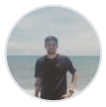
PL/SQL Tutorial : Everything You Need To Know About PL/SQL

Published on Nov 15,2019 *3.8K Views*



Mohammad Waseem

PL/SQL is a procedural language that overcomes the shortcomings faced by Structured Query Language. It is an extension of SQL and we can even use SQL queries without any hassle in any PL/SQL application or program. In this PL/SQL tutorial, we will go through the basic concepts of PL/SQL in detail. The following topics are covered in this article.

What is PL/SQL?

It stands for procedural language extension to the structured query language. Oracle created PL/SQL that extends some limitations of SQL to provide a more comprehensive solution for building mission-critical applications running on the oracle database.

Features

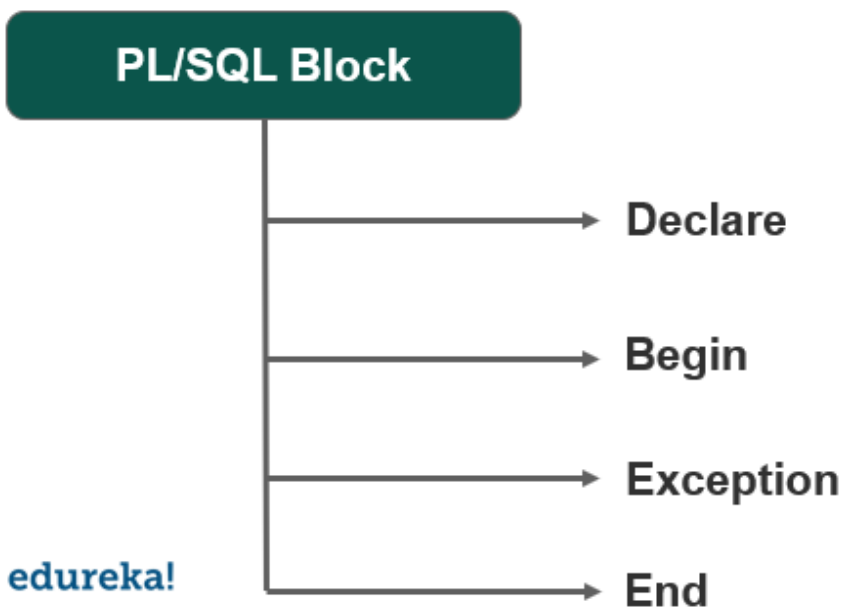
- PL/SQL provides the functionality of a procedural language such as decision making, iteration, etc.
- Using a single command, PL/SQL can execute a number of queries.
- We can also reuse PL/SQL units such as functions, triggers, procedures, etc that are stored in the database after the creation.
- PL/SQL also has an exception handling block that handles the exceptions in PL/SQL.
- Extensive error checking is also possible using PL/SQL
- The applications written in PL/SQL are portable to other hardware and operating systems provided oracle must be operational.

PL/SQL vs SQL

SQL	PL/SQL

SQL is a single query that is used to perform DDL and DML operations	PL/SQL is a block of codes that is used to define an entire program or procedure/function, etc
It does not really define how things need to be done, rather defines what needs to be done	PL/SQL defines how things need to be done
It executes a single statement	It executes a block of statements at once.
SQL is mainly used to manipulate the data	PL/SQL, on the other hand, is used to create applications
It cannot contain PL/SQL code	Since it is a SQL extension, it can contain SQL code in it

Block Structures In PL/SQL



PL/SQL typically organizes the code into blocks. The code block with no name is known as an anonymous block. It is known as the anonymous block because it is not saved in the oracle database. Let us take a look at an anonymous block in PL/SQL.

```
[DECLARE]
   declaration statements;
[BEGIN]
   execution statements;
   [EXCEPTION]
      exception statements;
END;
/
```

Looking at the diagram shown above, we can see that the block structure is divided into four parts, i.e declaration, begin, exception and end. Let us try to understand how the block structure works in PL/SQL. Out of all these sections, the execution section is mandatory and rest all are optional.

- **DECLARE** keyword is used to for the declaration section is used to declare data types and structures such as variables, functions, etc.
- **BEGIN** keyword is used for the execution section. It is mandatory and contains all the statements that need to be executed. This block is where the business logic is defined, we can use both procedural or SQL statements in this block.
- The **EXCEPTION** keyword is used for the exception section. It contains all the exception statements.
- **END** keyword marks the end of the block and the backward slash '/' tells the tool that you are using(Oracle Database Tool) to execute the PL/SQL block.

