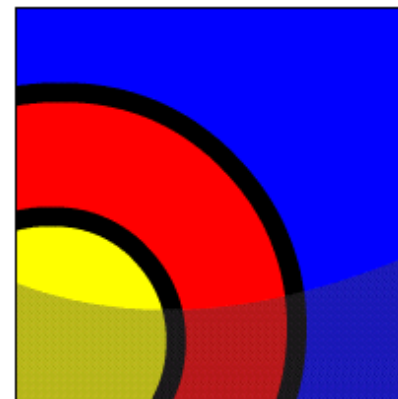


# Passing Parameters

## What Will I Learn?

In this lesson, you will learn to:

- List the types of parameter modes
- Create a procedure that passes parameters
- Identify three methods for passing parameters
- Describe the `DEFAULT` option for parameters





## Why Learn It?

To make procedures more flexible, it is important that varying data is either calculated or passed into a procedure by using input parameters.

Calculated results can be returned to the caller of a procedure by using `OUT` or `IN OUT` parameters.



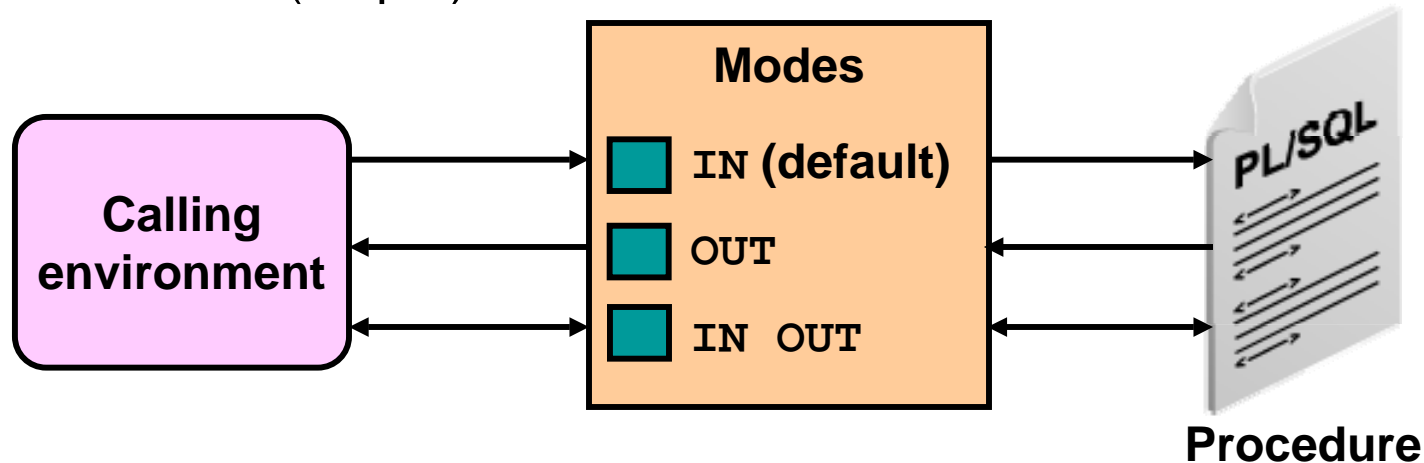
# Tell Me / Show Me

## Procedural Parameter Modes

Parameter modes are specified in the formal parameter declaration, after the parameter name and before its data type.

Parameter-passing modes:

- An **IN** parameter (the default) provides values for a subprogram to process.
- An **OUT** parameter returns a value to the caller.
- An **IN OUT** parameter supplies an input value, which can be returned (output) as a modified value.





# Tell Me / Show Me

The **IN** mode is the default if no mode is specified.

```
CREATE PROCEDURE procedure(param [mode] datatype)  
...
```

```
CREATE OR REPLACE PROCEDURE raise_salary  
  (p_id      IN my_employees.employee_id%TYPE,  
   p_percent IN NUMBER)  
IS  
BEGIN  
  UPDATE my_employees  
    SET    salary = salary * (1 + p_percent/100)  
    WHERE  employee_id = p_id;  
END raise_salary;
```

IN parameters can only be read within the procedure. They cannot be modified.



