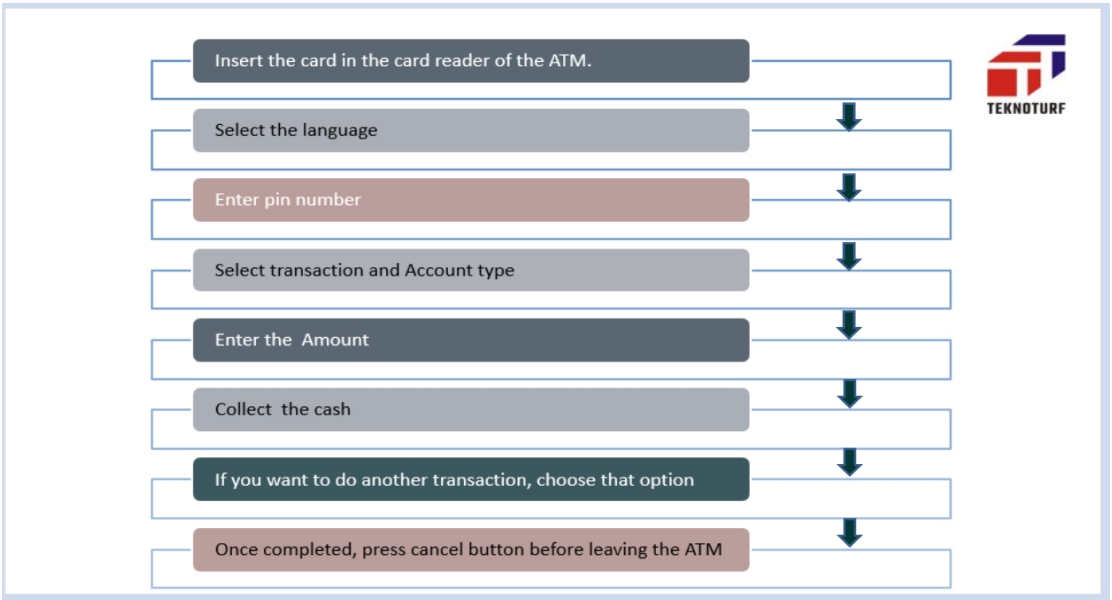
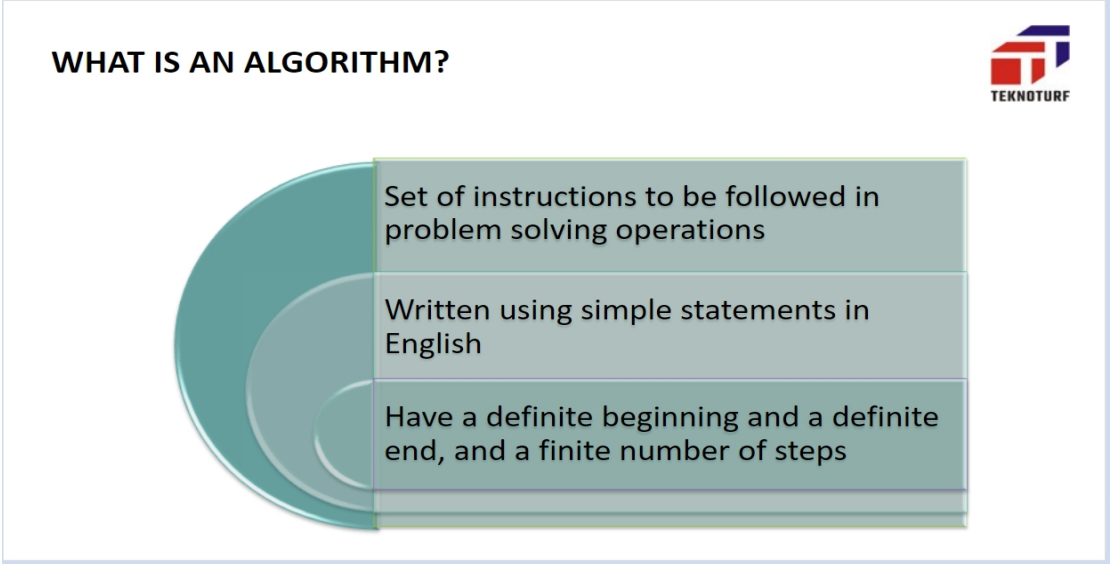
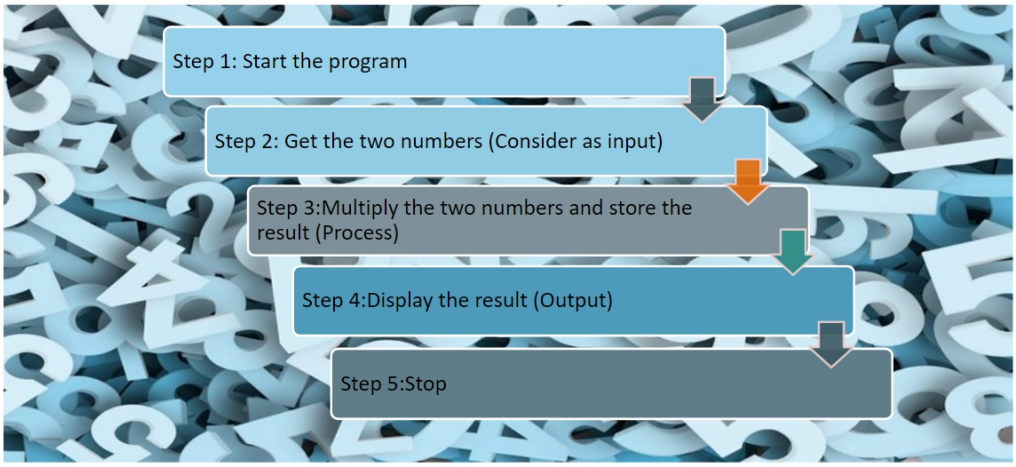
Logic Development - Introduction to Algorithm, Flowchart and Pseudocode



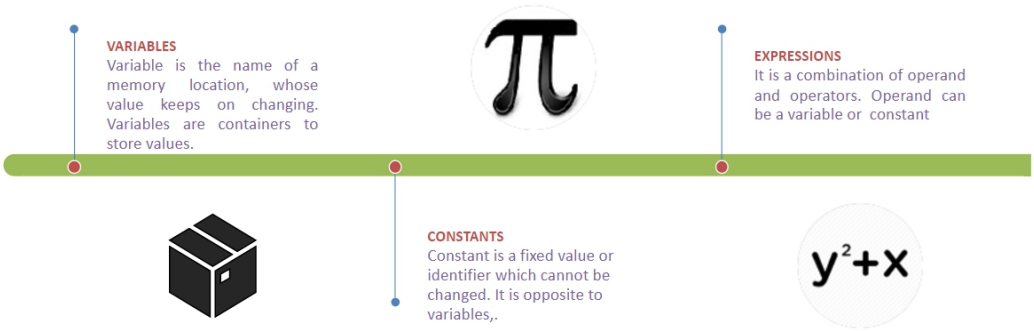


ALGORITHM TO MULTIPLY 2 NUMBERS



DEFINING VARIABLES, CONSTANTS & EXPRESSIONS

An algorithm consists of variables, constants and Expressions



BASIC OPERATORS AND OPERATIONS

Operator is a symbol that represents an action or process

There are several classifications of operators and each of them can have one or more operands, a specific data that is to be manipulated.

Basic classification of operators are
- Assignment Operator
- Arithmetic Operators
- Relational Operators
- Logical Operators etc.

ASSIGNMENT OPERATOR

Used to assign a value to the variable

Assignment Symbol (← or =)

- Example 1:

HEIGHT ← 5 (or) HEIGHT = 5

Assigns value 5 to the variable HEIGHT, statement is

- Example 2:

C=A+B

The sum value of variable A and variable B is assigned to the variable C.

ARITHMETIC OPERATORS

Used to perform arithmetic operations like addition, subtraction, multiplication, division etc.

Some of the basic arithmetic operators and its examples

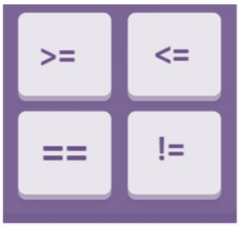
Operator	Meaning	Example
+	Addition	A + B
-	Subtraction	A – B
*	Multiplication	A *B
/	Division	A / B
^	Power(Exponentiation)	A^3 for A power 3
%	Remainder (Modulo)	A % B

RELATIONAL OPERATORS

Used to evaluate the value in the variables

Some of the basic relational operators and its examples

Operator	Meaning	Example
<	Less than	A < B
<=	Less than or equal to	A <= B
==	Equal to (Comparison)	A==B
!=	Not Equal to	A !=B
>	Greater than	A>B
>=	Greater than or equal to	A >=B



LOGICAL OPERATORS

Used to perform logical operations in the given expressions.