Here is a simple example to show how we can use the PL/SQL code.

```
BEGIN
        NULL;
END;
/
```

Now that we know how the block structure works in PL/SQL, let us understand the various aspects of PL/SQL like declaring, naming and assigning values to the variables.

PL/SQL Variables

The variable in PL/SQL is basically a name that varies or temporary storage location that supports a particular data type. Let us take a look at how we can use the variables in a PL/SQL program.

Variable Naming Rules

PL/SQL follows the following rules for naming variables.

- The variable cannot be more than 31 characters
- The name of the variable should start with an ASCII character. Since PL/SQL is case-sensitive, an uppercase letter and a lowercase letter will be different variables.
- After the first character, there has to be a special character($,_) or any number.

Naming Conventions

Use the following naming conventions listed below to use the variables.

Prefix	Data Type
v_	VARCHAR2
n_	NUMBER
t_	TABLE
r_	ROW
d_	DATE
b_	BOOLEAN

Declaration

Let's try to understand how variable declaration is done in PL/SQL

The declaration includes the variable name followed by the data type and separated by a semicolon. Following is an example to show how you can declare a variable in PL/SQL.

```
DECLARE
   v_name VARCHAR(25);
   n_age NUMBER(3);
BEGIN
  NULL;
END;
```

You can also add the length of the data type as we have done in the example above.

Anchors

The anchor basically refers to the use of the %TYPE keyword that to declare a variable with the data type associated with a column's data type of a particular column in a table.

Take a look at an example to understand this. Suppose we have a table EMPLOYEES, we can use the anchors in the following way.

```
DECLARE
    v_name EMPLOYEE.NAME%TYPE;
    n_age    EMPLOYEE.AGE%TYPE;
BEGIN
   NULL;
END;
/
```

Assignment

Variable assignment is quite easy, we can use the assignment operator to assign values to a variable. The following example shows how we can assign values to a variable.

```
DECLARE
   v_name VARCHAR(20);
   n_course VARCHAR(10);
BEGIN
  v_name = "edureka";
  v_course = "sql";
END;
/
```

Initialization

We can initialize a value for the variable in the declaration section too. The following example shows how we can initialize values to a variable.

```
DECLARE
v_name VARCHAR(20) = "edureka";
n_course VARCHAR(10) = "sql";
BEGIN
 NULL;
END;
/
```

Now that we know how we can work with the variables, let us try to understand how we will use functions in PL/SQL.

Function In PL/SQL

A function in PL/SQL is basically a named block that returns a value. It is also known as a subroutine or a subprogram, the following syntax shows how we can use functions in PL/SQL.

```
CREATE [OR REPLACE] FUNCTION function_name [(
   parameter_1 [IN] [OUT] data_type,
   parameter_2 [IN] [OUT] data_type,
   parameter_N [IN] [OUT] data_type]
    RETURN return_data_type IS

BEGIN
   statements
   return return_data_type;
   EXCEPTION

END;
/
```

First of all, you must specify a function name after the keyword. The function name has to start with a verb. A function may have none, one or more parameters that we specify in parameters. We have to specify the data type of each parameter explicitly, and then comes the mode which can either of the following.

- **IN –** The IN parameter is a read-only parameter.
- **OUT –** It is a write-only parameter
- **IN OUT –** The IN OUT parameter is both read-write parameter.

Here is a simple example to show how we use functions in PL/SQL.

```
CREATE OR REPLACE FUNCTION try_parse(
    iv_number IN VARCHAR2)
  RETURN NUMBER IS
BEGIN
   RETURN to_number(iv_number);
   EXCEPTION
     WHEN others THEN
        RETURN NULL;
END;
```

Calling A Function

Let us try to call the function that we have made in an anonymous block in the following example.


```
SET SERVEROUTPUT ON SIZE 1000000;
DECLARE
  n_x number;
  n_y number;
   n_z number;
BEGIN
   n_x := try_parse('256');
   n_y := try_parse('29.72');
   n_z := try_parse('pqrs');

   DBMS_OUTPUT.PUT_LINE(n_x);
   DBMS_OUTPUT.PUT_LINE(n_y);
   DBMS_OUTPUT.PUT_LINE(n_z);
END;
/
```

We can call the function in a SELECT statement too. Now that we know how we can use functions in PL/SQL, let us try to understand how we work with procedures in PL/SQL.

