

# Basic Feature of PL SQL Programming

Lab session 10



# Introduction

# Introduction

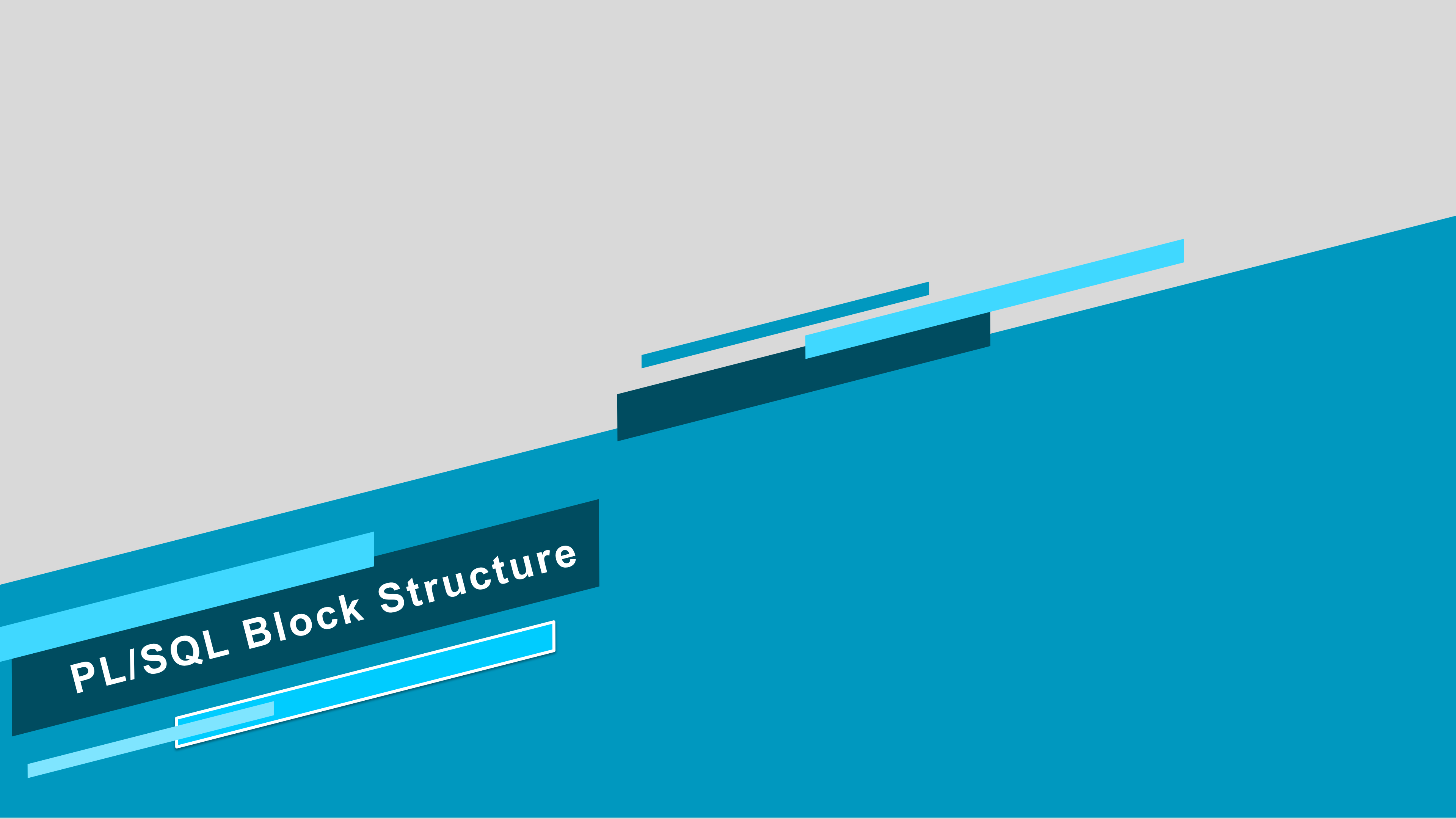
---

➤ In Oracle database management, **PL/SQL** is a **procedural language** extension to **Structured Query Language (SQL)**. The purpose of **PL/SQL** is to combine database language(SQL) and procedural programming language.

What is Procedural Programming language?

➤ **Procedural programming** is derived from structure programming, based upon the concept of the *procedure call*. Procedures, also known as routines, subroutines or functions. This language simply contain a series of computational steps to be carried out & Any given procedure might be called at any point during a program's execution, including by other procedures or itself(recursion). Procedural programming languages include [C](#), [Go](#), [Fortran](#), [Pascal](#), [Ada](#), and [BASIC](#).

PL/SQL offers modern software engineering features such as data encapsulation, exception handling and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.



# PL/SQL Block Structure

# PL/SQL Block Structure

---

A PL/SQL block consists of up to three sections: ***declarative*** (optional), ***executable*** (required - including Begin and end), and ***exception*** handling (optional).