Some of the basic logical operators and its examples

Operator	Example	Meaning
AND	A < B AND B < C	Result is True if both A<B and B<C are true else false
OR	A< B OR B < C	Result is True if either A<B or B<C are true else false
NOT	NOT (A >B)	Result is True if A>B is false else true

BEST PRACTICES TO BE KNOWN

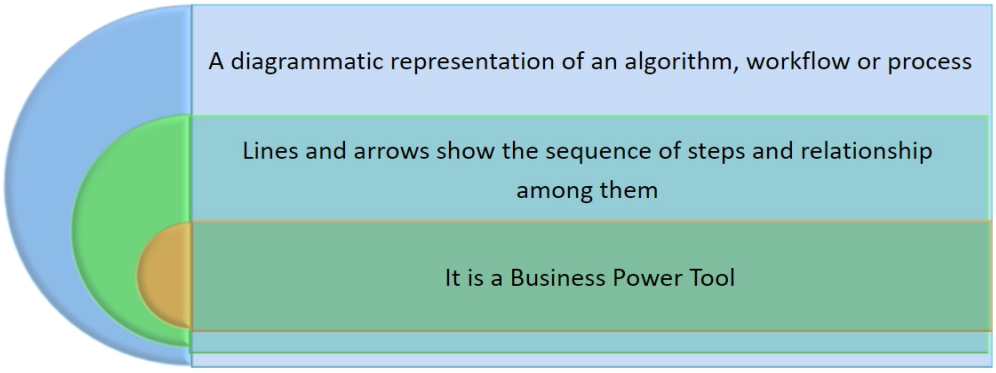
Some important rules to be followed, when using a variable

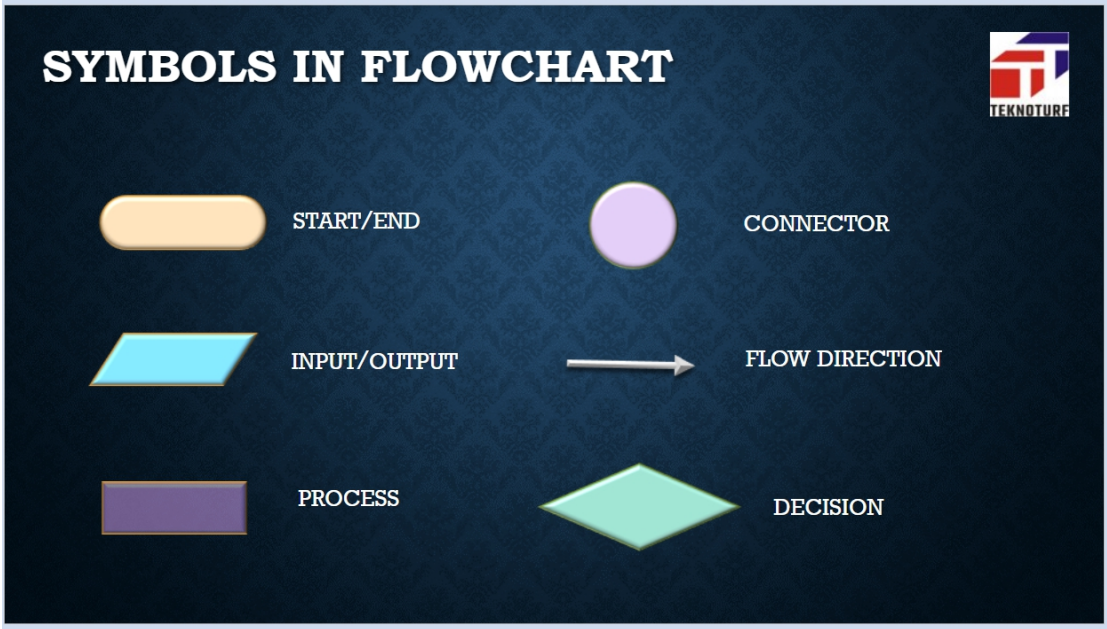
A variable name must define the exact explanation of its content. Use meaningful variable names by relating them with the purpose of the program and what it's for.

- Avoid variable names starting with numbers & symbols, except underscore(_).
- Avoid using reserved keywords like if, else
- Single variable name should not have space in between

Example:	Example:
age height roll_no reg_number	123 Xyz #x1 User name
Meaningful variable names	Bad practice of naming variables

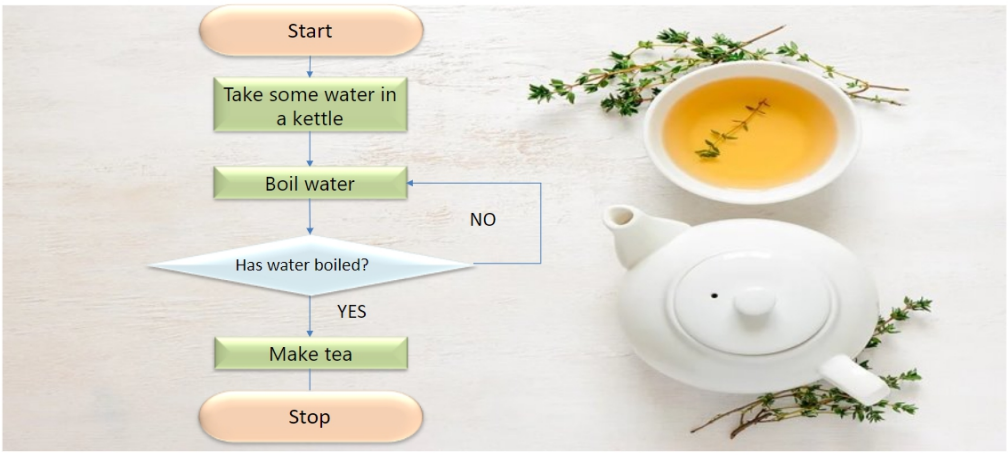
WHAT IS A FLOWCHART?



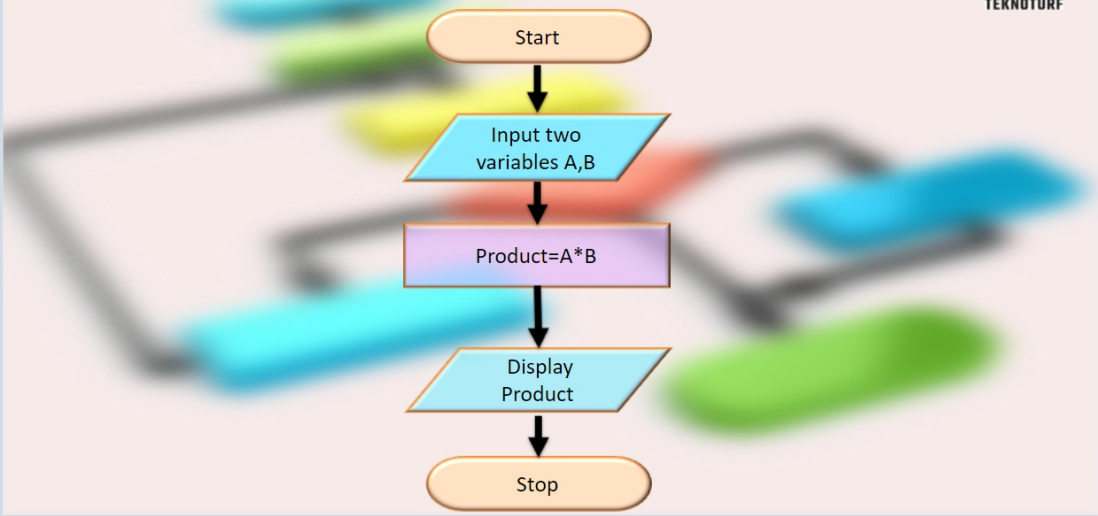


SAMPLE FLOWCHART FOR MAKING TEA

Basic Flowchart representation for making a cup of hot tea



SAMPLE FLOWCHART TO MULTIPLY 2 NUMBERS



PSEUDOCODE

- It is an Informal way of describing algorithms
- Does Not require any strict programming language syntax
- It is Easily readable and modular form

