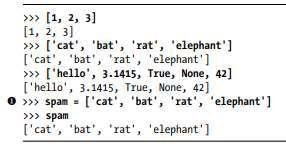
## MODULE 02 CHAPTER 1: LISTS

### The List Data Type

* + - A list is a value that contains multiple values in an ordered sequence.
    - List can be ordered collection of homogenous and nonhomogeneous elements
    - a list is a data structure that allows you to store and manipulate a collection of elements.
      * Lists are ordered, mutable, and can

contain elements of different data types. Each element in a list has an index, which represents its position in the list.

* + - A list value looks like this: ['cat', 'bat', 'rat', 'elephant'].
    - A list begins with an opening square bracket and ends with a closing square bracket, [].
    - Values inside the list are also called items and are separated with commas.



  ❶ is still assigned only one value: the list value(contains multiple values).

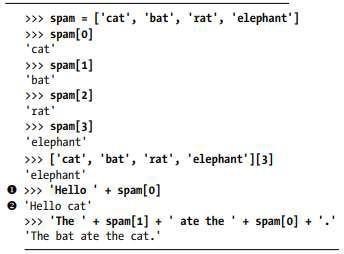
 

### Getting Individual Values in a List with Indexes

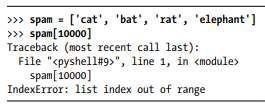
* + - Say you have the list ['cat', 'bat', 'rat', 'elephant'] stored in a variable named spam.
    - The Python code spam[0] would evaluate to 'cat', and spam[1] would evaluate to 'bat', and so on.



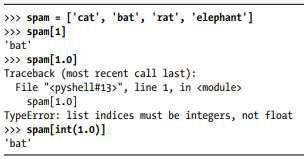
* + - The first value in the list is at index 0, the second value is at index 1, and the third value is at index 2, and so on.
    - For example, type the following expressions into the interactive shell.



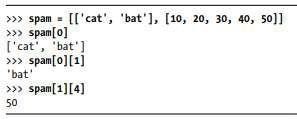
* + - The expression 'Hello ' + spam[0] evaluates to 'Hello ' + 'cat' because spam[0] evaluates to the string 'cat'. This expression in turn evaluates to the string value 'Hello cat'.
    - If we use an index that exceeds the number of values in the list value then, python gives IndexError.



* + - Indexes can be only integer values, not floats. The following example will cause a TypeError error:



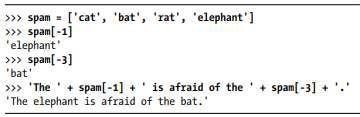
* + - Lists can also contain other list values. The values in these lists of lists can be accessed using multiple indexes.



* + - The first index dictates which list value to use, and the second indicates the value within the list value. **Ex**, spam[0][1] prints 'bat', the second value in the first list.

### Negative Indexes

* + - We can also use negative integers for the index. The integer value -1 refers to the last index in a list, the value -2 refers to the second-to-last index in a list, and so on.

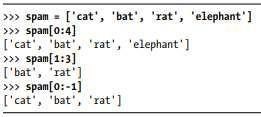


### Getting Sublists with Slices

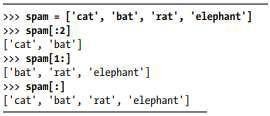
* + - An index will get a single value from a list, a slice can get several values from a list, in the form of a new list.
    - A slice is typed between square brackets, like an index, but it has two integers separated by a colon.

##### Difference between indexes and slices.

* spam[2] is a list with an index (one integer).
* spam[1:4] is a list with a slice (two integers).
  + - In a slice, the first integer is the index where the slice starts. The second integer is the index where the slice ends (but will not include the value at the second index).



* + -  As a shortcut, we can leave out one or both of the indexes on either side of the colon in the slice.
* Leaving out the first index is the same as using 0, or the beginning of the list.
* Leaving out the second index is the same as using the length of the list, which will slice to the end of the list.



In genral:

Indexing

is the process of accessing individual elements in a list by their

position using an index. In Python, list indices start from 0 for the

first element and go up to len(list) - 1 for the last element. You can use square brackets [] to access elements by their index.

Example:

python

my\_list = [1, 2, 3, 4, 5]

print(my\_list[0]) # Output: 1 print(my\_list[2]) # Output: 3

print(my\_list[-1]) # Output: 5 (negative index to access the last element)

Slicing:

Slicing

is the process of extracting a portion of a list by specifying a range of indices. It allows you to create a new list containing a subset of

the original list. The syntax for slicing is list[start:end:step], where start is the starting index, end is the ending index (exclusive), and step is the step size.

Example:

python

my\_list = [1, 2, 3, 4, 5]

print(my\_list[1:4]) # Output: [2, 3, 4] (slicing from index 1 to 4 -1)

print(my\_list[:3]) # Output: [1, 2, 3] (slicing from the beginning to index 3 -1) print(my\_list[2:]) # Output: [3, 4, 5] (slicing from index 2 to the end)

print(my\_list[::2]) # Output: [1, 3, 5] (slicing with a step size of 2)

Slicing

allows you to extract sublists from a list based on your requirements,

making it convenient to work with specific sections of a list.

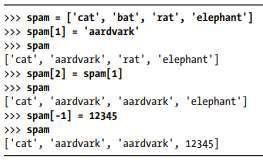
### Getting a List’s Length with len()

* + - The len() function will return the number of values that are in a list value.



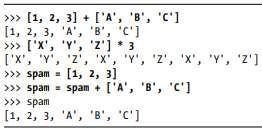
### Changing Values in a List with Indexes

* + - We can also use an index of a list to change the value at that index.
    - **Ex:** spam[1] = 'aardvark' means “Assign the value at index 1 in the list spam to the string 'aardvark'.”



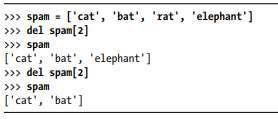
### List Concatenation and List Replication

* + - The + operator can combine two lists to create a new list value in the same way it combines two strings into a new string value.
    - The \* operator can also be used with a list and an integer value to replicate the list.



### Removing Values from Lists with del Statements

* + - The del statement will delete values at an index in a list.



* + - The del statement can also be used to delete a variable After deleting if we try to use the variable, we will get a NameError error because the variable no longer exists.
    - In practice, you almost never need to delete simple variables.
    - The del statement is mostly used to delete values from lists.

