RUBY ESSENTIAL

Anum

What is Ruby?

- Programming language
- Created in Japan in 1995 by Yukihiro "Matz" Matsumoto
- Syntax like Perl , python and smalltalk.
- Not a compiler language (like C++, Java, VB) . The complier language is a language whre you write a code and you have to run it through computer program or compiler in order to come out with an application that you can actually run at the end.
- It is interpreted language, requires ruby interpreter

Why Ruby?

- It is object oriented.
- Easily readable code
- Unsurprising syntax , naming, behavior. If you want to sort, it will sort, if you want to find, it will find, reverse, it will reverse and so on...
- Whitespace independent.
- No semicolons
- Lots of "syntactic sugar". It allows to write things in simpler way so that we have some short cut to ourselves.

Ruby and Ruby on Rails

Ruby	Ruby on Rails
It is a multipurpose language	It is a web framework written in ruby
Not just for web but you can make standalone , non internet applications.	

Mac OS – Ruby Installation

- Go to https://www.ruby-lang.org/ - download for mac ------> 1.9.1
- Mac OS 10.1 : may have problems
- Mac OS 10.2 -10.3: install/upgrade ruby
- Mac OS 10.4: ruby 1.8.2
- Mac OS 10.5 :ruby 1.8.6
- Text Editor: writing code , used plain text, Textmate text editor(micromates.com) is very good to used.
- How to open terminal :

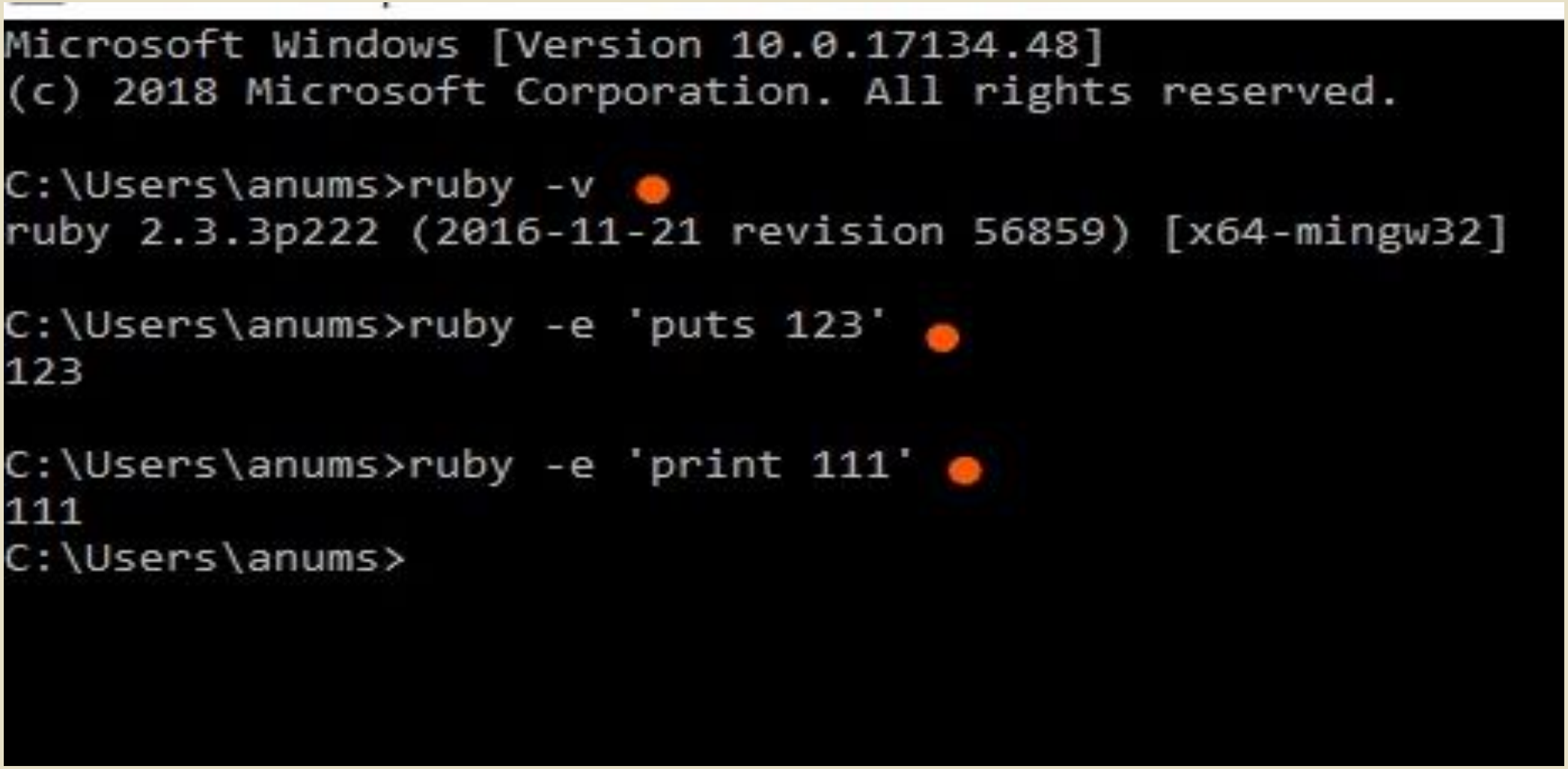
  Application -->utilities -->Terminal.app
- On terminal : to check if ruby install type **ruby –v**
- Type : **which ruby** to know where it is located

