* Ruby
  + Cross platform,interpreted and object-oriented language
  + Matz created Ruby
  + Ruby source file has .rb extension
  + File name is in lower case of class or module name
  + There are no special method from which execution begins
* p001hello.rb
* puts ‘Hello’
  + puts: prints on screen and moves to the next line
  + puts return nil
  + Parenthesis are optional in method call
  + Everything in ruby is an object
  + to use a method use dot operator after object
  + When we run Ruby application,an object to main is created, this provides access to Kernel method
* Features of Ruby :
  + Free Format :Start writing program from any line
  + Case Senstive : Loawer and uppercase letter are distinct
  + Commented :
    - #
    - =begin and =end
  + Statement Delimiter :
    - Multiple statement in one line is separated by semicolon
  + Keywords:
    - Reserved words cannot be used for other purpose
    - All are true in ruby except false and nil
    - prefix keyword with @,@@,$ and use them as instance,class or global variable name.
* Numbers in Ruby:
  + Numbers without decimal points are Integers
  + Numbers with decimal point are floatingpoint numbers
  + Integer literal are sequence of digit
  + Underscore can be use as thousand separator
* p002rubynumbers
* puts 1 + 2
* puts 2\*3
* puts 3 /2
* puts 10 -11
* puts 1.5 /2.6
  + Increament operator is not available in Ruby
  + Bracket parenthesis is evaluated first
  + Check marked operator is a kind of syntactic sugar :something looks like operator but is a method call
* Modulous Operator (%) : sign of the result is always same as the sign of the second operand
* Similarities and Difference between || and or
  + Similarities :
    - First argument is returned always, if first argument is fasle then second argument Is printed
  + Differences :
    - || has higher precedence than or
* Use of || :
  + assign a value to variable if it is not already set

@variable=@variable || “default value”

Strings:

* + Sequence of character between single and double quotes
* p003rubystrings.rb
* puts “Hello World”
* puts ‘Hello World’
* puts ‘I like’ + ‘Ruby’
* puts ‘It\’s my Ruby’
* puts ‘Hello’\*3
* PI=3.142
* puts PI
  + if puts is passed an object, then puts calls to\_s method of that object , prints the string returned by that method
  + Strings are mutable
  + back-tick or Grave accent :Command is send to the operating system as to be executed, the output of the command is printed on the screen
  + system : separate way to spawn process using Kernel system
* Variable and Assignment:
  + Assignment : storing a value
  + variable : String used to store the assigned value
  + bareword : combination of letters, numbers and underscores and not any symbol
  + Local variable : must start with underscore or lowercase letter
  + Method Calls can also be bareword
  + When ruby sees a bare it interprets 3 things :
    - if equal sign is present on right handside : then it is a local variable
    - bareword could be a keyword
    - bareword is assumed to be a method call
* Scope :
  + Visibility of variable
  + Global scope and global variable:
    - global scope : that covers entire program
    - global scope is for global variable
    - global variable : starts with $ sign
    - available everywhere in the program
    - global variable never go out of scope
  + Built in global variable :
    - variables that already initialized.
    - they are available anywhere in the program
    - $0 : name of the file Ruby is executing
    - $: contains the directories, that make up the path Ruby searched when ecxternal file is loaded
    - $$ : process if of the Ruby process
  + Local Scope:
    - top level (outside all definition block)
    - every class or module has its own scope
    - every method has its own scope
* Input:
  + gets and chomp are useful for getting the input
  + STDOUT is global constant: actual standard output stream for the program
* p005methods.rb
* puts “In which city do you stay?”
* STDOUT.flush
* city=gets.chomp
* puts “The city is ”+ city
  + flush :flushes any buffered data
  + gets:
    - accepts single line of data from standard input.
  + chomps: removes new line character
* Names :
  + Names are used to refer constants,variables,method,classes and module
  + first character helps in distinguishing its use
  + Variables:
    - can contain any datatype
    - we can use variables without declaration
    - variable name denotes its scope
  + Local variable:
    - declared within an object
    - lowercase letters or underscore

ex :sunil,\_z,hit\_and\_run

* + Instance variable :
    - declared within an object
    - belongs to whatever object self refers to
    - starts with @sign

