**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

* + **Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

//example use of the function CalculateAge

SELECT CalculateAge(TO\_DATE('1988-05-20', 'YYYY-MM-DD')) AS Age FROM dual;

2.-- Ensure DBMS output is enabled

SET SERVEROUTPUT ON;

//function declaration

CREATE OR REPLACE FUNCTION CalculateAge (

p\_dob IN DATE

) RETURN NUMBER IS

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

RETURN v\_age;

END;

/

//the block to use the function

DECLARE

v\_dob DATE := TO\_DATE('1961-10-24', 'YYYY-MM-DD'); -- You can change this DOB

v\_age NUMBER;

BEGIN

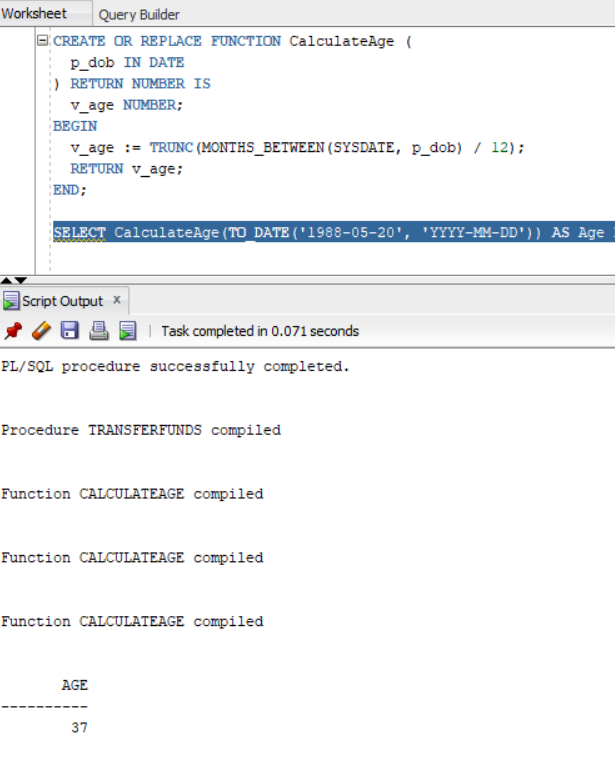
v\_age := CalculateAge(v\_dob);

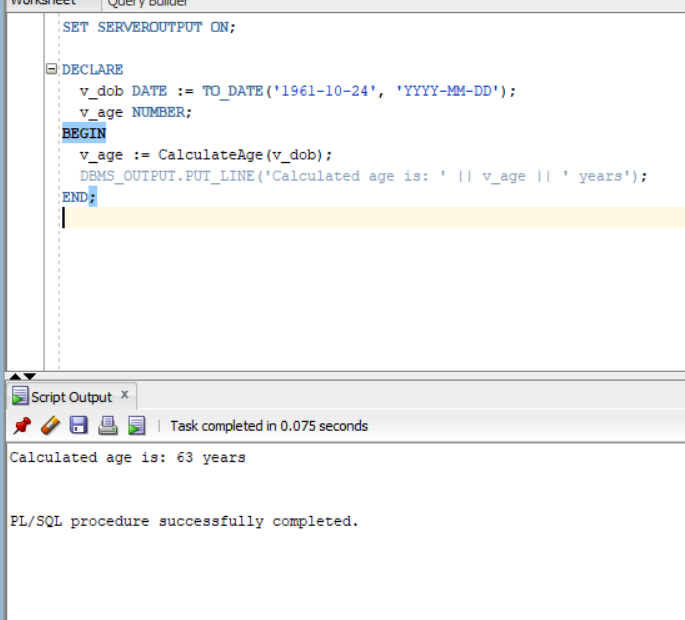
DBMS\_OUTPUT.PUT\_LINE('Calculated age is: ' || v\_age || ' years');

END;

/

OUTPUTS:





**Scenario 2:** The bank needs to compute the monthly installment for a loan.

* + **Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

p\_loan\_amount IN NUMBER,

p\_annual\_rate IN NUMBER, -- e.g., 6.5 for 6.5%

p\_duration\_years IN NUMBER -- loan duration in years

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_total\_months NUMBER;

v\_emi NUMBER;

BEGIN

v\_monthly\_rate := p\_annual\_rate / 12 / 100;

v\_total\_months := p\_duration\_years \* 12;

IF v\_monthly\_rate = 0 THEN

v\_emi := p\_loan\_amount / v\_total\_months;

ELSE

v\_emi := p\_loan\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_total\_months)

/ (POWER(1 + v\_monthly\_rate, v\_total\_months) - 1);

END IF;

RETURN ROUND(v\_emi, 2); -- Rounded to 2 decimal places

END;

/

SET SERVEROUTPUT ON;

DECLARE

v\_loan\_amount NUMBER := 500000;

v\_interest\_rate NUMBER := 7.5;

v\_years NUMBER := 10;

v\_monthly\_emi NUMBER;

BEGIN

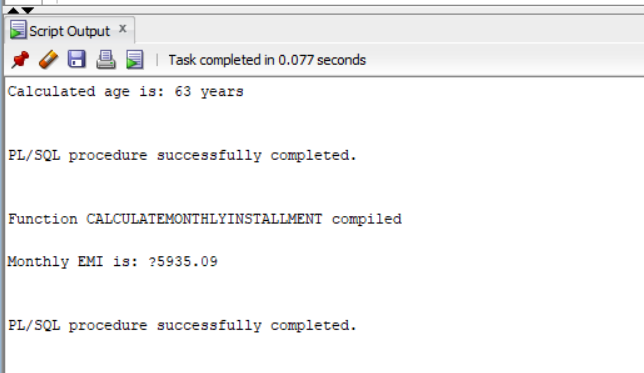
v\_monthly\_emi := CalculateMonthlyInstallment(v\_loan\_amount, v\_interest\_rate, v\_years);

DBMS\_OUTPUT.PUT\_LINE('Monthly EMI is: ₹' || v\_monthly\_emi);

END;

/

OUTPUT:



**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

* + **Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

p\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) RETURN BOOLEAN IS

v\_balance Accounts.Balance%TYPE;

BEGIN

SELECT Balance

INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE; -- Account does not exist

END;

/

SET SERVEROUTPUT ON;

DECLARE

v\_result BOOLEAN;

BEGIN

v\_result := HasSufficientBalance(3, 2000);

IF v\_result THEN

DBMS\_OUTPUT.PUT\_LINE('✅ Sufficient balance available.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('❌ Insufficient balance or invalid account.');

END IF;

END;

/

OUPUT:

