Introduction to Computer and Programming

Lecture 8

Yue Zhang
Westlake University

August 1, 2023

Chapter 8.

Mutable Types

Python Types

immutable types	mutable types
int	list
float	dict
function	set
bool	
string	
tuple	
NoneType	

Python Types

immutable types	mutable types
int	list
float	dict
function	set
bool	
string	
tuple	
NoneType	

Objects of Mutable types can be edited.

List

List literals

```
>>> type([1,2,3])
<class 'list'>
>>> l=['a']
>>> type(l)
<class 'list'>
>>> x='a'
>>> l=[1.0,x,5]
>>> type(l)
<class 'list'>
>>> l
[1.0, 'a', 5]
```

List

List Operators

```
>>> l=[1,'a','False']
>>> l+[1.0,(1,2,3)]
[1, 'a', 'False', 1.0, (1, 2, 3)]
>>> l*2
[1, 'a', 'False', 1, 'a', 'False']
```

- addition and multiplication

```
>>> 'a' in l
True
>>> 'b' in l
False
>>> 1 not in l
False
>>> 2 not in l
True
```

- member

```
>>> l[0]
1
>>> l[-1]
'False'
```

- get item

```
>>> l[0:2]
[1, 'a']
>>> l[-2:]
['a', 'False']
```

- get slice

WestlakeNLP

List

List Length

```
>>> l=[1,2,3]
>>> len(l)
3
```

```
>>> bool(l)
True
>>> l=[]
>>> bool(l)
False
```

- length
- nonempty / empty

WestlakeNLP

List

List to Tuple/String

```
>>> l=[3.0,'abc',True]
>>> t=tuple(l)
>>> t
(3.0, 'abc', True)
>>> t=(1,2,3.5)
>>> list(t)
[1, 2, 3.5]
>>> str(l)
"[3.0, 'abc', True]"
>>> list('hello!')
['h', 'e', 'l', 'l', 'o', '!']
```

WestlakeNLP

List

List Mutation

```
>>> l=[1.0,2.0,3.0]
>>> l[0]=0.0
>>> l
[0.0, 2.0, 3.0]
>>> l[-1]=10        # from right
>>> l
[0.0, 2.0, 10]
```

- set item

WestlakeNLP

List

List Mutation

```
>>> l=['a','b','c']
>>> l[0:2]=['x','y']
>>> l
['x', 'y', 'c']
```

```
>>> l[1:3]=['h','i','j']
>>> l
['x', 'h', 'i', 'j']
```

```
>>> l[1:2]=[]
>>> l
['x', 'i', 'j']
```

- set slice [start_index:end_index+1]
- replace with a larger slice
- replace with a smaller slice

WestlakeNLP

List

Tuples are not mutable.

```
>>> t=(1,2,3)
>>> t[0]
1
>>> t[0]=6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

WestlakeNLP

List

Methods

```
>>> l=[1,2,3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
```

- add at the end

```
>>> l.insert(0,5)
>>> l
[5, 1, 2, 3, 4]
>>> l.insert(3,'a')
>>> l
[0, 1, 2, 'a', 3, 4]
```

- insert at a location

```
>>> l.remove(1)
>>> l
[0, 2, 'a', 3, 4]
```

- remove at a location

```
>>> l.remove(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

- index error

Methods are **type-specific** functions for **instances**.

WestlakeNLP

List

Methods

```
>>> l=[1,2,3]
>>> x=[4,5,6]
>>> l.extend(x)
>>> l
[1, 2, 3, 4, 5, 6]
```

```
>>> l.reverse()
>>> l
[6, 5, 4, 3, 2, 1]
```

```
>>> l=[1,5,8,6,2,7]
>>> l.sort()
>>> l
[1, 2, 5, 6, 7, 8]
```