PL/SQL Procedure

A procedure is basically a block that does a specific task. Using a procedure we can wrap or encapsulate complex business logic and reuse them in both application and database layer.

Let us take a look at a simple example to understand how the procedure works in PL/SQL

```
CREATE OR REPLACE PROCEDURE adjust_salary(
    in_employee_id IN EMPLOYEES.EMPLOYEE_ID%TYPE,
    in_percent IN NUMBER
) IS
BEGIN
   -- update employee's salary
   UPDATE employees
   SET salary = salary + salary * in_percent / 100
   WHERE employee_id = in_employee_id;
END;
```

In the above example, we have two parameters, the procedure adjusts the salary by a given percentage and the UPDATE keyword updates the value in the salary information.

Procedure Header

The section before the keyword IS is called the procedure header. The following are a few pointers one must be familiar with while working with procedures.

- **schema –** It is the optional name of the schema that the procedure belongs to.
- **name –** The name of the procedure which should start with a verb.
- **parameters –** It is the optional list of parameters.
- **AUTHID –** It determines whether the procedure will execute with the privilege of the current user or the original owner of the procedure.

Procedure Body

Everything that comes after the IS keyword is called the procedure body. We have the declaration, exception and execution statements in the procedure body. Unlike the function, the RETURN keyword in a procedure is used to halt the execution and return the control to the caller.

Calling A Procedure

Let us see how we can call a procedure in PL/SQL.

```

EXEC procedure_name(param1,param2…paramN);
```

We can call the procedures with no parameters with just using the EXEC keyword and procedure name. Now that we know how we can work with procedures, let us try to understand how nested blocks are used in PL/SQL.

Nested Block

A nested block is nothing but a combination of one or more PL/SQL blocks to get better control over the execution and exceptional handling for the program.

Here is a simple example of a nested block.

```
SET SERVEROUTPUT ON SIZE 1000000;
DECLARE
  n_emp_id EMPLOYEES.EMPLOYEE_ID%TYPE := &emp_id1;
BEGIN
  DECLARE
    n_emp_id employees.employee_id%TYPE := &emp_id2;
    v_name   employees.first_name%TYPE;
  BEGIN
    SELECT first_name
    INTO v_name
    FROM employees
    WHERE employee_id = n_emp_id;

    DBMS_OUTPUT.PUT_LINE('First name of employee ' || n_emp_id ||
                                        ' is ' || v_name);
    EXCEPTION
      WHEN no_data_found THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' || n_emp_id || ' not found');
  END;
END;
/
```

The outer PL/SQL block in the above example is known as the parent block or enclosing block, the inner block, on the other hand, is known as the child block or the enclosed block.

It is not a great idea to use the variables with the same names in both the blocks because during the execution the child block variable will override the parent block variable. It happens because PL/SQL gives first priority to the variable inside its own block.

Block Label

We can overcome this issue with the block label that helps us make references to variables inside blocks using a label.

Here is a simple example to show how we can use a block label.

```
<<block_label>>
DECLARE
...
BEGIN
...
END;
```

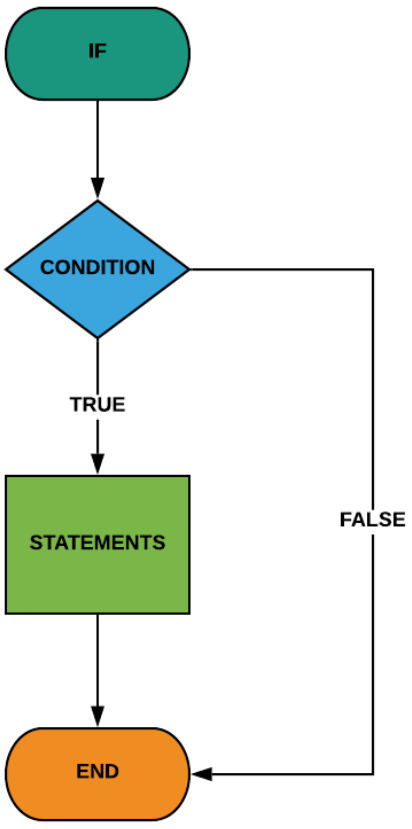
Using a block label helps to improve the readability of the code, gain better control and make references to the blocks. Now that we know how we can work with nested blocks, let us try to understand how the IF STATEMENT works in PL/SQL.

