Function: Syntax

```
CREATE [OR REPLACE] FUNCTION function_name
 [(argument1 [mode1] datatype1,
  argument2 [mode2] datatype2,
  . . .)]
RETURN datatype
IS|AS
function_body;
```


Function: Syntax

The slide shows the syntax for creating a function. In the syntax:

function_name	Is the name of the function to be created
argument	Is the name given to the function parameter (Every argument is associated with a mode and data type. You can have any number of arguments separated by a comma. You pass the argument when you invoke the function.)
mode	Is the type of parameter (Only IN parameters should be declared.)
datatype	Is the data type of the associated parameter
RETURN datatype	Is the data type of the value returned by the function
function_body	Is the PL/SQL block that makes up the function code

The argument list is optional in the function declaration. The difference between a procedure and a function is that a function must return a value to the calling program. Therefore, the syntax contains return_type, which specifies the data type of the value that the function returns. A procedure may return a value via an OUT or INOUT parameter.

Creating a Function

```
CREATE FUNCTION check_sal RETURN Boolean IS
v_dept_id employees.department_id%TYPE;
 v_empno   employees.employee_id%TYPE;
 v_sal     employees.salary%TYPE;
 v_avg_sal employees.salary%TYPE;
BEGIN
 v_empno:=205;
 SELECT salary,department_id INTO v_sal,v_dept_id FROM
employees
 WHERE employee_id= v_empno;
 SELECT avg(salary) INTO v_avg_sal FROM employees WHERE
department_id=v_dept_id;
 IF v_sal > v_avg_sal THEN
  RETURN TRUE;
 ELSE
  RETURN FALSE;
 END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   RETURN NULL;
END;
```

ORACLE

Function: Example

The check_sal function is written to determine whether the salary of a particular employee is greater than or less than the average salary of all employees working in the same department. The function returns TRUE if the salary of the employee is greater than the average salary of the employees in the department; if not, it returns FALSE. The function returns NULL if a NO_DATA_FOUND exception is thrown.

Note that the function checks for the employee with the employee ID 205. The function is hard-coded to check only for this employee ID. If you want to check for any other employees, you must modify the function itself. You can solve this problem by declaring the function such that it accepts an argument.

You can then pass the employee ID as parameter.

Invoking a Function

```
BEGIN
 IF (check_sal IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
END;
/
```

Results | Script Output

```
anonymous block completed
Salary > average
```


Invoking the Function

You include the call to the function in the executable section of the anonymous block.

The function is invoked as a part of a statement. Remember that the check_sal function returns Boolean or NULL. Thus the call to the function is included as the conditional expression for the IF block. Note: You can use the DESCRIBE command to check the arguments and return type of the function, as in the following example: DESCRIBE check_sal;

Passing a Parameter to the Function

```
DROP FUNCTION check_sal;
CREATE FUNCTION check_sal(p_empno employees.employee_id%TYPE)
RETURN Boolean IS
 v_dept_id employees.department_id%TYPE;
 v_sal     employees.salary%TYPE;
 v_avg_sal employees.salary%TYPE;
BEGIN
 SELECT salary,department_id INTO v_sal,v_dept_id FROM employees
   WHERE employee_id=p_empno;
 SELECT avg(salary) INTO v_avg_sal FROM employees
   WHERE department_id=v_dept_id;
 IF v_sal > v_avg_sal THEN
  RETURN TRUE;
 ELSE
  RETURN FALSE;
 END IF;
EXCEPTION
  ...
```

ORACLE

Passing a Parameter to the Function

Remember that the function was hard-coded to check the salary of the employee with employee ID 205. The code shown in the slide removes that constraint because it is rewritten to accept the employee number as a parameter. You can now pass different employee numbers and check for the employee's salary.

You learn more about functions in the course titled Oracle Database: Develop PL/SQL Program Units.

The output of the code example in the slide is as follows:

Results | Script Output | Explain | A

```
DROP FUNCTION check_sal succeeded.
FUNCTION check_sal Compiled.
```

Invoking the Function with a Parameter

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 205');
 IF (check_sal(205) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal(205)) THEN
 DBMS_OUTPUT.PUT_LINE('Salary > average');
 ELSE
 DBMS_OUTPUT.PUT_LINE('Salary < average');
 END IF;
DBMS_OUTPUT.PUT_LINE('Checking for employee with id 70');
 IF (check_sal(70) IS NULL) THEN
 DBMS_OUTPUT.PUT_LINE('The function returned
  NULL due to exception');
 ELSIF (check_sal(70)) THEN
 ...
 END IF;
END;
/
```

ORACLE

Invoking the Function with a Parameter

The code in the slide invokes the function twice by passing parameters. The output of the code is as follows:

Results | Script Output | Explain | Autotrace

```
anonymous block completed
Checking for employee with id 205
Salary > average
Checking for employee with id 70
The function returned NULL due to exception
```