SAMPLE PSEUDOCODE FOR MAKING TEA

Pseudocode representation for making a cup of tea

BEGIN

GET water, tea leaves, sugar

PLUG IN kettle

BOIL water

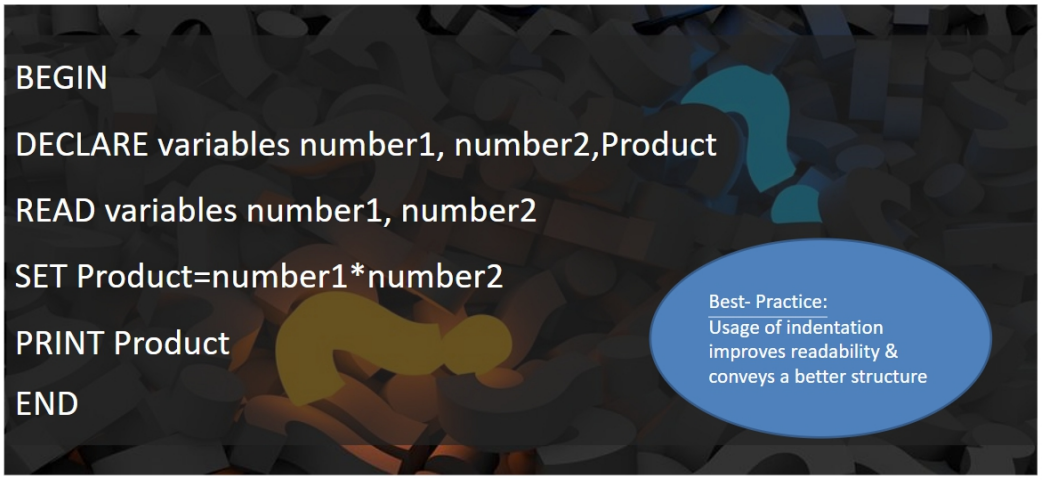
ADD tea leaves, milk

ADD sugar and stir

SERVE Tea

END

PSEUDOCODE TO MULTIPLY TWO NUMBERS



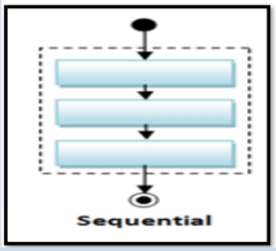
Logic Development - Selection Statements

FLOW OF A PROGRAM

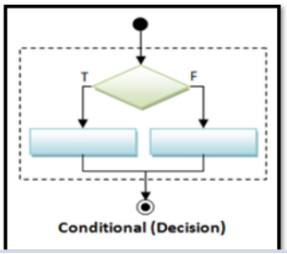
TEKNOTURF

There's an order in which the computer executes the statements in a program.

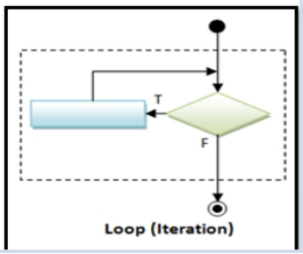
The orders even be combined to deal with a given problem.



Sequential



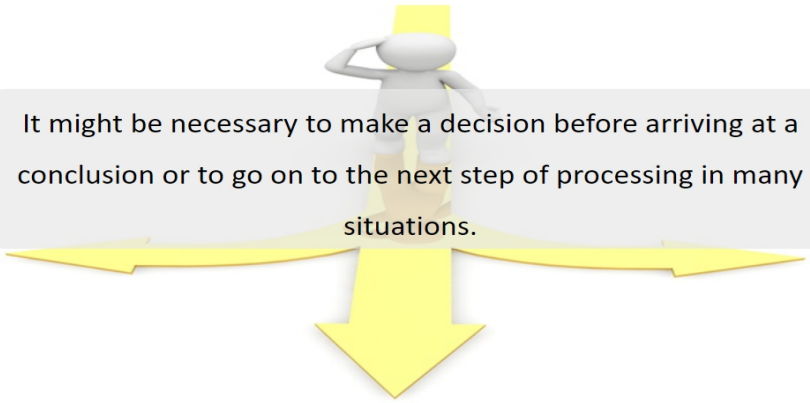
Conditional (Decision)



Loop (Iteration)

SELECTION OR CONDITIONAL STATEMENTS

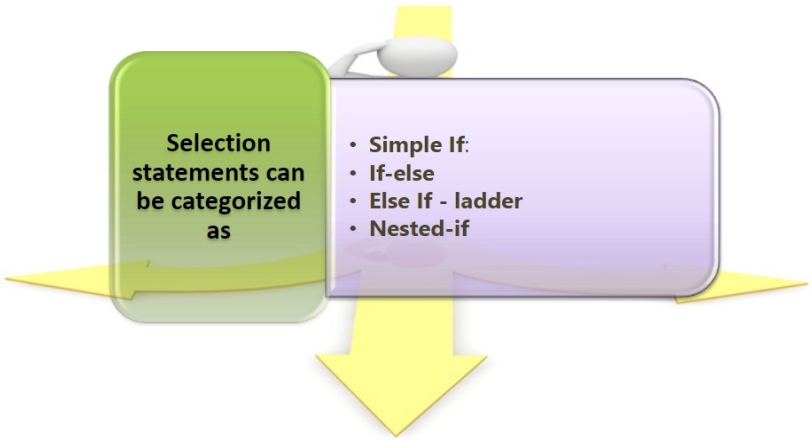
TEKNOTURF



It might be necessary to make a decision before arriving at a conclusion or to go on to the next step of processing in many situations.

SELECTION OR CONDITIONAL STATEMENTS

TEKNOTURF

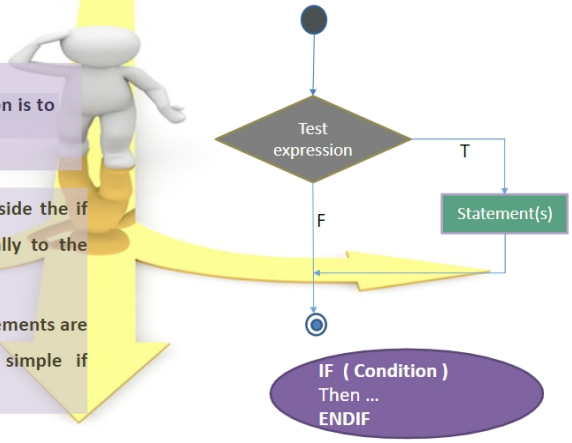


Selection statements can be categorized as

- Simple If
- If-else
- Else If - ladder
- Nested-if

SELECTION OR CONDITIONAL STATEMENTS-SIMPLE IF

Decides the sequence of execution of instructions based on the condition enclosed in simple if statement



Used for branching when a single condition is to be checked

- If the condition is true, the statements inside the if block are executed and moves sequentially to the next statement
- If the condition is false, the true block statements are skipped and the next statement after simple if statement gets executed

IF (Condition)
Then ...
ENDIF

ALGORITHM FOR SIMPLE-IF

Sample algorithm to print the square of a number if it is less than 6.

STEP 1:
Start the process

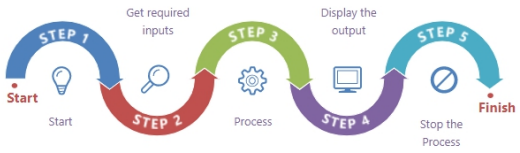
STEP 2:
Get the number

STEP 3:
Check whether number is less than 6, if YES GOTO step 4, if NO GOTO step 6

STEP 4:
Squares the number and store the result in variable square

STEP 5:
Display the square value

STEP 6:
Stop the process

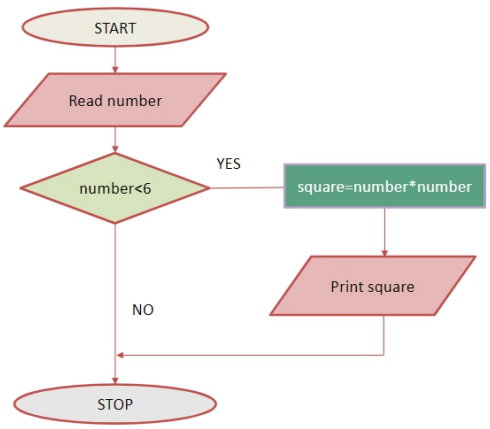


Example algorithm for Simple if logic

GOTO statement transfers control of execution to another statement

FLOWCHART FOR SIMPLE-IF

Basic Flowchart representation to print the square of a number if it is less than 6



(Start the process)

(Let the variable be a number and the value for num=5)

(This step checks whether 5 is less than 6. As 5 is less than 6, it squares the number (5*5)

(Displays 25)

(Ends the process)

PSEUDOCODE FOR SIMPLE-IF

Pseudocode representation to print the square of a number if it is less than 6

```
BEGIN
        DECLARE variables number, square
        READ number
        IF number<6
        THEN
                SET square ← number * number
                PRINT square
        ENDIF
END
```





SIMPLE IF –DRY RUN

Manual execution of steps in an algorithm is called as dry run. Sample dry run for a simple if.

Sample number be 5

Step no	number	number<6	square ← number * number	output
1	5			
2		Y	25	
3				25

SIMPLE IF –DRY RUN

Manual execution of steps in an algorithm is called dry run. Sample dry run for a simple if.

