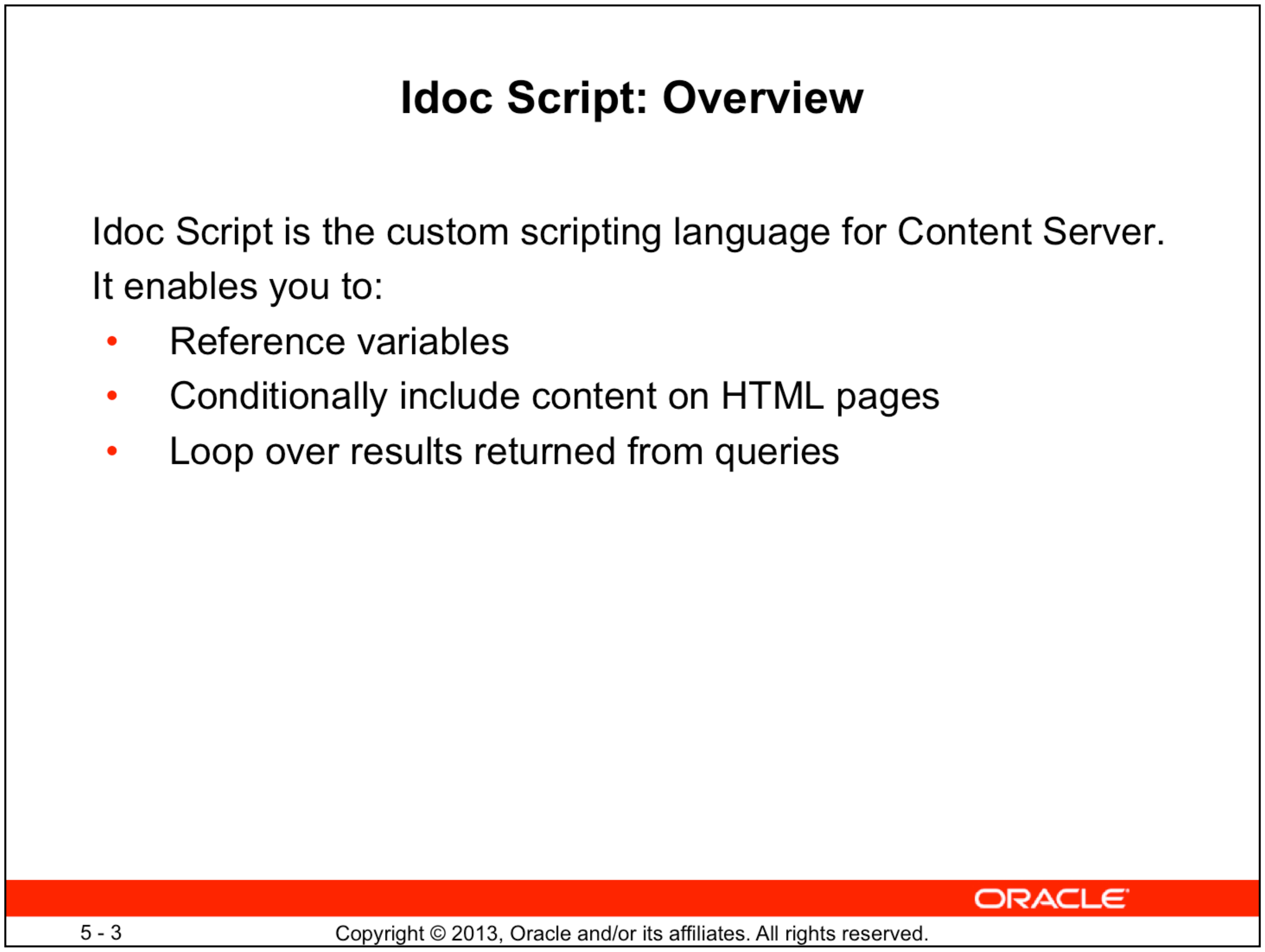
5 Using Idoc Script

Objectives

After completing this lesson, you should be able to:

- Describe the programming capabilities of Idoc Script
- Use Idoc Script to change the functionality and presentation of Content Server



Idoc Script: Overview

Idoc Script is the custom scripting language for Content Server.

It enables you to:

- Reference variables
- Conditionally include content on HTML pages
- Loop over results returned from queries

Idoc Script is the proprietary scripting language for Content Server. It is:

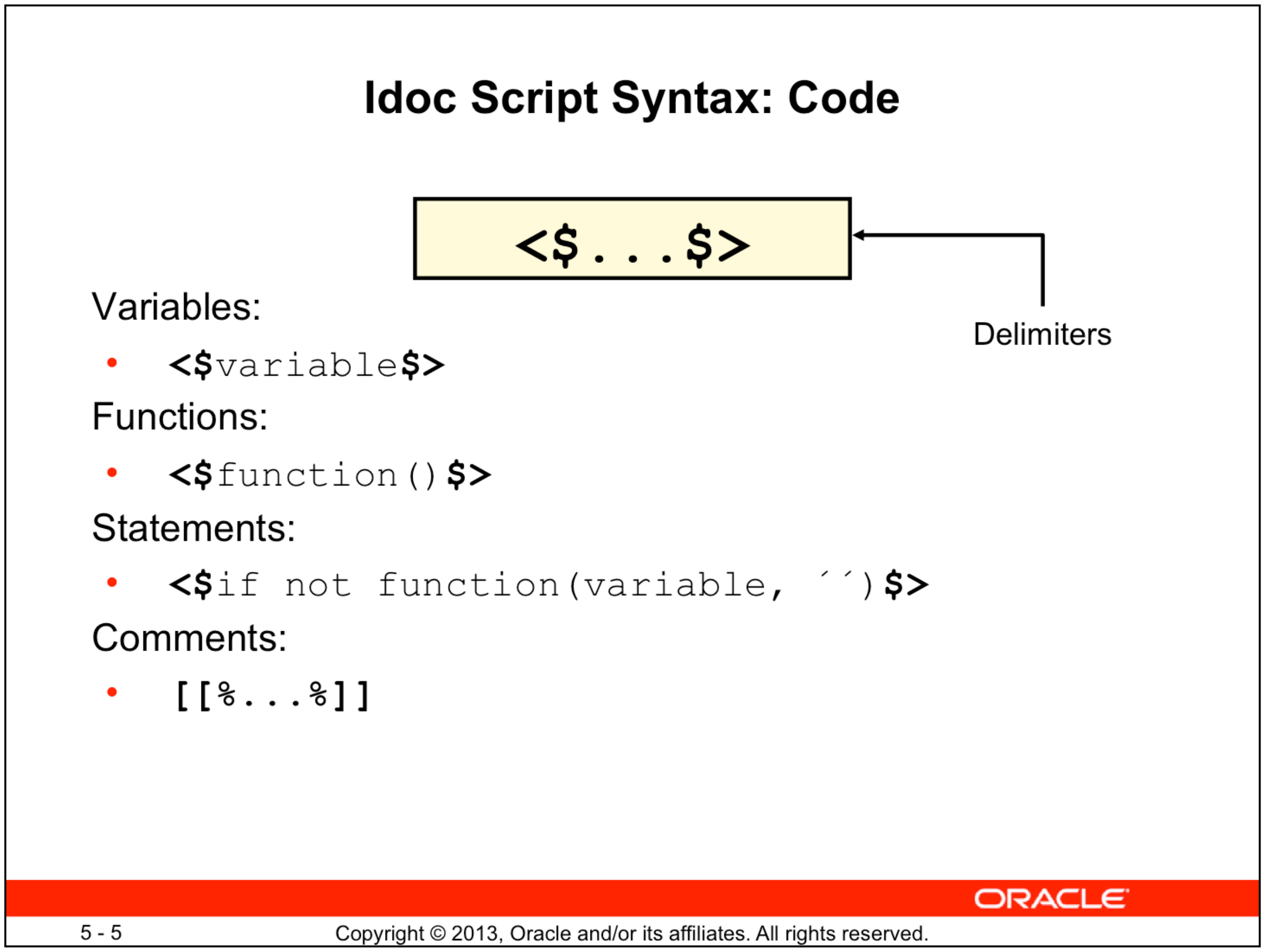
- A true scripting language
  - **Simple:** It has only a few keywords, common variables, and functions.
  - **Based on open and accepted standards (Java)**
  - **Easy:** Instead of writing a page of JSP code to get and display a page, you can write a paragraph of Idoc Script.
- Designed primarily to deal with the display of data that is returned by requests rather than make requests and return data
  - Idoc Script provides a way to process page elements after the browser has made a request, but before the requested page is returned.
- An integral part of Content Server
  - It is used in the core of Content Server.
  - Its advantage is in the quick access to custom variables and functions, globally or locally.

Idoc Script: Elements

There are five basic uses for Idoc Script:

- **Variables:** Define and substitute variable values.
- **Functions:** Use Idoc Script to perform actions and return results.
- **Conditionals:** Evaluate `if` and `else` clauses to include or exclude code from an assembled Content Server page.
- **Looping:** Repeat code for each row in a `ResultSet` that is returned from a query.
- **Includes:** Reuse pieces of Idoc Script and HTML code.