### Using for Loops with Lists

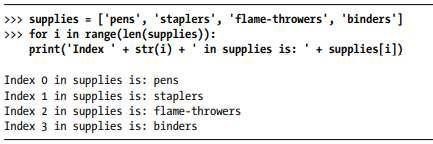
* + - A for loop repeats the code block once for each value in a list or list-like value.

**Program**

**Output:**



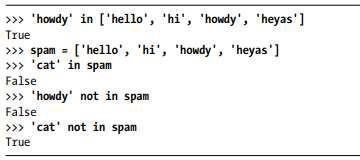
* + - A common Python technique is to use range (len(someList)) with a for loop to iterate over the indexes of a list.



* + - The code in the loop will access the index (as the variable i), the value at that index (as supplies[i]) and range(len(supplies)) will iterate through all the indexes of supplies, no matter how many items it contains.

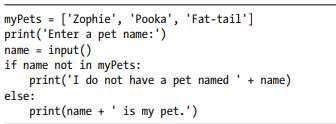
### The in and not in Operators

* + - We can determine whether a value is or isn’t in a list with the in and not in operators.
    - **in** and **not in** are used in expressions and connect two values: a value to look for in a list and the list where it may be found and these expressions will evaluate to a Boolean value.



* + - The following program lets the user type in a pet name and then checks to see whether the name is in a list of pets.

**Program**

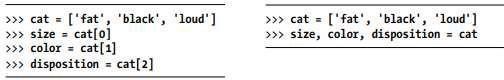


**Output**

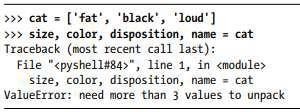


### The Multiple Assignment Trick

* + - The multiple assignment trick is a shortcut that lets you assign multiple variables with the values in a list in one line of code.

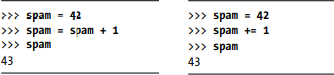


* + - Instead of left-side program wecould type the right-side program to assignmentmultiple variables but the number of variables and the length of the list must be exactly equal, or Python will give you a ValueError:

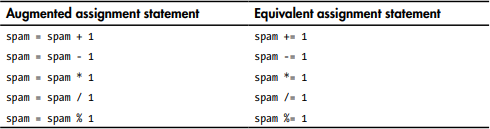


### Augmented Assignment Operators

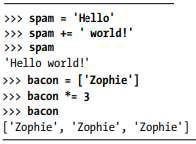
* + - When assigning a value to a variable, we will frequently use the variable itself



* + - Instead of left-side program we could use right-side program i.e., with the augmented assignment operator += to do the same thing as a shortcut.
    - The Augmented Assignment Operators are listed in the below table:



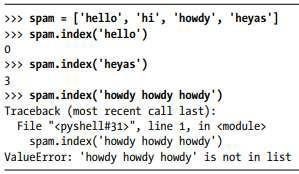
* + - The += operator can also do string and list concatenation, and the \*= operator can do string and list replication.



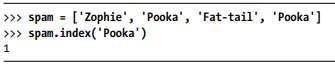
* 1. **Methods**
     + A method is same as a function, except it is “called on” a value.
     + The method part comes after the value, separated by a period.
     + Each data type has its own set of methods.
     + The list data type has several useful methods for finding, adding, removing, and manipulating values in a list.

### Finding a Value in a List with the index() Method

* + - List values have an index() method that can be passed a value, and if that value exists in the list, the index of the value is returned. If the value isn’t in the list, then Python produces a ValueError error.

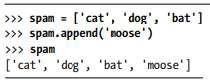


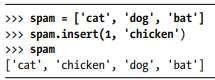
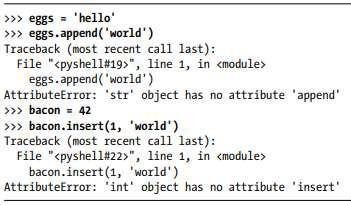
* + - When there are duplicates of the value in the list, the index of its first appearance is returned.



### Adding Values to Lists with the append() and insert() Methods

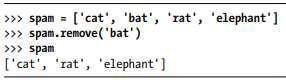
* + - To add new values to a list, use the append() and insert() methods.
    - The append() method call adds the argument to the end of the list.



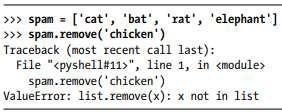
* + - The insert() method can insert a value at any index in the list. The first argument to insert() is the index for the new value, and the second argument is the new value to be inserted.
    - Methods belong to a single data type.
    - The append() and insert() methods are list methods and can be called only on list values, not on other values such as strings or integers.

### Removing Values from Lists with remove()

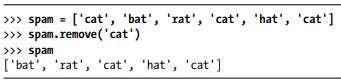
* + - The remove() method is passed the value to be removed from the list it is called on.



* + - Attempting to delete a value that does not exist in the list will result in a ValueError error.



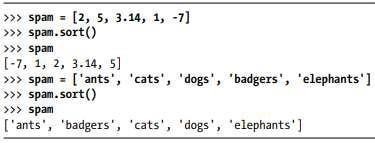
* + - If the value appears multiple times in the list, only the first instance of the value will be removed.

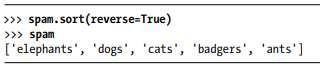


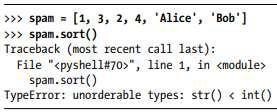
* + - The del statement is good to use when you know the index of the value you want to remove from the list. The remove() method is good when you know the value you want to remove from the list.

### Sorting the Values in a List with the sort() Method

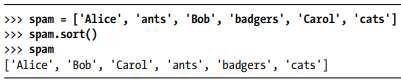
* + - Lists of number values or lists of strings can be sorted with the sort() method.



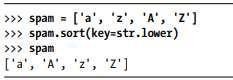
* + - You can also pass True for the reverse keyword argument to have sort() sort the values in reverse order.
    - There are three things you should note about the sort() method.
* **First,** the sort() method sorts the list in place; don’t try to return value by writing code like spam = spam.sort().
* **Second,** we cannot sort lists that have both number values and string values in them.



o **Third,** sort() uses “ASCIIbetical order(upper case)” rather than actual alphabetical order(lower case) for sorting strings.

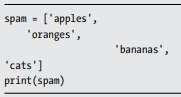


* + - If we need to sort the values in regular alphabetical order, pass str.lower for the key keyword argument in the sort() method call.



