(s)
for ch in s:
    print(ch)
s.reverse()
print(s)
s.pop()
print(s)
s.index("l")
x=sorted(s)
print(x)
p=array('i')
for x in range(1,6):
    p.append(x)
print(p)
```

typecodes

Type code	C Type	Python Type	Minimum size in bytes
`'b'`	signed char	int	1
`'B'`	unsigned char	int	1
`'u'`	Py_UNICODE	Unicode character	2
`'h'`	signed short	int	2
`'H'`	unsigned short	int	2
`'i'`	signed int	int	2
`'I'`	unsigned int	int	2
`'l'`	signed long	int	4
`'L'`	unsigned long	int	4
`'q'`	signed long long	int	8
`'Q'`	unsigned long long	int	8
`'f'`	float	float	4
`'d'`	double	float	8

Tuples

- Same as lists but
  - Immutable
  - Enclosed in parentheses
  - A tuple with a single element ***must*** have a comma inside the parentheses:
    - **a = (11,)**

Examples

- `>>> mytuple = (11, 22, 33)`

-
  ```
  >>> mytuple[0]
  11
  ```

-
  ```
  >>> mytuple[-1]
  33
  ```

-
  ```
  >>> mytuple[0:1]
  (11,)
  ```
- The comma is required!

Why?

- No confusion possible between **[11]** and **11**

- **(11)** is a perfectly acceptable expression
  - **(11) without the comma** is the integer 11
  - **(11,) with the comma** is a tuple containing the integer 11

Tuples are immutable

```
>>> mytuple = (11, 22, 33)
>>> saved = mytuple
>>> mytuple += (44,)
>>> mytuple
(11, 22, 33, 44)
>>> saved
(11, 22, 33)
```

Things that do not work

- **mytuple += 55**
  Traceback (most recent call last):Z
  ...
  TypeError:
  can only concatenate tuple (not "int") to tuple
  - Can understand that!

Sorting tuples

```
>>> atuple = (33, 22, 11)
>>> atuple.sort()
Traceback (most recent call last):
...
AttributeError:
'tuple' object has no attribute 'sort'
>>> atuple = sorted(atuple)
>>> atuple
[11, 22, 33]
```

Tuples are immutable!

sorted( ) returns a list!

Most other things work!

- `>>> atuple = (11, 22, 33)`

-
  ```
  >>> len(atuple)
  3
  ```

-
  ```
  >>> 44 in atuple
  False
  ```

-
  ```
  >>> [ i for [i for i in atuple]
  [11, 22, 33]
  ```

Converting sequences into tuples

```
>>> alist = [11, 22, 33]
>>> atuple = tuple(alist)
>>> atuple
(11, 22, 33)
>>> newtuple = tuple('Hello World!')
>>> newtuple
('H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!')
```

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1,2,3) : print (x, end = ' ')	1 2 3	Iteration

```
T=('C++', 'Java', 'Python)
```

Python Expression	Results	Description
T[2]	'Python'	Offsets start at zero
T[-2]	'Java'	Negative: count from the right
T[1:]	('Java', 'Python')	Slicing fetches sections

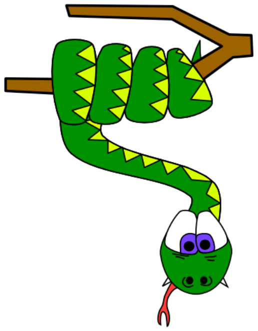
S.No.	Function & Description
1	**cmp(tuple1, tuple2)**Compares elements of both tuples.(Not in Python 3)
2	**len(tuple)**Gives the total length of the tuple.
3	**max(tuple)**Returns item from the tuple with max value.
4	**min(tuple)**Returns item from the tuple with min value.
5	**tuple(seq)**Converts a list into tuple.

Tuples…

