



DATA STRUCTURE

List and Dictionary

LIST

- Creating a new list
- List indexes
- Adding an element
- Removing an element
- Adding a list
- Sorting a list
- List concatenation
- List multiplication

Lists

An array with data separated by commas, denoted with `[]`. Lists are mutable (can be changed).

```
list1 = [1,2,3,4,5]  
list2 = ["a","b","c","d"]  
list3 = ["cat", 45, "dog", 23]
```

Creating a new list

To create a new list, assign the name of your list to an empty list or create your own elements within square brackets.

```
new_list = []  
new_list_2 = [3,6,9,12]
```

LIST INDEXES

Index	0	1	2	3	4	5	6	7	8	
	[1,	2,	3,	4,	5,	6,	7,	8,	9]
	-9	-8	-7	-6	-5	-4	-3	-2	-1	Index

- Every element in a list is denoted by an index.
- indexes start at 0 with the index of each subsequent element increasing by 1.
- Negative indexes refer to elements from the end of the list.

LIST INDEXES

- To refer to a specific index, type the name of the list followed by the index number within square brackets.

```
>>> testlist= [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> testlist[3]
4
>>> testlist[-2]
8
```

- To refer to a range of the indexes, type the name of the list followed by the start and end index number separated by a semicolon within square brackets.

```
>>> testlist[5:]
[6, 7, 8, 9]
```

LIST INDEXES

Index	0	1	2	3	4	
	1	2	3	4	5	
	-5	-4	-3	-2	-1	Index

```
>>> list1 = [1,2,3,4,5]
```

```
>>> print(list1[0])
```

```
1
```

```
>>> print(list1[3])
```

```
4
```

```
>>> print(list1[-1])
```

```
5
```

LIST INDEXES

Index	0	1	2	3	4	
	1	2	3	4	5	
	-5	-4	-3	-2	-1	Index

```
>>> print(list1[2:4])
```

```
[3,4]
```

```
>>> print(list1[2:])
```

```
[3,4,5]
```

```
>>> print(list1[:2])
```

```
[1,2]
```

```
>>> print(list1[:])
```

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


LENGTH OF A LIST

To get the length of a list, use the function **len(list)** to count the number of elements in a list.

```
>>> list1 = [3,6,9,12]
>>> len(list1)
4
```

APPENDING TO A LIST

To append to a list or add an element to the end of the list, use the function **list.append(element)** placing the element you would like to add within the brackets

```
>>> list1 = [1,2,3,4,5]
>>> list1.append(6)
>>> list1
[1,2,3,4,5,6]
```

REMOVING ELEMENTS

To remove an element from a list, use the function **list.remove(element)** placing the element you would like to remove within the brackets.

```
>>> list1 = [1,2,3,4,5]
>>> list1.remove(3)
>>> print(list1)
[1,2,4,5]
```

You can also use the function **del list[index]** placing the index of the element you would like to remove within the brackets.

```
>>> list1 = [1,2,3,4,5]
>>> del list1[2]
>>> list1
[1,2,4,5]
```

REMOVING ELEMENTS

To extract an element from a list, use the function `list.pop(index)` placing the index of the element you would like to extract within the brackets. This will give you the element that was “popped out” of the list.

```
>>> list1 = [1,2,3,4,5]
>>> extract_element = list1.pop(3)
>>> print(extract_element)
4

>>> print(list1)
[1,2,3,5]
```

ADDING LISTS

To add another list to your current list, use the function `list.extend(list2)` placing the name of the list you would like to join within the brackets.

```
>>> list1 = [1,2,3,4,5]
>>> list2 = [6,7,8]
>>> list1.extend(list2)
>>> print (list1)
[1,2,3,4,5,6,7,8]
```

This changes the original list as compared to creating a new list.

LIST CONCATENATION

Similar to strings, lists can also be concatenated (joined). They will be in the order of the lists you input

```
>>> list1 = [1,2,3]
>>> list2 = [4,5,6]
>>> list3 = list1 + list2
>>> print(list3)
[1,2,3,4,5,6]
```

This allows you to create a new list. This is different from `list.extend()` which changes the list being joined to.