### Exceptions to Indentation Rules in Python

* + - The amount of indentation for a line of code tells Python what block it is in.
    - lists can actually span several lines in the source code file. The indentation of these lines do not matter; Python knows that until it sees the ending square bracket, the listis not finished.

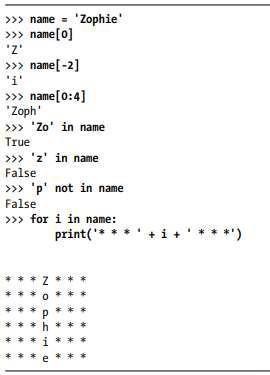


* + - We can also split up a single instruction across multiple lines using the \ line continuation character at the end.



### List-like Types: Strings and Tuples

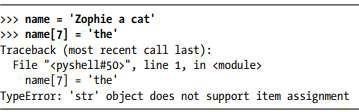
* + - Lists aren’t the only data types that represent ordered sequences of values.
    - **Ex,** we can also do these with strings: indexing; slicing; and using them with for loops, with len(), and with the in and not in operators.



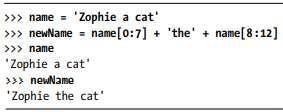
### Mutable and Immutable Data Types

**String**

* + - However, a string is immutable: It cannot be changed. Trying to reassign a single character in a string results in a TypeError error.



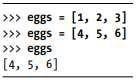
* + - The proper way to “mutate” a string is to use slicing and concatenation to build a new string by copying from parts of the old string.



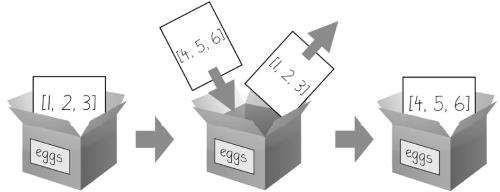
* + - We used [0:7] and [8:12] to refer to the characters that we don’t wish to replace. Notice that the original 'Zophie a cat' string is not modified because strings areimmutable.

**List**

* + - A list value is a mutable data type: It can have values added, removed, or changed.

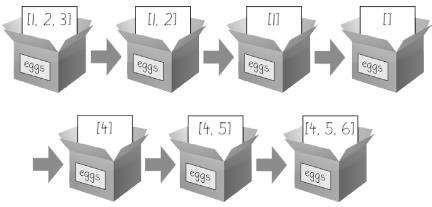
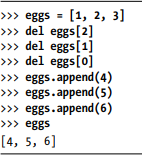


* + - The list value in eggs isn’t being changed here; rather, an entirely new and different list value ([4, 5, 6]) is overwriting the old list value ([1, 2, 3]).



**Figure:** When eggs = [4, 5, 6] is executed, the contents of eggs are replaced with a new list value.

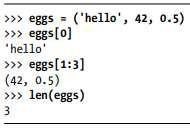
* + - If we want to modify the original list in eggs to contain [4, 5, 6], you would have to delete the items in that and then add items to it.



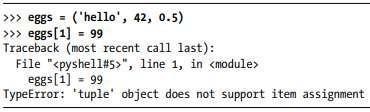
**Figure:** The del statement and the append() method modify the same list value in place.

### The Tuple Data Type

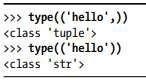
* + - The tuple data type is almost identical to the list data type, except in two ways.
    - **First**, tuples are typed with parentheses, ( and ), instead of square brackets, [ and ].



* + - **Second**, benefit of using tuples instead of lists is that, because they are immutable and their contents don’t change. Tuples cannot have their values modified, appended, or removed.

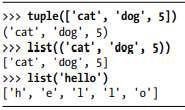


* + - If you have only one value in your tuple, you can indicate this by placing a trailing comma after the value inside the parentheses.



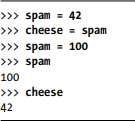
### Converting Types with the list() and tuple() Functions

* + - The functions list() and tuple() will return list and tuple versions of the values passed to them.

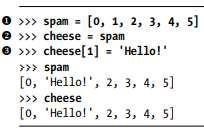


### References

* + - As , variables store strings and integer values.



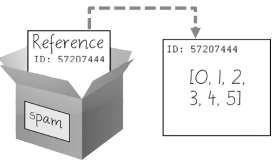
* + - We assign 42 to the spam variable, and then we copy the value in spam and assign it to the variable cheese. When we later change the value in spam to 100, this doesn’t affect the value in cheese. This is because spam and cheese are different variables that store different values.
    - But lists works differently. When we assign a list to a variable, we are actually assigning a list reference to the variable. A reference is a value that points to some bit of data, and a list reference is a value that points to a list.



* + - When we create the list ❶, we assign a reference to it in the spam variable. But the next line copies only the list reference in spam to cheese, not the list value itself. This means the values stored in spam and cheese now both refer to the same list.
    - There is only one underlying list because the list itself was never actually copied. So

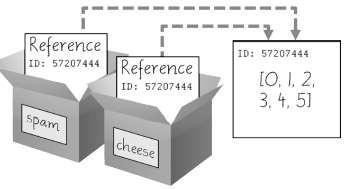
when we modify the first element of cheese, we are modifying the same list that spam refers to.

* + - List variables don’t actually contain lists—they contain references to lists.



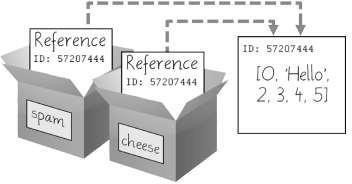
**Figure:** spam = [0, 1, 2, 3, 4, 5] stores a reference to a list, not the actual list.

* + - The reference in spam is copied to cheese. Only a new reference was created and stored in cheese, not a new list.



**Figure:** spam = cheese copies the reference, not the list

* + - When we alter the list that cheese refers to, the list that spam refers to is also changed, because both cheese and spam refer to the same list.

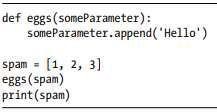


**Figure:** cheese[1] = 'Hello!' modifies the list that both variables refer to

* + - Variables will contain references to list values rather than list values themselves.
    - But for strings and integer values, variables will contain the string or integer value.
    - Python uses references whenever variables must store values of mutable data types, such as lists or dictionaries. For values of immutable data types such as strings, integers, or tuples, Python variables will store the value itself.

### Passing References

* + - References are particularly important for understanding how arguments get passed to functions.
    - When a function is called, the values of the arguments are copied to the parameter variables.

