Computer Science

Chapter 10 - Data Types and Structures

10.1 Data Types and Records

10.2 Arrays

10.3 Files

10.4 Introduction to Abstract Data Types (ADT)

By: Noureddine Tadjerout

Learning Objectives

10.1 Data Types and Records

Select and use appropriate data types for a problem solution

including integer, real, char, string, Boolean, date (pseudocode will use the following **data types**: **INTEGER**, **REAL**, **CHAR, STRING**, **BOOLEAN**, **DATE**, **ARRAY**, **FILE**)

Show understanding of the purpose of a record structure to hold a set of data of different data types under one identifier

- Write pseudocode to define a record structure.
- Write pseudocode to read data from a record structure and save data to a record structure

Learning Objectives

10.2 Arrays

Use the technical terms associated with arrays (Including index, upper and lower bound)

Select a suitable data structure (1D or 2D array) to use for a given task

Write pseudocode for 1D and 2D arrays

Write pseudocode to process array data

- Sort using a bubble sort
- Search using a linear search

10.3 Files

Show understanding of why files are needed

Write pseudocode to handle text files that consist of one or more lines

Learning Objectives

10.4 Introduction to Abstract Data Types (ADT)

Show understanding that an ADT is a collection of data and a set of operations on those data.

Show understanding that a stack, queue and linked list are examples of ADTs

Describe the key features of a stack, queue and linked list and justify their use for a given situation

Use a stack, queue and linked list to store data

(You will not be required to write pseudocode for these structures, but they should be able to add, edit and delete data from these structures)

Describe how a queue, stack and linked list can be implemented using arrays

Resources:

https://www.hoddereducation.co.uk/cambridgeasalevelcomputerscience

Resources:

Craig 'n' Dave For Students

CRAIG 'N' DAVE

To explore the entire series,
visit https://craigndave.org/free-videos

Resources:

Computer Science for Cambridge International AS & A Level

Endorsed by Cambridge

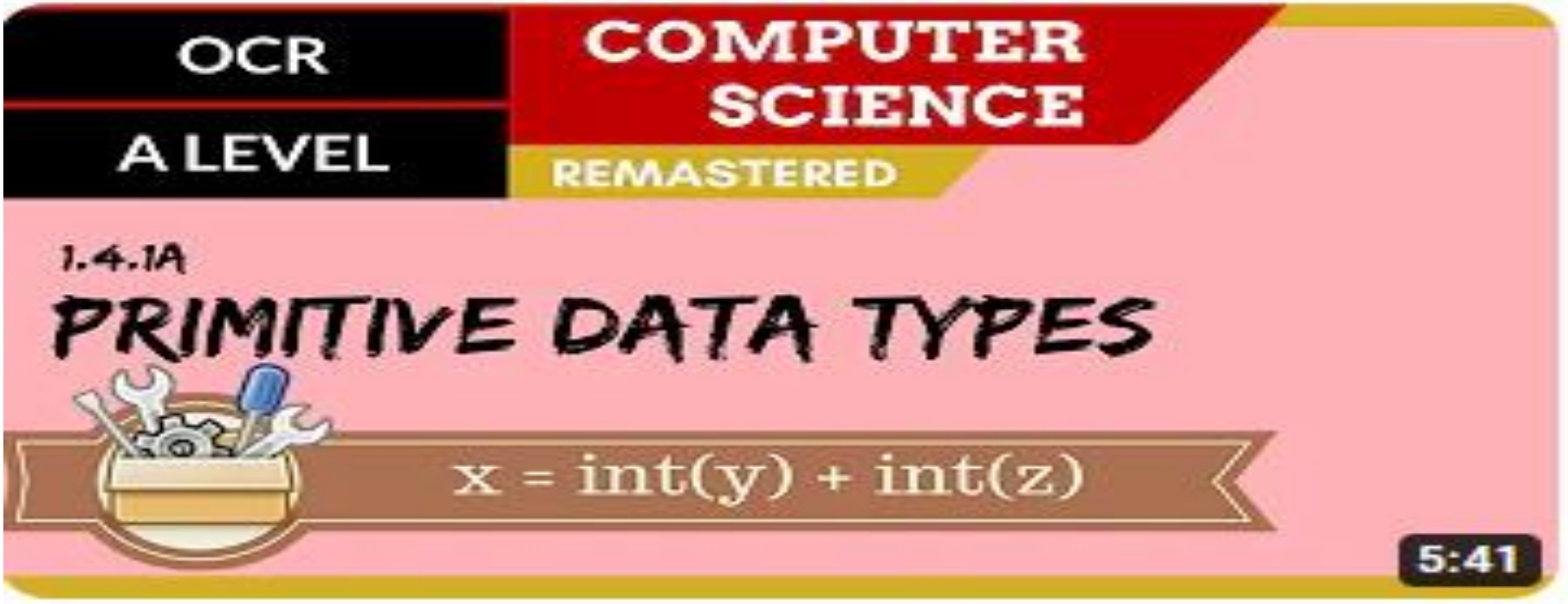
Description	**Supports the full syllabus for examination from 2021.** This coursebook provides a structured and progressive guide to the theoretical and practical skills required by the syllabus. It builds learners' confidence in using a range of technology and programming languages. Concepts are reinforced with exercises, discussion points and reflection tasks, as well as exam-style and past paper questions. An accompanying teacher's resource is also available and provides chapter by chapter support, lesson plans and answers. Printable worksheets provide differentiation and ready-made lesson materials. **Further information and sample material for this coursebook** > Publisher website **Related endorsed titles** > Teacher's Resource > Revision Guide
Publisher	Cambridge University Press
Author	Langfield, S and Duddell, D
ISBN	9781108733755
Published Date	2019
Website	education.cambridge.org

https://www.cambridgeinternational.org/programmes-and-qualifications/cambridge-international-as-and-a-level-computer-science-9618/published-resources/

10.1 Data Types and Records

10.1.1 - Data Types:

https://www.youtube.com/watch?v=A_R49e7su28



Learning Objectives

10.1 Data Types and Records

Candidates should be able to:	Notes and guidance
Select and use appropriate data types for a problem solution	including integer, real, char, string, Boolean, date (pseudocode will use the following data types: INTEGER, REAL, CHAR, STRING, BOOLEAN, DATE, ARRAY, FILE)
Show understanding of the purpose of a record structure to hold a set of data of different data types under one identifier	Write pseudocode to define a record structure. Write pseudocode to read data from a record structure and save data to a record structure

10.1 Data Types and Records

10.1.1 - Data Types:

INTEGER	A signed whole number
REAL	A signed number with a decimal point
CHAR	A single character
STRING	A sequence of zero or more characters
BOOLEAN	The logical values TRUE and FALSE
DATE	A date consisting of day, month and year, sometimes including a time in hours, minutes and seconds

- **int** (for integer)
- **float**
- **str** (for string)
- **list**
- **tuple**
- **dict** (for dictionary)
- **set**
- **bool** (for Boolean True/False)

- List[] :- Collection of elements can be changed (mutable mean
- Tuple():- Collection of elements can’t be changed (immutable mean unable to be changed).
- Set{} :- Collection of unique elements. Sets do not allow repetition

10.1 Data Types and Records

10.1.1 - Data Types:

In pseudocode and some programming languages, before data can be used, the type needs to be decided. This is done by declaring the data type for each item to be used. Each data item is identified by a unique name, called an **identifier**.

In pseudocode a declaration statement takes this form:

```
DECLARE <identifier> : <data type>
```

For example:

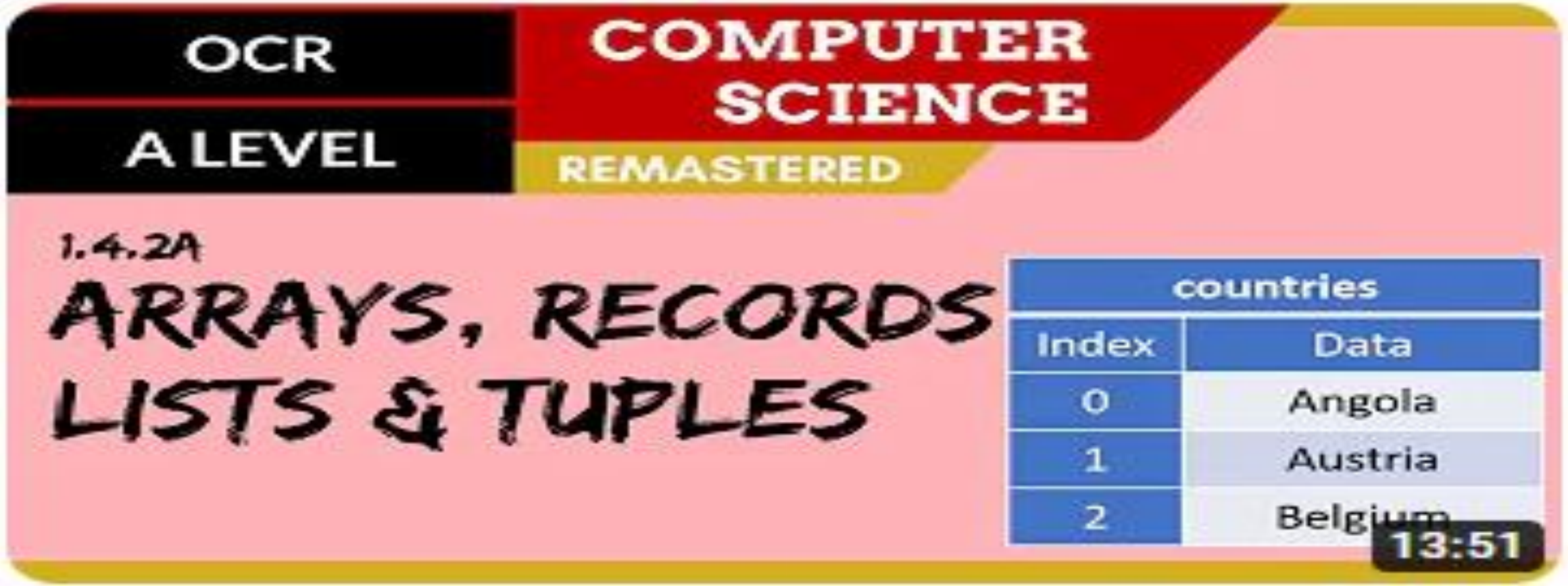
```
DECLARE myBirthday : DATE
```

- **Identifier:** a unique name applied to an item of data.
- **Data type:** a classification attributed to an item of data, which determines the types of value it can take and how it can be used.

10.1 Data Types and Records

Arrays, records, lists and tuples

https://www.youtube.com/watch?v=e3crB2Yi4ps



10.1 Data Types and Records

Arrays, records, lists and tuples

https://www.youtube.com/watch?v=h3Pm4n6icIM

10.1 Data Types and Records

10.1.2 - Records:

Records are composite data types that contain references to at least 1 other datatype within them. The advantage of records is that all related data is tied together in one place. They are incredibly useful and popular in many languages. This allow a programmer to refer to these items using the same identifier, enabling a structure approach to using related items. For example, a record for a book could include title, author, publisher, number of pages, and whether it is a fiction or non-fiction.

area.py - C:/Users/v90001010/AppData/Local/Programs/Py

File Edit Format Run Options Window Help

```
import types

giraffe = types.SimpleNamespace()

giraffe.name = "Gerry"
giraffe.height = 350
giraffe.age = 15
giraffe.origin = "Africa"
print( giraffe)
```

```
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py
namespace(age=15, height=350, name='Gerry', origin='Africa')
>>>
```

10.1 Data Types and Records

10.1.2 - Records:

A record data type is one example of a composite user-defined data type. A composite data type references other existing data types when it is defined. A composite data type must be defined before it can be used. Any data type not provided by a programming language must be defined before it can be used.

In pseudocode, a record data type definition takes the following form:

```
TYPE
<Typename>
     DECLARE <identifier> : <data type>
     DECLARE <identifier> : <data type>
     DECLARE <identifier> : <data type>
     ::
     ::
ENDTYPE
```

10.1 Data Types and Records

10.1.2 - Records:

For example, the book record data type could be defined like this:

```
TYPE
TbookRecord
    DECLARE title : STRING
    DECLARE author : STRING
    DECLARE publisher : STRING
    DECLARE noPages : INTEGER
    DECLARE fiction : BOOLEAN
ENDTYPE
```

10.1 Data Types and Records

10.1.2 - Records:

The data type, **TbookRecord**, is now available for use and an identifier may now be declared in the usual way

DECLARE Book : NTbookRecord

Items from the record are now available for use and are identified by:

<identifier><item identfier>

For example:

Book.author ⟵ "Noureddine Tadjerout"

Book.Fiction ⟵ FALSE

- **Record (data type):** a composite data type comprising several related items that may be of different data types.
- **Composite data type:** a data type constructed using several of the basic data types available in a particular programming language.

Learning Objectives

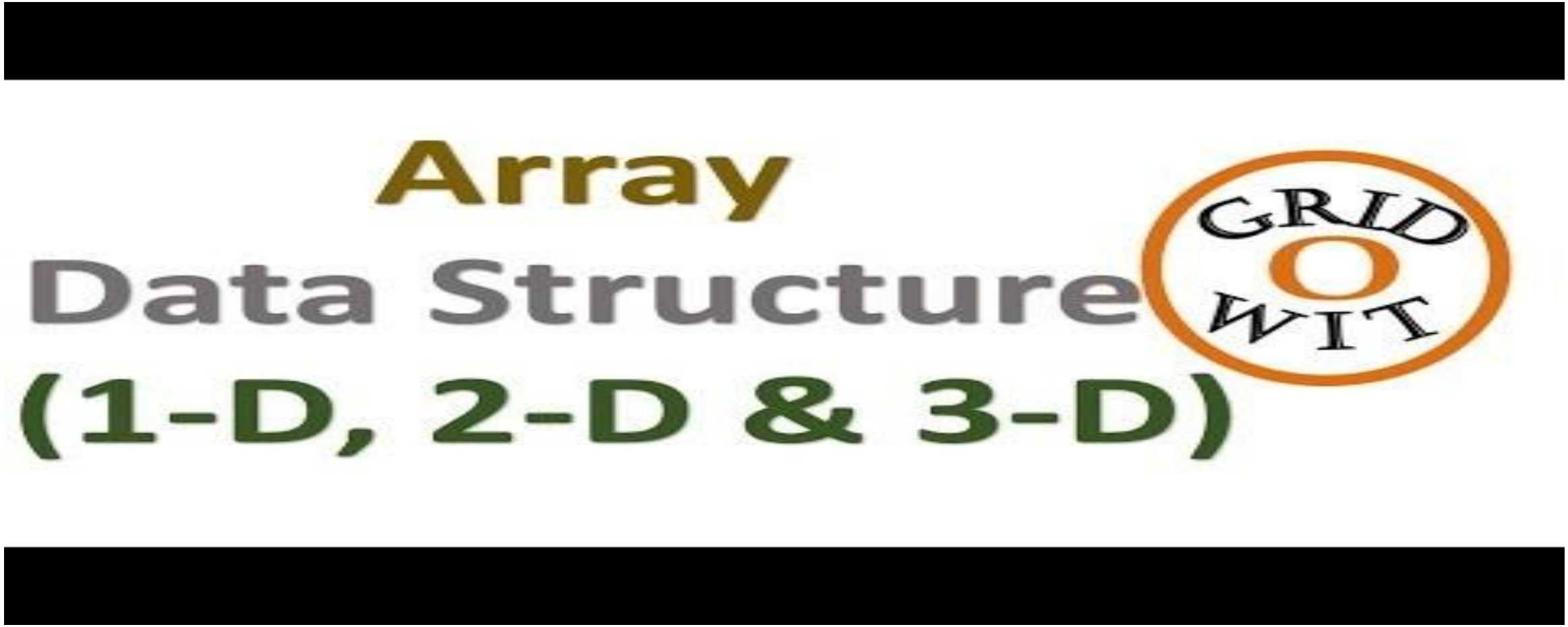
10.2 Arrays

Candidates should be able to:	Notes and guidance
Use the technical terms associated with arrays	Including index, upper and lower bound
Select a suitable data structure (1D or 2D array) to use for a given task	
Write pseudocode for 1D and 2D arrays	
Write pseudocode to process array data	Sort using a bubble sort Search using a linear search

10.2 Arrays

Click the link below the watch the video for Array Data Structure (1D 2D & 3D Array)

https://www.youtube.com/watch?v=JJ2iSrwIkVs



10.2 Arrays

An **array** is a data structure containing several elements of the same data type; these elements can be accessed using the same identifier name. The position of each element in an array is identified using the array's **index**. The index of the first element in an array is the **lower bound** and the index of the last element is the **upper bound**.

The lower bound of an array is usually set as zero or one. Some programming languages can automatically set the lower bound of an array.

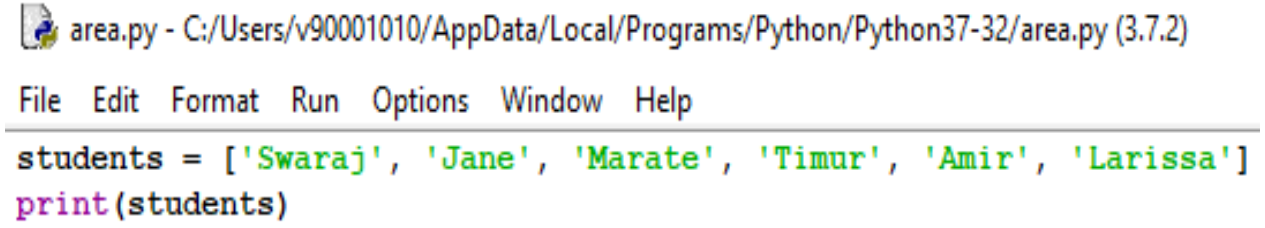
Arrays can be one-dimensional or multi-dimensional. In this chapter, we will look at **one dimensional (1D)** and **two-dimensional (2D) arrays**.

- **Array:** a data structure containing several elements of the same data type.
- **Index (array):** a numerical indicator of an item of data's position in an array.
- **Lower bound:** the index of the first element in an array, usually 0 or 1.
- **Upper bound:** the index of the last element in an array.

10.2 Arrays

An arrays is structure containing several elements of the same data type; these elements can be accessed using the same identifier name. The position of each element in an array is identified using the array's index. The index of the first element in an array is the lower bound and the index of the last element is the upper bound. Python arrays are a data structure like lists. They contain a number of objects that can be of different data types. In addition, Python arrays can be iterated and have a number of built-in functions to handle them For example, if you have a list of student names that you want to store, you may want to store them in an array.