Windows OS – Ruby Installation

- https://www.riuby-lang.org/ - download
- Install ruby interpreter : one click installer (currently v1.8.6)
- Plain Text editor (notepad ++, sublime, brackets)
- Command Line : start menu --> all programs --> accessories --> command prompt

- **I am using Windows Operating System.**

Go to terminal and check if Ruby is installed or not



First program in Ruby

- Go to any text editor like notepad++ or brackets or sublime .(I am using brackets)
- Type : puts 123 puts 121 and save it as first.rb where rb is the extension.
- Open terminal
- Navigate to that folder where you save the file
- Run the file as ruby first.rb or you can also type like this: Ruby first.rb (small r or capital R)
- You will see the output :

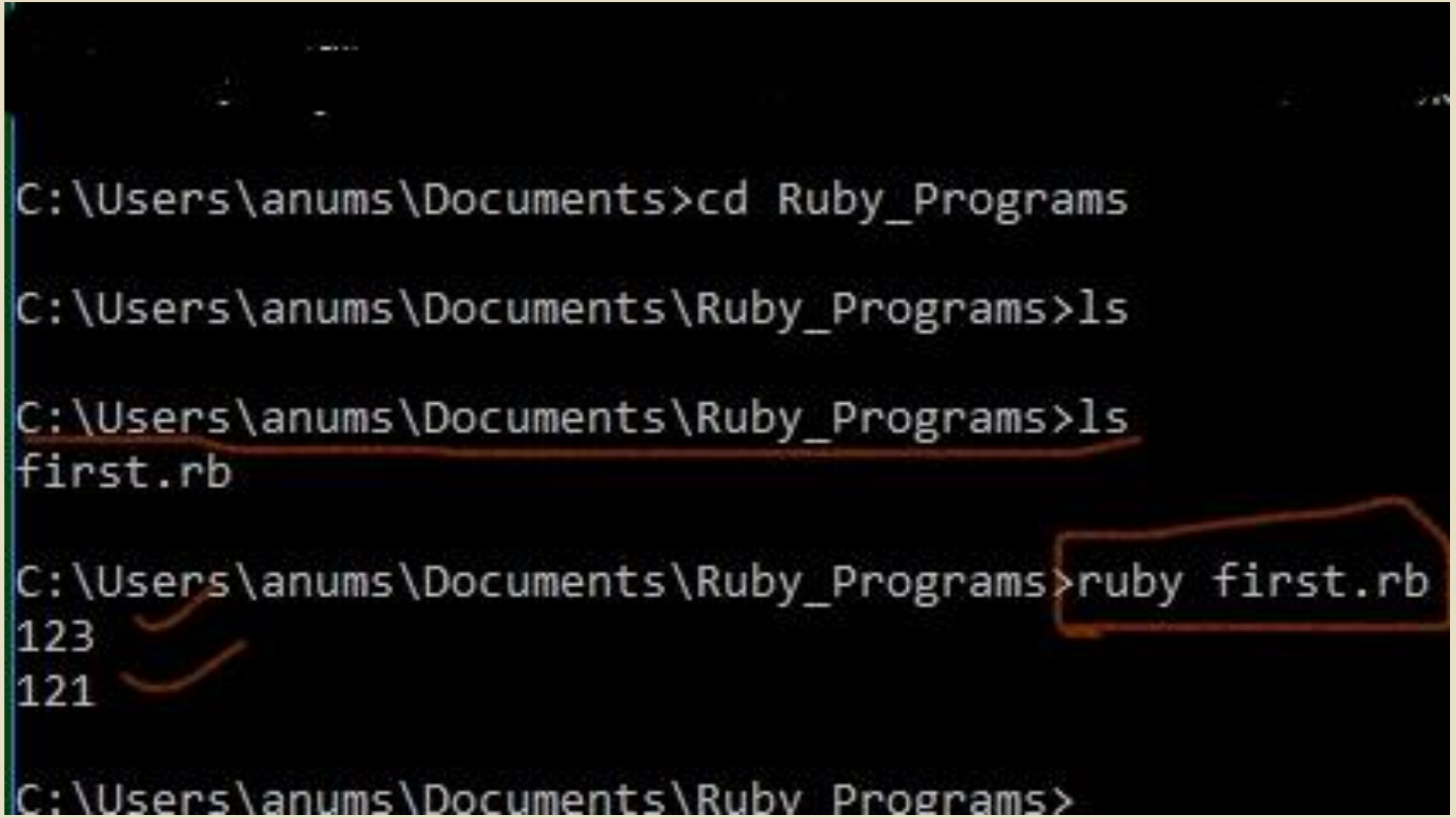
123

121

C:/Users/anums/Documents/Ruby_Programs/first.rb (

Debug Help

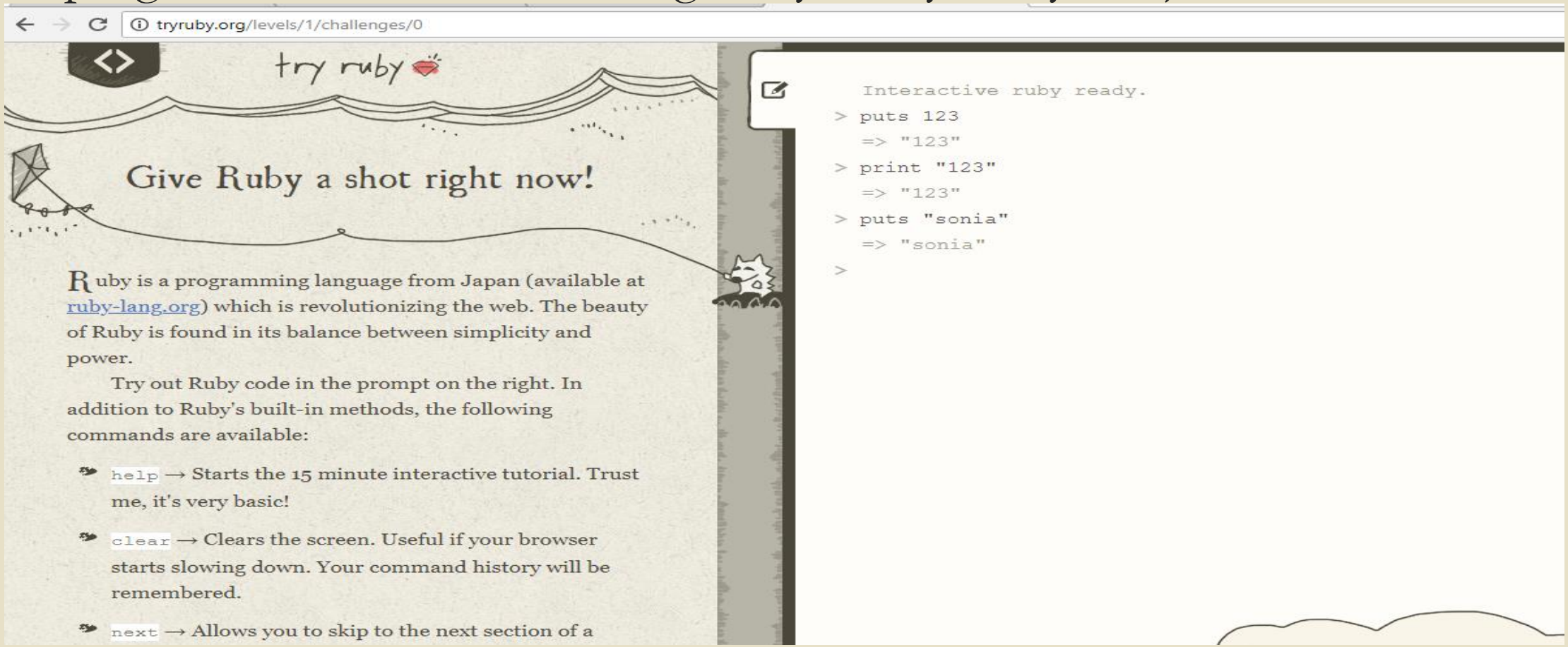
```
1   puts 123
2   puts 121
3
```


C:\Users\anums\Documents>cd Ruby_Programs
C:\Users\anums\Documents\Ruby_Programs>ls
C:\Users\anums\Documents\Ruby_Programs>ls
first.rb
C:\Users\anums\Documents\Ruby_Programs>ruby first.rb
123
121
C:\Users\anums\Documents\Ruby_Programs>

How to write comments ?

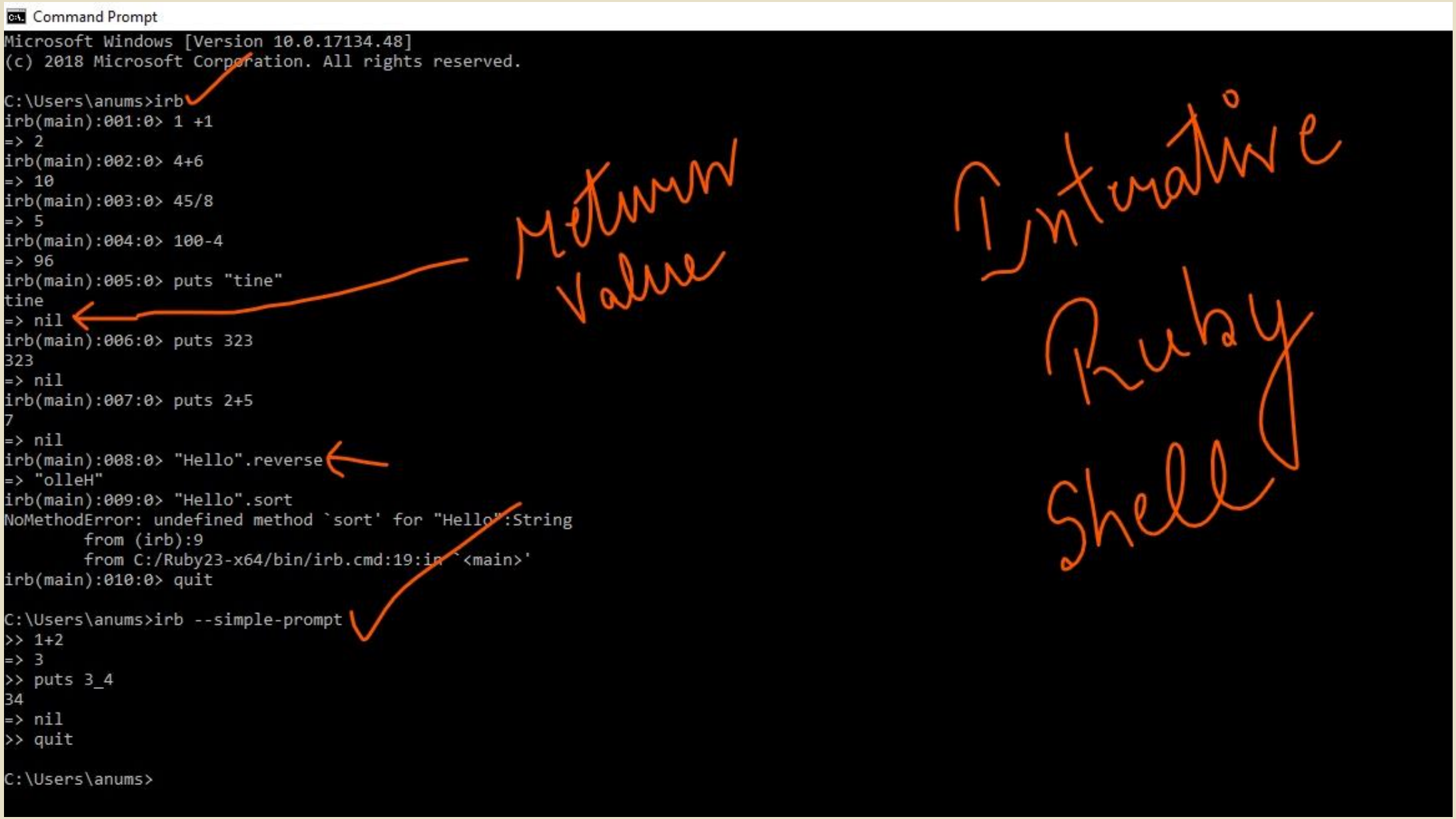
```ruby
# single line comment  (using hash sign)

puts 500
# print doesnot return a line return
print 300
puts 388

=begin

for mult-line comments use equal to begin and equal to end

.....
....
...
=end

puts "Hello"
puts "World"
```

