Title : **Computer Programming Language Control Flow Structure Syntax, Function and Cross Comparative Analysis**

Objectives/Goals

- Develop a good understanding and insight into OOP
- Understand Control Flow Structures including SCOPE
- Learn Python, Java, Javascript and Elixir Control Flow Structures (Syntax)
- Write the same control flow code using the above mentioned programming languages
- Cross compare control flow structure for all the above languages

NOTE
Elixir is a functional programming language

CODING TASK

- Initiate data structure either array/list dependent on language of integers
- Use an outer loop and inner loop to loop through the data structure and output the elements in a particular order/sequence (see below)
- Record any issues and workaround using all the different languages being used for this task
- Awareness of programming language prejudices and preferences (record)

NOTE
Depending on the programming language it is important to remember which data structures are ordered.

Text Editors / IDE's being used for project:

- Anaconda Navigator Jupyter Notebook (Python)
- VS Code (Javascript)
- IntelliJ (Java)

CODE OUTPUT

Data Structure elements: 10, 20, 30, 40, 50

```
Desired Output:

Iteration 1
Outer Loop: 10
Inner Loop: 20

Iteration 2
Outer Loop: 20
Inner Loop: 30

Iteration 3
Outer Loop: 30
Inner Loop: 40

Iteration 4
Outer Loop: 40
Inner Loop: 50

Iteration 5
Outer Loop: 50
Inner Loop: n/a
```

NOTE
The output must be in this exact sequence

Scope in Object Oriented Programming
The scope of an identifier name binding – an association of a name to an entity, such as a variable – is the region of a computer program where the binding is valid: where the name can be used to refer to the entity. Such a region is referred to as a scope block.

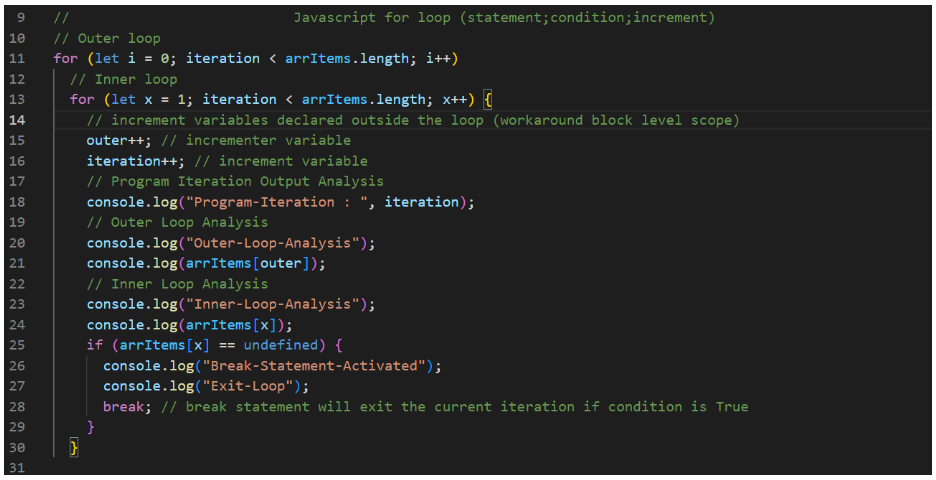
Scope is the area of the program where an item (be it variable, constant, function, etc.) that has an identifier name is recognized.

Block scope means a variable declared inside a loop cannot be accessed outside the loop. This poses problems for the specific sequence output we are looking for outlined above using Java, Python and Javascript. This is important to understand when working with OOP languages.

Javascript

Block scope means a variable declared inside a loop cannot be accessed outside the loop. This poses problems for the specific sequence output we are looking for outlined above using Javascript.

The problem that arises is the inner loop will execute all inner loop iterations each time the outer loop is iterated once based on statement, condition and increment value. This requires syntax knowledge and trial and error when building this code using Javascript.



```javascript
//                           Javascript for loop (statement;condition;increment)
// Outer loop
for (let i = 0; iteration < arrItems.length; i++)
  // Inner loop
  for (let x = 1; iteration < arrItems.length; x++) {
    // increment variables declared outside the loop (workaround block level scope)
    outer++; // incrementer variable
    iteration++; // increment variable
    // Program Iteration Output Analysis
    console.log("Program-Iteration : ", iteration);
    // Outer Loop Analysis
    console.log("Outer-Loop-Analysis");
    console.log(arrItems[outer]);
    // Inner Loop Analysis
    console.log("Inner-Loop-Analysis");
    console.log(arrItems[x]);
    if (arrItems[x] == undefined) {
      console.log("Break-Statement-Activated");
      console.log("Exit-Loop");
      break; // break statement will exit the current iteration if condition is True
    }
  }
```

