E-MODULE

CLASS-BCA/BSC(IT) III SEM

SUBJECT:- COMPUTATIONAL PROBLEM SOLVING USING PYTHON

TOPIC:- Introduction and Working of List and dictionaries in Python

SUBMITTED BY:- SAKSHI KAPOOR

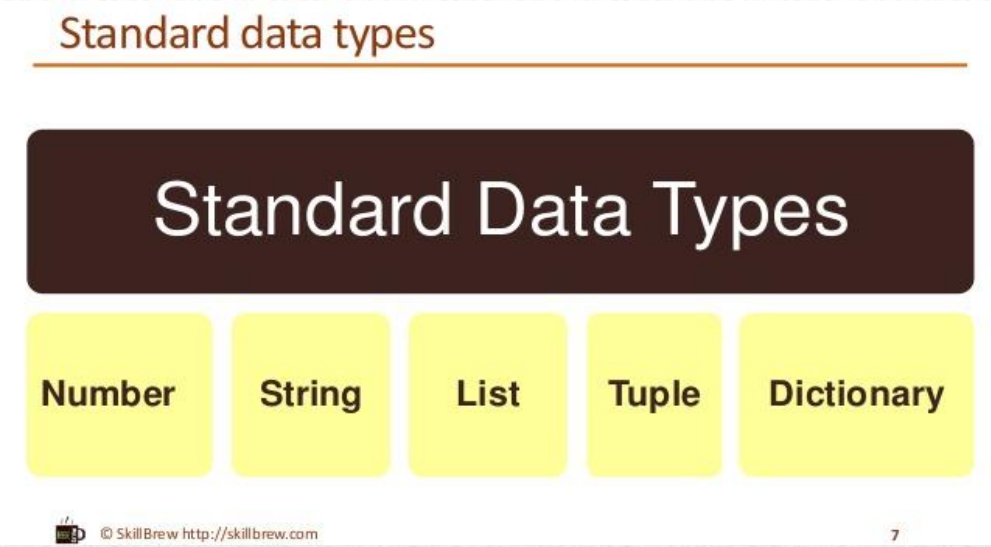
(DEPARTMENT OF COMPUTER SCIENCE & IT)

INTRODUCTION TO PYTHON

1. **Python** is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.
2. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java .
3. The language provides constructs intended to enable writing clear programs on both a small and large scale.
4. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including **object-oriented, imperative, functional programming, and procedural styles**. It has a large and comprehensive standard library

In contrast to other popular languages such as C, C++, Java, and C#, Python strives to provide a simple but powerful syntax.

Python is used for software development at companies and organizations such as Google, Yahoo, CERN, Industrial Light and Magic, and NASA. Experienced programmers can accomplish great things with Python, but Python's beauty is that it is accessible to beginning programmers and allows them to tackle interesting problems more quickly than many other, more complex languages that have a steeper learning curve.