Sample number be 5

Step no	number	number<6	square ← number * number	output
1	5			
2		Y	25	
3				25

Sample number be 15

Step no	number	number<6	square ← number * number	output
1	15			
2		N	-	
3				-

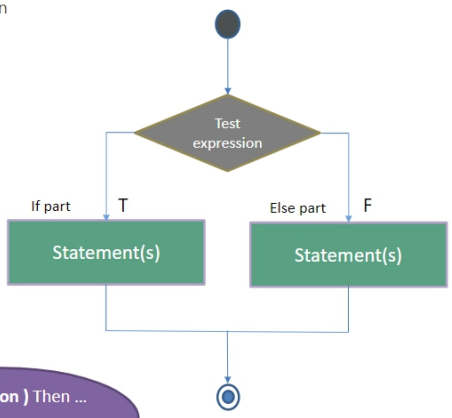
- Condition becomes false and hence no output

SELECTION STATEMENT: IF-ELSE

Decides on the execution of one of the two statements, depending upon the test expression.

Two way branching statement works based on condition

- If the condition is true, the statements inside the if block are executed.
- If the condition is false, if block statements are skipped, and the statements inside else block gets executed



IF (Condition) Then ...
ELSE
ENDIF

ALGORITHM FOR IF-ELSE

Sample algorithm to check whether a number is even or odd

STEP 1:
Start the process

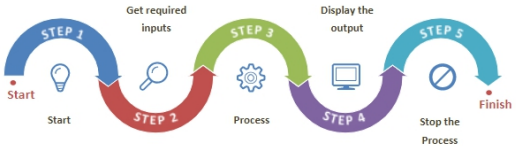
STEP 2:
Get number

STEP 3:
Check whether number mod 2 is equal to 0,if YES GOTO step 4, if NO GOTO step 5

STEP 4:
Display EVEN. GOTO step 6.

STEP 5:
Display ODD

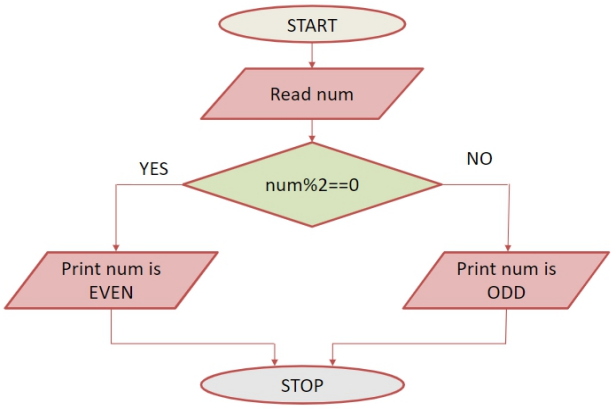
STEP 6:
Stop the process



Example algorithm for If/else logic

FLOWCHART FOR IF-ELSE

Basic Flowchart representation to check if a number is even or odd



(Start the process)

(Let the variable be num and the value for num=5)

(5 is divided by 2 and the remainder is obtained. This step checks whether the remainder value is equal to 0 or not)

(Displays the result as ODD)

(Ends the process)

PSEUDOCODE FOR IF-ELSE

Pseudocode representation to check if a number is even or odd

```
BEGIN
        DECLARE variables number , result
        READ number
                SET result ←— number%2
        IF result==0
        THEN
                PRINT number is EVEN
        ELSE
                PRINT number is ODD
        ENDIF
END
```

IF-ELSE DRY RUN

Sample number be 5

Step no	Number	Result ←—Number%2	Result	Result==0?	OUTPUT
1	5				
2		1	1		
3				N	
4					ODD

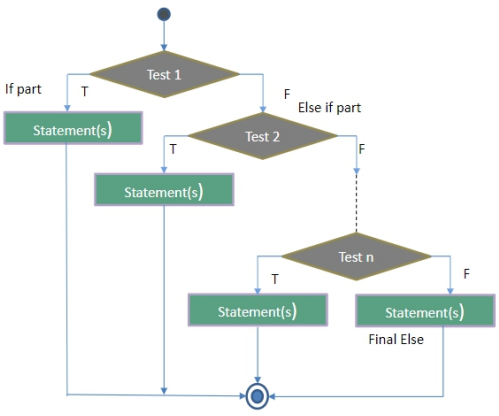
Condition becomes false and hence prints the number as ODD

SELECTION STATEMENT: ELSE-IF LADDER

Decides the execution of statements, when multiple conditions are to be checked

Used for branching when multiple test expressions are to be checked

- Conditions are checked one by one. If any of the conditions is evaluated to be true, that block alone gets executed and the other block statements are ignored.
- If all the conditions are evaluated to false, the default else part gets executed



ALGORITHM FOR ELSE-IF LADDER

Sample algorithm to generate grade report

STEP 1:
Start the process

STEP 2:
Read average

STEP 3:
If average is greater than or equal to 9,grade is set as "A' and GOTO step 7

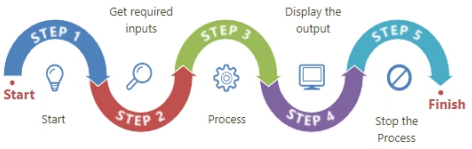
STEP 4:
If average is greater than or equal to 7,grade is set as "B' and GOTO step 7

STEP 5:
If average is greater than or equal to 5, grade is set as "C' and GOTO step 7

STEP 6:
grade is set as 'D'

STEP 7:
Print grade

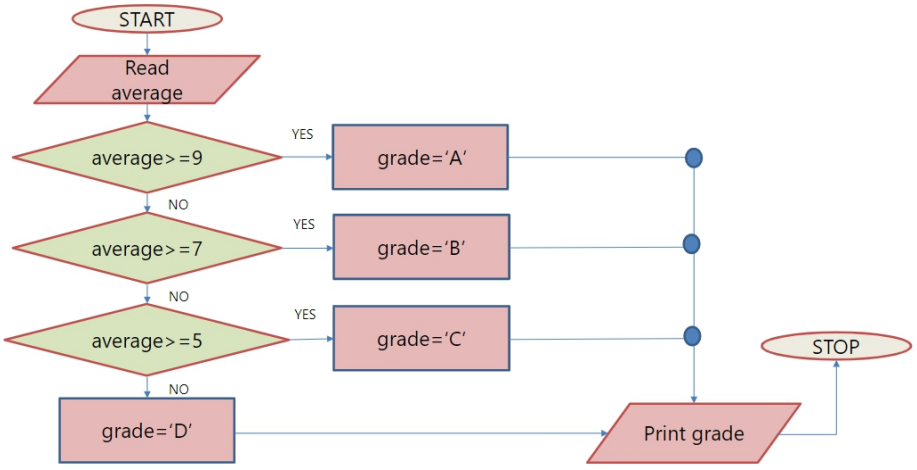
STEP 8:
Stop the process



Example algorithm for Else if ladder logic

FLOWCHART FOR ELSE-IF LADDER

Basic Flowchart to generate grade report



PSEUDOCODE FOR ELSE-IF LADDER

Pseudocode representation to generate grade report

```
BEGIN
    DECLARE variables average , grade
    READ average
      IF average>=9
        SET grade='A'
      ELSEIF average>=7
        SET grade='B'
      ELSEIF average>=5
        SET grade='C'
      ELSE
        SET grade='D'
    PRINT grade
        ENDIF
END
```

SELECTION –DRY RUN

Sample average value be 6

Step no	average	average>=9?	average>=7?	average>=5?	grade	output
1	6					
2		N				
3			N			
4				Y	C	
5						C

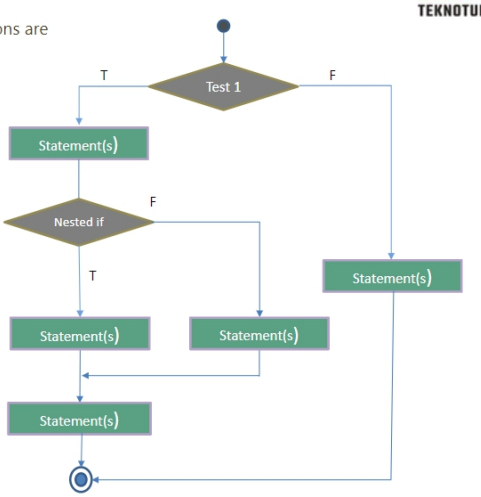
In step 4,condition becomes true and hence grade is set to C and prints the grade as C.

SELECTION STATEMENT: NESTED-IF

Nested if statements are used if there is one or more true conditions are to be checked

Used, if there is a sub condition to be tested inside another condition

- If else statement present inside the body of another "if" or "else".
- Decisions can be nested to any level



WORKING OF NESTED-IF

Nesting of if-else statements

- Outer condition is checked first.
- If the outer expression is evaluated to be true, the program flow gets inside the block and tests for the inner condition.

