# 4

# Writing Executable Statements

# Course Road Map

Lesson 1: Course Overview

**Unit 1: Introducing PL/SQL**

Unit 2: Programming with PL/SQL

Unit 3: Working with PL/SQL Code

Lesson 2: PL/SQL Overview

Lesson 3: Declaring PL/SQL Variables

**Lesson 4: Writing Executable Statements**

**You are here!**

Lesson 5: Using SQL Statements in PLSQL Programs

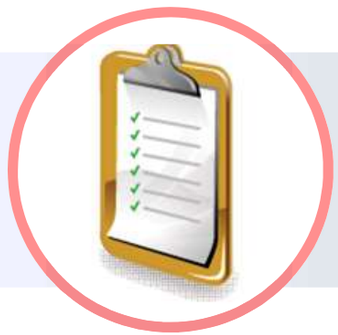
# Objectives

After completing this lesson, you should be able to do the following:

- Identify the lexical units in a PL/SQL block
- Use built-in SQL functions in PL/SQL
- Describe when implicit conversions take place and when explicit conversions have to be dealt with
- Write nested blocks and qualify variables with labels
- Write readable code with appropriate indentation
- Use sequences in PL/SQL expressions

# Agenda

- Writing executable statements in a PL/SQL block
- Writing nested blocks
- Using operators and developing readable code



# Lexical Units in a PL/SQL Block

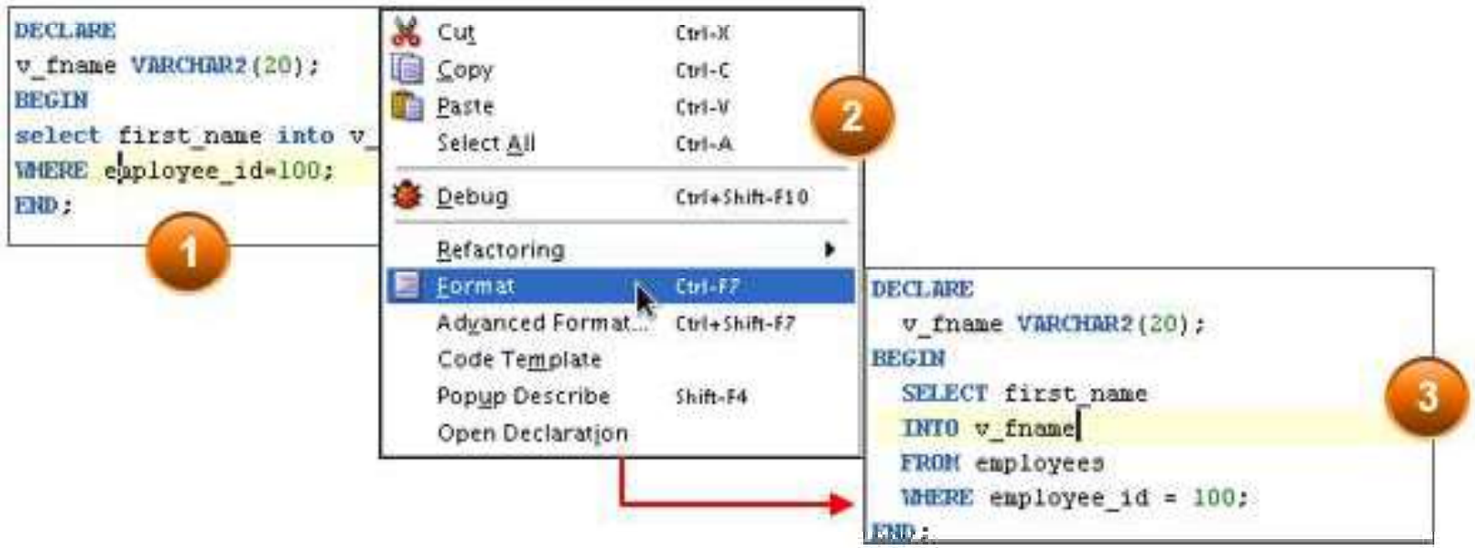
•Lexical units:

- Are building blocks of any PL/SQL block
- Are sequences of characters, including letters, numerals, tabs, spaces, returns, and symbols
- Can be classified as:
  - Identifiers: `v_fname, c_percent`
  - Delimiters: `; , +, -`
  - Literals: `John, 428, True`
  - Comments: `--, /* */`

