Lab 8: List - linear structure

Data structure is a way we organise data in memory. List is one of many data structures exist in programming. List is a linear structure, meaning the elements have a linear ordering. That is, there is a first element, second element, and so on.

List is one of sequence types in python. Others sequence types include tuple, dictionary. A string can also been seen as a sequence of characters.

In contrast to sequence type structures, data structures like Binary Search Tree (BST) organise data in a non-linear way. Non-linear has faster searching than linear.

Zero-based indexing

Elements in a list are identified by index. The first element has the index of 0, the second has 1, and so on. Always keep in mind that the indices start from 0, not 1.

List creation and element retrieval

To create a list, use square brackets [].

([] vs. list(): [] is four time faster: from timeit import timeit, timeit('[]') vs timeit('list()'))

To retrieve a particular element from a list, we need two things:

1. the name of the list and
2. the index of the element.

```
>>> mylist = [10, 20, 30]
>>> mylist [0]
10
>>> mylist [1]
20
>>> mylist [2]
30
>>> mylist = ['Hello','World']
>>> mylist [0]
'Hello'
>>> mylist [1]
'World'
>>> mylist [2]
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    mylist [2]
IndexError: list index out of range
>>>
```

Think of a list like a row of cubby shelf, where the elements are stored in the cubes of the shelf. In order to retrieve the item stored in a cube, you first need to know

a. where the shelf is - the name of the list;
b. then you need to know which cube - the index.

In the above example, *mylist* is the name of the list and *0* or *1* is the index. Note that, when we defined mylist the second time, it had only 2 elements (namely 0 and 1), if we try to retrieve element 3 of the list it will give you an error.

Calculation on list elements

Apart from using a different notation to represent a list element, they are just like ordinary variables.

```
>>> mylist = [10, 20, 30]
>>> mylist [0]
10
>>> mylist [1]
20
>>> mylist[0] + mylist[1]
30
>>> newTotal = mylist[0] +mylist[1]
>>> print(newTotal)
30
>>> mylist[0] = mylist[1]+mylist[2]
>>> print(mylist[0])
50
>>> mylist
[50, 20, 30]
>>>
```

The built-in **len()** function can be used to get the number of items in a list.

```
>>> mylist
[50, 20, 30]
>>> len(mylist)
3
>>> (mylist[0] + mylist[1] + mylist[2]) / len(mylist)
33.333333333333336
```

Traversing a list using FOR loop

The most common list operation is list traversal. Traversing a list is different to manually accessing elements, in that it iterates through the list one-by-one, normally from the beginning to the end and generally, using the **for** loop statement.

The syntax of a for loop is as follows:

```
for <loopvariable> in <sequence>:        # must always end with a colon (:)
    <statements to process>              # can have one or many statements
```

```
>>> num = [10, 20, 30]
>>> for k in num:
        print(k)


10
20
30
>>>
```

Task 1

Write a script called Lab8Task1.py. Given a list num = [20, 30, 40, 50, 60], write a for loop to sum up the elements and to display the total (the total should be 200).

```
num = [20,30,40,50,60]

total = 0
for k in num:
    total = total + k

print(total)
```

Modification 1: Extend the program to calculate the sum of each number doubled (this total should be 400). Write the code below. Then try it out in the script.

Modification 2: How do you calculate the average? Modify the program to include the average (the average should be 40). Write the code below. Then try it out in the script.

Final sample answer:

```
num = [20,30,40,50,60]

total = 0
double = 0
for k in num:
    total = total + k              # add the value of k
    double = double + 2*k          # double k before adding
    average = total /len(num)      # calculate the average

print('The total is: ', total)
print('The double is: ', double)
print('The average is: ', average)
```

Sample output

```
>>>
========== RESTART:
The total is:  200
The double is:  400
The average is:  40.0
>>>
```

Nested lists

Here is a simple example of nested list. Let us rewrite the list example as the following:

In the previous example, num = [[20], [30], [40], [50], [60]]

To calculate the sum of all numbers, code needs to be changed as follows:

```
num = [ [20], [30], [40], [50], [60] ]

total = 0
for k in num:
    total = total + k[0]          # k is no longer a number but a 1-list element
    print('Processing', k[0], 'sub-total so far is', total)

print('\nThe total is: ', total)
```

Sample output:

```
Processing 20 sub-total so far is 20
Processing 30 sub-total so far is 50
Processing 40 sub-total so far is 90
Processing 50 sub-total so far is 140
Processing 60 sub-total so far is 200

The total is:  200
>>>
```

We can also use a list to store a student record, as follows

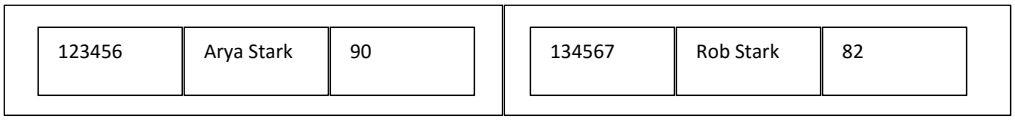
```
student1 = ['123456', 'Arya Stark', 90]
student2 = ['134567', 'Rob stark', 82]
```

To organise all student records together, we can place all the student record lists in another list (i.e. a list inside a list). This is known as a **nested list**.