ex :@sign,@\_,@Counter

* + class variable:
    - starts with @@
    - sharing among all objects.
    - Only copy exists for each class
    - rarely used in class program
    - global : starts with $ followed by name character.
  + Constants :
    - starts with an upper case letter followed by underscore
    - class name and module names are constants
  + Method Name :
    - should begin with lower case letter
    - ?,!,= allowed in method suffix
* Basic Types :
  + Numeric (subtype : Fixnum , Integer and Float)
  + String
  + Array
  + Hash
  + Object: Class object has method called class, that returns the class of an object
  + Symbol
  + Range
  + RegExp
* Method
  + use def and end to declare a method
  + No return type
  + Ruby returns the value of last statement executed
  + Paranthesis for method name is optional
* p008mymethods.rb
* def hello
* ‘Hello’
* end
* puts hello
* def hello1(name)
* ‘hello’+name
* end
* puts(hello1(‘sahana’))
* def hello2 name2
* ‘Hello’+name2
* end
* puts (hello2 ‘Sachin’)
* Interpolation Operator: #{…}
  + gets calculated separately and the result of calculation are pasted automatically into the string.
  + This is a process ofinserting the result of an expression into a string literal.
* Aliasing a method
* p010aliasmtd.rb
* def oldmtd
* “old method”
* end
* alias newmtd oldmtd
* def oldmtd
* “ old improved method”
* end
* puts oldmtd
* puts newmtd
* alias creates a new name that refers to an existing method, operator, global variable or regular exoreesion backreference ($&,$`,$’ and $+).
* alias cannot be used for local variables, instance variables, class variables and constants
* Variable number of parameter in Ruby can be accepted by using astreik
* Bang Method : (!)
  + Method ends with ! symbol
  + This does exactly opposite of method without !
  + A method without a bang, performs operation and creates anew object
  + But method with ! , modifies the existing object .
  + Some example of bang method

ex: sor/sort!,uppercase/upcase! and chomp/chomp!

* Method ending with ?
  + No special meaning in interpreter.
  + returns a value that answers the question
  + Mostly these method return true or false
* Strings:
  + Methods of String
    - reverse: backward version of String
    - length : gives length of the string
    - upcase: changes to uppercase
    - downcase: changes to lowercase
    - swapcase: switches cases of letter
    - capitalize: convert all character in string to lowcase except the first one
    - slice : gives substring of larger string
* Listing all methods of a class or object
  + String.methods.sort :
    - List of methods that the class object responds to
  + String.instance\_methods.sort
    - Method tells you all the instance method that instances of String are endowed with
  + String.instance\_methos(fasle).sort
    - Class instance metods without those of class’s ancestors
* Comparing two String:
  + == : test to check if the content is identical
  + String.equal? : test whether two strings are the same object.
* Using %w to create array:
  + Helps in creating array.
* Character Set:
  + Character set of character symbol
  + each has a numeric id : character code point
* Constructs
  + if …else… end
  + while
  + unless
    - executes only if the expression evaluates to fasle
  + exit: terminates program
  + ?: Ternary operator
  + Statement modifier :
    - simple shortcut, if body of if or while is one line
  + Case Expression:
  + nil is an Object
* Blocks:
  + chunk of code between braces or between do-end
  + can be associated with method invocation
  + Block is way of grouping statement.
  + block is written from the same line as method call
  + code in block is not executed when it is encountered, Instead, Ruby remembers the context in which block appears.
  + For single line block use braces and for multiline block use do-end
  + braces has high precendence than do
  + Any method with a block can be called as an implicit argument.
  + Inside block you can block using yield keyword
  + When we call amethod, then the method can yield control to that code block suspend execution of that method, execute the code in that block and return control to the method body.
  + block\_given?returns true if the block is executed in the current context

