Writing Executable Statements

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

Lexical Units in a PL/SQL Block

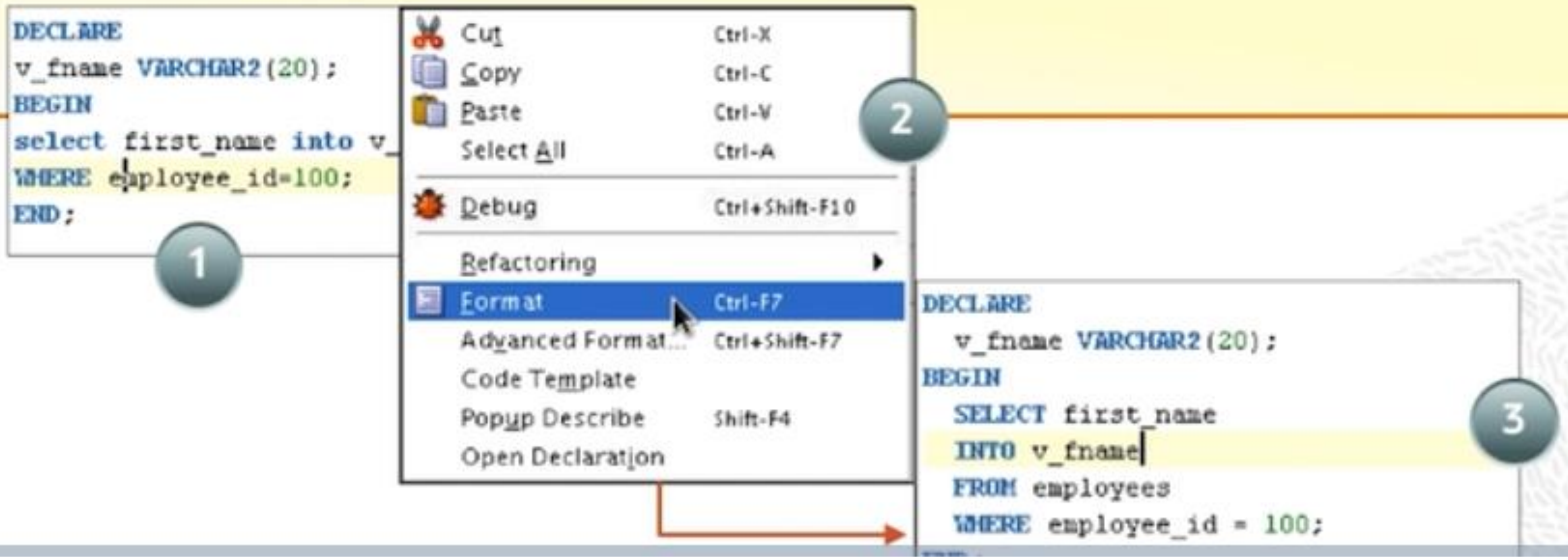
Lexical units:

- 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';
```



```
DECLARE
v_fname VARCHAR2(20);
BEGIN
select first_name into v_
WHERE employee_id=100;
END;
```