PL/SQL block structure:-

## **DECLARE** - Optional

- Variables, cursors, user-defined exceptions

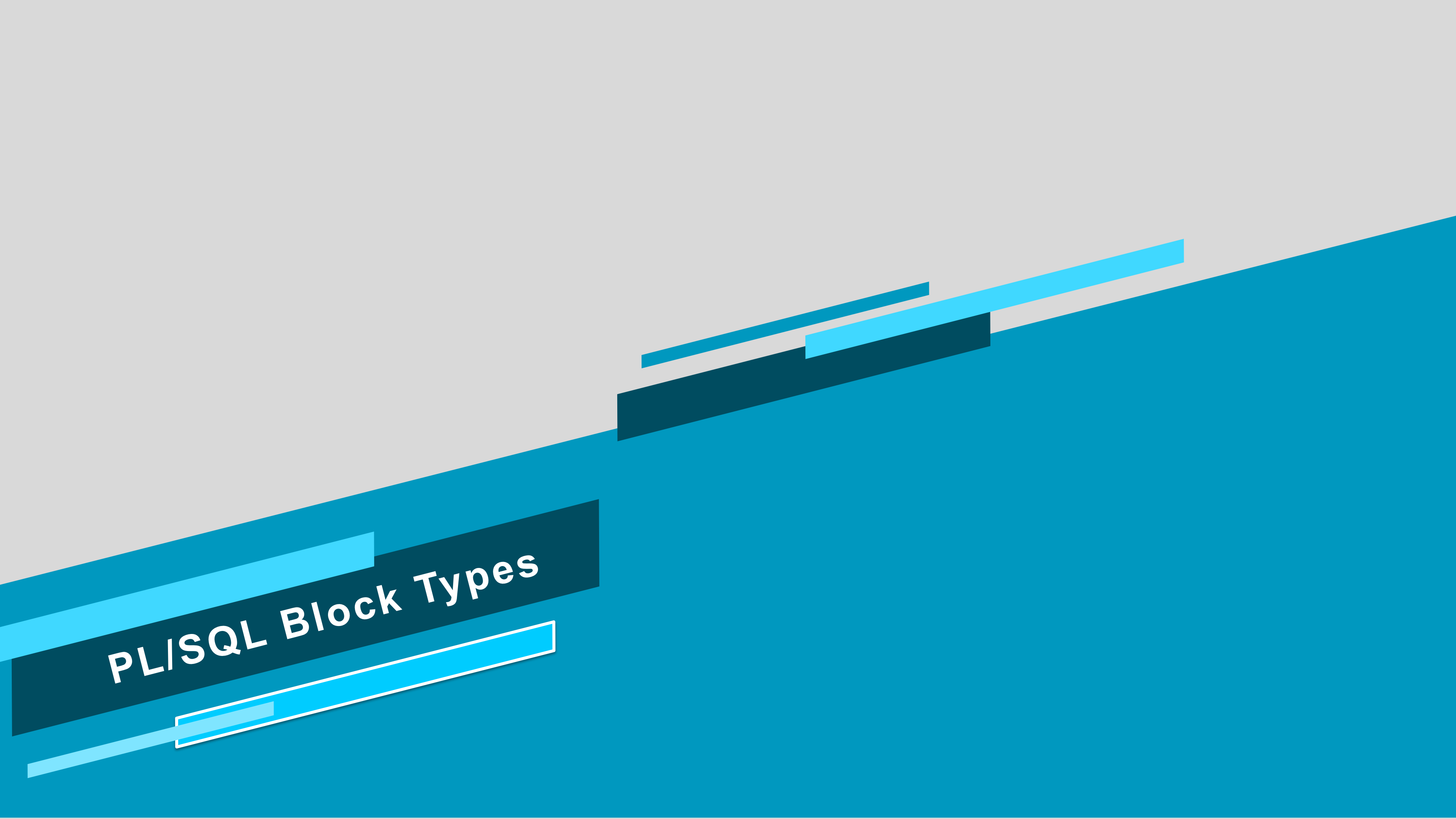
## **BEGIN** – mandatory

- SQL statements
- PL/SQL statements

## **EXCEPTION** – optional

- Actions to perform when errors occur

## **END;** - Mandatory



# PL/SQL Block Types

# PL/SQL Block Types

- The basic units that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. For example procedures and functions which known as subprograms, and anonymous blocks.

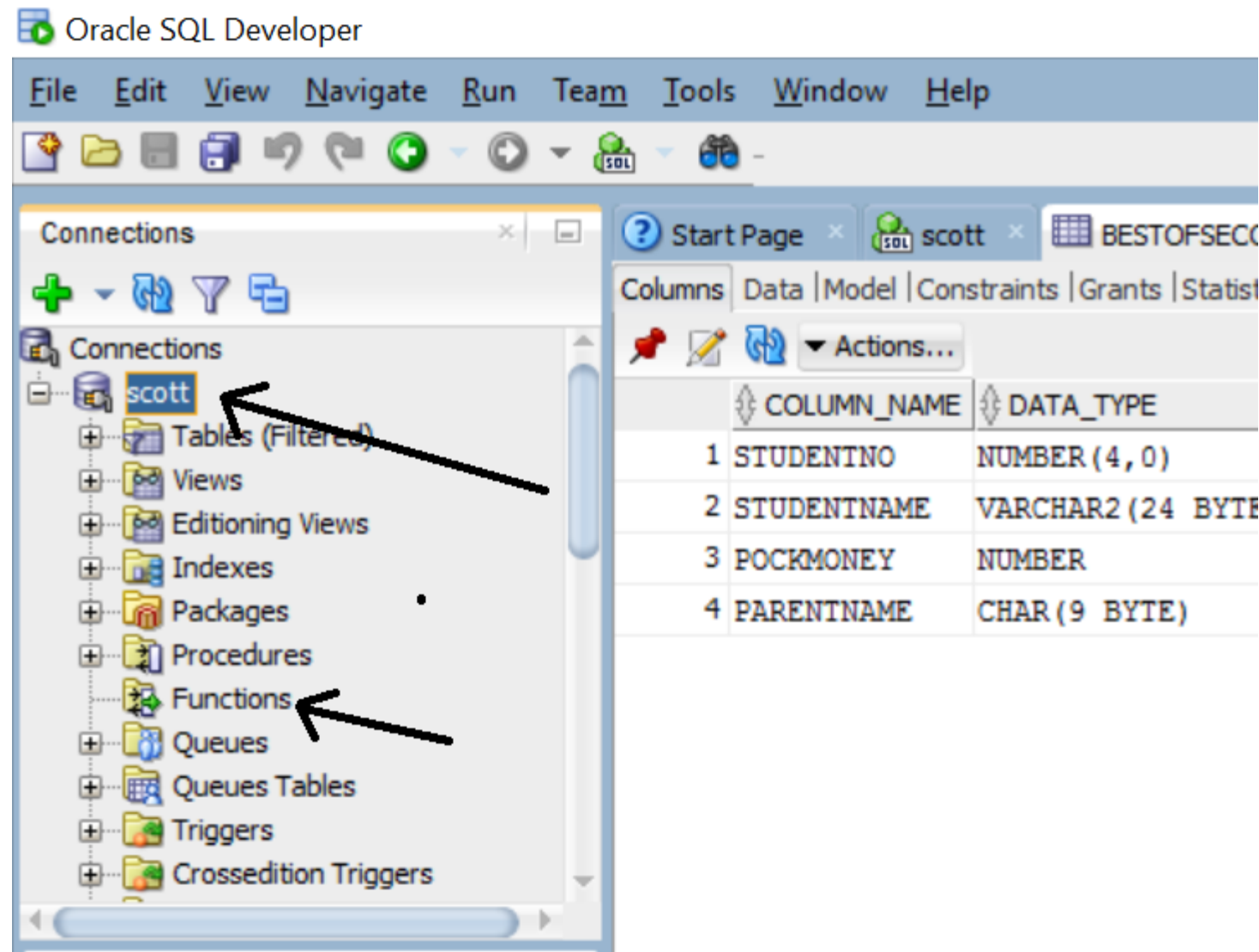
## Anonymous Blocks

- Anonymous blocks are unnamed blocks. They are declared at the point in an application where they are to be executed and are passed to the PL/SQL engine for execution at runtime means they are not stored in a database.

