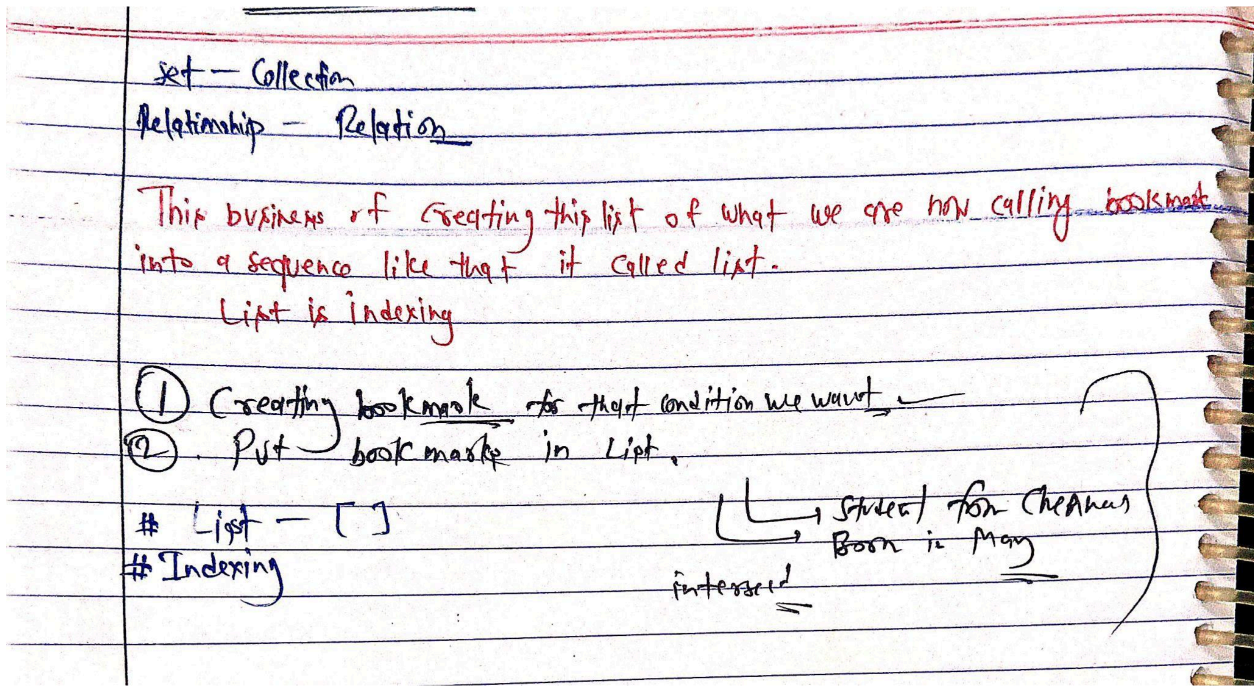
Week 5 : List, Insertion sort

Lecture 1 : Introduction to collections and list data structure



set — Collection
Relationship — Relation

This business of creating this list of what we are now calling bookmark into a sequence like that it called list.
List is indexing

(1) Creating bookmark for that condition we want
(2). Put bookmark in List.

List — [ ]
Indexing

→ Student from Chennai
→ Born in May
interested

Lecture 2 : Pseudocode for lists

Pseudocode: Introducing lists

Collections

- Variables keep track of intermediate values
- Often we need to keep track of a collection of values
  - Students with highest marks in Physics
  - Customers who have bought food items from SV Stores
  - Nouns that follow an adjective
- Simplest collection is a list
  - Sequence of values
  - Single variable refers to the entire sequence
  - Notation for lists
  - Primitive operations to manipulate lists

