UPSTATE
UNIVERSITY HOSPITAL

MENU

# Lesson 3: Variables

## Learning Objectives

- Understand:
  - What variables are
  - The significance of `null`
  - Scopes of variables
- Learn how to:
  - Create a variable
  - Initialize and reinitialize a variable

Collapse all

### What Is a Variable?

- A variable is a programming construct
- A variable is an identifier preceded by a dollar sign; example: `$var`
- The identifier is subject to certain naming conventions:
  - The first character must be either an alphabet or `_` (underscore)
  - Mixture of uppercase and lowercase characters is allowed
  - Numeric characters are allowed
  - Dashes ( – ) are allowed
  - Non-ascii Unicode characters are not allowed
  - Space characters and other whitespace characters are not allowed
- A variable has the following properties:
  - It has a name (or identifier); in the case of `$var`, `var` is the name or identifier of the variable

- It can be associated with a value; normally the value is a Java object
- When it is associated with a value, we will say that the variable is defined
- When it is associated with a value, it has a type; use `$var.Class.Name` to reveal the type
- The associated object has an address; try `#evaluate( $_ )`, which outputs something like `com.hannonhill.cascade.velocity.LocatorTool@9eb578e`

---

## Creating a Variable Using `#set`

---

- We can use the `#set` directive to create a variable; example:

```
#set( $var = "Hello" )
```

- There are five parts in a `#set` statement:
  - The `#set` directive
  - A pair of parentheses following the `#set` directive
  - Inside the pair of parentheses, a variable (the left hand side, or LHS), the assignment operator `=`, and a value to be assigned to the variable (the right hand side, or RHS)

- The RHS can be a very complicated expression; for example:

```
#set( $chanwModifier = $_.Class.forName( "java.lang.reflect.Modifier" ).getMethod(
    "toString",
    $_FieldTool.in(
      "java.lang.Integer" ).TYPE ).invoke(
      null, $method.getModifiers() ) )
```

- The `#set` directive can never be used to assign a `null` value to a variable
- If the RHS fails to yield a valid (i.e. non- `null`) value, then the assignment fails
- When an assignment fails, the variable either remains undefined (keeping its original `null` value), or retains its old value

---

## Creating a Variable in the `#foreach` Directive

---

- The `#foreach` directive is used to loop through a list of object
- Within the `#foreach` structure, a local variable is created
- The name/identifier of the local variable is created by you

- For every loop, a different object (i.e., value) will be associated with the local variable
- Example:

```
#foreach( $num in [ 1..5 ] )
    $num
#end
```

- There are seven parts in a `#foreach` structure:
  - The `#foreach` directive
  - A pair of parentheses following the `#foreach` directive
  - The `#end` directive
  - Inside the pair of parentheses, a local variable to hold the current value in each loop
  - The `in` keyword
  - A list of objects/values
  - The code of the structure between the ending parenthesis and the `#end` directive
  - When this loop is executed, the variable `$num` will hold the values 1, 2, 3, 4, and 5 in turn
  - `$num` is local in the sense that it is defined only within the `#foreach` structure and can never be used outside the structure

---

### Creating a Variable in the `#macro` Directive

---

- The `#macro` directive is used to create a macro
- A macro is a block of reusable code with a name
- Example of a macro:

```
#macro( myMacro $var )
    ## code of macro here
#end
```

- There are five or six parts in a macro:
  - The `#macro` directive
  - A pair of parentheses following the `#macro` directive
  - The `#end` directive
  - Inside the pair of parentheses, the identifier (or name) of the macro, optionally followed by one or more variables created by you
  - The code of the macro between the ending parenthesis and the `#end` directive

- To invoke the macro, use code of the following type:

```
#myMacro( "hey" )
```

- When the code of the macro is executed, the local variable `$var` will store the String `hey`
- `$var` is local in the sense that it is defined only within the macro and can never be used outside

## Pre-Defined Contextual Variables

- Within the context of a `#foreach` structure and a macro, there can be variables made available by the Velocity engine
- Inside a `#foreach` structure, there is a pre-defined variable `$foreach`
- The type of `$foreach` is `org.apache.velocity.runtime.directive.ForeachScope`
- Inside a macro, if certain conditions are met, there is a pre-defined variable `$bodyContent`
- The type of `$bodyContent` is `org.apache.velocity.runtime.directive.Block$Reference`
- We will look at these variables in Lesson 4 and Lesson 5

## A Variable without a Value?

- When using a `#set` directive to create a variable, the RHS may be evaluated to `null`
- For example, we saw code of the following type in Lesson 2:

```
#set( $group = $contentRoot.getChild( "brick-group" ) )
```