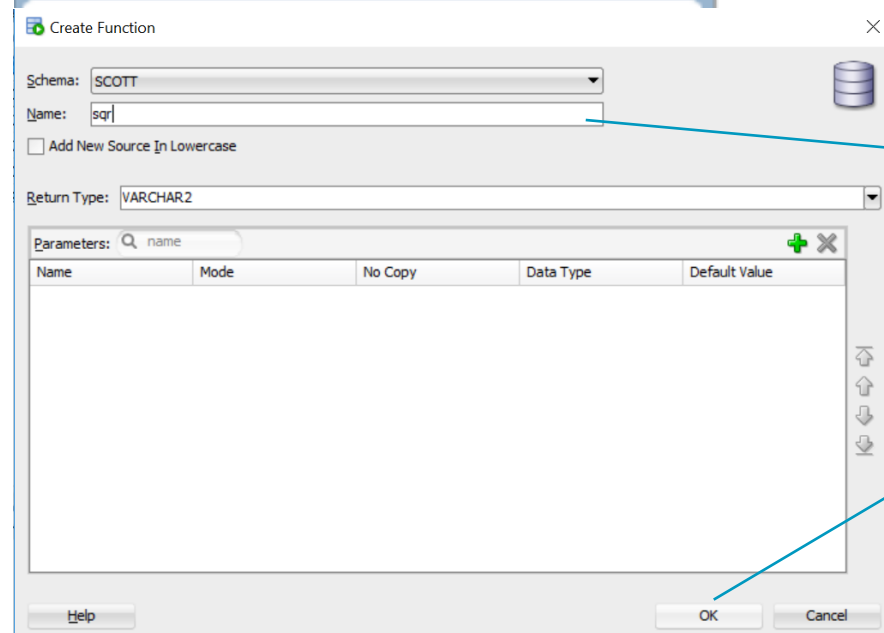
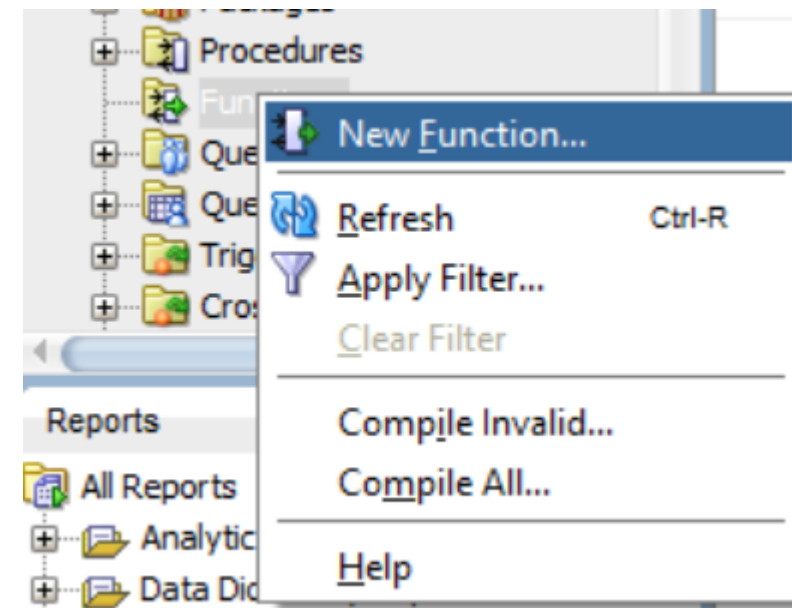
## Sub Programs

- Subprograms are named PL/SQL blocks that can take parameters and can be invoked. We can declare them either as *procedures* or as *functions*. Generally we use a procedure to perform an action and a function to compute a value.
- We can store subprograms at the server or application level.
  - Note:** A function is similar to a procedure, except that a function must return a value.
- Let do a exercise to understand the concept of Anonymous and subprograms.
- We will first create a sub program(named PL/SQL Block a function named as cube)
- Open PL/SQL Developer as we did in last classes.

# PL/SQL Block Types



- Right Click on functions and create new function with the following definition
- named function as Square Which will calculate the square of a number



Name of function

Press OK

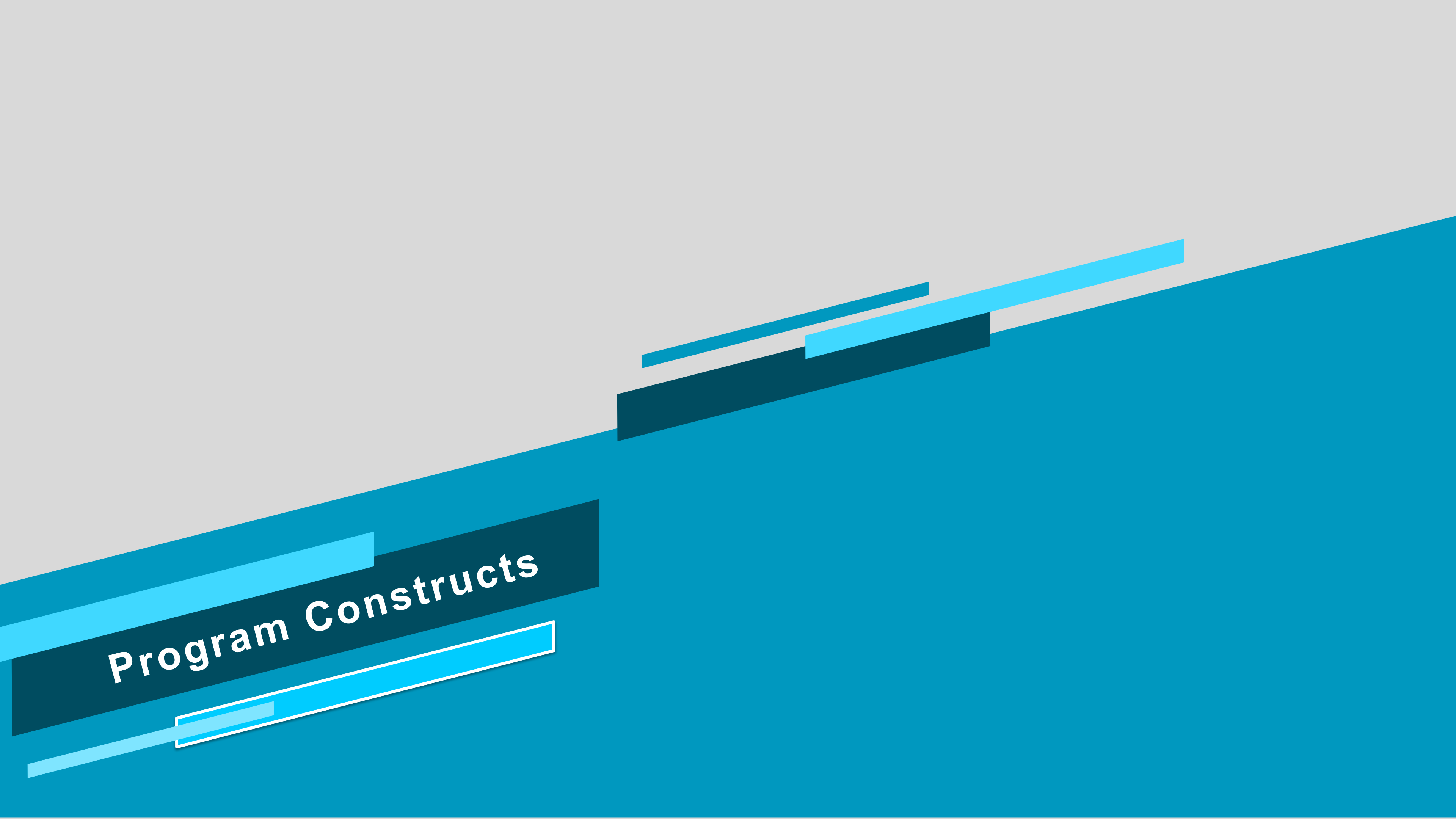
➤ Create Function video (Demonstration in class)



# PL/SQL Block Types

---

- Now let's proceed towards Anonymous block (**Anonymous block unnamed Demonstration**)
- Students before proceeding further let me discuss one thing that Oracle also provides Oracle Developer that creates forms and reports in Oracle (UI or Application level work). The work that is usually done by means of different languages like C#, VB (either using Visual Studio) or PHP (either using WordPress)
- In this lab we will discuss few points provided by Oracle Developer from application point of view (Oracle Forms & Reports)



# Program Constructs

# Programs Constructs

➤ Let see variety of different PL/SQL constructs that are used in PL/SQL Block. Their availability is based on environment in which they are executed.