Pseudocode for lists

- Sequence within square brackets
  - `[1,13,2]`
  - `["Vedanayagam","cane", "Monday","school"]`
  - `[]` — empty list
- Append two lists, `l1 ++ l2`
  - `l1` is `[1,13]`, `l2` is `[2,17,1]`
  - `l1++l2` is `[1,13,2,17,1]`
- Extend `l` with item `x`
  - `l = l ++ [x]`
- Examples
  - List of students born in May

```
mayList = []
while (Table 1 has more rows) {
   Read the first row X in Table 1
   if (X.MonthOfBirth == "May") {
     mayList = mayList ++
                       [X.Seqno]
   }
   Move X to Table 2
}
```

- Sequence within square brackets
  - `[1,13,2]`
  - `["Vedanayagam","cane", "Monday","school"]`
  - `[]` — empty list
- Append two lists, `l1 ++ l2`
  - `l1` is `[1,13]`, `l2` is `[2,17,1]`
  - `l1++l2` is `[1,13,2,17,1]`
- Extend `l` with item `x`
  - `l = l ++ [x]`
- Examples
  - List of students born in May
  - List of students from Chennai

```
chennaiList = []
while (Table 1 has more rows) {
   Read the first row X in Table 1
   if (X.TownCity == "Chennai") {
     chennaiList = chennaiList ++
                      [X.Seqno]
   }
   Move X to Table 2
}
```

Processing lists

- Typically, we need to iterate over a list
  - Examine each item
  - Process it appropriately
- `foreach x in l {`
  Do something with `x`
  `}`
  - `x` iterates through values in `l`
- Example
  - All students born in May who are from Chennai
  - Nested `foreach`

```
mayChennaiList = []
foreach x in mayList {
   foreach y in chennaiList {
     if (x == y) {
       mayChennaiList =
              mayChennaiList ++ [x]
     }
   }
}
```

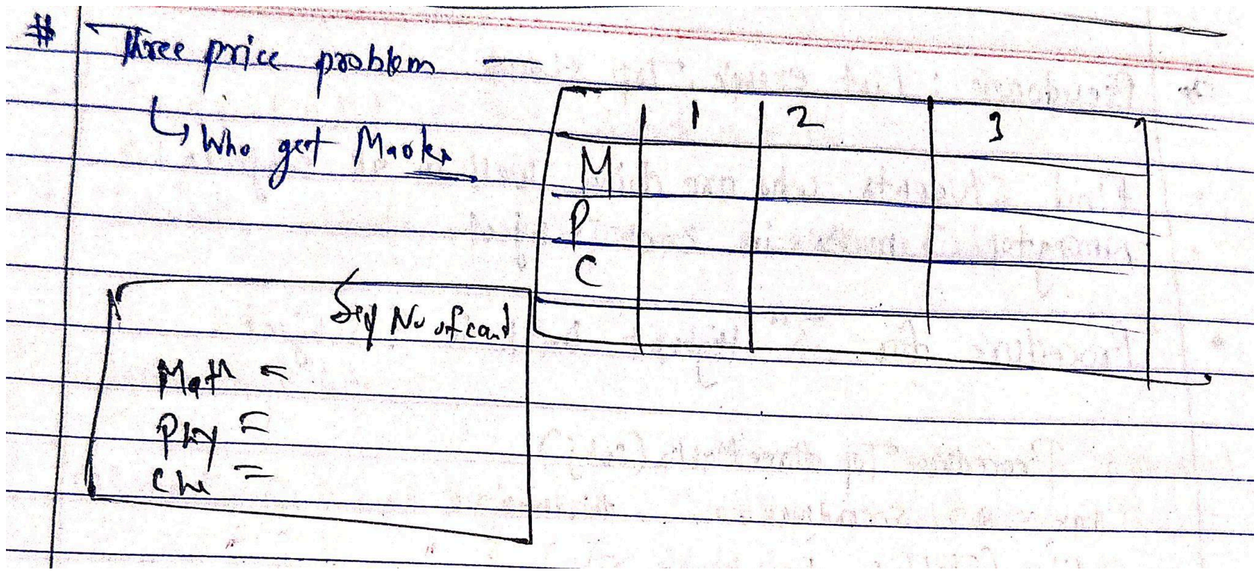
Summary

- A list is a sequence of values
- Write a list as `[x1,x2,...,xn]`
- Combine lists using `++`
  - `[x1,x2] ++ [y1,y2,y3]` $\mapsto$ `[x1,x2,y1,y2,y3]`
- Extending list `l` by an item `x`
  - `l = l ++ [x]`
- `foreach` iterates through values in a list

```
foreach x in l {
    Do something with x
}
```

