

TITLE: Python Programming: Problem Solving, Packages and Libraries

Lecture PPT
Edition

Lists: Learning Objectives

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 1** Compare (i) containers, (ii) sequences, and (iii) maps, and identify the similarities and differences between them.
- LO 2** Explain how a list is a sequence, which is a type of container and why all sequences are containers but all containers may not be sequences.
- LO 3** Understand that Python language has tried to create similar methods of access and traversal of sequences. So, similar functions and methods have been created for the different types of sequences.
- LO 4** Perform some common operations on lists, such as (i) creating lists (ii) list comprehension (iii) traversing over elements of a list (iv) ways of adding to a list.
- LO 5** Use some common list functions/methods, such as sorting, searching, adding, removing and reversing a list.
- LO 6** Discuss other concepts related to lists, such as (i) list in boolean context (ii) assigning a list to a variable versus copying a list (iii) concept of alias (iv) nested lists and using them for matrix operations

Containers/ Collections in Python

- **Containers are objects that hold an arbitrary number of other objects.**
- **Generally, containers provide a way to access the contained objects and to iterate over them.**
- **In general, a collection or a container is an object that contains other objects (or data types, such as int, bool, strings, and so on).**
- **In Python, strings (that is, str), bytes, lists, dictionaries, tuples and sets are all containers.**
- **Collections or containers in Python come in two basic forms: sequences and mappings.**
- **A sequence can be thought of as a string of beads placed one after another and numbered from 0 to n-1.**