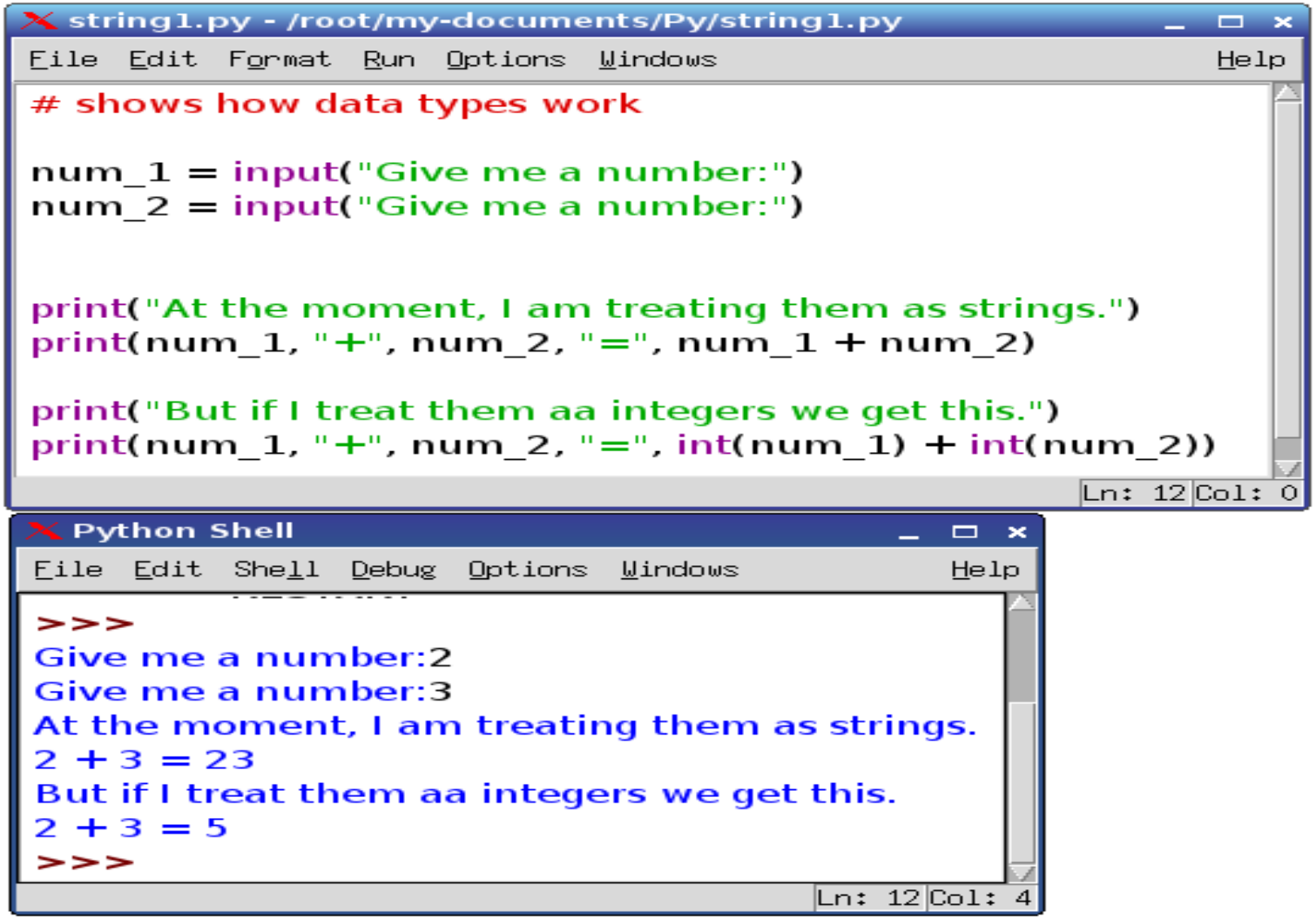


Type	Example
• Numeric: Integer, Float	x = 10 x = 1.0
• String	x= 'Mike'
• Boolean	y = True x = False
• List	my_list = [10, 20, 30]
• Tuple	my_tuple = ('Brett', 'Cisco', 'Cary',2015)
• Dictionary	my_dict = {"one":1, "two":2}
• Lists in Lists	my_list2=[[10,20,30], ['Cisco Live', 'May', 2016]]

PYTHON DATA TYPES

1. The Python Programming language provides the users with simple and efficient data types so that robust programs can be developed by using simple syntax and code structure.
2. Python provides data type NUMBER for all the integers , STRING for the collection of characters , LIST for collection of different types of data in same place and DICTIONARY for storing the values along with some specific key values also.