Arrays

* + list of items in order
  + every element in array is a variable
  + array are created using square bracket
  + size and length : gives length of the array
  + size-1 : last element of an array
  + retuns nil:
    - if we try to access element beyond last element
    - Element before the first element
  + Arrays are mutable : resizable
* Parallel Assignment:
  + Consider rvalue and lvalue
  + rvalue appears on the right nhand side
  + lvalue appears on left hand side
  + values in rvalues are evaluated first and stored in the array
  + then lhs is evaluated and the values are assigned to variables
  + If only one variable on lhs: then , rhs is an array
  + If more variables are on RHS, then assignment is done
* Environment Variables:
  + link between our program and the outside world
  + can access system environment using predefined variable ENV
* Ranges:
  + Express Sequence
  + start point and end point
  + sequence are reated using .. and …
  + 2 dot operator :inclusive range
  + 3 dot : exclude high value range
* Symbols:
  + looks like a variable name but its prefixed with colon

ex : :action,:line\_item

* + most basic ruby object that can be created
  + name and an internal id
  + symbol refer to same object throughout program
  + more efficient than string
  + When to use string versus Symbol
    - content of object tis important
    - identity of the object is important : use symbol
  + Ruby uses Symbol and symbol table holds them.
  + x=created by prefixing an operator,string, variable ,constant,method ,,class, module name with a colon
  + Symbols are useful when creating hashes and we want to have distinction between keys and values
* Hashes:
  + index hash with object
  + Collection of key value pair
  + retrieve value using key
  + hashes uses object as an index
  + hash has a default value, it is returned when access is made to key which does not exist :nil

ex:

h={ ‘dog’ => ‘canine’ ‘cat’=>’feline’ , ‘donkey’ => ‘asinine’,

12 => ‘dodecine’}

Symbols

h ={nickname: ‘IndianGuru’, language: ‘Marathi’ , lastname: ‘Talim’}

* Random Numbers:
  + method to choose number randomly is rand
* Read/Write Files:
  + File.open :opens file in different mode like r,r+,w,w+
  + File implements readline that reads an entrie file into an array, line by line
  + Traversing Directory Trees
    - Find supports top down traversal
* Regular Expression:
  + pattern matching and text processing
  + Regular Expression is simply a way of specifying a pattern of character to be matched in a string
  + Create regular expression by writing pattern between slash
  + Regular expression are object of Regexp
  + / : delimits the pattern
  + match pattern with match method
  + operator for match is =~
  + Literal character:
    - puts in regular expression matches itself
    - TO match special character use backslash
  + Wildcard character : .(dot)
    - used to match character at some point in pattern
  + Character class :
    - explicit list of character matching is done using square bracket
    - TO negate the match use ^
  + Special escape sequence for common character
    - to match digit /[0-9]/
    - to match digit use /\d/
    - \w : matches any digit,alphabet or underscore
    - \s matches any white space (tab, newine and space)
* Objects
  + Methods associated with an object tis obtained by called method function
  + Object has a unique id associated with : object\_id
  + respond\_to? : ASK an object whether it responds to any message
  + instance\_of? returns true if object is an instacnce of that class
* Garbage Collection: GC
  + if an object tis assigned nil, then it is eligible for Garbage Collection.
  + GC is Ruby is mark and sweep
    - mark stage:
      * checks if objects are still in use: if yes then that object is marked keep
    - sweep stage
      * if object is not used , then it is not marked.
      * Then free these objects which are not marked
* Ruby Proc
  + Blocks are not object, but thay can be converted to object of class Proc
  + To convert Blocks to object using lambdamethod of an class object
  + block created by lambda acts like Ruby
  + TO call block which is made object using Lambda we use call
  + We cannot pass method into another method : But proc can be passed to a method
  + Method cannot return another method : we can return a proc
* Including Other Files in Ruby :
  + require and load method
  + load method:
    - Ruby source file every time
  + require method
    - loads any given file only once
    - require(string) => true or false
      * if loading of file is successful , it returrns true
      * If filename does not exist, it searches in $: directory
* Inheritence
  + Class can only inherit from a single other class
  + Multiple inheritance is not possible
  + Inheritence are indicated using <
* Inheritence and Instance Variables: using super
  + When we invoke super with arguments,it calls method in parent class
* Usage of Super:
  + When you invoke **super** with no arguments Ruby sends a message to the parent of the current object, asking it to invoke a method of the same name as the method invoking **super**. It automatically forwards the arguments that were passed to the method from which it's called.
  + Called with an empty argument list - **super()**-it sends no arguments to the higher-up method, even if arguments were passed to the current method.
  + Called with specific arguments - **super(a, b, c)** - it sends exactly those arguments.
* Abstract Class:
  + Class that invokes certain undefined methods
* Exception :
  + Represent exceptional condition
  + Block of code is executed when an exception occurs
  + *Raising* an exception means stopping normal execution of the program and transferring the flow-of-control to the exception handling code where you either deal with the problem that's been encountered or exit the program completely
  + dealing with it or aborting the program - depends on whether you have provided a **rescue** clause
  + most of the subclasses extend a class known as **StandardError**
* Handling Exception :
  + Code that could raise an exception in a begin-end block
  + rescue clauses to tell Ruby
  + body of method definition : begin-end