Ruby terminal Online – **tryruby.org** (if you want to execute ruby programs online than installing Ruby into your system)



INTERACTIVE RUBY SHELL

- Allows us to interact with code in real time
- Works like a calculator
- Great for testing code
- Type **irb** (Interactive Ruby) in terminal and starts executing your code.


Command Prompt
Microsoft Windows [Version 10.0.17134.48]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Users\anums>irb
irb(main):001:0> 1 +1
=> 2
irb(main):002:0> 4+6
=> 10
irb(main):003:0> 45/8
=> 5
irb(main):004:0> 100-4
=> 96
irb(main):005:0> puts "tine"
tine
=> nil
irb(main):006:0> puts 323
323
=> nil
irb(main):007:0> puts 2+5
7
=> nil
irb(main):008:0> "Hello".reverse
=> "olleH"
irb(main):009:0> "Hello".sort
NoMethodError: undefined method `sort' for "Hello":String
from (irb):9
from C:/Ruby23-x64/bin/irb.cmd:19:in `<main>'
irb(main):010:0> quit
C:\Users\anums>irb --simple-prompt
>> 1+2
=> 3
>> puts 3_4
34
=> nil
>> quit
C:\Users\anums>
return value
Intuitive Ruby Shell

Ruby Documentation

- https://ruby-doc.org/core-2.5.1/ - read the documents here

- Or from terminal :

Type: ri upcase where ri stands for ruby information

You can see the use of upcase

Then press "q" to quit

Object Types

Object Types

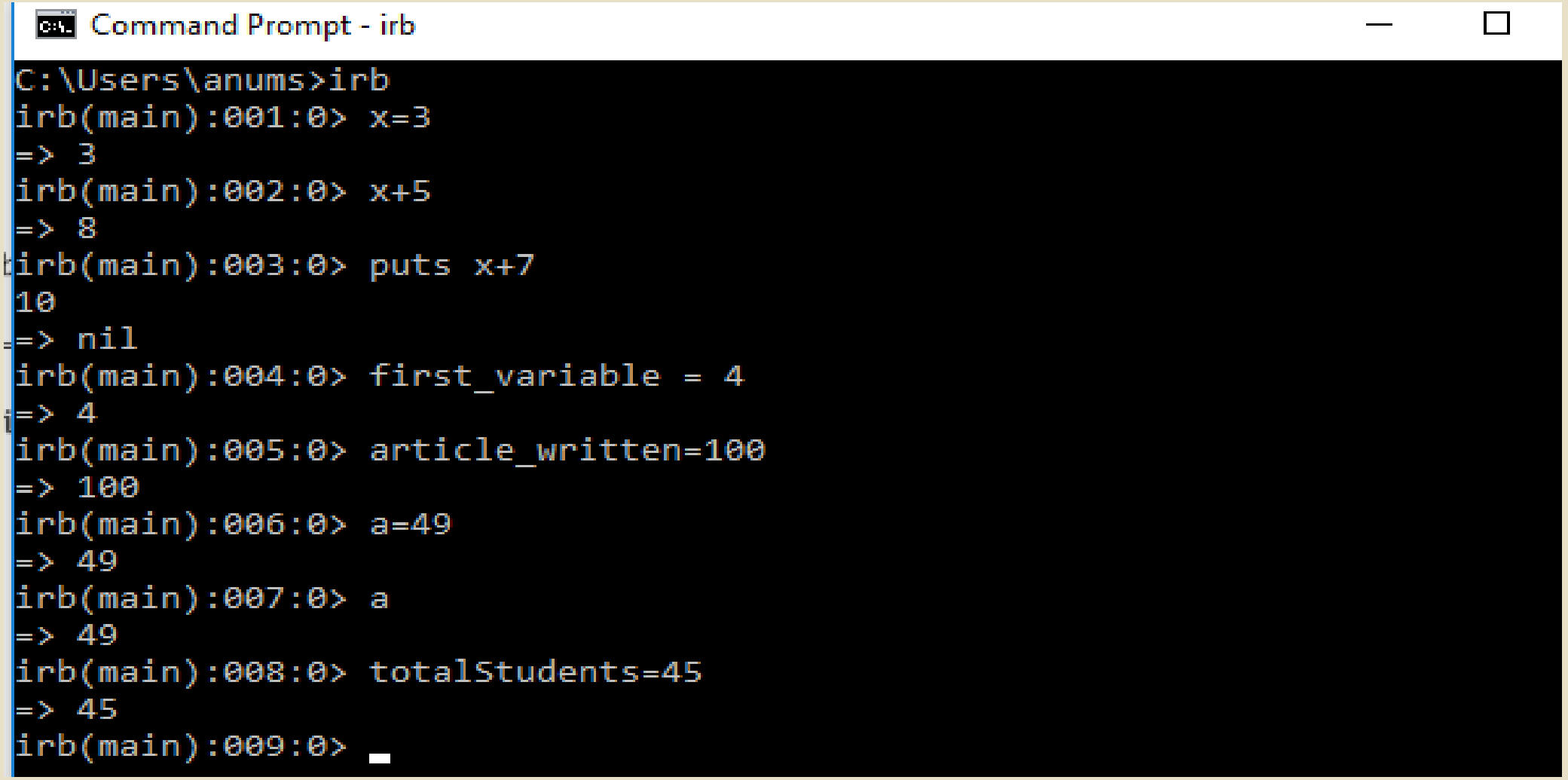
- Ruby is object oriented programming language.
- An object is the fundamental building block in ruby.