- add all items
- reverse
- sort to order

WestlakeNLP

List

Difference between mutation and variable reassignment.

```
>>> l1=[1,2,3]
>>> l2=l1
>>> x=[4,5,6]
```

```
>>> l1.extend(x)
>>> l1
[1, 2, 3, 4, 5, 6]
>>> l2
[1, 2, 3, 4, 5, 6]
```

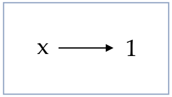
```
>>> l1=[1,2,3]
>>> l2
[1, 2, 3, 4, 5, 6]
>>> x
[4, 5, 6]
```

- mutation
- reassignment

Variable Assignment

Change the variable binding, but not the object themselves.

```
>>> x=1
>>> x
1
```

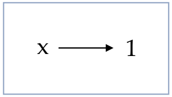


$$\begin{tikzcd} x \arrow[r] & 1 \end{tikzcd}$$

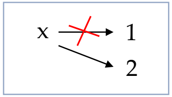
Variable Assignment

Change the variable binding, but not the object themselves.

```
>>> x=1
>>> x
1
```



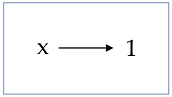
```
>>> x=2
>>> x
2
```



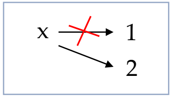
Variable Assignment

Change the variable binding, but not the object themselves.

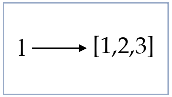
```
>>> x=1
>>> x
1
```



```
>>> x=2
>>> x
2
```



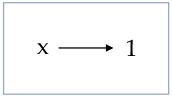
```
>>> l=[1,2,3]
>>> l
[1,2,3]
```



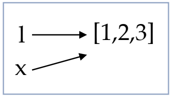
Variable Assignment

Change the variable binding, but not the object themselves.

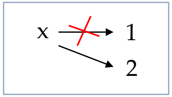
```
>>> x=1
>>> x
1
```


$$\begin{tikzcd} x \arrow[r] & 1 \end{tikzcd}$$

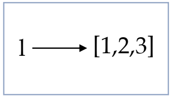
```
>>> x=l
>>> x
[1, 2, 3]
```


$$\begin{tikzcd} l \arrow[r] & {[1,2,3]} \\ x \arrow[ur] & \end{tikzcd}$$

```
>>> x=2
>>> x
2
```


$$\begin{tikzcd} x \arrow[r, "\times"] \arrow[dr] & 1 \\ & 2 \end{tikzcd}$$

```
>>> l=[1,2,3]
>>> l
[1,2,3]
```


$$\begin{tikzcd} l \arrow[r] & {[1,2,3]} \end{tikzcd}$$

Variable Assignment

Change the variable binding, but not the object themselves.

```
>>> x=1
>>> x
1
```

$$\begin{tikzcd} x \arrow[r] & 1 \end{tikzcd}$$

```
>>> x=2
>>> x
2
```

$$\begin{tikzcd} x \arrow[r, "\times" marking] \arrow[dr] & 1 \\ & 2 \end{tikzcd}$$

```
>>> l=[1,2,3]
>>> l
[1,2,3]
```

$$\begin{tikzcd} l \arrow[r] & {[1,2,3]} \end{tikzcd}$$

```
>>> x=l
>>> x
[1, 2, 3]
```

$$\begin{tikzcd} l \arrow[r] & {[1,2,3]} \\ x \arrow[ur] & \end{tikzcd}$$

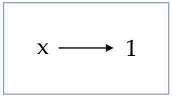
```
>>> l[0]=0
>>> l
[0, 2, 3]
>>> x
[0, 2, 3]
```

$$\begin{tikzcd} l \arrow[r] & {[0,2,3]} \\ x \arrow[ur] & \end{tikzcd}$$

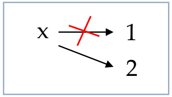
Variable Assignment

Change the variable binding, but not the object themselves.

