You are logged in as C. O'Keefe (Logout)

Core Ruby 41st Batch

Home ▶ PORPC101-41C ▶ 26 April - 2 May ▶ Week 4: Tutorial

Week 4: Tutorial

Before you begin

First complete: Weeks 1 to 3.

Week 4

During Weeks 1 to 3 the procedural style of programming (this continues to be used in languages such as C) was used to write our programs. Programming procedurally means you focus on the steps required to complete a task without paying particular attention to how the data is managed.

I shall introduce you to Regular Expressions. Many people find regular expressions difficult to use, difficult to read, un-maintainable, and ultimately counterproductive. You may end up using only a modest number of regular expressions in your Ruby and Rails applications. Becoming a regular expression wizard isn't a prerequisite for Rails programming. *However, it's advisable to learn at least the basics of how regular expressions work.*

Apart from Regular Expressions, this week, we shall be doing three more lessons as we want to understand and absorb what we read about Classes and Objects.

You need to read through the following pages.

- Regular Expressions
- Writing our own Class in Ruby
- Method Missing
- Ruby Procs

While doing so, please make a note of all your doubts, queries, questions, clarifications and after you have completed all the pages, post these on the forum here. There may be some questions that relate to something that has not been mentioned or discussed by me here; you could post the same too.

Some of the **important points to remember** after you have read through the above pages are:

- Regular expressions, though cryptic, is a powerful tool for working with text. Ruby has this feature built-in. It's used for pattern-matching and text processing.
- Many people find regular expressions difficult to use, difficult to read, un-maintainable, and ultimately counterproductive.
- You may end up using only a modest number of regular

7 June - 13 June

My courses

Settings

Course administration

My profile settings

expressions in your Ruby and Rails applications.

- **Becoming a regular expression wizard isn't a prerequisite for Rails programming.**
- It's advisable to learn at least the basics of how regular expressions work.
- A regular expression is simply a way of specifying a pattern of characters to be matched in a string.
- In Ruby, you typically create a regular expression by writing a pattern between slash characters (/pattern/). In Ruby, regular expressions are objects (of type **Regexp**) and can be manipulated as such. // is a regular expression and an instance of the **Regexp** class.
- An object is an entity that serves as a container for data and also controls access to the data. Associated with an object is a set of attributes, which are essentially no more than variables belonging to that object. Also associated with an object is a set of functions that provide an interface to the functionality of the object, called methods.
- Things an object **knows** about itself are called instance variables. They represent an object's state (the data - for example, the quantity and the product id), and can have unique values for each object of that type.
- Things an object can **do** are called methods.
- An object is a combination of state and methods that use the state.

- A class is used to construct an object. A class is a blueprint for an object.
- More than 30 built-in classes are predefined in the Ruby class hierarchy. The following class hierarchy is important,
- In Ruby, everything from an integer to a string is considered to be an object. And each object has built in 'methods' (Ruby term for functions) which can be used to do various useful things. To use a method, you need to put a dot after the object, and then append the method name. Some methods such as **puts** and **gets** are available everywhere and don't need to be associated with a specific object. Technically speaking, these methods are provided by Ruby's **Kernel** module (more on this later) and they are included in all Ruby objects (the **Kernel** module is included by class **Object**, so its methods are available in every Ruby

object). When you run a Ruby application, an object called **main** of class **Object** is automatically created. This object provides access to the **Kernel** methods.

- Ruby integers are objects of class **Fixnum** or **Bignum**. The **Fixnum** and **Bignum** classes represent integers of differing sizes. Both classes descend from **Integer** (and therefore **Numeric**). The floating-point numbers are objects of class **Float**, corresponding to the native architecture's double data type.
- A new class is defined typically using **class** Name ... **end**
- Classes in Ruby are first-class objects - each is an instance of class **Class**.
- **MUG THIS UP** **Class** *is an object*, and **Object** *is a class*." Hal Fulton
- When a new class is defined say Name, an object of type **Class** is created and assigned to a constant (Name. in this case). When **Name.new** is called to create a new object, the **new** class method in **Class** is run by default, which in turn invokes **allocate** to allocate memory for the object, before finally calling the new object's **initialize** method. The constructing and initializing phases of an object are separate and both can be over-ridden. The initialization is done via the **initialize** instance method while the construction is done via the **new** class method. **initialize** is not a constructor!
- Objects are created on the heap.
- In the statement:
  d = Dog.new('Labrador', 'Benzy')
  The variable d is known as a reference variable. It does not hold the object itself, but it holds something like a pointer or an address of the object. You use the dot operator (.) on a reference variable to say, "use the thing before the dot to get me the thing after the dot." For example:
  d.bark
- As soon as an object comes into existence, it already responds to a number of messages. Every object is "born" with certain innate abilities. To see a list of innate methods, you can call the **methods** method:
  puts d.methods
  The result is a list of all the messages (methods) this newly minted object comes bundled with. Amongst these many methods, the methods **object_id** and **respond_to?** are important.
- Every object in Ruby has a unique id number associated with it that can be found by the method **object_id**.
- You can determine in advance (before you ask the object to do something) whether the object knows how to handle the message you want to send it, by using the **respond_to?** method.
- You can ask any object of which class it's a member by using its **Object.class** method.
- **instance_of?** returns true if object is an instance of the given class.
- Literal 'Constructors' means you can use special notation, instead of a call to new, to create a new object of that class. Look at the example given for String. Symbol, Array, Hash, Range, Regexp
- Garbage Collection (GC): The Ruby object heap allocates a minimum of 8 megabytes. Ruby's GC is called mark-and-sweep. The "mark" stage checks objects to see if they are still in use. If an

object is in a variable that can still be used in the current scope, the object (and any object inside that object) is marked for keeping. If the variable is long gone, off in another method, the object isn't marked. The "sweep" stage then frees objects which haven't been marked. Ruby uses a conservative mark-and-sweep GC mechanism. There is no guarantee that an object will undergo garbage collection before the program terminates.

- Variables are used to hold references to objects. Variables themselves have no type, nor are they objects themselves.
- **method_missing** gives you a way to intercept unanswerable messages and handle them gracefully.
- Blocks are not objects, but they can be converted into objects of class **Proc**. This can be done by calling the **lambda** method of the module **Kernel**.
- Remember you cannot pass methods into other methods (but you can pass procs into methods), and methods cannot return other methods (but they can return procs).

Exercises: Please complete the Week 4 exercises and discuss in the Week 4 forum.

Quiz: Please take the quiz after you have completed Week 4. All the questions are either of multiple-choice and/or true/false and based on what you have learned in Week 4. You have only 2 attempts to complete the quiz. Grades will be allotted for the quiz.

Last modified: Friday, 15 June 2012, 7:23 PM