Figure showing a Python list as a sequence

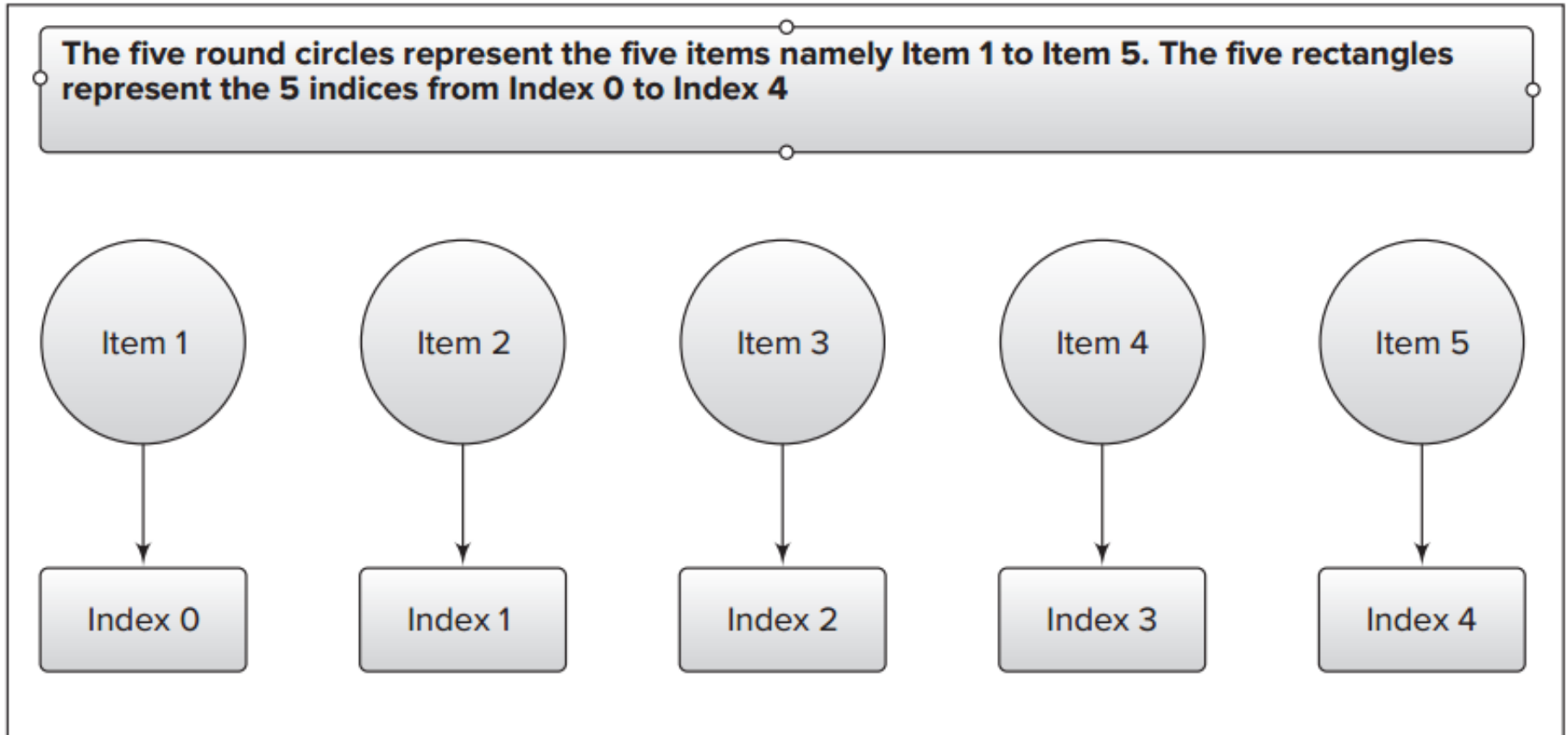


FIGURE 8.1 A list is a sequence and has an index for each item as shown in the figure

Analogy of a “mapping” to a “jar” which has items in it with strings attached to pull out these items

- A mapping on other hand can be thought of as a jar containing items, which have tags.
- The items are inside the jar and the tags are outside.
- If a person wants to pull out a particular item, he sees its tag or key and pulls it out.
- Hence, the access method of a container of type mapping is the tag or the key.

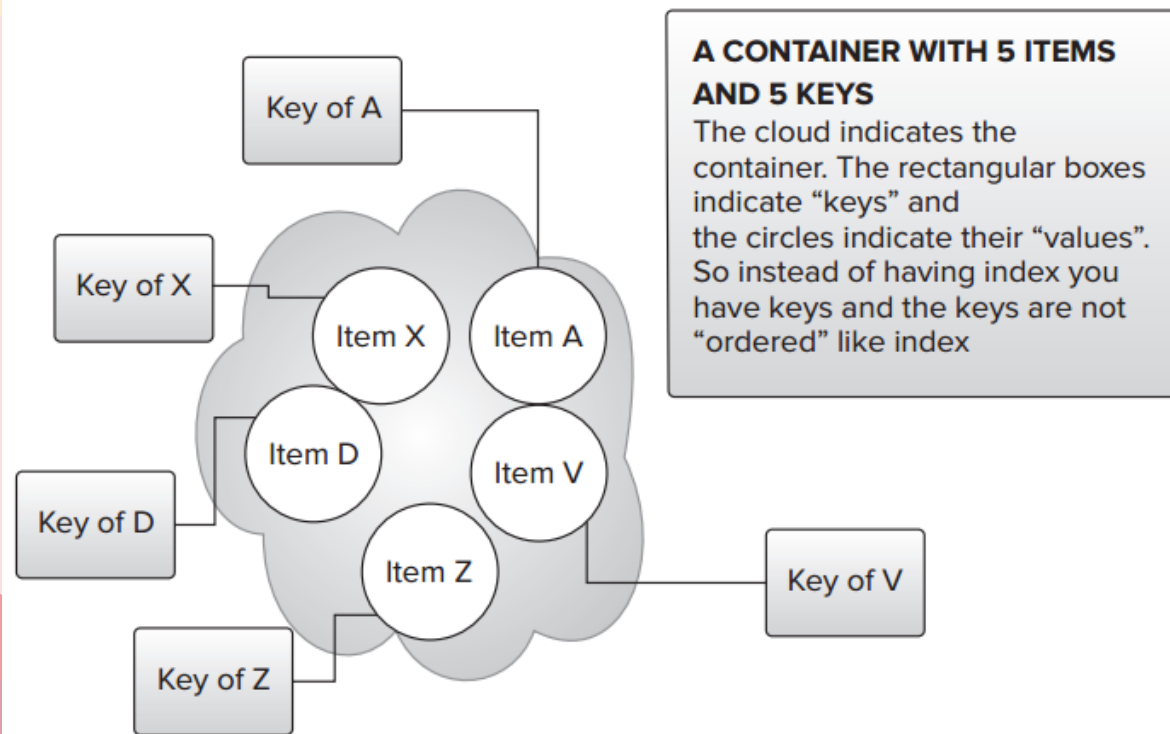


FIGURE 8.2 A dictionary as a container of key-value pairs

How collections differ from containers

- **Access method:** It refers to the manner in which the individual items in the container are accessed. For sequence you may use “index” and for a collection you may use its “key”
- **Storage:** This refers to what the container can store. A container may be restricted to storing only items of a particular type. For instance, a sequence of type string can only store individual characters. Similarly, a sequence of type ‘range’ function can only store integers.
- **Traversal or iteration:** Going over the items of a container (whether sequence or map) is called traversal or iteration. Programming languages, such as Python provide methods or functions to the programmer to go over the various objects in a container. These methods are called iterators. All iterables in Python allow access to elements using the ‘for ... in’ statement.