1. We are discussing Ruby 1.9 on the Windows platform. This course is appropriate for Linux/Mac users as well.
2. Ruby is an interpreted language
3. In Ruby, there's always more than one way to solve a given problem.
4. 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).
5. By convention, Ruby source files have the **.rb** file extension. In Microsoft Windows, Ruby source files sometimes end with .rbw, as in myscript.rbw.
6. In Ruby, program execution proceeds in general from top to bottom.
7. Features: Free format, Case sensitive, Two type of comments, Statement delimiters are not required, We have around 41 Keywords.
8. 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 \*values\* are true; in fact, everything is true except the reserved words false and nil.
9. We shall be referring to the documentation [**here**](http://www.ruby-doc.org/core-1.9.3/).
10. **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.
11. Parentheses are usually optional with a method call. These calls are all valid:  
    foobar  
    foobar()  
    foobar(a, b, c)  
    foobar a, b, c
12. 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).
13. **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.
14. Some very common Ruby operators:+ addition; - subtraction; \* multiplication; / division
15. The increment and decrement operators (++ and --) are not available in Ruby, neither in "pre" nor "post" forms.
16. Anything inside brackets is calculated first (or, more technically, given higher precedence).
17. Observe how the modulus operator (%) works in Ruby.
18. When you do arithmetic with integers, you'll get integer answers.
19. String literals are sequences of characters between single or double quotation marks.
20. In Ruby, strings are mutable. They can expand as needed, without using much time and memory.
21. String concatenation is joining of two strings, using the + operator.
22. The operator << is used to append to a string
23. Escape sequence is the \ character. Examples: \", \\, \n
24. '' (single quotes) or "" is an empty string.
25. 24. If you get a compilation error like - #<TypeError: cannot convert Fixnum into String> it means that you cannot really add a number to a string, or multiply a string by another string.
26. 25. Constants begin with capital letters. Example PI, Length
27. 26. A local 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 local variable initially. A runtime error will be generated if the local variable is used without being assigned a value.
28. 27. Use whitespace around the assignment operator:  
    foo = 1  
      
    not:  
      
    foo=1
29. 28. Use one initialization per line:  
      
    level = 0  
    size = 0  
      
    is preferred over:  
      
    level = size = 0
30. 29. x, y = y, x will interchange the values of x and y. Parallel assignment is any assignment expression that has more than one lvalue, more than one rvalue, or both. Multiple lvalues and multiple rvalues are separated from each other with commas.
31. 30. 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
32. .to\_i, .to\_f, .to\_s are used to convert to an integer, float, string respectively.
33. *31. Avoid* using Global scope and Global Variables. Global scope means scope that covers the entire program. Global variables are distinguished by starting with a dollar-sign ($) character. The Ruby interpreter starts up with a fairly large number of global variables already initialized. Global variables are distinguished by starting with a dollar-sign ($) 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*.
34. **gets** (get a string) and **chomp** (a string method) are used to accept input from a user.
35. **gets** returns a string and a '\n' character, while **chomp** removes this '\n'.
36. **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.
37. To format the output to say 2 decimal places, we can use the **Kernel**'s **format** method.
38. 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.
39. 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".
40. 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.
41. You can use variables in your Ruby programs without any declarations.
42. Variable name itself denotes its scope (local, global, instance, etc.).
43. **REMEMBER** the way local, instance, class and global variables, constants and method names are declared.
44. "?", "!" and "=" are the only weird characters allowed as method name suffixes.
45. The Ruby convention is to use underscores to separate words in a multiword method or variable name. By convention, most constants are written in all uppercase with underscores to separate words, LIKE\_THIS.
46. Ruby class and module names are also constants, but they are conventionally written using initial capital letters and camel case, LikeThis. More examples: MyModule, MyClass, My\_Constant, my\_variable.
47. Any given variable can at different times hold references to objects of many different types.
48. Variables in Ruby act as "references" to objects, which undergo automatic garbage collection.
49. 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.
50. The basic types in Ruby are **Numeric** (subtypes include **Fixnum**, **Integer**, and **Float**), **String**, **Array**,**Hash**, **Object**, **Symbol**, **Range**, and **RegExp**.
51. For the time being, remember that you can always see what object you are in (current object) by using the special variable **self**.
52. We use **def** and **end** to declare a method. Parameters are simply a list of local variable names in parentheses.
53. We do not declare the return type; a method returns the value of the last statement.
54. It is recommended that you leave a single blank line between each method definition.
55. Ruby allows parentheses to be omitted from most method declarations and / or invocations. In simple cases, this results in clean-looking code. In complex cases, however, it causes syntactic ambiguities and confusion.
56. Methods that act as Boolean queries are often named with a trailing ?
57. Methods that are "dangerous," or modify the receiver, might be named with a trailing ! (Bang methods)
58. 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.
59. For now remember that there is an interpolation operator #{...}
60. **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.
61. In Ruby, we can write methods that can accept variable number of parameters.
62. There's no limit to the number of parameters one can pass to a method.
63. The sequence in which the parameters are put on to the stack are left to right.
64. Refer to the **String** documentation to use the various methods available.
65. For double-quoted string literals, Ruby looks for substitutions - sequences that start with a backslash character - and replaces them with some binary value or does expression interpolation ie. within the string, the sequence #{expression} is replaced by the value of the expression.
66. It is to be noted that every time a string literal is used in an assignment or as a parameter, a new **String** object is created.
67. Observe how one can list all the methods of a class or object.
68. Comparing two strings for equality can be done by **==** or**.eql?** (for identical content) and **.equal?** (for identical objects).
69. **%w** is a common usage in strings.
70. Observe the usage of constructs: **if else end**, **while**, **if elsif end**
71. Ruby also has a negated form of the **if** statement, the **unless end**.
72. Case Expressions: This form is fairly close to a series of **if** statements: it lets you list a series of conditions and execute a statement corresponding to the first one that's true. **case** returns the value of the last expression executed. Usage: **case when else end**
73. **IMPORTANT**: Ruby Code blocks are chunks of code between braces or between do- end that you can associate with method invocations.
74. Code blocks may appear only in the source adjacent to a method call; the block is written starting on the same line as the method call's last parameter (or the closing parenthesis of the parameter list). The code in the block is not executed at the time it is encountered. Instead, Ruby remembers the context in which the block appears (the local variables, the current object, and so on) and then enters the method.
75. The Ruby standard is to use braces for single-line blocks and do- end for multi-line blocks. Keep in mind that the braces syntax has a higher precedence than the do..end syntax.
76. Inside a method, you can call a Ruby block using the **yield** keyword with a value.
77. You can provide parameters to the call to **yield**: these will be passed to the block. Within the block, you list the names of the arguments to receive the parameters between vertical bars (|).
78. The **do** and **end** identify a block of code that will be executed for each item.
79. An **Array** is just a list of items in order. Every slot in the list acts like a variable: you can see what object a particular slot points to, and you can make it point to a different object. You can make an array by using square brackets.
80. Arrays are indexed by integers and the index starts from 0.
81. A trailing comma in an array declaration is ignored.