Here is the basic syntax to declare an array in Python:



We have created an array! When we print our list of students, our output is the same as the array with data we declared:

```
>>>
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Python
['Swaraj', 'Jane', 'Marate', 'Timur', 'Amir', 'Larissa']
```

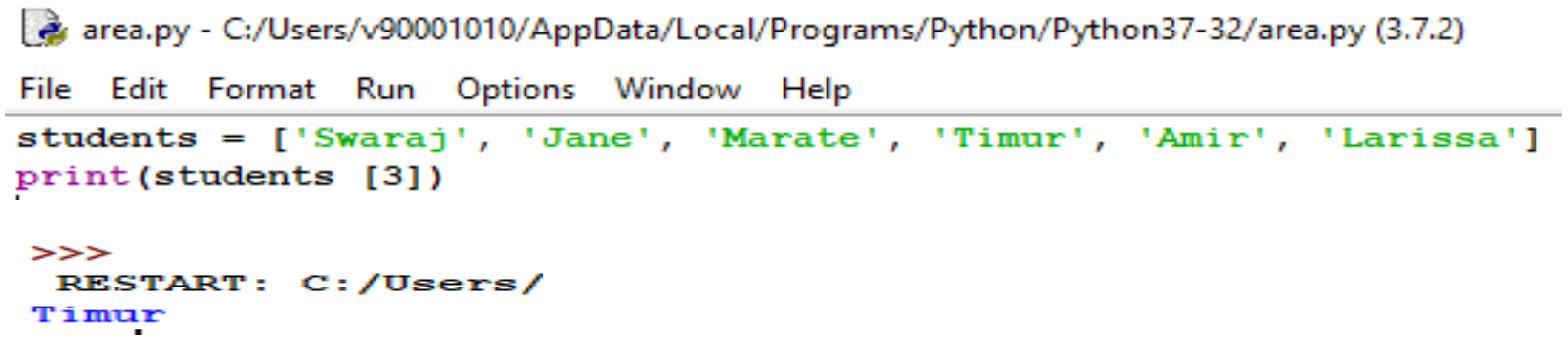
10.2 Arrays

Each item in a list can be referenced, or the list can be referenced as a whole. This means that we are able to perform operations on specific items in our array.

- This works because each value in an array has an *index* number, which tells us where the item is within an array. Indexes start with and correspond with a certain item in an array.
- Here is the index of array values we declared above :

"Swaraj"	"Jane"	"Marate"	"Amir"	"Timur"	"Larissa"
0	1	2	3	4	5

Because each item has its own index number, we can access each item in our array. We can do so by calling our list and specifying an index number, like so: for example : **print(students [3]) then it will print Timur only**



10.2 Arrays

10.2.1- 1 Dimensional (1D) Arrays:

A 1 array can be referred to as a list. Here is an example of a list with four elements and a lower bound of zero. When a 1D array is declared in pseudocode, the lower bound(LB), upper bound (UB) and data type are included.

Lower Bound ⟶

Upper Bound ⟶

Index	MyList
[0]	22
[1]	34
[2]	56
[3]	76

For example: `DECLARE <identifier> : ARRAY[LB:UB] OF <data type>`

DECLARE MyList : ARRAY [0:3] of INTEGER

The declare array can then be used as follows:

DECLARE myList ; Array[0:3] OF INTEGER

myList [2] ⟵ 56

10.2 Arrays

10.2.2 - 1 Dimensional (1D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 1:

area.py - C:/Users/v90001010/AppData/Local/Progr

File Edit Format Run Options Window Help

```
arr = [ 1, 2 ,3, 4, 5]
print (arr)
print (arr[2])
print (arr[4])
```

```
>>>
 RESTART: C:/Users/v90001010/App
[1, 2, 3, 4, 5]
3
5
```

Example 2:

File Edit Format Run Options Window Help

```
student_marks = ['Swaraj' , 45, 40.5, 'Noureddine' , 50,40]
marks = student_marks[1]+student_marks[2]
print(student_marks[0] + ' has got in total = %d + %f = %f ' % (student_marks[1], student_marks[2], marks ))
print(student_marks[3] + ' has got in total = %d + %f = %f ' % (student_marks[4], student_marks[5], marks ))
```

```
>>>
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Pythc
Swaraj has got in total = 45 + 40.500000 = 85.500000
Noureddine has got in total = 50 + 40.000000 = 85.500000
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

A 2D array can be referred to as a table, with rows and columns. Here is an example of a table with nine rows and three columns (27 elements) and lower bounds of zero.

MyArray

Column index

Row index	[r,0]	[r,1]	[r,2]	
[0,c]	27	31	17	← lower bound row
[1,c]	19	67	48	
[2,c]	36	98	29	
[3,c]	42	22	95	
[4,c]	16	35	61	
[5,c]	89	46	47	
[6,c]	21	71	28	
[7,c]	16	23	13	
[8,c]	55	11	77	← upper bound row

↑ lower bound column

↑ upper bound column

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

When a 2D array is declared in pseudocode, the lower bound for rows (LBR) and the upper bound for rows (UBR), the lower bound for columns (LBC) and the upper bound for columns (UBC), and data type are included:

```
DECLARE <identifier> : ARRAY[LBR:UBR, LBC:UBC] OF <data type>
```

For example:

```
DECLARE myArray : ARRAY[0:8,0:2] OF INTEGER
```

The declared array can then be used, as follows:

```
myArray[7,0] ← 16
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

A 2D array can be referred to as a table, with rows and columns. Here is an example of a table with four rows and three columns (12 elements) and lower bounds of zero.

[0,c]	**23**	**43**	**34**	⟵ Lower Bound row
[1,c]	**14**	**67**	**20**	
[2,c]	**56**	**88**	**7**	
[3,c]	**22**	**91**	**9**	⟵ Upper Bound row

```
DECLARE <identifier> : ARRAY[LBR:UBR, LBC:UBC] OF <data type>
```

For example:

DECLARE MyArray : ARRAY[0:3,0:2] OF INTEGER

MyArray[3,2] ⟵ 9

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Arrays vs lists in Python

- Python natively uses lists instead of arrays
- Python lists are very simple to use and highly flexible
- They can contain heterogeneous data(lots of different types of data)
- They make less efficient use of memory than arrays
- Certain math's/data science uses require the use of arrays instead of lists, but this is rare in general programming)

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 1:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py (3.7.2)

File Edit Format Run Options Window Help

```
from array import *

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]

print(T[0])

print(T[1][2])
```

```
 RESTART: C:/Users/v90001010/App
[11, 12, 5, 2]
10
```

Example 2:

File Edit Format Run Options Window Help

```
from array import *

T = [[11, 12, 5, 2], [15, 6,10], [10, 8, 12, 5], [12,15,8,6]]
for r in T:
   for c in r:
      print(c,end = " ")
   print()
```

```
>>>
 RESTART: C:/Users/v90001010/.
11 12 5 2
15 6 10
10 8 12 5
12 15 8 6
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 3:

area.py - C:/Users/v90001010/AppData/Local/Progra

File Edit Format Run Options Window Help

```
a = [[1, 2, 3], [4, 5, 6]]
print(a[0])
print(a[1])
b = a[0]
print(b)
print(a[0][2])
a[0][1] = 7
print(a)
print(b)
b[2] = 9
print(a[0])
print(b)
```

```
 RESTART: C:/Users/v90001010/Ap
[1, 2, 3]
[4, 5, 6]
[1, 2, 3]
3
[[1, 7, 3], [4, 5, 6]]
[1, 7, 3]
[1, 7, 9]
[1, 7, 9]
>>>
```

Example 4:

File Edit Format Run Options Window Help

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=' ')
    print()
```

```
 RESTART: C:/Users/v90001010
1 2 3 4
5 6
7 8 9
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 5:

File Edit Format Run Options Window Help

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
for row in a:
    for elem in row:
        print(elem, end=' ')
    print()
```

```
>>>
 RESTART: C:/Users/v90001010/
1 2 3 4
5 6
7 8 9
```

Example 6:

area.py - C:/Users/v90001010/AppData/Local/Programs/

File Edit Format Run Options Window Help

```
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
s = 0
for i in range(len(a)):
    for j in range(len(a[i])):
        s += a[i][j]
print(s)
```

```
>>>
 RESTART: C:/Users/v90001010
45
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 7:

```
File  Edit  Format  Run  Options  Window  Help
a = [[1, 2, 3, 4], [5, 6], [7, 8, 9]]
s = 0
for row in a:
    for elem in row:
        s += elem
print(s)
```

```
>>>
 RESTART: C:/Users/v90001010
45
>>>
```

Example 8:

```
File  Edit  Format  Run  Options  W
n = 3
m = 4
a = [[0] * m] * n
a[0][0] = 5
print(a[1][0])
```

```
>>>
 RESTART: C:/Users/v900010
5
>>>
```

10.2 Arrays

10.2.2 - 2 Dimensional (2D) Arrays:

Open Python IDLE and copy the code below for more practical activities

Example 9:

*area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Pyth

File Edit Format Run Options Window Help

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(n):
        if i < j:
            a[i][j] = 0
        elif i > j:
            a[i][j] = 2
        else:
            a[i][j] = 1
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

```
>>>
 RESTART: C:/Users/v90001010/
1 0 0 0
2 1 0 0
2 2 1 0
2 2 2 1
```

Example 10:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Pyth

File Edit Format Run Options Window Help

```
n = 4
a = [[0] * n for i in range(n)]
for i in range(n):
    for j in range(0, i):
        a[i][j] = 2
    a[i][i] = 1
    for j in range(i + 1, n):
        a[i][j] = 0
for row in a:
    print(' '.join([str(elem) for elem in row]))
```

```
>>>
 RESTART: C:/Users/v90001010/A
1 0 0 0
2 1 0 0
2 2 1 0
2 2 2 1
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Click the link below the watch the video for a linear search

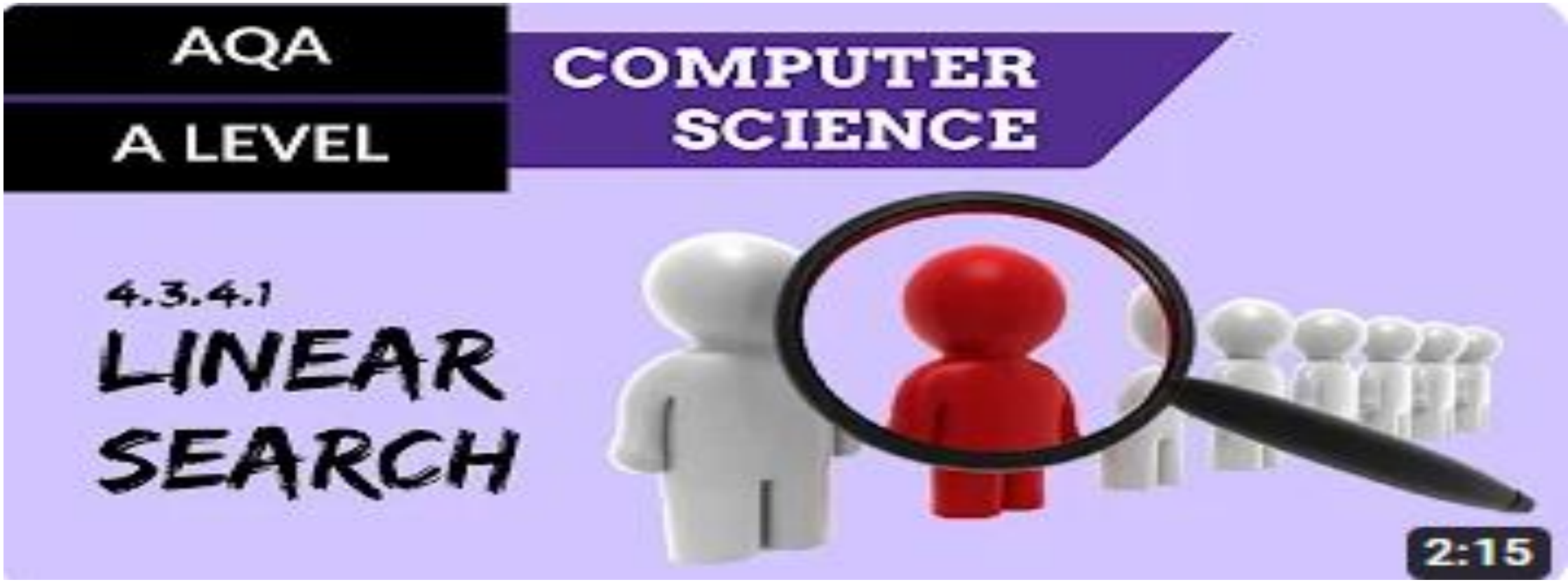
https://www.youtube.com/watch?v=0HXBo--PaAw
Recap IGCSE Linear Search

10.2 Arrays

10.2.3 - Using a Linear Search:

Click the link below the watch the video for a linear search

https://www.youtube.com/watch?v=K2UD7H8m13w



10.2 Arrays

10.2.3 - Using a Linear Search:

Click the link below the watch the video for a linear search

https://www.youtube.com/watch?v=BdjaHVIvJGs

10.2 Arrays

10.2.3 - Using a Linear Search:

Click the link below the watch the video for a linear search

https://www.youtube.com/watch?v=UldZOLylez4

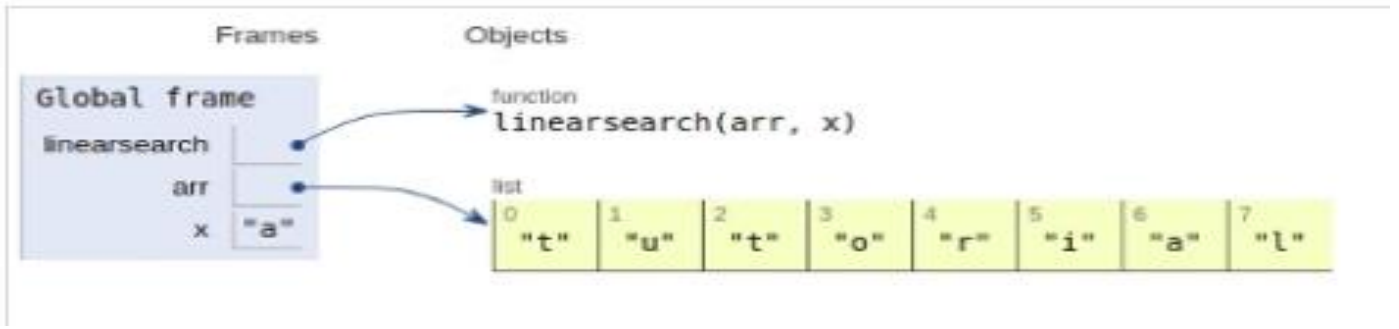
10.2 Arrays

10.2.3 - Using a Linear Search:

The linear search(**a.k.a sequential search**) algorithm is a simple search algorithm that starts at the left hand side of an array (index 0) and moves through the array one item at a time. Once the item being searched for is found the algorithm returns the index of the item in question. If the algorithm reaches the end of the array without finding the item then it either returns an error or it returns a non-valid index depending on the implementation.

Linear search (aka sequential search) is the most basic kind of search algorithm. It involves checking each item of the list (beginning to end), until the desired item is found.

The scope of the variables are shown in the figure –



10.2 Arrays

10.2.3 - Using a Linear Search:

Pseudocode linear search algorithm example 1

```
Linear search

pointer=0 WHILE pointer<LengthOfList AND list[pointer]!=searchedFor
   Add one to the pointer
ENDWHILE
IF pointer>=LengthOfList THEN
   PRINT("Item is not in the list")
ELSE
   PRINT("Item is at location "+pointer)
ENDIF
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Pseudocode linear search algorithm example 2

Algorithm	Pseudocode
Linear Search (Array A, Value x)	Procedure linear_search (list, value)

• Step 1: Set i to 1
• Step 2: if i > n then go to step 7
• Step 3: if A[i] = x then go to step 6
• Step 4: Set i to i + 1
• Step 5: Go to Step 2
• Step 6: Print Element x Found at index i and go to step 8
• Step 7: Print element not found
• Step 8: Exit
	procedure linear_search (list, value)
	
	for each item in the list
	if match item == value
	return the item's location
	end if
	end for
	
	end procedure

10.2 Arrays

10.2.3 - Using a Linear Search:

pseudocode linear search algorithm and identifier table to find if an item is in the populated 1D array myList. Example 3

```
DECLARE myList : ARRAY[0:9] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE item : INTEGER
DECLARE found : BOOLEAN
upperBound ← 9
lowerBound ← 0
OUTPUT "Please enter item to be found"
INPUT item
found ← FALSE
index ← lowerBound
REPEAT
      IF item = myList[index]
         THEN
            found ← TRUE
      ENDIF
      index ← index + 1
UNTIL (found = TRUE) OR (index > upperBound)
IF found
   THEN
      OUTPUT "Item found"
   ELSE
      OUTPUT "Item not found"
ENDIF
```

10.2 Arrays

10.2.3 - Using a Linear Search:

The pseudocode linear search algorithm and identifier table to find if an item is in the populated 1D array myList

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
item	Item to be found
found	Flag to show when item has been found

This method works for a list in which the items can be stored in any order, but as the size of the list increases, the average time taken to retrieve an item increases correspondingly.

10.2 Arrays

10.2.3 - Using a Linear Search:

Open Python IDLE and copy the code below for more practical activities

Example 1:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py (3.7

File Edit Format Run Options Window Help

```
def linearsearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return i
    return -1
arr = ['N','O','U','R','E','D','D','I','N','E']
x = 'I'
print("element found at index "+str(linearsearch(arr,x)))
```

```
>>>
 RESTART: C:/Users/v900010
element found at index 7
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Open Python IDLE and copy the code below for more practical activities

Example 2:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/

File Edit Format Run Options Window Help

```
def LinearSearch(array, n, t):

    for j in range(0, n):

        if (array[j] == t):

            return j

    return -1


array = [1, 3, 5, 7, 9]

t = 7
n = len(array)

result = LinearSearch(array, n, t)

if(result == -1):

    print("Element not found")

else:

    print("Element found at index: ", result)
```

```
...
 RESTART: C:/Users/v9000101
Element found at index:  3
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Open Python IDLE and copy the code below for more practical activities

Example 3:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py (3.7.2)

File Edit Format Run Options Window Help

```
def linear_search(items, item_to_search):
  i = 0

  while i < len(items):
    if list_of_numbers[i] == item_to_search:
      return i
    i = i + 1
  return -1

list_of_numbers = [21, 34, 5, 47, 11]

number_to_search = int(input("Enter number you want to search:"))
search_result = linear_search(list_of_numbers, number_to_search)

if search_result == -1:
        print("Given number is not found in the list")
else:
        print("Given number is found in the list at position:", search_result + 1)
```

```
 RESTART: C:/Users/v90001010/AppData/Local/Progra
Enter number you want to search: 21
Given number is found in the list at position: 1
>>>
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Open Python IDLE and copy the code below for more practical activities

Example 4:

linear search.py - C:/Users/v90001010/AppData/Local/Programs/Pythc

File Edit Format Run Options Window Help

```
pos = -1

def search(list, n):
    i = 0

    while i< len(list):
        if list[i] == n:
            globals()['pos'] = i
            return True
        i = i + 1;

    return False

list = [5,8,4,6,9,2]
n = 9

if search(list, n):
    print("Found at ",pos+1)
else:
    print("Not Found")
```

```
= RESTART: C:/Users
rch.py
Found at  5
```

10.2 Arrays

10.2.3 - Using a Linear Search:

Open Python IDLE and copy the code below for more practical activities

Example 5:

```
#Python program for Linear Search
#create array to store all the numbers
myList = [4, 2, 8, 17, 9, 3, 7, 12, 34, 21]
#enter item to search for
item = int(input("Please enter item to be found "))
found = False
for index in range(len(myList)):
    if(myList[index] == item):
        found = True
if(found):
    print("Item found")
else:
    print("Item not found")
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

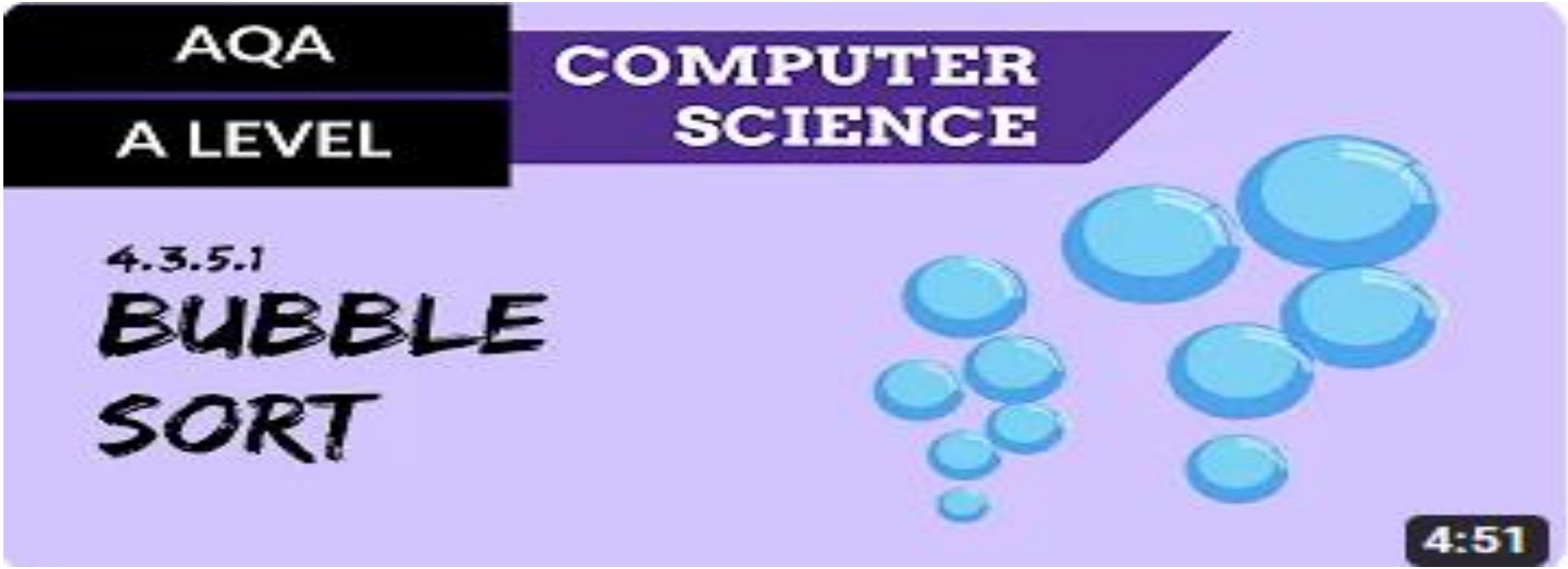
Click the link below the watch the video for Bubble Sort
https://www.youtube.com/watch?v=vy_domkFPxw

- **Recap IGCSE Binary Search**

10.2 Arrays

10.2.4 - Using a bubble Sort:

Click the link below the watch the video for Bubble Sort
https://www.youtube.com/watch?v=wIJ7_l2XrnE



10.2 Arrays

10.2.4 - Using a bubble Sort:

Click the link below the watch the video for Bubble Sort
https://www.youtube.com/watch?v=18OO361--1E

10.2 Arrays

10.2.4 - Using a bubble Sort:

Click the link below the watch the video for Bubble Sort
https://www.youtube.com/watch?v=ih-gRQYc_84

10.2 Arrays

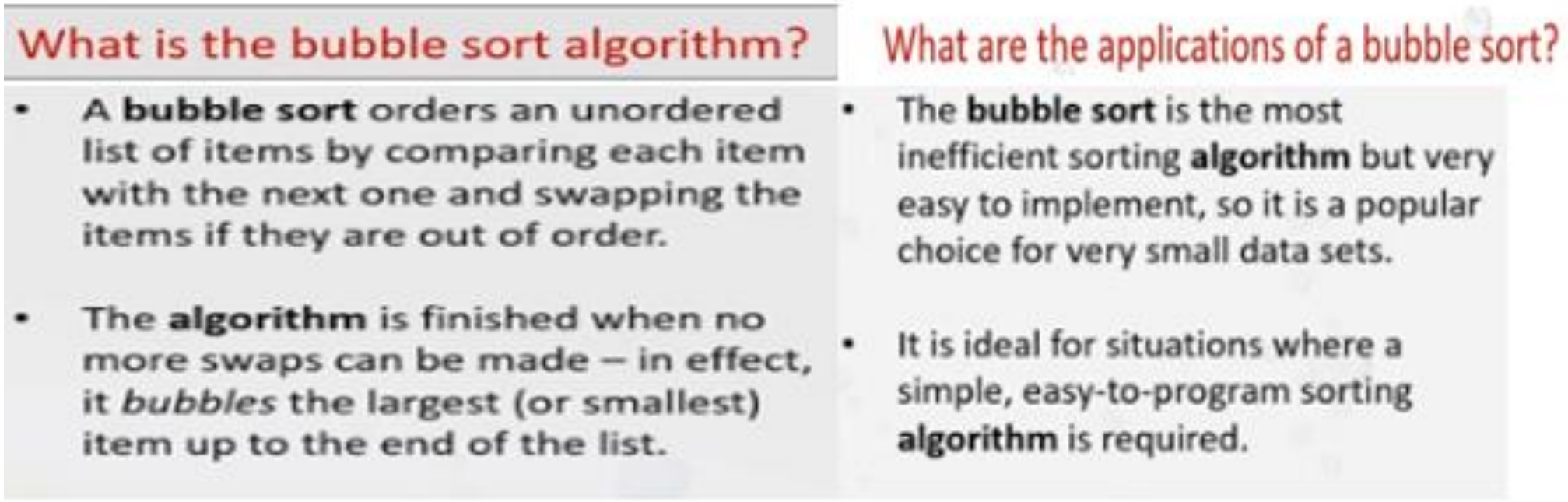
10.2.4 - Using a bubble Sort:

Click the link below the watch the video for Bubble Sort
https://www.youtube.com/watch?v=Vca808JTbI8

10.2 Arrays

10.2.4 - Using a bubble Sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high



10.2 Arrays

10.2.4 - Using a bubble Sort:

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n is the number of items.

Bubble Sort – $O(n^2)$

Bubble sort is a very easy to code Sorting Algorithm that sorts items in an array into some order.

It could be used for :

Numerical Order

Alphabetical Order

Time Order

Although it is simple to code it is incredibly slow, especially for larger arrays, and so is hardly ever used. You can tell it is really efficient by looking at the n squared part of the Big 0 value above, it shows an exponential relationship!!!

10 items – $n^2 = 100$

100 items – $n^2 = 10{,}000$

1000 items – $n^2 = 1{,}000{,}000$

10.2 Arrays

10.2.4 - Using a bubble Sort:

How does Bubble Sort work?

AS & A Level – You are required to know how it works and be able to write Code / Pseudocode for the algorithm.

The bubble sort algorithm works by sorting through the array in pairs:

1. It starts at the left of the array and inspects the first two items.
2. If the items are in the wrong order they are swapped around.
3. The algorithm then carries on with the next pair along and so forth.
4. When it reaches the end of the array it goes back to the beginning of the array and starts again.
5. If it completes a full pass of the array without swapping any items it knows that the array is sorted.

10.2 Arrays

10.2.4 - Using a bubble Sort:

How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.

The new array should look like this –

10.2 Arrays

10.2.4 - Using a bubble Sort:

How Bubble Sort Works?

Next we compare 33 and 35. We find that both are in already sorted positions.

Then we move to the next two values, 35 and 10.

We know then that 10 is smaller 35. Hence they are not sorted.

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this −

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this −

10.2 Arrays

10.2.4 - Using a bubble Sort:

How Bubble Sort Works?

Notice that after each iteration, at least one value moves at the end.

And when there's no swap required, bubble sorts learns that an array is completely sorted.

Now we should look into some practical aspects of bubble sort.

Algorithm

We assume **list** is an array of **n** elements. We further assume that **swap** function swaps the values of the given array elements

```
begin BubbleSort(list)

   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for

   return list

end BubbleSort
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Pseudocode example 1:

```
DECLARE myList : ARRAY[0:8] OF INTEGER
DECLARE upperBound : INTEGER
DECLARE lowerBound : INTEGER
DECLARE index : INTEGER
DECLARE swap : BOOLEAN
DECLARE temp : INTEGER
DECLARE top : INTEGER
upperBound ← 8
lowerBound ← 0
top ← upperBound

REPEAT
    FOR index = lowerBound TO top - 1
        Swap ← FALSE
        IF myList[index] > myList[index + 1]
          THEN
            temp ← myList[index]
            myList[index] ← myList[index + 1]
            myList[index + 1] ← temp
            swap ← TRUE
        ENDIF
    NEXT
    top ← top -1
UNTIL (NOT swap) OR (top = 0)
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Pseudocode example 2:

```
procedure bubbleSort( list : array of items )

    loop = list.count;

    for i = 0 to loop-1 do:
        swapped = false

        for j = 0 to loop-1 do:

            /* compare the adjacent elements */
            if list[j] > list[j+1] then
                /* swap them */
                swap( list[j], list[j+1] )
                swapped = true
            end if

        end for

        /*if no number was swapped that means
        array is sorted now, break the loop.*/

        if(not swapped) then
            break
        end if

    end for

end procedure return list
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Pseudocode example 3:

```
Set swapMade to TRUE
WHILE swapMade is TRUE
    Set swapMade to FALSE
    Start at position 0
    FOR position=0 TO listlength-2    (i.e. the last but one position)
        Compare the item at the position you are at…
        …with the one ahead of it
        IF they are out of order THEN
            Swap items and set swapMade to TRUE
        ENDIF
    NEXT position
ENDWHILE
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

The identifier table for the bubble sort algorithm.

Identifier	Description
myList	Array to be searched
upperBound	Upper bound of the array
lowerBound	Lower bound of the array
index	Pointer to current array element
swap	Flag to show when swaps have been made
top	Index of last element to compare
temp	Temporary storage location during swap

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 1:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py

File Edit Format Run Options Window Help

```
import time,random

l = [5,7,1,4,6,2,3,5,44]

def bubble_sort(l):
    while True:
        swapped = False
        for index in range(len(l)-1):
            if l[index] > l[index+1]:
                l[index], l[index+1] = l[index+1], l[index]
                swapped = True

        if swapped == False:
            return

bubble_sort(l)

print(l)
```

```
 RESTART: C:/Users/v90001010/A
[1, 2, 3, 4, 5, 5, 6, 7, 44]
>>>
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 2:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/area.py (3.7.2)

File Edit Format Run Options Window Help

```
def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements ( Traversing just means to process every character in a string, usually from left end to right end)
    for i in range(n-1):
    # range(n) also work but outer loop will repeat one time more than needed.

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j + 1] :
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("% d" % arr[i]),
```

```
 RESTART: C:/Users/v9000101(
Sorted array is:
 11
 12
 22
 25
 34
 64
 90
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 3:

area.py - C:/Users/v90001010/AppData/Local/Programs/Python/Py

File Edit Format Run Options Window Help

```
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp

alist = [54,26,93,17,77,31,44,55,20]
bubbleSort(alist)
print(alist)
```

```
>>>
 RESTART: C:/Users/v90001010/AppData/L
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 4:

area.py - C:/Users/v90001010/AppData/Local/Program

File Edit Format Run Options Window Help

```
def shortBubbleSort(alist):
    exchanges = True
    passnum = len(alist)-1
    while passnum > 0 and exchanges:
        exchanges = False
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                exchanges = True
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
        passnum = passnum-1

alist=[20,30,40,90,50,60,70,80,100,110]
shortBubbleSort(alist)
print(alist)
```

```
>>>
 RESTART: C:/Users/v90001010/AppData/Local/Pr
[20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 5:

bubble sorting.py - C:/Users/v90001010/AppData/Local/Progr

File Edit Format Run Options Window Help

```
def sort(nums):

    for i in range(len(nums)-1,0,-1):
        for j in range(i):
            if nums[j]>nums[j+1]:
                temp = nums[j]
                nums[j] = nums[j+1]
                nums[j+1] = temp


nums = [6, 7, 1, 4, 5, 2]
sort(nums)

print(nums)
```

```
= RESTART: C:/Users/v90001010/AppData/
ting.py
[1, 2, 4, 5, 6, 7]
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 6:

bubbleSort2.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python310/bubbleS..

File Edit Format Run Options Window Help

```
def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
if __name__ == "__main__":
  arr = [5, 1, 4, 2, 8]

  bubbleSort(arr)

  print("Sorted array is:")
  for i in range(len(arr)):
      print("%d" % arr[i], end=" ")
```

```
= RESTART: C:/Users/v90001010/
2.py
Sorted array is:
1 2 4 5 8
```

10.2 Arrays

10.2.4 - Using a bubble Sort:

Open Python IDLE and copy the code below for more practical activities

Example 7:

Pre-condition loop

```
#Python program for Bubble Sort
myList = [70,46,43,27,57,41,45,21,14]
top = len(myList)
swap = True
while (swap) or (top > 0):
        swap = False
        for index in range(top - 1):
                if myList[index] > myList[index + 1]:
                        temp = myList[index]
                        myList[index] = myList[index + 1]
                        myList[index + 1] = temp
                        swap = True
        top = top - 1
#output the sorted array
print(myList)
```

Learning Objectives

10.3 Files

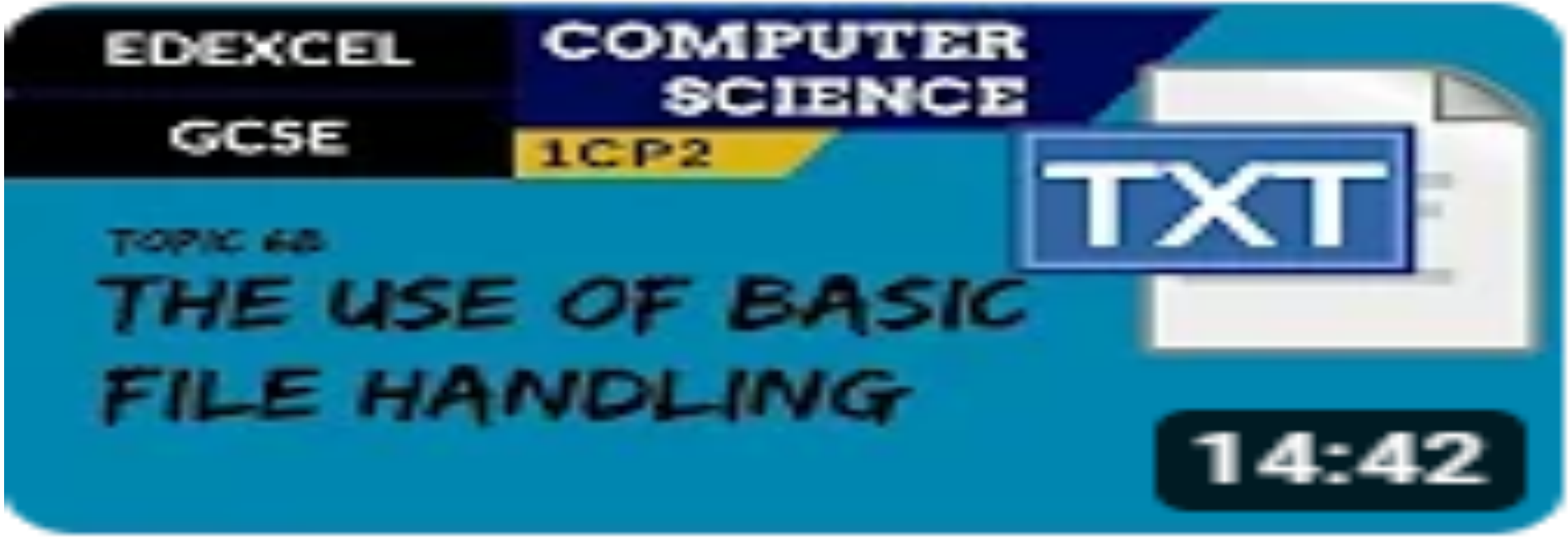
Candidates should be able to:	Notes and guidance
Show understanding of why files are needed	
Write pseudocode to handle text files that consist of one or more lines	

10.3 Files

Click the link below the watch the video for The use of basic file handling operations

https://www.youtube.com/watch?v=BU2Rr-BqlC0

Recap IGCSE:



10.3 Files

Click the link below the watch the video for File handling

https://www.youtube.com/watch?v=HQ--D7GTsbs

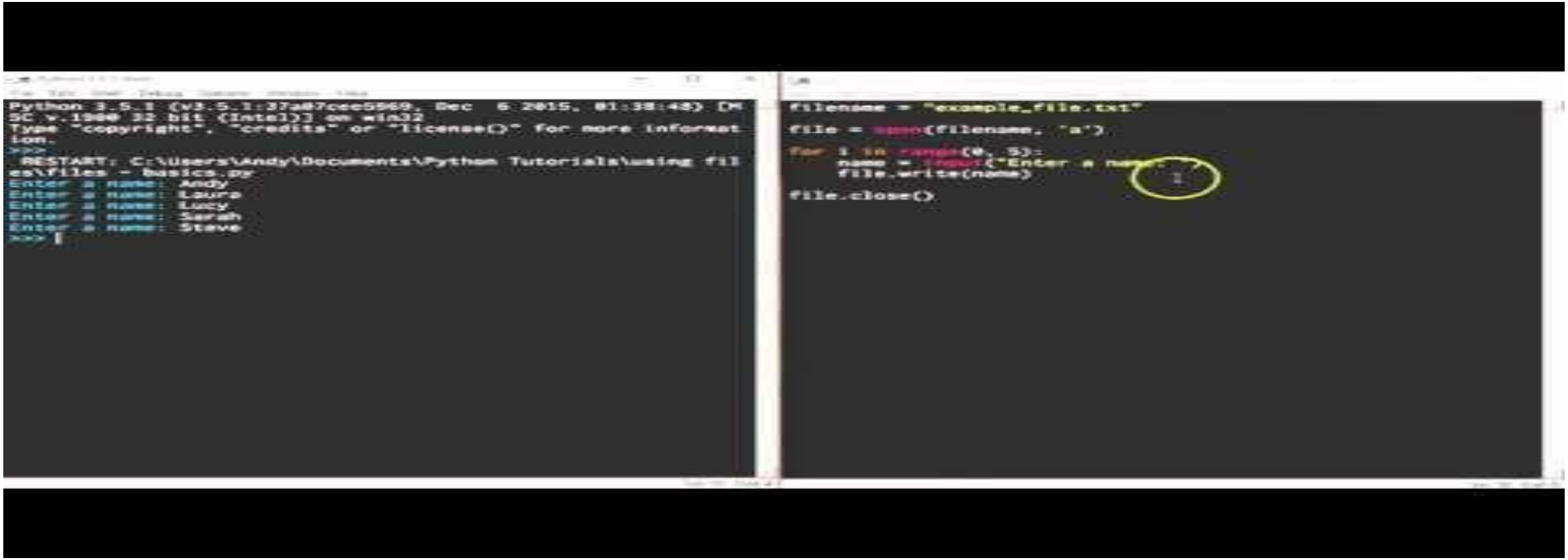
10.3 Files

Click the link below the watch the video for Reading and Writing to files in Python

https://www.youtube.com/watch?v=NyiBmf4tiBM

10.3 Files

Click the link below the watch the video for Appending data to a file in Python [Part 2]
https://www.youtube.com/watch?v=Y_6u0_8y3Ag



10.3 Files

Every program you write its need to be stored therefore you need a file and every file is identified by the filename also we are going to look how to use text files.

