PERL

Day 1

Agenda

- Introduction
- Modes of Writing a Perl Code
- Fundamentals
- Data Types & Variables
- Operators
- Input and Output
- Decision Making

Introduction

Introduction

- Perl is a general purpose, high level interpreted and dynamic programming language.
- Perl stands for **Practical Extraction and Report Language**.
- Perl was originally developed for the text processing like extracting the required information from a specified text file and for converting the text file into a different form.
- Perl supports both the **procedural** and **Object-Oriented programming**.
- Perl is a lot similar to C syntactically and is easy for the users who have knowledge of C, C++.

PERL History

- It was developed by **Larry Wall**, in **1987**.
- It all started when Larry Wall was working on a task to generate the reports from a lot of text files which have cross-references.
- Then he started to use **awk** for this task but soon he found that it is not sufficient for this task. So instead of writing a utility for this task, he wrote a new language i.e. Perl and also wrote the interpreter for it.
- He wrote the language Perl in **C** and some of the concepts are taken from **awk**, **sed**, and **LISP** etc.

PERL History (cont’d)

- At the beginning level, Perl was developed only for the **system management** and **text handling** but in later versions, Perl got the ability to handle regular expressions, and network sockets etc.
- The first version of Perl was 1.0 which released on December 18, 1987.

Why PERL?

- **Easy to start:**

  Perl is a high-level language so it is closer to other popular programming languages like C, C++ and thus, becomes easy to learn for anyone.

- **Text-Processing:**

  Perl has the high text manipulation abilities by which it can generate reports from different text files easily. Also, it can convert the files into some another form.

- **Contained best Features:**

  Perl contains the features of different languages like C, sed, awk, and sh etc.

Why PERL? (cont’d)

- **System Administration:**

  Due to having the different scripting languages capabilities Perl make the task of system administration very easy.

- **Web and Perl:**

  Perl can be embedded into web servers to increase its processing power and it has the DBI package, which makes web-database integration very easy.

Modes of Writing a Perl Code

Modes of Writing a Perl Code

- Perl is a **free-form language** which means it can be written, formatted and indented as per user's requirement.
- These modes can be categorized on the basis of their writing compatibility and mode of execution in the following ways:
  1- Interactive Mode
  2- Script Mode
  3- One-Liner Mode

1. Interactive Mode

- Interactive mode of writing a Perl code means the direct interaction with the interpreter.
- Interactive mode is a good way to get started as it helps to check the flow of code line by line and makes the debugging process easier.
- This interpreter is commonly known as **REPL– Read, Evaluate, Print, Loop**.
- Interactive mode provides an instant development and execution of code without the need to create a temporary file to store the source code.
- In the Interactive mode of Writing a Perl Code, the user has to write the code line by line and it gets executed at the same time.
- Can be entered by writing **“perl -de1”** in terminal.

2. Script Mode

- Script mode in Perl can be used by the help of a text editor to write the Perl program and to save it in a file called a script and then execute the saved file by using the command line.
- This file must be saved with a **.pl** extension and should be placed in the same folder of which the directory path is given to the command line.
- **perl File_Name.pl**

3. One-liner Mode

- Perl also provides a one-liner mode, which allows to type and execute a **very short script** of code directly on the command line.
- This is done to avoid the creation of Files to store the script for codes which are not very lengthy.
- **perl -e “ code goes here”;**
- The -e flag in the up given command tells the compiler that the script of the code is not stored in any kind of file but is written in the double codes immediately after this flag.
- These one-liners can be very useful for making changes **very quickly like finding information, changing file contents, etc.**

Fundamentals

Expressions

- An expression in Perl is something that returns a value on evaluating.
- Value 10 is an expression, $x + $y is an expression that returns their sum.
- Value 1 is an expression, $x > $y is an expression that returns their comparing result.

Statements

- A statement in Perl holds instructions for the compiler to perform operations.
- These statements perform the operations on the variables and values during the Run-time.
- Every statement in Perl must end with a semicolon(;)
- **Multi-Line Statements:**

  Statements in Perl can be extended to one or more lines by simply dividing it into parts.
- $x = $a + $b + $c + $d + $e + $f;

Block

- A block is a group of statements that are used to perform a relative operation.
- In Perl, multiple statements can be executed simultaneously (under a single condition or loop) by using curly-braces ({}).
- This forms a block of statements which gets executed simultaneously.

Comments

- Comments are used for enhancing the readability of the code. The interpreter will ignore the comment entries and does not execute them.
- **Single line Comment:**
  #This is a single line comment
- **Multi-line comment:**
  =begin line comments
  Line start from = is interpreted as the
  starting of multiline comment and =cut is
  consider as the end of multiline comment
  =cut

Hello World Program

```
#!/usr/bin/perl
use strict;          #Pragma
use warnings;    #Pragma
print "Hello World !\n";
```

- A pragma is a specific module in Perl package which has the control over some functions of the compile time or Run time behavior of Perl, which is strict or warnings

A-01-hello-world.pl

Data Types & Variables

Data Types

There are 3 data types in Perl as follows:

- Scalars
- Arrays
- Hashes (Associative Arrays)

A-02-variables.pl

Variables

- Variables in Perl are used to store and manipulate data throughout the program. When a variable is created it occupies memory space.
- Variables in Perl are case sensitive.
- It starts with $, @ or % as per the data type required, followed by zero or more letters, underscores and digits.
- The data type of a variable helps the interpreter to allocate memory and decide what to be stored in the reserved memory.

Declaration of a Scalar

1 - **Scalar Variables:** It contains a single string or numeric value. It starts with $ symbol.

Syntax: $var_name = value;

Example:

```
$item_one = "Hello";
$item_two = 2;
```

Modification of a Scalar

```
$name = "Ziyad";
$name = "Ahmed";


$number = 10;
$number = 20;
```

Declaration of an Array

2 - **Array Variables:**

It contains a randomly ordered set of values. It starts with @ symbol.