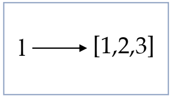
```
>>> x=1
>>> x
1
```



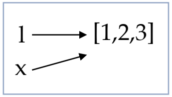
```
>>> x=2
>>> x
2
```



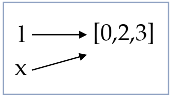
```
>>> l=[1,2,3]
>>> l
[1,2,3]
```



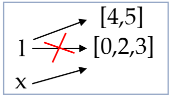
```
>>> x=l
>>> x
[1, 2, 3]
```



```
>>> l[0]=0
>>> l
[0, 2, 3]
>>> x
[0, 2, 3]
```



```
>>> l=[4,5]
>>> l
[4, 5]
>>> x
[0, 2, 3]
```



List

Copying A List

```
>>> l=[1,2,3]
>>> x=l
>>> x
[1, 2, 3]
```

```
>>> y=l[:]
>>> y
[1, 2, 3]
```

```
>>> import copy
>>> z=copy.copy(l)
>>> l[0]=0
>>> x
[0, 2, 3]
>>> y
[1, 2, 3]
>>> z
[1, 2, 3]
```

- assignment
- get slice
- copy module function

WestlakeNLP

List

Nested List – list of list

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> l[0]
[1, 2, 3]
>>> l[0][0]
1
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> c=[7,8,9]
>>> l=[a,b,c]
>>> l[1]
[4, 5, 6]
>>> l[1][0]
4
```

The items in *l* are themselves lists.

List

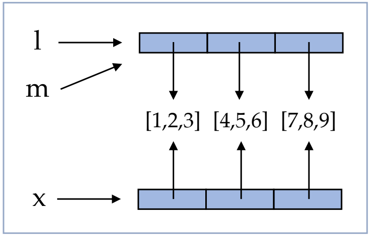
Nested List and Deep Copy

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

List

Nested List and Deep Copy

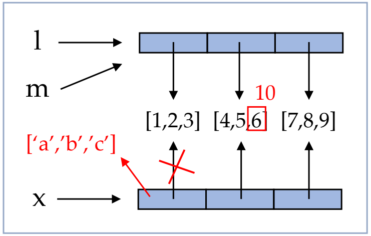
```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```



List

Nested List and Deep Copy

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

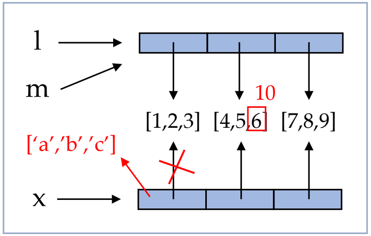


List

Nested List and Deep Copy

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

```
>>> y=copy.deepcopy(l)
>>> y[0]=['a','b','c']
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
>>> y[1][1]=100
>>> y
[['a', 'b', 'c'], [4, 100, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

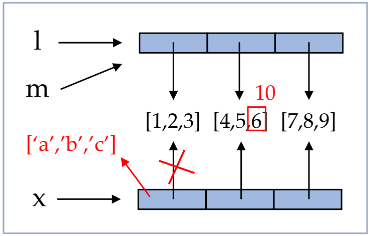


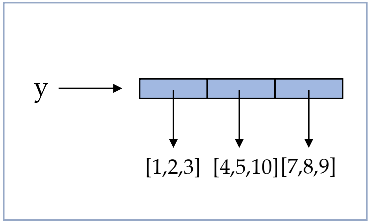
List

Nested List and Deep Copy

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

```
>>> y=copy.deepcopy(l)
>>> y[0]=['a','b','c']
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
>>> y[1][1]=100
>>> y
[['a', 'b', 'c'], [4, 100, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```