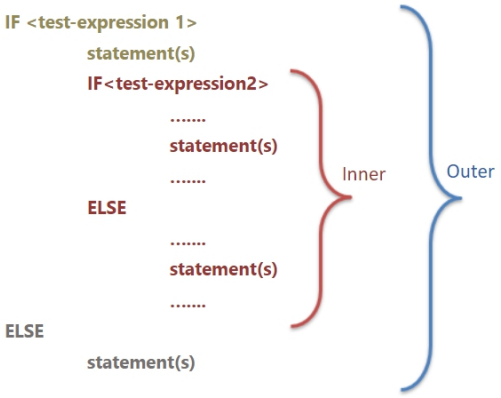
```
IF <test-expression 1>
        statement(s)
        .....
        ......
        ......
        .....
        ......
ELSE
        statement(s)
```

Outer

WORKING OF NESTED-IF

Nesting of if-else statements

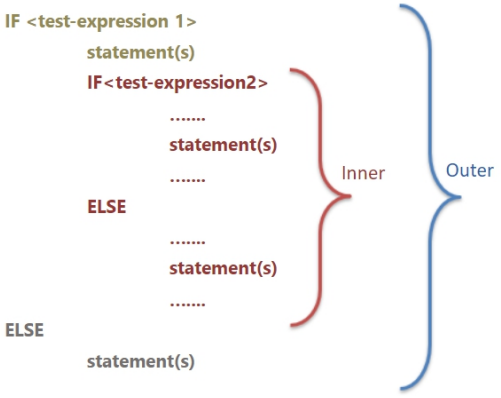
- Outer condition is checked first.
- If the outer expression is evaluated to be true, the program flow gets inside the block and tests for the inner condition.
  - If the inner condition is true, it executes the block of statements associated with it.
  - If the inner condition is false, it moves to the else part of the inner condition



WORKING OF NESTED-IF

Nesting of if-else statements

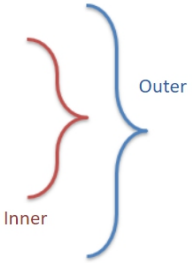
- Outer condition is checked first.
- If the outer expression is evaluated to be true, the program flow gets inside the block and tests for the inner condition.
  - If the inner condition is true, it executes the block of statements associated with it.
  - If the inner condition is false, it moves to the else part of the inner condition
- If the outer condition is false, it skips the outer condition part and moves to the else part of the outer condition.



NESTED-IF -EXAMPLE

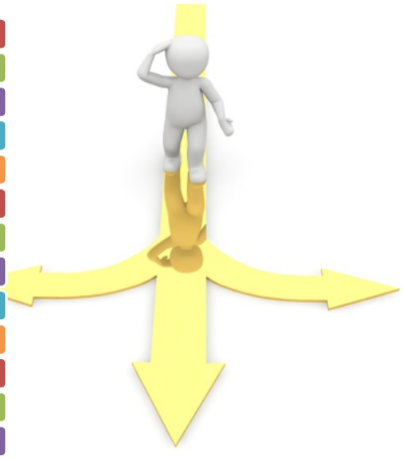
Sample Pseudo code to find the Profit and Loss

```
BEGIN
DECLARE variables cost_price, selling_price
INPUT cost_price, selling_price
IF (cost_price>0) AND (selling_price>0) THEN
        IF cost_price > selling_price THEN
                PRINT "You have Loss"
        ELSE IF cost_price < selling_price THEN
                PRINT "You have Profit"
        ELSE
                PRINT "No profit, No loss"
        END IF
ELSE
        PRINT "Invalid Price"
END IF
END
```



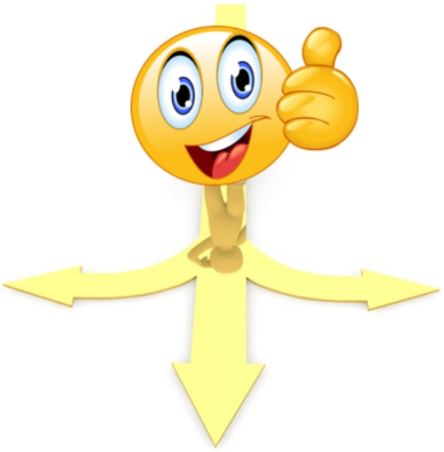
Time To think

```
Input the age
Declare variable age, registration_status
IF age>= 18THEN
PRINT “You may vote”
ELSE
PRINT “You must register to vote”
IF registration_status==‘YES’ THEN
ENDIF
PRINT “You are not Eligible to vote”
ELSE
ENDIF
BEGIN
END
```



Time To think

```
BEGIN
Declare variable age, registration_status
Input the age
IF age>= 18THEN
IF registration_status==‘YES’ THEN
PRINT “You may vote”
ELSE
PRINT “You must register to vote”
ENDIF
ELSE
PRINT “You are not Eligible to vote”
ENDIF
END
```



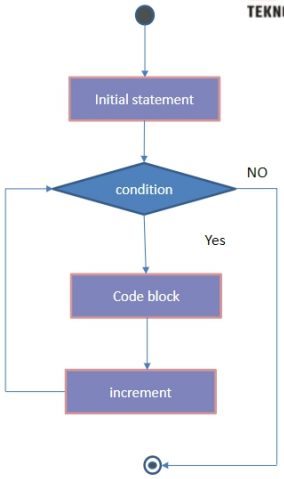
Logic Development- Looping statements

LOOPING OR ITERATION

FOR-LOOP STATEMENT

Repeats a statement or sequence of statements multiple times.
Iterates the sequence of statements until the condition becomes false.

- Code may be executed 0 number of times or as many times as per the required condition.
- For loop is Entry Controlled - condition is checked before entering the block of statements .



Used to execute block of statements, when the number of iterations are known

FOR (specific condition)
DO
.....
END FOR

ALGORITHM FOR - FOR LOOP LOGIC

Sample algorithm for printing the sum of marks of 4 subjects

STEP 1: Start the process

STEP 2: Let i=0,sum=0

STEP 3: Check whether i<4 , if YES GOTO step 4, if NO GOTO step 7

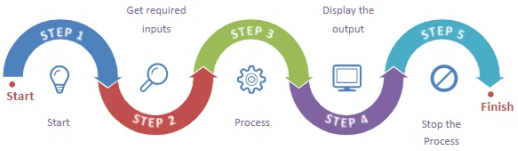
STEP 4: Read number

STEP 5: Add sum and the number, store the result in sum

STEP 6: Increment the i value by 1, GOTO step 3

STEP 7: Print sum

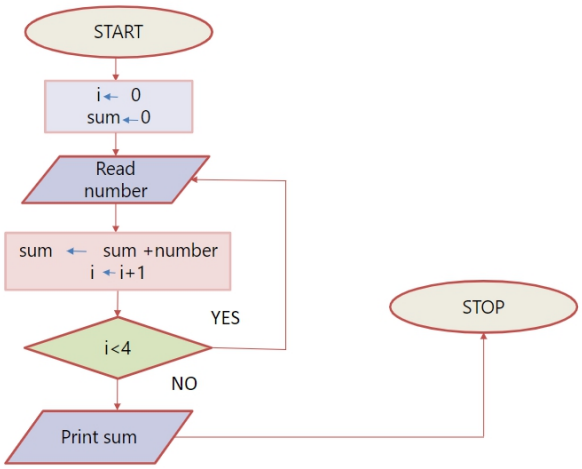
STEP 8: Stop the process



In this problem scenario the exact number of iterations for execution is known. Loop has to continue for the given number of times i.e., 4 times until it meets the condition . Hence FOR loop is used.

FLOWCHART FOR -FOR LOOP LOGIC

Basic flowchart for printing the sum of marks of 4 subjects



PSEUDOCODE FOR -FOR LOOP LOGIC

Pseudocode representation for printing the sum of marks of 4 subjects

```
BEGIN
    DECLARE variables i , sum, number
        SET i=0,sum=0
    FOR i=0 to 3 do
        READ number
        sum ← sum +number
        i=i+1
    END FOR
    PRINT sum
END
```

FOR LOOP –DRY RUN

Step no	i	Sum	Condition satisfied(Y/N) i<4	Read number	sum = sum+ number	i=i+1	Output
1	0	0					
2	0	0	Y				
3				70			
4					0+70=70		
5						0+1=1	
2	1	70	Y				
3				60			
4					70+60=130		
5						1+1=2	

2,3,4,5 steps are the block of statements which repeats 4 times for getting the marks of the four subjects

FOR LOOP–DRY RUN

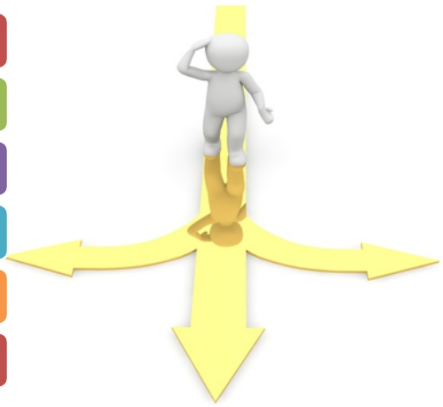
Step no	i	Sum	Condition satisfied(Y/N) i<4	Read number	sum = sum+ number	i=i+1	Output
2	2	130	Y				
3				80			
4					130+80=210		
5						2+1=3	
2	3	210	Y				
3				75			
4					210+75=285		
5						3+1=4	
2	4	285	N				
7							Prints 285

When condition fails, the loop gets terminated and it moves to step 7,where it prints the sum value as 285.