Some noteworthy aspects of sequences in Python

- There are three basic sequence types: list, tuple, and range objects. Additionally, there is a text sequence type, str (string), which means that a string can be treated as a sequence of characters.
- All sequence types are iterable, that is, they support iterations through their elements.
- A sequence can be indexed by position (0 through to (length – 1)).
- Out of the various sequences, the string type can only contain characters. The range() object (Discussed later) can generate integers. The other two, namely lists and tuples, can contain any object.
- Both strings and tuples are immutable.
- Lists, on the other hand, are mutable. So you can insert, delete, sort or append to a list.

Some sequence operations (Applicable to both mutable and non-mutable)

SOME COMMON SEQUENCE OPERATIONS

x in seq1	If x present in seq1 then True else False.
x not in seq1	If x not present in seq1 then True else false
seq1 + seq2	the “addition” or “concatenation” of seq1 and seq2
(seq1 * n) or (n * seq1)	n copies of seq1 ie seq1 concatenated n times;
seq1[i]	i-th item of seq1, starting from 0 ie origin is 0
seq1[i:j]	slice of seq1 from i to j (including i but excluding j)
seq1[i:j:k]	slice of seq1 from i to j with step k (excluding j)
max(seq1)	Gives the largest item of seq1
min(seq1)	smallest item of seq1 (Use a round bracket not square)
len(seq1)	length of seq1
seq1.index(x)	Gives index of the first instance of x in seq1
seq1.index(x, i)	first instance of x in seq1 at or after index i
seq1.index(x, i, j)	first instance of x in seq1 starting at (and including) index i and before index j (ie excluding index j).
seq1.count(x)	total number of instances of x in seq1

INDICES IN SLICE NOTATION

seq1[:]	slice of seq1 from 0 to the end (copy of itself)
seq1[i:]	slice of seq1 from i to the end
seq1[:j]	slice of seq1 from 0 to j.

DUPLICATE INDEX DEFINES EMPTY SLICE

seq1[i:i]	empty slice.
-----------	--------------

IF INDEX IS NEGATIVE, THE INDEX IS RELATIVE TO THE END OF THE STRING

seq1[-1]	last element of seq1; same as[len(seq1)-1]i-th element of seq1 from the end;
seq1[-i]	same as seq1[len(seq1)-i]

Mutable sequence operations

(Can apply to list but not to strings and tuples)

MUTABLE SEQUENCE OPERATIONS (Can apply to list but not to strings and tuples)	
seq1[i] = x	item i of seq1 is replaced by x.
seq1[i:j] = seq2	slice of seq1 from index i to j is replaced by contents of seq2. Note here seq2 need not be same type as seq1.
del seq1[i:j]	Removes items between index i to j but not including j. Same as seq1[i:j] = []
seq1[i:j:k] = seq2	the elements of seq1[i:j:k] are substituted by those of seq2
del seq1[i:j:k]	removes (with step of k) the elements of seq1[i:j:k] from the list.
seq1.copy()	creates a copy of seq1. Same as seq1[:]. The copy() method does not take any parameters
seq1.clear()	removes all items from seq1.
seq1.append(x)	It adds the item x at the end of the sequence;
seq1.extend(seq2)	extends seq1 with the contents of seq2;
seq1.insert(i, x)	inserts x into seq1 at the index given by i; same as seq1[i:i] = [x]
seq1.pop([i])	retrieves the item at index i and also removes it from seq1
seq1.remove(x)	The remove() method searches for item x in the list and removes the first matching element.
seq1.reverse()	reverses the items of seq1 in place

Common sequences in Python

Type	Construction	Mutable	Iterable
List	<code>[], [a], [a,b], list(), list(t)</code>	mutable	All can be used as <i>iterable</i> in for-loops
tuple	<code>(), (a), (a,b), tuple(), tuple(t)</code>	immutable	
str	<code>'ab', '''ab''', "ab", """ab""", str(), str(x)</code>	immutable	
range	<code>range(n), range(m,n), range(m,n,k)</code>	immutable	