82. You can access an array beyond its boundary limits; it will return **nil**.
83. We can add more elements to an existing array.
84. Refer to the **Array** documentation for a list of methods.
85. The method **each** of an Array calls a *block* once for each element in Array, passing that element as a parameter.
86. The variable inside the "goalposts" (i.e. | |) refers to each item in the array as it goes through the loop. You can give this any name you want.
87. Sequences have a start point, an end point, and a way to produce successive values in the sequence. In Ruby, these sequences are created using the ".." and "..." range operators and are called Range.
88. The two dot form creates an inclusive range, and the three-dot form creates a range that excludes the specified high value.
89. In Ruby, the sequence 1..100000 is held as a **Range** object containing references to two **Fixnum**objects.
90. The **.to\_a** method converts a **Range** to an **Array**.
91. Another use of the versatile range is as an interval test: seeing if some value falls within the interval represented by the range. We do this using ===, the case equality operator.
92. Ranges are not limited to integers or numbers. The beginning and end of a range may be any Ruby object.
93. A symbol looks like a variable name but it's prefixed with a colon.
94. You can think of :id as meaning the name of the variable id, and plain id as meaning the value of the variable.
95. Symbols are useful because a given symbol name refers to the same object throughout a Ruby program.
96. Symbols can be considered constants without values.
97. Symbols are more efficient than strings. Two strings with the same contents are two different objects, but for any given name there is only one **Symbol** object. This can save both time and memory.
98. When do we use a string versus a symbol?
99. If the contents (the sequence of characters) of the object are important, use a string.
100. If the identity of the object is important, use a symbol.
101. A **Symbol** object is created by prefixing an operator, string, variable, constant, method, class, module name with a colon.
102. If Fred is a constant in one context, a method in another, and a class in a third, the Symbol :Fred will be the same object in all three contexts.
103. Hashes are similar to arrays in that they are indexed collection of object references. However, while you index arrays with integers, you can index a hash with objects of any types: strings, regular expressions, and so on.
104. When you store a value in a hash, you actually supply two objects - the index (normally called the key) and the value.
105. **nil** is returned when an attempt is made to access keys that do not exist in the hash.
106. The method to get a randomly chosen number in Ruby is **rand**.
107. If you call **rand**, you'll get a float greater than or equal to 0.0 and less than 1.0. If you give it an integer parameter (by calling rand(5) ), you will get an integer value greater than or equal to 0 and less than 5.
108. The **File.open** method can open a file in different modes like 'r' Read-only, starts at beginning of file (default); 'r+' Read/Write, starts at beginning of file; 'w' Write-only, truncates existing file to zero length or creates a new file for writing.
109. **File.open** opens a new **File** if there is no associated block. If the optional block is given, it will be passed file as an argument, and the file will automatically be closed when the block terminates.
110. Always close a file that you open. In the case of a file open for writing, this is very important and can actually prevent lost data.
111. The **seek** method of class **IO**, seeks to a given offset an Integer (first parameter of method) in the stream according to the value of second parameter in the method. The second parameter can be**IO::SEEK\_CUR** - Seeks to first integer number parameter plus current position; **IO::SEEK\_END** - Seeks to first integer number parameter plus end of stream (you probably want a negative value for first integer number parameter); **IO::SEEK\_SET** - Seeks to the absolute location given by first integer number parameter.
112. 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.
113. Many people find regular expressions difficult to use, difficult to read, un-maintainable, and ultimately counterproductive.
114. You may end up using only a modest number of regular expressions in your Ruby and Rails applications.
115. Becoming a regular expression wizard isn't a prerequisite for Rails programming.
116. It's advisable to learn at least the basics of how regular expressions work.
117. A regular expression is simply a way of specifying a pattern of characters to be matched in a string.
118. 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.
119. 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.
120. 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.
121. Things an object can do are called methods.
122. An object is a combination of state and methods that use the state.
123. A class is used to construct an object. A class is a blueprint for an object.
124. More than 30 built-in classes are predefined in the Ruby class hierarchy. The following [**class hierarchy**](http://rubylearning.com/images/rubyclass.jpg)is important.
125. In Ruby, everything 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.
126. 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.
127. A new class is defined typically using class Name ... end
128. Classes in Ruby are first-class objects - each is an instance of class Class.
129. MUG THIS UP Class is an object, and Object is a class." Hal Fulton
130. 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!
131. Objects are created on the heap.
132. 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
133. 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.
134. Every object in Ruby has a unique id number associated with it that can be found by the method object\_id.
135. 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.
136. You can ask any object of which class it's a member by using its Object.class method.
137. instance\_of? returns true if object is an instance of the given class.
138. 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
139. 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.
140. Variables are used to hold references to objects. Variables themselves have no type, nor are they objects themselves.