# Tell Me / Show Me

## Using OUT Parameters: Example

```
CREATE OR REPLACE PROCEDURE query_emp
(p_id      IN  employees.employee_id%TYPE,
p_name     OUT employees.last_name%TYPE,
p_salary   OUT employees.salary%TYPE) IS
BEGIN
  SELECT  last_name, salary INTO p_name, p_salary
  FROM    employees
  WHERE   employee_id = p_id;
END query_emp;
```

```
DECLARE
  a_emp_name employees.last_name%TYPE;
  a_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(178, a_emp_name, a_emp_sal); ...
END;
```



## Tell Me / Show Me

### Using the Previous OUT Example

Create a procedure with OUT parameters to retrieve information about an employee. The procedure accepts the value 178 for employee ID and retrieves the name and salary of the employee with ID 178 into the two OUT parameters. The `query_emp` procedure has three formal parameters. Two of them are OUT parameters that return values to the calling environment, shown in the code box at the bottom of the previous slide. The procedure accepts an employee ID value through the `p_id` parameter. The `a_emp_name` and `a_emp_sal` variables are populated with the information retrieved from the query into their two corresponding OUT parameters.

**Note:** Make sure that the data type for the actual parameter variables used to retrieve values from OUT parameters has a size large enough to hold the data values being returned.



## Tell Me / Show Me

### Viewing OUT Parameters in Application Express

Use PL/SQL variables that are displayed with calls to the DBMS\_OUTPUT.PUT\_LINE procedure.

```
DECLARE
  a_emp_name employees.last_name%TYPE;
  a_emp_sal  employees.salary%TYPE;
BEGIN
  query_emp(178, a_emp_name, a_emp_sal);
  DBMS_OUTPUT.PUT_LINE('Name: ' || a_emp_name);
  DBMS_OUTPUT.PUT_LINE('Salary: ' || a_emp_sal);
END;
```