Some points to note about lists

- Items in a list always retain their original order because a list is an ordered set of items.
- The first item of a list always has index 0.
- If the list has n elements, the index of last item is always $(n-1)$.
- The items in the list do not have to be unique.
- A list can be thought of as a zero-index based array.
- Python permits use of negative index also. Using negative index, the index of the last item in the array is -1 . As you decrease index from -1 to -2 , you move from the last to the second last item and so on. As you decrease the negative index, you move over the list backwards.
- So the permissible index for a list of n items is $(-n, (n-1))$. Where the first item in a list of n elements has a positive index of 0 or a negative index of $-n$. Similarly, the last item in the array has a positive index of $n-1$ or a negative index of -1 .
- If you exceed the range in either direction, the Python interpreter will throw an error.

Various ways of constructing lists

CONSTRUCTING LISTS	
<code>[]</code>	creates an empty list
<code>[a]</code>	creates a list consisting of single element; same as <code>[a,]</code>
<code>[a, b, c]</code>	creates a list consisting of several elements; same as <code>[a,b,c,]</code>
<code>list()</code>	creates an empty list
<code>list(t)</code>	creates a list from the iterable t; same as <code>[x for x in t]</code>
<code>f(x)</code>	using functions returning lists, e.g. built-in function <code>sorted()</code>

List comprehension

Comprehensions are constructs that allow sequences to be built from other sequences. To create a list from another sequence, you need the following four things:

- An input sequence
- A variable representing a member of the input sequence
- An output expression
- An optional condition (Also called a predicate expression)

Example of how list comprehension works

The example code below uses list comprehension to construct a list L2 from a list L1. The condition is that only odd numbers from L1 are selected and they are “squared”.

```
1 >>> L1 = [1,2,3,4,5,6,7,8,9,10]
2 >>> L2 = [x**2 for x in L1 if x//2 != 0]
3 >>> L2
4 [4, 9, 16, 25, 36, 49, 64, 81, 100]
```

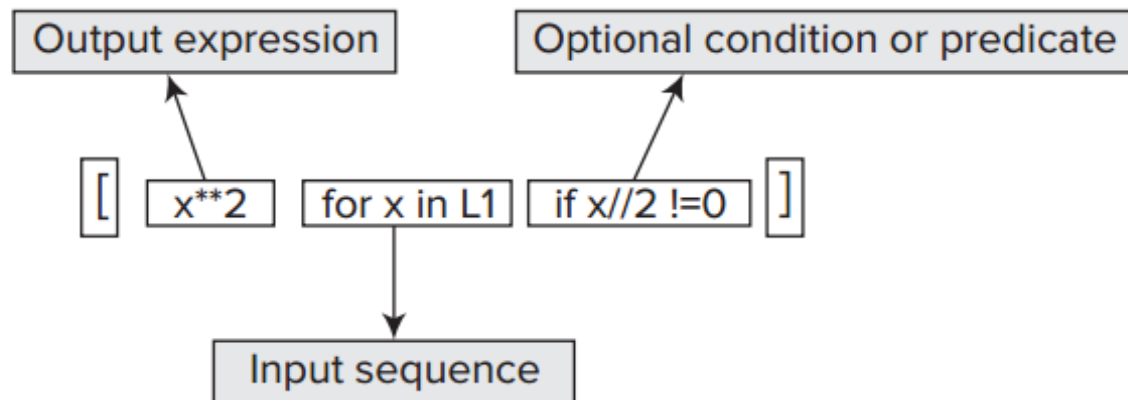


FIGURE 8.3 Three parts of syntax of ‘list comprehension’ namely (i) output expression (ii) input sequence and (iii) the optional condition or predicate

Traversing the list (ie iterating over items of a list)

Traversing a list means accessing elements of a list one by one. There are various ways of traversing a list. Some of the common methods are as follows:

- Using a while loop along with len() function to get the length of the list
- Using for/ in combination along with range() function
- Using for/in combination treating the list as a collection (that is, without using index of the items)

Note: Only the various ways are enumerated here. For examples see the book

Adding items to a list

There are four ways of adding to a list, as follows:

- **Using the + operator**
- **List function: insert()**
- **List function: append()**
- **List function: extend()**

Note: Only the various ways are enumerated here. For examples see the book

Example showing use of list method insert()