```
DECLARE
  v_fname VARCHAR2(20);
BEGIN
  SELECT first_name
  INTO v_fname
  FROM employees
  WHERE employee_id = 100;
```

Commenting Code

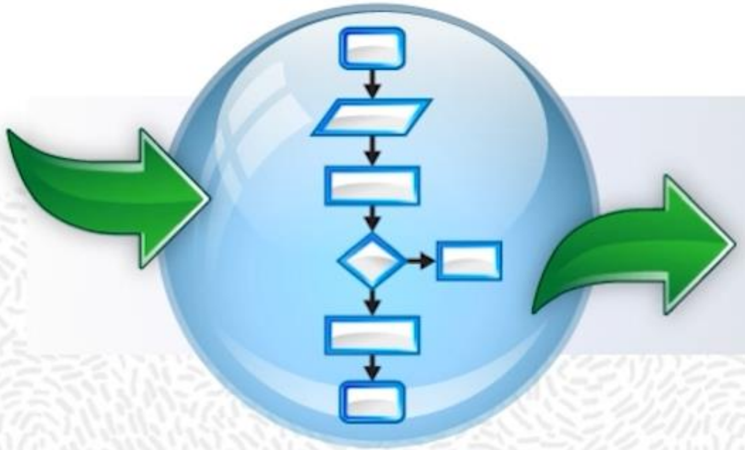
- Prefix single-line comments with two hyphens (--).
- Place a block comment between the symbols /* and */.

Example:

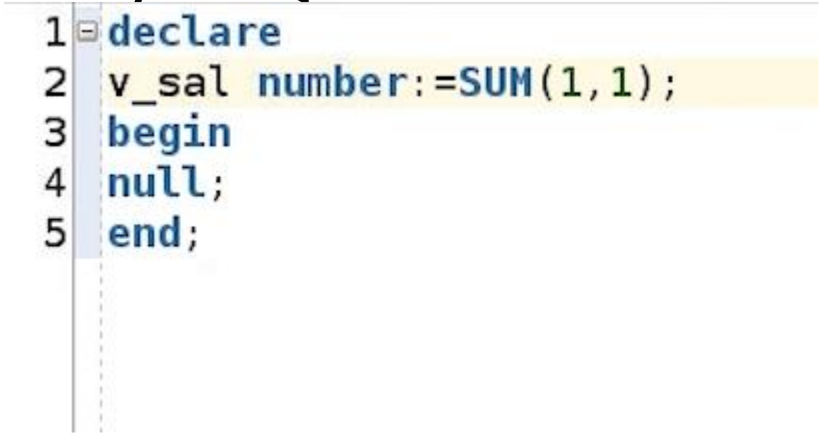
```
DECLARE
...
v_annual_sal NUMBER (9,2);
BEGIN
/* Compute the annual salary based on the
   monthly salary input from the user */
v_annual_sal := monthly_sal * 12;
--The following line displays the annual salary DBMS_OUTPUT.PUT_LINE(v_annual_sal);
END;
/
```

SQL Functions in PL/SQL

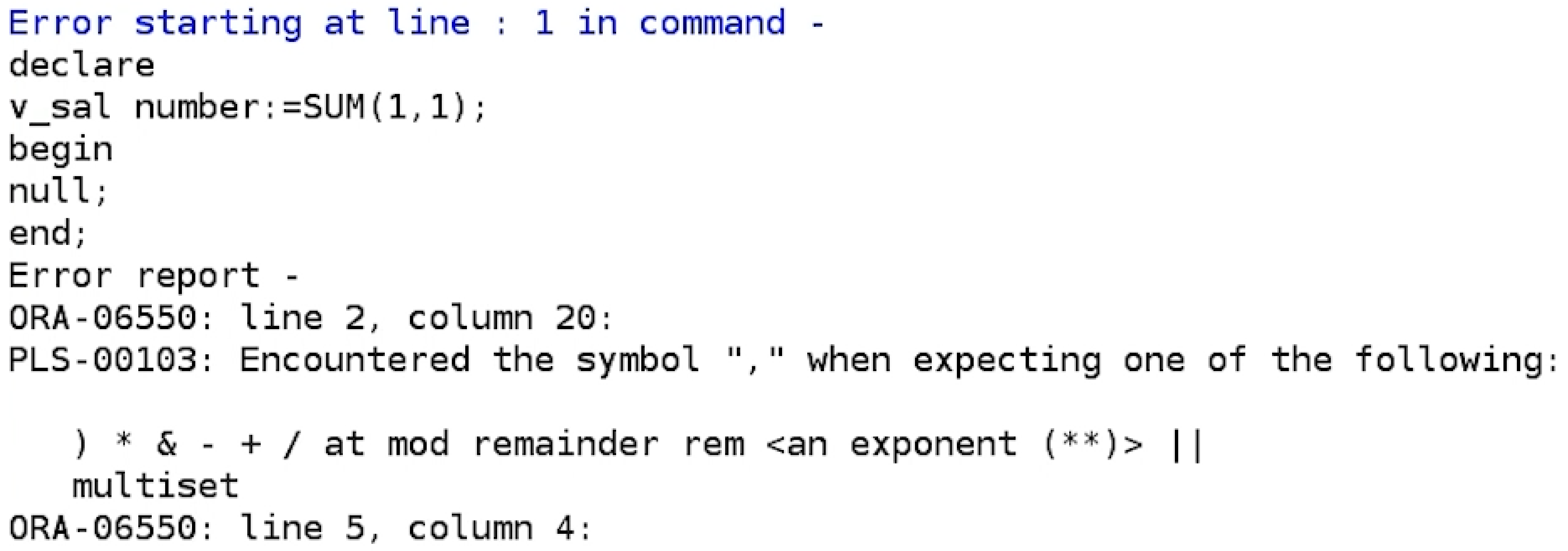
- Predefined functions that are used in SQL can also be used in PL/SQL.
- Functions that are available in procedural statements are:
  - Single-row functions
  - Built-in functions with Strings
  - Built-in functions with Numbers
  - Built-in functions with Dates
- Functions that are not available in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



Example : Not allowed group function in PL/SQL code

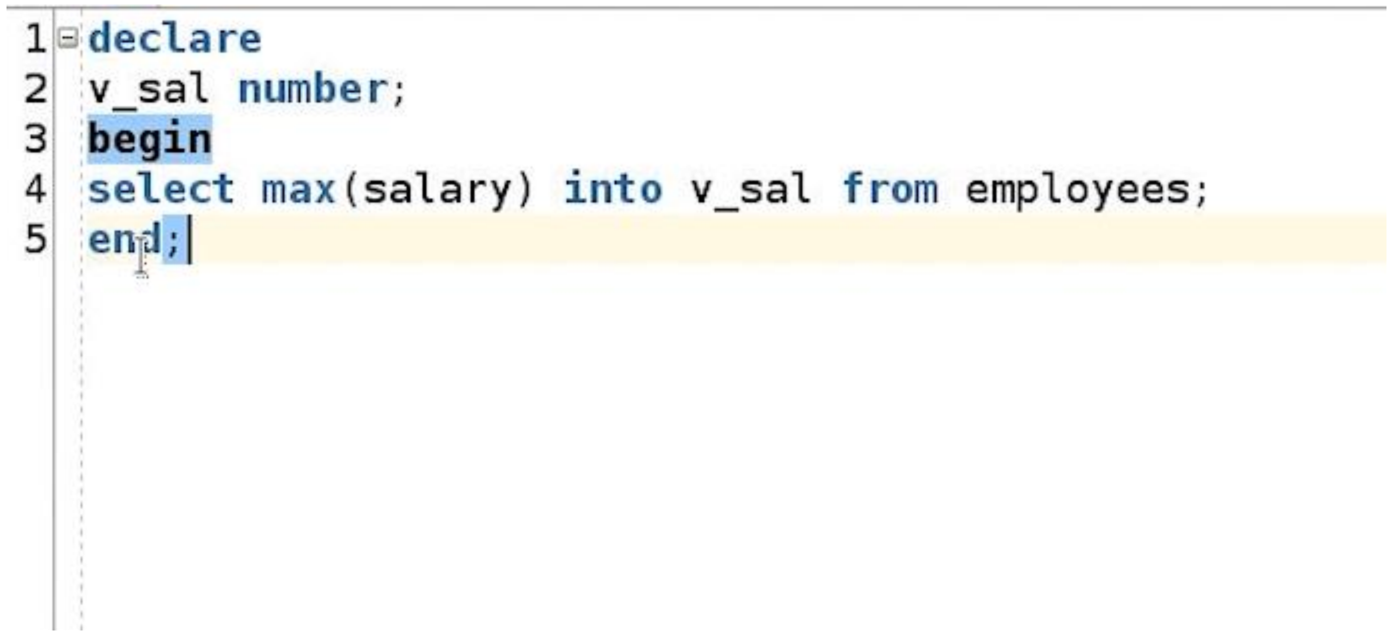