ORACLE



Idoc Script Syntax: Code

`<$...$>`

Delimiters

Variables:

- `<$variable$>`

Functions:

- `<$function()$>`

Statements:

- `<$if not function(variable, ´´)$>`

Comments:

- `[[%...%]]`

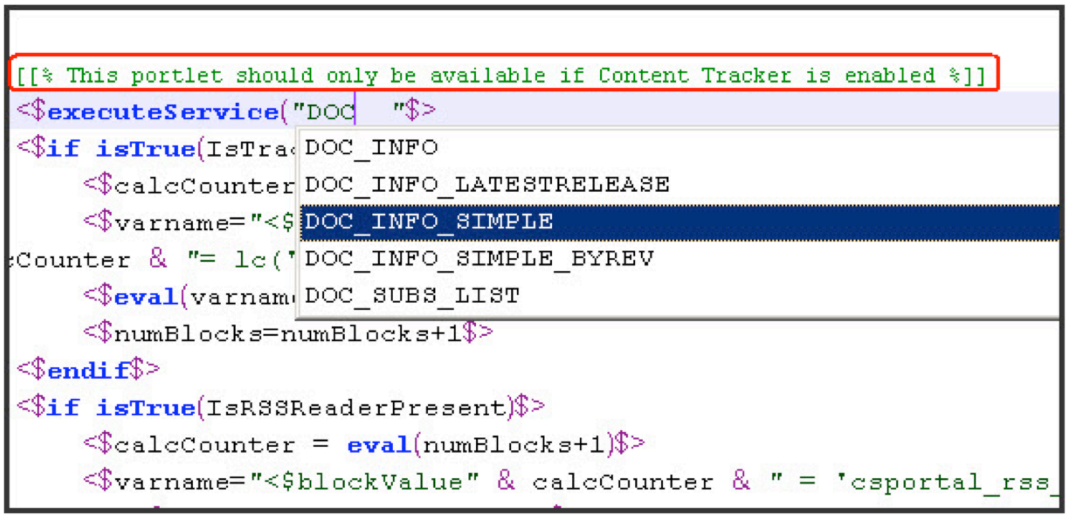
Standard Idoc Script statements begin with `<$` and end with `$>`.

Each individual Idoc Script statement requires its own `<$ $>` delimiters.

Idoc Script Syntax: Comments

```
[[%...%]]
```

Delimiters



```
[[% This portlet should only be available if Content Tracker is enabled %]]
<$executeService("DOC  "$>
<$if isTrue(IsTra
     <$calcCounter
     <$varname="<$
Counter & "= lc('
     <$eval(varnam
     <$numBlocks=numBlocks+1$>
<$endif$>
<$if isTrue(IsRSSReaderPresent)$>
     <$calcCounter = eval(numBlocks+1)$>
     <$varname="<$blockValue" & calcCounter & " = 'csportal_rss_
```

Variables

A variable enables you to define and substitute variable values.

- Defining Variables
- Referencing Variables
- Predefined Variables

5 - 7

These topics are covered in the next few slides.

Defining Variables

String Variable:

- **`<$variable_name="value"$>`**
- That is: `<$color="red"$>`

Numeric Variable:

- **`<$variable_name=number$>`**
- That is: `<$count=0$>`

Idoc Script supports multiple-value assignment clauses in a single script block, separated by commas.

- **`<$variable_name1="string_value1",`**
  `variable_name2=numeric_value2$>`
- That is: `<$day="Monday", month="April"$>`

Note: You can also define global variables. These are created with a different syntax because they are created in an Environment resource file, in a different format. This is covered in the lesson titled "Glue File and Environment Resource."

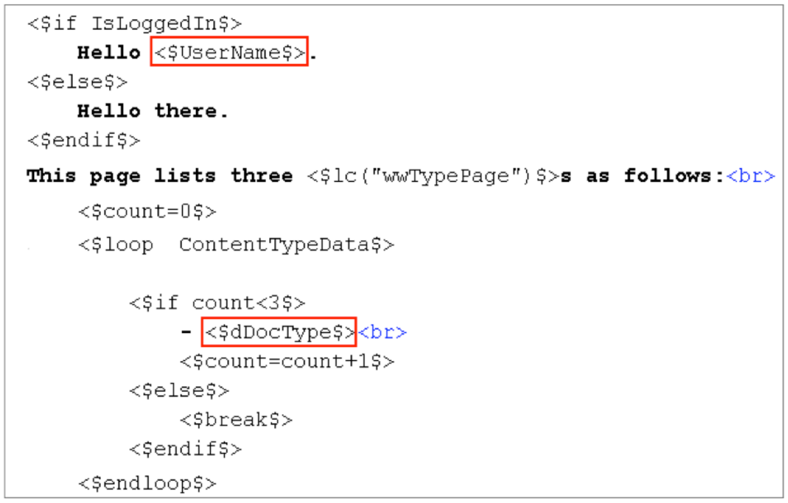
Variables

- ✓ Defining Variables
- Referencing Variables
- Predefined Variables

Referencing Variables

Variables can be referenced in Templates and other resource files with Idoc Script tags as follows:

```
<$variable_name$>
```



```
<$if IsLoggedIn$>
    Hello <$UserName$>.
<$else$>
    Hello there.
<$endif$>
This page lists three <$lc("wwTypePage")$>s as follows:<br>
    <$count=0$>
    <$loop  ContentTypeData$>

        <$if count<3$>
            - <$dDocType$><br>
            <$count=count+1$>
        <$else$>
            <$break$>
        <$endif$>
    <$endloop$>
```

Variables

- ✓ Defining Variables
- ✓ Referencing Variables
- • Predefined Variables

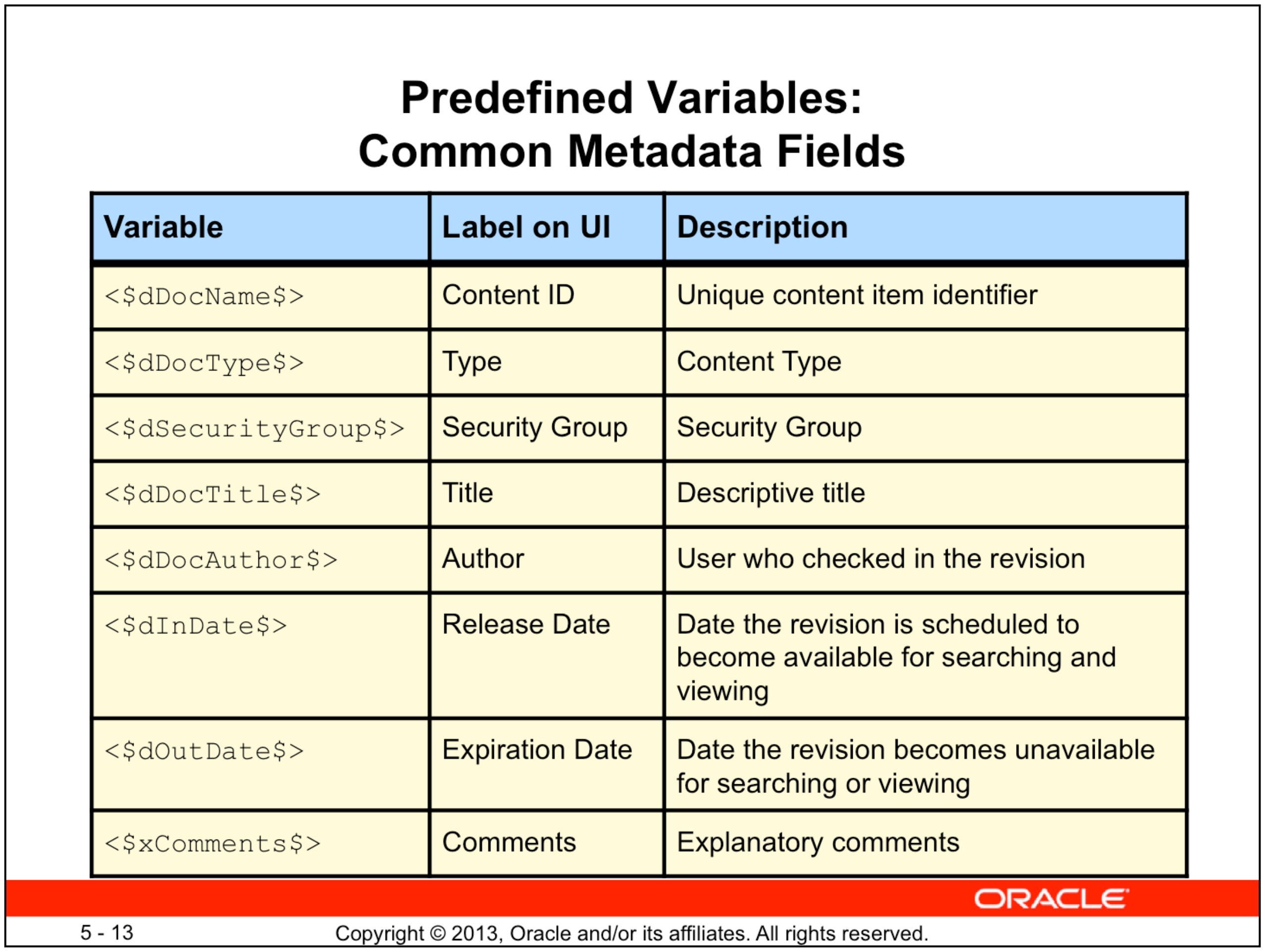
Predefined Variables

There are many predefined variables in Content Server. Some examples are described in the following table:

Variable	Description
`<$HttpCgiPath$>`	Retrieves the Content Server CGI path as a string
`<$IsLoggedIn$>`	Checks whether the current user is logged in
`<$UserIsAdmin$>`	Checks whether the current user has full administrative rights
`<$UserName$>`	Returns the login identification of the user currently logged in
`<$UserFullName$>`	Returns the full name of the user currently logged in
`<$UserRoles$>`	Returns a comma-separated list of roles that the currently logged-in user belongs to
`<$isNew$>`*	Checks whether the current page is the new checkin page

For a complete listing of predefined variables, refer to the documentation: *Developing with Oracle WebCenter Content.*

***isNew** is a page flag. For a more extended list of page flags, refer to the lesson titled “Changing Metadata Attributes.”