 **Program** O**utput**

* + - when eggs() is called, a return value is not used to assign a new value to spam.
    - Even though spam and someParameter contain separate references, they both refer to the same list. This is why the append('Hello') method call inside the function affects the list even after the function call has returned

### The copy Module’s copy() and deepcopy() Functions

*The `copy` module in Python provides functions for creating shallow and deep copies of objects. Both shallow and deep copies are used to create new copies of objects, but they differ in the way they handle nested objects.*

*The `copy()` function in the `copy` module is used to create shallow copies of objects. A shallow copy creates a new object and then inserts references to the nested objects found in the original object. This means that changes made to the nested objects in the original object will be reflected in the shallow copy. Here is an example:*

*python import copy*

*original\_list = [1, 2, [3, 4]] shallow\_copy = copy.copy(original\_list)*

*original\_list[2].append(5)*

*print(original\_list) # Output: [1, 2, [3, 4, 5]]*

*print(shallow\_copy) # Output: [1, 2, [3, 4, 5]]*

*In the example above, the `copy()` function creates a shallow copy of the `original\_list`. When we append the number 5 to the nested list in the `original\_list`, the same change is reflected in the shallow copy.*

*On the other hand, the `deepcopy()` function in the `copy` module is used to create deep copies of objects. A deep copy creates a new object and recursively inserts copies of the nested objects found in the original object. This means that changes made to the nested objects in the original object will not be reflected in the deep copy. Here is an example:*

*python import copy*

*original\_list = [1, 2, [3, 4]]*

*deep\_copy = copy.deepcopy(original\_list) original\_list[2].append(5)*

*print(original\_list) # Output: [1, 2, [3, 4, 5]]*

*print(deep\_copy) # Output: [1, 2, [3, 4]]*

*In the example above, the `deepcopy()` function creates a deep copy of the `original\_list`. When we append the number 5 to the nested list in the `original\_list`, the deep copy remains unaffected.*

*In summary, the `copy()` function creates shallow copies of objects, while the `deepcopy()`*

*function creates deep copies of objects. The choice between shallow and deep copies depends on whether you want changes made to nested objects to be reflected in the copied object or not.*

##### .Write a python program to check whether the search element is present in the list or what

def search\_element(search\_list, element): for item in search\_list:

if item == element: return True

return False

# Example usage my\_list = [1, 2, 3, 4, 5]

search\_element = 3

if search\_element(my\_list, search\_element):

print(f"The element {search\_element} is present in the list.") else:

print(f"The element {search\_element} is not present in the list.")

##### .For a given list num=[45,22,14,65,97,72], write a python program to replace all the integers divisible by 3 with “ppp” and all integers divisible by 5 with “qqq” and replace all the integers divisible by both 3 and 5 with “pppqqq” and display the output

num = [45, 22, 14, 65, 97, 72]

result = []

for n in num:

if n % 3 == 0 and n % 5 == 0: result.append("pppqqq")

elif n % 3 == 0: result.append("ppp")

elif n % 5 == 0:

result.append("qqq") else:

result.append(n) print(result)

##### What is the function to find the number of elements in the list? Write a program to count the number of elements in the list without using builtin function.

my\_list = [1, 2, 3, 4, 5]

count = 0

for \_ in my\_list: count += 1

print(f"Number of elements in the list: {count}")

**Tuples are immutable. Explain with Python programming example.**

Tuples are basically a [data type in python](https://www.toppr.com/guides/computer-science/introduction-to-python/getting-started-with-python/data-types-in-python/), We can also define tuples as lists that we cannot change.Therefore, we can call them immutable tuples. Hence, tuples are not modifiable in nature. These immutable tuples are a kind of group data type.

Example 1: Tuple with integers as elements

>>>tup = (22, 33, 5, 23)

>>>tup

(22, 33, 5, 23)

Example 2: Tuple with mixed data type

>>>tup2 = (‘hi’, 11, 45.7)

>>>tup2

(‘hi’, 11, 45.7)

Immutable Tuples

>>>tuple1 = (1, 2, 33, 44, 6)

>>>tuple1[4] = 10

Type Error: ‘tuple’ object does not support item assignment

**For a=[‘hello’, ‘how’, [1,2,3], [[10,20,30]]] what is the output of following statement (i) print( a[ : :**

**] ) (ii) print(a[-3][0]) (iii) print(a[2][ : -1]) (iv) print(a[0][ : : -1])**

**OUTPUT:**

>>> print( a[ : : ] )

[‘hello’, ‘how’, [1,2,3], [[10,20,30]]]

>>> print(a[-3][0]) h

>>> print(a[2][ : -1]) [ 1, 2 ]

>>> print(a[0][ : : -1]) Olleh

**Write a python program to accept n numbers and store them in a list. Then print the list without ODD numbers in it.**

n = int(input(“Enter the value of N: “)) list1 = [ ]

evenList = [ ]

print(f“Enter the {n} numbers ”) for i in range( n ) :

num = int(input( f“Enter number {i+1} :”)) list1.append(num)

for i in range(n) :

if list1[ i ] % 2 == 0: evenList.append(list1[ i ])

print(“List without ODD numbers: “) print(evenList)

## CHAPTER2: DICTIONARIES

## The Dictionary Data Type

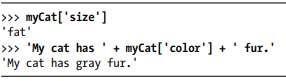
* + - A dictionary is a collection of many homogenous and heterogenous values.

Pairs dictionaries are mutable data structures that allow you to store key-value.

* + - Indexes for dictionaries can use many different data types, not just integers. Indexes for dictionaries are called keys, and a key with its associated value is called a key-value pair.
    - A dictionary is typed with braces, {}.



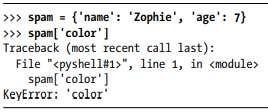
* + - This assigns a dictionary to the myCat variable. This dictionary’s keys are 'size', 'color', and 'disposition'. The values for these keys are 'fat', 'gray', and 'loud', respectively. You can access these values through their keys:



* + - Dictionaries can still use integer values as keys, but they do not have to start at 0 and can be any number.

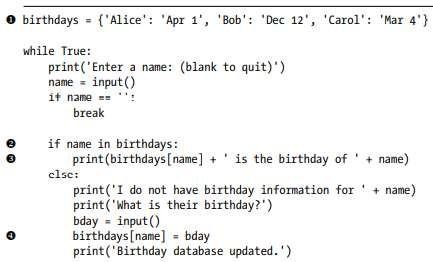
## Dictionaries vs. Lists

* + - Unlike lists, items in dictionaries are unordered.
    - There is no first, second, third etc……in dictionaries.
    - Since, dictionaries are unordered they cannot be sliced.
    - Trying to access a key that does not exist in a dictionary will result in a KeyErrorerror message, much like a list’s “out-of-range” IndexError error message.

