create or replace procedure mytest(
    num in number, num2 out number,
    num3 in out number)
as
begin
    -- num := 21; /* This would cause an error message. */
    num2 := 55;
    num3 := 44;
end;
/

-- This is how to call that procedure.
declare
    y number;
    z number;
    x number;
begin
    y := 888;
    z := 999;
    x := 777;
    dbms_output.put_line('Before call ' || y || ' ' || z || ' ' || x);
    mytest(y, z, x);  -- z and x are changed!!
    dbms_output.put_line('After call ' || y || ' ' || z || ' ' || x);
end;
/
create or replace procedure mytest(
    num in number, num2 out number,
    num3 in out number)
as
begin
    -- num := 21;
    num2 := 55;
    num3 := 44;
end;
/

-- This is how to call that procedure.
declare
    z number;
    x number;
begin
    z := 999;
    x := 777;
    mytest(888, z, x);  -- IN parameter called with a number;
    mytest(888, 22, x);  -- Would cause an error!
    mytest(888, z, 22);  -- Out parameters have to be variables.
    -- In Out parameters have to be variables.
    dbms_output.put_line( z || ' ' || x);
end;
/
-- Two more data types
-- Booleans cannot be output with put_line()!

declare
  b boolean;
  c number(10,3);
begin
  b := false;
--  b := 0;  This would give an error message.
  if b then
    dbms_output.put_line(2);
  end if;
  c := 1234567.891;
  dbms_output.put_line(c);
end;
/

-- 10 - 3 = 7 is the maximum number of positions
-- allowed BEFORE the decimal point.
-- 3 is the number of digits after the decimal point.
-- A recursive function
create or replace function factorial(
    num in number)
return number
as
begin
    if num = 0
    then return 1;
    else return num * factorial(num - 1);
    end if;
end;
/

-- This is how to call that function.
declare
    z number;
begin
    z := factorial(3);
    dbms_output.put_line(z);
end;
/
-- A useful procedure.
-- For lazy people. Like me.

create or replace procedure pr(
   str in varchar2)
as
begin
   dbms_output.put_line(str);
end;
/

begin
   pr('Listening to you I get the music');
   pr(1234);
end;
/
-- The "loop - end loop" loop.
-- An "almost endless" loop
-- You need to jump out with exit.

create or replace procedure vector
as
  i number;
begin
  i := 0;
  loop  -- potentially an endless loop
    dbms_output.put_line(i);
    i := i + 1;
    exit when i = 10;  -- This makes it "not endless"
  end loop;
end;
/

begin
  vector;
end;
/

-- This is a for loop, finally.
-- The most important tool of all programmers.

create or replace procedure matrix
as
  i number;
begin
  for i in 1 .. 10
  loop
    dbms_output.put_line(i);
  end loop;

end;
/

begin
  matrix;
end;
/

i number;
begin
  for i in 10 .. 15  -- works
    loop
    dbms_output.put_line(i);
    end loop;
  for i in -5 .. 5  -- works
    loop
    dbms_output.put_line(i);
    end loop;
  for i in 20 .. 20  -- works, done once
    loop
    dbms_output.put_line(i);
    end loop;
  for i in 5 .. 4   -- Nothing happens
    loop
    dbms_output.put_line(i);
    end loop;
  for i in reverse 9 .. 6  -- Nothing happens
    loop
    dbms_output.put_line(i);
    end loop;
  for i in reverse 6 .. 9  -- works, counts down
    loop
    dbms_output.put_line(i);
    end loop;
end;
-- More loops
-- This is a nested for loop.

create or replace procedure testnestedloop
as
    i number;
    j number;
    line varchar2(80);
begin
    for i in 1 .. 10
        loop
            line := i || '::';
            for j in 20 .. 30
                loop
                    line := line || ' ' || j;
                end loop;
            end loop;
            dbms_output.put_line(line);
        end loop;
end;
/

begin
    testnestedloop;
end;
/

-- Print a triangle matrix.

create or replace procedure matrix
as
  i number;
  j number;
  line varchar2(80);
begin
  for i in 1 .. 10  -- Count rows
    loop
    line := ";
    for j in 1 .. i+1  -- Count columns in row
      loop
        line := line || ' ' || j;
      end loop;
    dbms_output.put_line(line);
  end loop;
end;

begin
  matrix;
end;
/

create table x (  -- create or replace does not work
   num  number,
   sqr  number,
   cube number
)
/
/
insert into x values (1, 1, 1)
/
insert into x values (2, 4, 8)
/
insert into x values (3, 9, 27)
/
insert into x values (4, 16, 64)
/
We need to start with a philosophical question:

\[ A = A + B; \]  (long form)
\[ A = A + 1; \]  (long form)

means the same thing like

\[ A += B; \]  (short form)
\[ A++ \]  (short form; note the "1" is invisible here)
\[ ++A \]  THIS IS JAVA. This is NOT PL/SQL !!!!!!!!

So... what should we teach?
Only the long form? But why make people work harder?
Only the short form? But will people understand?
--- Teach both.
Which one first?
Obviously the long form is easier to understand, so we start with that.

We face the same problem with table access from PL/SQL.
There is a long form and a short form.
There are even intermediate forms. :
In the short form something really important is invisible.
-- I advance to processing select statements.
-- With a CURSOR.
-- Note that this example only processes
-- a single line.

declare

    anumber number; -- Three variables.
asquare number;
acube number := 0;

    cursor  thomas  is select * from x;
    -- ^^^  ^^^  ^^^^^^^^^^^^^^^
    -- a "type" a "name" an "initializer"

-- A cursor is like a pointer to the
-- first row in a table. Later it moves
-- down.
begin

    open thomas; -- like opening a file
    fetch thomas into -- load one row
        anumber, asquare, acube; -- into these variables;
        -- Then move down one row;
    close thomas; -- like "unloading" the table

    dbms_output.put_line(anumber || ' ZZZ ' || asquare || ' ZZZ ' || acube);
end;
/