- **Text file** contain a sequence of characters.
- **Text file** can include an end of liker character that enable the file to be read from and written to as lines of characters

In pseudocode, text file are handled using the following statements:

To **open** a **file** **before** reading from it or **writing** to it :

OPEN <file identifier> FOR <file mode>

File can be opened in one of the following modes:

- **READ**: reads data from the list
- **WRITE**: writes data to the file, any existing data stored in the file will be overwritten
- **APPEND:** adds data to the end of the file

10.3 Files

In pseudocode, text file are handled using the following statements:

- Once the file is opened in READ mode, it can be read from a line at a time:

```
READFILE <file identifier>, <variable>
```

- Once the file is opened in **WRITE** or **APPEND** mode, it can be **written** to a line at a time:

```
WRITEFILE <file identifier>, <variable>
```

In both cases, the variable must be of data type **STRING**.

- The **function EOF** is used to test for the end of a file. It returns a value TRUE if the end of a file has been reached and FALSE otherwise.

```
EOF(<file identifier>)
```

- When a file is no longer being used it should be closed:

```
CLOSEFILE <file identifier>
```

10.3 Files

This pseudocode shows how the file

myText.txt could be written to and read from:

Identifier name	Description
textLn	Line of text
myFile	File name

```
DECLARE textLn : STRING
DECLARE myFile : STRING
myFile ← "myText.txt"
OPEN myFile FOR WRITE
REPEAT
    OUTPUT "Please enter a line of text"
    INPUT textLn
    IF textLn <> ""
      THEN
        WRITEFILE, textLn
      ELSE
        CLOSEFILE(myFile)
    ENDIF

UNTIL textLn = ""
OUTPUT "The file contains these lines of text:"
OPEN myFile FOR READ
REPEAT
    READFILE, textLn
    OUTPUT textLn
UNTIL EOF(myFile)
CLOSEFILE(myFile)
```

10.3 Files

File Opening Modes

- 'r' – Read Mode: This is the default mode for ***open()***. The file is opened and a pointer is positioned at the beginning of the file's content.
- 'w' – Write Mode: Using this mode will overwrite any existing content in a file. If the given file does not exist, a new one will be created.
- 'r+' – Read/Write Mode: Use this mode if you need to simultaneously read and write to a file.
- 'a' – Append Mode: With this mode the user can append the data without overwriting any already existing data in the file.
- 'a+' – Append and Read Mode: In this mode you can read and append the data without overwriting the original file.
- 'x' – Exclusive Creating Mode: This mode is for the sole purpose of creating new files. Use this mode if you know the file to be written doesn't exist beforehand.

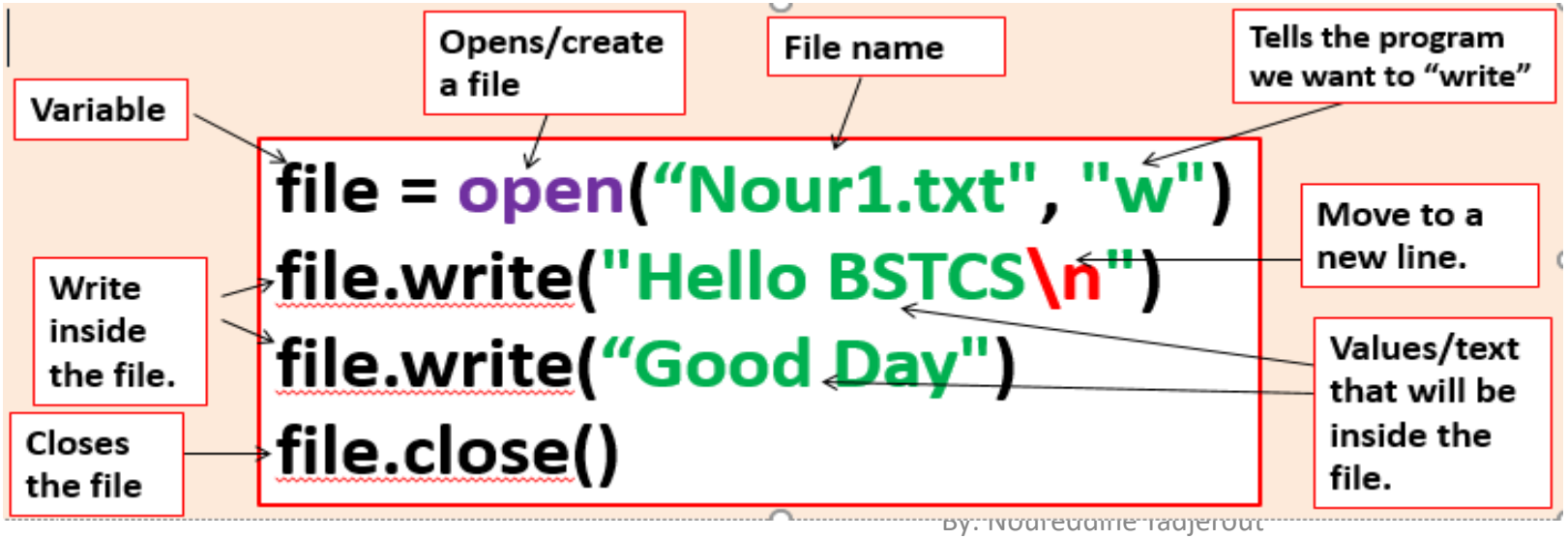
10.3 Files

Data files mean:

- **permanent** storage of data
- data can be **passed** to other programs
- data can be **sent** to other programmers/users
- data can be **accessed** as many times as needed without re-keying values

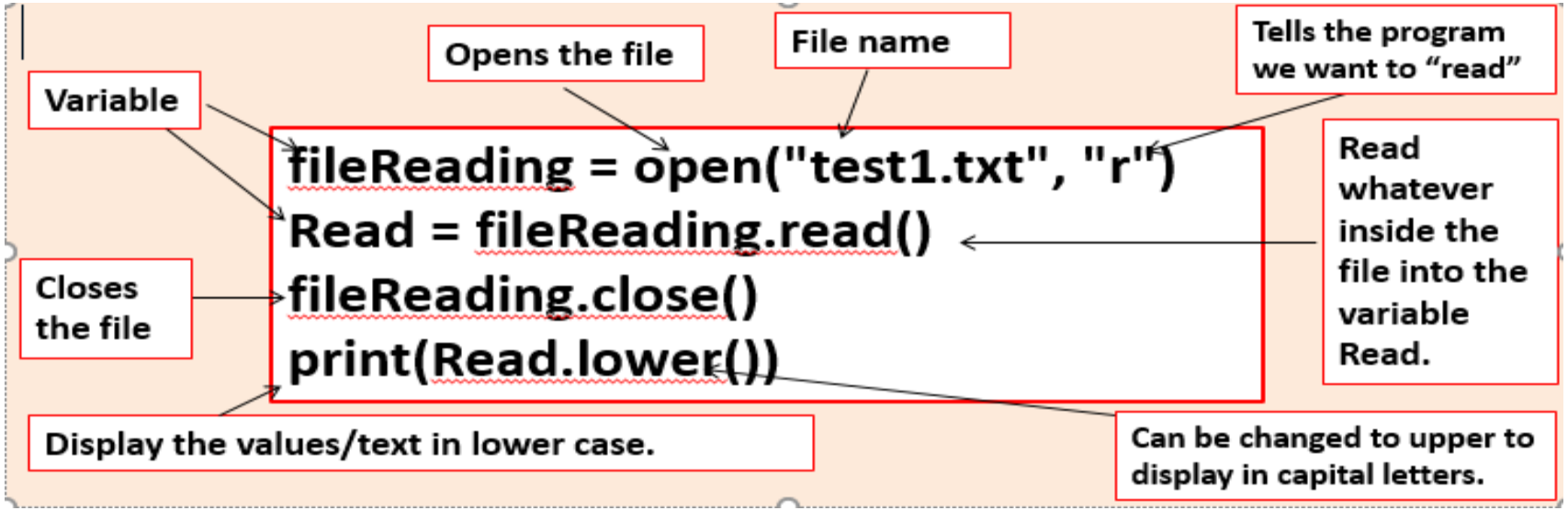
Text file: a sequence of lines, each of which consists of a sequence of characters.

Demonstrate Writing data to a file.



10.3 Files

Demonstrate Reading data to a file.



To get keyboard **input**, a process can use **ReadFile** or ReadConsole with a handle to the console's input buffer, or it can use **ReadFile** to read **input** from a file

10.3 Files

Open Python IDLE and copy the code below for more practical activities

Example 1:

area.py - C:/Users/v90001010/AppData/Local/Programs/

File Edit Format Run Options Window Help

```
#Demonstrate writing data to a file

file = open("Nour1.txt", "w")
file.write("Hello BSTCS\n")
file.write("Good Day")
file.close()


#Demonstrate reading data from a file.

fileReading = open("Nour1.txt", "r")
Read = fileReading.read()
fileReading.close()
print(Read.lower())
```

```
 RESTART: C:/Users/v90001010
hello bstcs
good day
```

Example 2:

File Edit Format Run Options Window Help

```
#Demonstrate writing data to a file

file = open("Nour1.txt", "w")
file.write("Hello AS/A Level CS Students\n")
file.write("I wish you success in your CS Exam")
file.close()


#Demonstrate reading data from a file.

fileReading = open("Nour1.txt", "r")
Read = fileReading.read()
fileReading.close()
print(Read.upper())
```

```
>>>
 RESTART: C:/Users/v90001010/AppData/Loc
HELLO AS/A LEVEL CS STUDENTS
I WISH YOU SUCCESS IN YOUR CS EXAM
```

10.3 Files

Open Python IDLE and copy the code below for more practical activities

Example 3:

area.py - C:/Users/v90001010/AppData/Local/Progr

File Edit Format Run Options Window Help

```
f = open("Nour1.txt", "r")

print(f.read())
```

```
 RESTART: C:/Users/v90001010/
Hello BSTCS.txt
```

Example 4:

area.py - C:/Users/v90001010/AppData/Local/Programs/Py

File Edit Format Run Options Window Help

```
f = open("demofile.txt", "r")

print(f.read())
```

```
>>>
 RESTART: C:/Users/v90001010/AppData/L
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

10.3 Files

Activity 1: Demonstrate writing data to a file.

```
file = open("test1.txt", "w")
file.write("Hello World\n")
file.write("Good Day")
file.close()
```

Extension:

1- Create a new file called "games.txt".
2- Write your top 3 favourite games on 1 line.
3- Close the file.
4- Comment on the code.

Task1:

1- Create a new file called "test1.txt".
2- Save the file on the python folder as test11.txt
3- Open IDLE and enter the code above
4- Comment on the code.

A process can use **WriteFile** or WriteConsole to write to either an active or inactive screen buffer, or it can use WriteFile to write to a file has been redirected. Processed **output** mode and **output** mode control the way characters are written or echoed to a screen buffer.

10.3 Files

Activity 2: Demonstrate reading data from a file.

```
fileReading = open("test1.txt", "r")
Read = fileReading.read()
fileReading.close()
print(Read.upper())
```

```
fileReading = open("test1.txt", "r")
Read = fileReading.read()
fileReading.close()
print(Read.lower())
```

Task2:

1- Create a file and store your name and surname inside it.

2- Display the content of the file in both uppercase and lowercase.

3- Comment on the code.

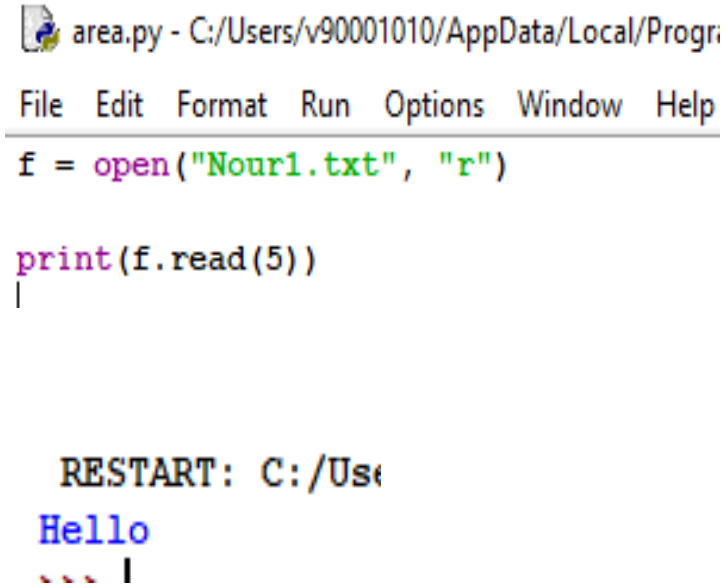
10.3 Files

Read Only Parts of the File:

By default the **read()** method returns the whole text, but you can also specify how many characters you want to return:

Example:

Return the 5 first characters of the file:



10.3 Files

Read Lines:

You can return one line by using the **readline()** method:

area.py - C:/Users/v90001010/AppData/Local/Progra

File Edit Format Run Options Window Help

```
f = open("Nour1.txt", "r")

print(f.readline())
```

```
>>>
 RESTART: C:/Users/v90001010/AppData
Hello AS/A Level CS Students
```

By calling **readline()** two times, you can read the two first lines:

File Edit Format Run Options Window

```
f = open("Nour1.txt", "r")

print(f.readline())
print(f.readline())
```

```
>>>
 RESTART: C:/Users/v90001010/AppData
Hello AS/A Level CS Students

I wish you success in your CS Exam
>>>
```

10.3 Files

By looping through the lines of the file, you can read the whole file, line by line:

File Edit Format Run Options Window

```
f = open("Nour1.txt", "r")
for x in f:
  print(x)
```

```
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Pyt
Hello AS/A Level CS Students

I wish you success in your CS Exam

Please make sure you do all your past exams. Thank you
```

Close Files:

Close the file when you are finish with it:

File Edit Format Run Options

```
f = open("Nour1.txt", "r")

print(f.readline())

f.close()
```

10.3 Files

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

- "**a**" - Append - will append to the end of the file
- "**w**" - Write - will overwrite any existing content

```
File  Edit  Format  Run  Options  Window  Help
f = open("Nour1.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:

f = open("Nour1.txt", "r")
print(f.read())
```

```
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Python/Pyt
Hello AS/A Level CS Students
I wish you success in your CS Exam
Please make sure you do all your past exams. Thank you
Now the file has more content!Now the file has more content!
```

10.3 Files

Example

Open the file “Nour1.txt" and overwrite the content:

```
File  Edit  Format  Run  Options  Window  Help
f = open("Nour1.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:

f = open("Nour1.txt", "r")
print(f.read())
```

```
''' 
 RESTART: C:/Users/v90001010/AppData/l
Woops! I have deleted the content!
```

Note: the "w" method will overwrite the entire file.

10.3 Files

How to append text or lines to a file in python?

To append some text to a file in the end, we first need to open the file with access mode ‘a’,

file_object = open('Nour1.txt', 'a')

- With file access mode **‘a’**, open() function first checks if file exists or not. If the file doesn’t exist, then it creates an empty file and opens it. Whereas, if the file already exists then it opens it. In both cases, it returns a file object, and it has write cursor, which points to the end of the opened file. Now, if you write anything to the file using this file object, then it will be appended to the end.
- Let’s use this to append text at the end of a file,

Now let’s append text ‘**hello**’ at the end of this file,

File Edit Format Run Options Window Help

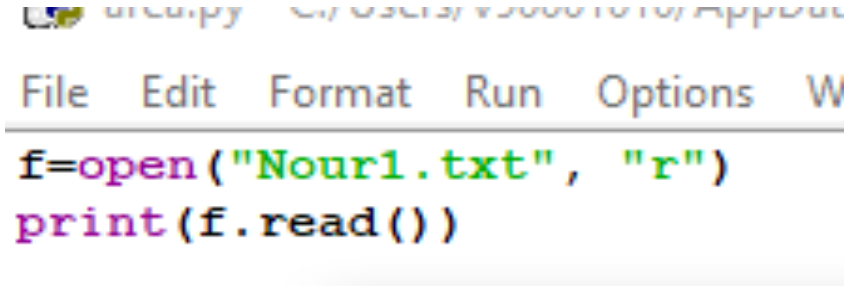
```
# Open a file with access mode 'a'
file_object = open('Nour1.txt', 'a')
# Append 'hello' at the end of file
file_object.write('hello')
# Close the file
file_object.close()
```

10.3 Files

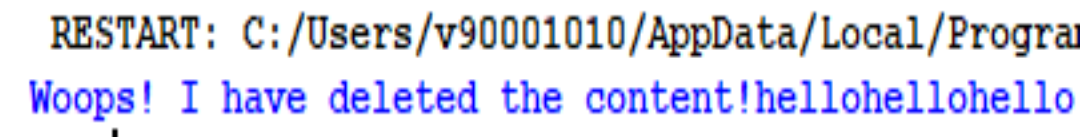
How to append text or lines to a file in python?

We opened the file '**Nour1.txt**' in append mode i.e. using access mode **'a'**. As cursor was pointing to the end of the file in the file object, therefore when we passed the string in write() function, it appended it at the end of the file. So, our text '**hello**' gets added at the end of the file '**Nour1.tx**t'.

Let's open the file Nour1.txt to see the append



Contents of the file 'Nour1.txt' will be now,



10.3 Files

AS/A level past paper 2 Exam Question 1:

(a) A concert venue uses a program to calculate admission prices and store information about ticket sales.

A number of arrays are used to store data. The computer is switched off overnight and data has to be input again at the start of each day before any tickets can be sold. This process is very time consuming.

(i) Explain how the program could use text files to speed up the process.

.....

.....

.....

.....

.....

..... [2]

10.3 Files

AS/A level past paper 2 Exam Question 1:

(ii) State the characteristic of text files that allow them to be used as explained in **part (a)(i)**.

.....

..... [1]

(iii) Information about ticket sales will be stored as a booking. The booking requires the following data:

- name of person booking
- number of people in the group (for example a family ticket or a school party)
- event type.

Suggest how data relating to each booking may be stored in a text file.

.....

.....

.....

..... [2]

10.3 Files

AS/A level past paper 2 Exam Answer 1:

Question	Answer	Marks
3(a)(i)	One mark per bullet point: • Data from the arrays is written to the files at the end of the day / before the program is terminated / computer is switched off • Data can then be read from the files at the start of the next day and written to / stored in the arrays • No need to (re-)enter the data manually // only need to enter data once **Note: Max 2 marks**	2
3(a)(ii)	• The data is retained when the program is terminated / after the computer is switched off // data is stored permanently // non-volatile storage	1
3(a)(iii)	One mark per bullet point: • Data items are combined to form a single string / saved as a single line in the file • Data items are separated by a special character // make each data item a fixed length ALTERNATIVE: • Convert all data items / 'number of people' to strings • Consecutive / each line stores a separate data item	2

10.3 Files

AS/A level past paper 2 Exam Question with Answer 2:

(ii) Simple algorithms usually consist of input, process and output.

Complete the table by placing ticks (‘✓’) in the relevant boxes.

Pseudocode statement	Input	Process	Output
`Temp ← SensorValue * Factor`		✓	
`WRITEFILE "LogFile.txt", TextLine`			✓
`WRITEFILE "LogFile.txt", MyName & MyIDNumber`		✓	✓
`READFILE "AddressBook.txt", NextLine`	✓	(✓)	

Learning Objectives

10.4 Introduction to Abstract Data Types (ADT)

Candidates should be able to:	Notes and guidance
Show understanding that an ADT is a collection of data and a set of operations on those data	
Show understanding that a stack, queue and linked list are examples of ADTs	Describe the key features of a stack, queue and linked list and justify their use for a given situation
Use a stack, queue and linked list to store data	Candidates will not be required to write pseudocode for these structures, but they should be able to add, edit and delete data from these structures
Describe how a queue, stack and linked list can be implemented using arrays	

10.4 Abstract Data Types (ADTs)

Note for AS/A Level student:

You need to be able to:

- Describe the key features of a stack, queue and linked list and justify the use for stack, queue and linked list.
- You will not be required to write pseudocode for these structures, but you should be able to **add, edit and delete** data from these structures.

10.4 Abstract Data Types (ADTs)

ADT is a collection of data and a set of operation on that data. A stack is a linear data structure that stores items in a **Last-In/First-Out (LIFO**) or **First-In/Last-Out (FILO)** manner. In stack, a new element is **added** at one end and an element is **removed** from that end only. The insert and delete operations are often called **push** (**Add an items to the stack**) and **pop** (**Remove an item or element from the stack**)

Click the link below the watch the video for Abstract Data Types vs. Data Structures

https://www.youtube.com/watch?v=nDeIz2Kq0RE

10.4 Abstract Data Types (ADTs)

Click the link below the watch the video: What are Abstract Data Types?

https://www.youtube.com/watch?v=XkoeF-xXo2o

4 Abstract Data Types (ADTs)

Stack and Queue - Part 1

https://www.youtube.com/watch?v=_7_jYMk_R9k

4 Abstract Data Types (ADTs)

Stack and Queue - Part 2

https://www.youtube.com/watch?v=0LepNscvzIY&t=40s

4 Abstract Data Types (ADTs)

Stack and Queue - Part

https://www.youtube.com/watch?v=WnjVVJ0kIgQ

4 Abstract Data Types (ADTs)

19.1.3 Understanding and using abstract data types (ADTs)

Stack and Queue - Part 4

https://www.youtube.com/watch?v=1hEK9yWs_cM

10.4 Abstract Data Types (ADTs)

In this Chapter we are going to look at three ADTs stack below:

- **Stack:** a list containing several elements or items operation on the **last in , first out(LIFO)** principal. Element can be added called (**push**) and element removed called (**pop**).

- **Queue:** a list containing several elements or items operating on the **first in , first out (FIFO)** principal , items can be added to the queue (**enqueue**) and removed from the queue (**dequeuer**).

- **Linked List:** a list containing several elements or items in which **each items in the list points to the next item in the list**. In a linked list a **new item is always added to the start of the list**.

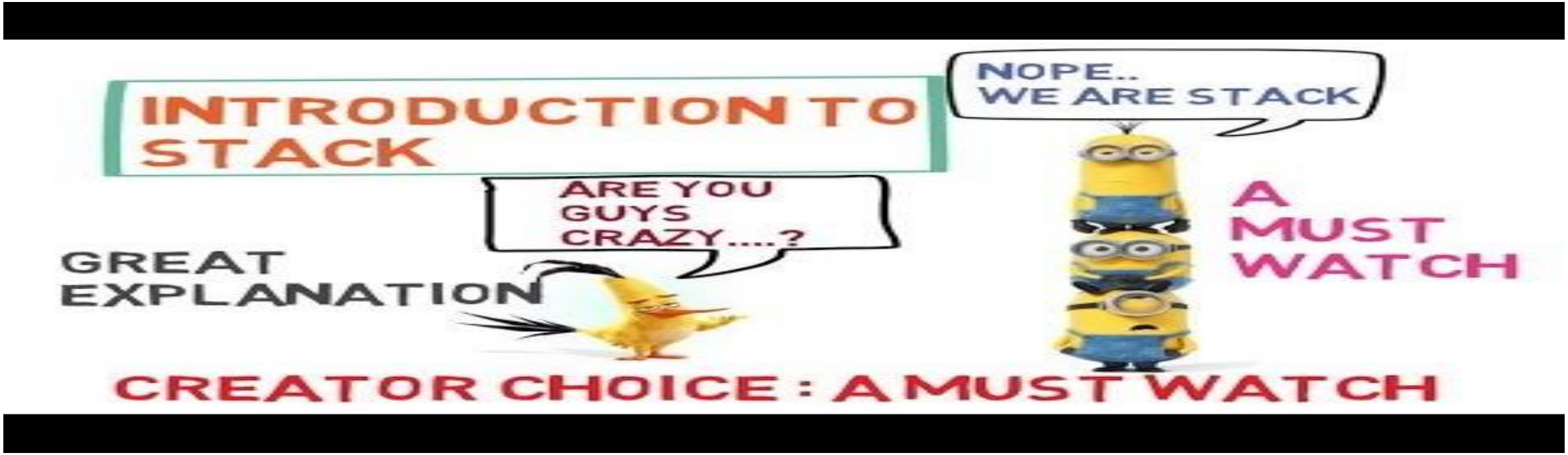
10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Stack: a list containing several elements or items operation on the **last in , first out(LIFO)** principal. Element can be added called (**push**) and element removed called (**pop**).

Click the link below the watch the video Stack operations:

https://www.youtube.com/watch?v=1SWr7q121gc

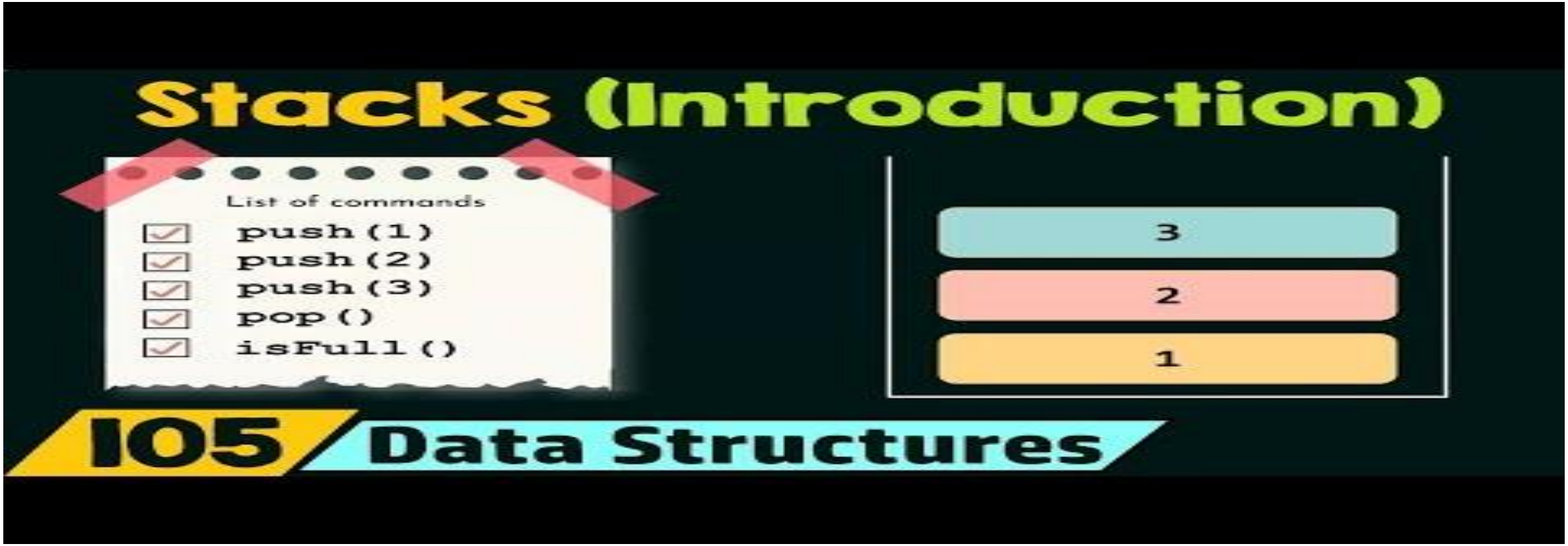


10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Click the link below the watch the video Stack operations:

https://www.youtube.com/watch?v=I37kGX-nZEI



10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the “top”. Stacks are ordered LIFO. The stack operations are given below.

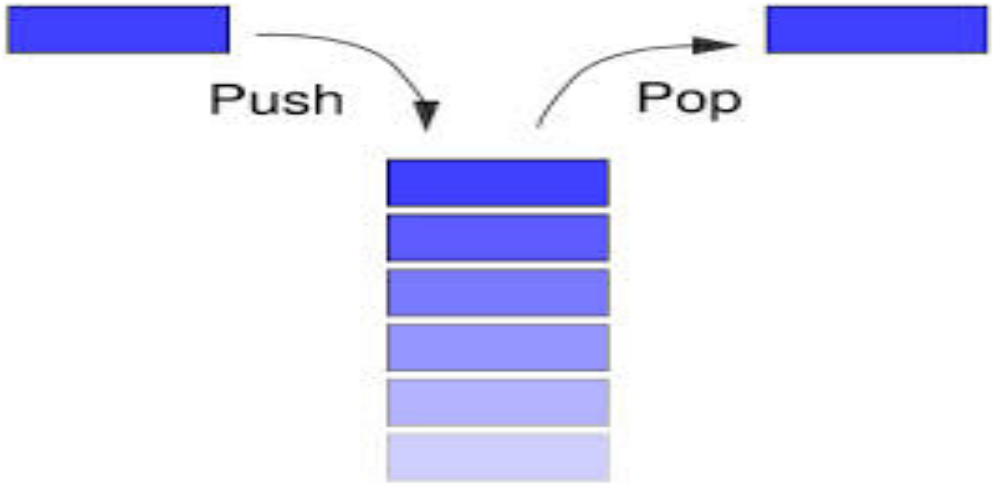
- **Stack():** creates a new stack that is empty. It needs no parameters and returns an empty stack.
- **push(item):** adds a new item to the top of the stack. It needs the item and returns nothing.
- **pop():** removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- **peek():** returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- **isEmpty():** tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- **size():** returns the number of items on the stack. It needs no parameters and returns an integer.

For example, if s is a stack that has been created and starts out empty, then it shows the results of a sequence of stack operations. Under stack contents, the top item is listed at the far right.

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

In the pushdown stacks only two operations are allowed: **push** the item into the stack, and **pop** the item out of the stack. A stack is a limited access data structure elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.



10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

The functions associated with stack are:

- **empty()**: Returns whether the stack is empty
- **size()**: Returns the size of the stack
- **top() / peek()**: Returns a reference to the topmost element of the stack
- **push(a)**: Inserts the element ‘a’ at the top of the stack
- **pop()**: Deletes the topmost element of the stack

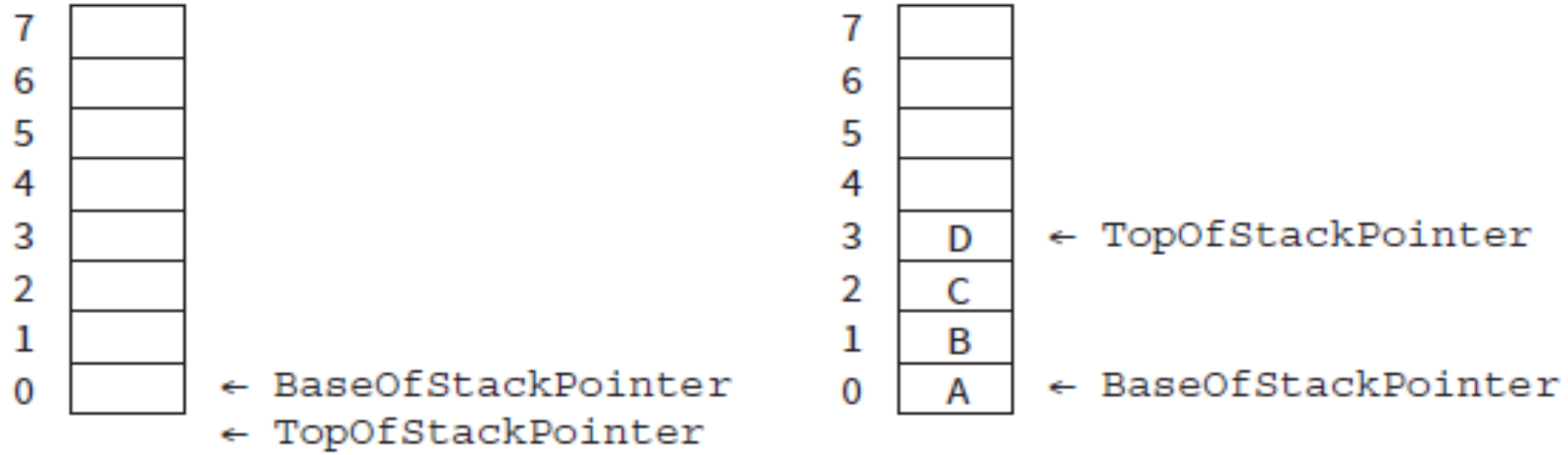
10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

We will show you, how we can represent a stack when we have added four items in this order: A, B, C, D. Note that the slots are shown numbered from the bottom as this feels more natural.

The **BaseOfStackPointer** will always point to the first slot in the stack.

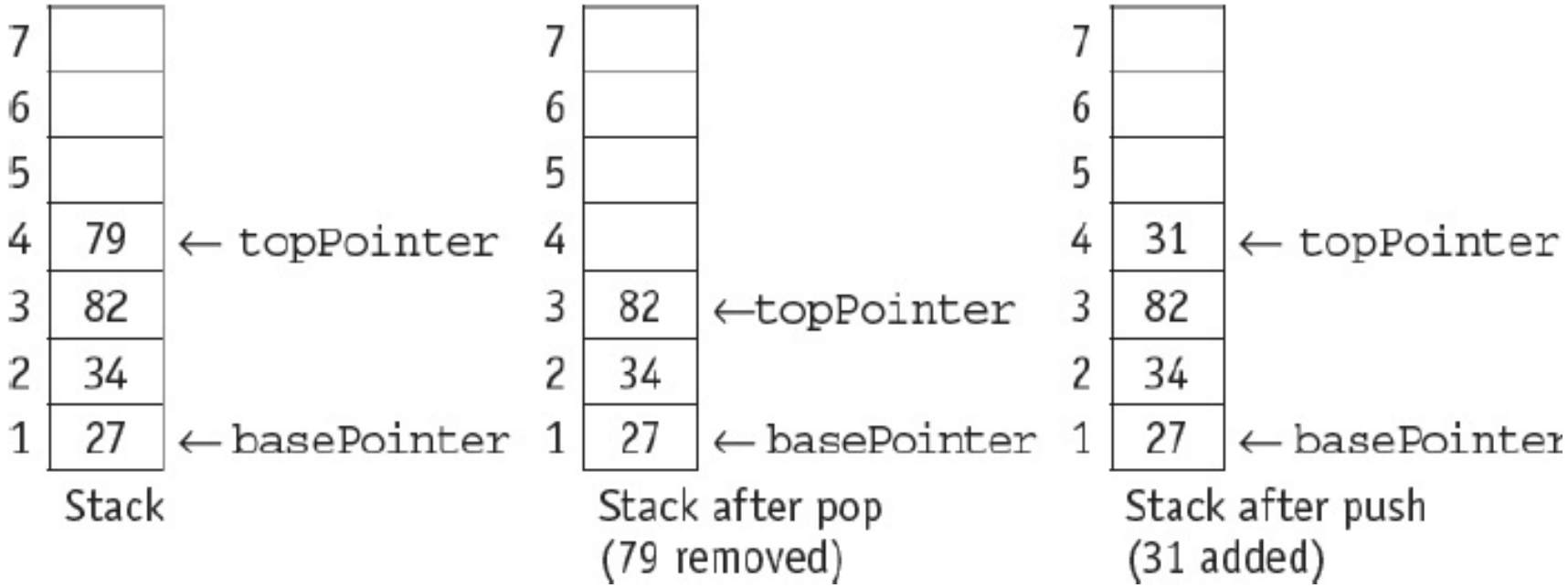
The **TopOfStackPointer** will point to the last element **pushed (added)** onto the stack. When an element is **popped (removed)** from the stack, the **TopOfStackPointer** will **decrease to point** to the element now at the top of the stack. When the stack is empty, **TopOfStackPointer** will have the value **–1**.



10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

The value of the **basePointer** always remains the same during stack operations:



A stack can be implemented using an array and a set of pointers. As an array has a finite size, the stack may become full and this condition must be allowed for.

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

In pseudocode, stack operations are handled using the following statements:

To set up a stack

```
DECLARE stack ARRAY[1:10] OF INTEGER
DECLARE topPointer : INTEGER
DECLARE basePointer : INTEGER
DECLARE stackful : INTEGER
basePointer ← 1
topPointer ← 0
stackful ← 10
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

In pseudocode, stack operations are handled using the following statements:

To push an item, stored in item, onto a stack

```
IF topPointer < stackful
  THEN
    topPointer ← topPointer + 1
    stack[topPointer] ← item
  ELSE
    OUTPUT "Stack is full, cannot push"
ENDIF
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

In pseudocode, stack operations are handled using the following statements:

To pop an item, stored in item, from the stack

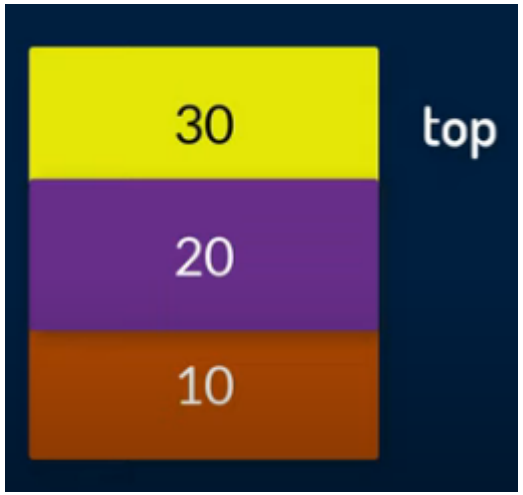
```
IF topPointer = basePointer - 1
  THEN
    OUTPUT "Stack is empty, cannot pop"
  ELSE
    Item ← stack[topPointer]
    topPointer ← topPointer - 1
ENDIF
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Open Python IDLE and copy the code below for more practical activity 1

```
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Python/Python37-32/q2.py
>>> stack =[]
>>> stack.append(10)
>>> stack.append(20)
>>> stack.append(30)
>>> stack
[10, 20, 30]
>>> stack.pop()
30
>>> stack.pop()
20
>>> stack.pop()
10
>>> len(stack)==0
True
>>> not stack
True
>>> stack.append(10)
>>> stack.append(20)
>>> stack[-1]
20
```



10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Open Python IDLE and copy the code below for more practical activity 2

Python Queue Methods.py - C:/Users/v90001010/AppDat

File Edit Format Run Options Window Help

```
class Stack :
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def is_empty(self):
        return (self.items == [])

s = Stack()
s.push(100)
s.push(50)
s.push("+")

while not s.is_empty():
    print( s.pop())
```

Python 3.7.2 Shell

File Edit Shell Debug Options Windo

```
Python 3.7.2 (tags/v3.7.2:9a3f
(Intel)] on win32
Type "help", "copyright", "cre
>>>
 RESTART: C:/Users/v90001010/A
eue Methods.py
+
50
100
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Open Python IDLE and copy the code below for more practical activity 3

*Stacks.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python310/Sta

File Edit Format Run Options Window Help

```
#In this example, you created an empty history object,
#and every time the user visited a new site, you added
#it to your history variable using appendleft().
#Doing so ensured that each new element was added
#to the head of the linked list.


from collections import deque
history = deque()

history.appendleft("https://www.epsomcollege.edu.my/")
history.appendleft("https://www.cam.ac.uk/")
history.appendleft("https://www.ox.ac.uk/")
history
deque(['https://www.epsomcollege.edu.my',
       'https://https://www.cam.ac.uk/',
       'https://https://www.ox.ac.uk/'])
```

IDLE Shell 3.10.8

File Edit Shell Debug Options Window Help

```
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11
AMD64)] on win32
Type "help", "copyright", "credits" or "lice
>>>
= RESTART: C:/Users/v90001010/AppData/Local/
>>> history.popleft()
'https://www.ox.ac.uk/'
>>> history.popleft()
'https://www.cam.ac.uk/'
>>> history
deque(['https://www.epsomcollege.edu.my/'])
```

There you go! Using popleft(), you removed elements from the head of the linked list until you reached the Epsom College home page

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

Open Python IDLE and copy the code below for more practical activity 4

stack.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python310/stack.py

File Edit Format Run Options Window Help

```
# Python program to
# demonstrate stack implementation
# using list

stack = []

# append() function to push
# element in the stack
stack.append('a')
stack.append('b')
stack.append('c')

print('Initial stack')
print(stack)

# pop() function to pop
# element from stack in
# LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)

# uncommenting print(stack.pop())
# will cause an IndexError
# as the stack is now empty
```

IDLE Shell 3.10.8

File Edit Shell Debug Options Window Help

```
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022, 16:50:30) [MSC
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
>>>
= RESTART: C:/Users/v90001010/AppData/Local/Programs/Python/Pyth
Initial stack
['a', 'b', 'c']

Elements popped from stack:
c
b
a

Stack after elements are popped:
[]
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

What is difference between self and __init__ methods in python Class?