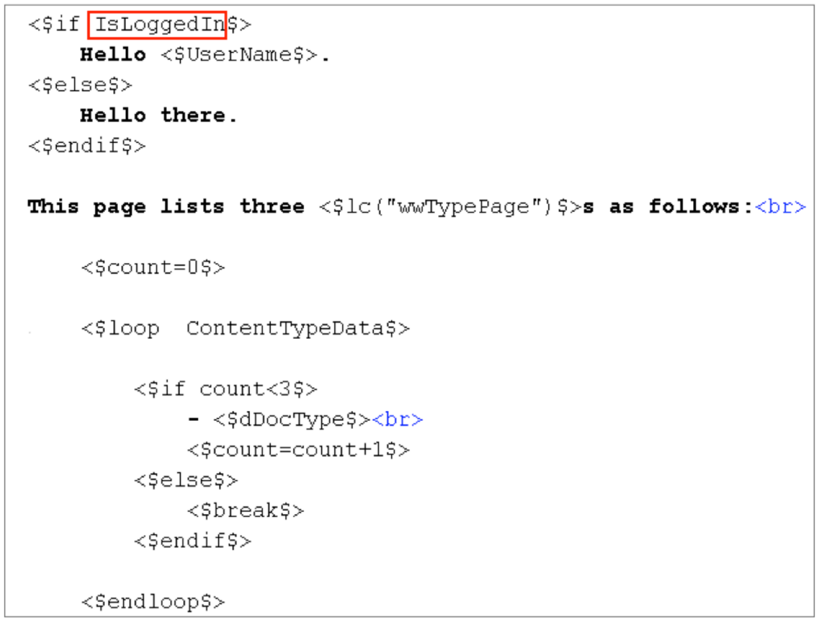
Predefined Variables: Common Metadata Fields

Variable	Label on UI	Description
`<$dDocName$>`	Content ID	Unique content item identifier
`<$dDocType$>`	Type	Content Type
`<$dSecurityGroup$>`	Security Group	Security Group
`<$dDocTitle$>`	Title	Descriptive title
`<$dDocAuthor$>`	Author	User who checked in the revision
`<$dInDate$>`	Release Date	Date the revision is scheduled to become available for searching and viewing
`<$dOutDate$>`	Expiration Date	Date the revision becomes unavailable for searching or viewing
`<$xComments$>`	Comments	Explanatory comments

5 - 13

For a complete listing of metadata fields, refer to the documentation: *Developing with Oracle WebCenter Content.*

Referencing a Variable



```
<$if IsLoggedIn$>
    Hello <$UserName$>.
<$else$>
    Hello there.
<$endif$>

This page lists three <$lc("wwTypePage")$>s as follows:<br>

    <$count=0$>

    <$loop  ContentTypeData$>

        <$if count<3$>
            - <$dDocType$><br>
            <$count=count+1$>
        <$else$>
            <$break$>
        <$endif$>

    <$endloop$>
```

Idoc Script Elements

There are five basic uses for Idoc Script:

- ✓ **Variables**
  - Define and substitute variable values.
- **Functions**
  - Use Idoc Script to perform actions and return results.
- **Conditionals**
  - Evaluate `if` and `else` clauses to include or exclude code from an assembled Content Server page.
- **Looping**
  - Repeat code for each row in a `ResultSet` that is returned from a query.
- **Includes**
  - Reuse pieces of Idoc Script and HTML code.

ORACLE

Functions

Functions perform actions and return results. Sometimes, these are the results of calculations and comparisons.

Idoc Script has a number of built-in functions that perform:

- Various string comparison and manipulation routines
- Date formatting
- `ResultSet` manipulation

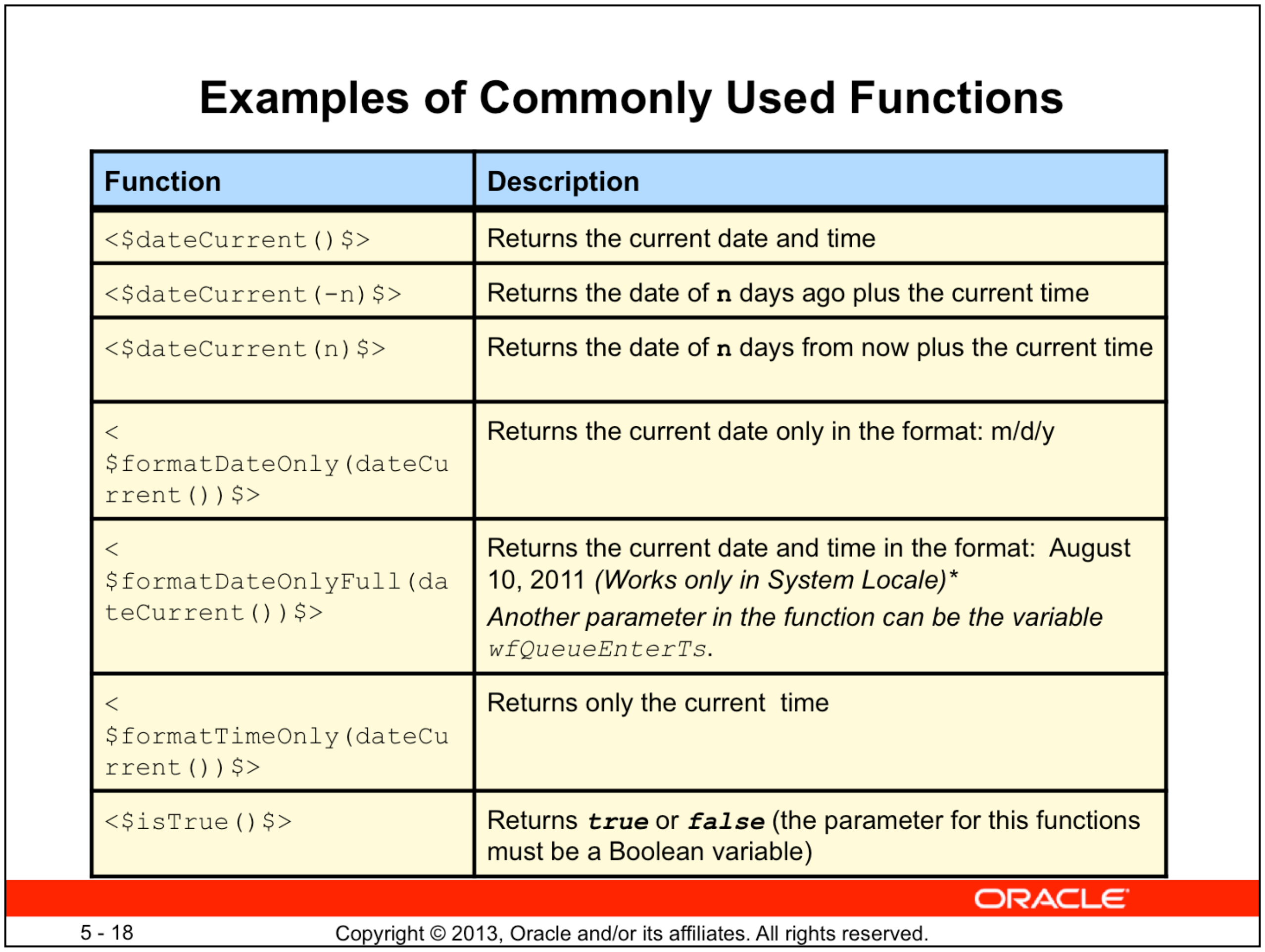
Note: An advanced Idoc Script programmer can create custom Idoc Script functions with Java. This is covered in the lesson titled "Creating Custom Idoc Script Functions with Java Code."

Functions and Parameters

Parameters (arguments) are passed to functions by enclosing the information in parentheses after the name of the function.

Some functions:

- Do not take parameters:
  - `<$function()$>`
- Take only one parameter:
  - `<$function(parameter)$>`
- Take several parameters:
  - `<$function(param1, param2,…,paramn)$>`



Examples of Commonly Used Functions

Function	Description
`<$dateCurrent()$>`	Returns the current date and time
`<$dateCurrent(-n)$>`	Returns the date of **`n`** days ago plus the current time
`<$dateCurrent(n)$>`	Returns the date of **`n`** days from now plus the current time
`<$formatDateOnly(dateCurrent())$>`	Returns the current date only in the format: m/d/y
`<$formatDateOnlyFull(dateCurrent())$>`	Returns the current date and time in the format: August 10, 2011 *(Works only in System Locale)** *Another parameter in the function can be the variable `wfQueueEnterTs`.*
`<$formatTimeOnly(dateCurrent())$>`	Returns only the current time
`<$isTrue()$>`	Returns ***`true`*** or ***`false`*** (the parameter for this functions must be a Boolean variable)

5 - 18 Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

***`<$formatDateWithPattern()$>`**

- It reformats a date/time to a specified date/time pattern.
- It takes two parameters:
  - The first parameter is a date string used by Content Server, or a date object created with the `parseDate()` or `dateCurrent()` functions.
  - The second parameter is the date/time pattern, such as MM/dd/yyyy.
- The capital letter Z denotes the use of a UTC time zone for the entry. The lowercase zzzz denotes the time offset (HHMM) from the UTC time, preceded by a plus (+) or minus (-) sign to indicate the offset.
- Output:
  - Returns date/time in the format specified by the pattern parameter
  - Returns `null` if the parameter cannot be evaluated
- Example
  - Displays Sat, 24 Jun 2012 12:08:56 -0700:
    - `<$formatDateWithPattern(dateCurrent(),"EEE, d MMM yyyy HH:mm:ss zzzz")$>`