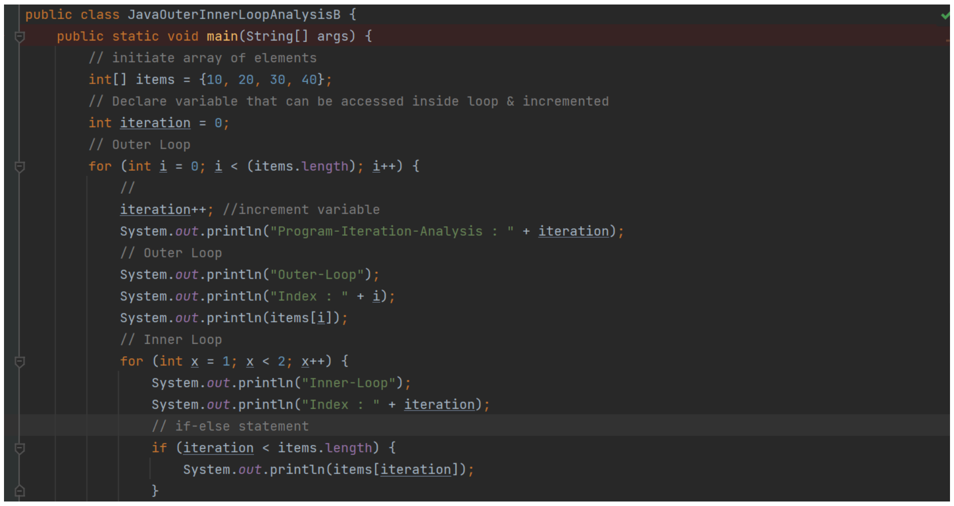
We can implement a workaround for block scope in Javascript, by declaring a variable outside the loop using the let keyword and incrementing this variable inside the loop. The let keyword in Javascript is very important for this specific task as we want to update our incrementer values from within the loops (block scope). In contrast for Javascript if we used the const keyword this would not work as this declaration keyword in Javascript cannot re-assign a value. This is a workaround to access and update variables from within the loops. If we did not declare these variables outside the loop, these variables would not be accessible outside the loop (block scope) and therefore would not be updated/incremented in this case. These variables will be used to subset and control the output sequence.

Java

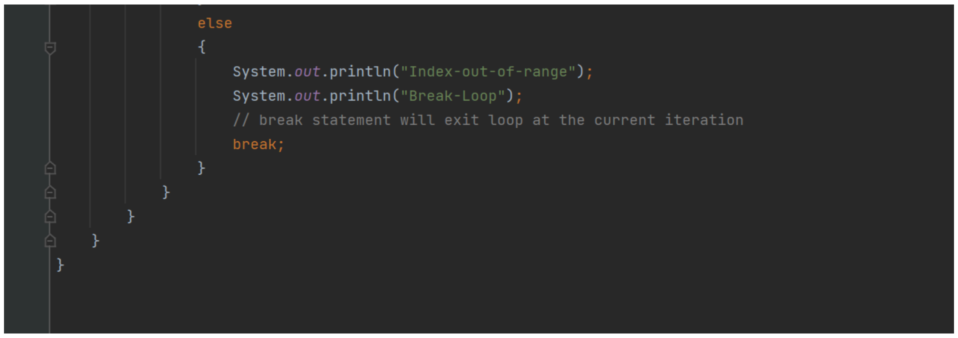
Once again we have to be aware of block level scope as highlighted previously in our Javascript code. This is the same for Java. For a variable in Java declared inside a loop, to be accessible outside a loop, we must declare the variable prior to the loop.

In contrast to Javascript we do not have keywords for declaring variables in Java or Python. However, in Java we do need to initiate the data type before initiating the variable name followed by a variable value. In Python we do not have keywords for initiating variables or have to declare the data type. Furthermore, in Python we do have the option to initiate the date type of the variable using a constructor function for example: variable name = int().

Notably, Java's syntax for initiating a for loop is pretty much the exact syntax used in Javascript, the only difference is variable declaration (keywords) in Javascript and for Java as mentioned initiating the data type prior to variable declaration.



```java
public class JavaOuterInnerLoopAnalysisB {
    public static void main(String[] args) {
        // initiate array of elements
        int[] items = {10, 20, 30, 40};
        // Declare variable that can be accessed inside loop & incremented
        int iteration = 0;
        // Outer Loop
        for (int i = 0; i < (items.length); i++) {
            //
            iteration++; //increment variable
            System.out.println("Program-Iteration-Analysis : " + iteration);
            // Outer Loop
            System.out.println("Outer-Loop");
            System.out.println("Index : " + i);
            System.out.println(items[i]);
            // Inner Loop
            for (int x = 1; x < 2; x++) {
                System.out.println("Inner-Loop");
                System.out.println("Index : " + iteration);
                // if-else statement
                if (iteration < items.length) {
                    System.out.println(items[iteration]);
                }
```