<u>Program Construct</u>	<u>Description</u>	<u>Availability</u>
Anonymous block	Unnamed PL/SQL block that is embedded within an application or is issued interactively	All PL/SQL environments
Stored procedure or function	Named PL/SQL block stored in the Oracle Server that can accept parameters and can be invoked repeatedly by name.	Oracle Server
Application procedure or function	Named PL/SQL block stored in an Oracle Developer application or shared library that can accept parameters and can be invoked repeatedly by name	Oracle Developer components – for example, forms
Package	Named PL/SQL module that groups related procedures, functions and identifiers	Oracle Server and Oracle Developer components – for example, Forms
Database trigger	PL/SQL block that is associated with a database table and is fired automatically when triggered by DML statements	Oracle Server
Application trigger	PL/SQL block that is associated with an application event and is fired automatically	Oracle Developer components – for example, Forms



# Variables in PL/SQL

# Variables in PL/SQL

## Use of Variables

### Temporary storage of data

- Data can be temporarily stored in one or more variables for processing later in the data flow process.

### Manipulation of Stored values

- Variables can be used for calculations and other data manipulations without accessing the database.

### Reusability

- Once declared, variables can be used repeatedly in an application simply by referencing them in other statements

### Ease of maintenance

- Best example is %Type & %RowType Variables
- %**Type** Is used to declare a field of ***same type(data type)*** that a particular column of a particular table has.

# Variables in PL/SQL

Let see % Type Implementation

```
DECLARE
```

```
  v_EmpName emp.ename%TYPE;
```

```
BEGIN
```

```
  SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
```

```
  DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
```

```
END;
```

We have given a variable named as **V\_EmpName** the same data type that **ename** column of **emp** table has by using % Type variable. Lets run query in SQL Plus:

**Set ServerOutput On**

```
1 DECLARE
2     v_EmpName  emp.ename%TYPE;
3 BEGIN
4     SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
5     DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
6* END;
SQL> /
Name = SMITH

PL/SQL procedure successfully completed.
```

# Variables in PL/SQL

% RowType is used to declare a **record** with **same type** as present in a **specified table** of a database  
Let see % RowType Implementation

DECLARE

  v\_emp emp%ROWTYPE;

BEGIN

  v\_emp.empno := 10;

  v\_emp.ename := 'Hammad';

END;

Now above v\_emp structure can be use wherever we want to use.

## Handling variables

- Declare and initialize variable in the declaration section
- Assign new values to variables in the executable section.
- Pass values into PL/SQL subprograms through parameters. There are three parameter modes, IN (default), OUT and IN OUT
- The **IN** parameter is used to pass values to the subprogram being called.
- The **OUT** parameter is used to return values to the caller of a subprogram.
- The **IN OUT** parameter is used to pass initial values to the subprogram being called and to return updated values to the caller.
- View the results from a PL/SQL block through output variables

**IN & OUT Variable Demo**  
**In class**

# Variables in PL/SQL

- The **IN OUT** parameter can be understood by the following practical example: let's create a function that has an in out type variable as:

```
create or replace FUNCTION inout_fn (outparm IN OUT VARCHAR2)
RETURN VARCHAR2 IS
```

```
BEGIN
  outparm := 'Coming out';
  RETURN 'return param';
END inout_fn;
```

- Use the above function as follows to understand IN OUT Concept

```
DECLARE
  retval VARCHAR2(20);
  ioval  VARCHAR2(20) := 'Going in';
BEGIN
  DBMS_OUTPUT.put_line('In: ' || ioval);
  retval := inout_fn(ioval);
  DBMS_OUTPUT.put_line('Out: ' || ioval);
  DBMS_OUTPUT.put_line('Return: ' || retval);
END;
```

In: Going in

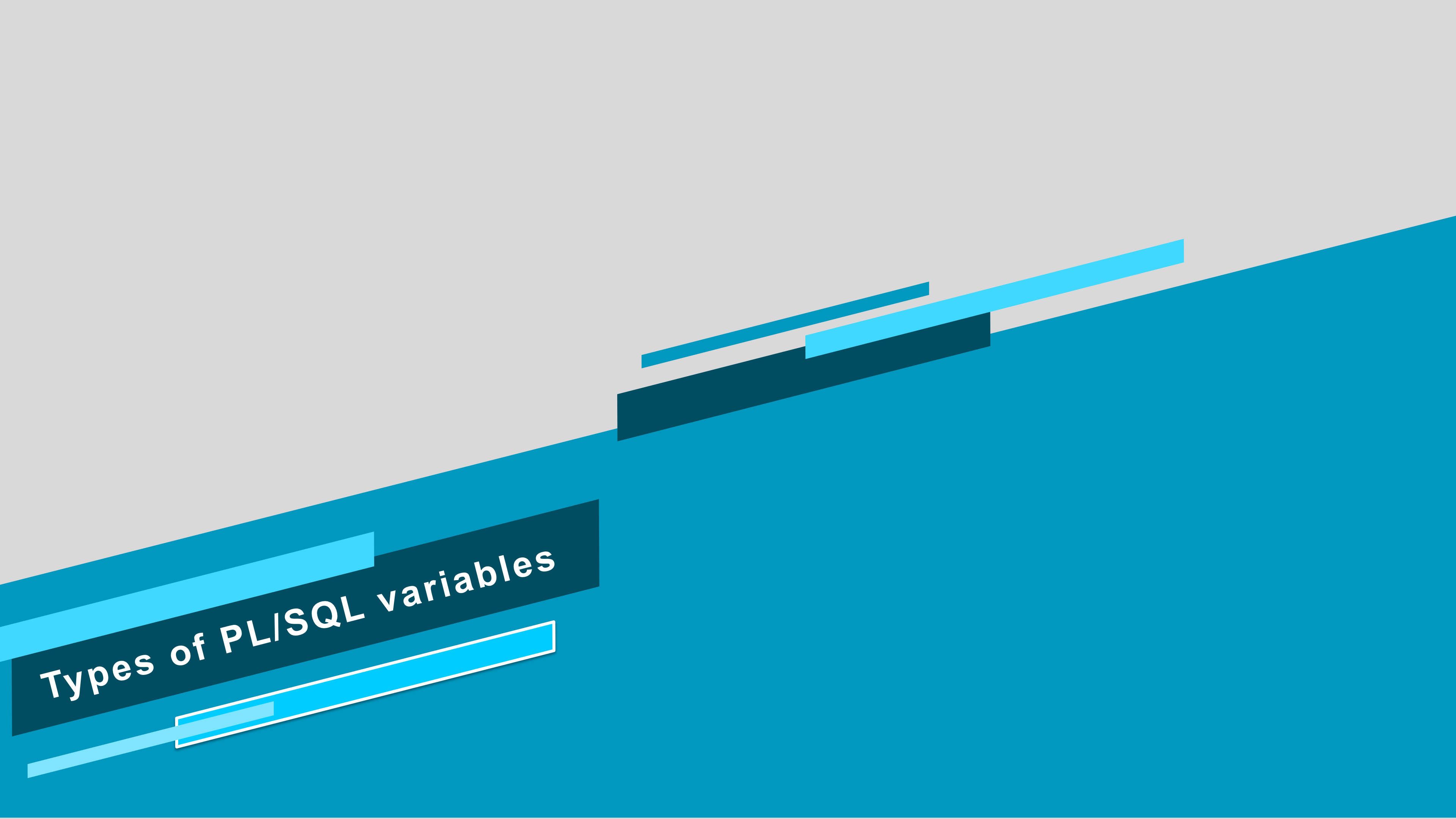
Out: 'Coming Out'

Return: 'Return Param'

Parameter value updated as 'Coming out'

**INOUT Variable Demo in class**





# Types of PL/SQL variables

# Types of PL/SQL Variables

## Scalar

Holds a single value and has no internal components

Scalar data types can be classified into four categories: number, character, date and Boolean.