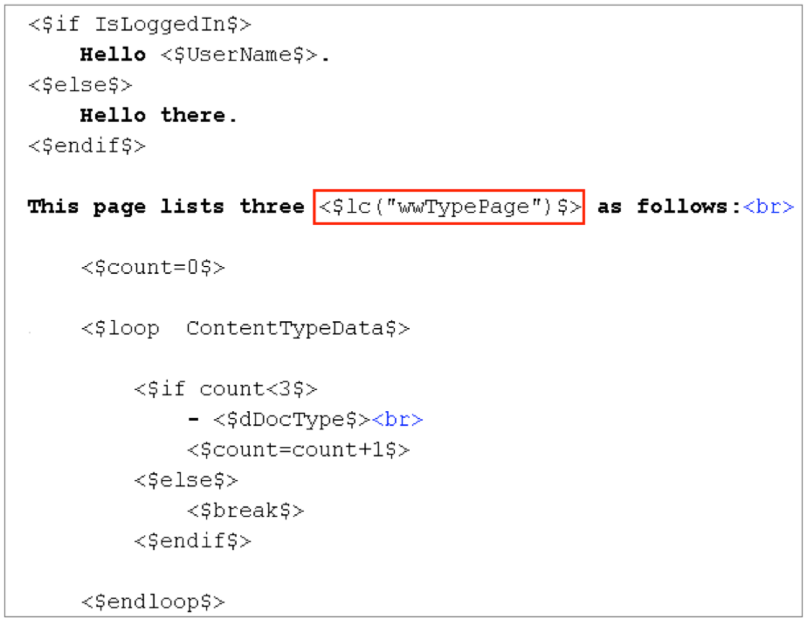
- Example:
  - Displays Sat, 24 Jun 2012 12:08:56 -0700:
    - `<$formatDateWithPattern(dateCurrent(),"EEE, d MMM yyyy HH:mm:ss zzzz")$>`
  - Displays 2012-06-24 14:30:33Z:
    - `<$formatDateWithPattern(dateCurrent(),"yyyy-MM-dd HH:mm:ssZ")$>`

Commonly Used Functions: Examples

Function	Description
`<$isComponentEnabled("ComponentName")$>`	Is a Boolean variable that checks the state of the specified component
`<$getUserValue("userMetadata"$>`	Returns the value of a user metadata field for the current user. Example parameters: `<$getUserValue("dFullName"$>` `<$getUserValue("uManager"$>`
`<$strEquals(variable, "value")$>`	Is a Boolean variable that checks whether the variable (first parameter) contains the value specified (second parameter, case-sensitive); returns ***true*** or ***false***
`<$strEqualsIgnorecase(variable, "value")$>`	Is a Boolean variable that checks whether the variable (first parameter) contains the value specified (second parameter) regardless of case; returns ***true*** or ***false***

ORACLE

Functions: Example



```
<$if IsLoggedIn$>
    Hello <$UserName$>.
<$else$>
    Hello there.
<$endif$>

This page lists three <$lc("wwTypePage")$> as follows:<br>

    <$count=0$>

    <$loop  ContentTypeData$>

        <$if count<3$>
            - <$dDocType$><br>
            <$count=count+1$>
        <$else$>
            <$break$>
        <$endif$>

    <$endloop$>
```

Idoc Script Elements

There are five basic uses for Idoc Script:

- ✓ **Variables**
  - Define and substitute variable values.
- ✓ **Functions**
  - Use Idoc Script to perform actions and return results.
- **Conditionals**
  - Evaluate `if` and `else` clauses to include or exclude code from an assembled Content Server page.
- **Looping**
  - Repeat code for each row in a `ResultSet` that is returned from a query.
- **Includes**
  - Reuse pieces of Idoc Script and HTML code.

Conditionals

A conditional statement enables you to use `if` and `else` clauses to:

- Control the execution of a block of code
- Include or exclude code from an assembled page

The conditional keywords are:

- Required
  - `<$if$>`
  - `<$endif$>`
- Optional
  - `<$else$>`
  - `<$elseif$>`

Conditional Statements: `if` Statement

Use the following statement to execute code only `if` a specified condition is `true`.

```
<$if condition$>
  execute code
<$endif$>
```

Conditional Statements: `if...else` Statement

Use this statement to execute:

- Some code `if` the condition is `true`
- Another code `if` the condition is `false`

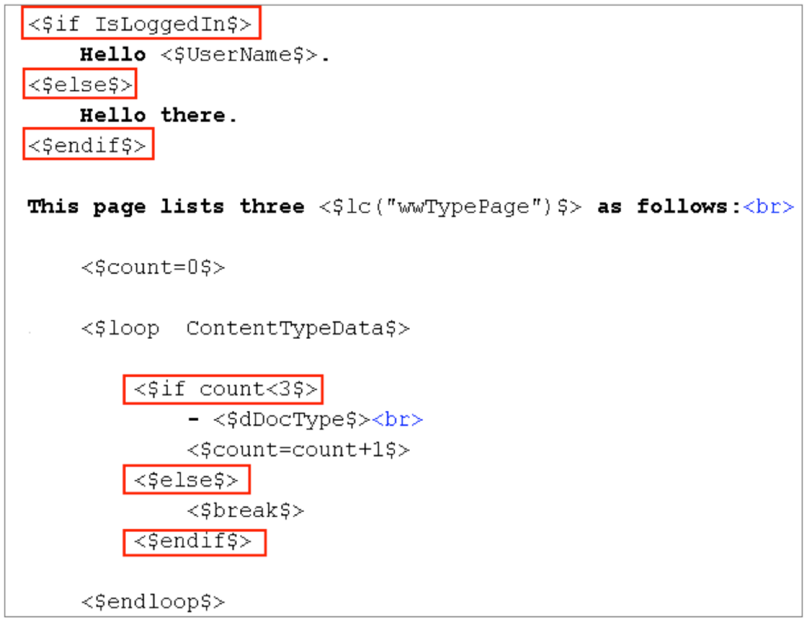
```
<$if conditionA$>
  execute codeA
<$else$>
  execute code
<$endif$>
```

Conditional Statements: `if...elseif...else` Statement

Use this statement to select one of many blocks of code to be executed.

```
<$if conditionA$>
  execute codeA
<$elseif conditionB$>
  execute codeB
<$else$>
  if no condition is true execute codeC
<$endif$>
```

Conditional Statements: Example



```
<$if IsLoggedIn$>
    Hello <$UserName$>.
<$else$>
    Hello there.
<$endif$>

This page lists three <$lc("wwTypePage")$> as follows:<br>

    <$count=0$>

    <$loop  ContentTypeData$>

        <$if count<3$>
            - <$dDocType$><br>
            <$count=count+1$>
        <$else$>
            <$break$>
        <$endif$>

    <$endloop$>
```

Operators and Wildcards

- Using Operators to Compare Integers
- Special String Operators
- Binary (Numeric) Operators
- Boolean Operators

ORACLE

5 - 28

Operators and Wildcards: Using Operators to Compare Integers

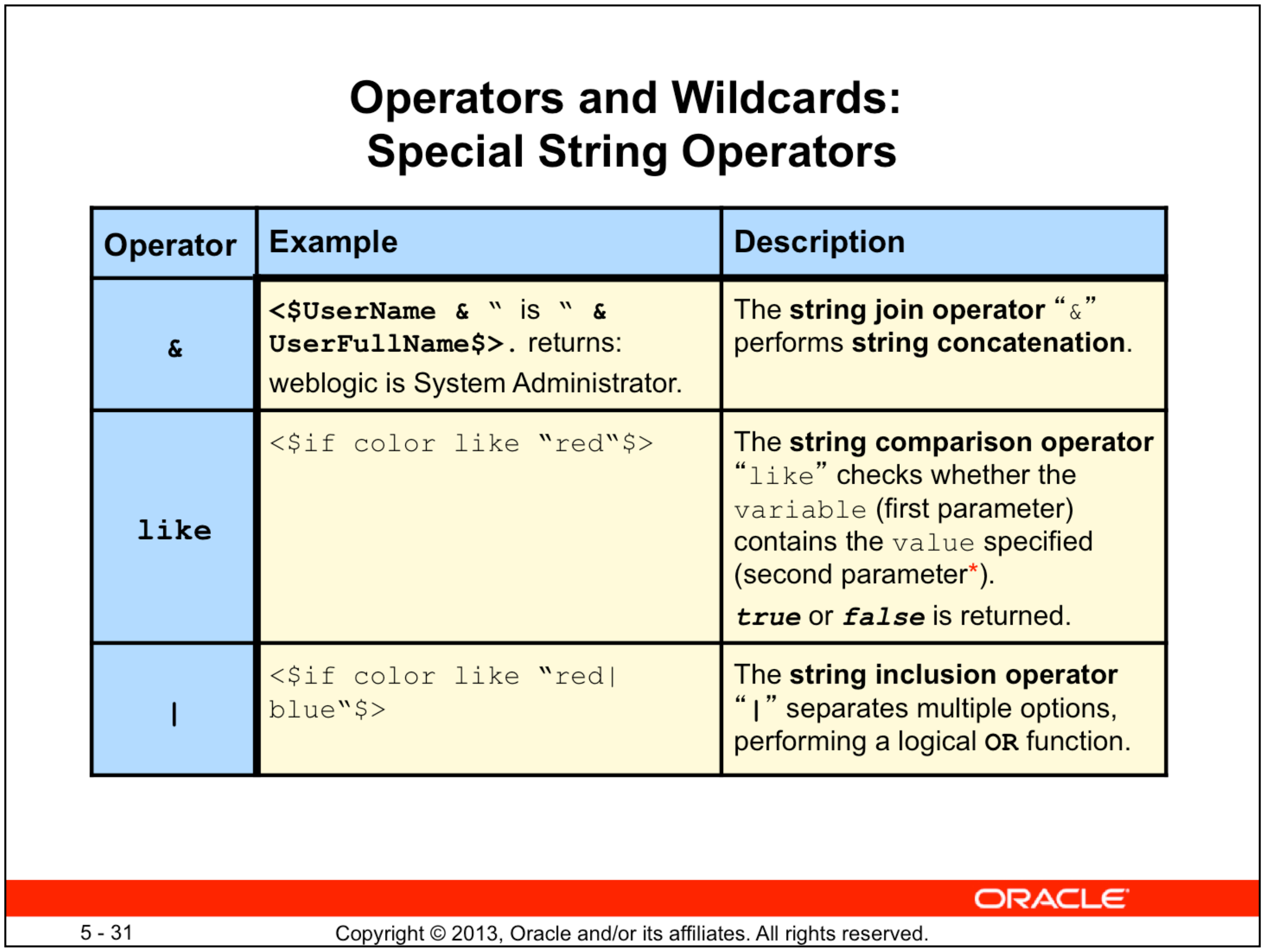
Refer to the following examples in the table:

Operator	Example based on `<$n=2$> <$x=3$>`	Description
==	`<$n==x$>` evaluates to `false`	Equality
!=	`<$n!=x$>` evaluates to `true`	Inequality
<	`<$n<x$>` evaluates to `true`	Less than
<=	`<$n<=x$>` evaluates to `true`	Less than or equal to
>	`<$n>x$>` evaluates to `false`	Greater than
>=	`<$n>=x$>` evaluates to `false`	Greater than or equal to

ORACLE

Operators and Wildcards

- ✓ Using Operators to Compare Integers
- Special String Operators
- Binary (Numeric) Operators
- Boolean Operators

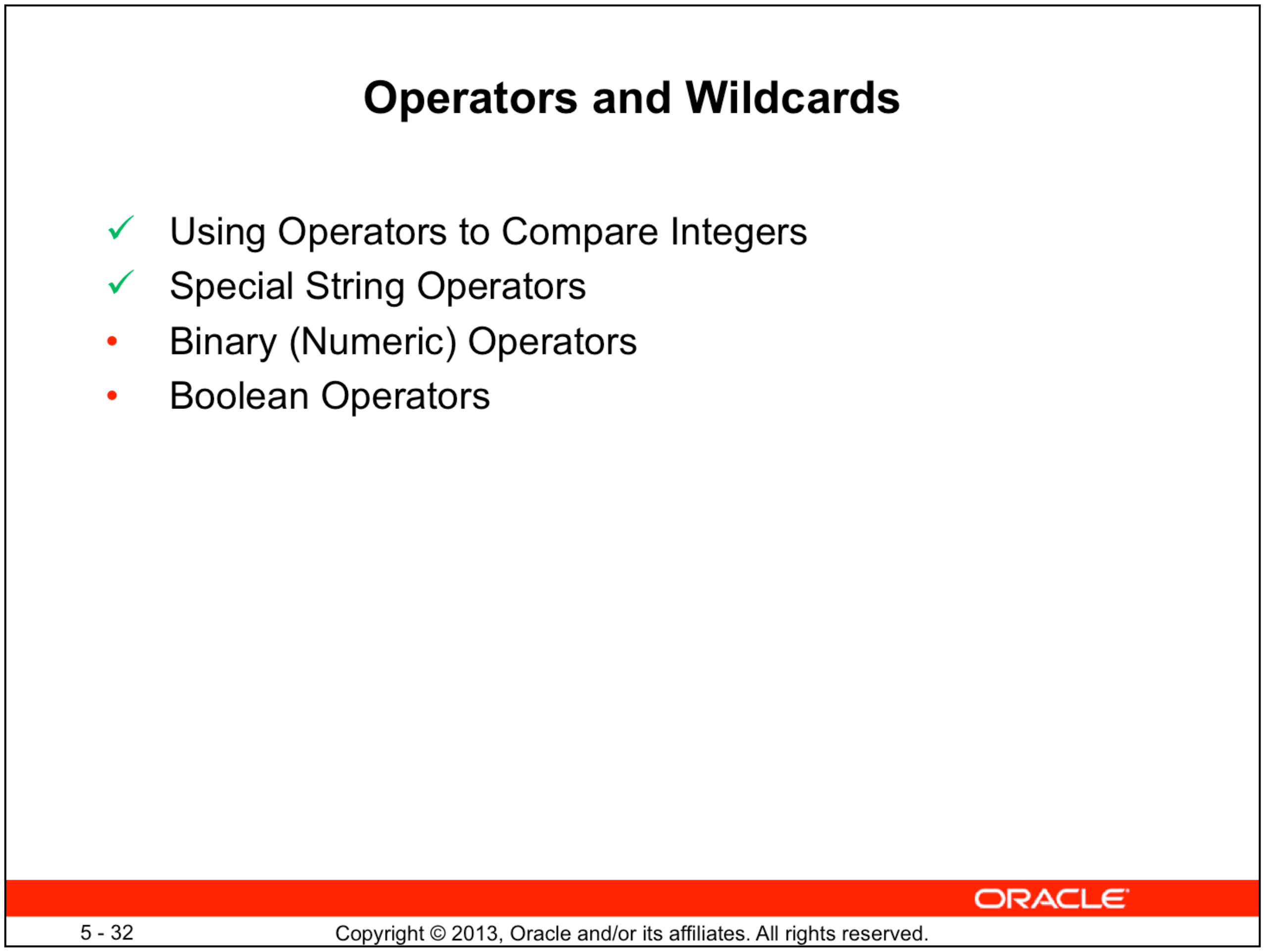


Operators and Wildcards: Special String Operators

Operator	Example	Description
`&`	`<$UserName & "` is `" & UserFullName$>.` returns: weblogic is System Administrator.	The **string join operator** “`&`” performs **string concatenation**.
`like`	`<$if color like “red“$>`	The **string comparison operator** “`like`” checks whether the `variable` (first parameter) contains the `value` specified (second parameter*). ***`true`*** or ***`false`*** is returned.
`|`	`<$if color like “red\|blue“$>`	The **string inclusion operator** “`|`” separates multiple options, performing a logical `OR` function.

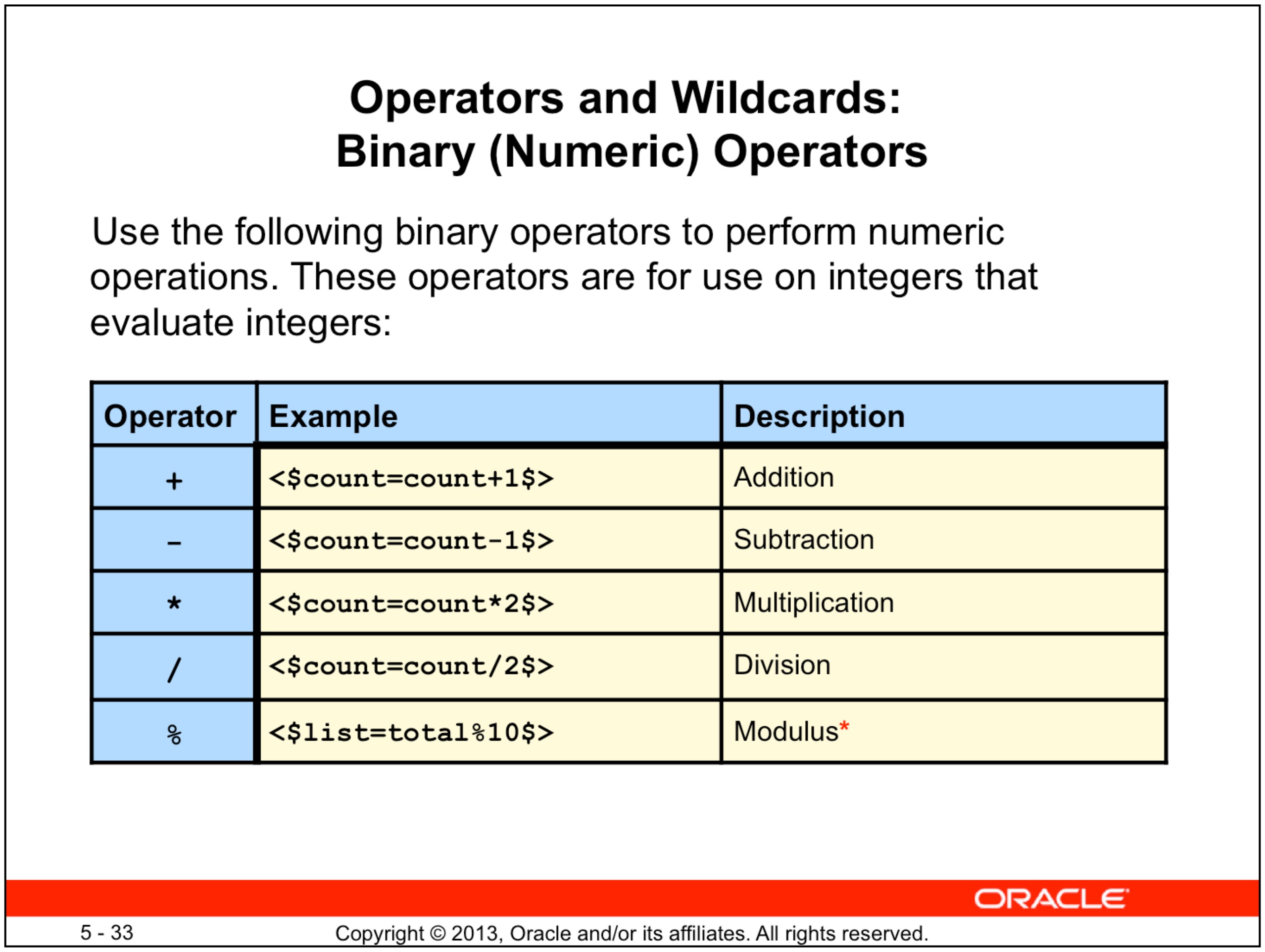
5 - 31

*The string on the right of the operator can use the `*` and `?` wildcard characters.