# PL/SQL Block Syntax and Guidelines

- Using Literals
  - Character and date literals must be enclosed in single quotation marks.
  - Numbers can be simple values or in scientific notation.
- Formatting Code: Statements can span several lines.

```
v_name := 'Henderson';
```



# Commenting Code

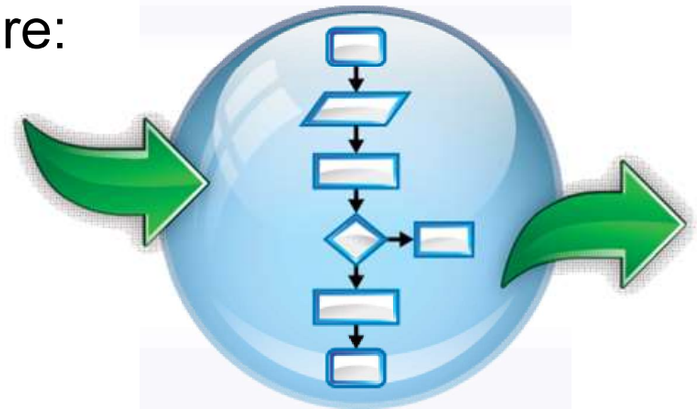
- Prefix single-line comments with two hyphens (--).
- Place a block comment between the symbols /* and */.

**Example:**

```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary
DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

# SQL Functions in PL/SQL

- Predefined functions that are used in SQL can also be used in PL/SQL.
- Functions that are available in procedural statements are:
  - Single-row functions
  - Built-in functions with Strings
  - Built-in functions with Numbers
  - Built-in functions with Dates
- Functions that are not available in procedural statements are:
  - `DECODE`
  - Group functions



# SQL Functions in PL/SQL: Examples

- Get the length of a string:

```
v_desc_size INTEGER(5);
v_prod_description VARCHAR2(70):='You can use this product with your radios
for higher frequency';

-- get the length of the string in prod_description
v_desc_size:= LENGTH(v_prod_description);
```

- Get the number of months an employee has worked:

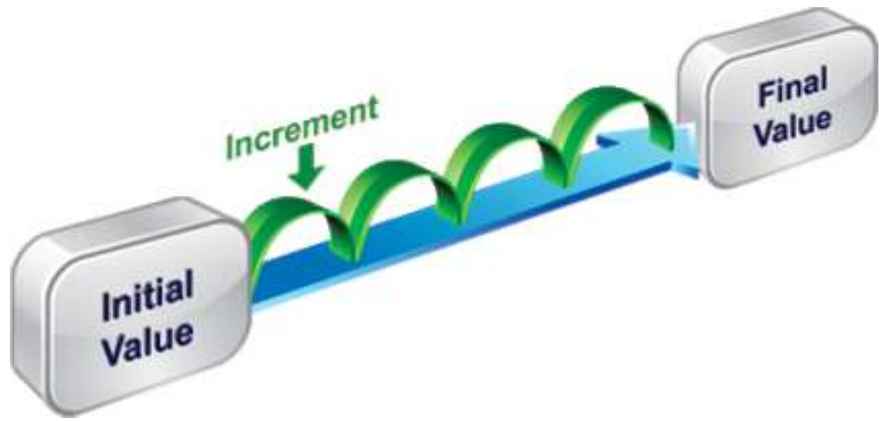
```
v_tenure:= MONTHS_BETWEEN (CURRENT_DATE, v_hiredate);
```

# Using Sequences in PL/SQL blocks

# Using Sequences in PL/SQL blocks

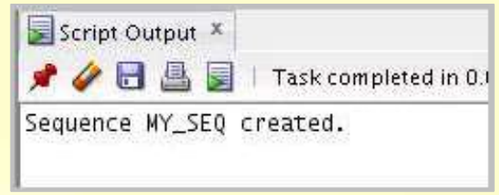
- Sequences are database objects that can be used by multiple users to generate sequential numbers.
- Sequences can be created through the `CREATE SEQUENCE` statement.