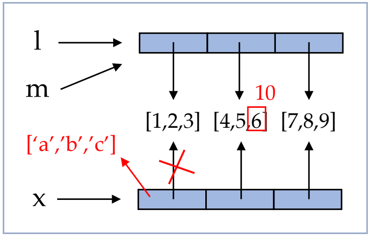


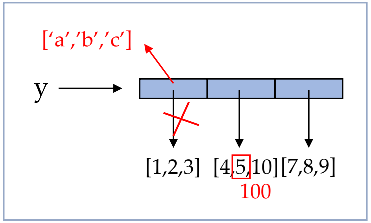
List

Nested List and Deep Copy

```
>>> l=[[1,2,3],[4,5,6],[7,8,9]]
>>> m=l
>>> import copy
>>> x=copy.copy(l)
>>> x[0]=['a','b','c']
>>> x
[['a', 'b', 'c'], [4, 5, 6], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> x[1][2]=10
>>> x
[['a', 'b', 'c'], [4, 5, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```

```
>>> y=copy.deepcopy(l)
>>> y[0]=['a','b','c']
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
>>> y[1][1]=100
>>> y
[['a', 'b', 'c'], [4, 100, 10], [7, 8, 9]]
>>> l
[[1, 2, 3], [4, 5, 10], [7, 8, 9]]
```





List

Iteration

```
>>> l=['a',lambda x:x+1,False]
>>> for i in l:
...     print(i,type(i))
...
a <class 'str'>
<function <lambda> at 0x102ccf940> <class 'function'>
False <class 'bool'>
```

List

Iteration

```
>>> l=['a','b','c']
>>> for (index,item) in enumerate(l):
...     print('The %dth item is %s'%(index,item))
...
The 0th item is a
The 1th item is b
The 2th item is c
```

List

range

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> for i in range(3,10):
...     print(i)
...
3
4
5
6
7
8
9
```

List

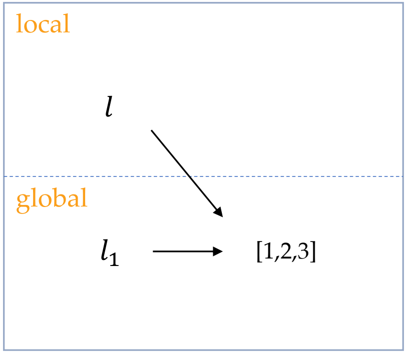
Lists as Function Arguments

```
>>> def duplicate(l):
...     l.extend(l)
...
>>> l1=[1,2,3]
>>> duplicate(l1)
>>> l1
[1, 2, 3, 1, 2, 3]
```

List

Lists as Function Arguments

```
>>> def duplicate(l):
...     l.extend(l)
...
>>> l1=[1,2,3]
>>> duplicate(l1)
>>> l1
[1, 2, 3, 1, 2, 3]
```



Argument passing is equivalent to assignment.

List

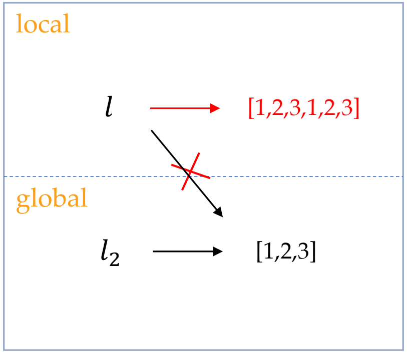
Lists as Function Arguments

```
>>> def duplicate(l):
...     l=l+l
...
>>> l2=[1,2,3]
>>> duplicate(l2)
>>> l2
[1, 2, 3]
```

List

Lists as Function Arguments

```
>>> def duplicate(l):
...     l=l+l
...
>>> l2=[1,2,3]
>>> duplicate(l2)
>>> l2
[1, 2, 3]
```



The local variable reassigned.

