

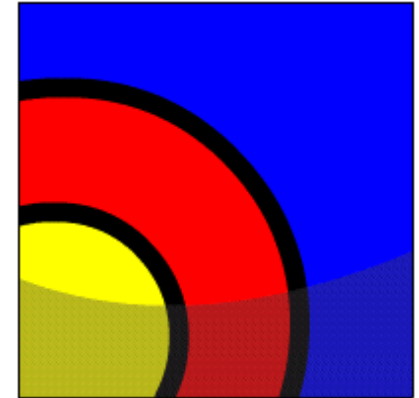
Recognizing the Scope of Variables



What Will I Learn?

In this lesson, you will learn to:

- Describe the rules for variable scope when a variable is nested in a block
- Recognize a variable-scope issue when a variable is used in nested blocks
- Qualify a variable nested in a block with a label
- Describe the scope of an exception
- Recognize an exception-scope issue when an exception is within nested blocks
- Describe the effect of exception propagation in nested blocks





Why Learn It?

You learned some information about nested blocks, scope of variables, and exception propagation in an earlier lesson. Now that you understand exception handling properly, in this lesson you first review what you know already, and then learn more.

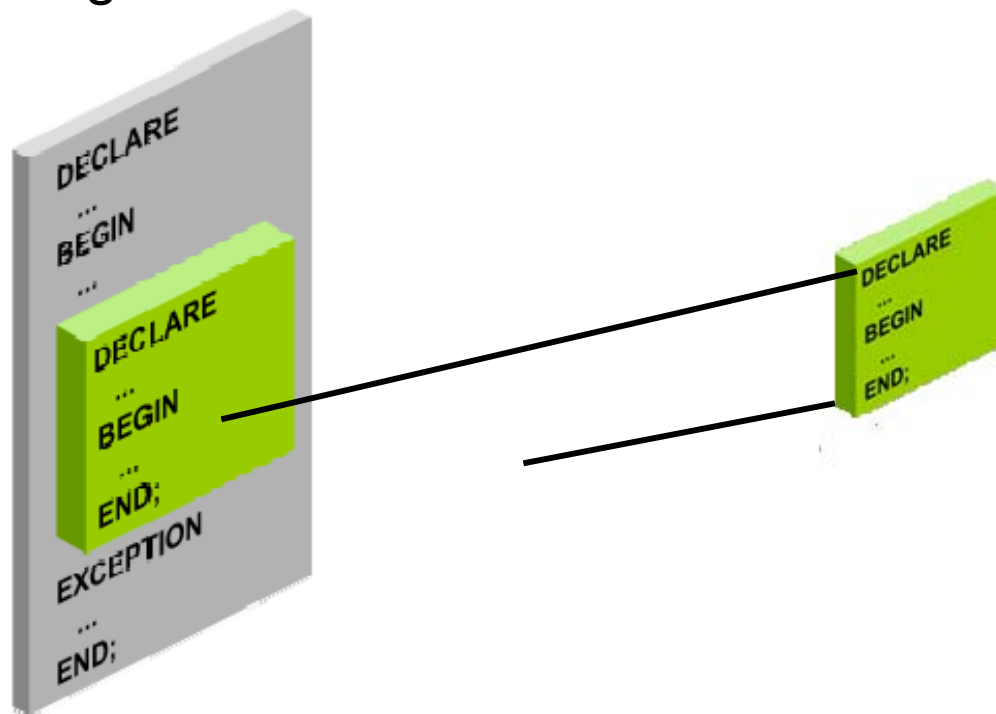
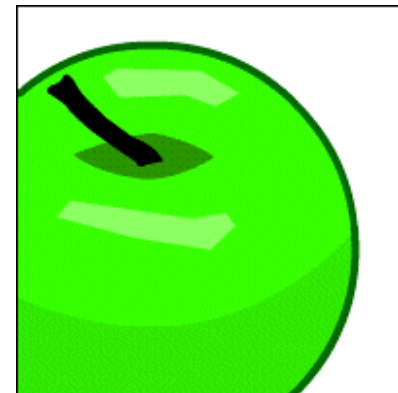


A named exception is a kind of PL/SQL variable. To handle exceptions correctly, you must understand the scope and visibility of exception variables. This is particularly important when using nested blocks.

Tell Me / Show Me

Review of Nested Blocks

Nested blocks are blocks of code inside blocks of code. There is an outer block and an inner block. You can nest blocks within blocks as many times as you need to, there is no practical limit to the depth of nesting Oracle allows.