```
declare
v_sal number:=SUM(1,1);
begin
null;
end;
```

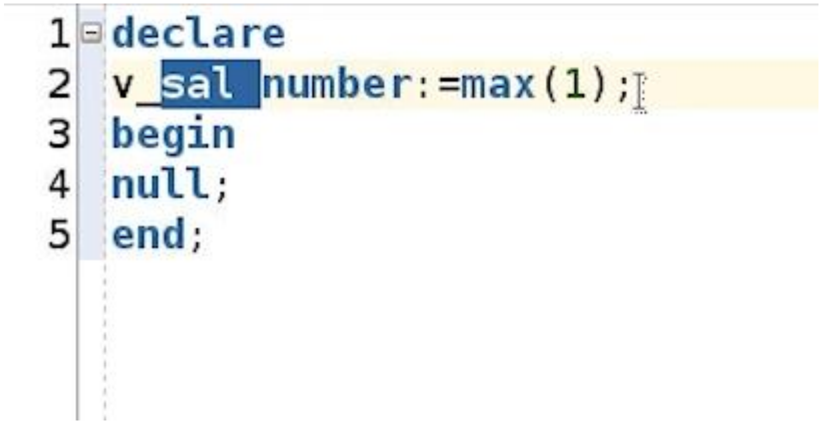