List

List Parameters

```
>>> def sum(x,y,*z):
...     s=x+y
...     for i in z:
...         s+=i
...     return s
...
>>> sum(1,2)            # z=[]
3
>>> sum(1,2,3)          # z=[3]
6
>>> sum(1,2,5,6)        # z=[5,6]
14
```

List

List Items as Arguments

```
>>> def sum(x,y):
...     return x+y
...
>>> l=[1,2]
>>> sum(*l)
3
```

List

List as Return Value

```
>>> def negatelist(l):
...     s=l[:]
...     for (i,x) in enumerate(s):
...         s[i]=-x
...     return s
...
>>> l=[1,-2,3,-5,0]
>>> negatelist(l)
[-1, 2, -3, 5, 0]
>>> l
[1, -2, 3, -5, 0]
```

Dict

Dicts are mapping Types.

key – value pairs

e.g., student ID –> Name

Dict

Dict Literal

```
>>> d={1:'a',2:'b',3:'c'}
>>> d
{1: 'a', 2: 'b', 3: 'c'}
>>> type(d)
<class 'dict'>
>>> d={1.0:1,True:3.5,(1,2):'abc'}
>>> d
{1.0: 3.5, (1, 2): 'abc'}
>>> d={[1,2]:'a'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

WestlakeNLP

Dict

Dict Operators – Equality

```
>>> d1={1:'a',2:'b',3:'c'}
>>> d2={3:'c',2:'b',1:'a'}
>>> d3={}
>>> d1==d2
True
>>> d1!=d3
True
```

- == and != operators

The order of key-value pairs does not matter to a dict.

WestlakeNLP

Dict

Dict Operators – look up

```
>>> d={1:'a',2:'b',3:'c'}
>>> d[1]
'a'
>>> d[3]
'c'
>>> d[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 5
```

- look up

WestlakeNLP

Dict

Dict Methods – look up with default value

```
>>> d={1:'a',2:'b',3:'c'}
>>> d.get(3,'2')
'c'
>>> d.get(5,'2')
'2'
```

WestlakeNLP

Dict

Dict Operators – membership

```
>>> d={1:'a',2:'b',3:'c'}
>>> 1 in d
True
>>> 4 in d
False
>>> 'a' in d
False
>>> 4 not in d
True
```

Membership refers to keys.

Dict

Dict Method – len

```
>>> d={1:'a',2:'b',3:'c'}
>>> len(d)
3
>>> len({})
0
```

Dict

Dict and Other Types

```
>>> d={1:'a',2:'b',3:'c'}
>>> bool(d)
True
>>> bool({})
False
>>> type(d)
<class 'dict'>
>>> list(d)
[1, 2, 3]
>>> t=((1,'a'),(2,'b'),(3,'c'))
>>> dict(t)
{1: 'a', 2: 'b', 3: 'c'}
```

Dict

Dict Mutation – set item

```
>>> d={1:'a',2:'b',3:'c'}
```

```
>>> d[0]='d'
>>> d
{1: 'a', 2: 'b', 3: 'c', 0: 'd'}
```

```
>>> d[1]='e'
>>> d
{1: 'e', 2: 'b', 3: 'c', 0: 'd'}
```

- add
- override

WestlakeNLP

Dict

Dict Mutation – delitem

```
>>> d={1:'a',2:'b',3:'c'}
>>> del d[1]
>>> d
{2: 'b', 3: 'c'}
```

Dict

Dict Mutation – delitem

```
>>> d={1:'a',2:'b',3:'c'}
>>> del d[1]
>>> d
{2: 'b', 3: 'c'}
```

del can be used for other types.

```
>>> a=1
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

```
>>> l=[1,2,3]
>>> del l[1]
>>> l
[1, 3]
```

- identifier
- list

Dict

Methods – update

```
>>> d={1:'a',2:'b',3:'c'}
>>> e={0:'d',1:'e'}
>>> d.update(e)
>>> d
{1: 'e', 2: 'b', 3: 'c', 0: 'd'}
```

Dict

Methods – pop

```
>>> d={1:'a',2:'b',3:'c'}
>>> d.pop(1)
'a'
>>> d
{2: 'b', 3: 'c'}
>>> d.pop(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

WestlakeNLP

Dict

Methods – pop