Tell Me / Show Me

Review of Nested Blocks

The example shown in the slide has an outer (parent) block (illustrated in blue) and a nested (child) block (illustrated in red). The variable `v_outer_variable` is declared in the outer block and the variable `v_inner_variable` is declared in the inner block.

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL 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;
```



Tell Me / Show Me

Review of Variable Scope

The scope of a variable is the block or blocks in which the variable is accessible, that is, it can be named and used. In PL/SQL, a variable's scope is the block in which it is declared plus all blocks nested within the declaring block.

What are the scopes of the two variables declared in this example?

```
DECLARE
  v_outer_variable VARCHAR2(20):='GLOBAL VARIABLE';
BEGIN
  DECLARE
    v_inner_variable VARCHAR2(20):='LOCAL 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;
```



Tell Me / Show Me

Review of Variable Scope (continued)

Examine the following code. What is the scope of each of the variables?

```
DECLARE
  v_father_name    VARCHAR2(20):='Patrick';
  v_date_of_birth  DATE:='20-Apr-1972';
BEGIN
  DECLARE
    v_child_name    VARCHAR2(20):='Mike';
  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;
```



Tell Me / Show Me

Review of Variable Scope (continued)

Why will this code not work correctly?

```
BEGIN
  DECLARE
    CURSOR emp_curs IS SELECT * FROM employees;
    v_emp_rec  emp_curs%ROWTYPE;
  BEGIN
    OPEN emp_curs;
    LOOP
      FETCH emp_curs INTO v_emp_rec;
      EXIT WHEN emp_curs%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_emp_rec.first_name);
    END LOOP;
  END;
  CLOSE emp_curs;
END;
```




Tell Me / Show Me

Review of Variable Scope

Will this code work correctly? Why or why not?

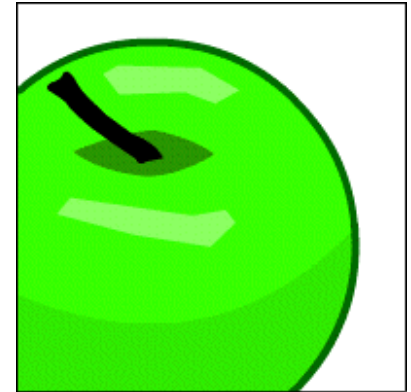
```
DECLARE
  CURSOR emp_curs IS SELECT * FROM employees;
BEGIN
  OPEN emp_curs;
  DECLARE
    v_emp_rec    emp_curs%ROWTYPE;
  BEGIN
    LOOP
      FETCH emp_curs INTO v_emp_rec;
      EXIT WHEN emp_curs%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_emp_rec.first_name);
    END LOOP;
  END;
  CLOSE emp_curs;
END;
```

Tell Me / Show Me

How does PL/SQL Resolve the Names of Variables?

When you reference the name of a variable within a block, PL/SQL first looks to see if a variable with that name has been declared within that block (a local variable). If it cannot find it, PL/SQL then looks at the enclosing block. If it cannot find it declared there either, PL/SQL looks at the next level of enclosing block (remember there can be three or more levels of nested blocks). And so on.

The next slide shows three levels of nested block.





Tell Me / Show Me

Three Levels of Nested Block

What is the scope of each of these variables?

```
DECLARE                                -- outer block
  v_outervar VARCHAR2(20);
BEGIN
  DECLARE                              -- middle-level block
    v_middlevar VARCHAR2(20);
  BEGIN                                -- innermost block
    v_outervar := 'Joachim';
    v_middlevar := 'Chang';
  END;
END;
END;
```



Tell Me / Show Me

Review of Variable Naming

Are the following declarations valid?

```
DECLARE                                -- outer block
  v_myvar VARCHAR2(20);
BEGIN
  DECLARE                                -- inner block
    v_myvar  VARCHAR2(15);
  BEGIN
    ...
  END;
END;
```



Tell Me / Show Me

Review of Variable Naming (continued)

In this example, the variable `v_date_of_birth` is declared twice.

```
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';
    DBMS_OUTPUT.PUT_LINE('Date of Birth: ' || v_date_of_birth);
    ...
```

Which `v_date_of_birth` is referenced in the `DBMS_OUTPUT.PUT_LINE` statement?

Tell Me / Show Me

Review of Variable Visibility

The visibility of a variable is the portion of the program where an in-scope variable can be accessed without using a qualifier. What is the visibility of each of the variables?