QN: 5

<u>Lecture 3 : Operations on the data collected in three prizes problem using lists</u>



Three price problem —

↳ Who get Marks

	1	2	3
M			
P			
C			

Seq No of card

Math =

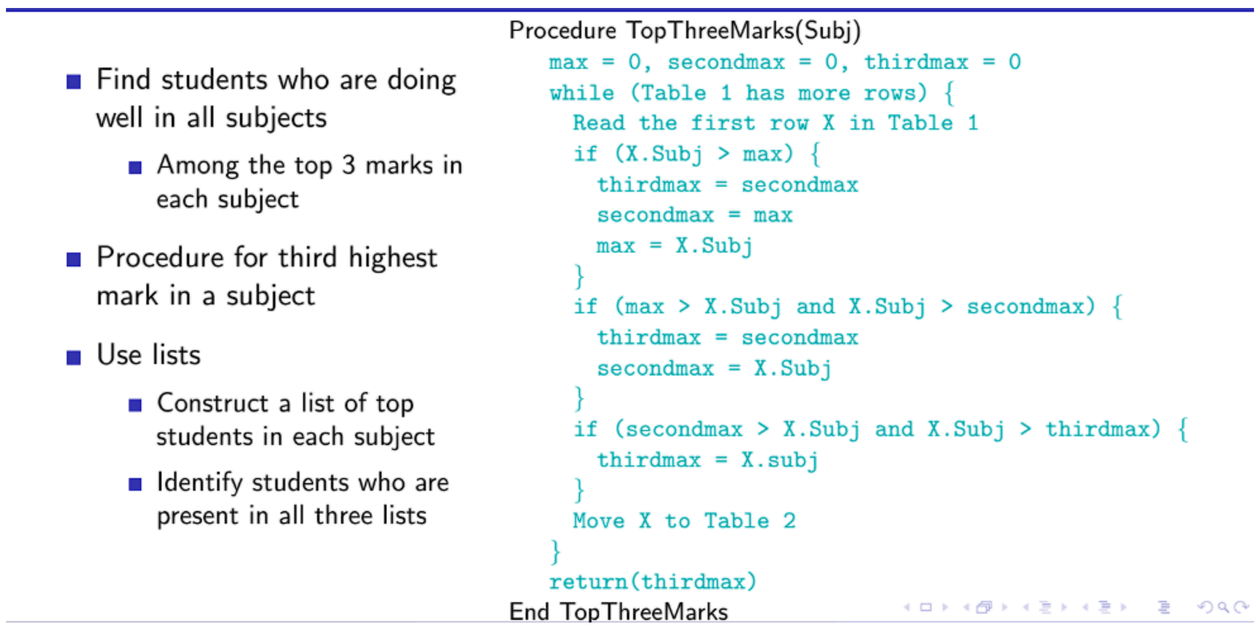
Phy =

Chem =

QN : 1,2

Lecture 4 : Pseudocode for operations on the data collected in three prizes problem using lists

Pseudocode: List example, top students



Identifying top students

- Find students who are doing well in all subjects
  - Among the top 3 marks in each subject
- Procedure for third highest mark in a subject
- Use lists
  - Construct a list of top students in each subject
  - Identify students who are present in all three lists

```
Procedure TopThreeMarks(Subj)
   max = 0, secondmax = 0, thirdmax = 0
   while (Table 1 has more rows) {
     Read the first row X in Table 1
     if (X.Subj > max) {
       thirdmax = secondmax
       secondmax = max
       max = X.Subj
     }
     if (max > X.Subj and X.Subj > secondmax) {
       thirdmax = secondmax
       secondmax = X.Subj
     }
     if (secondmax > X.Subj and X.Subj > thirdmax) {
       thirdmax = X.subj
     }
     Move X to Table 2
   }
   return(thirdmax)
End TopThreeMarks
```