- **self:**

The word 'self' is used to represent the instance of a class. By using the "self" keyword we access the attributes and methods of the class in python.

- **__init__ method:**

"__init__" is a reseved method in python classes. It is called as a constructor in object oriented terminology. This method is called when an object is created from a class and it allows the class to initialize the attributes of the class

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

What is difference between self and __init__ methods in python Class?

Example 2

```
File  Edit  Format  Run  Options  Window  Help
# A Sample class with init method
class BST:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

b = BST('Noureddine')
b.say_hi()
```

```
...
 RESTART: C:/Users/v90001010/A
Hello, my name is Noureddine
>>>
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

What is difference between self and __init__ methods in python Class?

Example 3:

File Edit Format Run Options Window Help

```
# A Sample class with init method
class BST:

    # init method or constructor
    def __init__(self, name):
        self.name = name

    # Sample Method
    def say_hi(self):
        print('Hello, my name is', self.name)

# Creating different objects
b1 = BST('Noureddine')
b2 = BST('Swaraj')
b3 = BST('Jane')

b1.say_hi()
b2.say_hi()
b3.say_hi()
```

File Edit Shell Debug Options Window Help

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:2
Type "help", "copyright", "credits" or "license()" for
>>>
 RESTART: C:/Users/v90001010/AppData/Local/Programs/Pyt
Hello, my name is Noureddine
Hello, my name is Swaraj
Hello, my name is Jane
>>>
```

10.4 Abstract Data Types (ADTs)

10.4.1 - Stack Operations:

What is difference between self and __init__ methods in python Class?

Example 5

- Find out the cost of a rectangular field with breadth(b=120), length(l=160). It costs x (2000) Sums per 1 square unit.

File Edit Format Run Options Window Help

```
class Rectangle:
    def __init__(self, length, breadth, unit_cost=0):
        self.length = length
        self.breadth = breadth
        self.unit_cost = unit_cost
    def get_area(self):
        return self.length * self.breadth
    def calculate_cost(self):
        area = self.get_area()
        return area * self.unit_cost
# breadth = 120 units, length = 160 units, 1 sq unit cost = Sum 2000
r = Rectangle(160, 120, 2000)
print("Area of Rectangle: %s sq units" % (r.get_area()))
```

```
...
 RESTART: C:/Users/v90001010/AppData/Local/Progra
Area of Rectangle: 19200 sq units
>>>
```

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Click the link below the watch the video for Queue operations:

https://www.youtube.com/watch?v=UpvDOm3prfI

10.4 Abstract Data Types (ADTs)

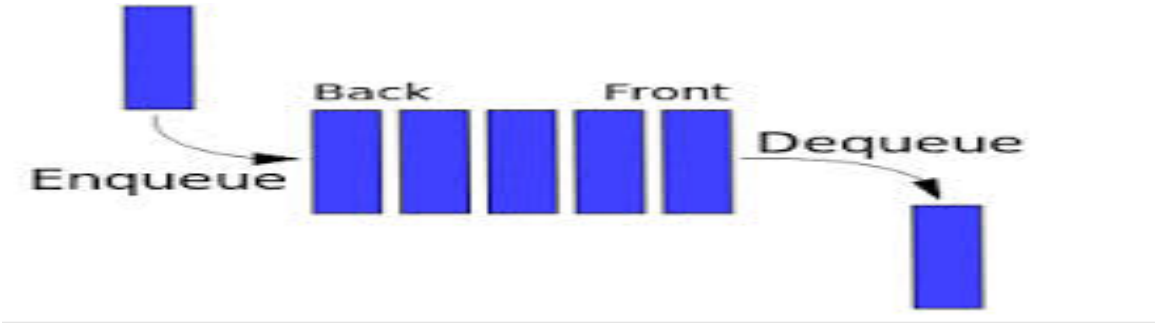
10.4.2 - Queue Operations:

An excellent example of a queue is a line of students in the food court of the Epsom College. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed **enqueue** and **dequeue.**

Enqueue means to insert an item into the back of the queue.

Dequeue means removing the front item.

The picture demonstrates the **FIFO** access. The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.



10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue:

FIFO & LILO and LIFO & FILO Principles

- **Queue: First In First Out (FIFO):** The first object into a queue is the first object to leave the queue, used by a queue.
- **Stack: Last In First Out (LIFO):** The last object into a stack is the first object to leave the stack, used by a stack

OR

- **Stack: First In Last Out (FILO):** The first object or item in a stack is the last object or item to leave the stack.
- **Queue: Last In Last Out (LILO):** The last object or item in a queue is the last object or item to leave the queue

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue:

Operations associated with queue are:

- **Enqueue:** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition
- **Dequeue:** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition
- **Front:** Get the front item from queue
- **Rear:** Get the last item from queue

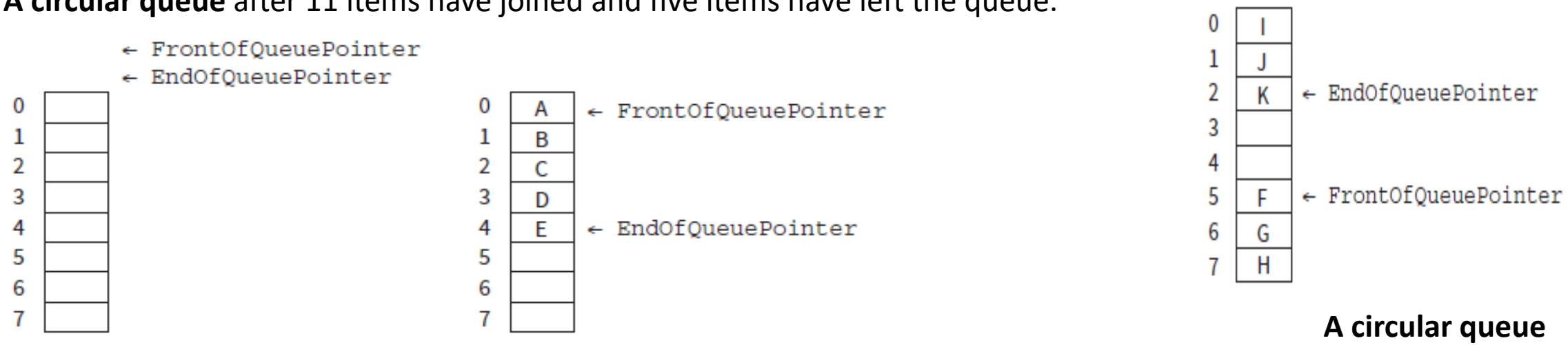
10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

When people form a queue, they join the queue at the end. People leave the queue from the front of the queue. If it is an orderly queue, no-one pushes in between and people don't leave the queue from any other position. In this example below we can shows how we can represent a queue when five items have joined the queue in this order: A, B, C, D, E.

To implement a queue using an array, we can assume that the front of the queue is at position **0**. When the queue is empty, the **EndOfQueuePointer** will have the value **–1**. When one value joins the queue, the **EndOfQueuePointer** will be incremented before adding the value to the array element where the pointer is pointing to. When the item at the front of the queue leaves, we need to move all the other items on slot forward and adjust **EndOfQueuePointe**r.

A circular queue after 11 items have joined and five items have left the queue.



A circular queue

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue: a list containing several elements or items operating on the **first in , first out (FIFO)** principal , items can be added to the queue (**enqueue**) and removed from the queue (**dequeuer**).

The value of the frontPointer changes after dequeue but the value of the rearPointer changes after enqueue:

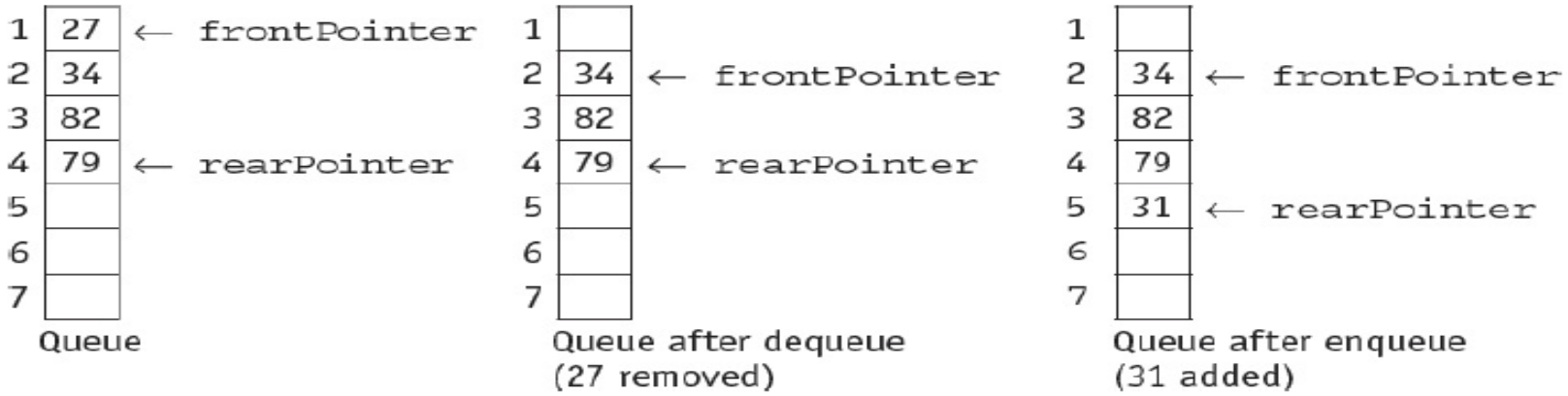


Figure 10.4.2.1 Circular queue operation

A queue can be implemented using an array and a set of pointers. As an array has a finite size, the queue may become full and this condition must be allowed for. Also, as items are removed from the front and added to the end of a queue, the position of the queue in the array changes. Therefore, the queue should be managed as a **circular queue** to avoid moving the position of the items in the array every time an item is removed.

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

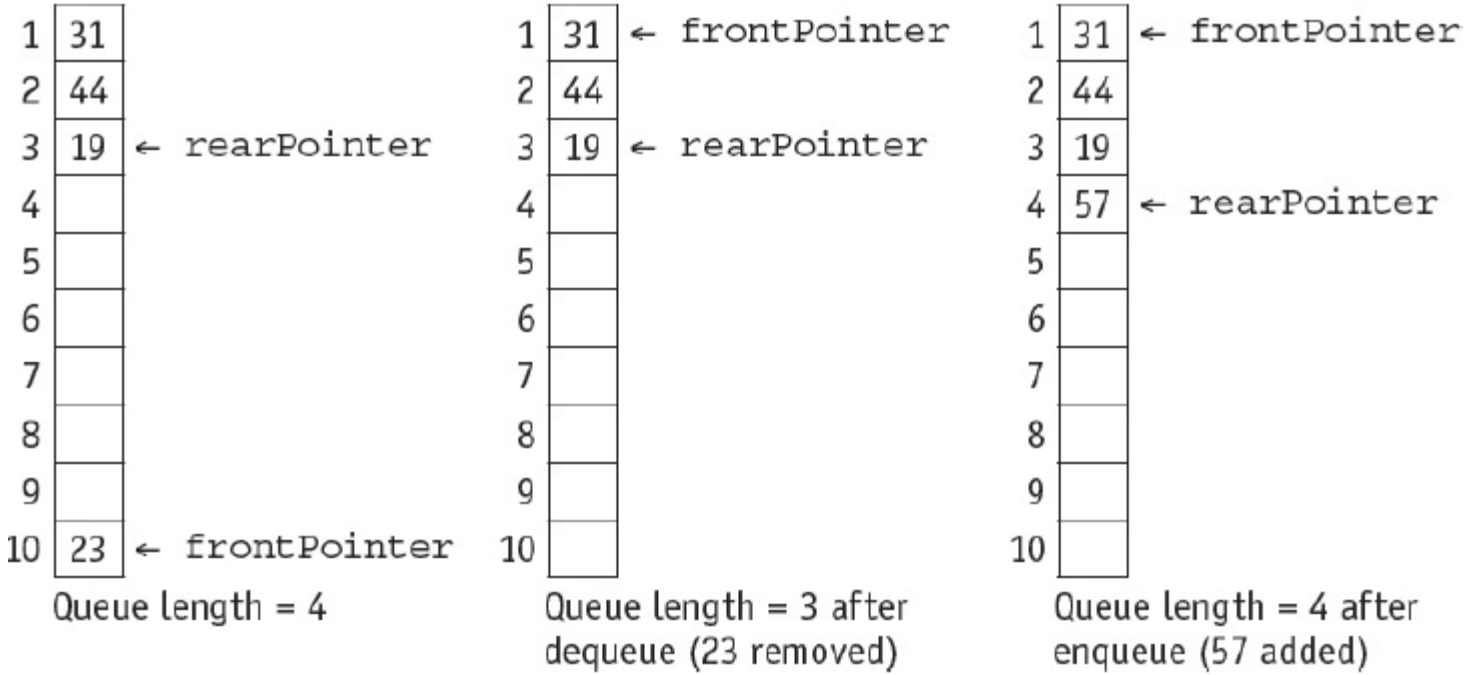


Figure 10.4.2.2 Circular queue operation

When a queue is implemented using an array with a finite number of elements, it is managed as a circular queue. Both pointers, **frontPointer** and **rearPointer**, are updated to point to the first element in the array (**lower bound**) after an operation where that pointer was originally pointing to the last element of the array (**upper bound**), providing the length of the queue does not exceed the size of the array.

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

In pseudocode, queue operations are handled using the following statements.

To set up a queue

```
DECLARE queue ARRAY[1:10] OF INTEGER
DECLARE rearPointer : INTEGER
DECLARE frontPointer : INTEGER
DECLARE queueful : INTEGER
DECLARE queueLength : INTEGER
frontPointer ← 1
endPointer ← 0
upperBound ← 10
queueful ← 10
queueLength ← 0
```

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

In pseudocode, queue operations are handled using the following statements.

To add an item, stored in item, onto a queue

```
IF queueLength < queueful
  THEN
    IF rearPointer < upperBound
      THEN
        rearPointer ← rearPointer + 1
      ELSE
        rearPointer ← 1
    ENDIF
    queueLength ← queueLength + 1
    queue[rearPointer] ← item
  ELSE
    OUTPUT "Queue is full, cannot enqueue"
ENDIF
```

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

In pseudocode, queue operations are handled using the following statements.

To remove an item from the queue and store in item

```
IF queueLength = 0
  THEN
    OUTPUT "Queue is empty, cannot dequeue"
  ELSE
    Item ← queue[frontPointer]
    IF frontPointer = upperBound
      THEN
        frontPointer ← 1
      ELSE
        frontPointer ← frontPointer + 1
    ENDIF
    queueLength ← queueLength - 1
ENDIF
```

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

A Queue is a FIFO (first in, first out) list with the following operations: Enqueue, Dequeue, Size, Font.

- **Queue():** creates a new queue that is empty. It needs no parameters and returns an empty queue.
- **enqueue(item):** adds a new item to the rear of the queue. It needs the item and returns nothing.
- **dequeue():** removes the item from the front of the queue. It needs no parameters and returns the item. The queue is modified.
- **front():** returns the front item from the queue but does not remove it. It needs no parameters. The queue is not modified.
- **isEmpty():** tests to see whether the queue is empty. It needs no parameters and returns a boolean value.
- **size():** returns the number of items on the queue. It needs no parameters and returns an integer.
- All operations o queue adt is same as stack except dequeue. It pick first item of the queue.

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Linear Queue:

A linear queue works exactly as in real life. When you go to stand in the queue you just stand at the rear, but when the person at the front of the queue has finished being served everyone else in the queue has to shuffle forward a little bit. This is highly inefficient as each item in the array holding the queue has to be moved along by one position. If your queue is 1 million items long you will need to move 1 million items!

We need a more efficient solution and for this we can use a circular queue.

Circular Queue:

In a circular queue, you keep track of the index of the front and rear of the queue.

- Each time you add a new item to the queue, you add it to the new empty space and then you increment the rear index by one.
- Each time you remove an item from the front of the queue you remove it and then increment the front index by one.
- If either index is equal to the length of the array then you need to reset that index to zero.

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue Methods

Here are the standard queue methods:

- **Enqueue:** This is where an item is added to the rear of the queue
- **Dequeue**: This is where an item is removed from the front of the queue and returned to the program.
- **Peek**: This is where the item at the front is returned, without removing the item from the queue

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Open Python IDLE and copy the code below for more practical activities

Python Queue Methods.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python3...

File Edit Format Run Options Window Help

```
from queue import Queue  # watch out with the capital letters

# Making the queue
queuename = Queue()

# We use .put() to insert value and .get() to pop the first one.
# You can think this as a normal queue

queuename.put(1)  # adds int 1 to index 0
# To access the queue, you need to change it to list
print(list(queuename.queue))
# Output : [1]

queuename.put(2)  # adds int 2 to index 1
print(list(queuename.queue))
# Output: [1,2]

queuename.get()  # removes index 0 (int 1)
print(list(queuename.queue))
# Output: [2]
```

```
 RESTART: C:/Users/
eue Methods.py
[1]
[1, 2]
[2]
```

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue: Open Python IDLE and copy the code below for more practical activities: Example 1

deque.py - C:/Users/v90001010/AppData/Local/Programs/Pythor

File Edit Format Run Options Window Help

```
from collections import deque
queue = deque()
queue
deque([])

queue.append("Ammar")
queue.append("Andy")
queue.append("Aiden")
queue
deque(['Ammar', 'Andy', 'Aiden'])
```

IDLE Shell 3.10.8

File Edit Shell Debug Options Window He

```
Python 3.10.8 (tags/v3.10.8:aaa
AMD64)] on win32
Type "help", "copyright", "cred
>>>
= RESTART: C:/Users/v90001010/A
>>> queue.popleft()
'Ammar'
>>> queue
deque(['Andy', 'Aiden'])
>>> queue.popleft()
'Andy'
>>> queue
deque(['Aiden'])
```

Every time you call **popleft(),** you remove the head element from the linked list, mimicking a real-life queue.

