Computational Thinking

Prof. Madhavan Mukund
Department of Computer Science
Chennai Mathematical Institute

Prof. G. Venkatesh
Indian Institute of Technology Madras

Mr. Omkar Joshi
Course Instructor
IITM Online Degree Programme

Computational Thinking

Tutorial on pseudocode for lists and dictionaries

Content

- List
  - Introduction
  - ++ operator
  - List functions
- Insertion sort
- Dictionary
  - Introduction
  - Dictionary functions
- Index vs. Key

List introduction

- List is a collection of values stored in a sequential order.
- List can store values of different datatypes.
- List can store duplicate values.
- List preserves the order of elements in it.
- An element in a list can be another list or a dictionary.
- List is initialised using [] notation.
- *l = []* or *length(l) == 0*, denotes that the list l is empty.

++ operator

- *list* = *list* ++ *list* (it requires list operands)
- ++ operator is used for two different operations, append and extend
  - Append: It merges two lists

    e.g. if l1 = [1, 2, 3] and l2 = [“a”, “b”, “c”] then l1 ++ l2 == [1, 2, 3, “a”, “b”, “c”]

  - Extend: It adds an element to a list

    e.g. if l = [1, 2, 3] and x = 4 then l ++ [x] == [1, 2, 3, 4]

    if l = [1, 2, 3] and x = 4 then [x] ++ l == [4, 1, 2, 3]

    l ++ x or x ++ l are incorrect statements because x is an integer variable not a list

++ operator (continue…)

- Adding lists to list

  e.g. if l = [], l1 = [1, 2, 3] and l2 = [a, b, c]

  l = l ++ [l1] then l == [[1, 2, 3]]

  l = l ++ [l2] then l == [[1, 2, 3], [a, b, c]]

- Adding dictionaries to list

  e.g. if l = [], d1 = {1: 1, 2: 4, 3: 9} and d2 = {“a”: “A”, “b”: “B”, “c”: “C”}

  l = l ++ [d1] then l == [{1: 1, 2: 4, 3: 9}]

  l = l ++ [d2] then l == [{1: 1, 2: 4, 3: 9}, {“a”: “A”, “b”: “B”, “c”: “C”}]

Input List (l)	Output / Return type				
	length(l)	**first(l)**	**last(l)**	**rest(l)**	**init(l)**
[]	0 Integer	Undefined	Undefined	[] List	[] List
[10]	1 Integer	10 integer	10 integer	[] List	[] List
[1, 2, 3, 4]	4 integer	1 integer	4 integer	[2, 3, 4] list	[1, 2, 3] list
["a", "b", "c"]	3 integer	"a" string	"c" string	["b", "c"] list	["a", "b"] list
[2, "b", 3, 1, "c", "a"]	6 integer	2 integer	“a" string	["b", 3, 1, "c", "a"] list	[2, "b", 3, 1, "c"] list
[[1, 2, 3], [1, 4, 9], [1, 8, 27]]	3 integer	[1, 2, 3] list	[1, 8, 27] list	[[1, 4, 9], [1, 8, 27]] list	[[1, 2, 3], [1, 4, 9]] list
[{1: 1, 2: 4, 3: 9}, {"a": "A", "b": "B", "c": "C"}]	2 integer	{1: 1, 2: 4, 3: 9} dictionary	{"a": "A", "b": "B", "c": "C"} dictionary	[{"a": "A", "b": "B", "c": "C"}] list	[{1: 1, 2: 4, 3: 9}] list
[[1, 8, 27], {1: 1, 2: 4, 3: 9}]	2 integer	[1, 8, 27] list	{1: 1, 2: 4, 3: 9} dictionary	[{1: 1, 2: 4, 3: 9}] list	[[1, 8, 27]] list

Insertion sort

```
inputList = [5, 2, 1, 3, 4]
sortedList = []
foreach x in inputList {
        sortedList = insertAnElement (sortedList, x)
}
Procedure insertAnElement (sl, x)
        …
        …
        …
End insertAnElement
```

```
Procedure insertAnElement (sl, x)
    tempList = []
    inserted = False
    foreach ele in sl {
        if (not(inserted)){
            if (x < ele) {
                tempList = tempList ++ [x]
                inserted = True
            }
        }
        tempList = tempList ++ [ele]
    }
    if (not(inserted)) {
        tempList = tempList ++ [x]
    }
    return (tempList)
End insertAnElement
```

Procedure call	sortedList / sl	x	tempList	inserted	ele	Remark
1	[]	5	[]	False	--	foreach loop will not execute because **sl** is empty
			[5]			Because **inserted** == False
2	[5]	2	[]	False	5	
			[2]	True		Because **x** < **ele**
			[2, 5]			
3	[2, 5]	1	[]	False	2	Because **x** < **ele**
			[1]	True		
			[1, 2]			
			[1, 2, 5]		5	
4	[1, 2, 5]	3	[]	False	1	
			[1]		2	
			[1, 2]		5	
			[1, 2, 3]	True		Because **x** < **ele**
			[1, 2, 3, 5]			
5	[1, 2, 3, 5]	4	[]	False	1	
			[1]		2	
			[1, 2]		3	
			[1, 2, 3]		5	
			[1, 2, 3, 4]	True		Because **x** < **ele**
			[1, 2, 3, 4, 5]			
	[1, 2, 3, 4, 5]					

Dictionary introduction

- Dictionary is a collection of elements stored as *key: value* pair.
- Dictionary can store keys and values of different datatypes.
- Dictionary can store duplicate values but keys must be unique.
- Dictionary does not preserve the order of elements in it.
- A value in a dictionary can be another dictionary or a list.
- Dictionary is initialised using {} notation.
- Dictionary value is accessed using *d[key]* notation

Dictionary functions

- *keys(d)* function returns a list of keys present in the dictionary.

  e.g. if d = {“a”: 1, “b”: 2, “c”: 3} then keys(d) can return

  [“a”, “b”, “c”] or [“a”, “c”, “b”] or [“b”, “c”, “a”] or [“b”, “a”, “c”] or

  [“c”, “a”, “b”] or [“c”, “b”, “a”]

  Dictionary does not preserve order. Therefore, keys function can return the list of keys in any order.

- *isKey(d, k)* function returns True if k is a key in dictionary d else it returns False

  e.g. if d = {“a”: 1, “b”: 2, “c”: 3} then isKey(d, “a”) will return True

  e.g. if d = {“a”: 1, “b”: 2, “c”: 3} then isKey(d, “z”) will return False

Index vs. Key

Index	Key
List has indices	Dictionary has keys
Index indicates the position of the element in the list	Key is an unique entity used to identify its value in the dictionary
Index is always an integer starting from 0 to n	Key can be of any datatype
e.g. l = [10, 20, 30, 40] then index 0 is 10, index 1 is 20 and so on.	e.g. d = {1: 10, 2: 20, 3: 30, 4: 40} then d[2] notation is used to access the value 20