- ❑Variables
- ❑Float
- ❑Strings
- ❑Array
- ❑Hashes
- ❑Symbols
- ❑Boolean
- ❑Ranges
- ❑Constant

Variables

- They are not objects
- Part of ruby language.
- Allows us to easily reference objects
- Will be undefined or act like an object

Variables


Command Prompt - irb
C:\Users\anums>irb
irb(main):001:0> x=3
=> 3
irb(main):002:0> x+5
=> 8
irb(main):003:0> puts x+7
10
=> nil
irb(main):004:0> first_variable = 4
=> 4
irb(main):005:0> article_written=100
=> 100
irb(main):006:0> a=49
=> 49
irb(main):007:0> a
=> 49
irb(main):008:0> totalStudents=45
=> 45
irb(main):009:0>

Variables: scope indicators

Global	$variable
Class	@@variable
Instance	@variable
Local	variable
Block	variable

Numbers: Integers

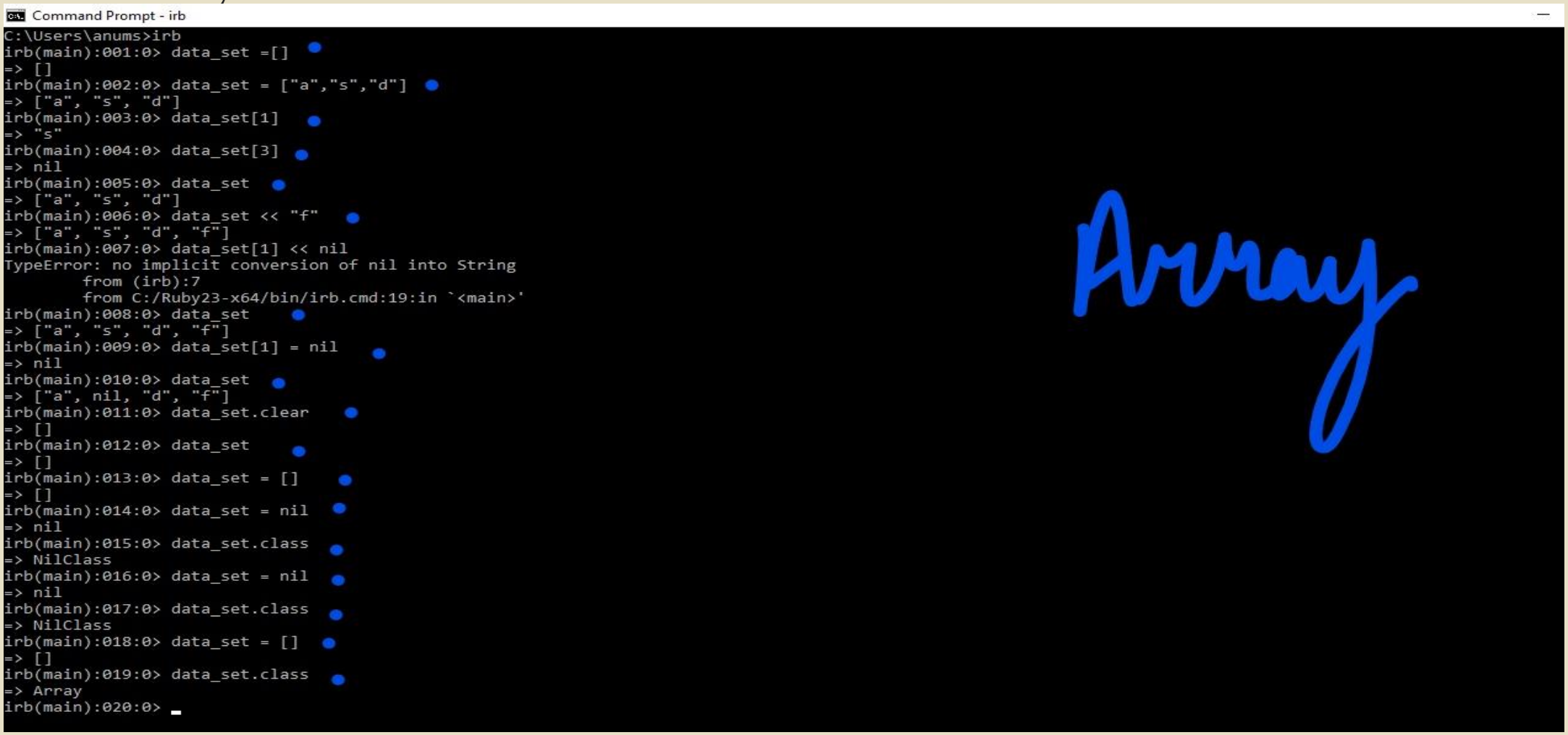
Numbers: Float

Strings

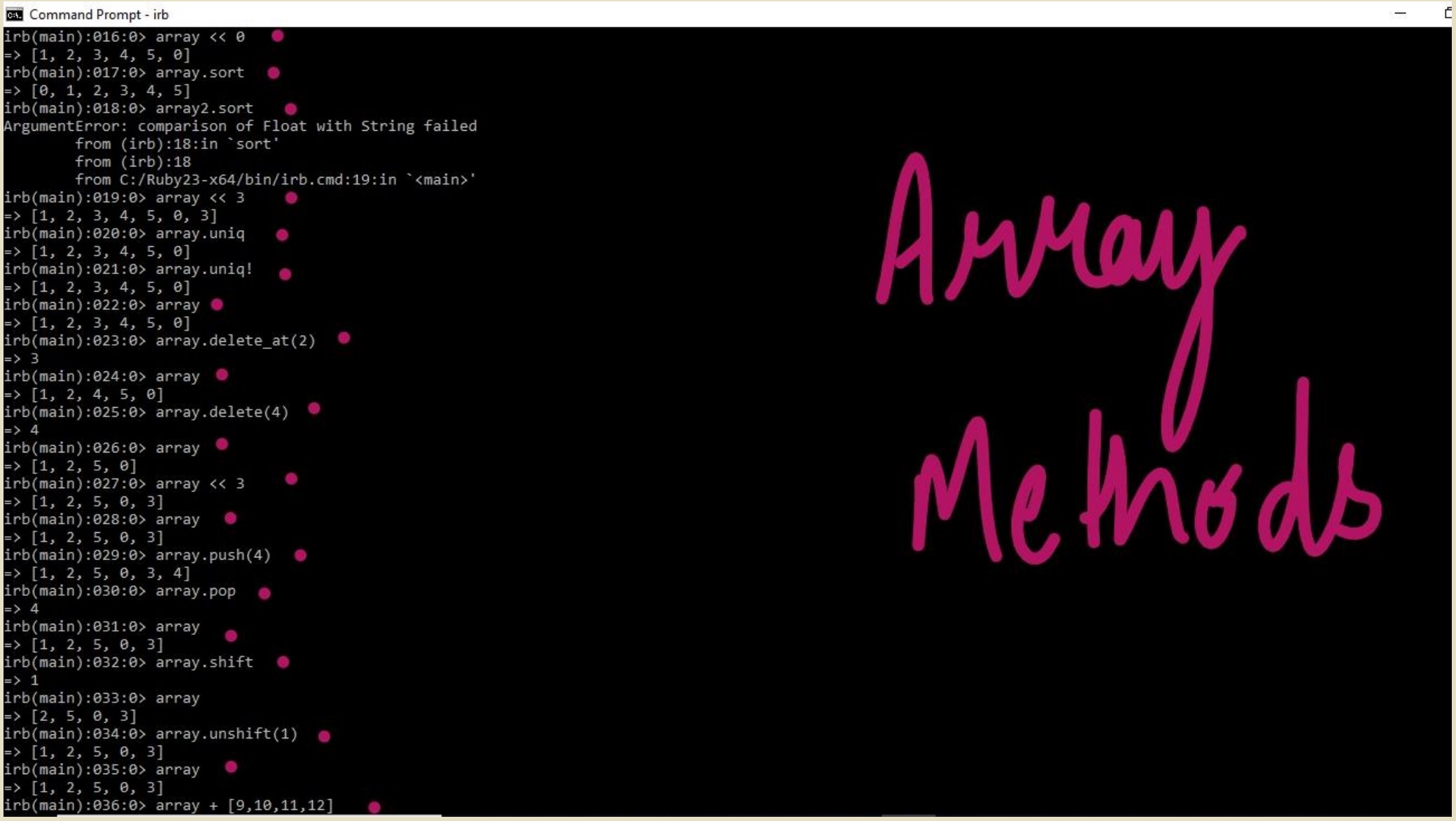

Command Prompt - irb
C:\Users\anums>irb
irb(main):001:0> "Hello"
=> "Hello"
irb(main):002:0> 'Hello'
=> "Hello"
irb(main):003:0> greeting-'Hello'