```
Error starting at line : 1 in command -
declare
v_sal number:=SUM(1,1);
begin
null;
end;
Error report -
ORA-06550: line 2, column 20:
PLS-00103: Encountered the symbol "," when expecting one of the following:

   ) * & - + / at mod remainder rem <an exponent (**)> ||
   multiset
ORA-06550: line 5, column 4:
```

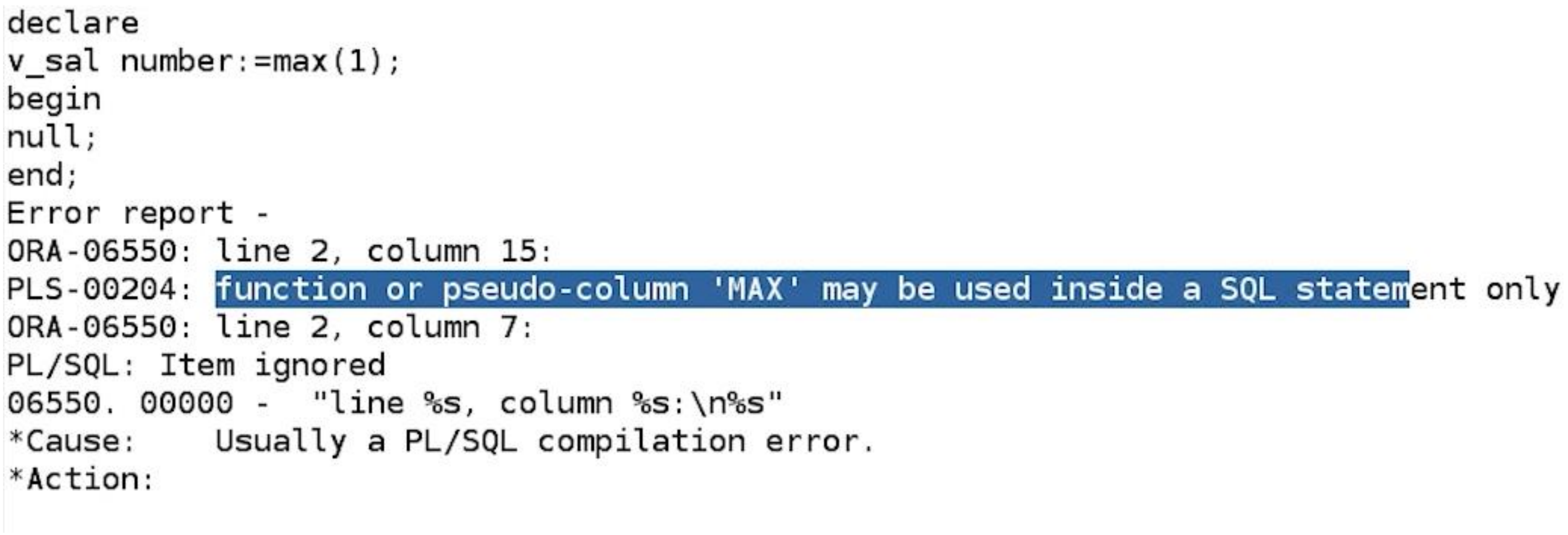
We can use group functions in SQL Code. Example :



```
declare
v_sal number;
begin
select max(salary) into v_sal from employees;
end;
```