## Composite

Composite types have internal components that can be manipulated individually, such as the elements of an array, record, or table. Lets take a example of composite Staff\_list made by emp table of scott.

DECLARE

TYPE **staff\_list** IS TABLE OF **emp**.empno%TYPE;

**staff** **staff\_list**;

**salary** emp.sal%TYPE;

**deptno** emp.deptno%TYPE;

BEGIN

staff := staff\_list(7369, 7499, 7698, 7654, 7521);

FOR i IN **staff**.FIRST..**staff**.LAST LOOP

SELECT **sal**, **deptno** INTO **salary**, **deptno** FROM **emp**

WHERE **emp**.empno = **staff**(i);

DBMS\_OUTPUT.PUT\_LINE (TO\_CHAR(**staff**(i)) ||

: ' || salary || ', ' || deptno );

END LOOP;

END;

## Output

7369: 800, 20  
7499: 1600, 30  
7698: 2850, 30  
7654: 1250, 30  
7521: 1250, 30

Composite  
variable Demo in  
class

# Types of PL/SQL Variables

## Reference

Holds values, called pointers that designate other program items

### Example:

```
variable dept_sel REFCURSOR  
BEGIN
```

Definition of cursor

```
    OPEN :dept_sel FOR SELECT * FROM DEPT;
```

```
    END;
```

```
    print dept_sel
```

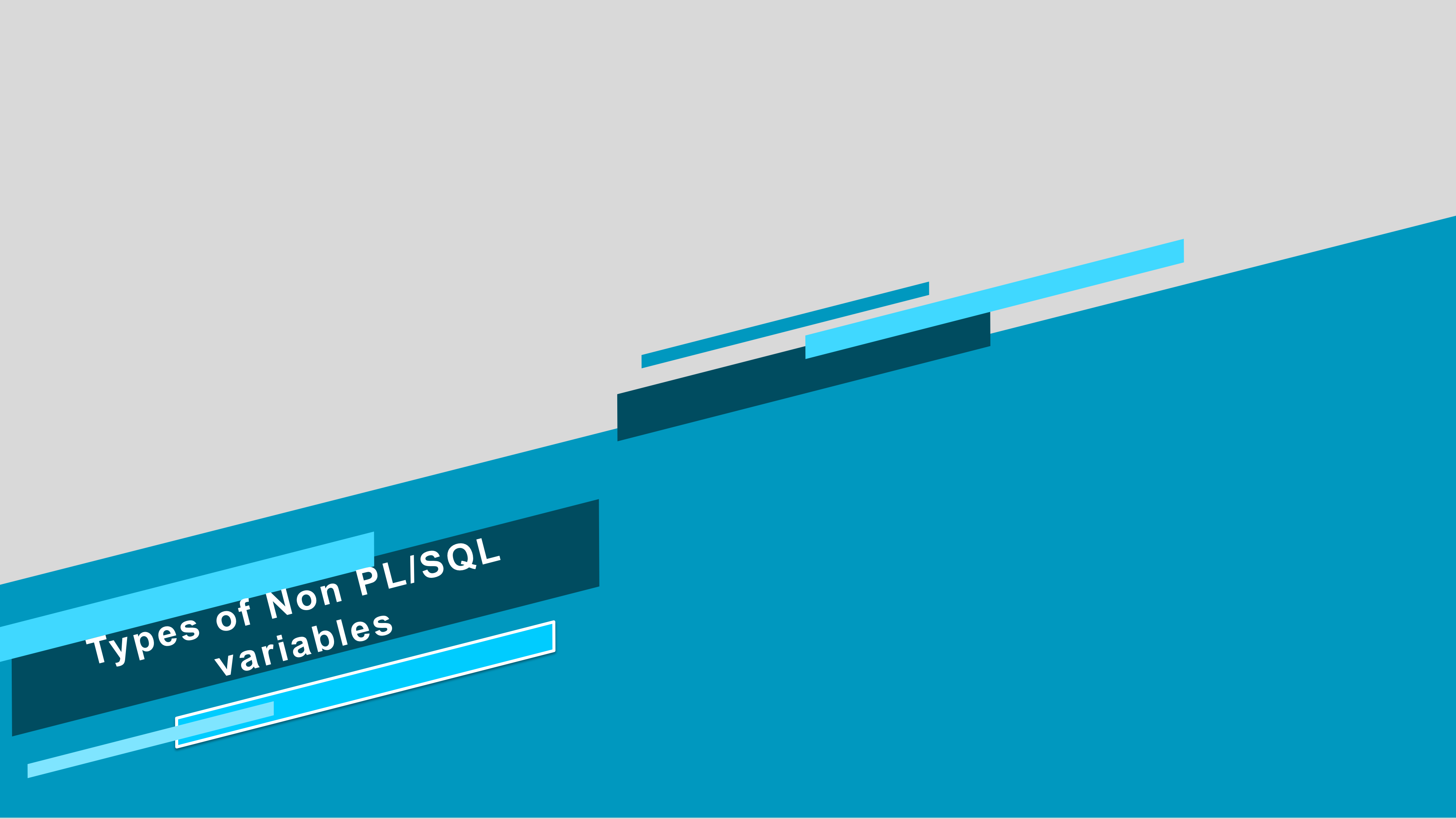
Pointing Cursor

Printing Cursor

## LOB(large objects)

**Reference variable Demo  
In class**

Hold values, called locators that specify the location of large objects (graphics, images) that are stored somewhere in the memory



# Types of Non PL/SQL variables

# Types of Non-PL/SQL Variables

The term non PL/SQL means the other variables on which PL SQL can Rely.

- Host language variables declared in precompiled programs (binding application variables to PL/SQL query or variables)
- Screen fields in Forms applications ( directly binding screen inputs to PL/SQL query or variables)
- SQL\*Plus host variables: PL/SQL does not have input/output capability of its own. In order to pass values into and out of a PL/SQL block, it is necessary to rely on the environment in which PL/SQL is executing. SQL\*Plus host (or bind) variables can be used to pass runtime values out of the PL/SQL block back to the SQL\*Plus environment. You can reference them in a PL/SQL block with a preceding column.