141. method\_missing gives you a way to intercept unanswerable messages and handle them gracefully.
142. 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 class **Object**.
143. 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).
144. The **load** method includes the named Ruby source file every time the method is executed.
145. The more commonly used **require** method loads any given file only once.
146. Note that you say require 'filename', not require 'filename.rb'.
147. In Ruby, classes are never closed: you can always add methods to an existing class. This applies to the classes you write as well as the standard, built-in classes. All you have to do is open up a class definition for an existing class, and the new contents you specify will be added to whatever's there.
148. The benefit of inheritance is that classes lower down the hierarchy get the features of those higher up, but can also add specific features of their own.
149. In Ruby, a class can only inherit from a single other class.
150. The methods of the **Object** class are available to all objects unless explicitly overridden or undefined using **undef :method\_name**.
151. Method overriding allows a subclass to provide a specific implementation of a method that is already provided by one of its superclasses. The implementation in the subclass overrides (replaces) the implementation in the superclass.
152. Nothing stops you from defining a method twice, however the new version takes precedence.
153. When you invoke **super** with no arguments Ruby sends a message to the parent of the current object, asking it to invoke a method of the same name as the method invoking **super**. It automatically forwards the arguments that were passed to the method from which it's called.
154. Called with an empty argument list - **super()** - it sends no arguments to the higher-up method, even if arguments were passed to the current method.
155. Called with specific arguments - **super(a, b, c)** - it sends exactly those arguments.
156. A Ruby class can have only one method with a given name.
157. Ruby gives you three levels of protection:
158. **public** - methods can be called by everyone. A class's instance methods are public by default.
159. **protected** - methods can be invoked only by objects of the defining class and its subclasses.
160. **private** - methods cannot be called with an explicit receiver - the receiver is always **self**. This means that private methods can be called only in the context of the current object. The initialize method is always private.
161. Access control is determined dynamically, as the program runs, not statically. You will get an access violation only when the code attempts to execute the restricted method.
162. Top-level methods are private instance methods of the**Object** class.
163. attr\_reader is reader only; attr\_writer is writer only and attr\_accessor is both reader and writer.
164. An exception is a special kind of object, an instance of the class **Exception** or a descendant of that class.
165. The **raise** method is from the **Kernel** module. By default, **raise** creates an exception of the**RuntimeError** class. To raise an exception of a specific class, you can pass in the class name as an argument to **raise**.
166. To do exception handling, we enclose the code that could raise an exception in a **begin-end** block and use one or more **rescue** clauses to tell Ruby the types of exceptions we want to handle.
167. It is to be noted that the body of a method definition is an implicit **begin-end** block; the begin is omitted, and the entire body of the method is subject to exception handling, ending with the end of the method.
168. If you write a **rescue** clause with no parameter list, the parameter defaults to **StandardError**.
169. If you need the guarantee that some processing is done at the end of a block of code, regardless of whether an exception was raised then the **ensure** clause can be used. **ensure** goes after the last **rescue**clause and contains a chunk of code that will always be executed as the block terminates. The **ensure**block will always run.
170. By default, the inspect message, which can be sent to any object, formats the object's ID and instance variables. It returns a string containing a human-readable representation of object. If not overridden, uses the to\_s method to generate the string.
171. When writing rescue clauses, ensure that you rescue from subclassed Exceptions first and then more general Exceptions, otherwise you will not rescue the more specific error. For example, you would rescue based on NoMethodError before NameError, and those before the more general (superclass) StandardError.
172. The **Time** class contains Ruby's interface to the set of time libraries written in C.
173. Time zero for Ruby is the first second GMT of January 1, 1970.
174. Ruby's **DateTime** class is superior to **Time** for astronomical and historical applications, but you can use**Time** for most everyday programs.
175. In Ruby, we rely less on the type (or class) of an object and more on its capabilities. Hence, Duck Typing means an object type is defined by what it can do, not by what it is.
176. Duck Typing refers to the tendency of Ruby to be less concerned with the class of an object and more concerned with what methods can be called on it and what operations can be performed on it.
177. In Ruby, we would use **respond\_to?** or might simply pass an object to a method and know that an exception will be raised if it is used inappropriately.
178. Syntactic sugar refer to special rules that let you write your code in a way that doesn't correspond to the normal rules but that is easier to remember and looks better.
179. Ruby allows you to define methods that end with an equal sign (=)
180. Mutable objects are objects whose state can change. Immutable objects are objects whose state never changes after creation.
181. Mutability is a property of an instance, not of an entire class. Any instance can become immutable by calling **freeze**.
182. The **freeze** method in class **Object** prevents you from changing an object, effectively turning an object into a constant. After we freeze an object, an attempt to modify it results in **RuntimeError**.
183. **freeze** operates on an object reference, not on a variable. This means that any operation resulting in a new object will work.
184. A method **frozen?** tells you whether an object is frozen or not.