IF Statement

PL/SQL has three IF STATEMENTS

- **IF-THEN** – It is the simplest IF STATEMENT if the condition is true the statements will execute, if the condition is false, it does nothing.
- **IF-THEN-ELSE –** In this, the ELSE clause is added for an alternative sequence of statements.
- **IF-THEN-ELSEIF –** It allows us to execute multiple test conditions in a sequence.

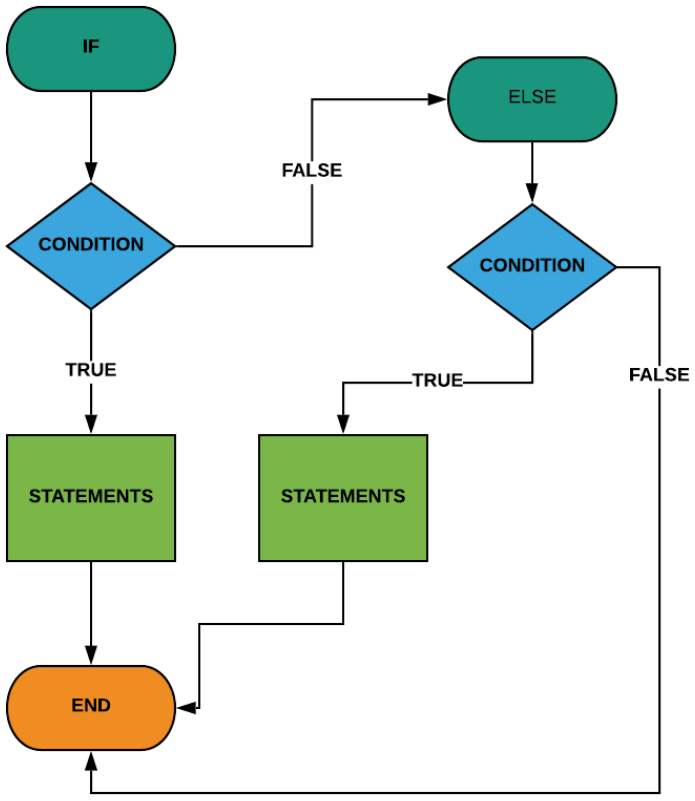
IF-THEN Syntax



tabases Training

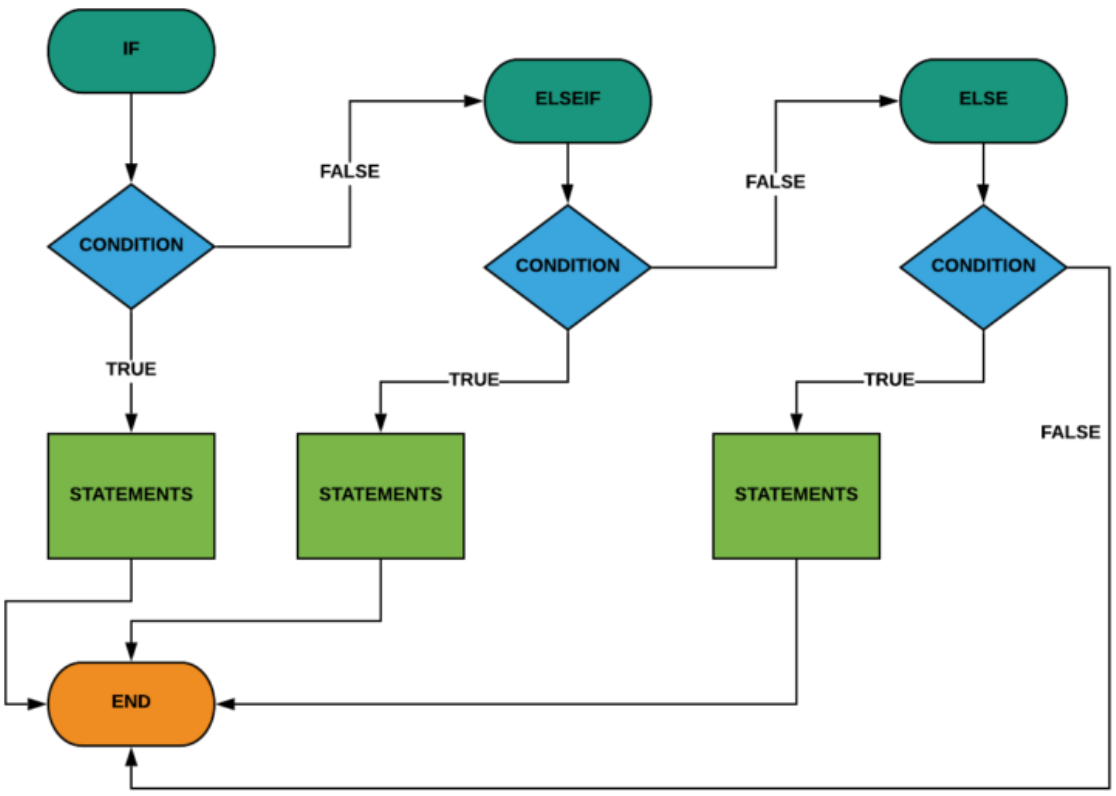

```
IF condition THEN
   sequence_of_statements;
END IF;
```