Syntax : @var_name = (val1, val2, val3, …..);

Example:

```
@price_list = (70, 30, 40);
@name_list = ("Apple", "Banana", "Guava");
```

A-03-array.pl

Modification of an Array

- An element of an array can be modified by passing the index of that element to the array and defining a new value to it:

```
@array = ("A", "B", "C", "D", "E");
$array[2] = "4";
# This will change the array to,
# @array = ("A", "B", "4", "D", "E");
```

- You can also use negative value to access the array from last.

```
print $array[-1]
```

Declaration of a Hash

3 - **Hash Variables:**

It contains (key, value) pair efficiently accessed per key. It starts with % symbol.

Syntax : %hash = (‘key1’ , ’val1’, ‘key2’, ‘val2’, …);

%hash2 = (‘key1’ => ‘val1’, ‘key2’ => ‘val2’, …);

Example:

```
%item_pairs = ("Apple" , 2, "Banana”, 3);
%pair_random = ("Hi" =>8, "Bye"=>9);
```

Modification of a Hash

- A value in a hash can be modified by using its Key:
  %Hash = ("A", 10, "B", 20, "C", 30)
  $Hash{"B"} = 46;

A-04-hash.pl

Scalar Keyword

- scalar keyword in Perl is used to convert the expression to scalar context.
- This is a forceful evaluation of expression to scalar context even if it works well in list context.
- Syntax:
  scalar expr

A-05-scalar-keyword.pl

Variable Context

- Based on the Context, Perl treats the same variable differently i.e., situation where a variable is being used.
- #!/usr/bin/perl
  # Defining Array variable
  @names = ('XYZ', 'LGH', 'KMR');
  # Assigning values of array variable to another array variable
  @copy = @names;
  # Assigning values of Array variable to a scalar variable
  $size = @names;

A-06-variable-context.pl

Scope of Variables

- The scope of a variable is the part of the program where the variable is accessible.
- In Perl, we can declare either **Global variables** or **Private variables.**
- Private variables are also known as lexical variables.

Global variables

- Global variables can be used inside any function or any block created within the program.
- Global variables can directly use and are accessible from every part of the program.

A-07-global-variable-1.pl
A-08-global-vaiable-2.pl

Private Variables

- Private variables in Perl are defined using **my** keyword before a variable.
- **my** keyword confines variables in a function or block in which it is declared. A block can either be a for loop, while loop or a block of code with curly braces around it.
- The local variable scope is local, its existence lies between those two curly braces (block of code), outside of that block this variable doesn't exist.
- **Note:** When private variables are used within a function or block, then they hide the global variables created with the same name.

A-09-private-variable.pl

Package Variables

- In Perl, we have one more type of scoping called Package Scoping.
- This is used when we need to make variables which can be used exclusively in different namespaces.
- **”main”** is the default namespace in every Perl program. Namespaces in Perl are defined using the package keyword.
- **Our** Keyword in Perl: **“our”** keyword only creates an alias to an existing package variable of the same name.
- **our** keyword allows to use a package variable without qualifying it with the package name, but only within the lexical scope of the **“our”** declaration.

A-10-package-variable-1.pl
A-11-package-variable-2.pl

Boolean Values

Boolean Values

- In most of the programming language **True** and **False** are considered as the boolean values. But Perl does not provide the type boolean for True and False.
- **True Values:**

  Any non-zero number i.e. except zero are True values in the Perl language. String constants like ‘true’, ‘false’, ‘ ‘(string having space as the character), ’00’(2 or more 0 characters) and “0\n”(a zero followed by a newline character in string) etc. also consider true values in Perl.
- **False Values:**

  Empty string or string contains single digit 0 or undef value and zero are considered as the false values in perl.

B-01-conditional.pl
B-02-true.pl
B-03-false.pl

String

String

- A string in Perl is a scalar variable and start with a ($) sign and it can contain alphabets, numbers, special characters.
- The String is defined by the user within a single quote (‘) or double quote (“).

G-01-string-1.pl
G-01-string-2.pl
G-01-string-3.pl

String Function

- length():

  This function is used to find the number of characters in a string. This function returns the length of the string.

- lc():

  This function return the lower case version of a string.

- uc():

  This function return the upper case version of a string.

- index():

  - This method will search a substring from a specified position in a string and returns the position of the first occurrence of the substring in the string.

  - If the position is omitted, it will search from the beginning of the string.

G-04-length.pl
G-05-lc.pl
G-06-uc.pl
G-07-index.pl

Arrays

Array Creation

- Array Creation:
  @arr = (1, 2, 3);
- Array creation using qw function:
  - qw() function is the easiest way to create an array of single-quoted words.
  - It takes an expression as an input and extracts the words separated by a whitespace and then returns a list of those words.
  Syntax:

```
qw (Expression)
qw /Expression/
qw 'Expression'
qw {Expression}
```

D-01-create-array.pl

Sequential Number Arrays

- Perl also provides a shortcut to make a sequential array of numbers or letters.
- It makes out the user's task easy. Using sequential number arrays users can skip out loops and typing each element when counting to 1000 or letters A to Z etc.
- Sequential Number Arrays:
  @array = (1..9); # array with numbers from 1 to 9
  @array = (a..h); # array with letters from a to h

D-02-create-sequential-array.pl

Size of an Array

- The size of an array(physical size of the array) can be found by evaluating the array in scalar context.
- Implicit Scalar Context:
  $size = @array;
- Explicit scalar context using keyword scalar:
  $size = scalar @array;
- Maximum index of array:
  And you can find the maximum index of array by using $#array.
  maximum_index = $#arr;

D-03-size-of-array.pl

Array Slices

- Array slicing is done to access a range of elements in an array in order to ease the process of accessing a multiple number of elements from the array.

- This can be done in two ways:
  1- Passing multiple Index values.
  2- Using Range Operator.

D-04-array-slicing-index.pl
D-05-array-slicing-range.pl

Reverse an array

- Perl has an inbuilt function to reverse an array or a string or a number.