INSERTING LIST ELEMENTS

```
>>> list1 = [1, 2, 3, 4, 5]
```

```
>>> list1.insert(3, 54)
```

index

value

```
>>> print(list1)
```

```
[1, 2, 3, 54, 4, 5]
```

REVERSING LISTS

You can reverse the order of a list using *list.reverse*, as seen below

```
>>> list1 = [14, 2, 3, 34, 55]
>>> list1.reverse()
[55, 34, 3, 2, 14]
```


NESTED LISTS

It is possible to create a list of lists and manipulate it too

```
>>> list1 = [ [1, 2],[3, 4],[5, 6]]  
>>> list1[1]  
[3,4]  
>>> list1[1][0]  
3
```

CHALLENGE: Using list1, print [5, 6]. Then try printing 5 from list1.

EXERCISE

Code out the appropriate list operations to find the desired output.

```
>>> ls00 = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]  
>>> ls01 = [ [1, 2, 3], [4, 5, 6], [7, 8, 9], [13, 14, 15] ]
```

1. Use `len` to calculate the length of the list with the most elements.
2. Append the number 16 in `ls00` at the last position.
3. Reverse the order of `ls01`.
4. Now insert the missing list of numbers in `ls01`.
5. Append `ls02 = [16, 17, 18]` to `ls01`.
6. Find the sum of the last 3 odd numbers in `ls00` and `ls01` respectively, by calling their indices.

EXERCISE

1. What is the length of the list with the most elements?

```
>>> len(ls00)  
15
```

- 2.

```
>>> ls00.append(16)
```

OR

```
>>> ls00.insert(15,16)
```

- 3.

```
>>> ls01.reverse()
```

4. Insert the missing list for ls01.

```
>>> ls01.insert(3, [10, 11, 12])  
>>> ls01  
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]]
```

EXERCISE

5. Append ls02 = [16, 17, 18] to ls01.

```
>>> ls02=[16,17,18]
>>> ls01.append(ls02)
>>> ls01
[ [1, 2, 3], [4, 5, 6], [7, 8, 9], [13, 14, 15],[16, 17, 18] ]
```

6. Find the sum of the odd numbers in ls00 and ls02 separately, by calling their indexes.

```
>>> odd_sum00 = ls00[-2] + ls00[-4] + ls00[-6]
>>> odd_sum00
39
```

```
>>> odd_sum01 = ls01[-1][-2] + ls01[-2][0] + ls01[-2][2]
>>> odd_sum01
45
```

SORTING A LIST

To sort a list, use the function **list.sort()** or **sorted()** to arrange the elements in ascending order. They will be arranged in increasing order if the elements are numbers and will be arranged alphabetically if the elements are strings.

```
>>> list1 = [5,3,4,1,2]
>>> list1.sort()
>>> print(list1)
[1,2,3,4,5]
```

```
>>> list2 = ["banana", "apple", "durian", "carrot"]
>>> list2.sort()
>>> list2
["apple", "banana", "carrot", "durian"]
```

SORTING A LIST

To sort a list, use the function **list.sort()** or **sorted()** to arrange the elements in ascending order. They will be arranged in increasing order if the elements are numbers and will be arranged alphabetically if the elements are strings.

```
>>> list1 = [5,3,4,1,2]
>>> sorted(list1)
[1,2,3,4,5]
```

```
>>> list2 = ["banana", "apple", "durian", "carrot"]
>>> sorted(list2)
["apple", "banana", "carrot", "durian"]
```

What if the list has both strings and number elements?

```
>>> list3 = ['apple', 'banana', 'carrot', 'durian', 5, 3, 4, 1, 2]
>>> list3.sort()
TypeError: '<' not supported between instances of 'int' and 'str'
```

Sorting nested lists

We can also sort the list according to the second value in each nested list by changing the **key**, which determines how the list is sorted. We will use a function to look at the second value in the list.

```
>>> list1 = [[1,7],[9,4],[3,5]]
>>> print(sorted(list1))
[[1,7],[3,5],[9,4]]

>>> def getKey(item):
    return item[1]

>>> list1.sort(key = getKey)
>>> print(list1)
[[9,4],[3,5],[1,7]]

>>> sorted(list1, key = getKey)
[[9,4],[3,5],[1,7]]
```


REVERSE-SORT LISTS

Using the `list.sort()` or the `sorted()` method on a nested list will sort the list according to the first value in each nested list. We can also reverse-sort the list by making function parameter `reverse` equal to `True`.

```
>>> list1 = [1, 2, 3, 4, 5]
>>> list1.sort(reverse = True)
>>> print(list1)
[5,4,3,2,1]
>>> sorted(list1, reverse = True)
[5,4,3,2,1]
```

DIFFERENCE BETWEEN LIST.SORT() AND SORTED()