Name: Grant

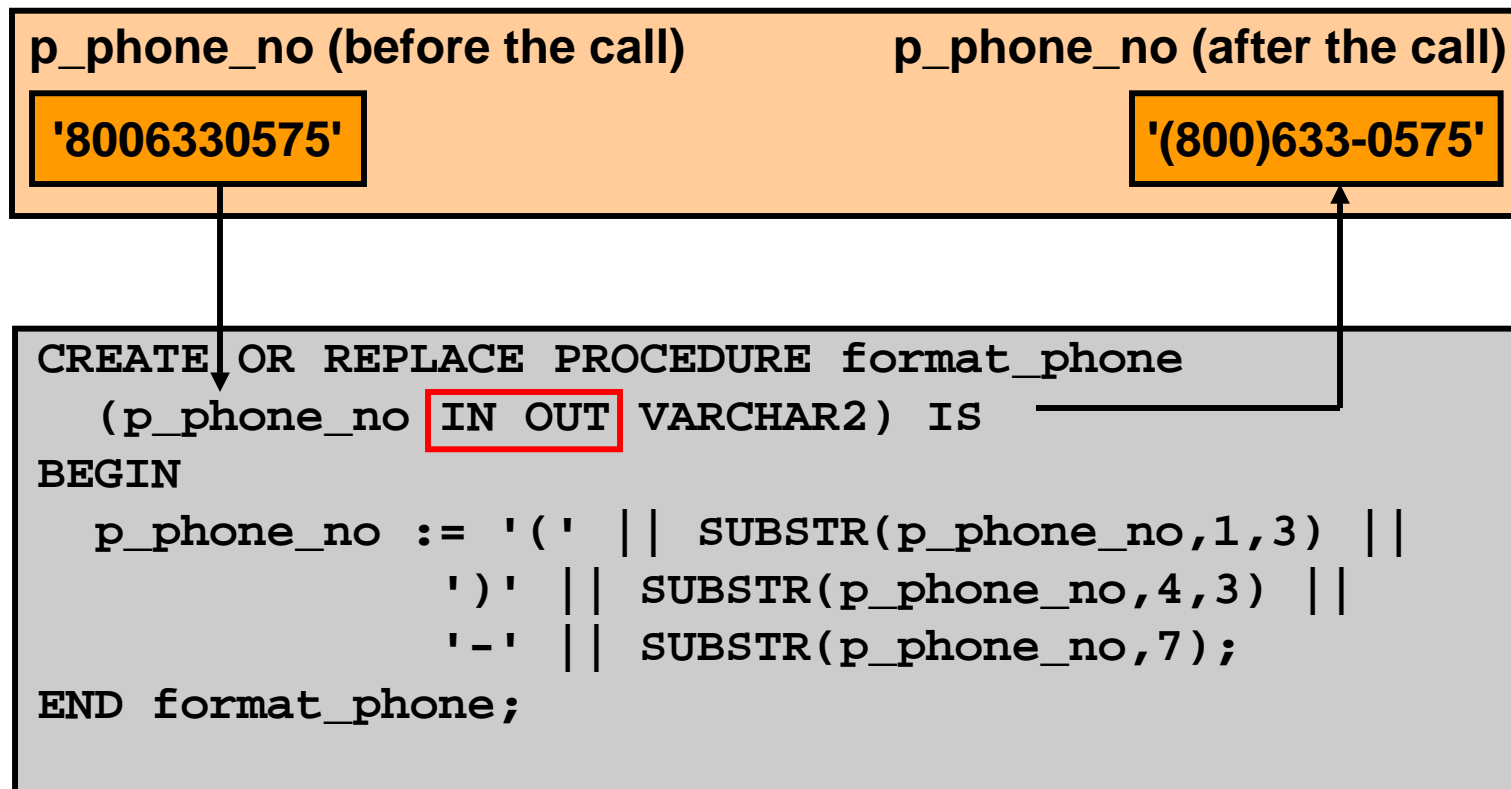
Salary: 7700



# Tell Me / Show Me

## Using IN OUT Parameters: Example

Calling environment





## Tell Me / Show Me

### Using the Previous IN OUT Example

Using an `IN OUT` parameter, you can pass a value into a procedure that can be updated within the procedure. The actual parameter value supplied from the calling environment can return as either of the following:

- The original unchanged value
- A new value that is set within the procedure

The example in the previous slide creates a procedure with an `IN OUT` parameter to accept a 10-character string containing digits for a phone number. The procedure returns the phone number formatted with parentheses around the first three characters and a hyphen after the sixth digit. For example, the phone string '8006330575' is returned as '(800)633-0575'.



## Tell Me / Show Me

### Calling the Previous IN OUT Example

The following code creates an anonymous block that declares `a_phone_no`, assigns the unformatted phone number to it, and passes it as an actual parameter to the `FORMAT_PHONE` procedure. The procedure is executed and returns an updated string in the `a_phone_no` variable, which is then displayed.

```
DECLARE
    a_phone_no VARCHAR2(13);
BEGIN
    a_phone_no := '8006330575' ;
    format_phone (a_phone_no);
    DBMS_OUTPUT.PUT_LINE('The formatted phone number is: '
                          || a_phone_no);
END;
```



# Tell Me / Show Me

## Summary of Parameter Modes

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value



# Tell Me / Show Me

## Syntax for Passing Parameters

There are three ways of passing parameters from the calling environment:

- **Positional:**
  - Lists the actual parameters in the same order as the formal parameters
- **Named:**
  - Lists the actual parameters in arbitrary order and uses the association operator ( ' $=>$ ' which is an equal and an arrow together) to associate a named formal parameter with its actual parameter
- **Combination:**
  - Lists some of the actual parameters as positional (no special operator) and some as named (with the  $=>$  operator).



# Tell Me / Show Me

## Parameter Passing: Examples

```
CREATE OR REPLACE PROCEDURE add_dept(  
    p_name IN my_depts.department_name%TYPE,  
    p_loc  IN my_depts.location_id%TYPE) IS  
BEGIN  
    INSERT INTO my_depts(department_id,  
                        department_name, location_id)  
        VALUES (departments_seq.NEXTVAL, p_name, p_loc);  
END add_dept;
```

- Passing by positional notation

```
add_dept ('EDUCATION', 1400);
```

- Passing by named notation

```
add_dept (p_loc=>1400, p_name=>'EDUCATION');
```

- Passing by combination notation

```
add_dept ('EDUCATION', p_loc=>1400);
```



# Tell Me / Show Me

## Parameter Passing

Will the following call execute successfully?

```
add_dept (p_loc => 1400, 'EDUCATION');
```

Answer: **No**, because when using the combination notation, positional notation parameters must be listed before named notation parameters.