D-06-reverse-array.pl

Sorting of Arrays

- Perl has a built-in sort() function to sort an array of alphabets and numbers.
- Syntax:
  sort @Array
- Sorting of Arrays in Perl can be done in multiple ways:
  1- Use of ASCII values to sort an Array.
  2- Use of Comparison function (cmp).
  3- Alphabetical order of Sorting(Case insensitive).
  4- Sorting of an Array of Numbers

D-07-sort-1-ascii.pl
D-08-sort-2-cmp.pl
D-09-sort-3-cmp-case-sensitive.pl
D-10-sort-4-numbers.pl

join() function

- join() function is used to combine the elements of a List into a single string with the use of a separator provided to separate each element.
- This function returns the joined string.
- Syntax:
  join(Separator, List)
    Separator: provided to separate each element while joining
    List: to be converted to single String

D-11-array-join.pl

map() function

- map() function in Perl evaluates the operator provided as a parameter for each element of List.
- For each iteration, $_ holds the value of the current element, which can also be assigned to allow the value of the element to be updated.
- map() function runs an expression on each element of an array and returns a new array with the updated results.
- It returns the total number of elements generated in scalar context and list of values in list context.
- Syntax:
  map(operation, List)
  operation: to be performed on list elements
  List: whose elements need to be changed

D-12-array-map.pl

Arrays (push, pop, shift, unshift)

- Perl provides various inbuilt functions to add and remove the elements in an array.

FUNCTION	DESCRIPTION
push	Inserts values of the list at the end of an array
pop	Removes the last value of an array
shift	Shifts all the values of an array on its left
unshift	Adds the list element to the front of an array

push function

- This function inserts the values given in the list at an end of an array.
- Multiple values can be inserted separated by comma. This function increases the size of an array.
- It returns number of elements in new array.
- Syntax:

  push(Array, list)

D-13-array-push.pl

pop function

- This function is used to remove the last element of the array.
- After executing the pop function, size of the array is decremented by one element.
- This function returns undef if the array is empty otherwise returns last element of the array.
- Syntax:

  pop(Array)

D-14-array-pop.pl

shift function

- This function returns the first value in an array, removing it and shifting the elements of the array list to the left by one.
- Shift operation removes the value like pop but is taken from the start of the array instead of the end as in pop.
- This function returns undef if the array is empty otherwise returns first element of the array.
- Syntax:

  shift(Array)

D-15-array-shift.pl

unshift function

- This function places the given list of elements at the beginning of an array.
- Thereby shifting all the values in an array by right.
- Multiple values can be unshift using this operation.
- This function returns the number of new elements in an array.
- Syntax:

  unshift(Array, List)

D-16-array-unshift.pl

Hash

Hash

- A hash is a set of key-value pairs. Perl stores elements of a hash such that it searches for the values based on its keys.
- Perl requires the keys of a hash to be strings.
- A hash key must be unique. If a new key-value pair is added to a hash and that key is existing then its corresponding value is overwritten.

Creating Hashes

- The value is directly assigned as shown below and data is added to existing hash:

```
$stud{'Comp'} = 45;
$stud{'Inft'} = 42;
$stud{'Extc'} = 35;
```

- Using list which gets converted to hash by taking individual pairs:

```
%stud = ('Comp', 45, 'Inft', 42, 'Extc', 35);
```

- One way is using => to indicate the key/value pairs as shown below:

```
%stud = ('Comp' => 45, 'Inft' => 42, 'Extc' => 35);
```

E-01-hash.pl

Accessing Hash Elements

- To access the individual element from the hash, variable is prefixed with a dollar sign ($) and then append the element key within curly braces after the name of the variable.
- Example:
  print "$stud1{'Comp'}\n";

Extracting Keys and Values

- Sometimes there is a need to extract keys and values of a hash to perform various operations on it.
- Hash allows to extract these keys and values with the use of inbuilt functions.
- keys function:

  keys %HASH

  Returns an array of all the keys present in the HASH
- values function:

  values %HASH

  Returns an array with all the values of HASH.

E-02-keys-function.pl
E-03-values-function.pl

each function

- This function returns a Two-element list consisting of the key and value pair for the next element of a hash when called in List context, so that you can iterate over it.
- A 2-element list of key-value pairs for the List context whereas only the key for the scalar context.
- Syntax:
  ($key, $value) = each(%hash)

E-04-each-function.pl

Iterating Over Hashes

- Iterating over hashes:
  - To access the value in a hash user must the know the key associate to that value.
  - the keys of a hash are not known prior then with the help of keys function, user can get the list of keys and can iterate over those keys.
  - Example:

```
my @fruits = keys %rateof;
for my $fruit (@fruits) {
    print "The color of '$fruit' is $rateof{$fruit}\n";
}
```

Size of a hash

- The number of key/value pairs is known as the size of hash.
- To get the size, the first user has to create an array of keys or values and then he can get the size of the array.
- Syntax:

  print scalar keys % hash_variable_name;

E-05-size-of-hash.pl

Adding and Removing Elements in Hashes

- User can easily add a new pair of key/values into a hash using simple assignment operator:

```
$rateof{'Potato'} = 20;
```

- Deleting an element from hash using delete function:

```
delete $rateof{'Mango'};
```

E-06-add-element-to-hash.pl
E-07-delete-element-from-hash.pl

Operators

Operators

Operators Can be categorized based upon their different functionality:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Ternary Operator
- Quote Like Operators
- String Manipulation Operators
- Range Operator
- Auto Increment & Decrement Operator
- Arrow Operator

Arithmetic Operators

These are used to perform arithmetic/mathematical operations on operands.
Note: $a = 10, $b = 4;

Operation	Operator	Example	Result
Addition	+	$a + $b;	14
Subtraction	–	$a - $b;	6
Multiplication	*	$a * $b;	40
Division	/	$a / $b;	2.5
Modulus	%	$a % $b;	2
Exponent	**	$a**$b;	1000

F-01-arithmetic.pl

Relational (Equality) Operators

- Perl has two types of comparison operator sets.
- One is for numeric scalar values and one is for string scalar values.

NUMERIC	STRING	DESCRIPTION
==	eq	Equals to
!=	ne	Not Equals to
<	lt	Is less than
>	gt	Is greater than
<=	le	Is less than or equal to
>=	ge	Is greater than or equal to

F-02-relational.pl

Relational (Equality) Operators

Relational operators are used for comparison of two values. These operators will return either 1(true) or nothing(i.e. 0(false)).

Equal To	==	If two values are equals then return 1 otherwise return nothing.
Not equal	!=	Check if the two values are equal or not. If not equal then returns 1 otherwise returns nothing.
Comparison of equal	< = >	If left operand is less than right then returns -1, if equal returns 0 else returns 1.
Greater than	>	If left operand is greater than right returns 1 else returns nothing.
Less than	<	If left operand is lesser than right returns 1 else returns nothing.
Greater than equal	>=	If left operand is greater than or equal to right returns 1 else returns nothing.
Less than equal	<=	If left operand is lesser than or equal to right returns 1 else returns nothing.

