Core Ruby 41st Batch

Home ▶ PORPC101-41C ▶ 5 April - 11 April ▶ Week 1: Tutorial

Navigation

Home

- My home

  Site pages

  My profile

  Current course

  PORPC101-41C

    Participants

    General

    5 April - 11 April

    📄 Read: The Philosophy of Ruby

    📄 Ruby Coding Style / Convention

    📄 How to use IRB

    📄 Course Ruby Programs

    📄 **Week 1: Tutorial**

    📄 Some useful Ruby methods

    📄 Week 1: Exercises

    💬 Week 1: Forum

    📄 *PARALLEL TRACK* Start using Git and GitHub

    💬 Git/GitHub Discussions

    ✅ Week 1: Quiz

    12 April - 18 April

    19 April - 25 April

Week 1: Tutorial

Welcome to the Online Ruby Programming Course.
The response to this course has been over-whelming with participants from all across the globe. Initially the pace of the course would be slow but would pick up as the weeks go by. Without further ado, let's get cracking.

Before we begin:
Have you read and understood the following three pages?
Read This First
Course FAQ
RubyLearning FAQ

Let's Start:
Ruby History and Philosophy
Yukihiro Matsumoto, commonly known as 'Matz' created the Ruby language in 1993. An interview with him in 2001, talks about the *history* of Ruby.

a. Please read through the following pages, in the given order. While doing so, <u>please make a note</u> of all your doubts, queries, questions, clarifications and after you have completed all the pages, post these on the Lesson 1: 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.

You need to read the following page - Introduction, to understand the conventions being used in the topics mentioned below:

- Installation
- First Ruby program
- Features of Ruby
- Numbers in Ruby
- String fundamentals
- Variables and Assignment
- Scope

Settings

Course administration

My profile settings

○ Getting Input
○ Ruby Names
○ More on Ruby Methods
○ Writing own Ruby methods

Note: All the programs in the above lessons work with the development release of Ruby 1.9

b. Some of the **important points to remember** <u>after you have read through the above pages are</u>:

○ We are discussing Ruby 1.9 on a Windows platform. *This course is appropriate for Linux/Mac users as well.*
○ Ruby is an interpreted language
○ In Ruby, there's always more than one way to solve a given problem.
○ The code examples would be run in the SciTE editor and ensure that you have done the relevant settings as mentioned on the page "First Ruby program".
○ Code layout is pretty much up to you; indentation is not significant (but using two-character indentation will make you friends in the community if you plan on distributing your code).
○ By *convention*, Ruby source files have the **.rb** file extension. In Microsoft Windows, Ruby source files sometimes end with .rbw, as in **myscript.rbw**.
○ In Ruby, program execution proceeds in general from top to bottom.
○ Features: Free format, Case sensitive, Two type of comments, Statement delimiters are not required, Around 41 Keywords, and all Ruby keywords are written using ASCII characters, and all operators and other punctuation are drawn from the ASCII character set.
○ You may be used to thinking that a false value may be represented as a zero, a null string, a null character, or various other things. But in Ruby, all of these are true; in fact, everything is true except the reserved words **false** and **nil**.
○ We shall be referring to the documentation here - http://www.ruby-doc.org/core-1.9.3/
○ **puts** (s in puts stands for string; puts really means put string) simply writes onto the screen whatever comes after it, but then it also automatically goes to the next line.
○ Parentheses are usually optional with a method call. These calls are all valid:
foobar

```
foobar()
foobar(a, b, c)
foobar a, b, c
```

- In Ruby, numbers without decimal points are called integers, and numbers with decimal points are usually called floating-point numbers or, more simply, floats (you must place at least one digit before the decimal point).
- **Note**: The **Fixnum** and **Bignum** classes represent integers of differing sizes. Both classes descend from **Integer** (and therefore **Numeric**). Ruby is able to deal with extremely large numbers, and unlike many other programming languages, there are no inconvenient limits. Ruby does this with different classes, one called **Fixnum** (default) that represents easily managed smaller numbers, and another, aptly called **Bignum**, that represents "big" numbers Ruby needs to manage internally. Ruby will handle Bignums and Fixnums for you, and you can perform arithmetic and other operations without any problems. Results might vary depending on your system's architecture, but as these changes are handled entirely by Ruby, there's no need to worry.
- Some very common Ruby operators: + addition; - subtraction; * multiplication; / division
- The increment and decrement operators (++ and --) are not available in Ruby, neither in "pre" nor "post" forms.
- Anything inside brackets is calculated first (or, more technically, given higher precedence).
- Observe how the modulus operator (%) works in Ruby.
- When you do arithmetic with integers, you'll get integer answers.
- String literals are sequences of characters between single or double quotation marks.
- In Ruby, strings are mutable. They can expand as needed, without using much time and memory.
- String concatenation is joining of two strings, using the + operator.
- The operator << is used to append to a string
- Escape sequence is the \ character. Examples: \", \\, \n
- '' is an empty string.
- If you get a compilation error like -
  #<TypeError: can't convert Fixnum into String>
  it means that you can't really add a number to a string, or multiply a string by another string.