NameError: undefined local variable or method `greeting' for main:Object
from (irb):3
from C:/Ruby23-x64/bin/irb.cmd:19:in `<main>'
irb(main):004:0> greeting='Hello'
=> "Hello"
irb(main):005:0> target='World'
=> "World"
irb(main):006:0> greeting + ' ' + target
=> "Hello World"
irb(main):007:0> "kina"*4
=> "kinakinakinakina"
irb(main):008:0> '7'*4
=> "7777"
irb(main):009:0> 'I\'m escaped.'
=> "I'm escaped."
irb(main):010:0> "I said, \"I'm escapsed.\""
=> "I said, \"I'm escapsed.\""
irb(main):011:0> puts "\ta\tb\nc\nd"
a b
c
d
=> nil
irb(main):012:0> puts '\ta\tb\nc\nd'
\ta\tb\nc\nd
=> nil
irb(main):013:0> puts "I want to say #{greeting} #{target}."
I want to say Hello World.
=> nil
irb(main):014:0> puts 'I want to say #{greeting} #{target}.'
I want to say #{greeting} #{target}.
=> nil
irb(main):015:0> puts "1+1 = #{1+1}"
1+1 = 2
=> nil
irb(main):016:0> "Hello".capitalize
=> "Hello"
irb(main):017:0> "Hello".downcase
=> "hello"
Strings