- list.sort() can only be used for lists while sorted() can be used on any iterable data types (strings, tuples, dictionaries)

```
>>> list1 = [5, 3, 4, 1, 2]
>>> string1 = "lazyrabbit"
```

```
>>> list1.sort()
[1, 2, 3, 4, 5]
>>> string1.sort()
AttributeError: 'str' object has no attribute 'sort'
```

```
>>> sorted(list1)
[1, 2, 3, 4, 5]
>>> sorted(string1)
['a', 'a', 'b', 'b', 'i', 'l', 'r', 't', 'y', 'z']
```

DIFFERENCE BETWEEN LIST.SORT() AND SORTED()

- list.sort() changes the original list while sorted() retains the original list and creates a new sorted list

```
>>> list1 = [5, 3, 4, 1, 2]
```

```
>>> list2 = [9, 4, 6, 0, 1]
```

```
>>> list1.sort()
```

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

```
>>> list1
```

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

```
>>> sorted(list2)
```

```
[0, 1, 4, 6, 9]
```

```
>>> list2
```

```
[9, 4, 6, 0, 1]
```

LIST MULTIPLICATION

Similar to strings, lists can also be multiplied.

```
>>> list1 = [1,2,3]
>>> print (list1 * 3)
[1,2,3,1,2,3,1,2,3]
```

EXERCISE

Your colleague and yourself are summarising test results for a certification practice test. In the lists below, the first number is the examinee name and the number is their score.

```
mylist = [[ "Mingqi T.", 9], ["Wen Qi W.", 6] , ["Jenn A.", 8]]  
herlist = [[ "Siti H.",7], [ "Abinayaa", 5]]
```

1. Use indexes to find the average mark for both teams.
2. Combine both lists under the variable “teamlist”.
3. Your colleague just updated you with her missing team member’s score. Add on to “teamlist” this information: [“Ahmed B.”, 8]
4. Compute the average mark of all the team members.
5. Sort teamlist in reverse alphabetical order.

EXERCISE

```
mylist = [[ "Mingqi T.", 9], ["Wen Qi W.", 6] , ["Jenn A.", 8]]  
herlist = [[ "Siti H.",7], [ "Abinayaa", 5]]
```

1. Use indexes to find the average mark for both teams.

```
my_ave = (mylist[0][1] + mylist[1][1] + mylist[2][1]) / len(mylist)  
her_ave = (herlist[0][1] + herlist[1][1]) / len(herlist)
```

2. Combine both lists under the variable “teamlist”.

```
teamlist = mylist + herlist
```

3. Your colleague just updated you with her missing team member's score. Add on to “teamlist” this information: [“Ahmed B.”, 8]

```
teamlist.append(["Ahmed B.", 8])
```

EXERCISE

4. Compute the average mark of all the team members.

```
team_ave =(teamlist[0][1] + teamlist[1][1] + teamlist[2][1] + teamlist[3][1]  
+ teamlist[4][1] + teamlist[5][1]) / len(teamlist)
```

5. Sort teamlist in reverse alphabetical order

```
teamlist.sort(reverse = True)
```

QUICK CHECK!

Code out the instructions given to get the desired outputs using the following lists.

```
mylist = [ [ 1, 2] , [3, 4] , [8, 0]]
```

code	output
Insert [5,6] between [3,4] and [8,0]	
Sort mylist by the second values of each sublist, in descending order.	
Count the number of sublists in mylist.	
Print the last sublist.	
Insert [5,6] between [3,4] and [8,0]. Then put 9 at the end of the new list. Print the new list.	

QUICK CHECK!

Code out the instructions given to get the desired outputs using the following lists.

```
mylist = [ [ 1, 2] , [3, 4] , [8, 0]]
```

code	output
Insert [5,6] between [3,4] and [8,0]	[[1, 2],[3, 4],[5, 6],[8, 0]]
Sort mylist by the second values of each sublist, in descending order.	[[3, 4], [1, 2],[8, 0]]
Count the number of sublists in mylist.	3
Print the last sublist.	[8, 0]
Insert [5,6] between [3,4] and [8,0]. Then put 9 at the end of the new list. Print the new list.	[[1, 2],[3, 4],[5, 6],[8, 0], 9]

DICTIONARY

- Creating a new dictionary
- Dictionary Indexes
- Adding key-value pairs

DICTIONARIES

An array with data (each containing a key paired with a value, known as a key-value pair) separated by a comma, denoted with { }.

A colon (:) is used to assign the **value** to the **key**.

```
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

CREATING A NEW DICTIONARY

To create a new dictionary, assign a variable to an empty dictionary, or create your own **key-value** pairs within curly brackets.

