Introduction to Python Programming Unit 2

Akshata S. Bhayyar

Assistant Professor, Dept. of CSE, MSRIT

LIST

- A list is an ordered set of values, where each value is identified by an index
- The values that make up a list are called its elements.
- Also called as sequences

LIST VALUES

- Ways to create a new list
  - enclose the elements in square brackets [ ]:

Eg1 [10, 20, 30, 40]

Eg2 ["spam", "bungee", "swallow"]

Eg3 ["hello", 2.7, 5]

LIST VALUES

- Lists that contain consecutive integers:

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> range(1,5)
[1, 2, 3, 4]
```

```
>>> range(1, 10, 2)
[1, 3, 5, 7, 9]
```

- Empty list it is denoted [].

ACCESSING ELEMENTS

```
>>> l2
['string', 1, 0.987]
>>> l2[0]
'string'
>>> l2[2]
0.987
```

ACCESSING ELEMENTS

```
>>> l2[2]="Hi"
>>> l2
['string', 1, 'Hi']
```

If you try to read or write an element that does not exist, you get a runtime error:

ACCESSING ELEMENTS

- If an index has a negative value, it counts backward from the end of the list:

```
>>> l2
['string', 1, 'Hi']
>>> l2[-1]
'Hi'
>>> l2[-2]
1
>>> l2[-3]
'string'
>>> l2[-4]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    l2[-4]
IndexError: list index out of range
```

Using Loops

```
l1=[1,"hi",22,89.567,"hello"]
i=0
while i<5:
    print(l1[i])
    i=i+1
```

LIST LENGTH

- The function len returns the length of a list

```
l1=[1,"hi",22,89.567,"hello"]
i=0
while i<len(l1):
    print(l1[i])
    i=i+1
```

LIST within another LIST

- The function len returns the length of a list

```
l1=[1,"hi",22,['a','b','c'],"hello"]
i=0
while i<len(l1):
    print(l1[i])
    i=i+1
```

List membership

- in is a boolean operator that tests membership in a sequence.

```
>>> "hi" in l1
True
>>> l1
[1, 'hi', 22, ['a', 'b', 'c'], 'hello']
>>> "hi" in l1
True
>>> 5 in l1
False
```

List operations

- The + operator concatenates lists

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

List operations

- Similarly, the * operator repeats a list a given number of times

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

```
------------------
[1, 2, 3, 1, 2, 3]
```

List slices

```
>>> l1=['a','b','c','d','e','f','g','h']
>>> l1[2:5]
['c', 'd', 'e']
>>> l1[:5]
['a', 'b', 'c', 'd', 'e']
>>> l1[2:]
['c', 'd', 'e', 'f', 'g', 'h']
>>> l1[:]
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Lists are mutable

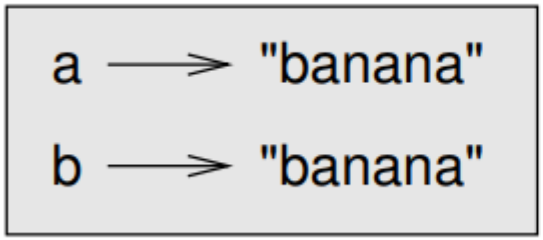
- lists are mutable, which means we can change their elements.

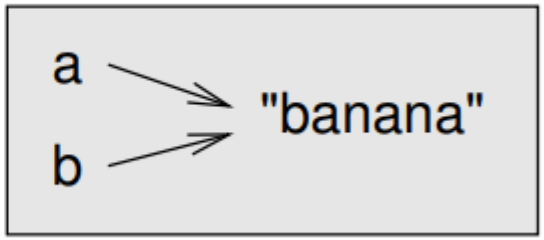
```
>>> l1
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
>>> l1[2]="hello"
>>> l1
['a', 'b', 'hello', 'd', 'e', 'f', 'g', 'h']
```

List deletion

```
>>> l1=['a','b','c','d','e','f','g','h']
>>> del l1[1]
>>> l1
['a', 'c', 'd', 'e', 'f', 'g', 'h']
>>> del l1[2:5]
>>> l1
['a', 'c', 'g', 'h']
```

Objects and values





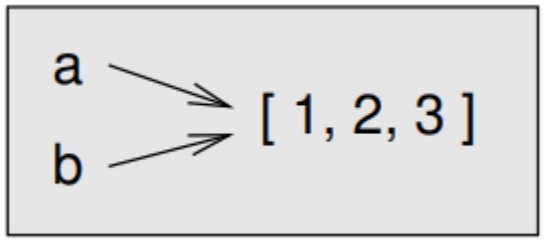

```
>>> id(a)
135044008
>>> id(b)
135044008
```

Objects and values

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> id(a)
135045528
>>> id(b)
135041704
```

Aliasing

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



Cloning lists

If we want to modify a list and also keep a copy of the original