Logical Operators

These operators are used to combine two or more conditions/constraints or to complement the evaluation of the original condition in consideration.

- **Logical AND:** The ‘and’ operator returns true when both the conditions in consideration are satisfied.
- **Logical OR:** The ‘or’ operator returns true when one (or both) of the conditions in consideration is satisfied.
- **Logical NOT:** The ‘not’ operator will give 1 if the condition in consideration is satisfied. For example, not($d) is true if $d is 0.

F-03-logical.pl

Assignment Operators

- Assignment operators are used to assigning a value to a variable.
- he left side operand of the assignment operator is a variable and right side operand of the assignment operator is a value.

F-04-assigment.pl

Assignment Operators (Cont'd)

Assignment	Sign	Example	Short for
Simple Assignment	=	$a = 10;	
Add Assignment	+=	$a += $b	$a = $a + $b
Subtract Assignment	-=	$a -= $b	$a = $a - $b
Multiply Assignment	*=	$a *= $b	$a = $a * $b
Division Assignment	/=	$a /= $b	$a = $a / $b
Modulus Assignment	%=	$a %= $b	$a = $a % $b
Exponent Assignment	**=	$a **= $b	$a = $a ** $b

Ternary Operator

- It is a conditional operator which is a shorthand version of if-else statement.
- It has three operands and hence the name ternary.
- It will return one of two values depending on the value of a Boolean expression.
- condition ? first_expression : second_expression;

F-05-ternary.pl

Quote-like Operators

In these operators, {} will represent a pair of delimiters that user choose. In this category there are three operators as follows:

- **q{ } :** It will encloses a string in single quotes(”) and cannot interpolate the string variable. For Example: q{ITI} gives ‘ITI’.
- **qq{ }:** It will encloses a string in double quotes(“”) and can interpolate the string variable. For Example: qq{ITI} gives “ITI”.
- **qx{ } :** It will encloses a string in invert quotes(“). For Example: qq{ITI} gives `ITI`.

F-06-quote.pl

String Manipulation Operators

The String is defined by the user within a single quote(’) or double quote(“”). There are different types of string operators in Perl, as follows:

- **Concatenation Operator (.) :**
  Perl strings are concatenated with a Dot(.) symbol.
- **Repetition Operator (x):**
  The x operator accepts a string on its left-hand side and a number on its right-hand side. It will return the string on the left-hand side repeated as many times as the value on the right-hand side.

F-07-string.pl

Range Operator(..)

- In Perl, range operator is used for creating the specified sequence range of specified elements.
- So this operator is used to create a specified sequence range in which both the starting and ending element will be inclusive. For example, 7 .. 10 will create a sequence like 7, 8, 9, 10.
- @res = (1..4);

F-08-range.pl

Auto Increment Operator

Auto Increment(++):

Increment the value of an integer.

- When placed before the variable name (also called pre-increment operator), its value is incremented instantly.

For example, ++$x.

- And when it is placed after the variable name (also called post-increment operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement.

For example, $x++.

F-09-increment.pl

Auto Decrement Operator (Cont’d)

Auto Decrement Operator(–):

Decrement the value of an integer.

- When placed before the variable name (also called pre-decrement operator), its value is decremented instantly.

For example, --$x.

- And when it is placed after the variable name (also called post-decrement operator), its value is preserved temporarily until the execution of this statement and it gets updated before the execution of the next statement.

For example, $x--.

- **Note:** The increment and decrement operators are called unary operators as they work with a single operand.

Arrow Operator(->)

- This operator is used for dereferencing a Variable or a Method from a class or an object.
- Example:
  $ob->$x is an example to access variable $x from object $ob. Mostly this operator is used as a reference to a hash or an array to access the elements of the hash or the array.

F-10-arrow.pl.pl

Operator Precedence & Associativity Table

OPERATOR	PRECEDENCE	ASSOCIATIVITY
->	Arrow Operator	Left to Right
++, —	Increment, decrement	Non Associative
**	Exponent	Right to Left
!, +, -, ~	Not, unary plus, unary minus, complement	Right to Left
!=, ~=	Not equal to, complement equal to	Left to Right
*, /, %	Multiply, Divide, Modulus	Left to Right

Operator Precedence & Associativity Table (Cont'd)

OPERATOR	PRECEDENCE	ASSOCIATIVITY
<>, <=, >=	Comaparison, Less than equal to, right than equal to	Non Associative
&	Bitwise AND	Left to Right
|, ^	Bitwise OR, EX-OR	Left to Right
&&	AND	Left to Right
||	OR	Left to Right
..	Range Operator	Non Associative
*=, /=, +=, -=	Multiply equal to, Divide equal to, plus equal to	Right to Left

Input and Output

print() function

- print function in Perl is used to print the values of the expressions in a List passed to it as an argument.
- Print operator prints whatever is passed to it as an argument whether it be a string, a number, a variable or anything.

G-01-print-1.pl
G-01-print-2.pl

say() function

- say() function automatically adds a new line at the end of the statement, there is no need to add a newline character ‘\n’ for changing the line.
- We have used ‘use 5.010‘ to use the say() function because newer versions of Perl don’t support some functions of the older versions and hence

G-03-say.pl

Input

- Perl allows the programmer to accept input from the user to perform operations on. This makes it easier for the user to give input of its own and not only the one provided as Hard Coded input by the programmer.
- Input to a Perl program can be given by keyboard with the use of <STDIN>. Here, STDIN stands for Standard Input.
- Though there is no need to put STDIN in between the ‘diamond’ or ‘spaceship’ operator i.e, <>.
- Syntax:

  $x = <STDIN>;** or **$x = <>;

G-04-input-1.pl
G-05-input-2.pl

Input (Cont’d)

- After giving Input, there is a need to press ENTER. This ENTER is used to tell the compiler to execute the next line of the code. But, <STDIN> takes this ENTER key pressed as a part of the Input given and hence when we print the line.
- A new line will automatically be printed after the Input string. To avoid this, a function chomp() is used.
- This function will remove the newline character added to the end of the Input provided by the user.