TIME TO THINK

Pseudocode to Print the numbers from 1 to 10

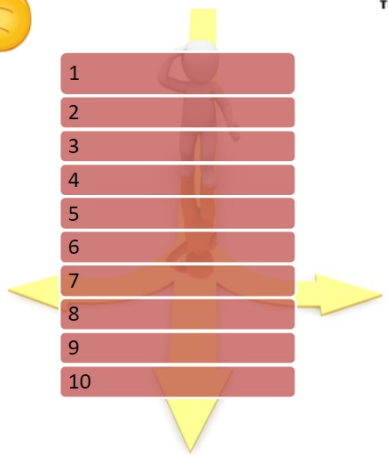
```
DECLARE variable number
BEGIN
PRINT number
END
FOR number=0 to 9 THEN
END FOR
```



TIME TO THINK

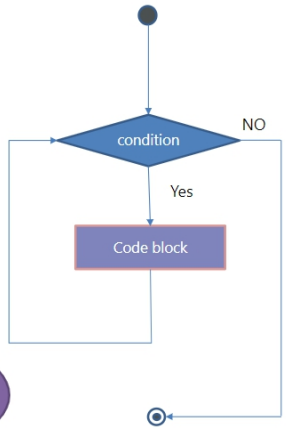
Pseudocode to print the numbers from 1 to 10

```
BEGIN
Declare variable number
FOR number=0 to 9 THEN
PRIINT number+1
END FOR
END
```



WHILE-LOOP STATEMENT

Repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.



- Code may be executed 0 number of times or as many times as per the required condition.
- While loop is also entry controlled as the conditions are checked before entering the block of statements.

Used to execute block of statements, when the number of iterations are not known

WHILE (condition)
DO
.....
END WHILE

ALGORITHM FOR –WHILE LOOP

Sample algorithm for accepting the numbers, printing it only if the number is greater than 0. Loop continues until a negative number is given.

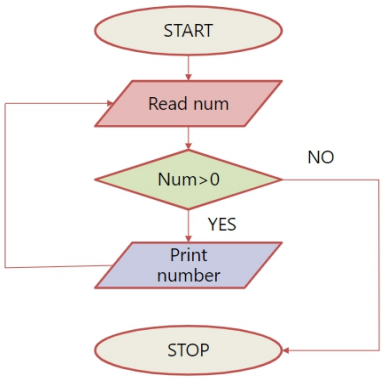
- **STEP 1:** Start the process
- **STEP 2:** Get the number
- **STEP 3:** Check whether number>0,if YES GOTO step 4, if NO GOTO step 5
- **STEP 4:** Display the number. GOTO Step 2
- **STEP 5:** Stop the process

Example algorithm for while-loop logic

- In this problem scenario the exact number of iterations for the execution is unknown. Loop has to continue until the user enters the valid numbers which meets the condition. Hence the logic of WHILE loop is used.

FLOWCHART FOR – WHILE LOOP

Basic flowchart for accepting the numbers and printing it only if the number is greater than 0



(Start the process)

(Let the variable be num and the value for num=5)

(checks if num is greater than 0. Here 5 is greater than 0.)

(Prints 5 and the loop continues.. until user enters a negative value)

(End the process)

PSEUDOCODE FOR –WHILE LOOP

Pseudocode representation for accepting the numbers and printing it only if the number is greater than 0.

```
BEGIN
        DECLARE variable number
        READ number
        WHILE number>0
                PRINT number
                READ number
        END WHILE
END
```





WHILE LOOP –DRY RUN

Step no	Read number	Condition satisfied(Y/N) number<6	Output
1	1		
2		Y	
3			Prints 1
1	2		
2		Y	
3			Prints 2

The numbers 1 and 2 gets displayed as the numbers satisfies the condition that it should be greater than 0. That's why the numbers 1 and 2 gets printed as an output.

WHILE LOOP –DRY RUN

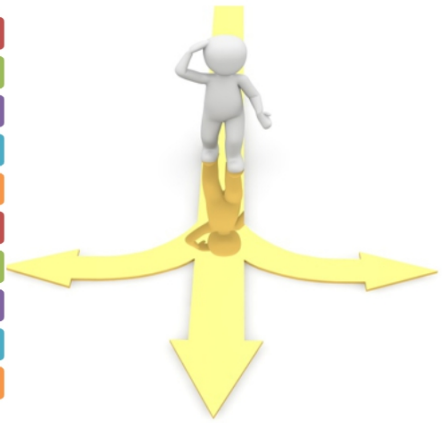
Step no	Read number	Condition satisfied(Y/N) number<6	Output
1	3		
2		Y	
3			Prints 3
1	-7		
2		N	
5	-	-	-

The number 3 get displayed but when the number is less than 0, i.e -7, in this case the loop get terminated and goes to the end of the process.

TIME TO THINK

Pseudocode to calculate the sum of digits

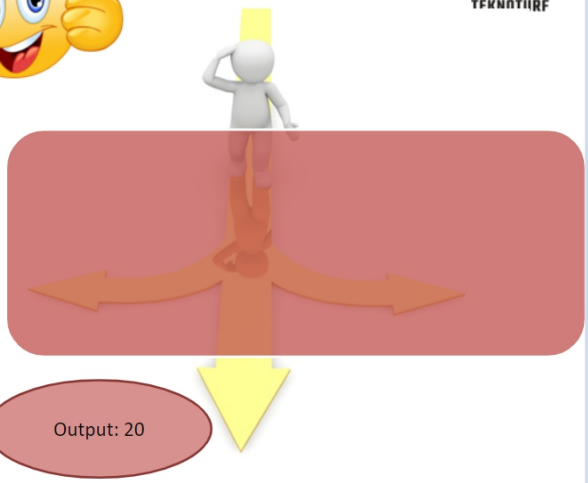
```
DECLARE variables number, sum
BEGIN
PRINT sum
SET sum=0
WHILE number != 0
Sum= sum+(number%10)
Number=number%10
READ number
END WHILE
END
```



TIME TO THINK

Pseudocode to calculate the sum of digits

```
BEGIN
DECLARE number, sum
SET sum=0
READ number
WHILE number != 0
Sum= sum+(number%10)
Number=number%10
END WHILE
PRINT SUM
END
```

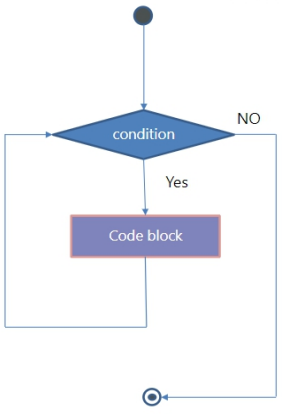


Output: 20

DO-WHILE -LOOP STATEMENT

More like a while statement.
Except that it tests the condition at the end of the loop, after executing the block of statements at least once.

- Code may be executed at least once or as many times as per the required condition.
- Do-While loop is exit controlled as condition is checked after entering the block of statements.




```
DO
.... ...
UNTIL(Condition)
```

ALGORITHM- DO WHILE LOOP

Sample algorithm for accepting the numbers and printing it at least once before checking the condition

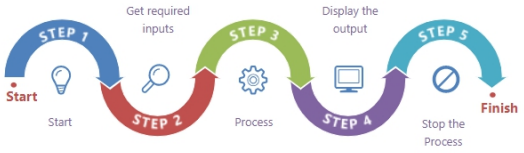
STEP 1:
Start the process

STEP 2:
Get the number

STEP 3:
Display the number. GOTO Step 4

STEP 4:
Check whether number<6,if YES GOTO step 2, if NO GOTO step 5

STEP 5:
Stop the process

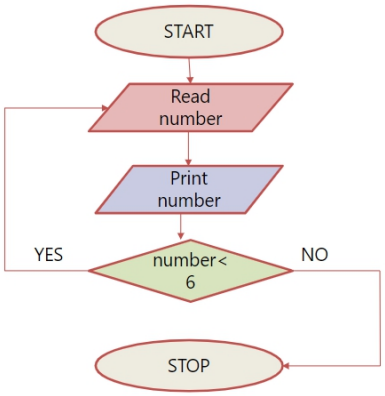


Example algorithm for
Do while-loop logic

In this problem scenario the statements executes at least once, before checking the actual condition in the loop. Loop has to continue until the user enters the valid numbers which meets the condition. Hence the logic of DO WHILE loop is used.