```
dictionary0 = {}  
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

DICTIONARY KEYS AND VALUES

To access a specific value, type the name of the dictionary followed by the key associated with the value in a square bracket.

```
squares = {1:1, 2: 4, 3:9, 4:16, 5:25}
>>> print(squares[1])           # ⇒ 1
>>> print(squares[4])           # ⇒ 16
```

To see what keys and values are in the dictionary, type `.keys()` or `.values()` after the dictionary name.

```
>>> squares.keys()
dict_keys([1, 2, 3, 4, 5])
>>> squares.values()
dict_values([1, 4, 9, 16, 25])
```

ADDING KEY-VALUE PAIRS

To add a key-value pair, assign a **value** to a **key** for the dictionary.

```
>>> squares[6] = 36
>>> print(squares)
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

CHANGING KEYS AND VALUES

You can change dictionary values.

```
squares = {1:1, 2: 4, 3:9, 4:16, 5:25}
```

```
>>> squares[3] = "wrong square"  
>>> print(squares)  
{1:1, 2: 4, 3:"wrong square", 4:16, 5:25}
```

REMOVING KEYS AND VALUES

You can also remove dictionary keys and values.

```
squares = {1:1, 2: 4, 3:9, 4:16, 5:25}
```

```
#Pop out a certain value from dictionary using key
print(squares.pop(4))          #==> 16
print(squares)                  #==> {1: 1, 2: 4, 3: 9, 5: 25}
```

```
#Delete a key-value pair or entire dictionary with del
del squares[2]
print(squares)                  #==> {1: 1, 3: 9, 5: 25}
```

```
del squares
print(squares)                  #==> NameError: name 'squares'
                                is not defined
```


LENGTH OF A DICTIONARY

To get the length of a dictionary, use the function `len(dict1)` to count the number of key-value pairs in the dictionary.

```
>>> dict1 = {1: "A", 2: "B", 3: "C"}  
>>> print(len(dict1))  
3
```

EXERCISE I

Try creating your own dictionary using your personal details, like age and name.

EXERCISE I

Try creating your own dictionary using your personal details, like age, name and so on.

```
mydict = {"name": "Li Mingyao", "age": 24, "nationality":  
"Singaporean"}
```

Now try to print one of the values by calling the dictionary key!

EXERCISE II

2018 Company Profits	
Quarter	Profits
Q1 (Jan - Mar)	540390
Q2 (Apr - Jun)	320980
Q3 (Jul - Sep)	550021
Q4 (Oct- Dec)	180900

1. Create a dictionary called profits with the information above. The key will be the quarter (Q1, Q2...) and the value will be the profit for that quarter.
2. Find the monthly average for each quarter, and for the year. Which quarter(s) are below average?
3. In November, an employee was arrested for embezzling \$350,000 from the month's profits. This money should be added back into profits. Update profits with the correct quarterly profits.
4. Create a new dictionary called fin_summ with the new monthly averages for each quarter, to the nearest dollar. Use `int(number)` or `round(number)` to round your numbers.

EXERCISE II

1. Create a dictionary called profits with the information above. The key will be the quarter (Q1, Q2...) and the value will be the profit for that quarter.

```
profits = {"Q1":540349, "Q2": 320980, "Q3": 550021, "Q4":  
180900}
```

EXERCISE II

- Find the monthly average for each quarter, and for the year. Which quarter(s) are below average for the year? Round down the average to the nearest whole

```
q1_ave = int(profits["Q1"] / 3)
q2_ave = int(profits["Q2"] / 3)
q3_ave = int(profits["Q3"] / 3)
q4_ave = int(profits["Q4"] / 3)
y_ave = int((profits["Q1"] + profits["Q2"] + profits["Q3"] +
profits["Q4"])/12)

print(q1_ave, q2_ave, q3_ave, q4_ave, y_ave)
```

Q2 and Q4 are below the annual average.

#Shortcut to finding y_ave

```
y_ave = int(sum(profits.values()) / 12)
```

EXERCISE II

3. In November, an employee was arrested for embezzling \$350,000 from the month's profits. Update profits with the correct quarterly values.

```
profits["Q4"] = profits["Q4"] + 350000
```

OR (same as above)

```
profits["Q4"] += 350000
```

EXERCISE II

4. Create a dictionary called `fin_summ` with the new monthly averages for each quarter, to the nearest dollar. You can use the same keys as in `profits`.