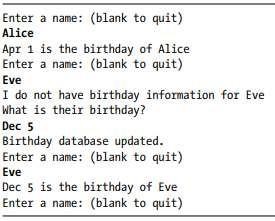


* + - We can have arbitrary values for the keys that allows us to organize our data inpowerful ways.
    - **Ex:** we want to store data about our friends’ birthdays. We can use a dictionary withthe names as keys and the birthdays as values.

***Birthday dictionary program (very imp)***

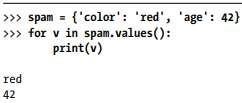


### Output:



* + - We create an initial dictionary and store it in birthdays **1**.
    - We can see if the entered name exists as a key in the dictionary with the in keyword **2**.
    - If the name is in the dictionary, we access the associated value using square brackets **3**; if not, we can add it using the same square bracket syntax combined with the assignment operator **4**.

## The keys(), values(), and items() Methods

* + - There are three dictionary methods that will return list-like values of the dictionary’skeys, values, or both keys and values: keys(), values(), and items().
    - Data types (dict\_keys, dict\_values, and dict\_items, respectively) can be used in forloops
    - A for loop can iterate over the keys, values, or key-value pairs in a dictionary by usingkeys(), values(), and items() methods.
    - The values in the dict\_items value returned by the items() method are tuples of thekey and value.

1. ***keys():***

*Python dictionary keys() function is used to return a view object that contains a list of all the keys in the dictionary. This Python dictionary keys() method is used to retrieve all the keys so we know what kind of information is stored.*

1. ***values():***

*The values() method returns a view object. The view object contains the values of the dictionary, as a list. The view object will reflect any changes done to the dictionary.*

*Python allows the values in a dictionary to be any type – string, integer, a list, another dictionary, boolean, etc. However, keys must always be an immutable data type, such as strings, numbers, or tuples.*

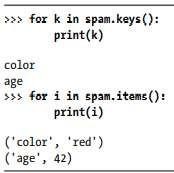
1. ***items()***

*In Python Dictionary, items() method is used to return the list with all dictionary keys with values.*

*Syntax: dictionary . items() Parameters:*

*This method takes no parameters.*

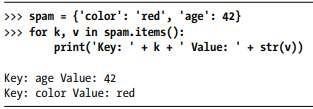
*Returns: A view object that displays a list of a given dictionary's (key, value) tuple pair*



* + If we want a true list from one of these methods, pass its list-like return value to thelist() function.

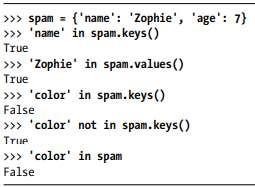


* + The list(spam.keys()) line takes the dict\_keys value returned from keys() and passes it to list(), which then returns a list value of ['color', 'age'].
  + We can also use the multiple assignment trick in a for loop to assign the key and valueto separate variables.



## Checking Whether a Key or Value Exists in a Dictionary

* + We can use the **in** and **not in** operators to see whether a certain key or value exists ina dictionary



***Set()***

*The `set()` function in Python is a built-in function that is used to create a set object. A set is an unordered collection of unique elements, meaning that each element appears only once in the set.*

*Here is the syntax:*

*set(iterable)*

*Here is an example:*

*numbers = [1, 2, 3, 3, 4, 5, 5] unique\_numbers = set(numbers)*

*print(unique\_numbers) # Output: {1, 2, 3, 4, 5}*

*In the above example, the `set()` function is used to create a set object*

*`unique\_numbers` from the `numbers` list. Notice that the duplicate elements are automatically removed, and only the unique elements are included in the set.*

## The get() Method

**Dictionary.get(key,default)**

Dictionaries have a get() method that takes two arguments:

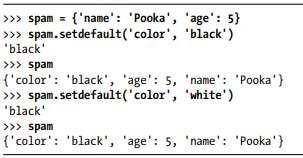
* + The key of the value to retrieve
  + A fallback value to return if that key does not exist.

## The set default function

* + To set a value in a dictionary for a certain key only if that key does not already have a value



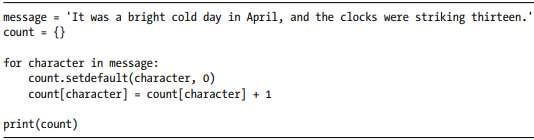
* + The setdefault() method offers a way to do this in one line of code.
  + Setdeafault() takes 2 arguments:
    - The first argument is the key to check for, and
    - The second argument is the value to set at that key if the key does notexist. If the key does exist, the setdefault() method returns the key’s value.



* + The first time setdefault() is called, the dictionary in spam changes to

{'color': 'black', 'age': 5, 'name': 'Pooka'}. The method returns the value 'black' because this is now the value set for the key 'color'. When spam.setdefault('color', 'white') is called next, the value for that key is not changed to 'white' because spam already has a key named 'color'.

**Ex:** program that counts the number of occurrences of each letter in a string.



* + The program loops over each character in the message variable’s string, counting howoften each character appears.
  + The setdefault() method call ensures that the key is in the count dictionary

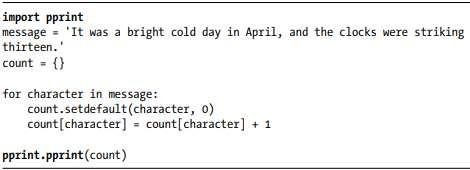
(with a default value of 0), so the program doesn’t throw a KeyError error when count[character] = count[character] + 1 is executed.

## Output:



* 1. **Pretty Printing**
     + Importing pprint module will provide access to the pprint() and pformat() functions that will “pretty print” a dictionary’s values.
     + This is helpful when we want a cleaner display of the items in a dictionary than what print() provides and also it is helpful when the dictionary itself contains nested lists ordictionaries..

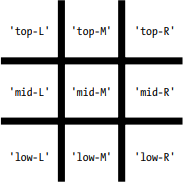
**Program:** counts the number of occurrences of each letter in a string.