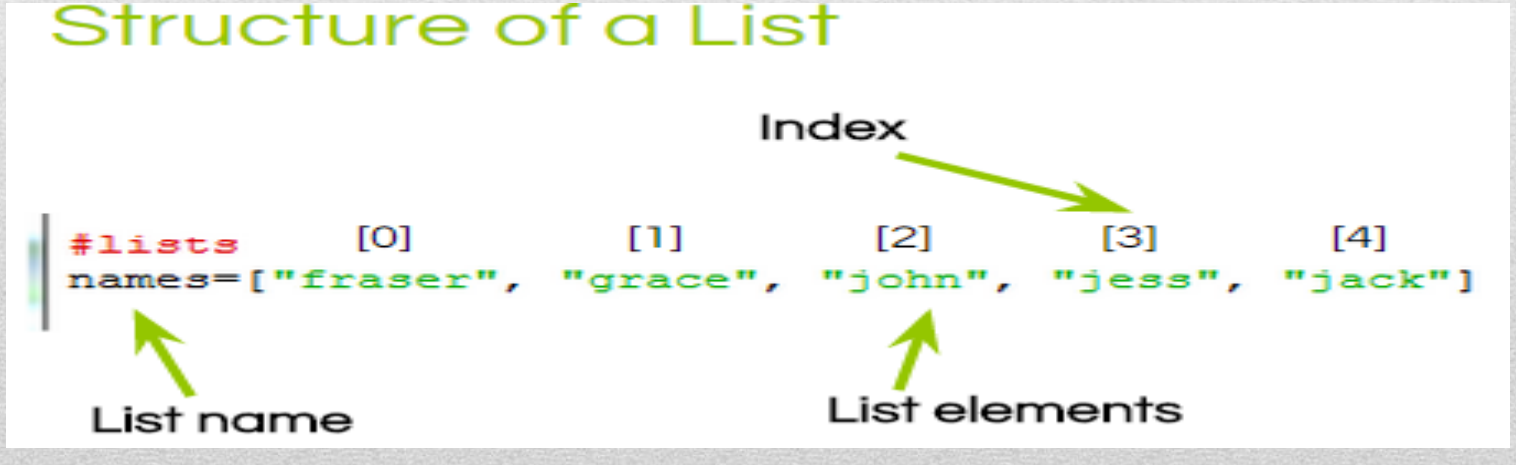
EXAMPLE OF A SIMPLE PROGRAM IN PYTHON IDLE



LIST DATA TYPE

INTRODUCTION TO LISTS

1. LIST data type in python is a collection of heterogeneous data that represents sequenced objects.
2. Python Lists are mutable i.e. changes can be made in list anytime.
3. This sequenced data type provide the users with facility to store different type of data in same place and in sequential order.
4. The simple structure of LISTS make them useful and effective data type.
5. The structure of a python list is as follows:-



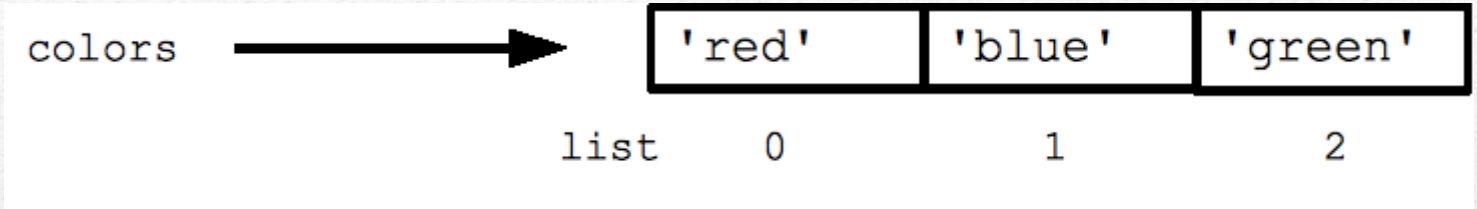
6. The structure of list explains that the elements of list will be included in square brackets [].
7. The right-hand side of the assignment statement is a literal list. The elements of the list appear within
square brackets ([ ]), the elements are separated by commas.
The following statement:

a = []

assigns the empty list to a. We can print list literals and lists referenced through variables:

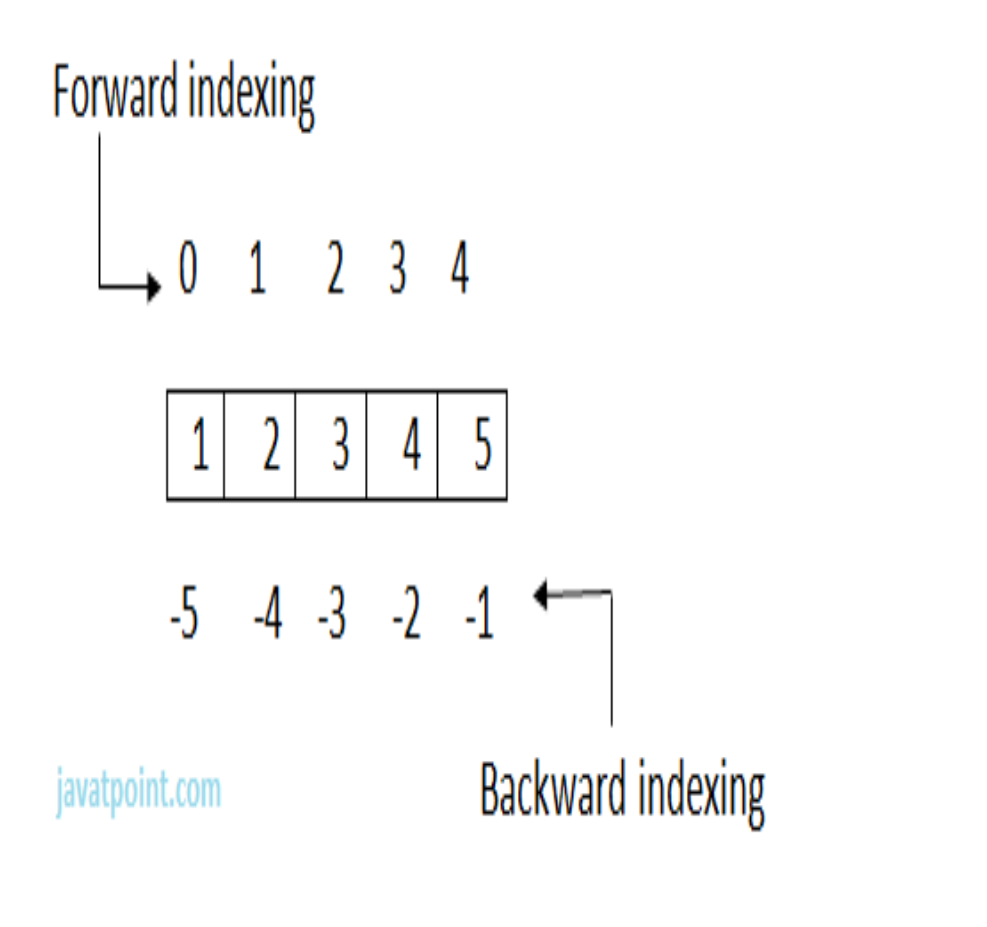
```
lst = [2, -3, 0, 4, -1] # Assign the list
print([2, -3, 0, 4, -1]) # Print a literal list
print(lst) # Print a list variable
```