- Constants begin with capital letters. Example PI, Length
- A variable springs into existence as soon as the interpreter sees an assignment to that variable. It is a good practice to assign nil to a variable initially.
- Use whitespace around the assignment operator:

```
foo = 1
not:
foo=1
```

- Use one initialization per line:

```
level = 0
size = 0
is preferred over:
level = size = 0
```

- x, y = y, x will interchange the values of x and y.
- Local variables must start with either a lowercase letter or the underscore character (_), and they must consist entirely of letters, numbers, and underscores.Examples: india, _usa, some_var
- .to_i, .to_f, .to_s are used to convert to an integer, float, string respectively.
- Avoid using Global scope and Global variables. Global scope means scope that covers the entire program. Global variables are available from everywhere within an application, including inside classes or objects. Global variables are distinguished by starting with a dollar-sign ($) character. The Ruby interpreter starts up with a fairly large number of global variables already initialized. Global variables don't mesh well with the ideals of object-oriented programming, as once you start using global variables across an application, your code is likely to become dependent on them. Because the ability to separate blocks of logic from one another is a useful aspect of object-oriented programming, global variables are not favored.
- **gets** (get a string) and **chomp** (a string method) are used to accept input from a user.
- **gets** returns a string and a '\n' character, while **chomp** removes this '\n'.
- **STDOUT** is a global constant which is the actual standard output stream for the program. **flush** flushes any buffered data within io to the underlying operating system (note that this is Ruby internal buffering only; the OS may buffer the data as well). The usage is not mandatory but recommended.

- To format the output to say 2 decimal places, we can use the Kernel's **format** method.
- Ruby Names are used to refer to constants, variables, methods, classes, and modules. *The first character of a name helps Ruby to distinguish its intended use*.
- Lowercase letter means the characters "a" though "z", as well as "_", the underscore. Uppercase letter means "A" though "Z," and digit means "0" through "9."
- A name is an uppercase letter, lowercase letter, or an underscore, followed by Name characters: This is any combination of upper- and lowercase letters, underscore and digits.
- You can use variables in your Ruby programs without any declarations. Variable name itself denotes its scope (local, global, instance, etc.).
- **REMEMBER** the way local, instance, class and global variables, constants and method names are declared.
- "?", "!" and "=" are the only weird characters allowed as method name suffixes.
- The Ruby convention is to use underscores to separate words in a multi-word method or variable names. For Class names, module names and constants the convention is to use capitalization, rather than underscores, to distinguish the start of words within the name. Examples: my_variable, MyModule, MyClass, MyConstant.
- Any given variable can at different times hold references to objects of many different types.
- Variables in Ruby act as "references" to objects, which undergo automatic garbage collection.
- For the time being, remember that Ruby is dynamically typed and that in Ruby, everything you manipulate is an object and the results of those manipulations are themselves objects.
- The basic types in Ruby are **Numeric** (subtypes include **Fixnum**, **Integer**, and **Float**), **String**, **Array**, **Hash**, **Object**, **Symbol**, **Range**, and **Regexp**.
- For the time being, remember that you can always see what object you are in (current object) by using the special variable **self**.
- We use **def** and **end** to declare a method. Parameters are simply a list of local variable names in parentheses.
- We do not declare the return type; a method returns the value of the last line.
- It is recommended that you leave a single blank line between each method definition.
- As per the Ruby convention, methods need parenthesis around their parameters.
- Methods that act as queries are often named with a trailing ?
- Methods that are "dangerous," or modify the receiver, might

be named with a trailing ! (Bang methods)

- Ruby lets you specify default values for a method's arguments-values that will be used if the caller doesn't pass them explicitly. You do this using the assignment operator.
- For now remember that there is an interpolation operator #{...}
- **alias** creates a new name that refers to an existing method. When a method is aliased, the new name refers to a copy of the *original* method's body. If the method is subsequently redefined, the aliased name will still invoke the original implementation.
- In Ruby, we can write methods that can accept variable number of parameters.
- There's no limit to the number of parameters one can pass to a method.
- The sequence in which the parameters are put on to the stack are left to right.
- Whether Ruby passes parameters by value or reference is very debatable - *it does not matter*.

c. Exercises:

Please complete the Week 1 exercises and discuss the same in the Week 1: Forum.

d. Quiz:

Please take the quiz after you have completed Week 1 lessons. All the questions are either of multiple-choice and/or true/false and based on what you have learned in Week 1. You have only 2 attempts to complete the same. Grades will be allotted for the quiz.

e. Some Ruby methods:

This week, we will take a look at four Ruby methods - explanation and a simple example. Don't forget to check this out.

Last modified: Sunday, 20 January 2013, 1:07 AM

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