```
CREATE SEQUENCE emp_sequence
INCREMENT BY 1
START WITH 1
NOMAXVALUE;
```

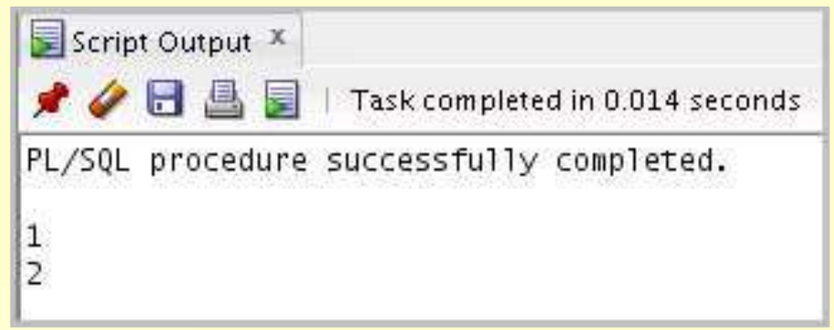


# Using Sequences in PL/SQL Blocks

```
CREATE SEQUENCE my_seq
INCREMENT BY 1
START WITH 1
NOMAXVALUE;
```

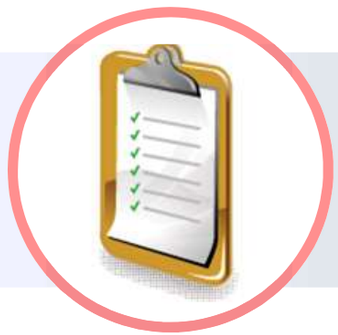



```
DECLARE
  v_new_id NUMBER;
BEGIN
  v_new_id := my_seq.NEXTVAL;
  DBMS_OUTPUT.PUT_LINE(v_new_id);
  DBMS_OUTPUT.PUT_LINE(my_seq.NEXTVAL);
END;
```



# Agenda

- Writing executable statements in a PL/SQL block
- Writing nested blocks
- Using operators and developing readable code

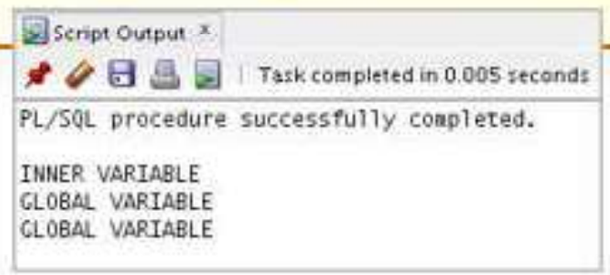


# Nested blocks

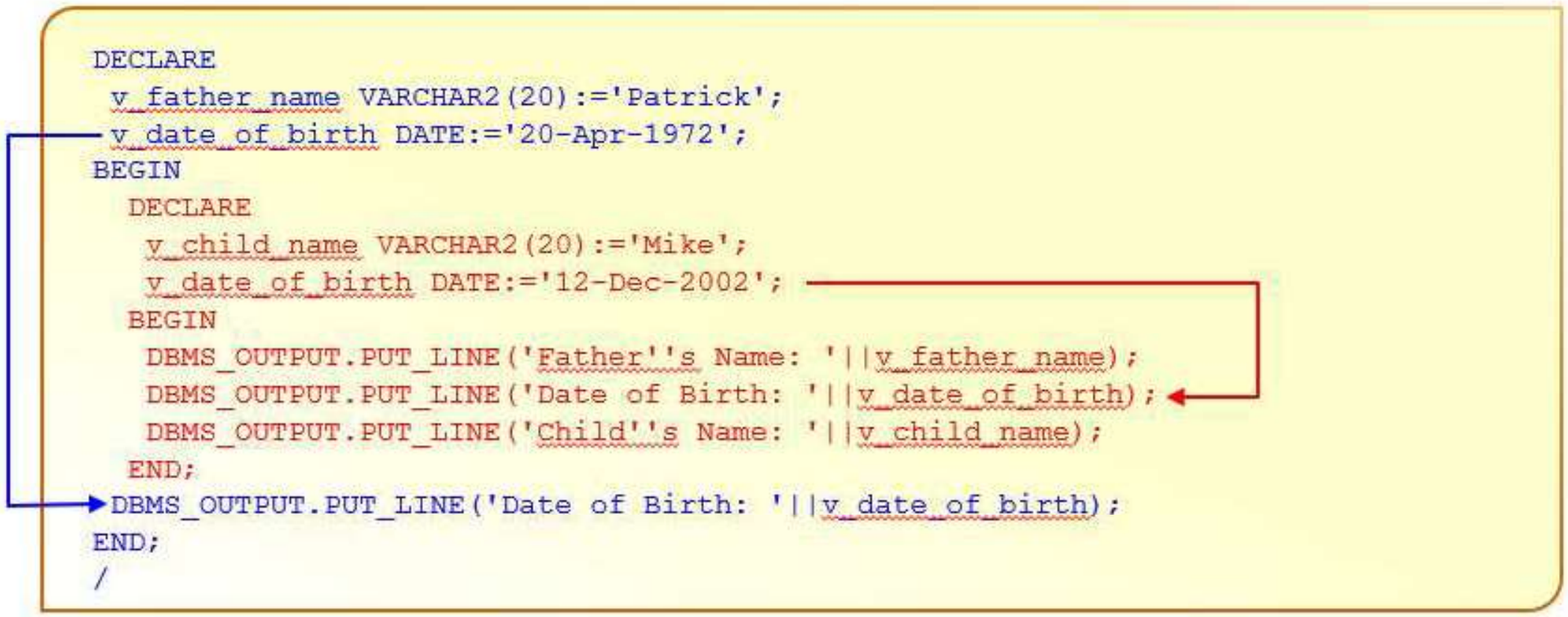
- PL/SQL blocks can be nested.
  - An executable section (`BEGIN … END`) can contain nested blocks.
  - An exception section can contain nested blocks.

# Nested Blocks: Example