The following script takes a list

```
myL = ['A', 'quick', 'fox']
```

and shows how the string 'brown' is inserted at index 2 using method

```
myL.insert(2, 'brown')
```

```
1 >>> myL = ['A', 'quick', 'fox']
2 >>> myL.insert(2, 'brown') # Will insert 'brown' at index 2
3 >>> myL
4 ['A', 'quick', 'brown', 'fox']
```

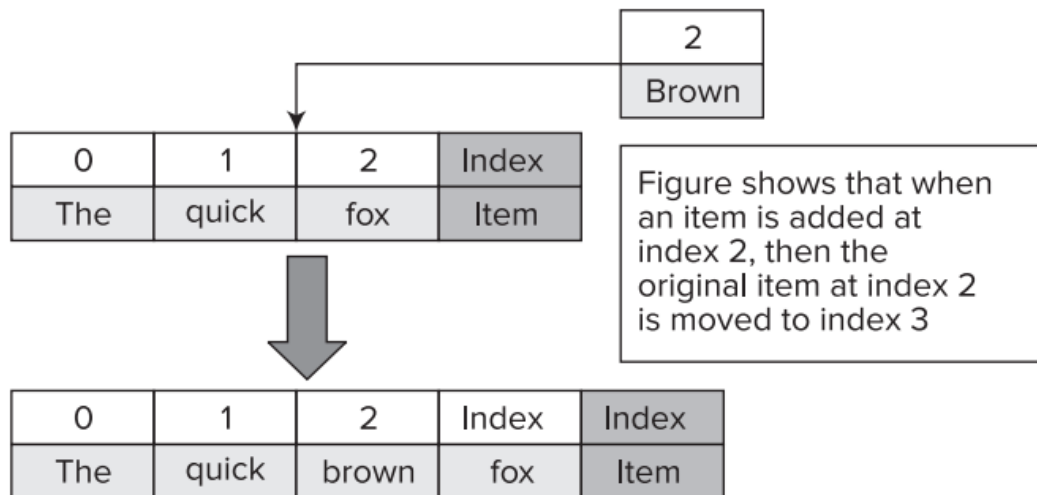


FIGURE 8.4 The item originally present at index 2 is bumped up by 1 (moves to index 3) when an item is added at a particular index

Concept of aliasing

- **An alias is an alternate name for the same object.**
- **In Python, variables refer to objects, if you assign a variable to another, both variables will refer to the same object and an alias is created.**
- **If the object being referred to is immutable, (that is, it cannot be modified in place), then creating an alias will not make a difference.**
- **However, if the object being referred to is mutable, then aliasing can lead to unexpected bugs.**

Following topics are not discussed in this presentation. (For reasons given in brackets)

- **List slicing (same as slicing of a string and already discussed)**
- **List extended slicing (same as extended slicing of a string and already discussed)**
- **List functions and methods. (They are quite straight forward and given well in the book along with examples)**
- **Searching, adding, removing, reversing,**

Nested lists and using them as Matrix/ Matrices

(Explained with example from book)