```
declare
v_sal number:=max(1);
begin
null;
end;
```



```
declare
v_sal number:=max(1);
begin
null;
end;
Error report -
ORA-06550: line 2, column 15:
PLS-00204: function or pseudo-column 'MAX' may be used inside a SQL statement only
ORA-06550: line 2, column 7:
PL/SQL: Item ignored
06550. 00000 -  "line %s, column %s:\n%s"
*Cause:    Usually a PL/SQL compilation error.
*Action:
```

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);
```

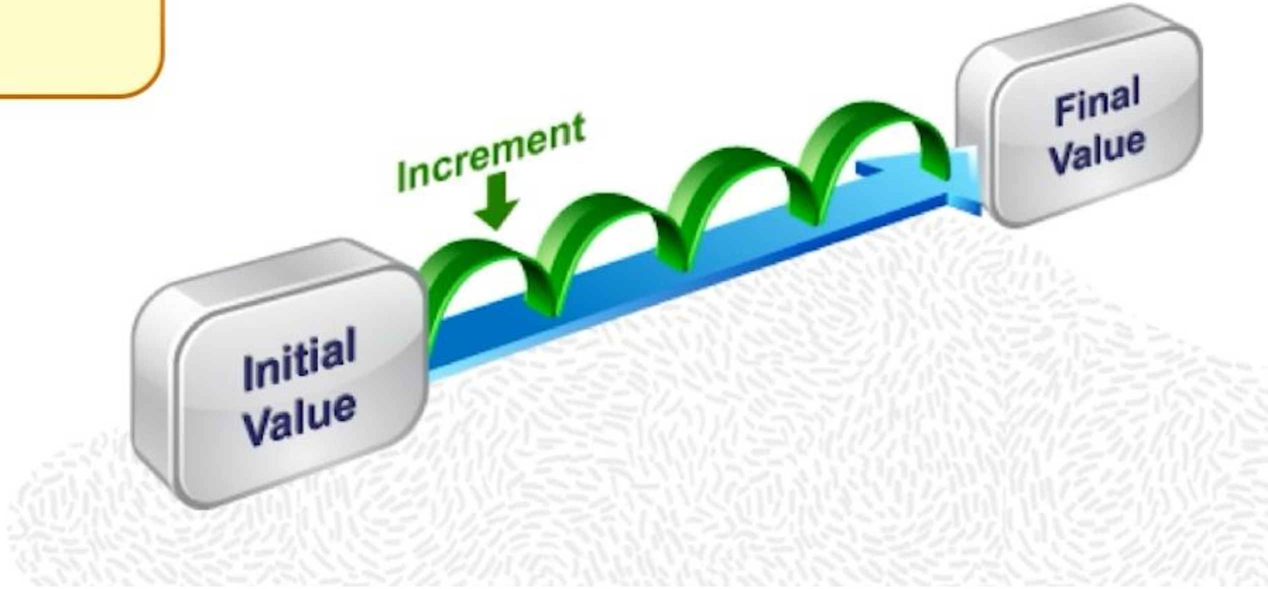
Using Sequences in PL/SQL blocks

What should be her employee_id ?
Should I search the database to find the last employee id ?
How accurate would that be, because I can't repeat the employee_id ?
NEW HIRE
MANAGER