Example:

```
SELECT EMPNO, LASTNAME, WORKDEPT  
  INTO : CBLEMPO, :CBLNAME, :CBLDEPT  
FROM CORPDATA.EMPLOYEE  
WHERE EMPNO = :EMPID
```

In this example, the host variable **CBLEMPO** receives the value from **EMPNO**, **CBLNAME** receives the value from **LASTNAME**, and **CBLDEPT** receives the value from **WORKDEPT**.



# Declaring PL/SQL Variable

# Declaring PL/SQL Variable

- It is necessary to declare all PL/SQL identifiers in the declaration section before referencing them in the PL/SQL block. It is not necessary to assign a value to a variable at the time of declaration.

## Syntax

*identifier* **[CONSTANT]** *datatype* **[NOT NULL]** **[:= | DEFAULT** *expr* **];**

A constant holds a value that once declared and cant be changed

Constraints can be defined

## Examples

```
v_hiredate DATE;  
v_deptno NUMBER(2) NOT NULL := 10;  
v_location VARCHAR2(13) := 'Atlanta';  
c_comm. CONSTANT NUMBER := 1400;  
v_count BINARY_INTEGER := 0;  
v_orderdate DATE := SYSDATE + 7;  
pi constant number := 3.141592654;
```

## GuideLines

- Name the identifier according to the same rules used for SQL objects.

# Declaring PL/SQL Variable

## GuideLines(cont.)

- Use naming conventions – for example, **v\_name** to represent a variable, **g\_name** to represent global variables and **c\_name** to represent a constant variable.
- If NOT NULL constraint is used, it is needed to assign a value.
- Declaring only one identifier per line makes code more easy to read and maintain.
- Identifiers must not be longer than 30 characters. The first character must be a letter; the remaining characters can be letters, numbers, or special symbols.
- Another way to assign values to variables is to select or fetch database values into it (already discussed). The following example, computes a 10% bonus on the salary of an employee:-

```
SELECT SAL * 0.10  
INTO V_BONUS  
FROM EMP  
WHERE EMPNO = 7369;
```

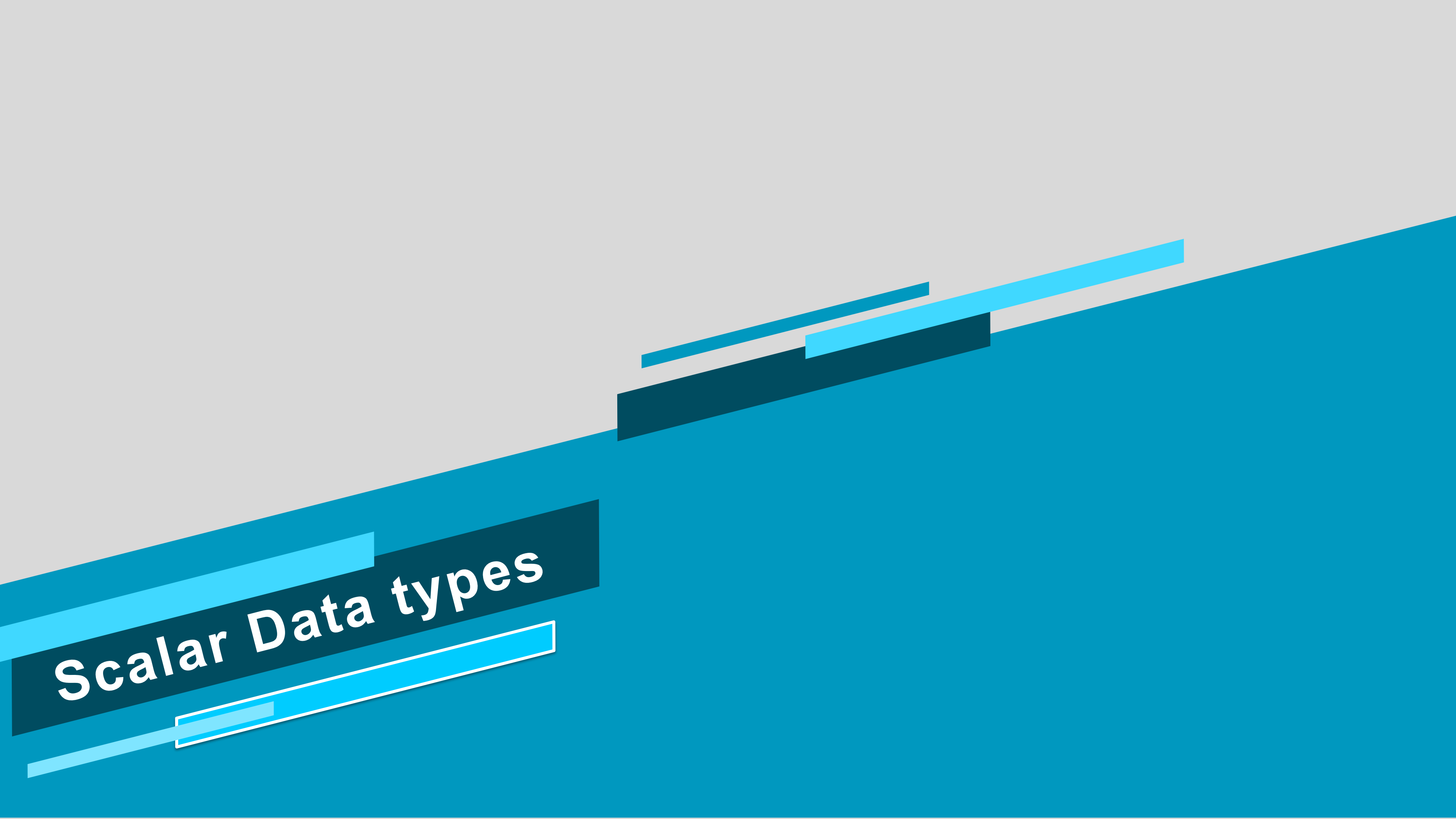
- V\_BONUS is variable bonus which will use in another computation or storage.



# Declaring PL/SQL Variable

---

- By default, variables are initialized to NULL.
- Assignment operator is used for non typical values. We use DEFAULT keyword instead of assignment operator to initialize variables for typical values(greater than a block size =1024 bytes).  
g\_mgr NUMBER(4) **DEFAULT** 7839;
- NOT NULL Constraint (Same description as we studied previously)