```
>>> d={1:'a',2:'b',3:'c'}
>>> d.pop(1,'d')
'a'
>>> d
{2: 'b', 3: 'c'}
>>> d.pop(0,'d')
'd'
>>> d
{2: 'b', 3: 'c'}
```

WestlakeNLP

Dict

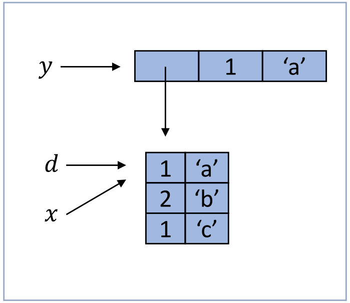
Methods – clear

```
>>> d={1:'a',2:'b',3:'c'}
>>> d.clear()
>>> d
{}
```

Dict

Methods – mutation and assignment

```
>>> d={1:'a',2:'b',3:'c'}
>>> x=d
>>> y=(d,1,'a')
>>> y[0].update({0:'e',1:'d'})
>>> x
{1: 'd', 2: 'b', 3: 'c', 0: 'e'}
>>> d
{1: 'd', 2: 'b', 3: 'c', 0: 'e'}
>>> del x[3]
>>> x
{1: 'd', 2: 'b', 0: 'e'}
>>> d
{1: 'd', 2: 'b', 0: 'e'}
```



Dict

Methods – keys & values

```
>>> d={1:'a',2:'b',3:'c'}
>>> d.keys()
dict_keys([1, 2, 3])
>>> d.values()
dict_values(['a', 'b', 'c'])
>>> d.items()
dict_items([(1, 'a'), (2, 'b'), (3, 'c')])
```

WestlakeNLP

Dict – dicts and loops

iterate keys

```
>>> d={1:'a',2:'b',3:'c'}
>>> for k in d:
...     print('k =',k,'d[k] =',d[k])
...
k = 1 d[k] = a
k = 2 d[k] = b
k = 3 d[k] = c
>>> for k in d.keys():
...     print('k =',k,'d[k] =',d[k])
...
k = 1 d[k] = a
k = 2 d[k] = b
k = 3 d[k] = c
```

Dict – dicts and loops

iterate values and items

```
>>> d={1:'a',2:'b',3:'c'}
>>> for v in d.values():
...     print(v)
...
a
b
c
>>> for t in d.items():
...     print(t)
...
(1, 'a')
(2, 'b')
(3, 'c')
```

WestlakeNLP

Dict – dicts and loops

The dict object cannot be modified during iteration.

```
>>> d={'a':1,'b':2,'c':3}
>>> for k in d:
...     if d[k]%2==0:
...         del d[k]
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: dictionary changed size
   during iteration
>>> d
{'a': 1, 'c': 3}
```

Dict – dicts and loops

Solution – iterate elsewhere

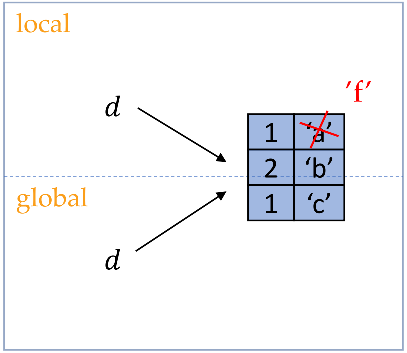
```
>>> d={'a':1,'b':2,'c':3}
>>> keys=list(d.keys())
>>> for k in keys:
...     if d[k]%2==0:
...         del d[k]
...
>>> d
{'a': 1, 'c': 3}
```

WestlakeNLP

Dict

Dicts and Functions – argument

```
>>> def f(d):
...     d[1]='f'
...
>>> d={1:'a',2:'b',3:'c'}
>>> f(d)
>>> d
{1: 'f', 2: 'b', 3: 'c'}
```



Argument passing is assignment.

Dict

Dicts and Functions – dict arguments

```
>>> def f(a,b,**d):
...     print('**explicit arguments**')
...     print('a=',a)
...     print('b=',b)
...     print('**packed arguments**')
...     for k in d:
...         print(k,'=',d[k])
...
```

- Here, **d represents dict arguments.

Dict

Dicts and Functions – dict arguments