# Tell Me / Show Me

## Parameter Passing

Will the following call execute successfully?

```
add_dept ( 'EDUCATION' );
```

```
ORA-06550: line 2, column 1:  
PLS-00306: wrong number or types of arguments in call to 'ADD_DEPT'  
ORA-06550: line 2, column 1:  
PL/SQL: Statement ignored  
1. begin  
2. add_dept( 'EDUCATION' );  
3. end;
```

**No:** You must provide a value for each parameter unless the formal parameter is assigned a default value. But what if you really want to omit an actual parameter, or you don't know a value for the parameter? Specifying default values for formal parameters is discussed next.





## Tell Me / Show Me

### Using the DEFAULT Option for IN Parameters

You can assign a default value for formal IN parameters. This provides flexibility when passing parameters.

```
CREATE OR REPLACE PROCEDURE add_dept(  
    p_name my_depts.department_name%TYPE := 'Unknown',  
    p_loc  my_depts.location_id%TYPE DEFAULT 1400)  
IS  
BEGIN  
    INSERT INTO my_depts (...)  
        VALUES (departments_seq.NEXTVAL, p_name, p_loc);  
END add_dept;
```

The code shows two ways of assigning a default value to an IN parameter. The two ways shown use:

- The assignment operator (:=), as shown for the `p_name` parameter
- The `DEFAULT` option, as shown for the `p_loc` parameter



## Tell Me / Show Me

### Using the DEFAULT Option for Parameters

The following are three ways of invoking the `add_dept` procedure:

- The first example assigns the default values for each parameter.
- The second example illustrates a combination of position and named notation to assign values. In this case, using named notation is presented as an example.
- The last example uses the default value for the `name` parameter and the supplied value for the `p_loc` parameter.

```
add_dept;  
add_dept ('ADVERTISING', p_loc => 1400);  
add_dept (p_loc => 1400);
```

## Tell Me / Show Me

### Guidelines for Using the `DEFAULT` Option for Parameters

- You cannot assign default values to `OUT` and `IN OUT` parameters *in the header*, but you can in the body of the procedure.
- Usually, you can use named notation to override the default values of formal parameters. However, you cannot skip providing an actual parameter if there is no default value provided for a formal parameter.
- A parameter inheriting a `DEFAULT` value is different from `NULL`.



## Tell Me / Show Me

### Working with Parameter Errors During Runtime

- **Note:** All the positional parameters should precede the named parameters in a subprogram call. Otherwise, you receive an error message, as shown in the following example:

```
BEGIN
  add_dept (name =>'new dept', 'new location');
END;
```

- The following error message is generated:

```
ORA-06550: line 2, column 33:
PLS-00312: a positional parameter association may not follow a named association
ORA-06550: line 2, column 6:
PL/SQL: Statement ignored
1. BEGIN
2.      add_dept(name=>'new dept', 'new location');
3. END;
```

## Tell Me / Show Me

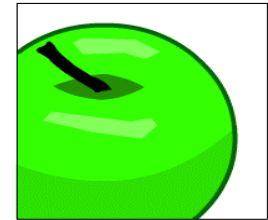
### Terminology

Key terms used in this lesson include:

IN parameter

OUT parameter

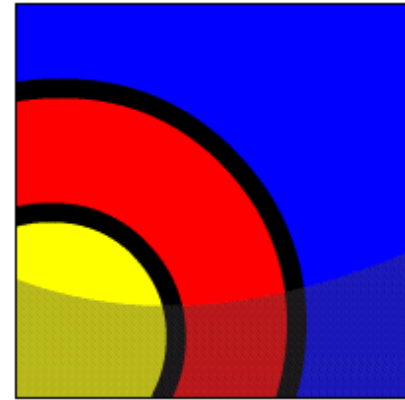
IN OUT parameter



## Summary

In this lesson, you learned to:

- List the types of parameter modes
- Create a procedure that passes parameters
- Identify three methods for passing parameters
- Describe the `DEFAULT` option for parameters





## Try It / Solve It

The exercises in this lesson cover the following topics:

- Listing the types of parameter modes
- Creating a procedure with parameters
- Identifying three methods for passing parameters
- Describing the `DEFAULT` option for parameters