STORAGE STRUCTURE AND ACCESSING OF LISTS



1. The list is stored in python with an index and a corresponding value for that index.
2. In the above example the name of the List is “colors” and the list is having three elements as List[0],list[1] and list[2] .
3. Whereas the list[0] , list[1]and list[2] are the index values for the list colors and red, blue and green are the corresponding list values.

INDEXING OF LISTS



- The elements from a list can be accessed by either forward indexing or negative indexing which is also known as positive or negative indexing.
- The index for forward indexing starts from o and the backward indexing starts from -1.
- In order to access the elements from the last of the list the negative indexing is used .

For e.g.:- To access the element 5 out from the list –

```
>>>list_name[-1]
```

LIST METHODS

Python provides many of the in-built functions along with the List so that accessing of values from the list and performing different calculation on the list can become easy.

- **Len(list_name)**
- **Append (list_name)**
- **Pop()**
- **Count()**
- **Sort()**
- **Insert()**
- **reverse()**

Name	Purpose
`len(x)`	Calculate the length of the list **x**
`x.append(y)`	Add the object **y** to the end of the list **x**
`x.extend(y)`	Extend the list **x** with the contents of the list **y**
`x.insert(n, y)`	Insert object **y** into the list **x** before position **n**
`x.pop()` `x.pop(n)`	Remove and return the first object in the list Remove and return the object at position **n**
`x.count(y)`	Count the number of times object **y** is in the list
`x.sort()`	Sorts the list **x** in-place
`sorted(x)`	Returns sorted version of **x** (does not change **x**)
`x.reverse()`	Reverses the list **x** in-place

- **list.append(elem)** -- adds a single element to the end of the list. Common error: does not return the new list, just modifies the original.
- **list.insert(index, elem)** -- inserts the element at the given index, shifting elements to the right.
- **list.extend(list2)** adds the elements in list2 to the end of the list. Using + or += on a list is similar to using extend().
- **list.index(elem)** -- searches for the given element from the start of the list and returns its index. Throws a ValueError if the element does not appear (use "in" to check without a ValueError).
- **list.remove(elem)** -- searches for the first instance of the given element and removes it (throws ValueError if not present)
- **list.sort()** -- sorts the list in place (does not return it). (The sorted() function shown below is preferred.)
- **list.reverse()** -- reverses the list in place (does not return it)
- **list.pop(index)** -- removes and returns the element at the given index. Returns the rightmost element if index is omitted (roughly the opposite of append()).

PYTHON PROGARM SHOWING THE USAGE OF LIST FUNCTIONS

```
list = ['larry', 'curly', 'moe']
list.append('shemp')         ## append elem at end
list.insert(0, 'xxx')        ## insert elem at index 0
list.extend(['yyy', 'zzz'])  ## add list of elems at end
print list  ## ['xxx', 'larry', 'curly', 'moe', 'shemp', 'yyy', 'zzz']
print list.index('curly')    ## 2

list.remove('curly')         ## search and remove that element
list.pop(1)                  ## removes and returns 'larry'
print list  ## ['xxx', 'moe', 'shemp', 'yyy', 'zzz']
```

ITERATION OVER LISTS

USING FOR LOOP

1. The python list can be easily rendered using the iterative nature of various loops and various items can be entered in the list by the user at same time.
2. Python's for and in looping constructs are very useful.
3. The *for* construct -- for **var** in **list** -- is an easy way to look at each element in a list (or other collection).
4. For example:-

```
squares = [1, 4, 9, 16]
sum = 0
for num in squares:
  sum += num
print sum  ## 30
```

5. The *in* construct on its own is an easy way to test if an element appears in a list (or other collection) -- **value** in **collection** -- tests if the value is in the collection, returning True/False.

```
list = ['larry', 'curly', 'moe']
 if 'curly' in list:
   print 'yay‘
```

THE RANGE FUNCTION

The range(n) function yields the numbers 0, 1, ... n-1, and range(a, b) returns a, a+1, ... b-1 -- up to but not including the last number. The combination of the for-loop and the range() function allow us to build a traditional numeric for loop:

```
## print the numbers from 0 through 99
 for i in range(100):
   print i
```

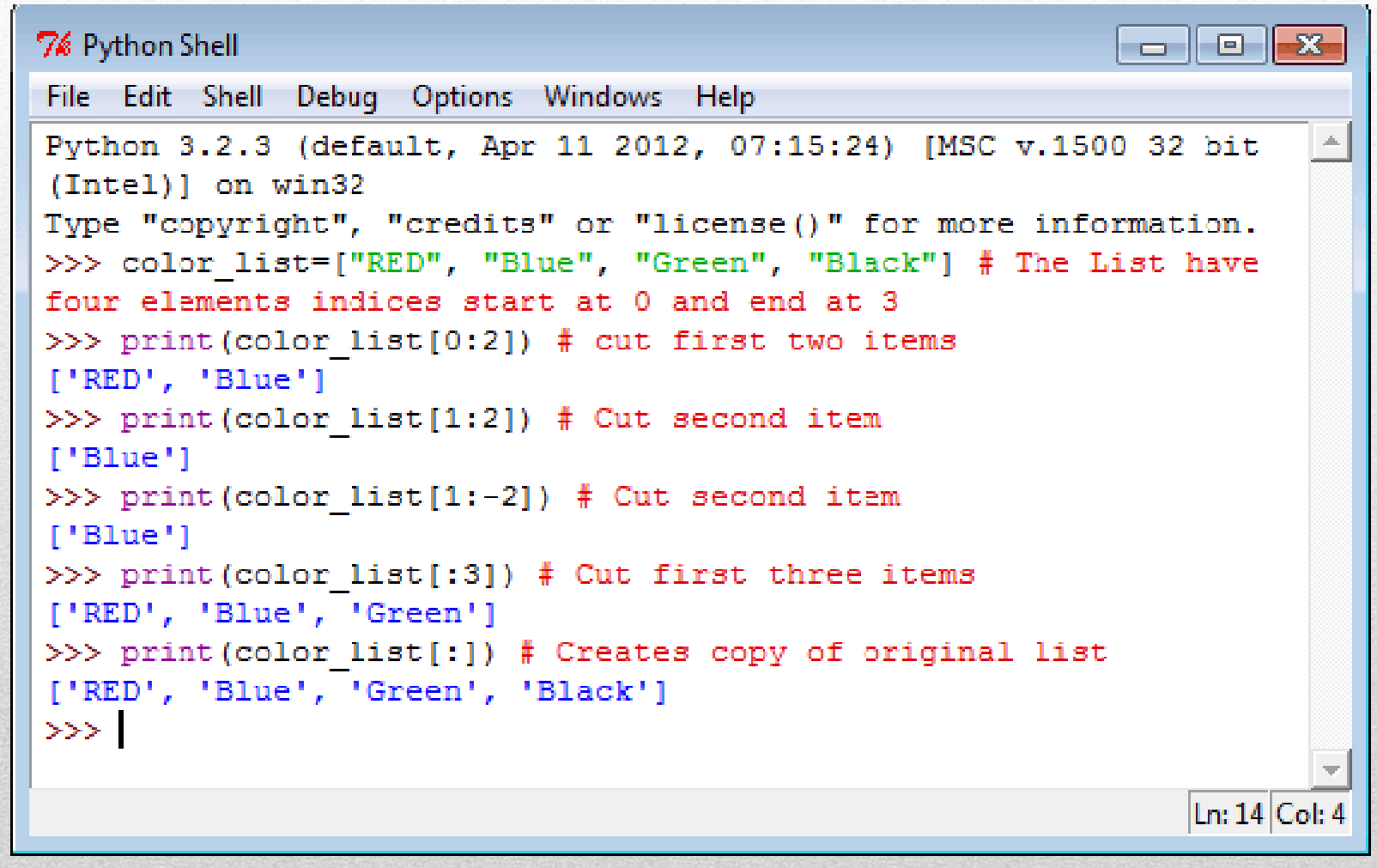
This will output the numbers from 1 to 99 as i will encounter the elements from 1 to 99 one by one and print each element of the list.

WHILE LOOP AND LISTS

Python also has the standard while-loop, and the *break* and *continue* statements work as in C++ and Java, altering the course of the innermost loop. The for/in loops solves the common case of iterating over every element in a list, but the while loop gives us total control over the index numbers.