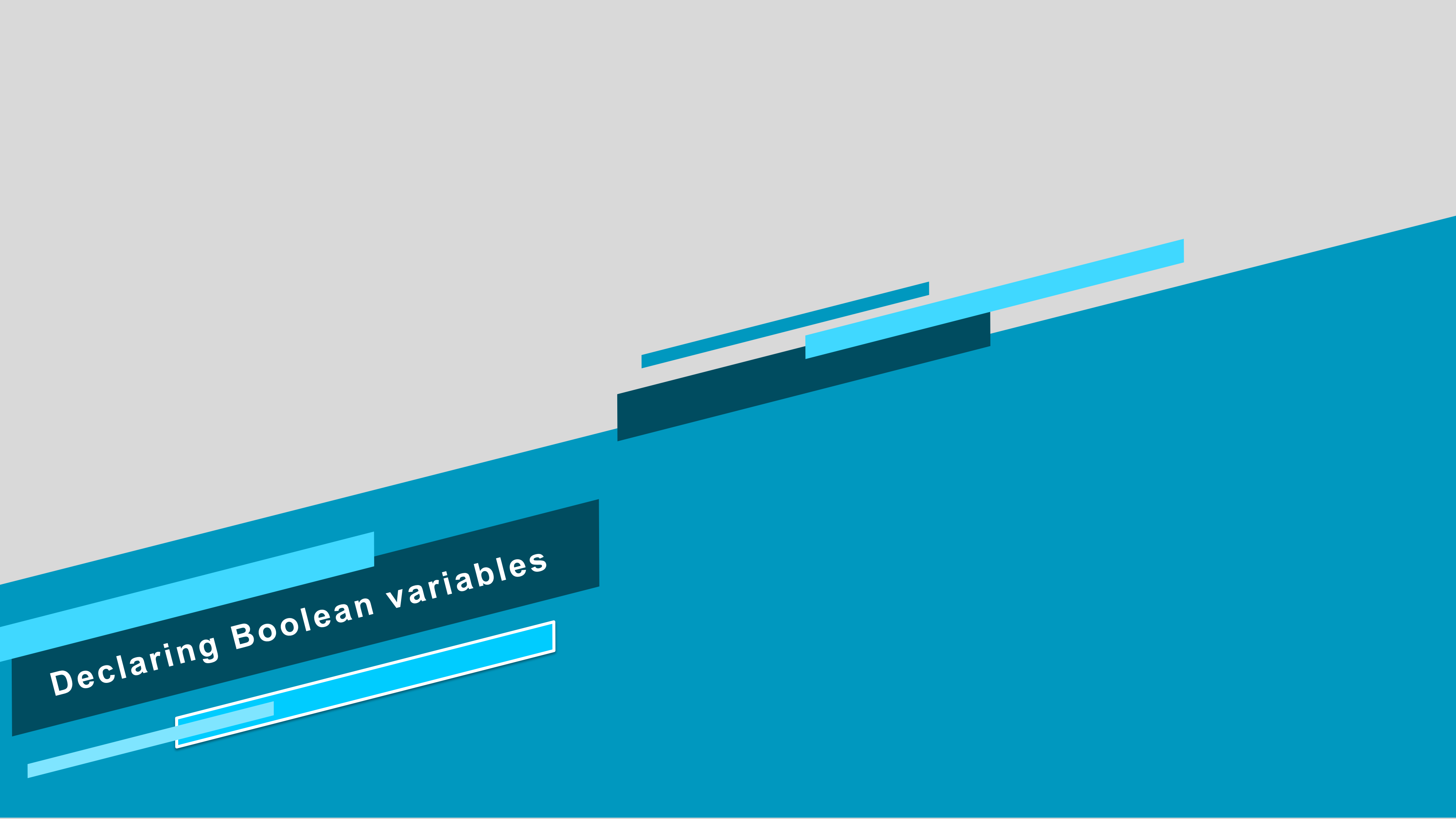


# Scalar Data types

# Scalar DataTypes

DATATYPE	DESCRIPTION
VARCHAR2(maximum length)	Base type for variable-length character data up to 32,767 bytes . There is no default size for VARCHAR2 variables and constants.
CHAR(maximum Length )	Base type for fixed-length character data upto 32, 760 bytes. If the maximum length is not specified, the default length is set to 1.
NUMBER(p, s)	Base type for fixed and floating-point numbers.
DATE	Date and time values between January 1, 4712 B.C. (Before Christ)and December 31, 9999 A.D.
LONG RAW	Base type for binary data and byte strings up to 32,760 bytes.LONG RAW data is not interpreted by PL/SQL.
LONG	Base type for variable-length character data up to 32,760 bytes. The maximum width of a LONG datSabase column is 2,147,483,647 bytes.
BOOLEAN	Base type that stores one of three possible values used for logical calculations TRUE, FALSE, or NULL.
BINARY INTEGER	Base type for integers between –2,147,483,647 and 2,147,483,647.
PLS_INTEGER	Base type for signed integers between –2,147,483,647 and 2,147,483,647. PLS_INTEGER values require less storage and are faster than NUMBER and BINARY_INTEGER values.

➤ For LOB datatype variable (we studied in depth in Lab 08)