```
DECLARE
 v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
   v_inner_variable VARCHAR2(20):='INNER VARIABLE';
  BEGIN
   DBMS_OUTPUT.PUT_LINE(v_inner_variable);
   DBMS_OUTPUT.PUT_LINE(v_outer_variable);
  END;
 DBMS_OUTPUT.PUT_LINE(v_outer_variable);
END;
```



# Variable Scope and Visibility

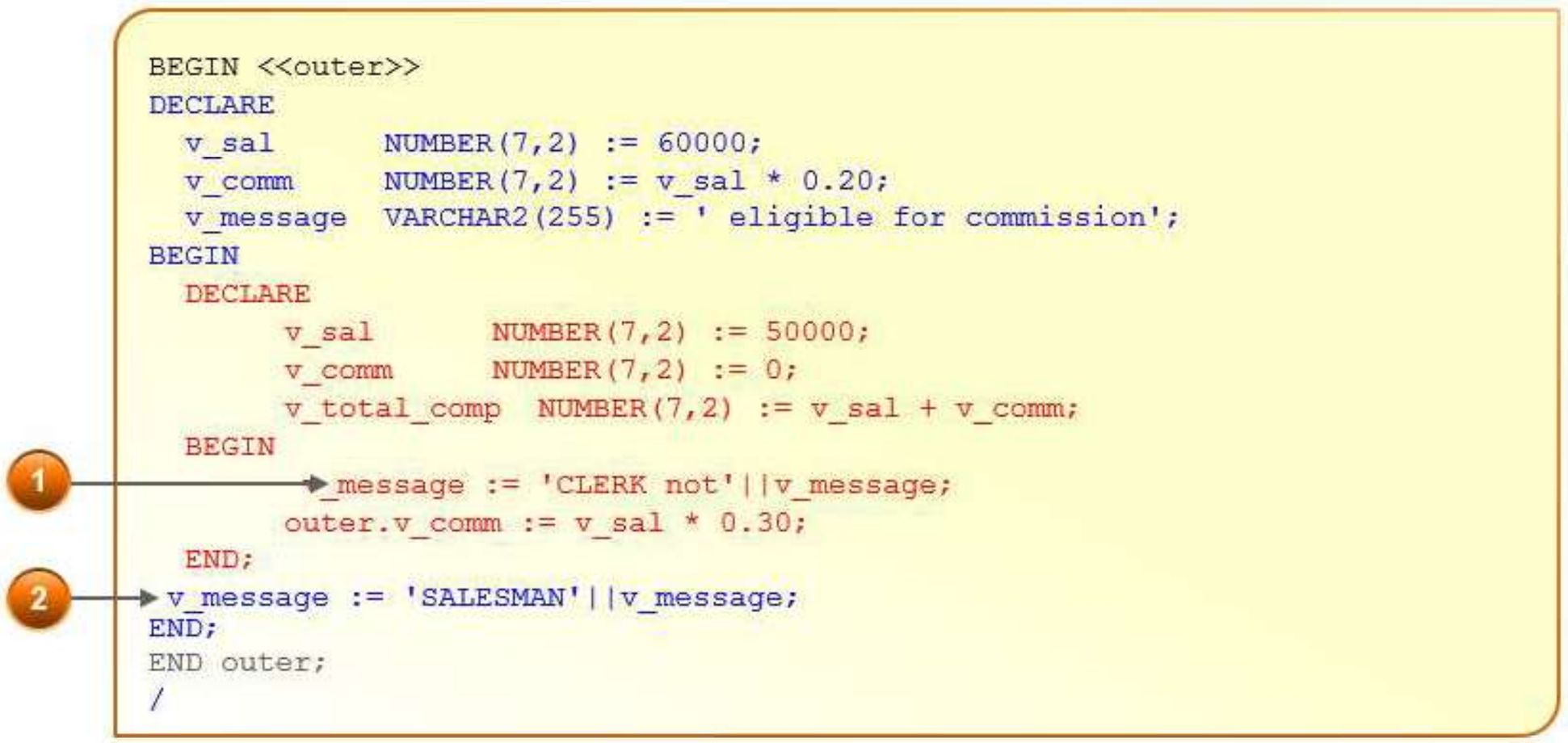