Operators and Wildcards

- ✓ Using Operators to Compare Integers
- ✓ Special String Operators
- Binary (Numeric) Operators
- Boolean Operators



Operators and Wildcards: Binary (Numeric) Operators

Use the following binary operators to perform numeric operations. These operators are for use on integers that evaluate integers:

Operator	Example	Description
+	`<$count=count+1$>`	Addition
-	`<$count=count-1$>`	Subtraction
*	`<$count=count*2$>`	Multiplication
/	`<$count=count/2$>`	Division
%	`<$list=total%10$>`	Modulus*

ORACLE

5 - 33

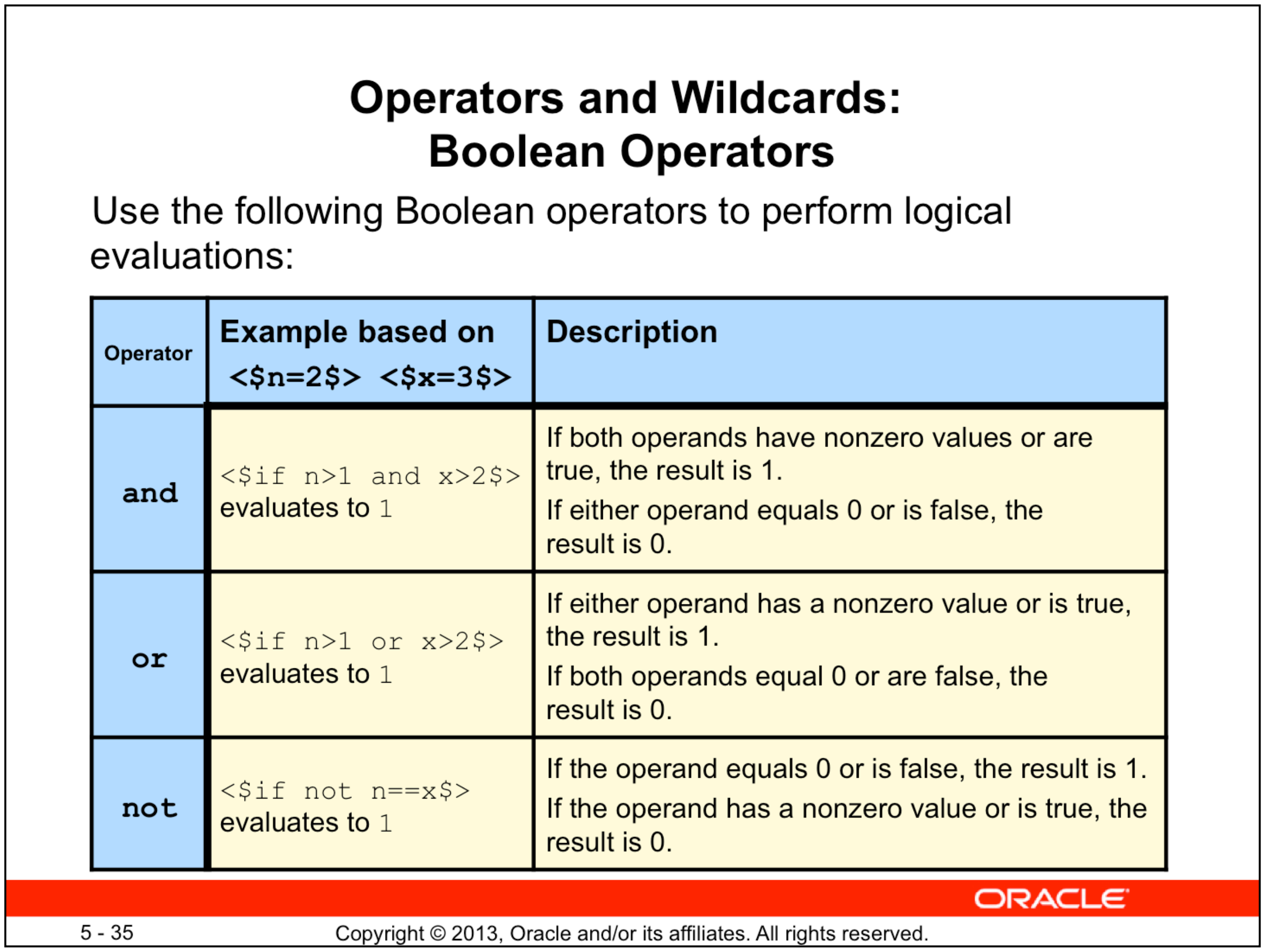
*Modulus provides the remainder of two values divided by each other

Use Case: Search Results page links

- n=10 → results per page
- x=24%n → x remainder to display on the last page
- x=4 → three pages with four results on the last page

Operators and Wildcards

- ✓ Using Operators to Compare Integers
- ✓ Special String Operators
- ✓ Binary (Numeric) Operators
- Boolean Operators



Operators and Wildcards: Boolean Operators

Use the following Boolean operators to perform logical evaluations:

Operator	Example based on `<$n=2$> <$x=3$>`	Description
`and`	`<$if n>1 and x>2$>` evaluates to `1`	If both operands have nonzero values or are true, the result is 1. If either operand equals 0 or is false, the result is 0.
`or`	`<$if n>1 or x>2$>` evaluates to `1`	If either operand has a nonzero value or is true, the result is 1. If both operands equal 0 or are false, the result is 0.
`not`	`<$if not n==x$>` evaluates to `1`	If the operand equals 0 or is false, the result is 1. If the operand has a nonzero value or is true, the result is 0.

5 - 35

Order of evaluation:

- Boolean operators evaluate from left to right. Therefore, if the first operand is sufficient to determine the result of an operation, the second operand is not evaluated.
- Parentheses can be used to affect the order of evaluation.

Evaluating Strings

- Boolean operators can be used to evaluate strings.
  - Example: `<$if isNew and isCheckin$>`
- If the string is defined, it is evaluated to `true`.
- If the string is `NULL`, it is evaluated as `false`.

Idoc Script Elements

There are five basic uses for Idoc Script:

- ✓ **Variables**
  - Define and substitute variable values.
- ✓ **Functions**
  - Use Idoc Script to perform actions and return results.
- ✓ **Conditionals**
  - Evaluate `if` and `else` clauses to include or exclude code from an assembled Content Server page.
- **Looping**
  - Repeat code for each row in a `ResultSet` that is returned from a query.
- **Includes**
  - Reuse pieces of Idoc Script and HTML code.

Looping

Loop structures allow you to repeat code. Looping can be accomplished in the following ways with Idoc Script:

- `ResultSet` looping
- While looping
- Advanced `ResultSet` manipulation

`ResultSet` Looping

`ResultSet` looping repeats a block of code for each row in a `ResultSet`.

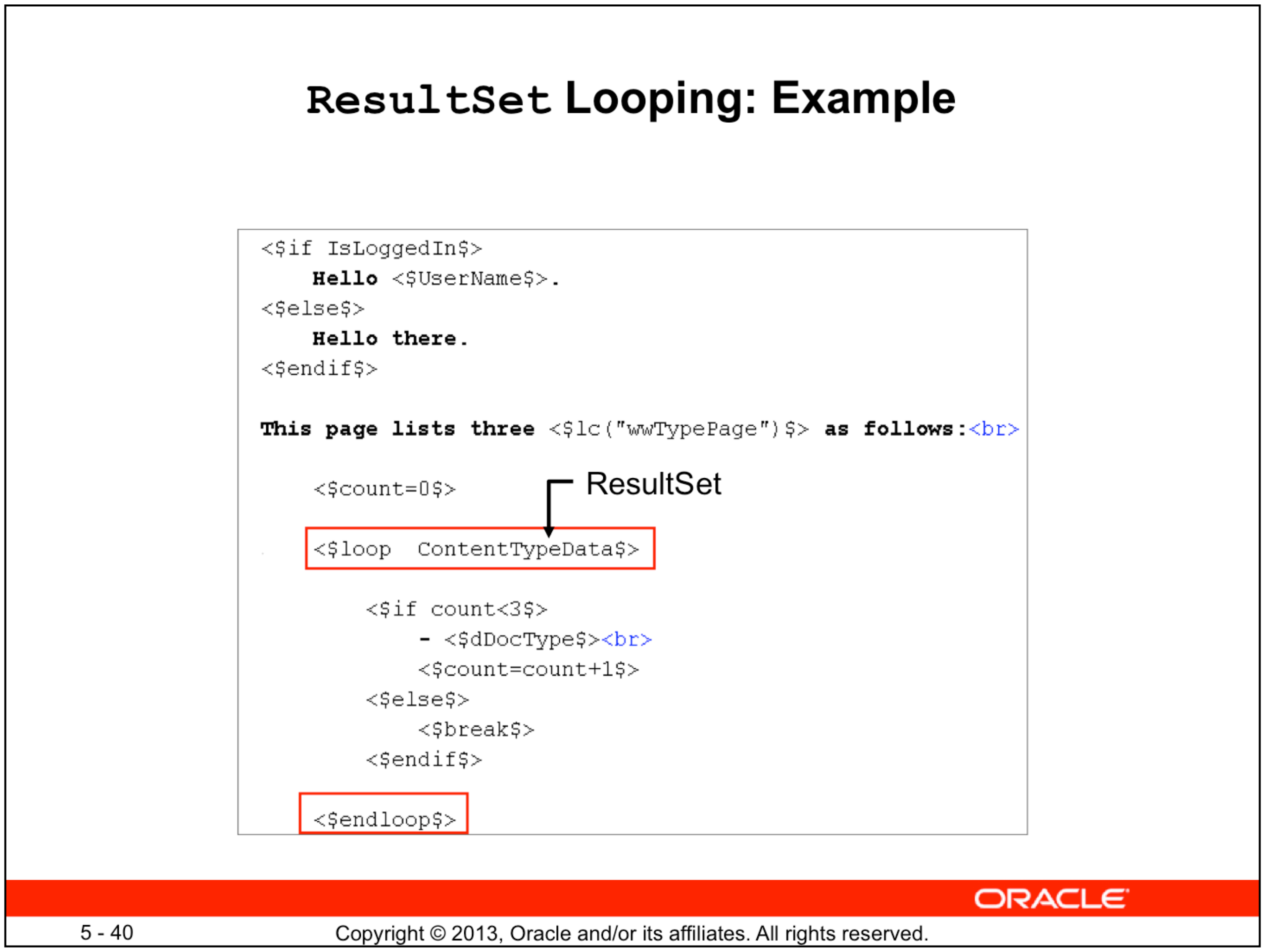
The name of the `ResultSet` to be looped through is specified as a variable by using the following syntax:

```
<$loop ResultSetName$>
    code
<$endloop$>
```