10.4 Abstract Data Types (ADTs)

10.4.2 - Queue Operations:

Queue: Open Python IDLE and copy the code below for more practical activities: Example 2

queue1.py - C:/Users/v90001010/AppData/Local/Programs/Python/Python310/qu

File Edit Format Run Options Window Help

```
# Python program to
# demonstrate queue implementation
# using list

# Initializing a queue
queue = []

# Adding elements to the queue
queue.append('N')
queue.append('O')
queue.append('U')
queue.append('R')

print("Initial queue")
print(queue)

# Removing elements from the queue
print("\nElements dequeued from queue")
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))
print(queue.pop(0))

print("\nQueue after removing elements")
print(queue)

# Uncommenting print(queue.pop(0))
# will raise and IndexError
# as the queue is now empty
```

File Edit Shell Debug Options Window Help

```
Python 3.10.8 (tags/v3.10.8:aaaf517, Oct 11 2022
AMD64)] on win32
Type "help", "copyright", "credits" or "license(
>>>
= RESTART: C:/Users/v90001010/AppData/Local/Prog
Initial queue
['N', 'O', 'U', 'R']

Elements dequeued from queue
N
O
U
R

Queue after removing elements
[]
```

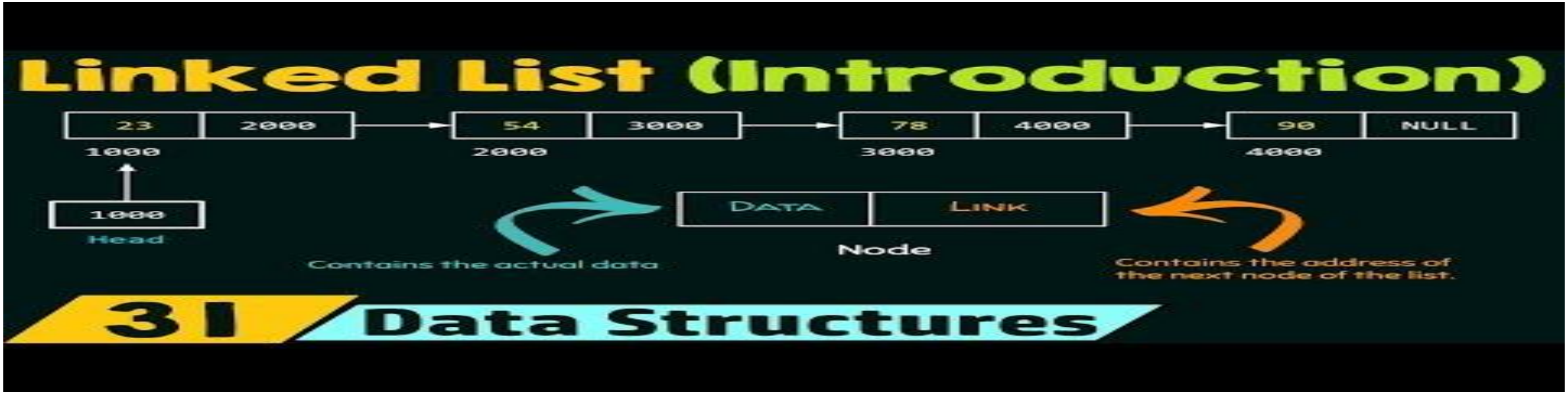
10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked List: a list containing several elements or items in which **each items in the list points to the next item in the list**. In a linked list a **new item is always added to the start of the list**.

Click the link below the watch the video for Linked List

https://www.youtube.com/watch?v=R9PTBwOzceo



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked lists – Part 1

https://www.youtube.com/watch?v=sdO9cPdgVAk&t=3s

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked lists – Part 2

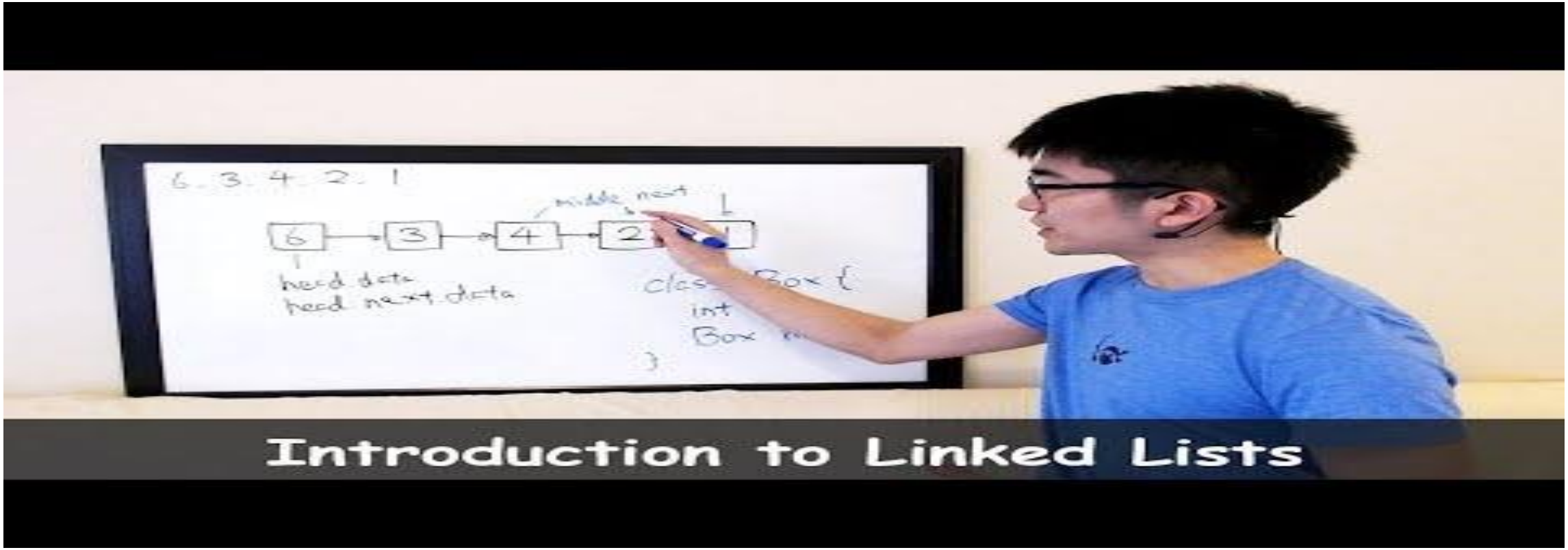
https://www.youtube.com/watch?v=RJMfJJVed2s

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Click the link below the watch the video for linked List

https://www.youtube.com/watch?v=WwfhLC16bis

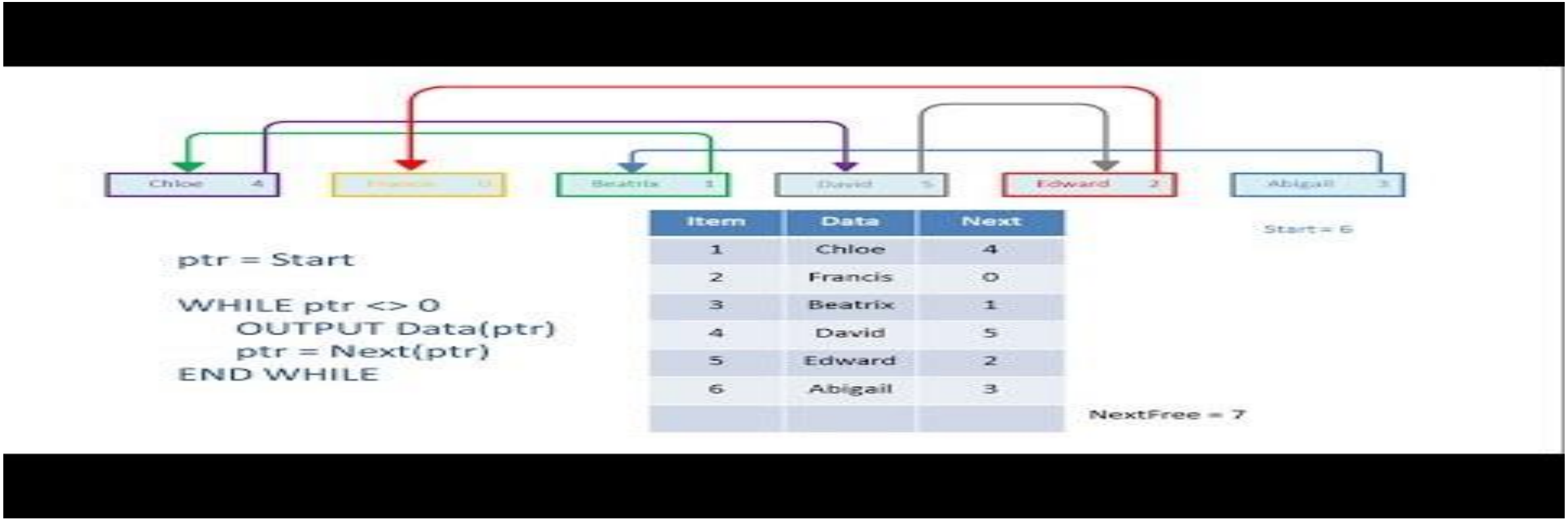


10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Click the link below the watch the video for linked List

https://www.youtube.com/watch?v=QyFuV07lGSc

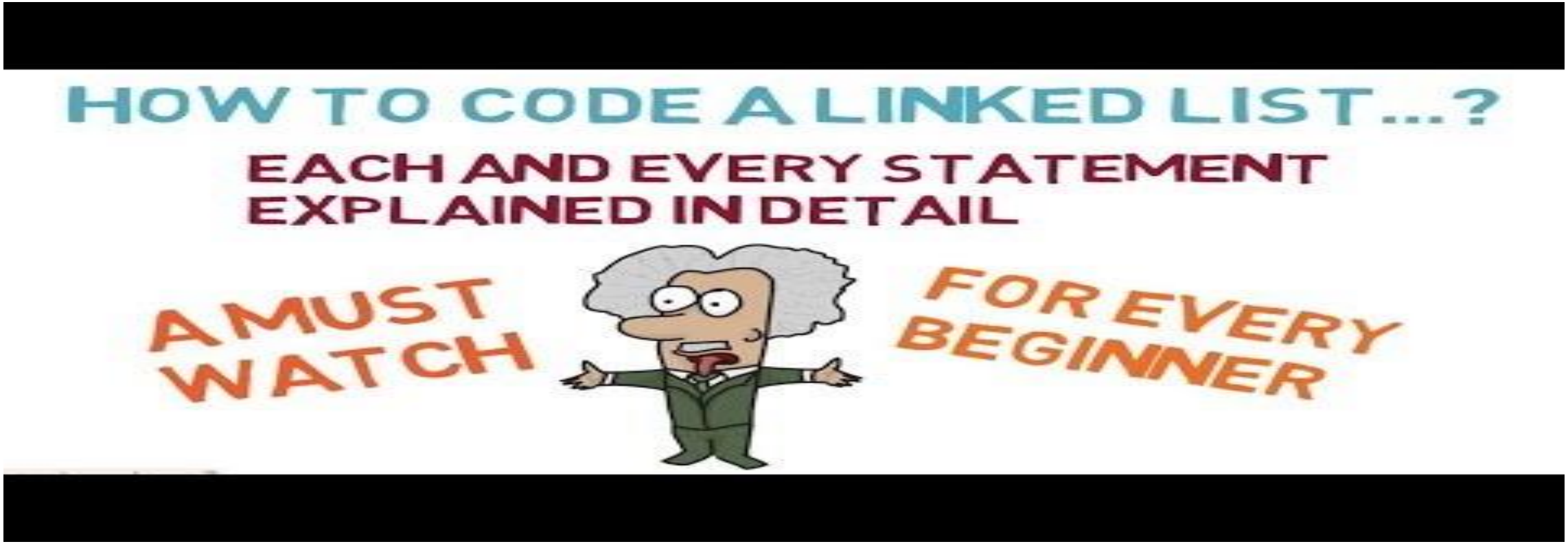


10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Click the link below the watch the video for Linked List

https://www.youtube.com/watch?v=VmWvpwxa-rM

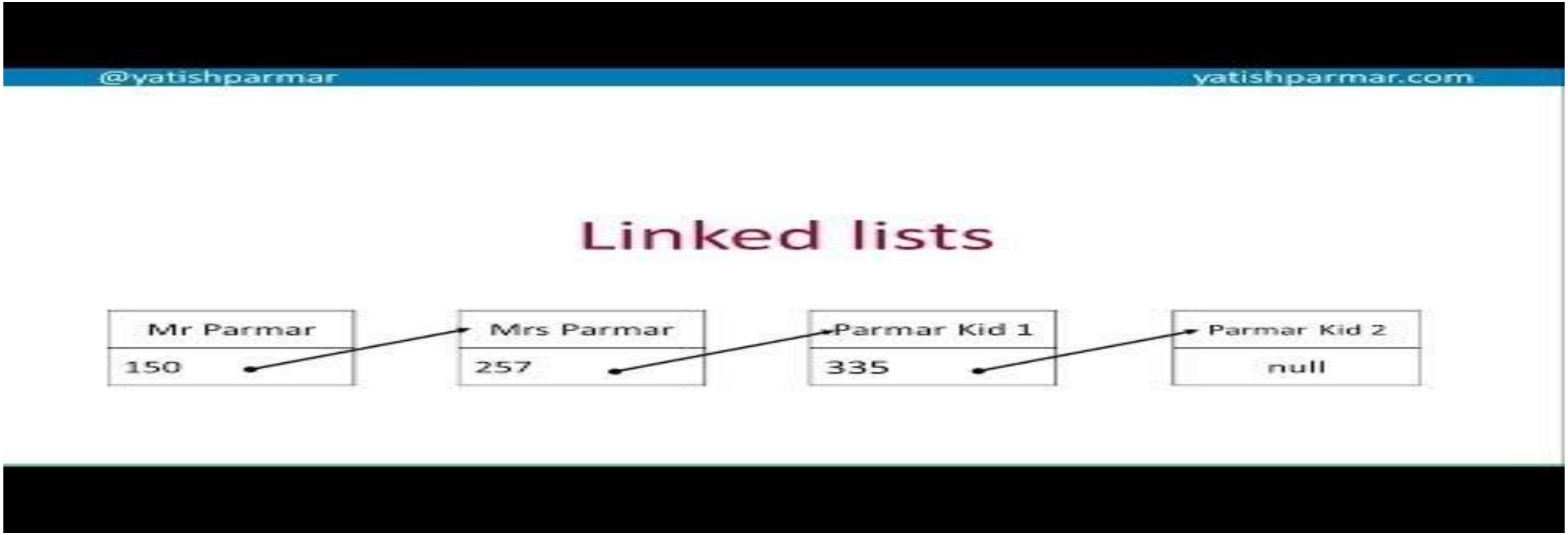


10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Click the link below the watch the video for linked List

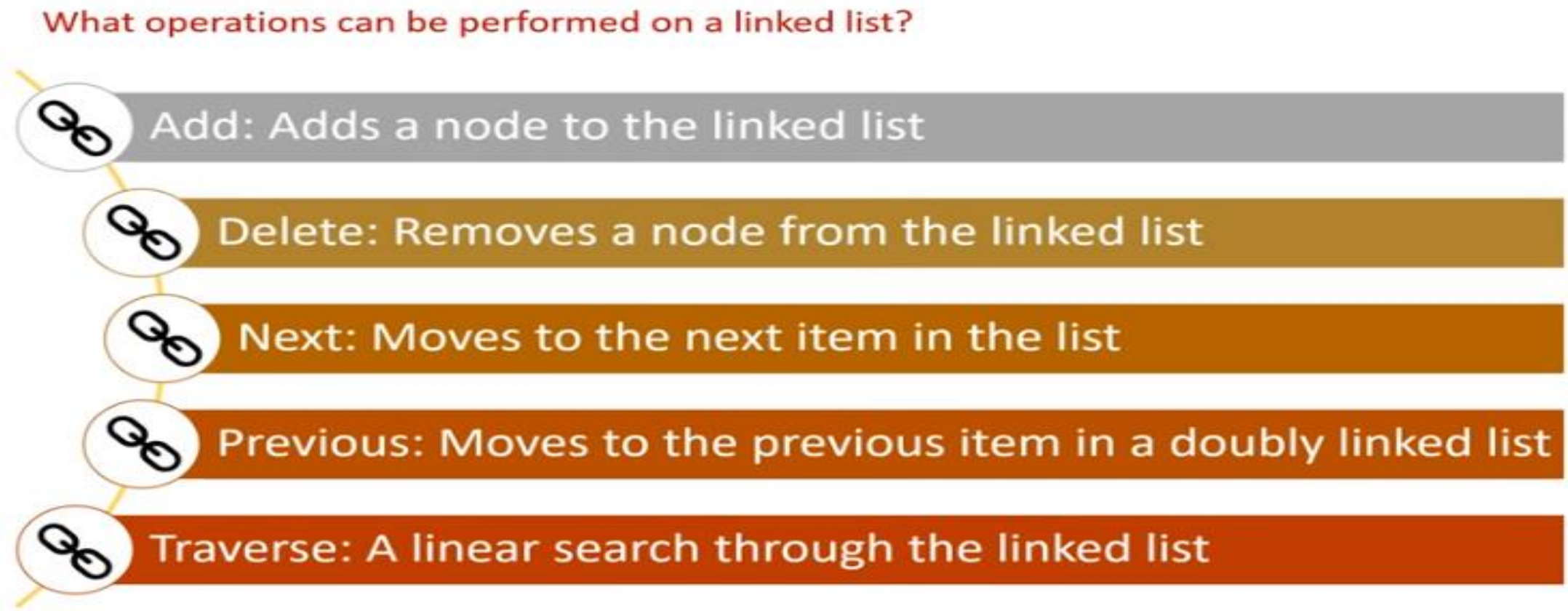
https://www.youtube.com/watch?v=78gRSe2J48M



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked lists:



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

We used an array as a linear list. In a linear list, the list items are stored in consecutive locations. This is not always appropriate. Another method is to store an individual list item in whatever location is available and link the individual item into an ordered sequence using pointers.