- If there is no block attached to the relevant region, the variable `$contentRoot` is undefined
- Since `$contentRoot` is undefined, the method call of `getChild` is also undefined
- Velocity will skip any assignment if the RHS is undefined
- Since no value is assigned to the variable `$group`, assuming that the variable is introduced here for the first time, it is undefined
- When a variable, which is undefined, is output, it is output as a String
- Important distinction between an undefined variable and the assignment of `null`:

- A variable starts with a `null` value before it is assigned a non- `null` value
- Velocity will skip an assignment when the RHS is evaluated to `null`
- When a `#set` statement is used to create a variable, and if the RHS is evaluated to `null`, no value will be assigned to the new variable, and its initial value remains to be `null`
- This may give the impression that the `#set` can assign `null` to a variable
- It can be proved otherwise if the variable already stores a non- `null` value; that is, the `#set` directive is NOT used to create a variable, but to reassign a value to an already existing variable:

  ```
  #set( $var = "Hello" )
  #set( $var = $contentRoot.getChild( "brick-group" ) )
  $var
  ```

  Assuming that `$contentRoot` is undefined, the last line will output the String `Hello`, not `$var`.

- A statement like `#set( $var = null )` will cause an error
- If we want the Velocity engine to ignore an undefined variable, instead of outputting something like `$var` (as a String), we can use the quiet reference notation to output the empty String:

  ```
  #set( $var = $contentRoot.getChild( "brick-group" ) )
  $!var
  ```

  The quiet reference notation is an additional exclamation mark between the dollar sign and the identifier of a variable.

---

### The Significance of `null`

---

- The keyword `null` can only appear in two contexts:
  - `$_PropertyTool.isNull( null )` returns `true`
  - `null` can be an argument in a Java method invocation (see the example above)
- The String `null` can be used as the name of a variable; i.e., `$null` is a valid variable
- Otherwise, `null` cannot appear anywhere in Velocity:
  - `null` cannot occur as a standalone statement
  - `$_PropertyTool.isNull( null.Class.Name )` causes an error
  - `#set( $var = null )` causes an error

  - `#if( null )` causes an error
  - `null` cannot be used as a list in `#foreach`
  - `null` cannot be used as an argument in an invocation of a macro

- An undefined variable like `$var` is associated with the `null` value; yet the variable can appear anywhere:
  - When an undefined variable is assigned to another variable, the assignment fails but will not cause an error
  - In a boolean context, an undefined variable is evaluated to `false`
  - In a looping structure, an undefined variable used as a list will fail but will not cause an error
  - An undefined variable can be passed into a macro invocation without causing an error
  - `$_PropertyTool.outputProperties( $object )` outputs `The object is null`
  - Conclusion: an undefined variable is not equivalent to `null`

---

## Scopes of a Variable

---

- The scope of a variable is the portion of your code where this variable is visible
- When a variable is associated with a value, and when the variable is in scope, we can retrieve the value of the variable by referencing the variable
- There are two types of scopes:
  - Global
  - Local

- A global variable is a variable whose value can be accessed anywhere in a format, with some minor exceptions
- A local variable is a variable whose scope is defined by a block of code; normally, its scope ends with an `#end` directive

---

## `#set` and Global Variables

---

- The `#set` directive is always associated with global variables
- When a variable is created by the `#set` directive successfully, from that point on (after the execution of the statement), the variable can be accessed anywhere in the format,

with one exception: the global variable can be overshadowed by a local variable of the same name

- A global variable can be accessed inside a macro:

```
#macro( echoGlobalVariable )
    $myVar
#end

#set( $myVar = 3 )

#echoGlobalVariable  ## => 3
```

Here the macro `echoGlobalVariable` is invoked after the creation of the global variable `$myVar`, and the value of the variable is available to the macro

- A variable created inside a macro can be accessed outside the macro:

```
#macro( createGlobalVariable )
    #set( $newVar = 14 )
#end

## before the macro is invoked, $newVar is undefined
$newVar  ## => $newVar

## after the macro is invoked, the variable is defined
#createGlobalVariable
$newVar  ## => 14
```

- If two variables of the same name are created both inside (not using `#set`) and outside of a macro, then the local variable will overshadow the global one within the macro, and the global variable is not accessible:

```
#macro( echoGlobalVariable $myVar ) ## locally created in macro
    $myVar
#end

#set( $myVar = 3 )

$myVar ## => 3
#echoGlobalVariable( 16 )  ## => 16
```

But if no argument is passed into the invocation of `#echoGlobalVariable`, then the global variable becomes visible again:

```
$myVar ## => 3
#echoGlobalVariable  ## => 3
```

- To further complicate matters, a local variable can become global if the local one is overwritten by a global one within a macro by using `#set`:

```
#macro( createEchoGlobalVariable $myVar )
    $myVar
    #set( $myVar = 14 )
    $myVar
#end

#createEchoGlobalVariable( 32 )
$myVar  ## => 14
```