```
>>> def f(a,b,**d):
...     print('**explicit arguments**')
...     print('a=',a)
...     print('b=',b)
...     print('**packed arguments**')
...     for k in d:
...         print(k,'=',d[k])
...
```

```
>>> f(1,2,c=3,d=4,e=5)
**explicit arguments**
a= 1
b= 2
**packed arguments**
c = 3
d = 4
e = 5
```

- Here, **d represents dict arguments.
- where in f(1,2,c=3,d=4,e=5), d={'c':3,'d':4,'e':5}

Dict

Dicts and Functions – dict arguments

```
>>> def f(a,b,**d):
...     print('**explicit arguments**')
...     print('a=',a)
...     print('b=',b)
...     print('**packed arguments**')
...     for k in d:
...         print(k,'=',d[k])
...
```

```
>>> f(1,2,c=3,d=4,e=5)
**explicit arguments**
a= 1
b= 2
**packed arguments**
c = 3
d = 4
e = 5
```

```
>>> f(a=0,b=-1,e=-5)
**explicit arguments**
a= 0
b= -1
**packed arguments**
e = -5
```

- Here, **d represents dict arguments.
- where in f(1,2,c=3,d=4,e=5), d={'c':3,'d':4,'e':5}
- where in f(a=0,b=-1,e=-5), d={'e':-5}

Dict

Dicts and Functions – dict as arguments

```
>>> def f(a,b):
...     return a+b
...
>>> d={'a':1,'b':2}
>>> f(**d)
3
```

Set

Set Literal

```
>>> a=1
>>> f=5.6
>>> s={a,f,24,3.6}
>>> s
{24, 1, 3.6, 5.6}
>>> type(s)
<class 'set'>
```

WestlakeNLP

Set

Set Operators

```
>>> s={1,2,3}
>>> 1 in s
True
>>> 3 not in s
False
>>> 4 not in s
True
```

Set

Size and Empty Set

```
>>> s={1,2,3}
>>> len(s)
3
>>> bool(s)
True
>>> s={}
>>> bool(s)
False
```

WestlakeNLP

Set

Set Operators

```
>>> a={1,2,3}
>>> b={4,5,6}
>>> c={3}
>>> d={3,1,2}
>>> a==d
True
>>> a!=c
True
>>> c!=b
True
>>> a>c                 # superset
True
>>> a>=c
True
>>> a<b                 # subset
False
>>> b.issuperset(c)
False
```

Set

Set Operators

```
>>> a={1,2,3}
>>> b={3,4}
```

```
>>> a&b
{3}
```

```
>>> a|b
{1, 2, 3, 4}
```

```
>>> a-b
{1, 2}
>>> b-a
{4}
>>> a.symmetric_difference(b)
{1, 2, 4}
```

- intersection
- union
- difference

WestlakeNLP

Set

Set Mutation

```
>>> s={1,2,3}
>>> s.add(0)
>>> s
{0, 1, 2, 3}
>>> s.remove(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 5
```

```
>>> s.discard(5)
>>> s
{0, 1, 2, 3}
```

- no errors and warnings

Set

Set Mutation

```
>>> s={1,2,3}
>>> x={4,3}
>>> y={1}
>>> s.update(x)
>>> s
{1, 2, 3, 4}
>>> s.intersection_update(y)
>>> s
{1}
```

Set

Set Iterations

```
>>> s={1,5,6,7,3,8}
>>> for i in s:
...     print(i)
...
1
3
5
6
7
8
```

Set

Bit Set

```
>>> a=0b1101
>>> b=0b1011
>>> bin(a&b)
'0b1001'
>>> bin(a|b)
'0b1111'
>>> bin(a^b)
'0b110'
```

- $\{0, 2, 3\}$
- $\{0, 1, 3\}$
- intersection - $\{0, 3\}$
- union - $\{0, 1, 2, 3\}$
- difference - $\{1, 2\}$

WestlakeNLP

This week check-off:

Mutable Types