Constructing the lists

- Obtain cutoffs in each subject
- Initialize lists for each subject
- Scan each row
- For each subject, check if the marks are within the top three
- If so, append to the list for that subject

```
cutoffMaths = TopThreeMarks(Mathematics)
cutoffPhys = TopThreeMarks(Physics)
cutoffChem = TopThreeMarks(Chemistry)

mathsList = []
physList = []
chemList = []

while (Table 1 has more rows) {
   Read the first row X in Table 1
   if (X.Mathematics >= cutoffMaths) {
     mathsList = mathsList ++ [X.SeqNo]
   }
   if (X.Physics >= cutoffPhys) {
     physList = physList ++ [X.SeqNo]
   }
   if (X.Chemistry >= cutoffChem) {
     chemList = chemList ++ [X.SeqNo]
   }
   Move X to Table 2
}
```

Find the overall toppers

- First find students who are toppers in Maths and Physics

```
mathsPhysList = []

foreach x in mathsList {
   foreach y in PhysList {
     if (x == y) {
       mathsPhysList = mathsPhysList ++ [x]
     }
   }
}
```

- Then match these toppers with toppers in Chemistry

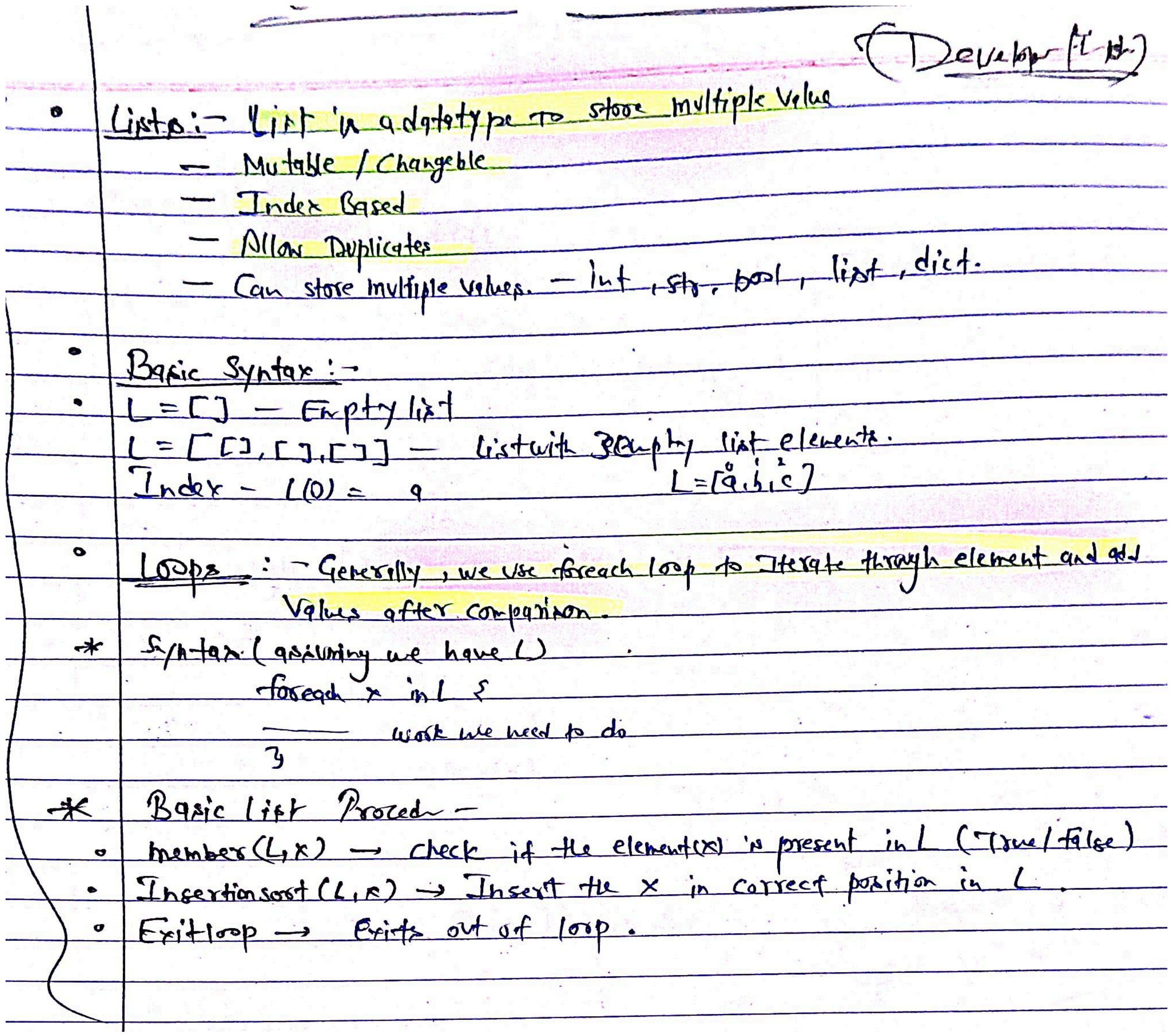
```
mathsPhysChemList = []

foreach x in mathsPhysList {
   foreach y in chemList {
     if (x == y) {
       mathsPhysChemList =
                   mathsPhysChemList ++ [x]
     }
   }
}
```