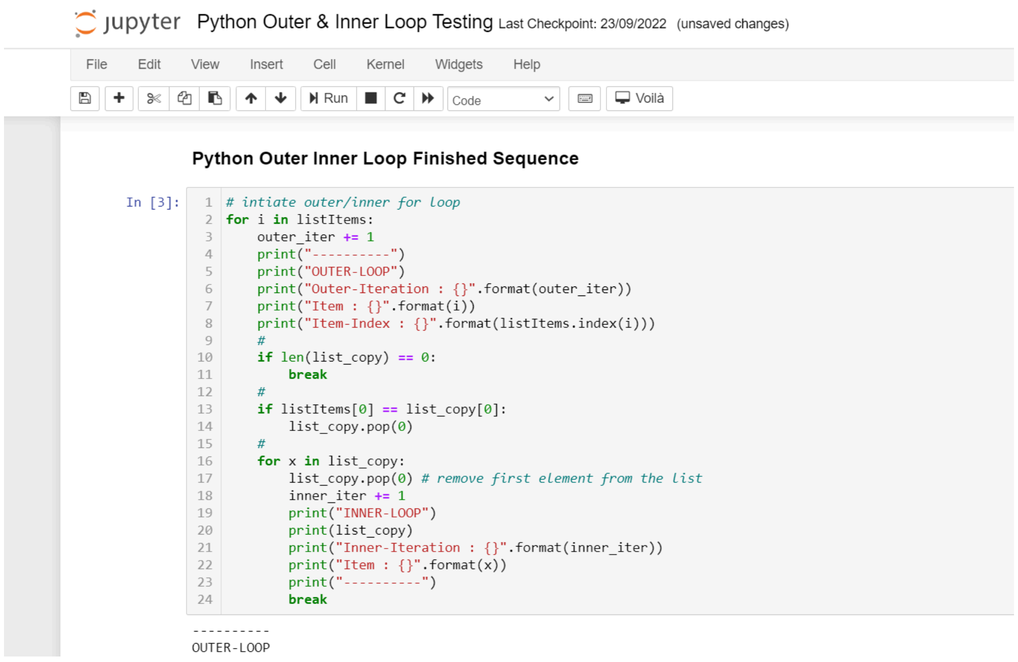
```java
                else
                {
                    System.out.println("Index-out-of-range");
                    System.out.println("Break-Loop");
                    // break statement will exit loop at the current iteration
                    break;
                }
            }
        }
    }
}
```

We can note the break statement being used above in Java which is the same keyword for Java, Javascript and Python. This keyword exits the loop at the current iteration stopping the program in this instance reaching out of reach index and raising an error.

Python

The first thing to mention is the excellent (preferred) preference for using Jupyter Notebook for keeping our code clean and readable in comparison to VSCode and InteliJ. However, this is from a rookie's limited experience but still holds validation. Particularly the output and is easy to read and interpretable in itself. Furthermore, our ability to add side notes via Markdown text is very useful and helps the interpreter gain a better understanding of the notebook's function.

When we cross compare the syntax used to implement a for loop in Python in comparison to Java and Javascript we can see some differentiable elements. We can note we do not initiate a variable, condition or increment the value to control the code. We do not use parenthesis but a semi colon followed by whitespace. Whitespace is highly important in Python as it defines the code block.


jupyter
Python Outer & Inner Loop Testing Last Checkpoint: 23/09/2022 (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code
Voilà

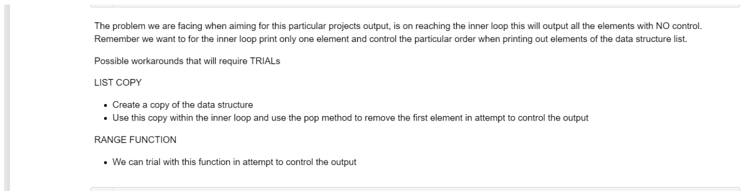
Python Outer Inner Loop Finished Sequence

In [3]:

```python
# intiate outer/inner for loop
for i in listItems:
    outer_iter += 1
    print("----------")
    print("OUTER-LOOP")
    print("Outer-Iteration : {}".format(outer_iter))
    print("Item : {}".format(i))
    print("Item-Index : {}".format(listItems.index(i)))
    #
    if len(list_copy) == 0:
        break
    #
    if listItems[0] == list_copy[0]:
        list_copy.pop(0)
    #
    for x in list_copy:
        list_copy.pop(0) # remove first element from the list
        inner_iter += 1
        print("INNER-LOOP")
        print(list_copy)
        print("Inner-Iteration : {}".format(inner_iter))
        print("Item : {}".format(x))
        print("----------")
        break
```

```
----------
OUTER-LOOP
```

It is interesting to see how the above syntax differs from the other programming languages but still performs the same task. We could have built Python's try and except clause into our code but the following syntax was used in preference.



The problem we are facing when aiming for this particular projects output, is on reaching the inner loop this will output all the elements with NO control. Remember we want to for the inner loop print only one element and control the particular order when printing out elements of the data structure list.

Possible workarounds that will require TRIALs

LIST COPY

- Create a copy of the data structure
- Use this copy within the inner loop and use the pop method to remove the first element in attempt to control the output

RANGE FUNCTION

- We can trial with this function in attempt to control the output

We can see how useful the markup text in Jupyter Notebook is for the direction of our work.