Decision Making

Decision Making

- A programming language uses control statements to control the flow of execution of the program based on certain conditions.
- These are used to cause the flow of execution to advance and branch based on changes to the state of a program.
- Decision Making Statements in Perl :

  - if
  - if - else
  - Nested if
  - if -elsif ladder
  - unless
  - unless - else
  - unless - elsif

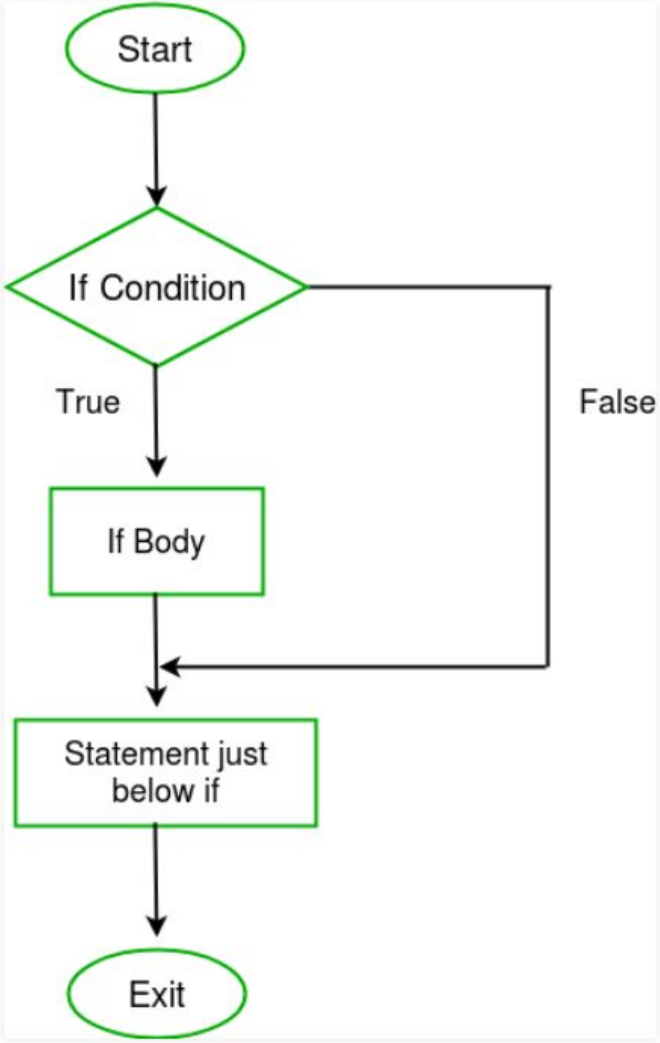
if statement

- It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.
- Syntax :

```
if(condition)
{
      # code to be executed
}
```

- Note : If the curly brackets { } are not used with if statements than there will be compile time error. So it is must to use the brackets { } with if statement.

H-01-if.pl

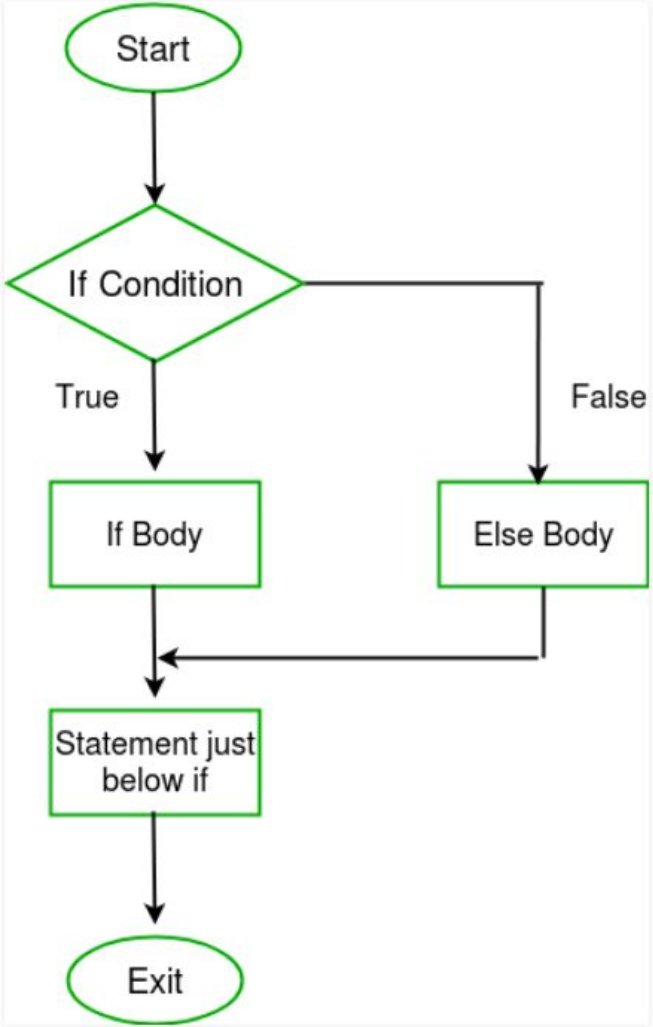

Start
If Condition
True
False
If Body
Statement just
below if
Exit

if – else Statement

- The if statement evaluates the code if the condition is true but what if the condition is not true, here comes the else statement. It tells the code what to do when the if condition is false.
- Syntax :

```
if(condition)
{
	# code if condition is true
}
else
{
	# code if condition is false
}
```