IF-THEN-ELSE Syntax




```
IF condition THEN
   sequence_of_if_statements;
ELSE
   sequence_of_else_statements;
END IF;
```

IF-THEN-ELSEIF Syntax




```
IF condition1  THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

Now that we are done with the IF STATEMENT let us look at the CASE statement in PL/SQL.

CASE Statement

The CASE statement basically helps in executing a sequence of statements based on a selector. A selector, in this case, can be anything, it can be a variable, function or a simple expression. Here is a simple example to show the syntax of the CASE statement in PL/SQL.

```
[<<label_name>>]
CASE [TRUE | selector]
   WHEN expression1 THEN
 sequence_of_statements1;
   WHEN expression2 THEN
 sequence_of_statements2;
   ...
   WHEN expressionN THEN
 sequence_of_statementsN;
  [ELSE sequence_of_statementsN+1;]
END CASE [label_name];
```

In the above syntax, after the CASE keyword comes the selector. PL/SQL will evaluate the selector only once to determine which statement needs to be executed.

Followed by the selector is the WHEN keyword. If the expression satisfies the selector then the corresponding statement after THEN keyword gets executed.

Now that we know how we can use a CASE statement, let us try to understand how we will use the loop statements in the PL/SQL.

Loop Statement

A loop statement in PL/SQL is an iterative statement that allows you to execute a sequence of statements multiple times. Here is a simple example to show the syntax of a loop statement in PL/SQL.

```
LOOP
   sequence_of_statements;
END LOOP;
```

There has to be at least one executable statement in between the LOOP and END LOOP keyword.

Loop with EXIT Statement

The EXIT and EXIT when statements allow you exit the loop. EXIT WHEN statement terminates the loop conditionally while EXIT terminates the execution unconditionally.

```
LOOP
   ...
   EXIT WHEN condition;
END LOOP;
```

Loop Label

A loop label is used to qualify the name of the loop counter variable when used in a nested loop. Following is the syntax of a loop label.

```
<<label>>
LOOP
   sequence_of_statements;
END LOOP label;
```

Now that we know how we can use the loop statements let us take a look at while loop statements for better understanding.

While Loop Statement

We can use the WHILE loop statement when the number of executions are not defined until the execution starts. The following syntax is used for a WHILE loop statement in PL/SQL.

```
WHILE condition
LOOP
   sequence_of_statements;
END LOOP;
```

The condition in the syntax is a boolean value or expression that evaluates to be either TRUE, FALSE or NULL. If the condition is TRUE, the statements will be executed, if it is FALSE, the execution stops and the control goes to the next executable statement.

Now that we know how we can use a WHILE loop statement, let us take a look at the FOR loop statement.

For Loop Statement

A FOR loop statement in PL/SQL allows us to execute a sequence of statements for a definite number of times. Following is the syntax to use FOR loop statement in PL/SQL

```
FOR loop_counter IN [REVERSE] lower_bound .. higher_bound
LOOP
   sequence_of_statements;
END LOOP;
```

PL/SQL creates a local variable loop_counter automatically with an INTEGER data type for the loop so that you don't have to declare it explicitly. The lowerbound..higherbound is the range over which the loop iterates. Also, you must have at least one executable statement between LOOP and END LOOP keywords.

Now that we know how we can use the loop statements in PL/SQL, let us take a look at exceptional handling in PL/SQL.

Exceptional Handling

In PL/SQL any kind of error is treated as an exception. An exception can be treated as a special condition that can change or alter the execution flow. In PL/SQL, there are two types of exceptions.

- **System Exception –** It is raised by the PL/SQL run-time when it detects an error.
- **Programmer-Defined Exception –** These exceptions are defined by the programmer in a specific application.

Defining An Exception

An exception in PL/SQL has to be declared before it can be raised. We can define the exception using the EXCEPTION keyword like we have done in the example below.

