OOP Concepts

Variable Scope

In Ruby, variables are used to support and contain data of any type. Ruby supports the following five types of variables:

* Local variables
* Instance variables
* Class variables
* Global variables
* Constants

In a case sensitive programming language like Ruby, the variable name itself defines its scope. Now that you know the many types of Ruby variables, let’s see how to use each of them and what the differences are.

Local variables are variables that are only accessible in its block of which it was initialized. Local variable name begins with a lowercase letter or an underscore followed by the characters of the name. Here’s an example:

def examplemethod

mynumber = 20

p mynumber

end

examplemethod #=> 20

p mynumber #=> name error

In the example above, a method, <code>examplemethod</code>, is defined and the local variable, <code>mynumber</code>, is set to <code>20</code>. The method is then called upon and outputs ‘20’ because the defined method contains, <code>p mynumber</code>. <code>p mynumber</code> is now called and outputs a name error because the local variable isn’t accessible outside of the method of which its created.

Instance variables are variables that belong to a particular object instance. These types of variables begin with a <code>@</code> followed by the variable name.

Class variables are variables that are shared among all objects of a class. It’s important to note that only one copy of a class variable exists for that particular class. These types of variables begin with <code>@@</code> followed by the variable name.

Here’s an example of instance and class variables:

class Donuts

@@yum = true

def initialize(type)

@type = type

end

def best

"#{@type} donuts are the best!"

end

def is\_donut?

@@yum

end

end

d1 = Donuts.new('Glazed')

d2 = Donuts.new('Sprinkled')

d3 = Donuts.new('Chocolate')

p d1.best #=> "Glazed donuts are the best!"

p d2.best #=> "Sprinkled donuts are the best!"

p d3.best #=> "Chocolate donuts are the best!"

p d1.is\_donut? #=> true

p d2.is\_donut? #=> true

p d3.is\_donut? #=> true

In the example above, the created class, <code>Donuts</code>, contains one class and one instance variable. <code>@@yum</code> is a class variable which its value is shared by all instances of the class. The constructor method, (<code>initialize</code>), is called when the object, <code>@type</code> is created. This is an instance variable and is specific to a concrete object. The concrete object in this case is a type of donut.

The first three outputs calls the <code>best</code> method which each message is unique to that instance variable. The second three outputs calls the <code>is\_donut?</code> method which outputs the same because each method returns the class variable.

Global variables are variables that can be accessed from anywhere in the program. Global variables begin with <code>$</code> followed by its variable name. Since global variables alter the global status, it is highly discouraged them unless absolutely necessary. Here is an example:

In the example above, the global variable is called upon in the method, <code>print\_coffee</code> and outputs a statement.

Constants are variables work the same as global variables except a warning is generated when a constant is reassigned after its initialization. Constants that are defined within a class or module can be accessed within that class or module. Constants can also be accessed globally by being defined outside a class or module. Constants my not be defined within methods. A constant name begins with an uppercase letter followed by the name (usually a constant name is in all uppercase letters). Here is an example:

In the example above, the constants are defined within the class and hold the same value when a new class, <code>Tea</code>, is created.

Knowing and using these types of Ruby variables allows programmers to be knowledgeable and practical in implementing object-oriented design code that is manageable, extensible, and pleasing.