```
a=[1,2,3]
b=a[:]
del b[1]
print(b)
print(a)
```

```
[1, 3]
[1, 2, 3]
```

Cloning lists

If we want to modify a list and also keep a copy of the original

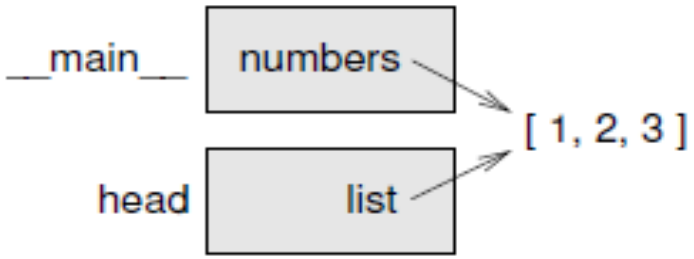
```
a=[0,1,2,3,4,5,6,7,8]
b=a[3:5]
print(b)
del b[1]
print(b)
print(a)
```

```
[3, 4]
[3]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

List parameters

```
def head(list):
    return list[0]


num=[1,2,3,4,5]
print(head(num))
```



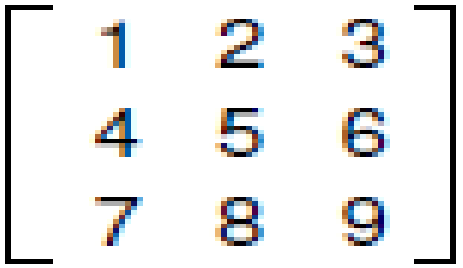
Nested lists

```
list1=[1,2,3,['a','b','c'],4,5]
print(list1)
list2=list1[3]
print(list2)
no=list1[3][1]
print(no)
```

```
[1, 2, 3, ['a', 'b', 'c'], 4, 5]
['a', 'b', 'c']
b
```

Matrices

sted lists are often used to represent matrices




```
matrix=[[1,2,3],[4,5,6],[7,8,9]]
for i in range(3):
    print(matrix[i])
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

Strings and lists

```
song="The rain in spain.."
print(song.split())
```

```
['The', 'rain', 'in', 'spain..']
```

Strings and lists- delimiter

```
song="The rain in spain.."
print(song.split())
print(song.split('ai'))
```

```
['The', 'rain', 'in', 'spain..']
['The r', 'n in sp', 'n..']
```

Strings and lists- join

```
song="The rain in spain.."
list1=song.split()
print(list1)
list2=" "
print(list2.join(list1))
```

```
['The', 'rain', 'in', 'spain..']
The rain in spain..
```

List Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

List Methods

1.append()

syntax

list.append (element)

```
List = ['Mathematics', 'chemistry', 1997, 2000]
List.append(20544)
print(List)
```

```
['Mathematics', 'chemistry', 1997, 2000, 20544]
```

List Methods

insert()

syntax

list.insert(position, element)

```
List = ['Mathematics', 'chemistry', 1997, 2000]
# Insert at index 2 value 10087
List.insert(2,10087)
print(List)
```

Mathematics', 'chemistry', 10087, 1997, 2000]

List Methods

extend()

syntax

List1.extend(List2)

```
List1 = [1, 2, 3]
List2 = [2, 3, 4, 5]

# Add List2 to List1
List1.extend(List2)
print(List1)

# Add List1 to List2 now
List2.extend(List1)
print(List2)
```

[1, 2, 3, 2, 3, 4, 5]

[2, 3, 4, 5, 1, 2, 3, 2, 3, 4, 5

List Methods

sum()

syntax

sum(List)

```
List = [1, 2, 3, 4, 5]
print(sum(List))
```

List Methods

count()

syntax

List.count(element)

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.count(1))
```

4

List Methods

Length()

syntax

len(list_name)

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(len(List))
```

10

List Methods

index()

syntax

List.index(element[start[end]
])

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.index(2))
```

1

List Methods

Deletion of List Elements

To Delete one or more elements, i.e. remove an element, many built-in functions can be used, such as pop() & remove() and keywords such as del.

pop()

The index is not a necessary parameter, if not mentioned takes the last index.

Syntax:

list.pop([index])

```
List = [2.3, 4.445, 3, 5.33, 1.054, 2.5]
print(List.pop())
```

List Methods

```
>>> lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Tutorialspoint', 'Python']
>>> lst.pop(3)
'Java'
>>> lst
['CPlusPlus', 'Hello', 'Python', 'Tutorialspoint', 'Python']
```

```
>>> lst.remove('Python')
>>> lst
['CPlusPlus', 'Hello', 'Tutorialspoint', 'Python']
```

List Methods

reverse()

```
>>> lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Tutorialspoint', 'Python']
>>> lst.reverse()
>>> lst
['Python', 'Tutorialspoint', 'Java', 'Python', 'Hello', 'CPlusPlus']
```

List Methods

Sort( )

```
>>> lst = [2, 3, 7, 1, 13, 8, 49]
>>> print(lst)
[2, 3, 7, 1, 13, 8, 49]
>>> lst.sort()
>>> print(lst)
[1, 2, 3, 7, 8, 13, 49]

>>> lst.sort(reverse=True)
>>> lst
[49, 13, 8, 7, 3, 2, 1]
```

Pure functions and Modifiers

Pure functions : It does not modify any of the objects passed to it as arguments and it has no side effects, such as displaying a value or getting user input.