```
DECLARE
 v_father_name VARCHAR2(20):='Patrick';
 v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
   v_child_name VARCHAR2(20):='Mike';
   v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
   DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
   DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
  END;
 DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
END;
/
```

# Using a Qualifier with Nested Blocks

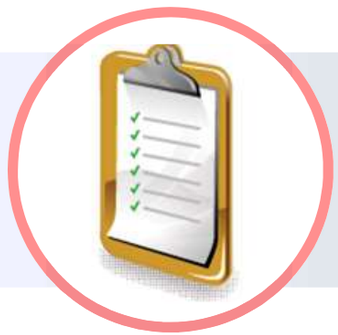
```
BEGIN <<outer>>
DECLARE
 v_father_name VARCHAR2(20):='Patrick';
 v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
   v_child_name VARCHAR2(20):='Mike';
   v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
   DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '
                        ||outer.v_date_of_birth);
   DBMS_OUTPUT.PUT_LINE('Child''s Name: '||v_child_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);
  END;
END;
END outer;
```

# Challenge: Determining the Variable Scope



# Agenda

- Writing executable statements in a PL/SQL block
- Writing nested blocks
- Using operators and developing readable code



# Operators in PL/SQL

- Logical
- Arithmetic
- Concatenation
- Parentheses to control order of operations

Same as in SQL

- Exponential operator (**)

# Operators in PL/SQL: Examples

- Increment the counter for a loop.

```
loop_count := loop_count + 1;
```

- Set the value of a Boolean flag.

```
good_sal := sal BETWEEN 50000 AND 150000;
```

- Validate whether an employee number contains a value.

```
valid  := (empno IS NOT NULL);
```

# Programming Guidelines

- Make code maintenance easier by:
  - Documenting the code with comments
  - Developing a case convention for the code
  - Developing naming conventions for identifiers and other objects
  - Enhancing readability by indenting

# Indenting Code

For clarity, indent each level of code.

```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
/
```

```
DECLARE
 v_deptno        NUMBER(4);
 v_location_id   NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    v_deptno,
          v_location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

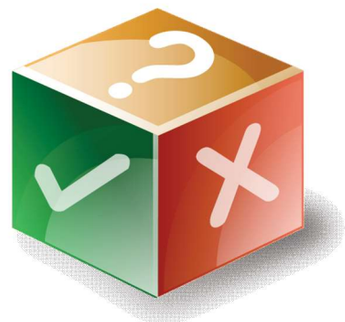
# Quiz

Q

You can use most single-row SQL functions such as number, character, conversion, and date in PL/SQL expressions.

a. True

b. False



# Summary

In this lesson, you should have learned how to:

- Identify the lexical units in a PL/SQL block
- Use built-in SQL functions in PL/SQL
- Write nested blocks to break logically related functionalities
- Decide when to perform explicit conversions
- Qualify variables in nested blocks
- Use sequences in PL/SQL expressions

# Practice 4: Overview

This practice covers the following topics:

- Reviewing scoping and nesting rules
- Writing and testing PL/SQL blocks