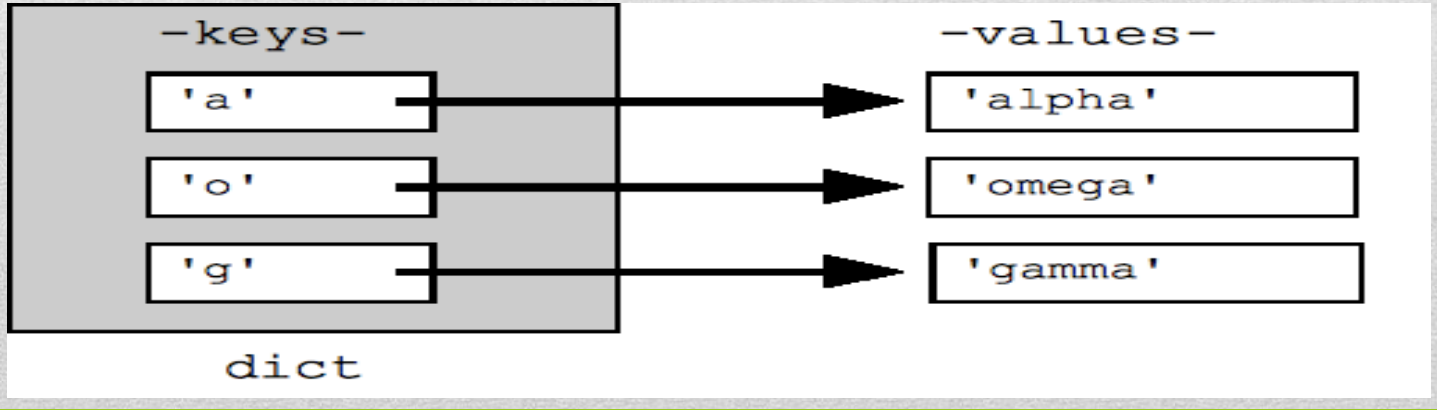
```
## Access every 3rd element in a list
  i = 0
  while i < len(a):
    print a[i]
    i = i + 3
```

Program showing the concept of creating a list and slicing the list into many parts by using positive and negative indexing.

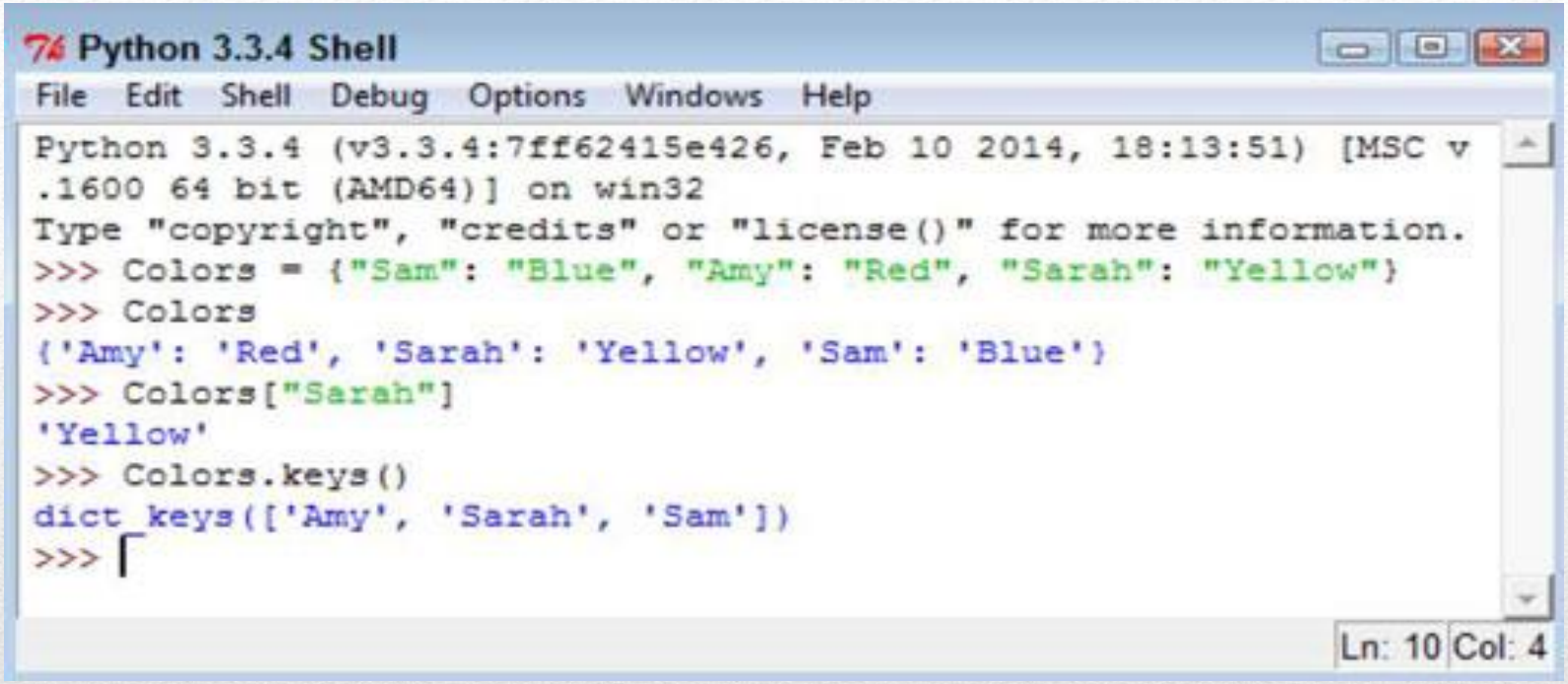


DICTIONARIES IN PYTHON

1. Another useful data type built into Python is the *dictionary*.
2. Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays”.
3. Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tuple contains any mutable object either directly or indirectly, it cannot be used as a key. We can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().