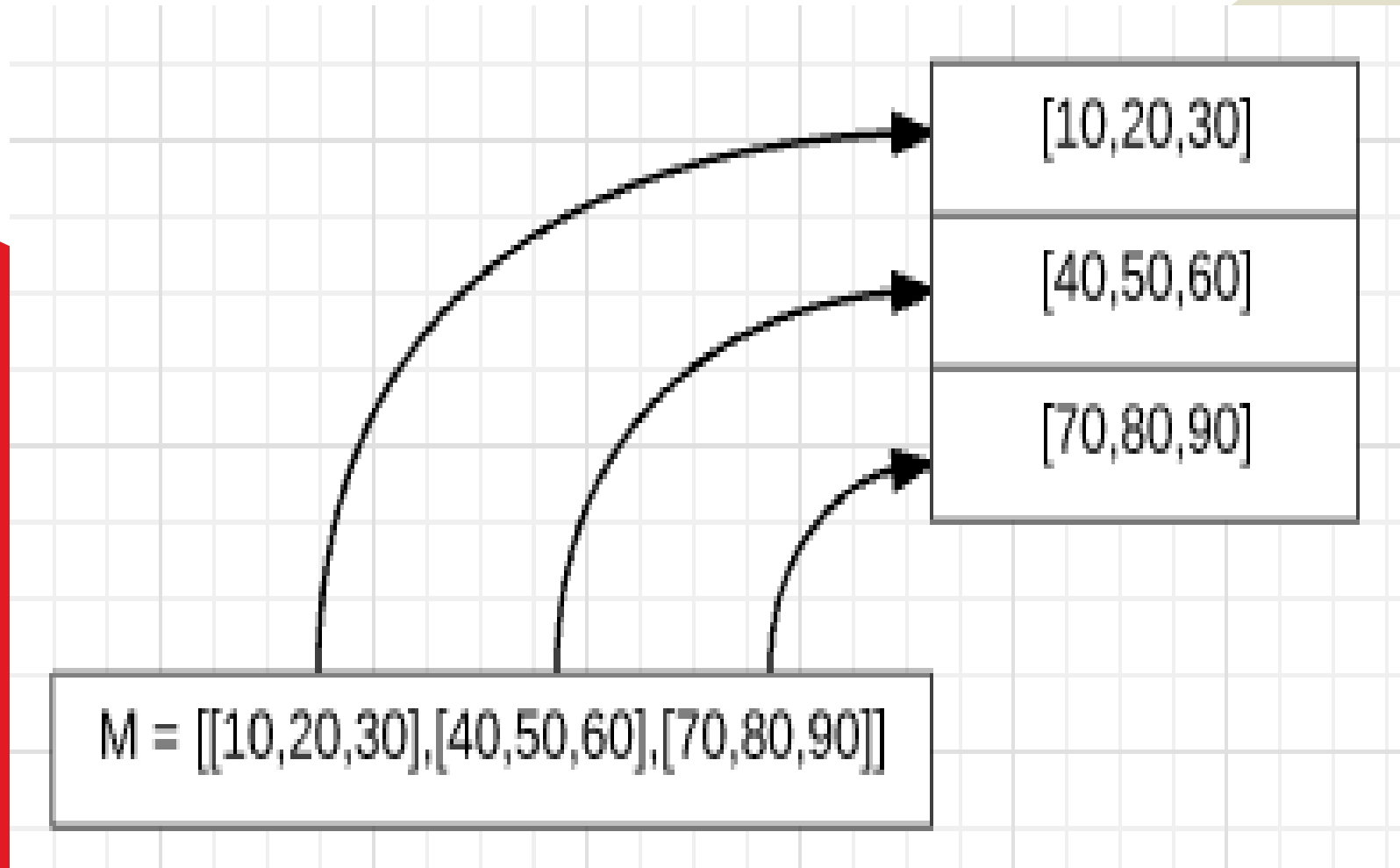
- In Python, lists can be nested to get multi-dimension lists.
- A two-dimension list can be used as a matrix. For instance, a basic 3 x3 matrix in Python can be represented as:
$$M = [[10,20,30],[40,50,60],[70,80,90]]$$
- Here [10, 20, 30] is the first row, [40, 50, 60] is the second row and [70, 80, 90] is the third row. The rows and columns in matrices in maths are represented using notation $r \times c$ where r represents the rows and c represents the columns. Further note that the row and column numbers start from 1, while the lists used to represent matrices start from index 0.