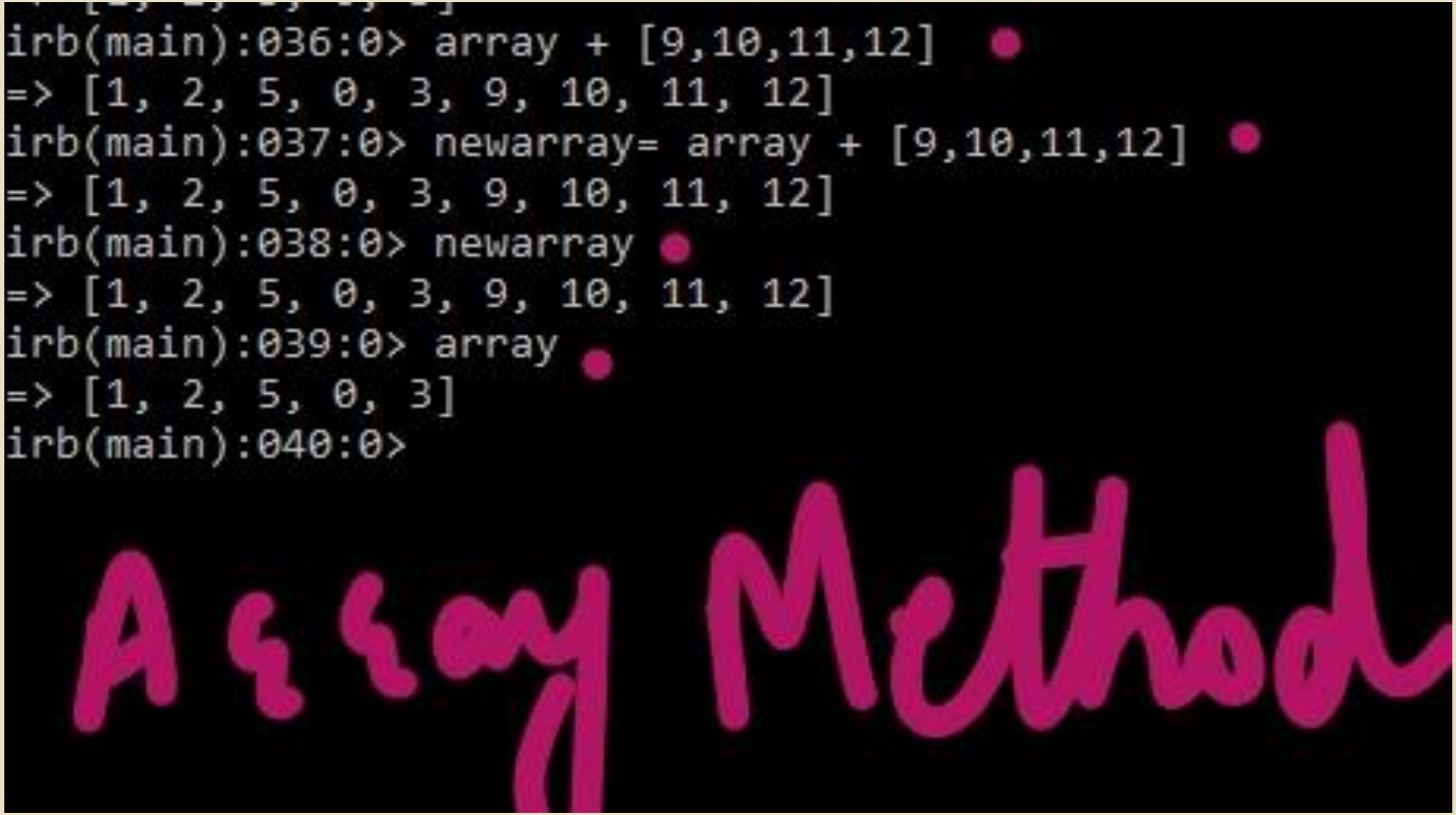

irb(main):018:0> "Hello".upcase
=> "HELLO"
irb(main):019:0> "Hello".length
=> 5
irb(main):020:0> "Hello".reverse.upcase
=> "OLLEH"
irb(main):021:0> "Hello".reverse.upcase.length
=> 5
irb(main):022:0> "Hello".reverse
=> "olleH"
irb(main):023:0>
Strings