The first occurrence of `$myVar` outputs 32, and the second and third occurrences output 14

- The global variable is accessible again if the local one is not assigned a value:

```
#macro( echoVariable $myVar )
    $myVar
#end

#set( $myVar = 3 )

#echoVariable       ## no argument passed in, the global value then => 3
#echoVariable( 4 )  ## => 4
```

- Lesson learned: Be very careful with variable names when working with macros

---

## `#set` and Local Variables

---

- Since `#set`, with valid assignment, is always associated with global variables, assignment of values to local variables using `#set` is impossible
- Values associated with local variables are always assigned by the Velocity engine
- It is always pointless to assign a value to a local variable without actually turning the local variable into a global variable
- Reconsider the example we saw above:

```
#macro( createEchoGlobalVariable $myVar )
    $myVar
    #set( $myVar = 14 )
    $myVar
#end

#createEchoGlobalVariable( 32 )
$myVar  ## => 14
```

- Before the local variable `$myVar` is turned into a global variable, it stores whatever value passed into the macro invocation
- Once `$myVar` is turned into a global variable, the value stored in the local variable is lost;

in fact, the local variable ceases to exist

- This also means that it is impossible to reassign a value to a global variable within a macro by passing in the global variable as an argument; passing in a variable by reference is impossible

```
#set( $myVar = $_PropertyTool )

#myMacro( $myVar $_ )

#macro( myMacro $var $obj )
    #set( $var = $obj )
#end

$myVar.Class.Name
$var.Class.Name
```

- The intention of this code snippet is to reassign the `$_` object to the variable `$myVar` in the macro
- The variable `$myVar`, when passed into the macro, is not related to the variable `$var` at all within the `#set` statement
- Instead, `$var` becomes a global variable, leaving `$myVar` untouched
- There is no easy way to pass a variable into a macro by reference, hoping to assign a different value to the variable within the macro (See Lesson 2: #define and #evaluate of the intermediate course for more discussion)

---

## Initializing and Reinitializing a Variable

---

- By using the `#set` directive, a new global variable can be created and initialized with a valid value
- If the RHS fails to be evaluated to a valid value, the `#set` statement fails and the variable remains undefined
- We can use `$_PropertyTool.isNull` to test the variable
- However, if we are dealing with a loop, where the variable in question was assigned a valid value in the previous loop, then after the failure of the `#set` statement, the variable still stores the old value from the previous loop
- To fix this problem, it is generally a good idea to assign the empty String to the variable before the second assignment that can go wrong
- After the second assignment, instead of testing for `null`, test for the empty String
- Example:

```
#set( $myVar = "" )
#set( $myVar = $contentType.getChild( "child" ) )

## if $myVar still stores the empty String, then the assignment fails
```

## Global Variables, Parameters, and Java Method Invocations

- When invoking a Java method through a Java object, if the method requires parameters, then the values passed in must be literal values, or global variables storing literal values
- Java method invocations do not allow operations used to compute argument values
- Consider the following example:

  ```
  #set( $str = "Hello" )
  $str.charAt( 4 )
  ```

  Here we output the character 'o'.
- Now if we change `$str.charAt( 4 )` to `$str.charAt( 3 + 1 )`, this change will cause a lexical error
- To perform the computation of the argument value, we have to create a global variable and pass it into the method invocation:

  ```
  #set( $str   = "Hello" )
  #set( $index = 3 + 1 )
  $str.charAt( $index )
  ```

## Variable Interpolation

- Variable Interpolation is the process of evaluating variables within the context of a String, so that values of embedded variables, not variables themselves, are yielded in the String literal
- Variable Interpolation is possible only when variables appear within double quotes defining a String literal
- Consider the following:

```
#set( $var1 = 'one' )
#set( $var2 = 'two' )
#set( $var3 = 'three' )

#set( $result1 = '$var1, $var2, and $var3' )
$result1 ## => $var1, $var2, and $var3
#set( $result2 = "$var1, $var2, and $var3" )
$result2 ## => one, two, and three
```

- The three variables `$var1`, `$var2`, and `$var3` are first created
- Then the variable names are embedded in String literals
- For `$result1`, since single quotes are used, variable interpolation is not allowed, and character sequences like `$var1` are treated as Strings
- For `$result2`, the variable names occur within double quotes, and these names are replaced by corresponding values
- When variable interpolation is possible, curly brackets can be used to separate variable names from other characters: `#set( $result3 = "${var1}self" )`

---

## Examples

- introductory/03_variables

## Challenges

- How do we pass variables around by reference?
- Can we create a variable on the fly, using user input or a String from some source as the variable name?