FLOWCHART FOR – DO WHILE LOOP

Basic flowchart for accepting the numbers and printing it at least once before checking the condition



(Start the process)

(Let the variable be number and the value for number=5)

(Prints 5)

(checks if number is less than 6. Here 5 is less than 6. and the loop continues..)

(End the process)

PSEUDOCODE FOR –DO WHILE LOOP

Pseudocode representation for accepting the numbers and printing it only if the number is less than 6.

```
BEGIN
    DECLARE variable number
    DO
    READ number
    PRINT number
    UNTIL number<6
END
```

Nested-LOOP Statement

TEKNOTURF

Used when one or more loops are needed inside another looping statements like while, for, or do.. while .
Loops can be nested to any level.

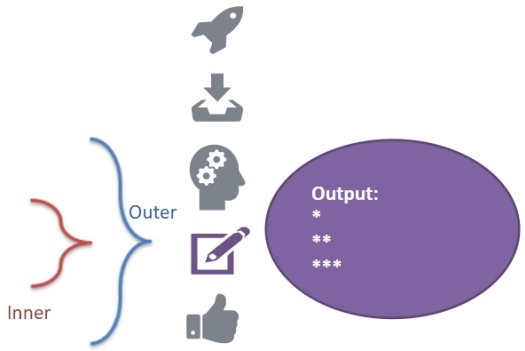
- A final note on loop nesting is that you can put any type of loop inside any other type of loop. For example, a 'for' loop can be inside a 'while' loop or vice versa.
- How does this work - the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again.

PSEUDOCODE FOR –NESTED FOR LOOP

TEKNOTURF

Pseudocode representation for Printing pattern

```
BEGIN

    DECLARE variables i, j, size

    SET size=3

    FOR   i=0 to size   DO
          FOR   j=0 to  (i+1)   DO
                PRINT "*"
          END  FOR
    PRINT newline     // Skips to new line

    END  FOR

END
```



NESTED FOR LOOP –DRY RUN

TEKNOTURF

Let the size be 3

Step no	i	Condition satisfied(Y/N) i<size	j	Condition satisfied(Y/N) j<i+1	Output	j=j+1	i=i+1
1	0	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	N			
5					Prints newline		i=0+1
1	1	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	Y			
3					Prints *	j=1+1	
4			2	N			

NESTED FOR LOOP –DRY RUN

Let the size be 3

Step no	i	Condition satisfied(Y/N) i<size	j	Condition satisfied(Y/N) j<i+1	Output	j=j+1	i=i+1
5					Prints newline		i=1+1
1	2	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	Y			
3					Prints *	j=1+1	
4			2	Y			
3					Prints *	j=2+1	
4			3	N			
5	3	N					

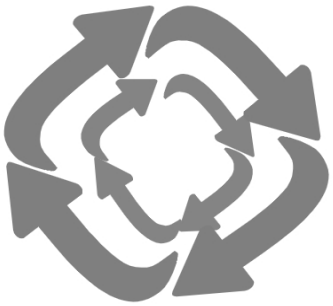
NESTED FOR LOOP –DRY RUN

Let the size be 3

Step no	i	Condition satisfied(Y/N) i<size	j	Condition satisfied(Y/N) j<i+1	Output	j=j+1	i=i+1
4			2	N			
5					Prints newline		i=1+1
1	2	Y					
2			0	Y			
3					Prints *	j=0+1	
4			1	Y			
3					Prints *	j=1+1	
4			2	Y			
3					Prints *	j=2+1	
4			3	N			

Nested-LOOP Statement

- Outer loops executes first, based on the condition.
- Outer loop triggers the inner loop to execute completely until the condition fails in the inner loop.
- After which it again comes back to the outer loop for a second outer iteration and triggers the inner loop again .
- This process continues until the condition fails in the outer loop.



Looping structures

- Initial conditions that need to be applied before the loop begins to execute
- The invariant relation that must be applied after each iteration of the loop
- The condition under which the iterative process must be terminated

Logic Development - Arrays

INTRODUCTION TO ARRAYS

TEKNOTURF

An array is a variable name which is used to store large amount of data

An array can be considered as a container with equally spaced slots, where the data could be stored

Data would be stored continuously, and the type of data would be the same

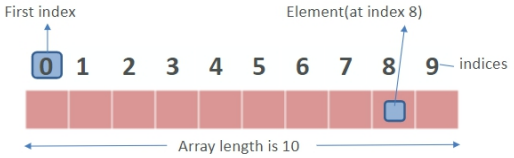
ONE-DIMENSIONAL ARRAYS

TEKNOTURF

1-d arrays are arrays treated as a linear list of values

Elements are stored sequentially and can be accessed using INDEX of the array

All locations in an array are numbered from 0 to n-1, where n is the total number of elements.



- Assume that an array variable marks[10] is created for storing marks of 10 subjects,
- To store a mark value 93 at the 1st position, the assignment of value would be marks[0]←93
- To access location 1, we would use notation as array_variable_name[index] i.e., marks[0]

Accessing values in 1d array

SAMPLE ALGORITHM FOR 1-D ARRAY

TEKNOTURF

To read the marks of 5 students in a group and print the average for that group. Assumption: marks is an array of 5 marks.

STEP 1:
Start the process. Create array as marks[5]

STEP 2:
Use variables index, average, sum. Set the index value as 0, sum value as 0. average value as 0.

STEP 3:
Check whether i<5,if YES GOTO step 4, if NO GOTO step 7 .

STEP 4:
Read marks[index].

STEP 5:
ADD sum and marks[index] and store the result in sum.

STEP 6:
Increment the index value by 1. GOTO step 3

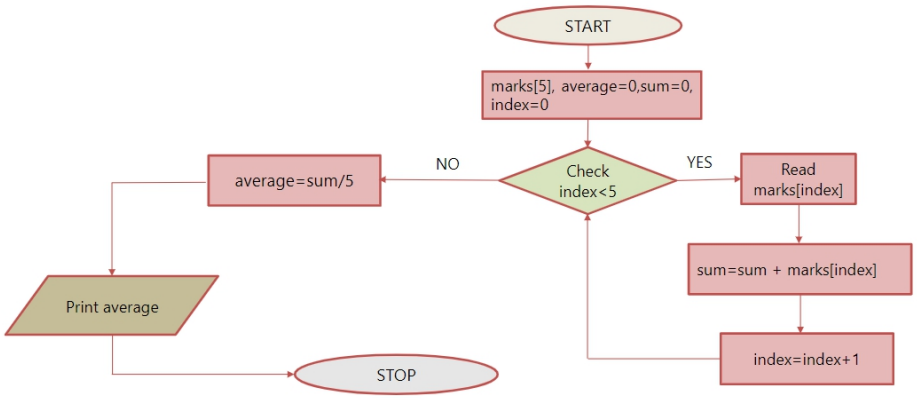
STEP 7:
Divide the sum value by 5 and store the result in average.

STEP 8:
Display the average value

STEP 9:
Stop the process

FLOWCHART FOR 1-D ARRAY

Basic flowchart to read the marks of 5 students and print the average for that group. Assumption: marks is an array of 5 marks



SAMPLE PSEUDOCODE FOR 1-D ARRAY

To read the marks of 5 students and print the average. Assumption: marks is an array of 5 marks

```
BEGIN
    DECLARE variables index , sum, average, marks[5]
        SET index=0,sum=0
    FOR index ← 0 to 4 do
        READ marks[index]
        sum ← sum + marks[index]
        i ← i+1
    END FOR
    AVERAGE ← sum/5
    PRINT average
END
```

1D ARRAY USING FOR LOOP–DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
1	0	0						
2	0	0	Y					
3				98				
4					0+98=98			
5						0+1=1		
2	1	98	Y					
3				87				
4					98+87=185			
5						1+1=2		

Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

1D ARRAY USING FOR LOOP–DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
2	2	185	Y					
3				95				
4					185+95=280			
5						2+1=3		
2	3	280	Y					
3				87				
4					280+87=367			
5						3+1=4		

Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

1D ARRAY USING FOR LOOP–DRY RUN

Step no	index	Sum	Condition satisfied(Y/N) i<5	Read marks[index]	sum = sum+ marks[index]	i=i+1	Average = sum/5	Output
2	4	367	Y					
3				88				
4					367+88=455			
5						4+1=5		
2	5	455	N					
7							91	
8								Prints 91

Step no. 2,3,4 and 5 are the block of statements which repeats for getting the marks of five subjects.