Modifiers: the caller keeps a reference to the objects it passes, so any changes the function makes are visible to the caller. Functions that work this way are called modifiers.

Tuples

Tuple

- A tuple that is similar to a list except that it is immutable.
- Syntactically, a tuple is a comma-separated list of va

```
>>> t='a','b','c'
>>> t
('a', 'b', 'c')
>>> t1=(1,2,3)
>>> t1
(1, 2, 3)
```

Tuple

- To create a tuple with a single element, we have to include the final comma:

>>> t1 = (’a’,)

- Without the comma, Python treats (’a’) as a string in parentheses:

```
>>> t1[0]
1
>>> t1[0:3]
(1, 2, 3)
>>> t1[0:2]
(1, 2)
```

Tuple

If we try to modify one of the elements of the tuple, we get an error:

```
>>> t1[0]='A'
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    t1[0]='A'
TypeError: 'tuple' object does not support item assignment
>>> type(t1)
<class 'tuple'>
```

Tuple packing and unpacking

Tuple packing is nothing but the creation of tuple, whereas **tuple unpacking** means to extract tuple values and store them in individual variables.

```
>>> tuple_packing = ("StudyTonight", 'Technical website', 12)
>>> (name_of_website, type_of_website, no_of_characters) = tuple_packing
>>> print(name_of_website)
StudyTonight
>>> print(type_of_website)
Technical website
>>> print(no_of_characters)
12
```

Tuple assignment

```
>>> t
('a', 'b', 'c')
>>> a,b,c=10,20,30
>>> a
10
```

Tuple assignment

```
>>> t=('a','b','c')
>>> a,b,c=1,2
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a,b,c=1,2
ValueError: not enough values to unpack (expected 3, got 2)
>>> a,b,c=1,2,3,4
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    a,b,c=1,2,3,4
ValueError: too many values to unpack (expected 3)
```

Tuples as return values

- Functions can return tuples as return values

```
def swap(x,y):
    return y,x

x,y=swap(4,5)

print(x)
print(y)
```

Composability of Data Structures

- Tuples items can themselves be other tuples.

```
Actress_info=(("Julia","Robert"),(8,"November",1967),
              "Actress",("Atlanta","Georgia"),
              [("Duplicity",2009),("Nottinh Hill",1999)])
print(Actress_info)
```

```
(('Julia', 'Robert'), (8, 'November', 1967), 'Actress', ('Atlanta', 'Georgia'),
[('Duplicity', 2009), ('Nottinh Hill', 1999)])
```

Dictionary

Dictionaries

- In a dictionary, the indices are called keys, so the elements are called key-value pairs.
- One way to create a dictionary is to start with the empty dictionary and add elements.
- The empty dictionary is denoted {}

Dictionaries

```
>>> ISEM={}
>>> ISEM['PYC12']='Physics'
>>> ISEM['PLC142']='Python'

>>> ISEM['PLC142']
'Python'

>>> print(ISEM)
{'PYC12': 'Physics', 'PLC142': 'Python'}
```

Dictionaries

nother way to create a dictionary

```
>>> Subjects={1:'Maths',2:'Physics',3:'Electronics',4:'Python'}
>>> print(Subjects)
{1: 'Maths', 2: 'Physics', 3: 'Electronics', 4: 'Python'}
...
```

```
print(len(Subjects)
)
```

Dictionary operations

1. The del statement removes a key-value pair from a dictionary

```
Subjects={1:'Maths', 2:'Physics', 3:'Electronics',4:'Python'}
print(len(Subjects))
del Subjects[1]
print(Subjects)
```

```
{2: 'Physics', 3: 'Electronics', 4: 'Python'}
```

2. print(len(Subjects))

Dictionary methods

- A method is similar to a function, it takes arguments and returns a value but the syntax is different.

- print(Subjects.keys())

dict_keys([2, 3, 4])

Dictionary methods

- The items method returns both, in the form of a list of tuples—one for each key-value pair:

```
print(Subjects.items())
```

```
dict_items([(2, 'Physics'), (3,
'Electronics'), (4, 'Python')])
```

Dictionary methods

- the method has key takes a key and returns true (1) if the key appears in the dictionary:

```
>>> eng2sp.has_key('one')
True
>>> eng2sp.has_key('deux')
False
```

Aliasing and copying

- Whenever two variables refer to the same object, changes to one affect the other.
- If you want to modify a dictionary and keep a copy of the original, use the

```
>>> Subjects={1:'Maths', 2:'Physics', 3:'Electronics',4:'Python'}
>>> ISEM=Subjects
>>> copied=Subjects.copy()
>>> ISEM
    {1: 'Maths', 2: 'Physics', 3: 'Electronics', 4: 'Python'}
>>> copied
    {1: 'Maths', 2: 'Physics', 3: 'Electronics', 4: 'Python'}
```