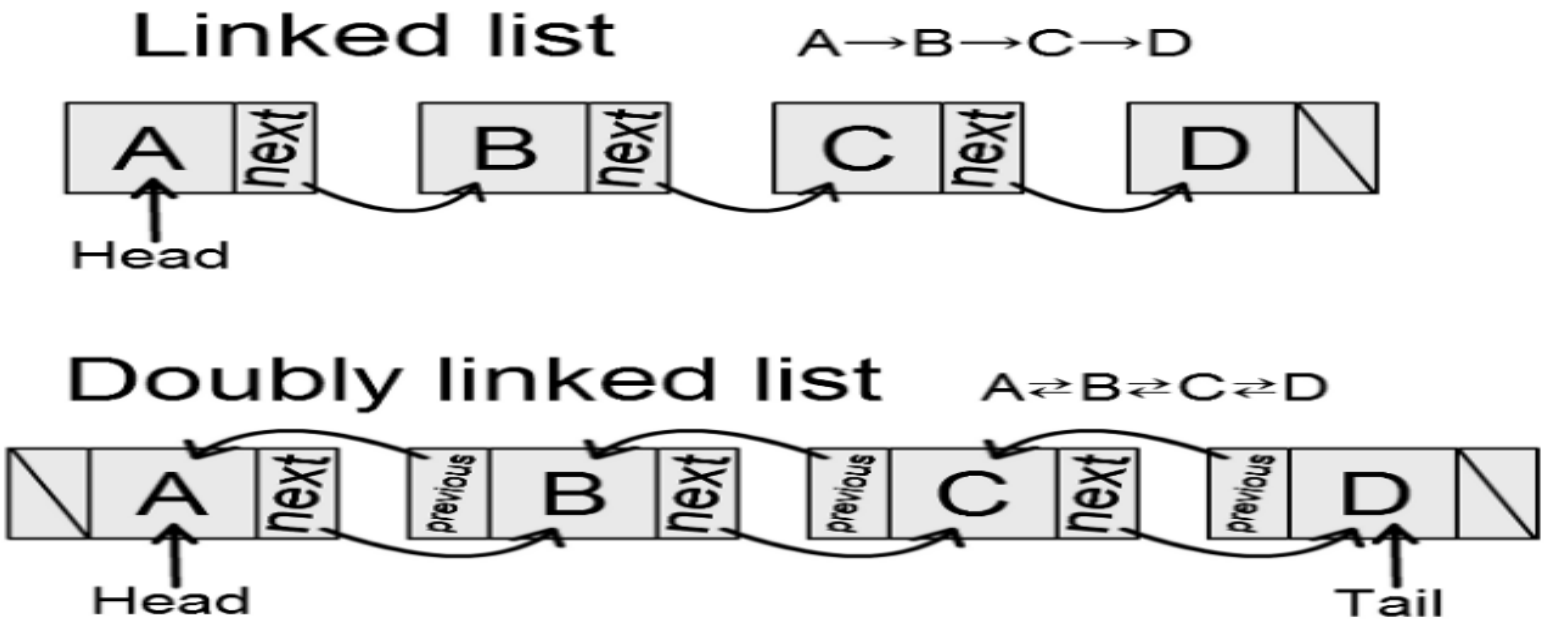
- An element of a list is called a node. A node can consist of several data items and a pointer, which is a variable that stores the address of the node it points to.
- A pointer that does not point at anything is called a null pointer. It is usually represented by ∅. A variable that stores the address of the first element is called a start pointer.
- **Node:** an element of a list
- **Pointer:** a variable that stores the address of the node it points to
- **Null pointer:** a pointer that does not point at anything
- **Start pointer:** a variable that stores the address of the first element of a linked list

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked List is an Abstract Data Type (ADT) that holds a collection of **Nodes**, the nodes can be accessed in a sequential way. **Linked List doesn't provide a random access to a Node.**

Usually, those Nodes are connected to the next node and/or with the previous one, this gives the **linked** effect. When the Nodes are connected with only the **next** pointer the list is called **Singly Linked List** and when it's connected by the **next** and **previous** the list is called **Doubly Linked List**.



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

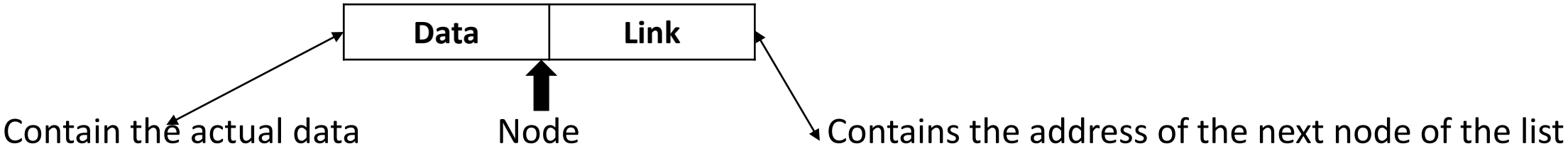
Single Linked List: Navigation is Forward only

Doubly Linked List: Forward and backward navigation is possible

Circular Linked List: Last element is linked to the first element

Single Linked List:

A single linked is a list made up of nodes that consist of two parts: Data and Link



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Linked lists: **Open Python IDLE and copy the code below for more practical activities: Example 1**

*Creation of Linked list.py - C:/Users/v90001010/AppData/Local/Programs/Python/Py

File Edit Format Run Options Window Help

```
class Node:
    def __init__(self, dataval=None):
        self.dataval = dataval
        self.nextval = None

class SLinkedList:
    def __init__(self):
        self.headval = None

    def listprint(self):
        printval = self.headval
        while printval is not None:
            print (printval.dataval)
            printval = printval.nextval

list = SLinkedList()
list.headval = Node("Mon")
e2 = Node("Tue")
e3 = Node("Wed")

# Link first Node to second node
list.headval.nextval = e2

# Link second Node to third node
e2.nextval = e3

list.listprint()

#(Singly linked lists can be traversed in only forward
#direction starting form the first data element.
#We simply print the value of the next dat
#a element by assigning the pointer of the next node to
#the current data element.)
```

```
= RESTART: C:/Users/v90001010
f Linked list.py
Mon
Tue
Wed
```

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

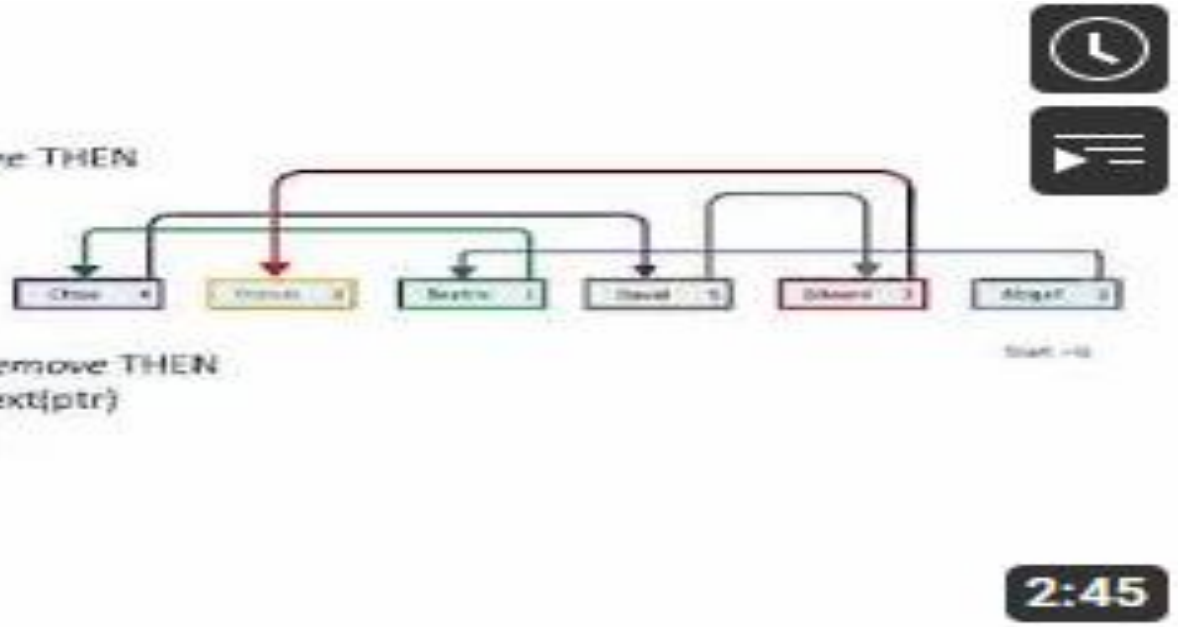
Remove an item from Linked List

https://www.youtube.com/watch?v=c3PtImqO1E8

```
ptr = Start
prevPtr = Start

IF Data(ptr) = item to remove THEN
      Start = Next(ptr)
      EXIT PROCEDURE
END IF

WHILE ptr <> 0
      IF Data(ptr) = item to remove THEN
            Next(prevPtr) = Next(ptr)
            EXIT PROCEDURE
      END IF
      prevPtr = ptr
      ptr = Next(ptr)
END WHILE
```



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Deleting items from a linked list

Deleting items from a linked list The algorithm to delete an item from the linked list myLinkedList could be written as a procedure in pseudocode as shown below.

```
DECLARE itemDelete : INTEGER
DECLARE oldIndex : INTEGER
DECLARE index : INTEGER
DECLARE startPointer : INTEGER
DECLARE heapStartPointer : INTEGER
DECLARE tempPointer : INTEGER
CONSTANT nullPointer = -1
PROCEDURE linkedListDelete(itemDelete)
  // check for list empty
  IF startPointer = nullPointer
    THEN
      OUTPUT "Linked list empty"
    ELSE
      // find item to delete in linked list
      index ← startPointer
      WHILE myLinkedList[index] <> itemDelete AND
        (index <> nullPointer) DO
          oldIndex ← index
          index ← myLinkedListPointers[index]
      ENDWHILE
```

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Deleting items from a linked list

```
        IF index = nullPointer
          THEN
            OUTPUT "Item ", itemDelete, " not found"
          ELSE
          // delete the pointer and the item
           tempPointer ← myLinkedListPointers[index]
           myLinkedListPointers[index] ← heapStartPointer
           heapStartPointer ← index
           myLinkedListPointers[oldIndex] ← tempPointer
      ENDIF
  ENDIF
ENDPROCEDURE
```

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Deleting items from a linked list

Here is the identifier table.

Identifier	Description
startPointer	Start of the linked list
heapStartPointer	Start of the heap
nullPointer	Null pointer set to −1
index	Pointer to current list element
oldIndex	Pointer to previous list element
itemDelete	Item to delete from the list
tempPointer	Temporary pointer

The trace table below shows the algorithm being used to delete 36 from myLinkedList.

startPointer	heapStartPointer	itemDelete	index	oldIndex	tempPointer
Already set to 4	Already set to 5	36	4	4	
			3	3	
			2		
	2				1

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Deleting items from a linked list

The linked list, myLinkedList, will now be as follows.

		myLinkedList	myLinkedListPointers
	[0]	27	–1
	[1]	19	0
heapStartPointer →	[2]	36	6
	[3]	42	1
	[4]	16	3
startPointer →	[5]	18	4
	[6]		7
	[7]		8
	[8]		9
	[9]		10
	[10]		11
	[11]		–1

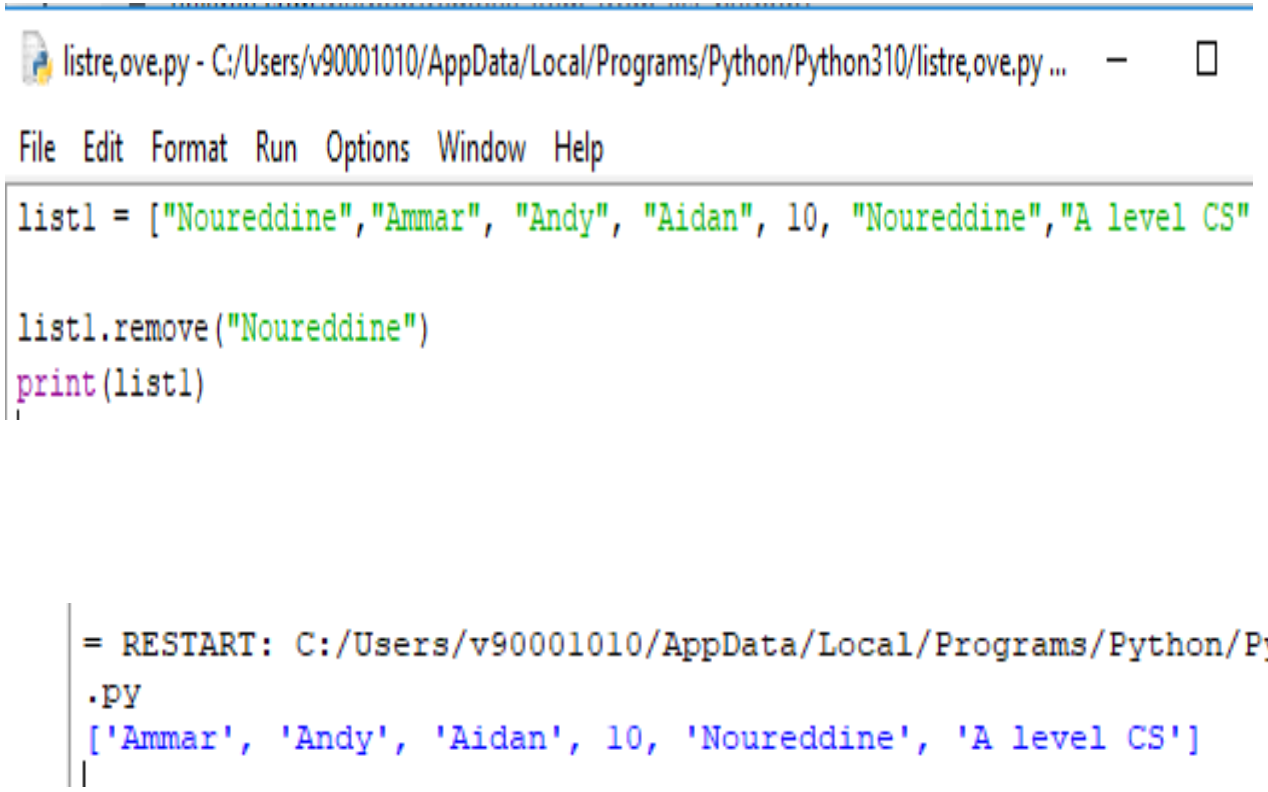
updated pointers

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

delete an element from the linked list:

Remove item from list by value

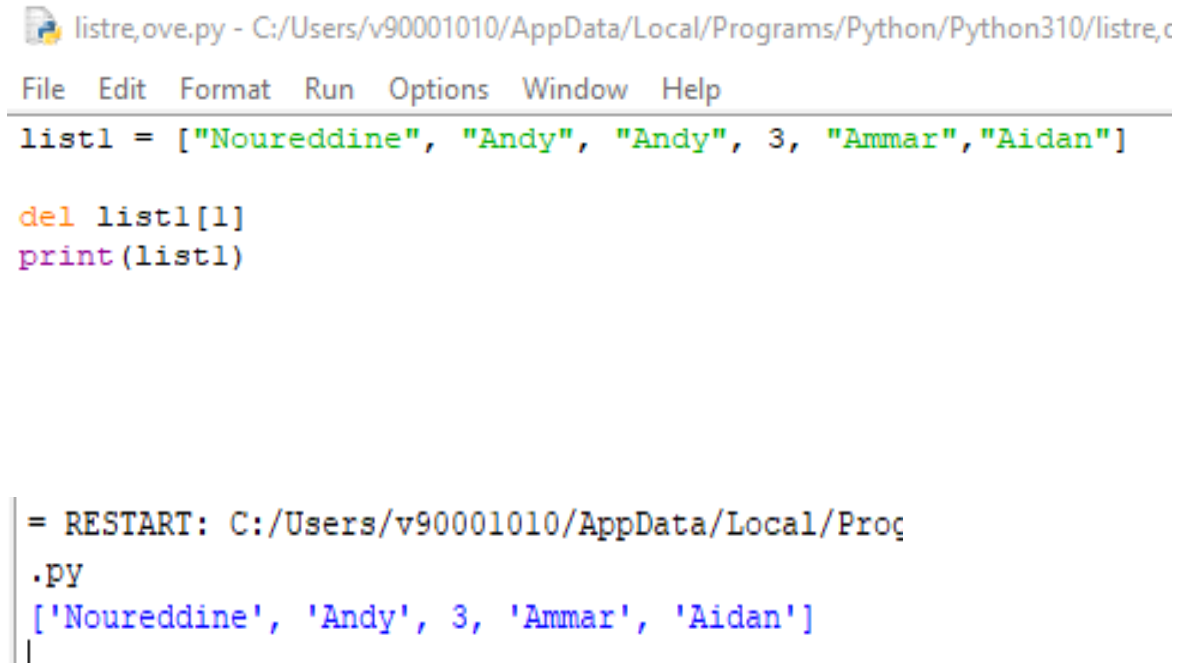


10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Delete an element from the linked list:

remove items using **del**

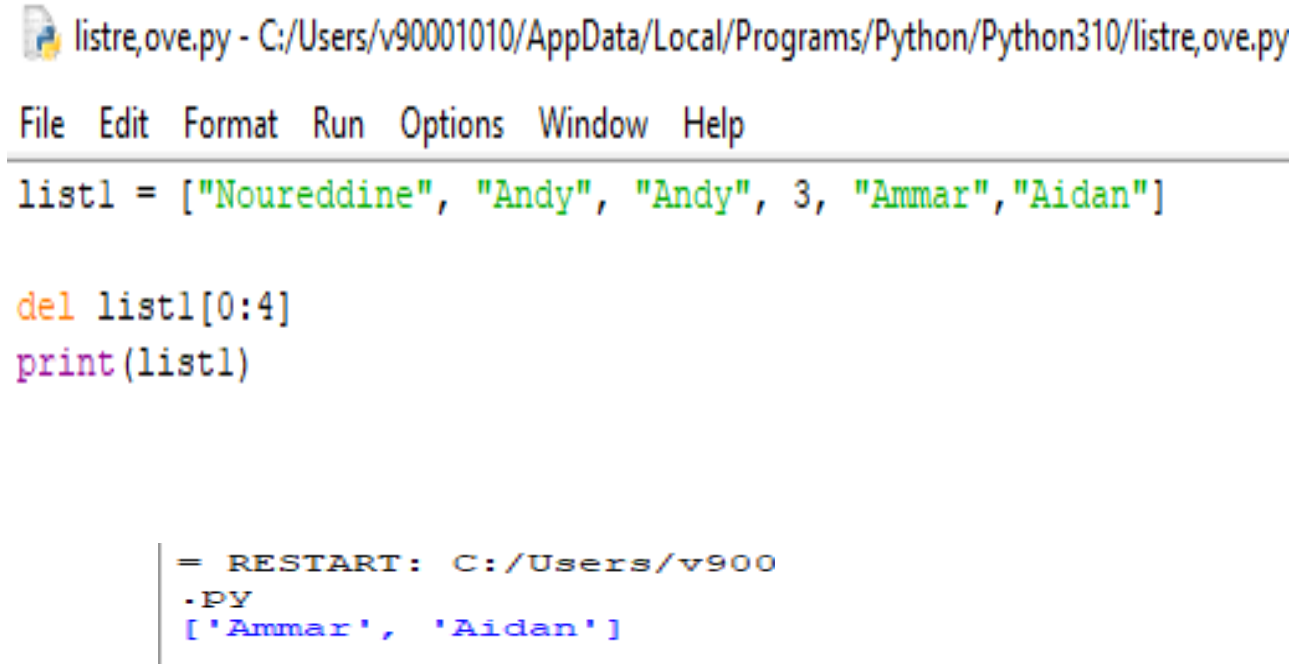


10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Delete an element from the linked list:

And if you are looking to **remove multiple items from a list**, **adding an index range** would help you achieve this.



10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

Delete an element from the linked list:

Delete the last element of the list is by using the **del** statement.

*delete an item from link list.py - C:/Users/v90001010/Appl

File Edit Format Run Options Window Help

```
# program to delete the last
# last element from the list
#using del

list = [1,2,3,4,5]
print("Original list: " +str(list))

# call del operator
del list[-1]

# print the updated list
print("Updated list: " +str(list))
```

```
= RESTART: C:/Users/v90001010/AppDa
item from link list.py
Original list: [1, 2, 3, 4, 5]
Updated list: [1, 2, 3, 4]
```

10.4 Abstract Data Types (ADTs)

10.4.3 - Linked List Operations:

delete an element from the linked list:

```
def delete(itemDelete):
    global startPointer, heapStartPointer
    if startPointer == nullPointer:
        print("Linked List empty")
    else:
        index = startPointer
        while myLinkedList[index] != itemDelete and index != nullPointer:
            oldindex = index
            index = myLinkedListPointers[index]
        if index == nullPointer:
            print("Item ", itemDelete, " not found")
        else:
            myLinkedList[index] = None
            tempPointer = myLinkedListPointers[index]
            myLinkedListPointers[index] = heapStartPointer
            heapStartPointer = index
            myLinkedListPointers[oldindex] = tempPointer
```