Summary

- Lists are useful to collect items that share some property
- Nested iteration can find common elements across two lists
- Can group lists to process more than two lists
  - Find common items across four lists, `list1`, `list2`, `list3`, `list4`
  - Nested iteration on `list1`, `list2` constructs `list12` of common items in first two lists
  - Nested iteration on `list3`, `list4` constructs `list34` of common items in last two lists
  - Nested iteration on `list12`, `list34` finds common items across all four lists

Lecture 5 : Basic List Operations



(Developer (L.H.)

- Lists :- List is a datatype to store multiple Value
  - Mutable / Changeble
  - Index Based
  - Allow Duplicates
  - Can store multiple values. — Int, str, bool, list, dict.

- Basic Syntax :-
- L = [] — Empty list
  L = [[], [], []] — list with 3 empty list elements.
  Index - L(0) = a   L = [a, b, c] (0, 1, 2)

- Loops :- Generally, we use foreach loop to Iterate through element and add Values after comparison.

* Syntax (assuming we have L)

```
foreach x in L {
    ———      work we need to do
}
```

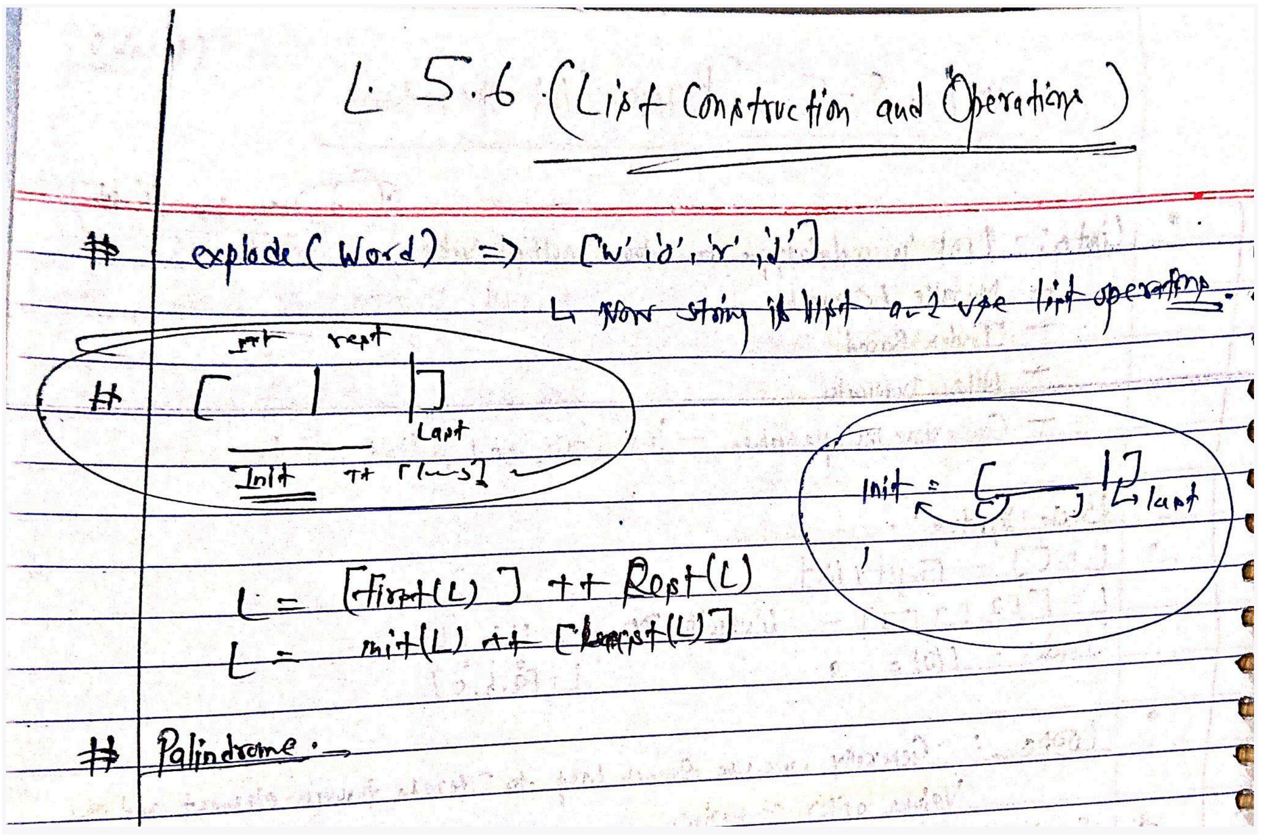
* Basic List Procedure -
- member(L, x) → check if the element(x) is present in L (True / False)
- Insertionsort (L, x) → Insert the x in correct position in L.
- Exitloop → Exits out of loop.

Let L be a list and

1. length(L) returns the number of elements in L. For example length([1, 4, 6]) returns 3
2. first(L) returns the first element of list L. For example first([1, 4, 6]) returns 1
3. rest(L) returns a list after removing the first element of L. For example rest([1, 4, 6]) returns [4, 6]

QN : 3

Lecture 6 : List construction and operations



L 5.6 (List Construction and Operations)

explode(Word) => ['W','o','r','d']

↳ Now string is list and use list operations.

[|]

first rest last

Init ++ [last]

init = [ ] [last]

L = [first(L)] ++ Rest(L)

L = init(L) ++ [last(L)]