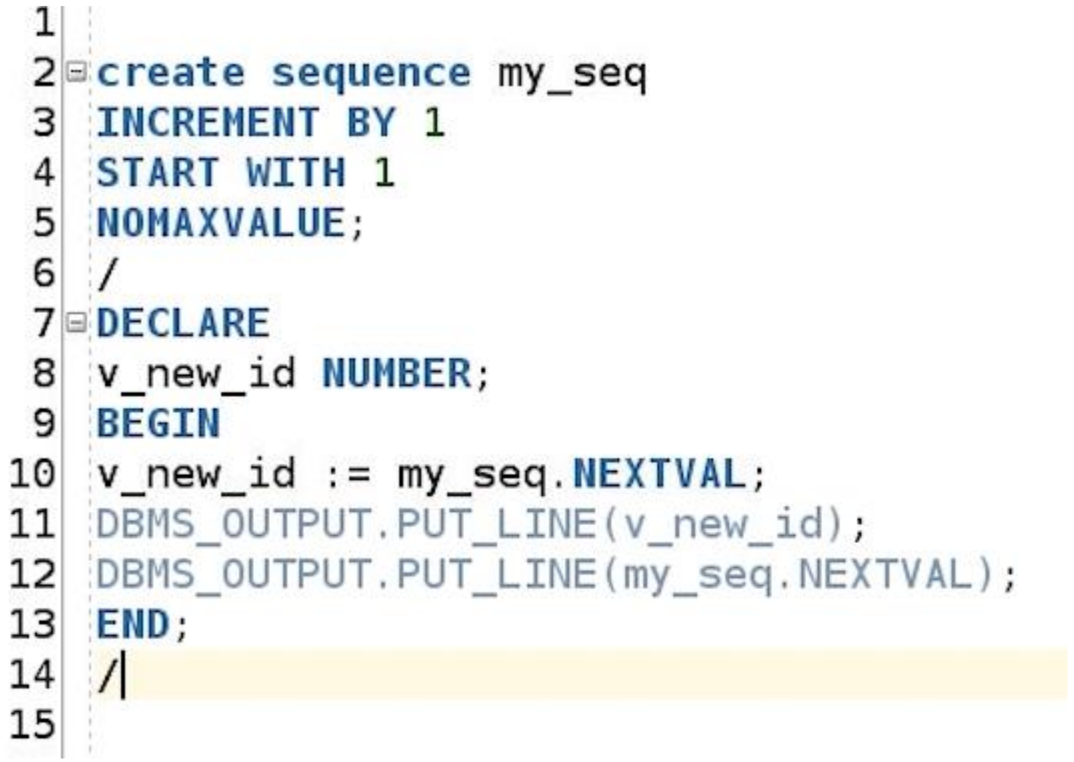
Using Sequences in PL/SQL blocks

- 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;
```



Example :

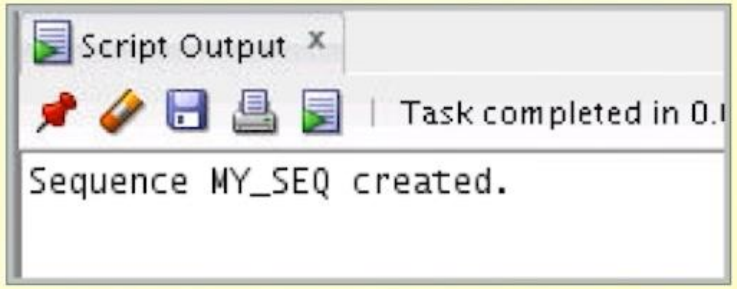


```

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;
/

```

```
1
2


PL/SQL procedure successfully completed.



PL/SQL procedure successfully completed.

3
4


PL/SQL procedure successfully completed.

5
6


PL/SQL procedure successfully completed.
```

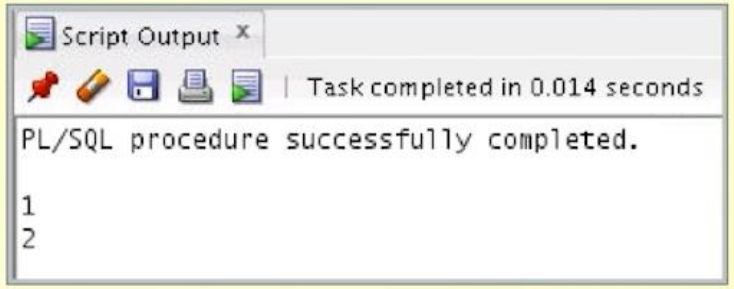
Using Sequences in PL/SQL Blocks

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