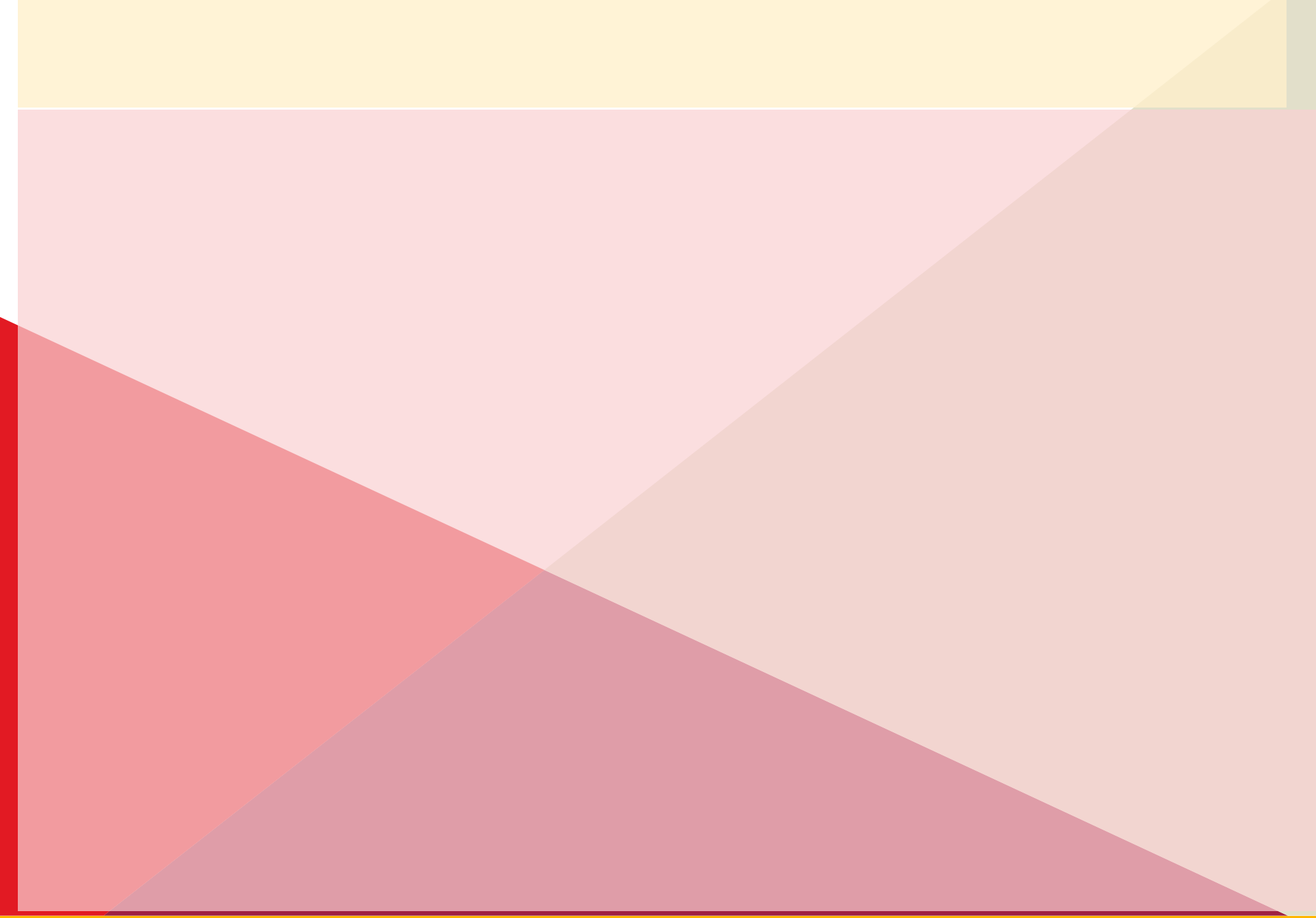
The following script shows how this matrix is represented as a two-dimension Python list

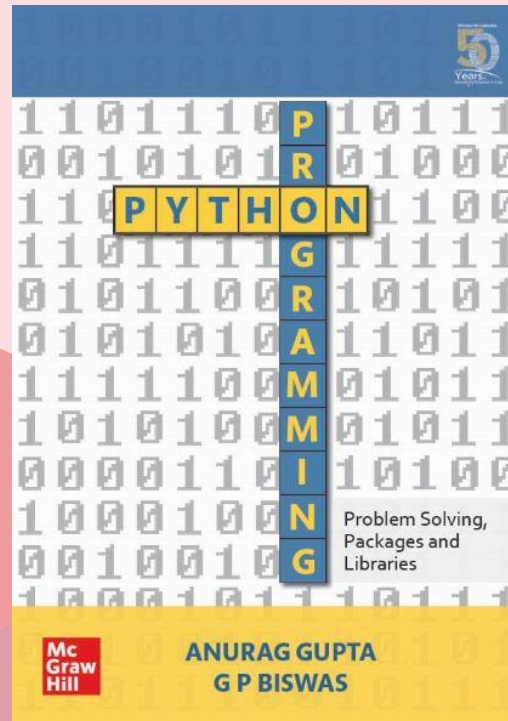
```
1 >>>M = [[10, 20, 30],    # The outer bracket is not closed. Idle expects more input
2           [40, 50, 60],
3           [70, 80, 90]]  # Now outer bracket closed so end of input
4 >>>
```

Figure shows the rows and columns of Matrix

$M = [[10,20,30],[40,50,60],[70,80,90]]$










Because learning changes everything.®

Thank You!

For any queries or feedback contact us at:

 support.india@mheducation.com

 1800-103-5875

 www.mheducation.co.in

in