```

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;

```



Tell Me / Show Me

Review of Variable Visibility (continued)

The `v_date_of_birth` variable declared in the outer block has scope even in the inner block. This variable is visible in the outer block. However, it is not visible in the inner block because the inner block has a local variable with the same name. The `v_father_name` variable is visible in the inner and outer blocks. The `v_child_name` variable is visible only in the inner block.

```
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';
  ...
```

What if you want to reference the outer block's `v_date_of_birth` within the inner block?



Tell Me / Show Me

Review of Block Labels

We can label a block using the <<...>> syntax shown here. You can use this label to access the variables that have scope but are not visible. In this example, the outer block has the label, <<outer>>.

```
<<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';
  ...
```

Labeling is not limited to the outer block; you can label any block.



Tell Me / Show Me

Using Block Labels to Gain Variable Visibility

Using the `outer` label to qualify the `v_date_of_birth` identifier, you can now display the father's date of birth in the inner block.

```
<<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;
```

Father's Name: Patrick

Date of Birth: 20-APR-72

Child's Name: Mike

Date of Birth: 12-DEC-02

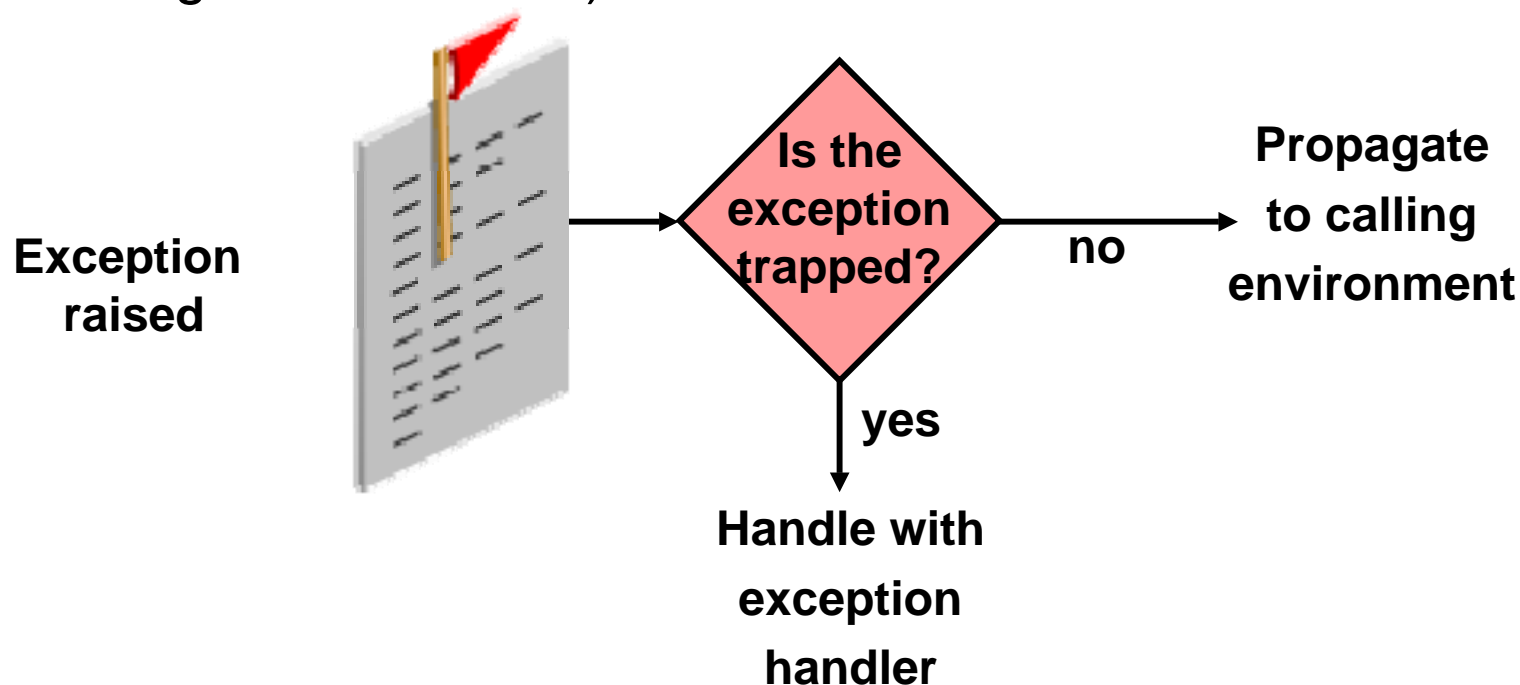
Statement processed.

Tell Me / Show Me

Exception Handling in Nested Blocks

You can deal with an exception by:

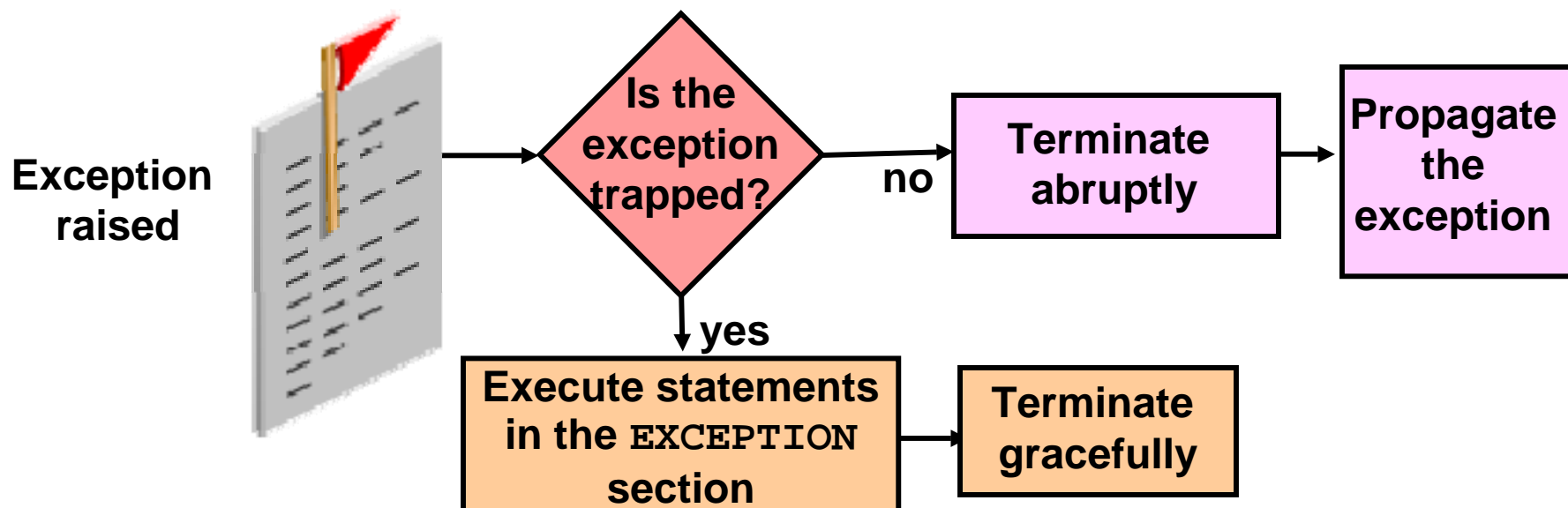
- Handling it (“trapping it”) in the block in which it occurs, or
- Propagating it to the calling environment (which can be a higher-level block)



Tell Me / Show Me

Propagating Exceptions to an Outer Block

If the exception is raised in the executable section of the inner block and there is no corresponding exception handler, the PL/SQL block terminates with failure and the exception is propagated to an enclosing block.





Tell Me / Show Me

Propagating Exceptions to an Outer Block (continued)

In this example, an exception occurs during the execution of the inner block. The inner block's `EXCEPTION` section does not deal with the exception. The inner block terminates unsuccessfully and PL/SQL passes (propagates) the exception to the outer block. The outer block's `EXCEPTION` section successfully handles the exception.