```
Sequence MY_SEQ created.
```

```
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;
```

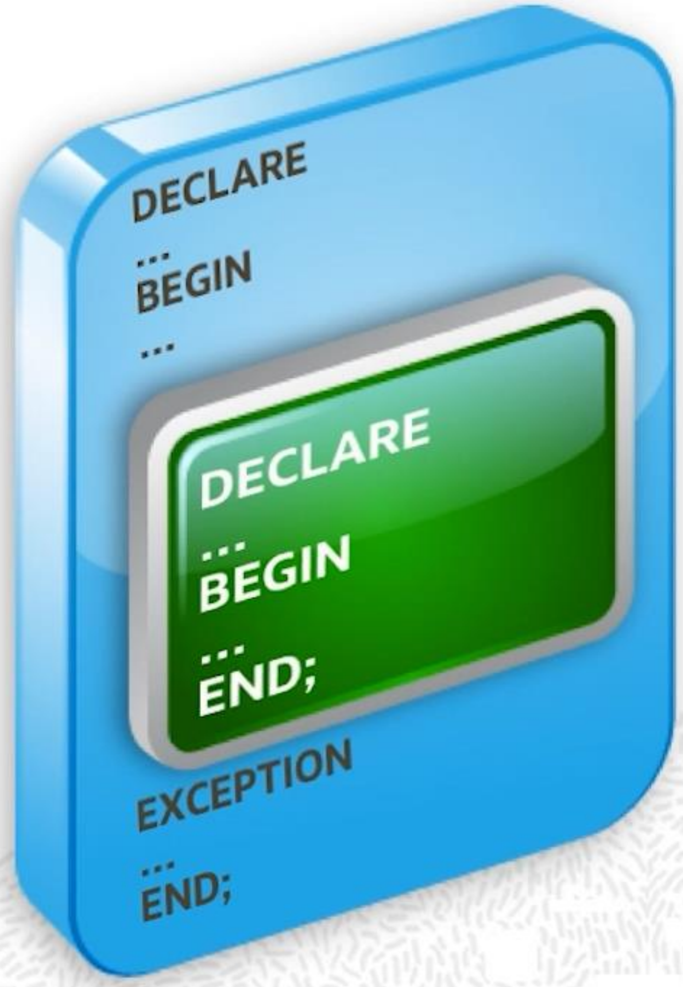


```
PL/SQL procedure successfully completed.

1
2
```

Nested blocks

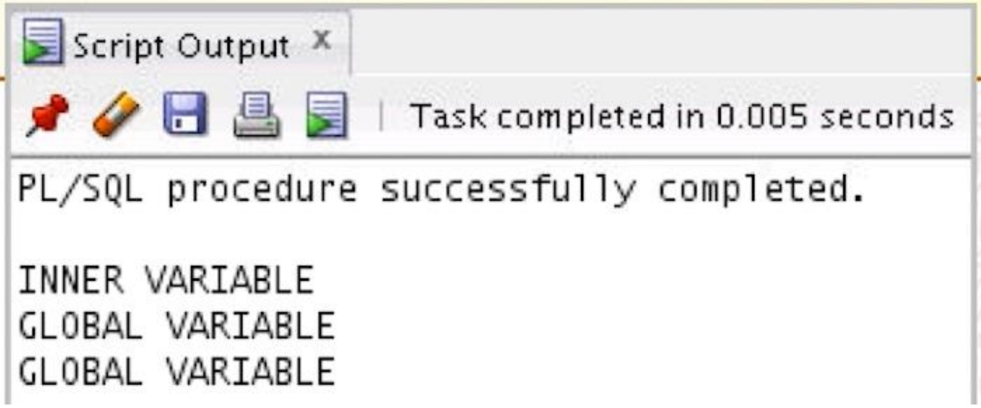
PL/SQL blocks can be nested.

- 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;
```



```
PL/SQL procedure successfully completed.

INNER VARIABLE
GLOBAL VARIABLE
GLOBAL VARIABLE
```