1. It is best to think of a dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary).
2. A pair of braces creates an empty dictionary: {}.
3. Placing a comma-separated list of key : value pairs within the braces adds initial key : value pairs to the dictionary; this is also the way dictionaries are written on output.
4. The main operations on a dictionary are storing a value with some key and extracting the value given the key.
5. It is also possible to delete a key : value pair with del.
6. If we store using a key that is already in use, the old value associated with that key is forgotten.



Python program creating a dictionary named as Colors having the keys and values as:-

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

```
#!/usr/bin/python dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
 print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
dict['Name']: Manni
```

(b) Keys must be immutable which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example:

```
#!/usr/bin/python
dict = {['Name']: 'Zara', 'Age': 7}
 print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last): File "test.py", line 3, in <module> dict =
{['Name']: 'Zara', 'Age': 7}; TypeError: list objects are unhashable
```

Built-in Dictionary Functions & Methods

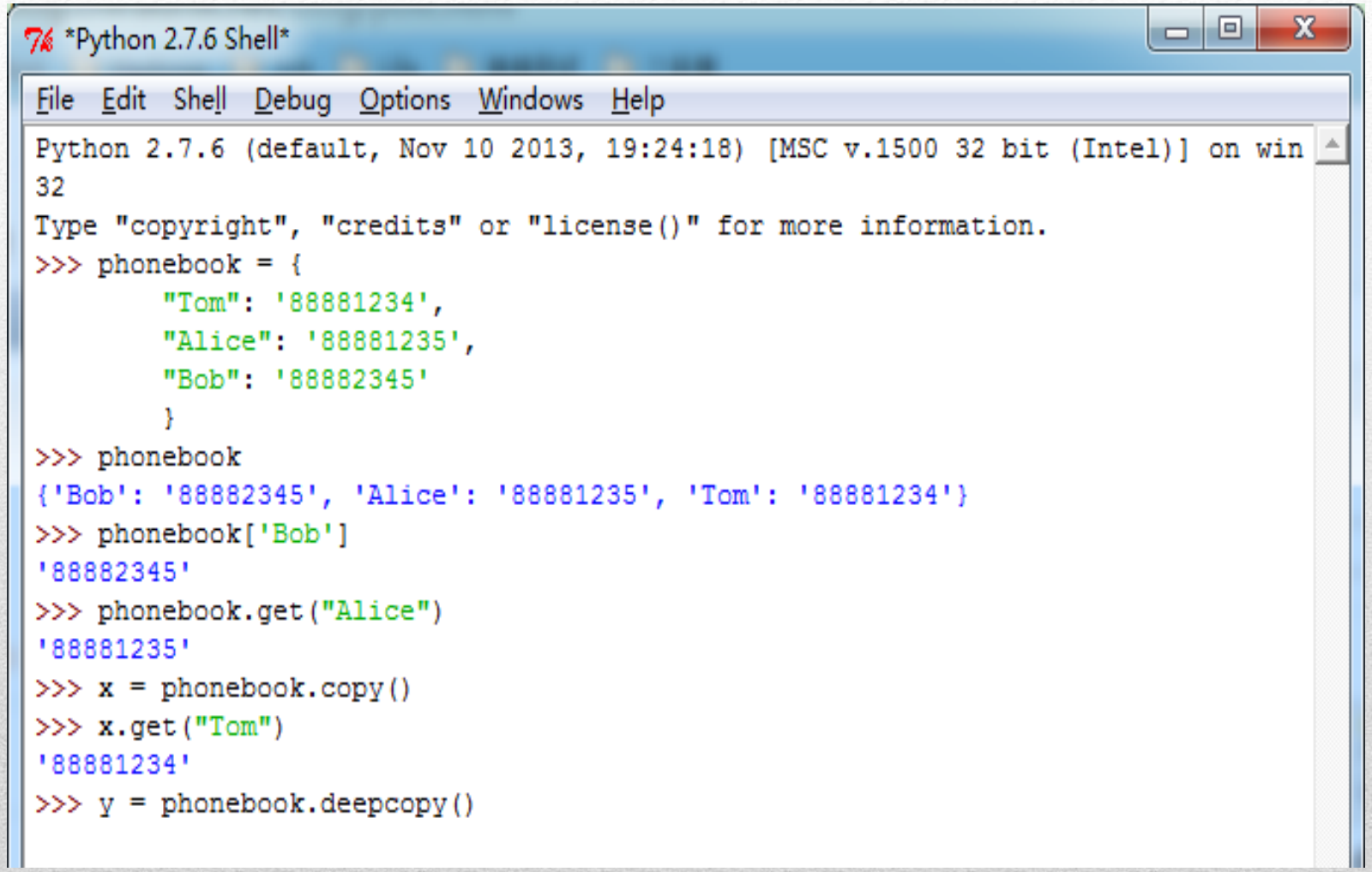
SN	Methods with Description
1	**dict.clear()** Removes all elements of dictionary *dict*
2	**dict.copy()** Returns a shallow copy of dictionary *dict*
3	**dict.fromkeys()** Create a new dictionary with keys from seq and values *set* to *value*.
4	**dict.get(key, default=None)** For *key* key, returns value or default if key not in dictionary
5	**dict.has_key(key)** Returns *true* if key in dictionary *dict*, *false* otherwise
6	**dict.items()** Returns a list of *dict*'s (key, value) tuple pairs
7	**dict.keys()** Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None)** Similar to get(), but will set dict[key]=default if *key* is not already in dict
9	**dict.update(dict2)** Adds dictionary *dict2*'s key-values pairs to *dict*
10	**dict.values()** Returns list of dictionary *dict*'s values