The following calculates a nested list of student records:

```
records = []

records.append(['123456', 'Arya Stark', 90])
records.append(['134567', 'Rob stark', 82])
```



Try this:

```
>>> records = []
>>> records.append(['123456','Arya Stark', 90])
>>> records.append(['134567','Rob Stark', 82])
>>>
>>> records[0][0]    # will show 123456
'123456'
>>> records[0][1]   # will show Arya Stark
'Arya Stark'
>>> records[0][2]   # will show Arya 90
90
>>> records[1][0]    # will show 134567
'134567'
>>> records[1][1]    # will show Rob Stark
'Rob Stark'
>>> records[1][2]   # will show Arya 82
82
>>>
>>> len(records)    # will show 2 as there are 2 student sets
2
>>> len(records[0])    # will show 3 as a student set has 3 elements
3
>>>
```

To access the records, try the following

```
records = []

records.append(['123456', 'Arya Stark', 90])
records.append(['134567', 'Rob Stark', 82])

# the following will manually access the first record
rec = records[0]
print(rec[0])        # will print 123456
print(rec[1])        # will print Arya Stark
print(rec[2])        # will print 90

# the following will manually access the first record
rec = records[1]
print()
print(rec[0])        # will print 134567
print(rec[1])        # will print Rob Stark
print(rec[2])        # will print 82

# the following will access the records of all students using a for loop
for rec in records:
    print()
    print('Student Id:' , rec[0])
    print('Student Name:' , rec[1])
    print('Student Grade:' , rec[2])
```

A sample output is as shown below:

```
123456
Arya Stark
90

134567
Rob Stark
82

Student Id: 123456
Student Name: Arya Stark
Student Grade: 90

Student Id: 134567
Student Name: Rob Stark
Student Grade: 82
>>>
```

Analysis: The loop runs two times.

In the first iteration of the loop, the **rec** variable is set to have the first student record. Therefore, rec[0], rec[1], rec[2] represent the **first** student ID, name, and grade respectively.

In the second iteration of the loop, variable rec is assigned with the second student record, and therefore, rec[0], rec[1], rec[2] are used to access the **second** student ID, name, and grade.

Task 3: Process a list of student's records, using nested list. Use date.

```
student_mark = [['Brad Pitt', 80],['Johnny Depp', 87], ['Tom Cruise', 82]]

print('These are my students')
for k in student_mark:
    print(k[0])

print('\nThese are their marks:')
for k in student_mark:
    print(k[1])
```

Sample Output:

```
These are my students
Brad Pitt
Johnny Depp
Tom Cruise

These are their marks:
80
87
82
>>>
```

Task 4: List with calculations

```
"
Program that will check a list and show statistics
Max mark, Min mark, Total, Average, Number passed, Number failed
"

student_mark = [['Darius',50], ['Luke', 40], ['James', 60], ['Haley', 70], ['Liam', 30]]

# initialise variables
sum = 0
passcount = 0
failcount = 0
passmark = 45
max_mark = 0
min_mark = 100

for k in student_mark:
    sum = sum + k[1]

    # to check and calculate pass count and fail count
    if k[1] >= passmark:
        passcount = passcount + 1
        result = 'pass'
    else:
        failcount = failcount + 1
        result = 'fail'

    print(k[0], 'scored', k[1], '...', result)
```

```
    if max_mark < k[1]:
        max_mark = k[1]
    if min_mark > k[1]:
        min_mark = k[1]

#notice that the average is calculated outside the loop
average = sum / len(student_mark)

print('\nClass total is:', sum)
print('Class average is:', average)
print('Highest mark is', max_mark)
print('Lowest mark is', min_mark)
print('Number of students that passed:', passcount)
print('Number of students that failed:' , failcount)
```

Sample Output:

```
Darius scored 50 ... pass
Luke scored 40 ... fail
James scored 60 ... pass
Haley scored 70 ... pass
Liam scored 30 ... fail

Class total is: 250
Class average is: 50.0
Highest mark is 70
Lowest mark is 30
Number of students that passed: 3
Number of students that failed: 2
>>>
```

Task 5: Process a list with date.

```
from datetime import datetime

date1 = datetime.strptime('1-10-77', '%d-%m-%y')
date2 = datetime.strptime('17-3-60', '%d-%m-%y')
date3 = datetime.strptime('3-4-86', '%d-%m-%y')

manager=[]
manager.append(['Jaeden', date1, 500])
manager.append(['Ethan', date2, 205])
manager.append(['Shiv', date3, 620])

for k in manager:
    date = datetime.strftime(k[1], '%d %b %Y')

    print(k[0], 'was born on', date, 'and earns', k[2])
```

Sample Output:

```
Jaeden was born on 01 Oct 1977 and earns 500
Ethan was born on 17 Mar 2060 and earns 205
Shiv was born on 03 Apr 1986 and earns 620
>>>
```

Task 5: Format the output.

```
from datetime import datetime

date1 = datetime.strptime('1-10-77', '%d-%m-%y')
date2 = datetime.strptime('17-3-60', '%d-%m-%y')
date3 = datetime.strptime('3-4-86', '%d-%m-%y')

manager=[]
manager.append(['Jaeden', date1, 500])
manager.append(['Ethan', date2, 205])
manager.append(['Shiv', date3, 620])

format_temp = "{0:<15}{1:<15}{2:<15}"   # will format the 3 fields to 15 character length

for k in manager:
   date = datetime.strftime(k[1], '%d %b %Y')

   print(format_temp.format(k[0], date, k[2]))
```

Sample Output:

```
Jaeden         01 Oct 1977    500
Ethan          17 Mar 2060    205
Shiv           03 Apr 1986    620
>>>
```