**Output:**



### Using Data Structures to Model Real-World Things A Tic-Tac-Toe Board

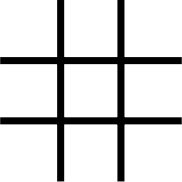
* + - A tic-tac-toe board looks like a large hash symbol (#) with nine slots that can each contain an X, an O, or a blank. To represent the board with a dictionary, we can assign each slot a string-value key as shown in below figure.

**Figure:** The slots of a tic-tactoe board with their corresponding keys

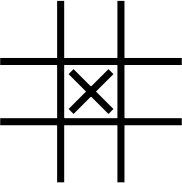
* + - We can use string values to represent what’s in each slot on the board: 'X', 'O', or ' ' (a space character).
    - To store nine strings. We can use a dictionary of values for this.
      * The string value with the key 'top-R' can represent the top-right corner,
      * The string value with the key 'low-L' can represent the bottom-left corner,
      * The string value with the key 'mid-M' can represent the middle, and so on.
    - Store this board-as-a-dictionary in a variable named theBoard.



* + - The data structure stored in the theBoard variable represents the tic-tactoe board in the below Figure.

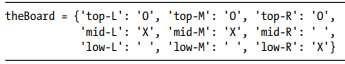


**Figure:** An empty tic-tac-toe board

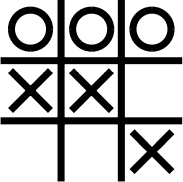
* + - Since the value for every key in theBoard is a single-space string, this dictionary represents a completely clear board. If player X went first and chose the middle space, you could represent that board with this dictionary as shown below:

**Figure:** A first move

* + - A board where player O has won by placing Os across the top might look like this:

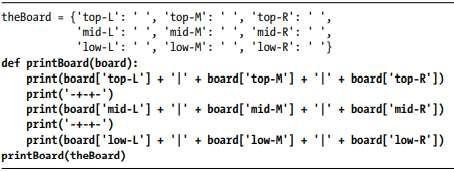


* + - The data structure in theBoard now represents tic-tac-toe board in the below Figure.



**Figure:** Player O wins.

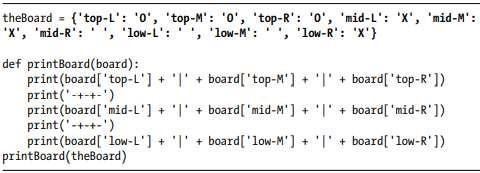
* + - The player sees only what is printed to the screen, not the contents of variables.
    - The tic-tac-toe program is updated as below.



**Output:**



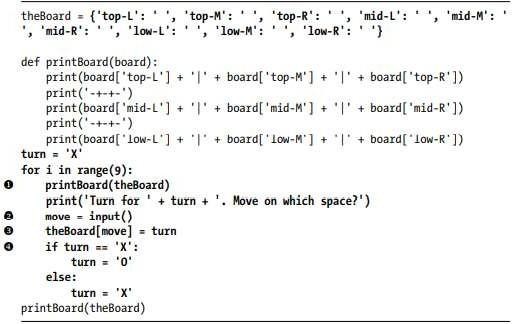
* + - The printBoard() function can handle any tic-tac-toe data structure you pass it.

**Program**

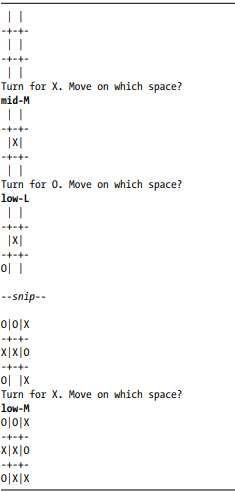
**Output:**



* + - Now we created a data structure to represent a tic-tac-toe board and wrote code in printBoard() to interpret that data structure, we now have a program that “models” the tic-tac-toe board.
    - **Program:** allows the players to enter their moves.



**Output:**



## The methods used in dictionaries

**Ex:{ ‘name’ :’Figo’, ‘model’ : 2017,’make’ : ‘food’}**

## 1.dt.keys():

It will return the values to the keys of dictionaries ‘name’

‘model’

‘make’

# dt.values():

It is a function which is used to return the values to the string.

Figo 2017

Food

# dt.items():

It is a function which returns the key-value pair ‘name’ : Figo

‘model’ : 2017 ‘make’: Food

# dt.pop():

It will remove key value pair from the (item) from the list Ex:dt.pop(model)

Output:{‘name’ :’Figo’, ’make’ : ‘food’ }

# dt.get():

It will return the value of specified index.

Ex:dt.get(model)

Output: Figo

# dt.clear():

It will clear the whole dictionary.

# dt1=dict{}:

It creates an empty dictionary.

# dt.update();

It will add or update the item anywhere in the dictionary.