```
EXCEPTION_NAME EXCEPTION;
```

To raise an exception, we use the RAISE keyword.

```
RAISE EXCEPTION_NAME;
```

So that was all about PL/SQL, I hope this article has helped you in adding value to your knowledge. For more information on SQL or Databases, you can refer to our comprehensive reading list here: **Databases Edureka**.

*If you wish to get a structured training on MySQL, then check out our **MySQL DBA Certification Training** which comes with instructor-led live training and real-life project experience. This training will help you understand MySQL in-depth and help you achieve mastery over the subject.*

*Got a question for us? Please mention it in the comments section of "**PL/SQL Tutorial**" and I will get back to you.*

‹›

Comments

0 Comments

Join the discussion

Enter your comment here...

SQL Essentials Training & Certification

7k Enrolled Learners
Weekend/Weekday
Self Paced

Reviews
5 (2750)

MySQL DBA Certification Training

4k Enrolled Learners
Weekend
Live Class

Reviews
5 (1550)

MongoDB Certification Training

16k Enrolled Learners
Weekend
Live Class

Reviews
4 (6150)

Teradata Certifica Training

3k Enrolled Learners
Weekend
Live Class

Reviews
5 (1000

‹›

Browse Categories

Artificial Intelligence | BI and Visualization | Big Data | Blockchain | Cloud Computing | Cyber Security | Data Science | Data Warehousing and ETL | DevOps | Digital Marketing | Enterprise | Front End Web Development | Mobile Development | Operating Systems | Programming & Frameworks | Project Management and Methodologies | Robotic Process Automation | Software Testing | Systems & Architecture

edureka!

TRENDING CERTIFICATION COURSES

DevOps Certification Training
AWS Architect Certification Training
Big Data Hadoop Certification Training
Tableau Training & Certification
Python Certification Training for Data Science
Selenium Certification Training
PMP® Certification Exam Training
Robotic Process Automation Training using UiPath
Apache Spark and Scala Certification Training
Microsoft Power BI Training
Online Java Course and Training
Python Certification Course

TRENDING MASTERS COURSES

Data Scientist Masters Program
DevOps Engineer Masters Program
Cloud Architect Masters Program
Big Data Architect Masters Program
Machine Learning Engineer Masters Program
Full Stack Web Developer Masters Program
Business Intelligence Masters Program
Data Analyst Masters Program
Test Automation Engineer Masters Program
Post-Graduate Program in Artificial Intelligence & Machine Learning
Post-Graduate Program in Big Data Engineering

COMPANY

About us
News & Media
Reviews
Contact us
Blog
Community
Sitemap
Blog Sitemap
Community Sitemap
Webinars

WORK WITH US

Careers
Become an Instructor
Become an Affiliate
Become a Partner
Hire from Edureka

DOWNLOAD APP

Download on the App Store | GET IT ON Google Play

CATEGORIES

CATEGORIES

Cloud Computing | DevOps | Big Data | Data Science | BI and Visualization | Programming & Frameworks | Software Testing | Project Management and Methodologies | Robotic Process Automation | Frontend Development | Data Warehousing and ETL | Artificial Intelligence | Blockchain | Databases | Cyber Security | Mobile Development | Operating Systems | Architecture & Design Patterns | Digital Marketing

TRENDING BLOG ARTICLES

TRENDING BLOG ARTICLES

Selenium tutorial | Selenium interview questions | Java tutorial | What is HTML | Java interview questions | PHP tutorial | JavaScript interview questions | Spring tutorial | PHP interview questions | Inheritance in Java | Polymorphism in Java | Spring interview questions | Pointers in C | Linux commands | Android tutorial | JavaScript tutorial | jQuery tutorial | SQL interview questions | MySQL tutorial | Machine learning tutorial | Python tutorial | What is machine learning | Ethical hacking tutorial | SQL injection | AWS certification career opportunities | AWS tutorial | What Is cloud computing | What is blockchain | Hadoop tutorial | What is artificial intelligence | Node Tutorial | Collections in Java | Exception handling in java | Python Programming Language | Python interview questions | Multithreading in Java | ReactJS Tutorial | Data Science vs Big Data vs Data Analyt... | Software Testing Interview Questions | R Tutorial | Java Programs | JavaScript Reserved Words and Keywor... | Implement thread.yield() in Java: Exam... | Implement Optical Character Recogniti... | All you Need to Know About Implemen...