```
DECLARE      -- outer block
  e_no_rows  EXCEPTION;
BEGIN
  BEGIN      -- inner block
    IF ... THEN RAISE e_no_rows; -- exception occurs here
    ...
  END;      -- Inner block terminates unsuccessfully
  ...      -- Remaining code in outer block's executable
  ...      -- section is skipped
EXCEPTION
  WHEN e_no_rows THEN - outer block handles the exception
  ...
END;
```

Tell Me / Show Me

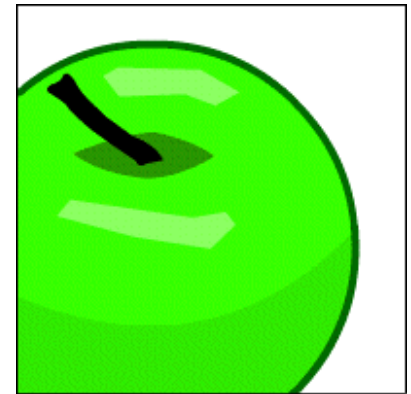
Propagating Exceptions from a Sub-Block

If a PL/SQL raises an exception and the current block does not have a handler for that exception, the exception propagates to successive enclosing blocks until it finds a handler.

When the exception propagates to an enclosing block, the remaining executable actions in that block are bypassed.

One advantage of this behavior is that you can enclose statements that require their own exclusive error handling in their own block, while leaving more general exception handling (for example `WHEN OTHERS`) to the enclosing block.

The next slide shows an example of this.





Tell Me / Show Me

Propagating Predefined Oracle Server Exceptions from a Sub-Block

Employee_id 999 does not exist. What is displayed when this code is executed?

```
DECLARE
    v_last_name      employees.last_name%TYPE;
BEGIN
    BEGIN
        SELECT last_name INTO v_last_name
            FROM employees WHERE employee_id = 999;
        DBMS_OUTPUT.PUT_LINE('Message 1');
    EXCEPTION
        WHEN TOO_MANY_ROWS THEN
            DBMS_OUTPUT.PUT_LINE('Message 2');
    END;
    DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```



Tell Me / Show Me

Propagating User-named Exceptions from a Sub-Block

What happens when this code is executed?

```
BEGIN
  DECLARE
    e_myexcep    EXCEPTION;
  BEGIN
    RAISE e_myexcep;
    DBMS_OUTPUT.PUT_LINE('Message 1');
  EXCEPTION
    WHEN TOO_MANY_ROWS THEN
      DBMS_OUTPUT.PUT_LINE('Message 2');
  END;
  DBMS_OUTPUT.PUT_LINE('Message 3');
EXCEPTION
  WHEN e_myexcep THEN
    DBMS_OUTPUT.PUT_LINE('Message 4');
END;
```



Tell Me / Show Me

Scope of Exception Names

Predefined Oracle server exceptions, such as `NO_DATA_FOUND`, `TOO_MANY_ROWS`, and `OTHERS` are not declared by the programmer. They can be raised in any block and handled in any block.

User-named exceptions (non-predefined Oracle server exceptions and user-defined exceptions) are declared by the programmer as variables of type `EXCEPTION`. They follow the same scoping rules as other variables.

Therefore, a user-named exception declared within an inner block cannot be referenced in the exception section of an outer block.

Tell Me / Show Me

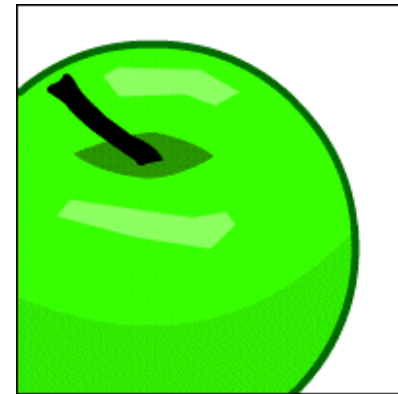
Terminology

Key terms used in this lesson include:

Scope

Visibility

Qualifier

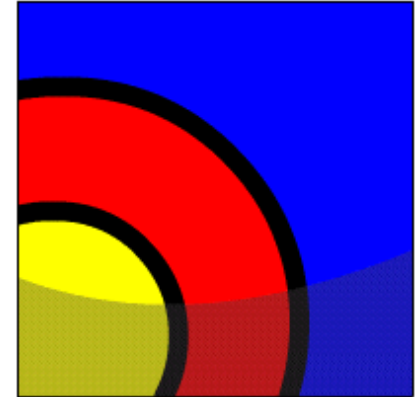




Summary

In this lesson, you learned to:

- Describe the rules for variable scope when a variable is nested in a block.
- Recognize a variable-scope issue when a variable is used in nested blocks
- Qualify a variable nested in a block with a label
- Describe the scope of an exception
- Recognize an exception-scope issue when an exception is within nested blocks
- Describe the effect of exception propagation in nested blocks





Try It / Solve It

The exercises in this lesson cover the following topics:

- Describing the rules for variable scope when a variable is nested in a block.
- Recognizing a variable-scope issue when a variable is used in nested blocks
- Qualifying a variable nested in a block with a label
- Describing the scope of an exception
- Recognizing an exception-scope issue when an exception is within nested blocks
- Describing the effect of exception propagation in nested blocks