H-02-if-else.pl

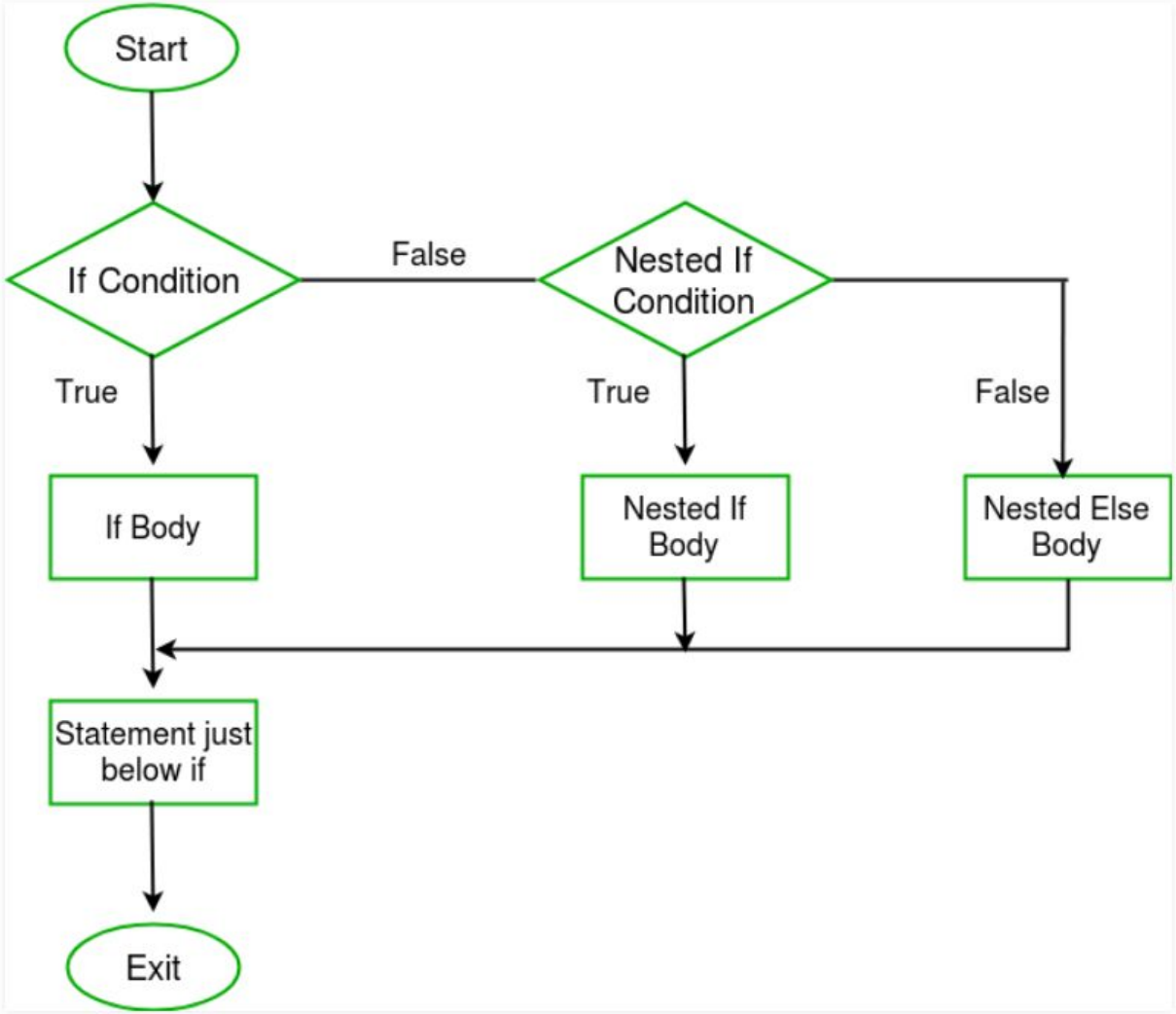

Start
If Condition
True
False
If Body
Else Body
Statement just below if
Exit

Nested – if Statement

- if statement inside an if statement is known as nested if. if statement in this case is the target of another if or else statement. When more then one condition needs to be true and one of the condition is the sub-condition of parent condition, nested if can be used.
- Syntax :

```
if (condition1) {
   # Executes when condition1 is true
   if (condition2)
   {
      # Executes when condition2 is true
   }
}
```

H-03-if-nested.pl


Start
If Condition
False
Nested If Condition
True
True
False
If Body
Nested If Body
Nested Else Body
Statement just below if
Exit