Ex:dt.update({‘color’ : ‘red’}

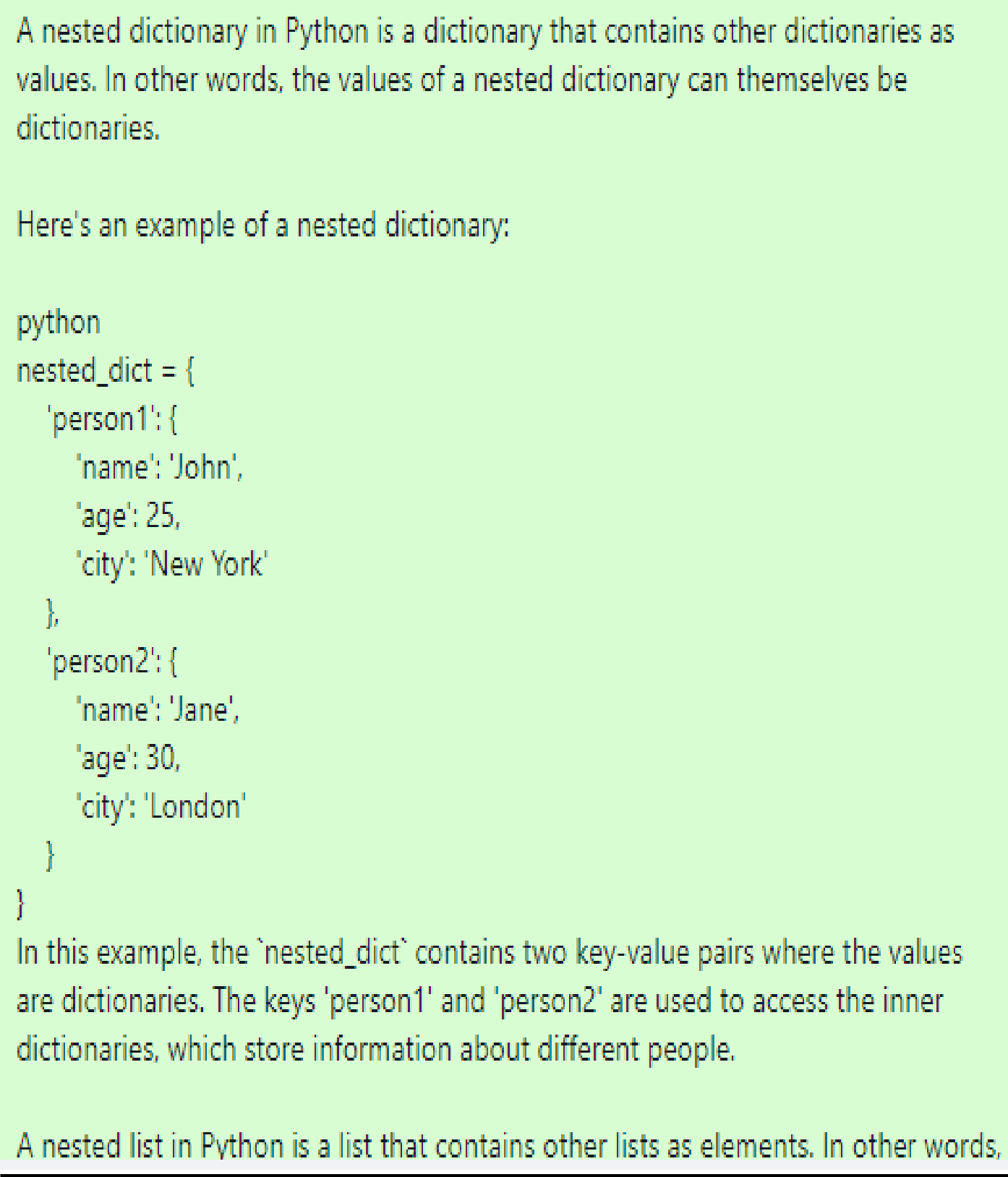
Output:{ ‘name’ :’Figo’, ‘model’ : 2017 , ’make’ : ‘food’, ‘color’ : ‘red’}

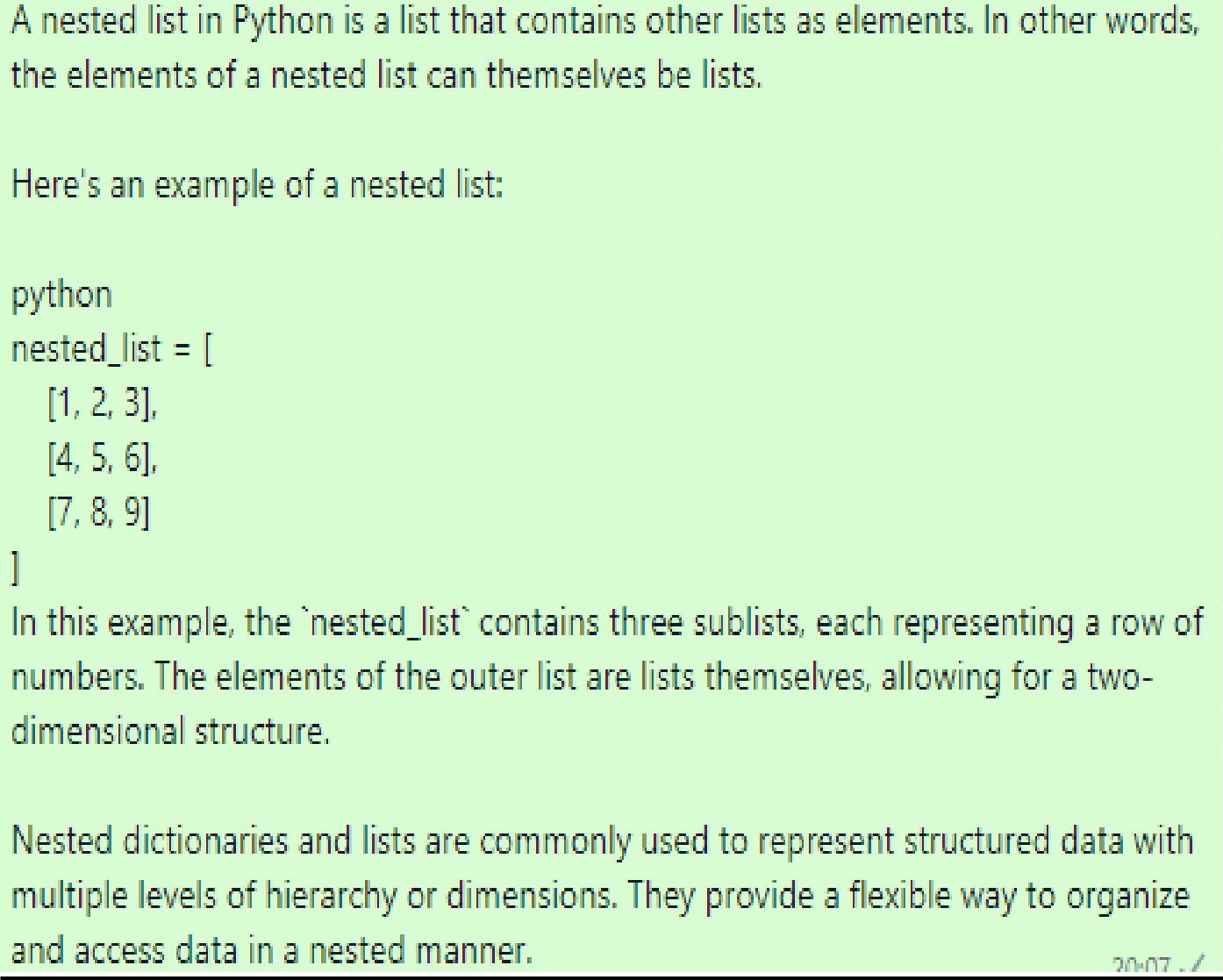
# dt.popitem():

Removes last inserted key value pair Ex:dt.popitem()

Output: { ‘name’ :’Figo’, ‘model’ : 2017 , ’make’ : ‘food’}

‘

**Nested Dictionaries and Lists**

* + **Program:** which contains nested dictionaries in order to see who is bringing what to apicnic

.