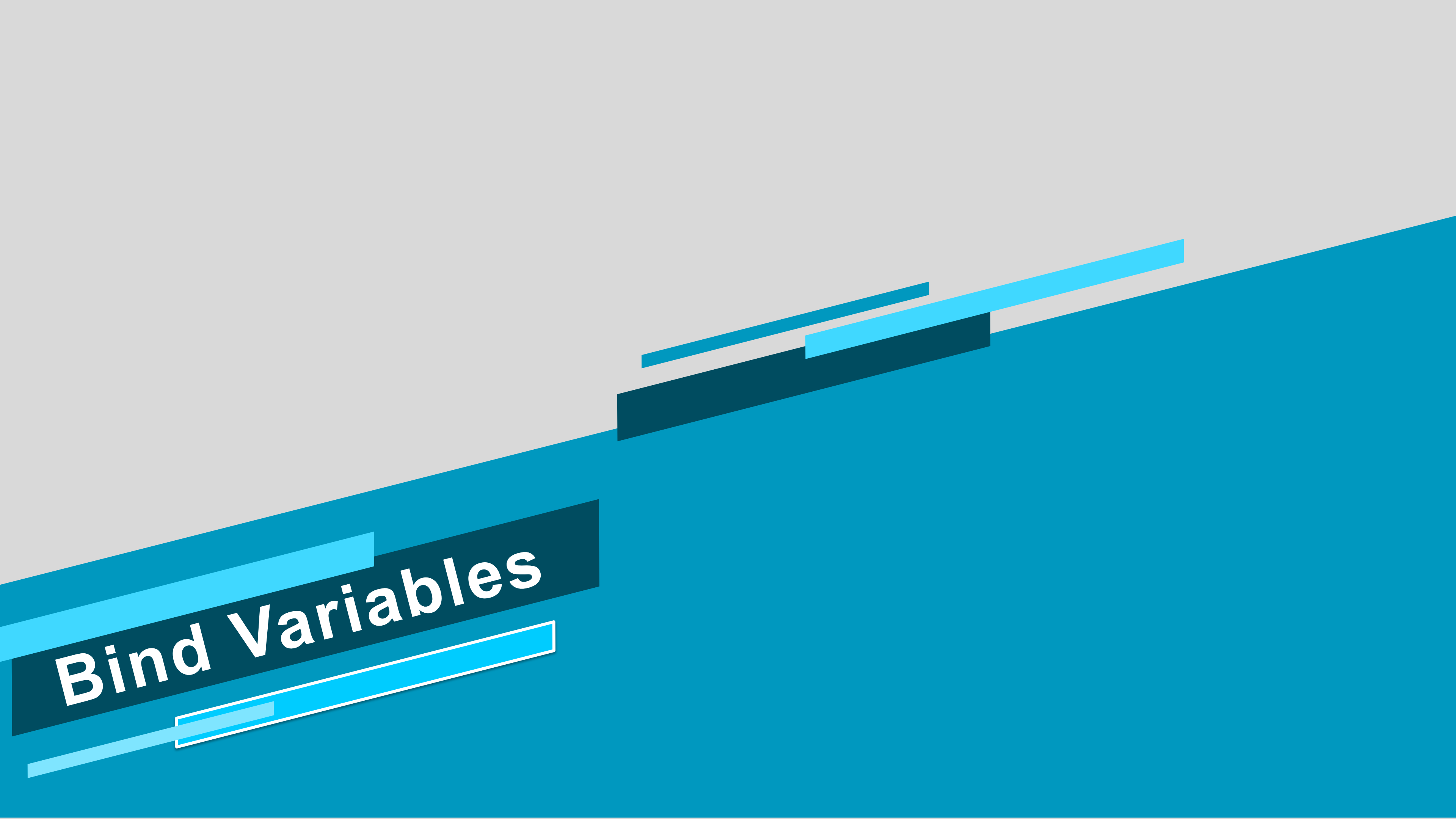
# Declaring Boolean variables

# Declaring Boolean Variables

---

- In PL/SQL only True ,False & Null Value can be assigned to variable.
- We can compare variables in both SQL & PL/SQL statements with Boolean on output.
- To declare & initialize a Boolean variable we can write

```
V_Comm_Sal BOOLEAN := (V_SAL<V_SAL2);
```



# Bind Variables

# BIND VARIABLE

- **Bind variables** are **variables** you create in **SQL\*Plus** and then reference in PL/SQL.

## Creating Bind Variable

- We declare bind variable in the SQL\*Plus environment by using **VARIABLE** command. For example, to declare a variable named as *return\_code* of type NUMBER :-

```
VARIABLE return_code NUMBER;
```

- Similarly to declare a variable named as *return\_msg* of type VARCHAR2 :-

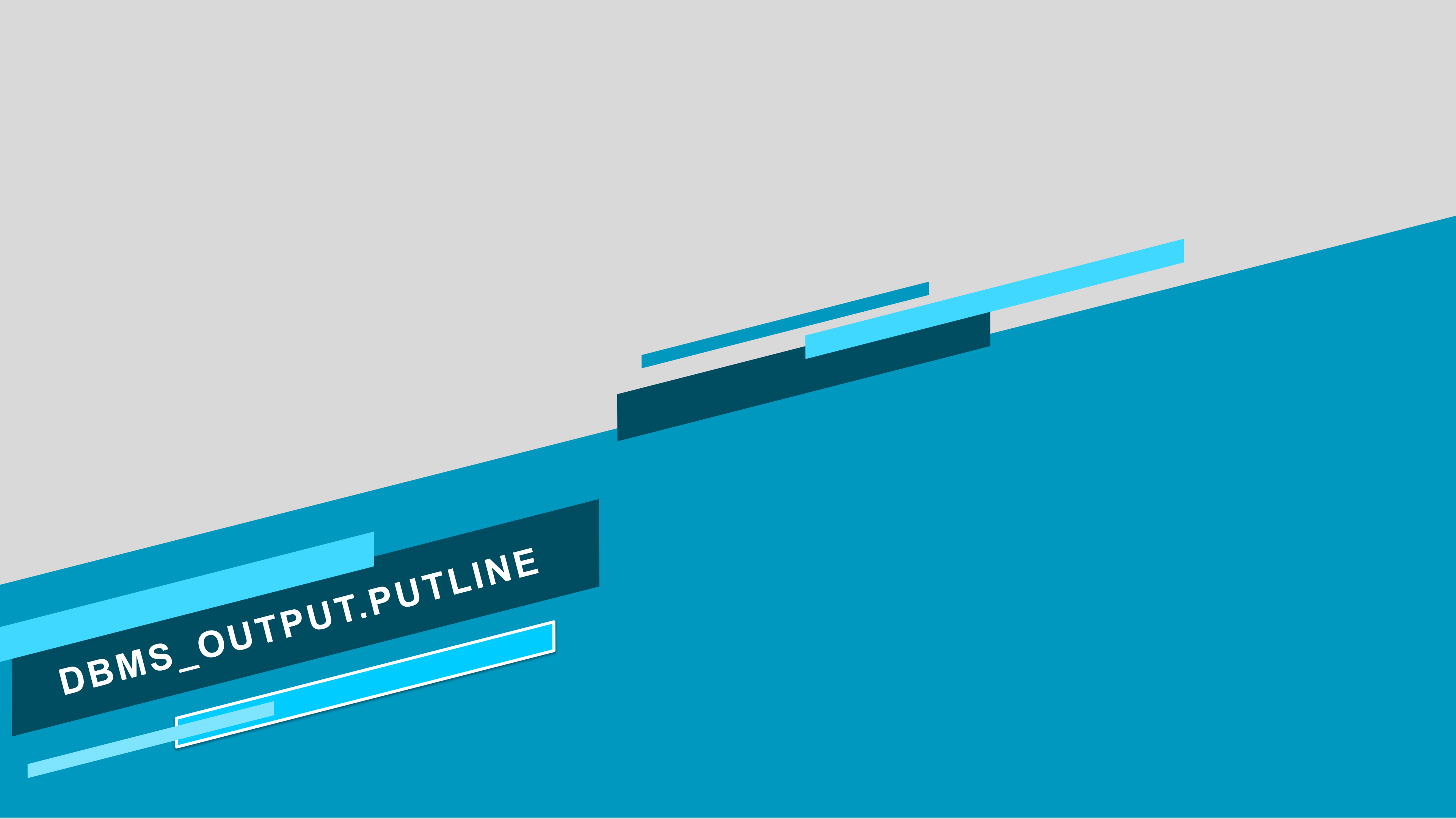
```
VARIABLE return_msg varchar2(30);
```

## Displaying Bind Variable

To display the current value of bind variables in the SQL\*Plus environment, we use the command PRINT. To print *return\_code* :

Print *return\_code*





DBMS\_OUTPUT.PUTLINE



# DBMS\_OUTPUT.PUT\_LINE

- This command is used to print the information from **PL/SQL** Block we have already cover this in our examples.

## Example

The following script computes the monthly salary and prints it to the screen, using **DBMS\_OUTPUT.PUT\_LINE**

```
VARIABLE g_monthly_salary NUMBER
```

```
SET SERVEROUTPUT ON
```

```
ACCEPT p_annual_sal PROMPT 'Please enter the annual salary :'
```

```
DECLARE
```

```
v_sal NUMBER(9, 2) := &p_annual_sal;
```

```
BEGIN
```

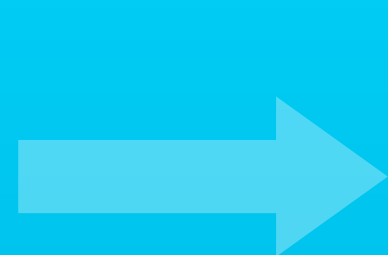
```
    V_sal := v_sal / 12;
```

```
DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' || TO_CHAR(v_sal));
```

```
END;
```

Accept input variable  
Value is set up by using :=&  
The value input by user at runtime

**DBMS\_OUTPUT PUT\_LINE Demo in class**



Finished