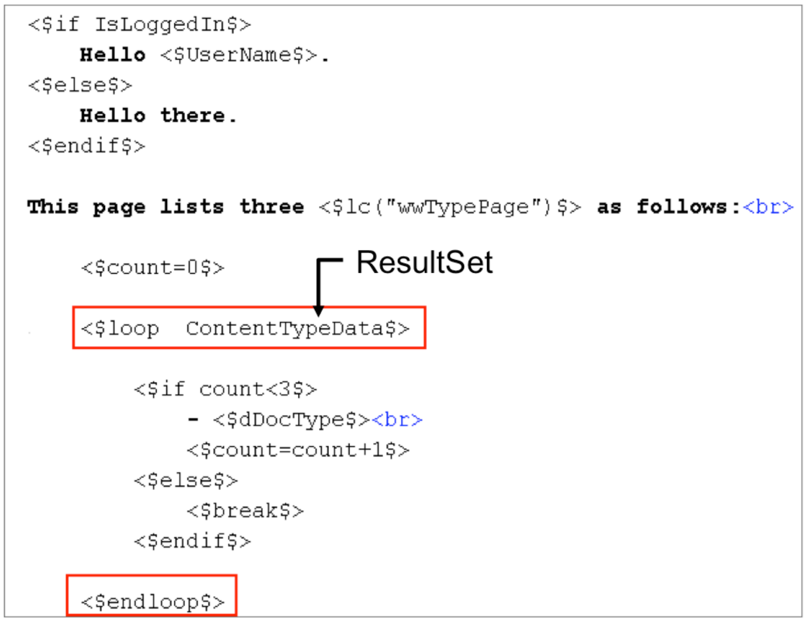
In addition to terminating the loop by using `<$endloop$>`, you can use `<$break$>`. This causes the innermost loop to be exited.

`ResultSet` Looping: Special Considerations

- The code between `<$loop$>` and `<$endloop$>` is repeated once for each row in the `ResultSet`.
- When in the `ResultSet` loop, you can reference any column of the `ResultSet`.
- Substitution of values depends on which row is currently being accessed in the loop.
- When in a `ResultSet` loop, that `ResultSet` becomes *active* and has priority over any other `ResultSet` when evaluating variables and conditional statements.



ResultSet Looping: Example



```
<$if IsLoggedIn$>
    Hello <$UserName$>.
<$else$>
    Hello there.
<$endif$>

This page lists three <$lc("wwTypePage")$> as follows:<br>

    <$count=0$>

    <$loop  ContentTypeData$>

        <$if count<3$>
            - <$dDocType$><br>
            <$count=count+1$>
        <$else$>
            <$break$>
        <$endif$>

    <$endloop$>
```

Inside a `ResultSet` loop, the following special values can be used:

`<$ResultSetName.columnName$>`

- The value of `columnName` in `ResultSetName`

`<$ResultSetName.#row$>`

- The current row index (numbered from zero)

`<$ResultSetName.numRows$>`

- The total rows in a `ResultSet`

`<$getValue()$>`

- Retrieves the value of a particular column from a specific `ResultSet`
- Retrieves information about `ResultSet` rows
- Retrieves the value of a particular metadata field from `local`, `active`, or `environment` data

  (Refer to the online *Developing with Oracle WebCenter Content* reference guide for more information.)

`while` Looping

`while` looping enables you to create a *conditional* loop.

The `while` loop uses the following syntax:

```
<$loopwhile Condition$>
    code
<$endloop$>
```

Example:

```
<$count=0$>
<$loopwhile count<10$>
    <$count=count+2$>
<$endloop$>
```

Idoc Script Elements

There are five basic uses for Idoc Script:

- ✓ **Variables**
  - Define and substitute variable values.
- ✓ **Functions**
  - Use Idoc Script to perform actions and return results.
- ✓ **Conditionals**
  - Evaluate `if` and `else` clauses to include or exclude code from an assembled Content Server page.
- ✓ **Looping**
  - Repeat code for each row in a `ResultSet` that is returned from a query.
- **Includes**
  - Reuse pieces of Idoc Script and HTML code.

HTML Includes

An Include defines pieces of code that are used to build the Content Server webpages.

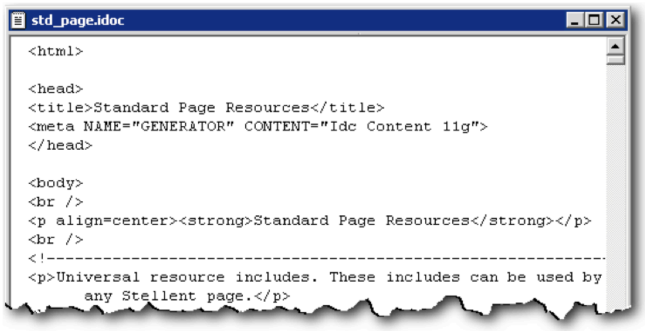
Includes:

- Defined once in a Resource file
- Referenced by Template files

HTML Includes: Characteristics

Some characteristics of HTML Includes:

- Standard Includes are defined in the `std_page.idoc`* file.
- They can contain Idoc Script, as well as HTML code, JavaScript, Java applets, cascading style sheets, and comments.
- They can be defined in:
  - The same file that they are called from
  - A separate file
- Multiple includes can be defined in a file.



```
<html>

<head>
<title>Standard Page Resources</title>
<meta NAME="GENERATOR" CONTENT="Idc Content 11g">
</head>

<body>
<br />
<p align=center><strong>Standard Page Resources</strong></p>
<br />
<!------------------------------------------------------------
<p>Universal resource includes. These includes can be used by
      any Stellent page.</p>
```

***std_page.idoc**

- Is located in the `D:/Oracle/Middleware/Oracle_ECM1/ucm/idc/resources/core/idoc` directory
  - It is also referred to as the `<IdcHomeDir>resources/core/idoc` directory in the documentation.
- Should not be edited:
  - Any required code changes must be made from within the custom components.

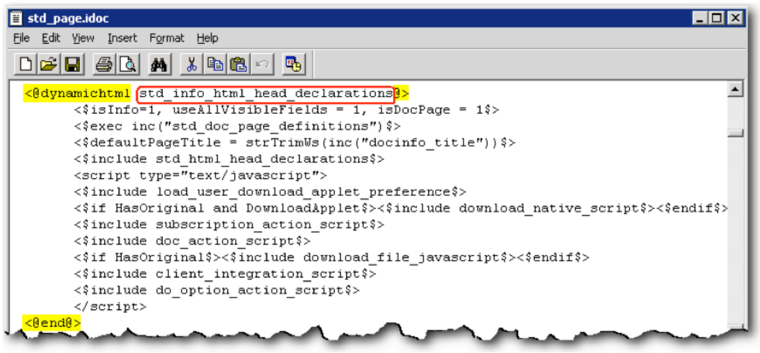
HTML Includes: Defining an HTML Include

A custom Include is defined in an HTM resource file. The Include definition delimiters are:

- `<@....@>`

The syntax is as follows:

```
<@dynamichtml include_name@>
    custom code
<@end@>
```

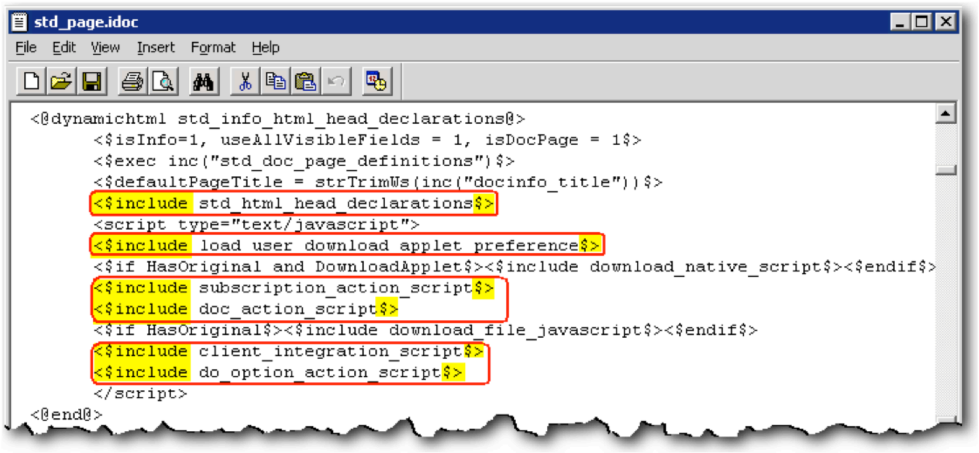


```
<@dynamichtml std_info_html_head_declarations@>
      <$isInfo=1, useAllVisibleFields = 1, isDocPage = 1$>
      <$exec inc("std_doc_page_definitions")$>
      <$defaultPageTitle = strTrimWs(inc("docinfo_title"))$>
      <$include std_html_head_declarations$>
      <script type="text/javascript">
      <$include load_user_download_applet_preference$>
      <$if HasOriginal and DownloadApplet$><$include download_native_script$><$endif$>
      <$include subscription_action_script$>
      <$include doc_action_script$>
      <$if HasOriginal$><$include download_file_javascript$><$endif$>
      <$include client_integration_script$>
      <$include do_option_action_script$>
      </script>
<@end@>
```

Calling an HTML Include: `include` Keyword

An Include is called from an HTM resource file or an HTM Template page file by using the following Idoc Script format:

`<$include` *`include_name`***`$>`**

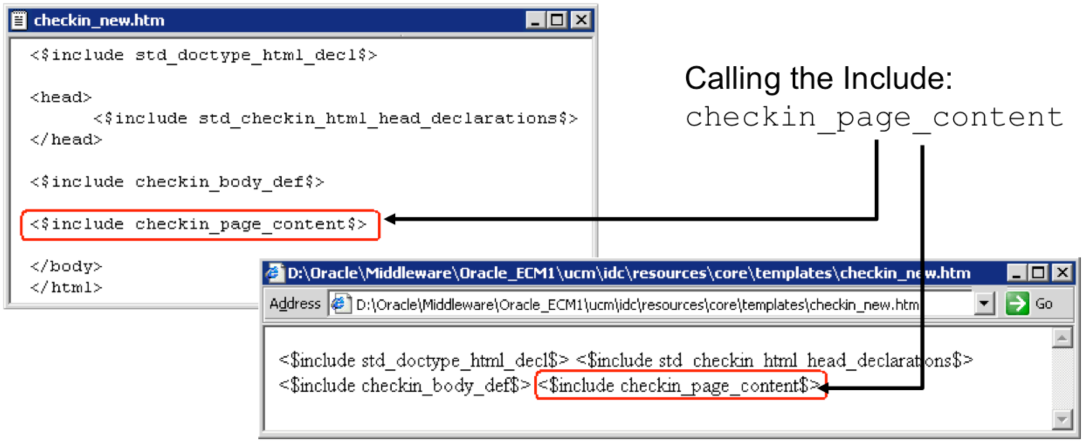


```
<@dynamichtml std_info_html_head_declarations@>
      <$isInfo=1, useAllVisibleFields = 1, isDocPage = 1$>
      <$exec inc("std_doc_page_definitions")$>
      <$defaultPageTitle = strTrimWs(inc("docinfo_title"))$>
      <$include std_html_head_declarations$>
      <script type="text/javascript">
      <$include load_user_download_applet_preference$>
      <$if HasOriginal and DownloadApplet$><$include download_native_script$><$endif$>
      <$include subscription_action_script$>
      <$include doc_action_script$>
      <$if HasOriginal$><$include download_file_javascript$><$endif$>
      <$include client_integration_script$>
      <$include do_option_action_script$>
      </script>
<@end@>
```

Calling an HTML Include from an HTM Template File: Example

Template pages are dynamically assembled by calling the Includes defined in `std_page.idoc`.

Example: The Template page for the **New Check In** page calls the **`checkin_page_content`** Include.



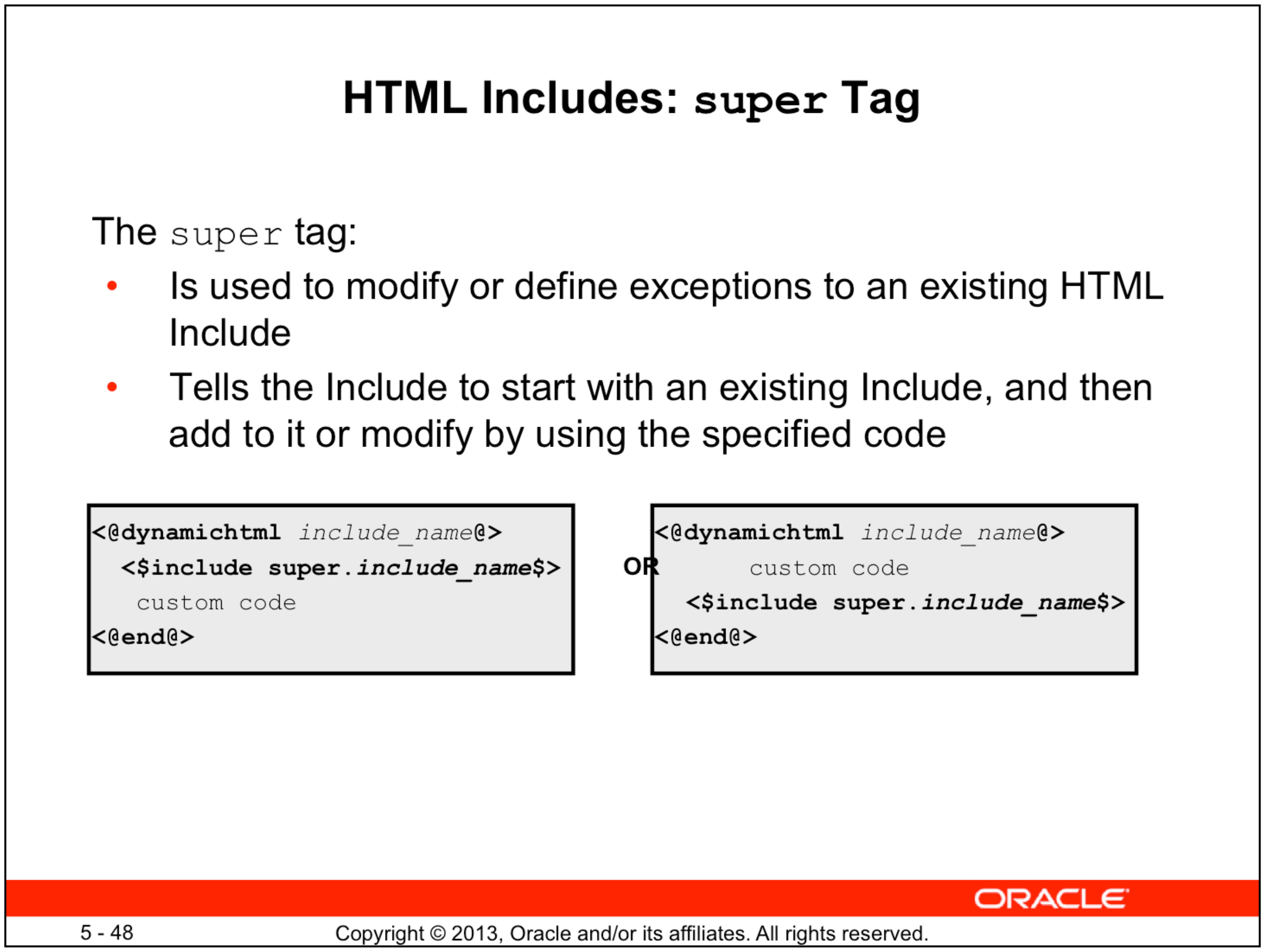
```
<$include std_doctype_html_decl$>

<head>
      <$include std_checkin_html_head_declarations$>
</head>

<$include checkin_body_def$>

<$include checkin_page_content$>

</body>
</html>
```

D:\Oracle\Middleware\Oracle_ECM1\ucm\idc\resources\core\templates\checkin_new.htm

```
<$include std_doctype_html_decl$> <$include std_checkin_html_head_declarations$>
<$include checkin_body_def$> <$include checkin_page_content$>
```

`checkin_new.htm`

- Is located in the `D:/Oracle/Middleware/Oracle_ECM1/ucm/idc/resources/core/templates` directory
  - It is also referred to as the `<IdcHomeDir>resources/core/templates` directory in the documentation.
- Should not be edited:
  - Any required code changes must be made from within the custom components.



HTML Includes: `super` Tag

The `super` tag:

- Is used to modify or define exceptions to an existing HTML Include
- Tells the Include to start with an existing Include, and then add to it or modify by using the specified code

```
<@dynamichtml include_name@>
  <$include super.include_name$>
   custom code
<@end@>
```

OR

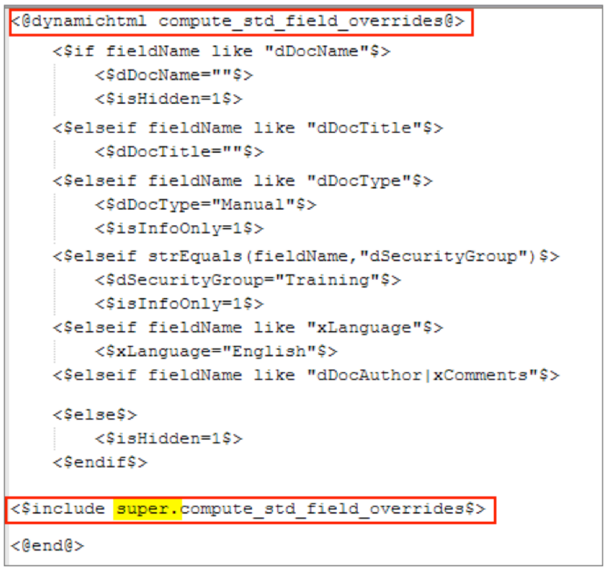
```
<@dynamichtml include_name@>
       custom code
   <$include super.include_name$>
<@end@>
```

`super` Tag

- This tag is useful when you are customizing standard code that is likely to change from one software version to the next. When you are upgarding to a new version of Content Server, the `super` tag ensures that your components are using the most recent version of the Include they are overriding, modifying only the specific code that you need to customize your Content Server instance.
- It can refer to the following:
  - Standard Include
  - Custom Include
- It is useful when you are making small customizations to large Includes.
- The Include `super` tag can be placed before or after the custom code.

HTML Includes: `super` Tag

Example:



```
<@dynamichtml compute_std_field_overrides@>
    <$if fieldName like "dDocName"$>
        <$dDocName=""$>
        <$isHidden=1$>
    <$elseif fieldName like "dDocTitle"$>
        <$dDocTitle=""$>
    <$elseif fieldName like "dDocType"$>
        <$dDocType="Manual"$>
        <$isInfoOnly=1$>
    <$elseif strEquals(fieldName,"dSecurityGroup")$>
        <$dSecurityGroup="Training"$>
        <$isInfoOnly=1$>
    <$elseif fieldName like "xLanguage"$>
        <$xLanguage="English"$>
    <$elseif fieldName like "dDocAuthor|xComments"$>

    <$else$>
        <$isHidden=1$>
    <$endif$>

<$include super.compute_std_field_overrides$>

<@end@>
```

Summary

In this lesson, you should have learned how to:

- Describe the programming capabilities of Idoc Script
- Use Idoc Script to change the functionality and presentation of Content Server

Practice 5: Overview

This practice covers the following topics:

- Securing the navigation entries of profiles based on user roles
- Calculating derived values (for Security Group)
- (Optional) Exploring Idoc Script