```
#creating Empty tuple
t=tuple()
print(type(t))
t=()
print(type(t))
#initializing tuple
t=(1,)#initialization with single element
t=(1,'ab',56.67,67)
print(t)
#adding two Tuples
s=(1,2,3)+('ab',4)
print(s)
#repetition
print(s*3)
#tuples are immutable--add or modifying not allowed
```

Tuples

```
#iterating tuples
t=(1,'ab',3,45.67,'bg')
for i in t:
    print(i)
for i in range(len(t)):#tuples support integer indexing
    print(t[i])
#functions
print(len(t))
t=(23,56,21,67,43)
print(max(t))
print(min(t))
l=[1,2,3]
t=tuple(l)#conversion
print(t)
t=(23,56,21,67,43)
l=sorted(t)
print(l)
#t.sort()--tuples dont have sort functions
l=reversed(t)
print(tuple(l))
```



Sets

Method	Description
add()	Add an element to a set
clear()	Remove all elements form a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set

More..

pop()	Remove and return an arbitary set element. Raise KeyErrorif the set is empty
remove()	Remove an element from a set. If the element is not a member, raise a KeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Built in functions..

Function	Description
all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Retrun the sum of all elements in the set.

Set...

```
#creating an empty set
s=set()
print(type(s))
s={}#doesn't creates a set --creates empty dictionary
print(type(s))
#initializaton
s={1,11,2,34,23}
s.add(63)
print(s)
#set is not reversible
#s.sort() cant use sort with set
print(sorted(s))#returns a sorted list
s.remove(2)
print(s)
#print(s[1:4])--set is not subscriptable
```

Set…

```
s={11,22,13,45,34,67,89,'gh'}
p={98,23,13,66,45,89,'jhon',56.78,56,32,'ab','gh'}
t={'gh','hj',66,67}
print(s.union(p,t))
print(s.intersection(p,t))
print(p.difference(s,t))#returns difference between 2 or more sets
#print(s,p)
s.difference_update(p,t)#updates s with s difference p,t
print(s)
```

Set...

```
s={11,22,13,45,34,67,89,'gh'}
p={98,23,13,66,45,89,'jhon',56.78,56,32,'ab','gh'}
t={'gh','hj'}
s.update(p,t)#update set s with union of s & p & t
print(s)
s={11,22,13,45,34,67,89}
s.discard(34)#removes element- if not found do nothing
print(s)
#s.remove(34)#removes element if not found raise error
s.discard(34)
s.intersection_update(p)#works with two sets only
#update set element with intersection-returns None
print(s)
```

Set…

```
s={11,22,13,45,34,67,89}
p={98,23,13,66,45,89,'jhon',56.78,56,32,'ab','gh'}
t={'gh','hj'}
print(s.symmetric_difference(p))#works with two sets
#returns symmetric difference of two sets
print(s)
s.symmetric_difference_update(p)#works with two sets
#updates set s with symmetric difference of s & p
print(s)
```

Set...

```
s={1,2,3}
p={1,2,3,4,5,6}
t={7,8}
print(s.issubset(p))
print(p.issuperset(s))
print(s.isdisjoint(t))
print(sum(s))
```

Misc…

```
S={1,2,3}
if all(s):
    print("All elements are true")
else:
    print("All elements are not true")
s={0,0,23}
if any(s):
    print("At least one is true")
else:
    print("All are false")
l=[1,2,3]
if all(l):
    print("All elements are true")
d={None:0,'ab':None}
if all(d):
    print("All elements are true")
else:
    print("Not true")
#all and any works with all types
```

Misc…

```
from copy import copy, deepcopy
l=[1,2,3,['ab','bc']]
t=l
print(t,l,id(t),id(l))
p=copy(l)#shallow copy--copies references in new objects
#therefore ids of elements are same
print(id(p[3]),id(l[3]))#ids same
print(p,l,id(p),id(l))
s=deepcopy(l)#copies elements in new object thereofore ids
different
print(id(s[3]),id(l[3]))#ids different
print(s,l,id(s),id(l))
```