PSEUDOCODE FOR 1-D ARRAY- FINDING MAXIMUM

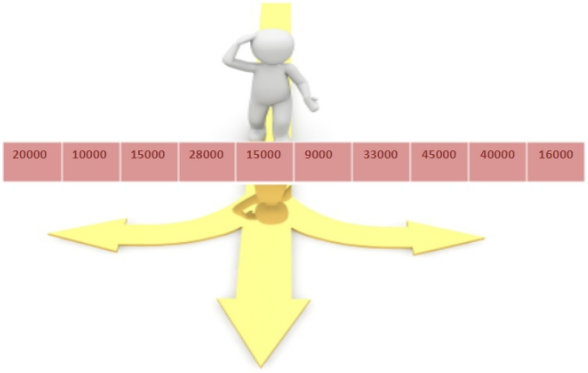
Assume the size of an array is 10. Given Pseudocode To find the maximum element in an array of 10 elements

```
BEGIN
        DECLARE variables index , maximum, array[10]
          FOR  index ← 0 to 9  DO
                READ array[index]
          END FOR
           SET maximum ← array[0]
        FOR  index ← 0 to 9  DO
          IF maximum < array[index]
                maximum ← array [index]
          END IF
        END FOR
          PRINT maximum
END
```

TIME TO THINK

Assume the size of an array is 10.

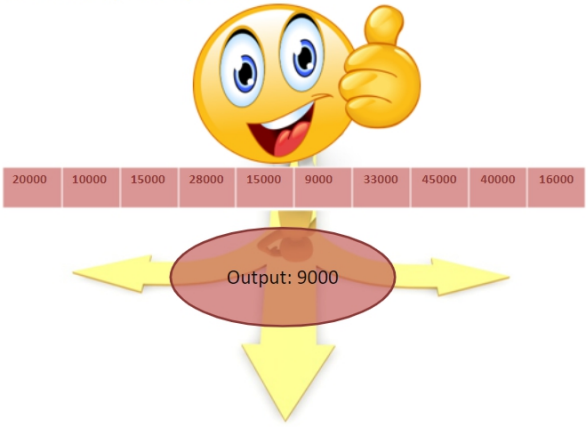
```
DECLARE variables index , minimum_salary, array[10]
BEGIN
PRINT minimum_salary
SET minimum _salary← array[0]
FOR index ← 0 to 9 DO
IF minimum_salary > array[index]
FOR index ← 0 to 9 DO
READ array[index]
END FOR
Minimum_salary ← array [index]
END FOR
END
END IF
```



TIME TO THINK

Assume the size of an array is 10. Given Pseudocode To find the minimum salary in an array of 10 elements

```
BEGIN
DECLARE variables index , minimum_salary, array[10]
FOR index ← 0 to 9 DO
READ array[index]
END FOR
SET minimum_salary← array[0]
FOR index ← 0 to 9 DO
IF minimum_salary > array[index]
minimum_salary← array [index]
END IF
END FOR
PRINT minimum_salary
END
```



ARRAYS -EXAMPLE

Two Arrays are said to be compatible

- if they are of the same size
- if the i'th element in the first array is greater than or equal to the i'th element in the second array for all i values.

Array 1	2	3	8	6	1
Array 2	1	1	1	1	1

- ✓ Size of array 1 is equal to size of array 2
- ✓ 1st column : 2 > 1
- ✓ 2nd column : 3 > 1
- ✓ 3rd column : 8 > 1
- ✓ 4th column : 5 > 1
- ✓ 5th column : 1>=1

Compatible

PSEUDOCODE FOR COMPATIBLE ARRAYS

Assume the size of two arrays is 15.

```
BEGIN
DECLARE variables i , number, a[15], b[15], flag
GET number
SET flag=0
FOR i ← 0 to number-1 DO
        READ a[i]
END FOR
FOR i ← 0 to number-1 DO
        READ b[i]
END FOR
FOR index ← 0 to number-1 DO
        IF a[i] < b[i]
                SET flag=1
                BREAK
        END IF
END FOR
IF flag==0
        PRINT Compatible
ELSE
        PRINT Incompatible
END
```

2-D Array

The two-dimensional array is treated like a matrix.

The values are arranged in rows and columns.

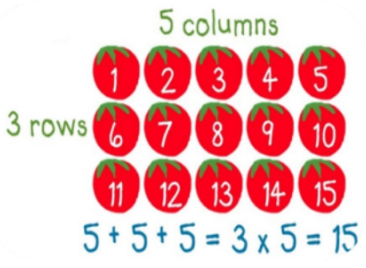
An array which has 3 rows and 4 columns

	0	1	2	3
0	45	22	-5	11
1	-1	45	89	23
2	0	34	90	76

2-D Array

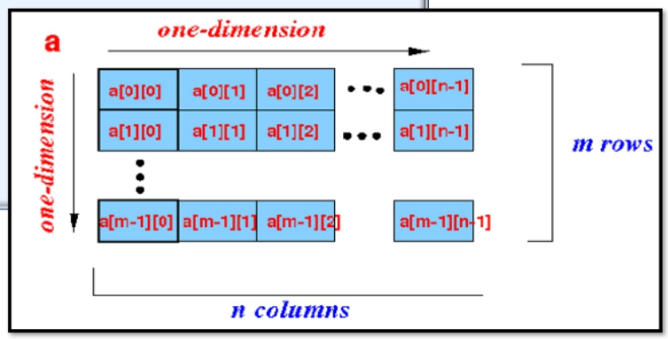
Pseudo code for reading the marks of 3 students for 5 subjects and printing the average of each student

```
BEGIN
DECLARE student_marks[3][5]
SET average TO 0, sum TO 0
FOR student_index=0 TO 2
FOR mark_index=0 TO 4
INPUT mark
SET student_marks[student_index][mark_index] TO mark
NEXT mark_index
NEXT student_index
```



(contd...)

```
FOR student_index= 0 TO 2
FOR mark_index=0 TO 4
SET sum TO sum+ student_marks[student_index][mark_index]
NEXT mark_index
SET average TO sum/5
PRINT average
NEXT student_index
END
```



Manipulating Arrays

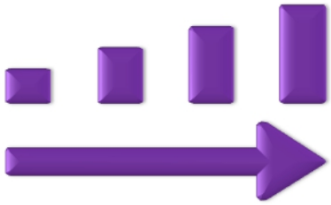
WHAT IS SEARCHING?

Process of finding a particular item in a collection of items

- A search typically answers either True or False depending on whether the item is present or not.
- Search operation can be done using searching algorithms

SEARCHING

If an element is to be found from a specific location in an array, the array has to be traversed from the first position until the element is found.



- It sequentially scans the array , comparing each array item with the search element.
- If search element is found in an array then the index of that element is returned.

ALGORITHM FOR SEARCHING

STEP 1:
Start the process

STEP 2:
Create an array variable employee_id[8], i , n

STEP 3:
Set i value as 0.

STEP 4:
Check whether i<5,if YES GOTO step 5 and if NO GOTO step 14.

STEP 5:
Read employee_id[i]

STEP 6:
Increment i value by 1, and GOTO 4

STEP 7:
Get n from user (number to be searched)

STEP 8:
Get n from user (number to be searched)

STEP 9:
Set i value as 0 and Check whether i<5,if YES GOTO step 10, if NO GOTO step 13

STEP 10:
Compare n value with employee_id[i], if Equal GOTO step 11 and not EQUAL GOTO step 12

ALGORITHM FOR SEARCHING

STEP 11:
Display "Number is present". GOTO step 14

STEP 12:
Increment i value by 1 and GOTO step 9

STEP 13:
.Display "Number is not present in an array"

STEP 14:
Stop the process.

3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8

SORTING

Sorting is the process of placing elements from a collection in some kind of order

- Ordering of data allows for easy and faster access of data

Eg:

- Sorting names in telephone directory
- Sorting list of cities by using population

Sorting - Activities

A sort repeatedly compares adjacent elements of an array

Go through multiple passes over the array.

In every pass:

- Compare adjacent elements in the list
- Exchange the elements if they are out of order
- Each pass moves the largest elements to the end or smallest element to the beginning.

After every pass, all elements after the last swap are sorted and do not need to be checked again, thereby skipping to track swapped variables.

Repeating this process in several passes eventually sorts the array into ascending (or descending) order.

Sorting- Algorithm

Let A be a linear array of n numbers. Swap is a temporary variable for swapping (or interchange) the position of the numbers

1. Input n numbers of an array A
2. Initialize i = 0 and repeat through step 4 if (i < n)
3. Initialize j = 0 and repeat through step 4 if (j < n – i – 1)
4. If (A[j] > A[j + 1])
   (a) Swap = A[j]
   (b) A[j] = A[j + 1]
   (c) A[j + 1] = Swap
5. Display the sorted numbers of array A
6. Exit.