*iscuss list and dictionary data structure with example for each.*

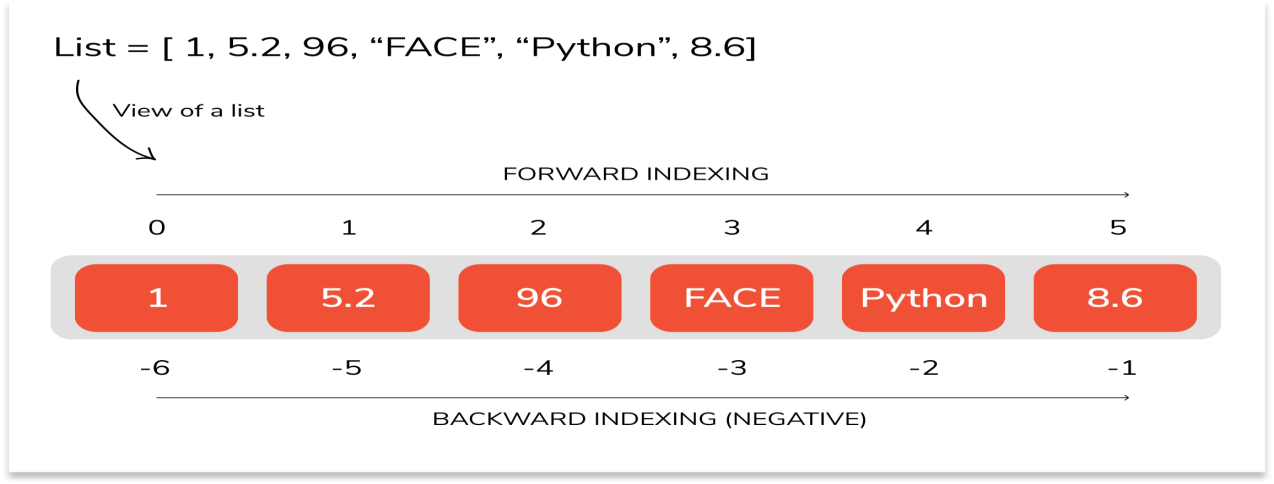
***The List Data Type***

* *A list is an ordered sequence of multiple values.*
* *It is a data structure in Python.*
* *The values inside the lists can be of any type (like integer, float, strings, lists, tuples, dictionaries etc) and are called as elements or items.*
* *The elements of lists are enclosed within square brackets separated*

*by commas.*

*For example, ls1=[10,-4, 25, 13]*

*ls2=[“Tiger”, “Lion”, “Cheetah”]*



* + *Forward Indexing - Starts from index 0 from the beginning of the list moving towards the end.*
  + *Negative Indexing - Starts from index -1 from the end of the list moving towards the beginning.*

*>>> ls=[34, 'hi', [2,3],-5]*

*>>> print(ls[1]) hi*

*>>> print(ls[2]) [2, 3]*

*Traversing a List*

* *A list can be traversed using for loop.*
* *If we need to use each element in the list, we can use the for loop and in operator as below*
* *Example: To Print the elements of a list*

*>>> ls=[34, 'hi', [2,3],-5]*

*>>> for item in ls:*

*print(item) OUTPUT:*

*34*

*hi [2,3]*

*-5*

* *List elements can be accessed with the combination of range() and len() functions as well –*

*Example:*

*ls=[1,2,3,4]*

*for i in range(len(ls)): ls[i]=ls[i]\*\*2 print(ls)*

***The dictionary data type:***

* + *A dictionary is a collection of unordered set of* ***key:value*** *pairs, with the requirement that keys are unique in one dictionary.*
  + *The values in dictionary are accessed using keys.*
  + *A key in dictionary can be any immutable type like strings, numbers and tuples. (The tuple can be made as a key for dictionary, only if that tuple consist of string/number/ sub-tuples).*
  + *Each key maps to a value.*
* *An empty dictionary can be created using two ways –*

*d= {}*

*OR*

*d=dict()*

* *To add items to dictionary, we can use square brackets as –*

*>>> d={}*

*>>> d["Mango"]="Fruit"*

*>>> d["Banana"]="Fruit"*

*>>> d["Cucumber"]="Veg"*

*>>> print(d)*

*{'Mango': 'Fruit', 'Banana': 'Fruit', 'Cucumber': 'Veg'}*

* + *To initialize a dictionary at the time of creation itself, one can use the code like –*

*>>> tel\_dir={'Tom': 3491, 'Jerry':8135}*

*>>> print(tel\_dir)*

*{'Tom': 3491, 'Jerry': 8135}*

*>>> tel\_dir['Donald']=4793*

*>>> print(tel\_dir)*

*{'Tom': 3491, 'Jerry': 8135, 'Donald': 4793}*

*Write a python program to accept n numbers and store them in a list. Then print the list without ODD numbers in it.*

*n = int(input(“Enter the value of N: “)) list1 = [ ]*

*evenList = [ ]*

*print(f“Enter the {n} numbers ”) for i in range( n ) :*

*num = int(input( f“Enter number {i+1} :”)) list1.append(num)*

*for i in range(n) :*

*if list1[ i ] % 2 == 0: evenList.append(list1[ i ])*

*print(“List without ODD numbers: “) print(evenList)*

***Lists***

***Max()***

*The `max()` function in Python is a built-in function that is used to find the largest element in a list or any iterable. It takes an iterable as an argument and returns the maximum element.*

*Here is the syntax:*

*max(iterable, \*[, key, default]) example:*

*numbers = [5, 8, 2, 10, 3] maximum = max(numbers) print(maximum) # Output: 10*

***min()***

*The `min()` function in Python is a built-in function that is used to find the smallest element in a list or any iterable. It takes an iterable as an argument and returns the minimum element.*

*Here is the syntax:*

*min(iterable, \*[, key, default]) Here is an example:*

*numbers = [5, 8, 2, 10, 3]*

*minimum = min(numbers) print(minimum) # Output: 2*

#### Using string slicing operation write python program to reverse each word in a given string (eg: input: “hello how are you”, output: “olleh woh era uoy”)

*inStr = input(“Enter a string: “)*

*slicedStr = inStr.split( ) newStr = ‘’*

*for words in slicedStr : rev = words [ : : -1 ]*

*newStr + = rev + “ ”*

*print(inStr) print(newStr)*