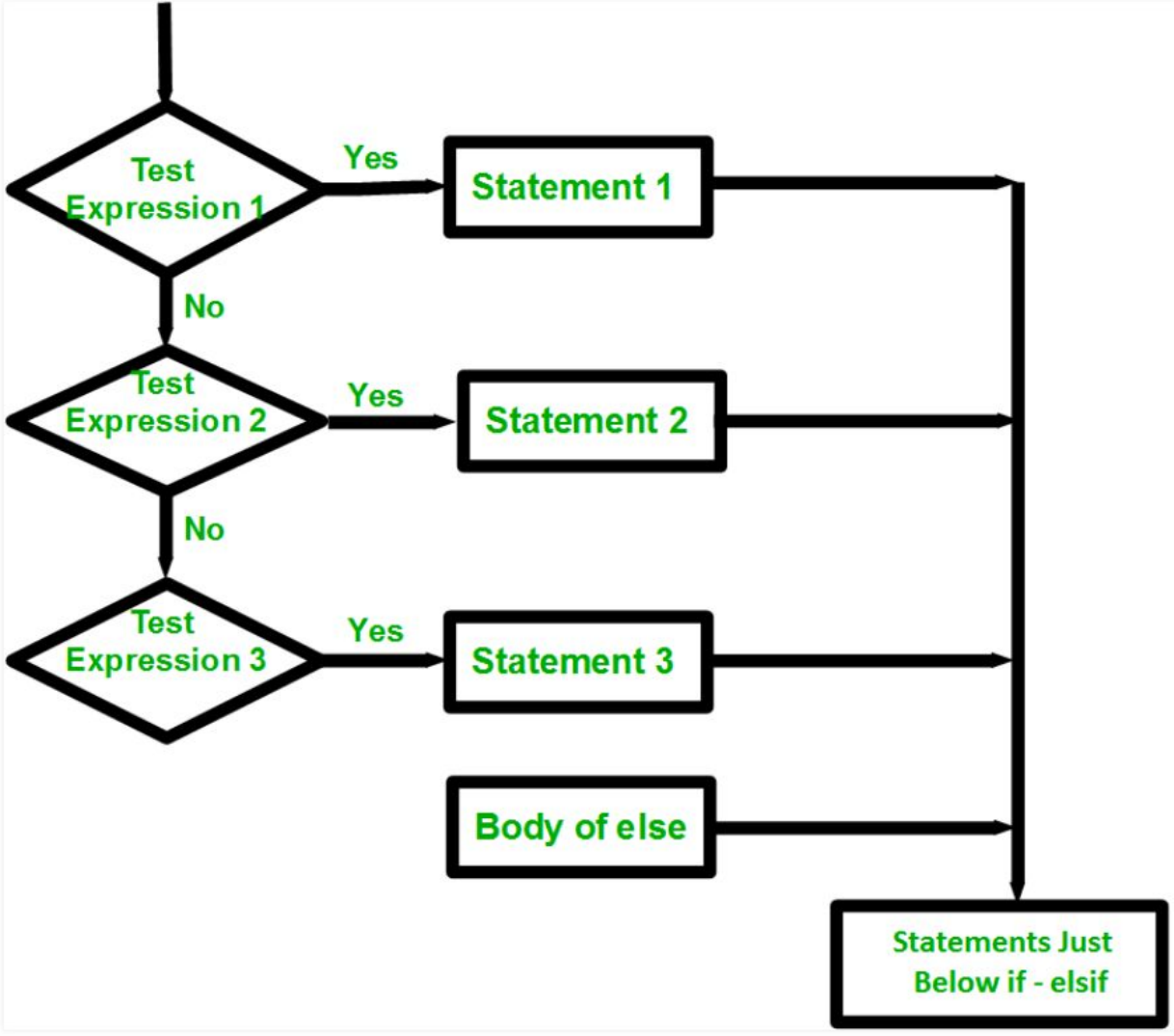
If – elsif – else ladder Statement

- Here, a user can decide among multiple options.
- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that get executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

If – elsif – else ladder Statement (Cont'd)

```
if(condition1)
{
            # code to be executed if condition1 is true
}
elsif(condition2)
{
            # code to be executed if condition2 is true
}
elsif(condition3)
{
            # code to be executed if condition3 is true
}
  ...
else
{
            # code to be executed if all the conditions are false
}
```

H-04-if-ladder.pl

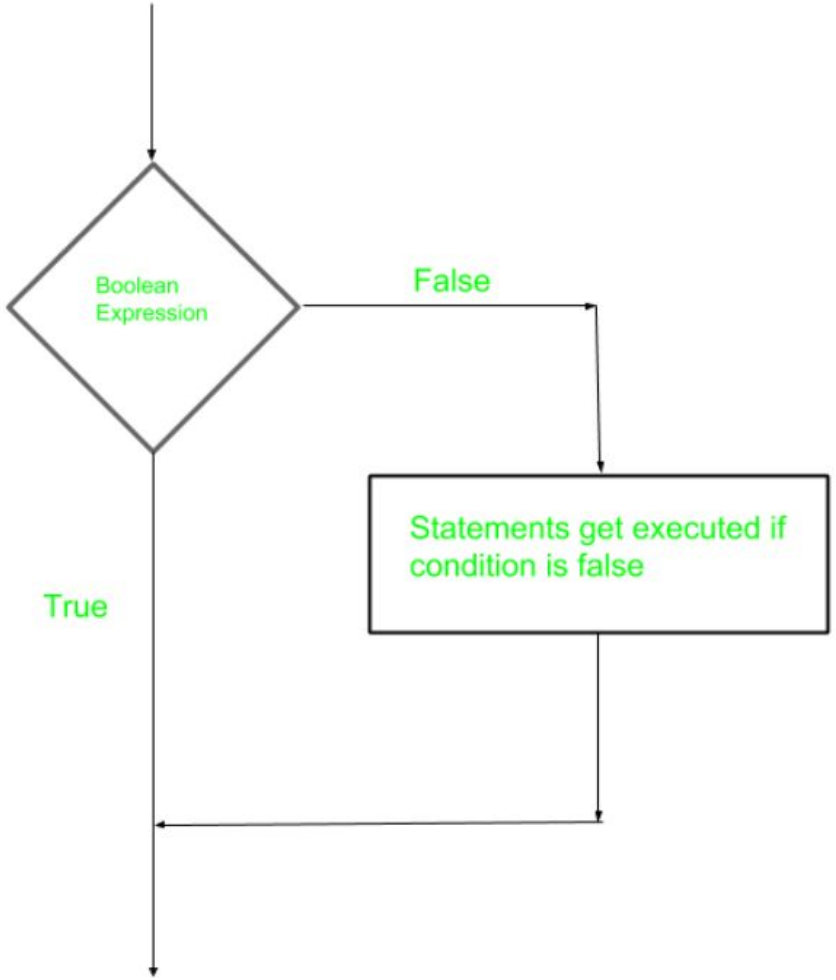

Test
Expression 1
Yes
Statement 1
No
Test
Expression 2
Yes
Statement 2
No
Test
Expression 3
Yes
Statement 3
Body of else
Statements Just
Below if - elsif

unless Statement

- In this case if the condition is false then the statements will execute.
- The number 0, the empty string “”, character ‘0’, the empty list (), and undef are all false in a boolean context and all other values are true.
- Syntax :

```
unless(boolean_expression)
{
   # will execute if the given condition is false
}
```