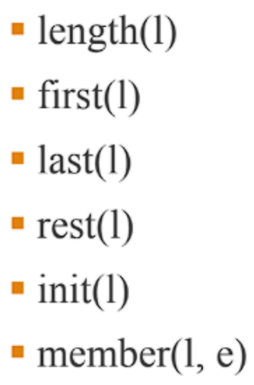
Palindrome :-

Let L be a list then

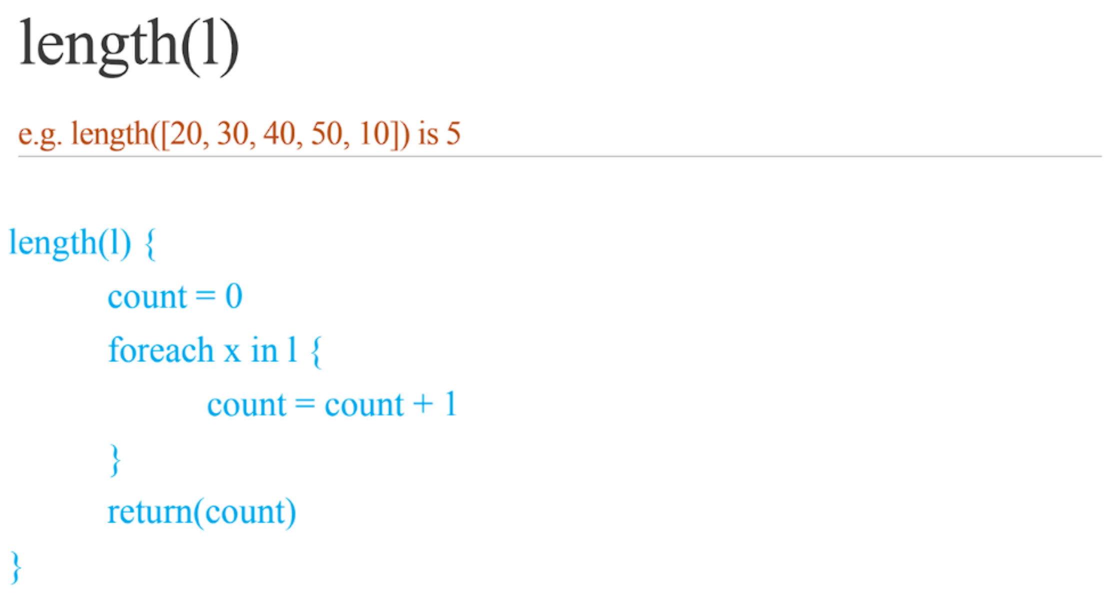
1. length(L) returns the number of elements in L. For example length([1, 4, 6]) returns 3
2. last(L) returns the last element of list L. For example first([1, 4, 6]) returns 6
3. init(L) returns a list after removing the last element of list L. For example init([1, 4, 6]) returns [1, 4]

Tutorial 5.1: Tutorial on pseudocode for list functions

List functions



- length(l)
- first(l)
- last(l)
- rest(l)
- init(l)
- member(l, e)



length(l)

e.g. length([20, 30, 40, 50, 10]) is 5

```
length(l) {
    count = 0
    foreach x in l {
        count = count + 1
    }
    return(count)
}
```

first(l)

e.g. first([20, 30, 40, 50, 10]) is 20

```
first(l) {
        foreach x in l {
                return(x)
        }
}
```

last(l)

e.g. last([20, 30, 40, 50, 10]) is 10

```
last(l) {
        foreach x in l {
                e = x
        }
        return(e)
}
```

NOTE : IF WE PASS EMPTY LIST THEN RETURN VALUE IS UNDEFINED BECAUSE WE CAN'T COMPUTE 1ST OR LAST OF AN EMPTY LIST

rest(l)

e.g. rest([20, 30, 40, 50, 10]) is [30, 40, 50, 10]

```
rest(l) {
        found = False
        restList = []
        foreach x in l {
                if (found)  {
                        restList = restList ++ [x]
                }
                else {
                        found = True
                }
        }
        return(restList)
}
```

init(l)

e.g. init([20, 30, 40, 50, 10]) is [20, 30, 40, 50]

```
init(l) {
        found = False
        initList = []
        foreach x in l {
                if (found)  {
                        initList = initList ++ [prev]
                }
                else {
                        found = True
                }
                prev = x
        }
        return(initList)
}
```

member(l, e)

e.g. member([20, 30, 40, 50], 30) is True, member([20, 30, 40, 50], 10) is False

```
member(l, e) {
    foreach x in l {
        if (e == x) {
            return(True)
        }
    }
    return(False)
}
```

PA : 3,4,6,7,8,9
GA : 3,4,5,6,8