Variable Scope and Visibility

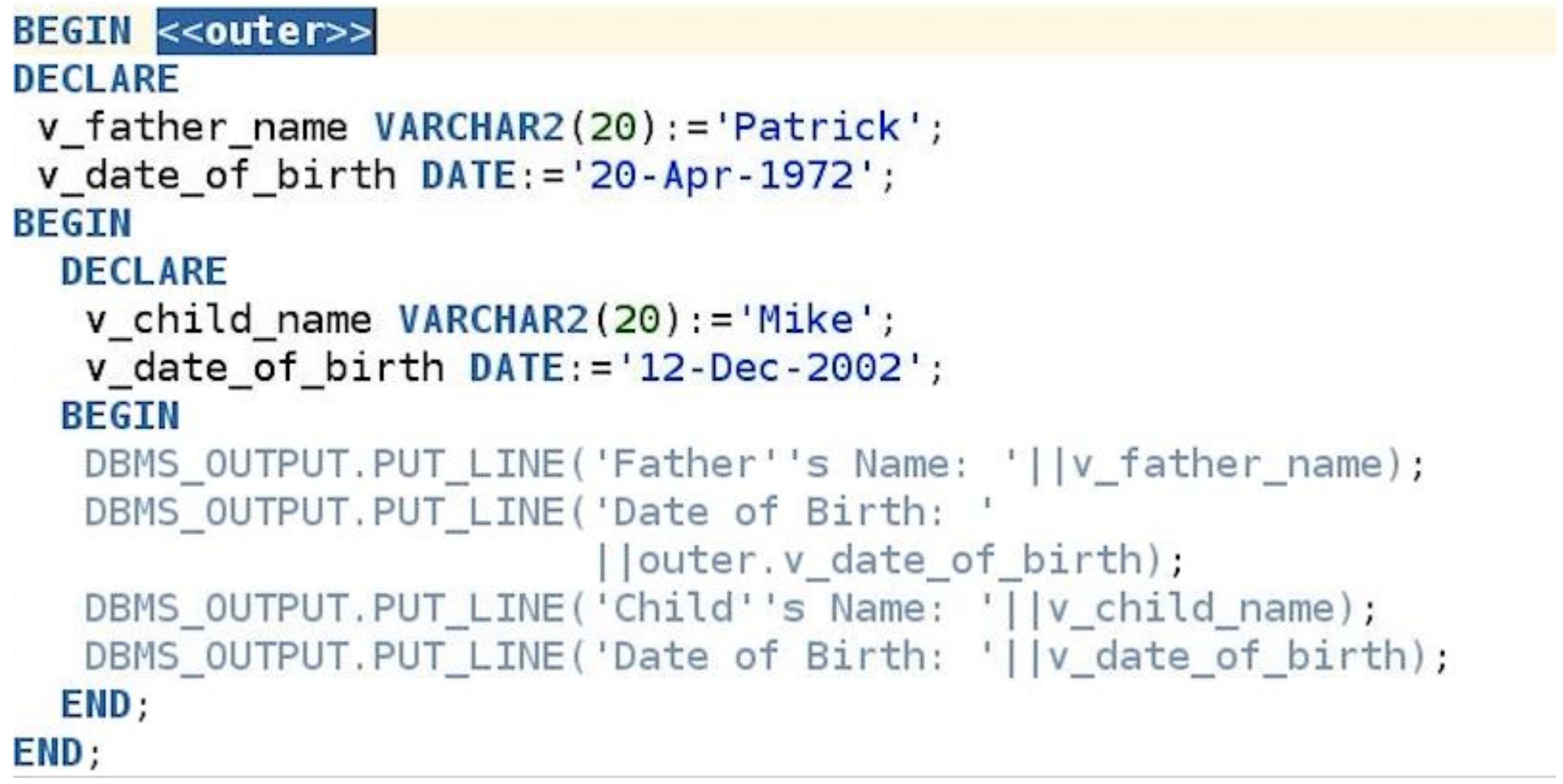
```
DECLARE
 v_father_name VARCHAR2(20):='Patrick';
 v_date_of_birth DATE:='20-Apr-1972';
BEGIN
  DECLARE
   v_child_name VARCHAR2(20):='Mike';
   v_date_of_birth DATE:='12-Dec-2002';
  BEGIN
   DBMS_OUTPUT.PUT_LINE('Father''s Name: '||v_father_name);
   DBMS_OUTPUT.PUT_LINE('Date of Birth: '||v_date_of_birth);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

```
BEGIN <<outer>>
DECLARE
  v_sal      NUMBER(7,2) := 60000;
  v_comm     NUMBER(7,2) := v_sal * 0.20;
  v_message  VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
     v_sal    NUMBER(7,2) := 50000;
     v_comm        NUMBER(7,2) := 0;
     v_total_comp  NUMBER(7,2) := v_sal + v_comm;
  BEGIN
1     v_message := 'CLERK not'||v_message;
     outer.v_comm := v_sal * 0.30;
  END;
2 v_message := 'SALESMAN'||v_message;
END;
END outer;
/
```



```
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;
```

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

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