H-05-unless.pl

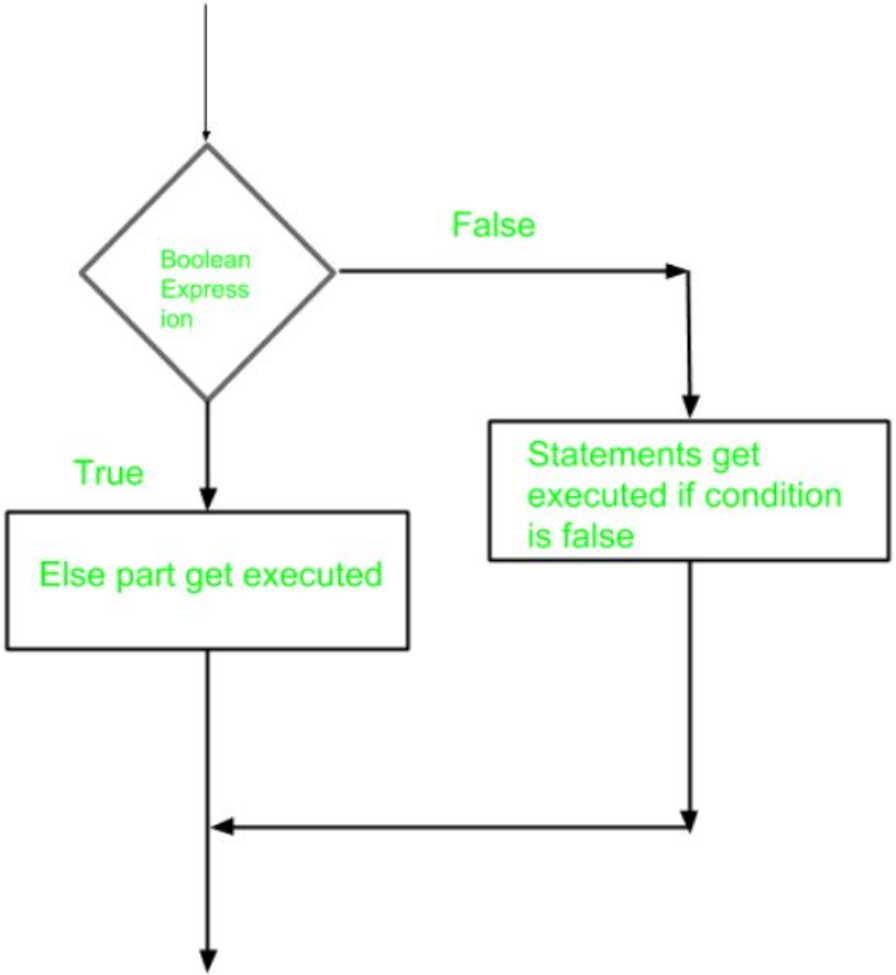

Boolean
Expression
False
Statements get executed if
condition is false
True

Unless-else Statement

- Unless statement can be followed by an optional else statement, which executes when the boolean expression is true.
- Syntax :

```
unless(boolean_expression)
{
      # execute if the given condition is false
}
else
{
      # execute if the given condition is true
}
```

H-06-unless-else.pl

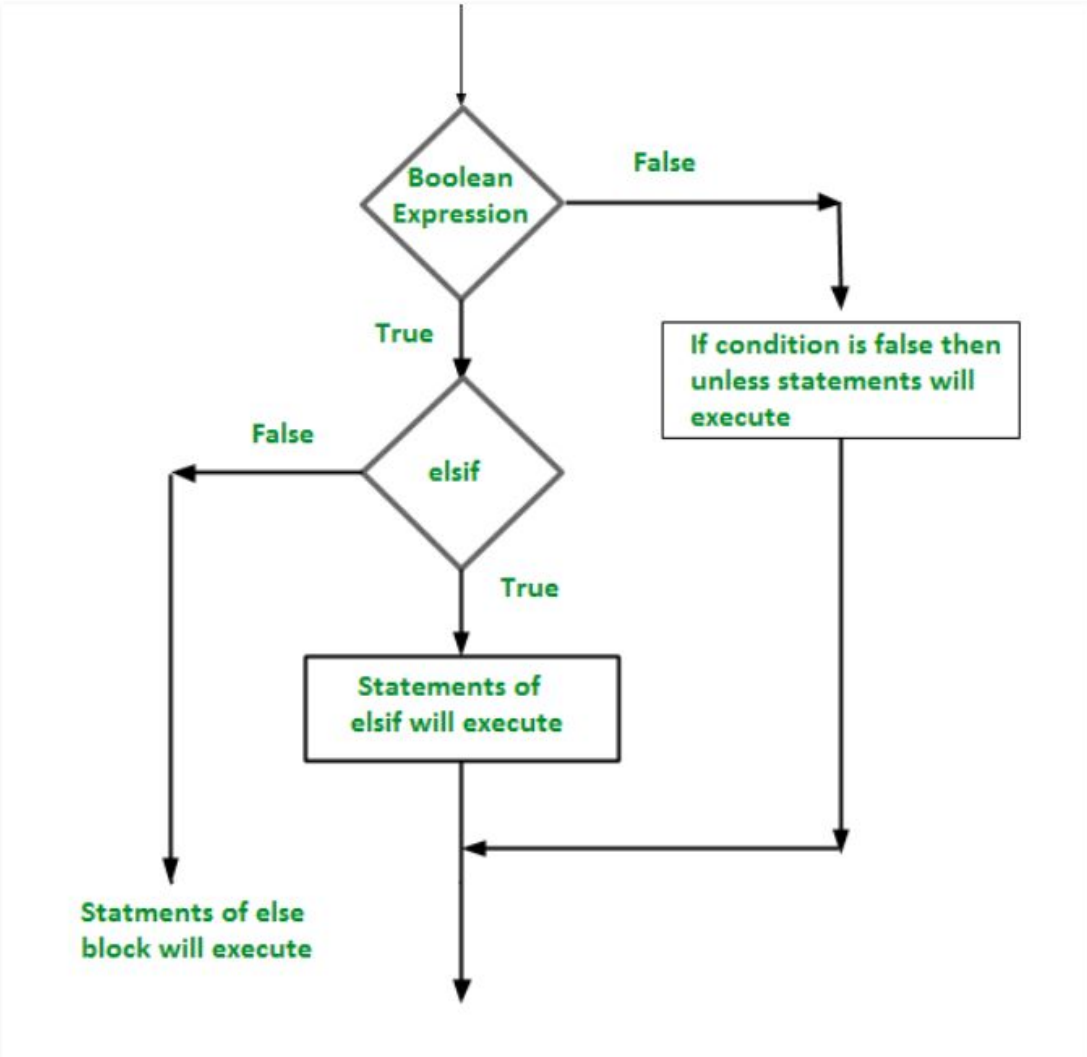

Boolean
Express
ion
False
True
Else part get executed
Statements get
executed if condition
is false

Unless – elsif Statement

- Unless statement can be followed by an optional elsif…else statement, which is very useful to test the various conditions using single unless…elsif statement.
- Syntax :

```
unless(boolean_expression 1)
{
      # Executes when the boolean expression 1 is false
}
elsif( boolean_expression 2)
{
      # Executes when the boolean expression 2 is true
}
else
{
      # Executes when the none of the above condition is met
}
```

H-07-unless-ladder.pl


Boolean Expression
False
True
If condition is false then unless statements will execute
False
elsif
True
Statements of elsif will execute
Statments of else block will execute