Ruby

cross platform

interpreted

object oriented

Ruby is scripting language.

No special method from where execution begins

begins execution from first line and goes upto last line

puts : prints on the console

Parantheses are optionally

In Ruby ,everything from an integer to string is considered to be an object

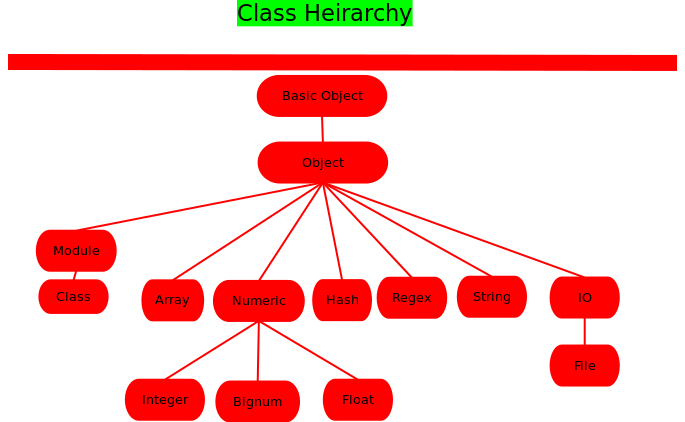
Each object has a method built on it

TO use methods of the object, use dot followed by method name

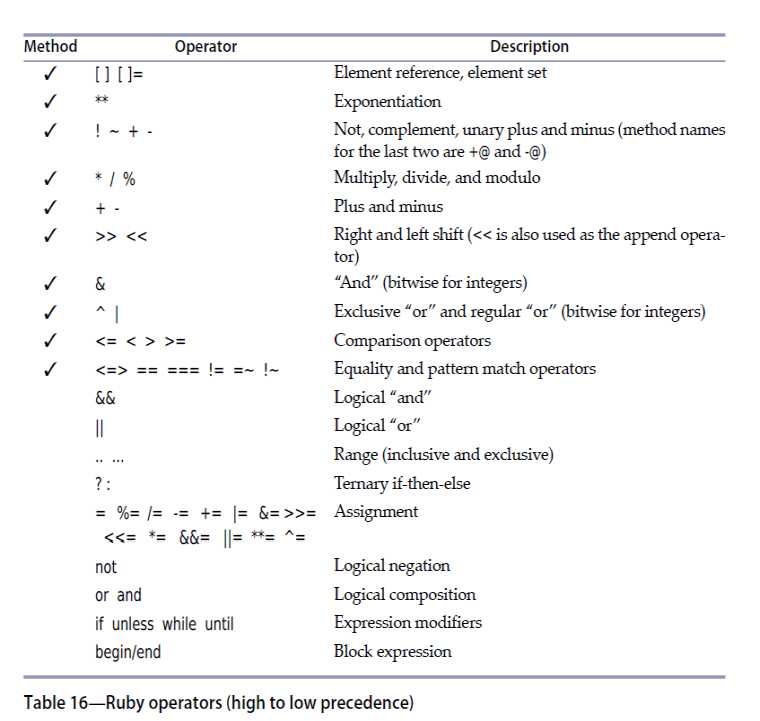
puts and gets need not associate with a specific object

Features of Ruby :

* + Free foramat : Write program from anywhere
  + Case Sensitive :
  + Comments :
    - # : Single Line
    - =begin and =end : Multi line comment
  + Statement Delimiter :
    - Multiple statement in single line must be delimited by semicolon
    - \ : states, line has still not ended
  + Keywords: reserve words cannot be used for other purposes
  + Remember :
    - All values are true
    - Except False and nil
* Numbers
  + Integers : Numbers without decimal
    - Integers are object of class Fixnum or Bignum
  + floating point number : numbers with decimal
    - floating point number is object of FLoat
  + Underscore can be used between for thousands separator
    - ex : 1\_000\_000\_000



Operators and Precedence : highest to Lowest



Increament and Decreament operator (++ and --) are not available in Ruby instead we can use += and -=

* Ruby modulus Operator :the sign of the result (for % operator) is always the same as the sign of the second operand.

Strings:

* + sequence of characters between single nd double quotes
  + If **puts** is passed an object that is not a string, **puts** calls the **to\_s** method of that object and prints the string returned by that method.
  + Strings are **mutable**. They can expand as needed, without using much time and memory. Ruby stores a string as a sequence of characters.
  + To execute command use the **back-tick** .When used with puts, outputs the command output
  + **system** : spawns separate kernel process and run the command.
    - Returns true, if it executes successfully.
    - false is returned, if command return non zero exit status.
    - nil, if command fails execution
* Variables and Assignments
  + To store a number or a string in your computer's memory for use later in your program, you need to give the number or string a name. Programmers often refer to this process as *assignment* and they call the names *variables*
  + variable is combination of numbers, letters, underscore and not any symbol : bareword
* Whenever ruby sees a bareword , it interprets one of these things
  + If there's an equal sign (=) to the right of the bareword, it's a *local variable* undergoing an assignment.
  + Ruby has an internal list of keywords and a bareword could be a *keyword*.
  + If the bareword is not (a) or (b) above, the bareword is assumed to be a *method call*.
* If no method exists, Ruby raises Name error
* Scope of Variables : (global and local variables)
  + Global scope : scope of entire program
    - Only global variable has global scope
    - global variable starts with $
    - Built in global variables
      * $: directories that make the path
      * $$ : process id
  + Local scope :
    - Top level (definition blocks ) has its own local scope
    - Every class has its own local scope
    - Every method definition (def) has its own local scope
* Getting Input
  + puts : writes to the screen
  + gets : gets string
  + STDOUT :
    - global and standard output stream
  + flush : flushes any buffer data of io
  + gets :accepts single of data fom std input
  + chomp : get retruns a string with \n. But chomp removes \n
* Ruby Names :
  + used to Refer constants, variables ,methods and class
  + a-zA-Z0-9,\_
* Variables : (local,global,instance)
  + local variable :declared within object
    - lower letters followed by underscore
  + Instance variable : declared within an object ,belongs to object self refers to
    - Name starts with at sign “@” followed by name
  + class variable : declared within a class
    - starts with two @@ followed by name
    - shared among all objects of a class
    - used at top level are defines as objects and behaves as global variables
  + global variable : entire program
    - starts with $
* Constants
  + starts with uppercase letter
  + class name module name are constants
* Method Names :
  + should begin with lower case
  + ?,!,= allowed as suffixes
* Basic types of Ruby
  + Numeric : Fixnum, Integer and Float
  + String
  + Array
  + Hash
  + Object
  + Symbol
  + Range
  + RegExp
* Object has a method called class that returns the class of an object
* private\_methods : is a method of the Object class
* self :
  + Objects are noun, then methods are verb.
  + Every method needs an object, tell, which object tis running the method
  + To know the object of we are currently in we use self.
* Methods :
  + def and end are used for methods
  + do not declare the return type , returns the last statement executed in method
  + leave blank between two methods

Interpolation :

* + Refers to the process of inserting the result of an epression into a string literal .
  + Way to interpolate is : is to place the expression wtithin #{}
* Aliasing Method
  + Creates a new name that refers existing method
  + When a method is aliased, new name refers to copy of orginal method
  + if the method is subsequently redefined, aliased name will still invoke the original implementation
* Method accepting variable number of parameter :
  + asterisk (\*) : splat argument , is taking in all arguments.
  + These arguments are stored in array.
* Bang(!) Method :
  + Modifies object in place and end in an exclamation mark
  + method labeled ! is dangerous
  + Method with same name but one with bang and other without bang
  + Methods without bang, creats a new object and that reflects a result
  + method with bang, performs action on same object and replace the orginal value
* Method name ending with ?
  + Method has no special meaning to Ruby interpreter
  + Any method ends with? returns a value that answer the question posed by method
* Strings
  + Methods
    - reverse
    - length
    - upcase
    - downcase
    - swapcase
    - slice
  + ! Method
    - upcase!
    - downcase!
    - swapcase!
    - capitalize!
  + single quotes :
    - backslash is not special if character that follows is anything other than quote or backslash
  + double quotes:
    - backslash
      * looks for substitution – sequence that starts with backslash and replace it with binary
      * expression interpolation
* Methods of a class or object
  + String.methods.sort
    - list of methods that Class object String responds to
  + String.instance\_methods.sort
    - instance methods that instance of Strng are endowed with
  + String.instance\_methods(false).sort
    - View a class’s instance methods without those of class’s ancestor
* Comparing two strings for equality
  + == and String.eql? returns same result
  + String.equal? : check if two strings are the same object
* %w :Creating array of words
* Conditions
  + If :
  + elif
  + unless : opposite of if
    - Executes code, only if expression associated with unless is false or nil
    - =begin and =end
    - Conditional ?:
      * (condition) ? (result **if** condition is **true**) : (result **if** condition is **false**)

Ruby Blocks

* + Chunk of code between do –end , this can associate with method invocation
  + May appear only in source adjacent to a method call
  + use braces for single line and multiple line use do and end
  + call the block inside method using yield
  + blocks can have their own arguments
  + block\_given? : check if block is defined with method call
  + Black Variable:
    - local to block
    - block parameters are separated by semicolon : prevents it from clobbering any variables outside its scope

Arrays :

* + list of ordered items
  + every element in the list acts like a variable
  + index of first value is 0
  + size,length : return number of elements in the array
  + last element : size-1
  + negative index: end of arry
  + nil : if we try to read element fromend or beginning
* Command line arguments :
  + special array called ARGV is used to store arguments
  + Append parameters onto the end of command and the program processes them
* Symbols:
  + Variable name prefixed with colon
  + no need of pre declaring and assigning
  + it is just name and internal id
  + given symbol refers to same object
* Hashes:
  + also known as Associate arrays,maps or dictionaries
  + index a hash with objects of any types
  + while storing in hash , we pass 2 valuesx i.e index and the value
  + retrieve with index
  + hashes can use any object as index
  + Attempt to access key that do not exist in hash is nil
* Using Symbols as Hash Keys