Iterating Through a Dictionary

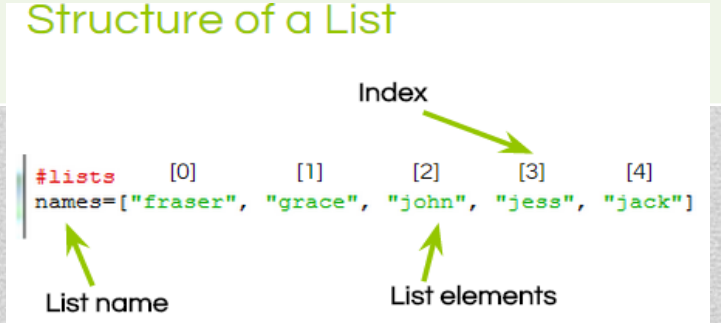
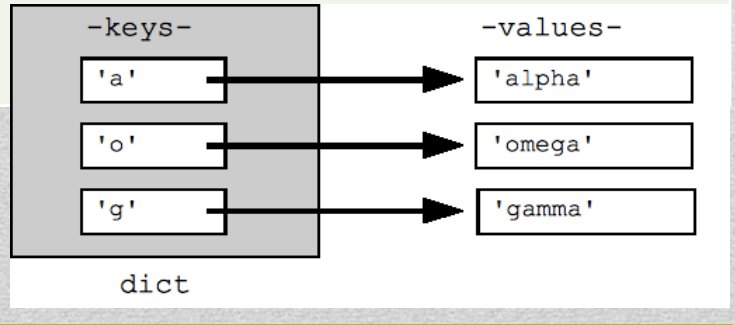
```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
 print(squares[i])
```

Dictionary Membership Test

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
# Output: True
print(1 in squares)
# Output: True
print(2 not in squares)
# membership tests for key only not value
```


Python 2.7.6 Shell
File Edit Shell Debug Options Windows Help
Python 2.7.6 (default, Nov 10 2013, 19:24:18) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> phonebook = {
"Tom": '88881234',
"Alice": '88881235',
"Bob": '88882345'
}
>>> phonebook
{'Bob': '88882345', 'Alice': '88881235', 'Tom': '88881234'}
>>> phonebook['Bob']
'88882345'
>>> phonebook.get("Alice")
'88881235'
>>> x = phonebook.copy()
>>> x.get("Tom")
'88881234'
>>> y = phonebook.deepcopy()

DIFFERENCE BETWEEN LIST AND DICTIONARY

LIST	DICTIONARY
1. List data type in python is a collection of heterogeneous data that represents sequenced objects.	1. Dictionary is the collection of keys and values where every value will have a corresponding key.
2. Python Lists are mutable i.e. changes can be made in list anytime.	2. Dictionary is immutable.
3. Elements of lists are included in square brackets[].	3. Elements of dictionaries are included in curly brackets{}.
4. 	4. 

REFERENCES

1. Learning Python Book by David Ascher and Mark Lutz
2. Fundamentals of Python: First Programs Book by Kenneth Lambert
3. Introduction to computer science using Python by Charles Dierbach
4. **Introduction to Computation and Programming Using Python with Application to Understanding Data by Guttag John V.**
5. **Data Structures and Algorithms in Python by Micheal T.Goodrich**

THANK YOU