```
fin_summ = {"Q1": q1_ave, "Q2": q2_ave, "Q3": q3_ave, "Q4":  
int(profits["Q4"] / 3)}  
  
print(fin_summ)
```


EXERCISE III

Given a nested list, use a nested loop to print all the items in the list.

```
list1 =  
[['Matt', 'Ben', 'Jerry'], [1, 2, 3, 4, 5], ['Denmark', 'England', 'Scotland']]
```

EXERCISE III

Given a nested list, use a nested loop to print all the items in the list.

```
list1 =  
[['Matt','Ben','Jerry'],[1,2,3,4,5],['Denmark','England','Scotland']]  
  
for i in list1:           #i refers to sublist  
    for x in i:           #x refers to elements in each sublist  
        print(x)
```

EXERCISE IV

Create a list called `dept_promoted` of all the people from `dept_list` that were promoted.

```
dept_list = ["Amy", "Simon", "Vaishak", "Adam", "Siti"]  
promoted = ["Simon", "Vaishak", "Siti", "MingYao", "Khairul"]
```

EXERCISE IV

Create a list called `dept_promoted` of all the people from `dept_list` that were promoted.

```
dept_list = ["Amy", "Simon", "Vaishak", "Adam", "Siti"]
promoted = ["Simon", "Vaishak", "Siti", "MingYao", "Khairul"]
dept_promoted = []

for employee in dept_list:
    for name in promoted:
        if name == employee:
            dept_promoted.append(name)
```

BONUS CHALLENGES

For those of you who want
something extra

BONUS CHALLENGE 1

Create a dictionary called **marks** with keys **Math**, **English**, **Science** with a default value of **0**

```
>>> print(marks)
{'Math':0, 'English':0, 'Science':0}
```

BONUS CHALLENGE 1

Create a dictionary called **marks** with keys **Math**, **English**, **Science** with a default value of **0**

```
>>> marks={'Math':0,'English':0,'Science':0}
```

OR

```
>>> marks = {}.fromkeys(['Math','English','Science'], 0)
```

BONUS CHALLENGE 1

1. Add in a new subject History with a score of 90
2. Add in a new subject Geography with a score 60
3. Update the English score to be 70
4. Remove the last item of the dictionary
5. Remove Science from the dictionary and assign its value to a variable called **science_marks**

```
>>> marks  
{ 'Math':0, 'English':70, 'History':90 }  
>>> science_marks  
0
```


BONUS CHALLENGE 1

1. Add in a new subject History with a score of 90
2. Add in a new subject Geography with a score 60
3. Update the English score to be 70
4. Remove the last item of the dictionary
5. Remove Science from the dictionary and assign its value to a variable called **science_marks**

```
>>> marks['History'] = 90
>>> marks['Geography'] = 60
>>> marks['English'] = 70
>>> marks.popitem() OR del marks['Geography']
>>> science_marks = marks.pop('Science')
```

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

```
>>> for i in marks:  
    print(i)
```

Math
English
History

```
>>> for i in marks.keys():  
    print(i)
```

Math
English
History

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

```
>>> for i in marks.values():  
    print(i)
```

0

70

90

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

```
>>> for i in marks.items():  
    print(i)  
  
( 'Math', 0)  
( 'English', 70)  
( 'History', 90)
```

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

```
>>> subjects = sorted(marks)
>>> subjects
['English', 'Math', 'Science']
```

```
>>> subjects = sorted(marks.keys())
>>> subjects
['English', 'Math', 'Science']
```

BONUS CHALLENGE 1

6. Use a for loop to print the dictionary keys
7. Use a for loop to print the dictionary values
8. Use a for loop to print the key value pairs
9. Create a sorted list of the dictionary keys titled **subjects**
10. Create a sorted list of the dictionary values titled **scores**

```
>>> scores=sorted(marks.values())  
>>> scores  
[0, 70, 90]
```

BONUS CHALLENGE 2

Create a 2-dimensional array called **2D_array**, use a nested for loop to print out each number in the array on a separate line

```
>>> 2D_array = [[1, 2, 3, 4], [5, 6], [7, 8, 9],[10, 11, 12]]
```

1
2
3
4
5
6
7
8
9

BONUS CHALLENGE 2

Create a 2-dimensional array called **2D_array**, use a nested for loop to print out each number in the array on a separate line

```
>>> 2D_array = [[1, 2, 3, 4], [5, 6], [7, 8, 9],[10, 11, 12]]
>>> for i in range(len(2D_array)):
    for j in range(len(2D_array[i])):
        print(2D_array[i][j])
```

1

2

3

4

5

6

7

8

9