Arrays – an ordered collection



Array Method


Command Prompt - irb
irb(main):016:0> array << 0
=> [1, 2, 3, 4, 5, 0]
irb(main):017:0> array.sort
=> [0, 1, 2, 3, 4, 5]
irb(main):018:0> array2.sort
ArgumentError: comparison of Float with String failed
from (irb):18:in `sort'
from (irb):18
from C:/Ruby23-x64/bin/irb.cmd:19:in `<main>'
irb(main):019:0> array << 3
=> [1, 2, 3, 4, 5, 0, 3]
irb(main):020:0> array.uniq
=> [1, 2, 3, 4, 5, 0]
irb(main):021:0> array.uniq!
=> [1, 2, 3, 4, 5, 0]
irb(main):022:0> array
=> [1, 2, 3, 4, 5, 0]
irb(main):023:0> array.delete_at(2)
=> 3
irb(main):024:0> array
=> [1, 2, 4, 5, 0]
irb(main):025:0> array.delete(4)
=> 4
irb(main):026:0> array
=> [1, 2, 5, 0]
irb(main):027:0> array << 3
=> [1, 2, 5, 0, 3]
irb(main):028:0> array
=> [1, 2, 5, 0, 3]
irb(main):029:0> array.push(4)
=> [1, 2, 5, 0, 3, 4]
irb(main):030:0> array.pop
=> 4
irb(main):031:0> array
=> [1, 2, 5, 0, 3]
irb(main):032:0> array.shift
=> 1
irb(main):033:0> array
=> [2, 5, 0, 3]
irb(main):034:0> array.unshift(1)
=> [1, 2, 5, 0, 3]
irb(main):035:0> array
=> [1, 2, 5, 0, 3]
irb(main):036:0> array + [9,10,11,12]
Array
Methods

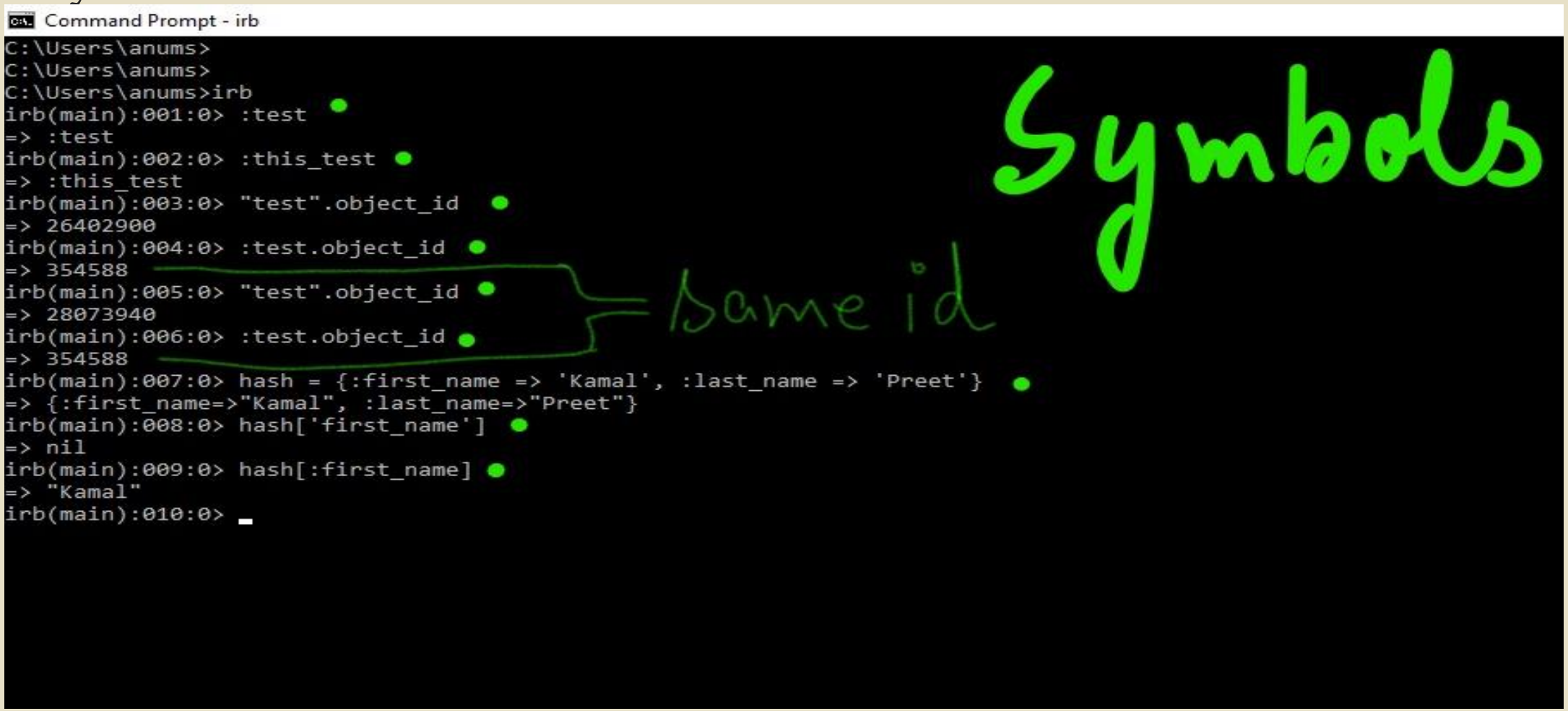

irb(main):036:0> array + [9,10,11,12]
=> [1, 2, 5, 0, 3, 9, 10, 11, 12]
irb(main):037:0> newarray= array + [9,10,11,12]
=> [1, 2, 5, 0, 3, 9, 10, 11, 12]
irb(main):038:0> newarray
=> [1, 2, 5, 0, 3, 9, 10, 11, 12]
irb(main):039:0> array
=> [1, 2, 5, 0, 3]
irb(main):040:0>
Array Method

Hashes – unordered, object-indexed collection of objects or (key-value pairs)

When to use array / hashes

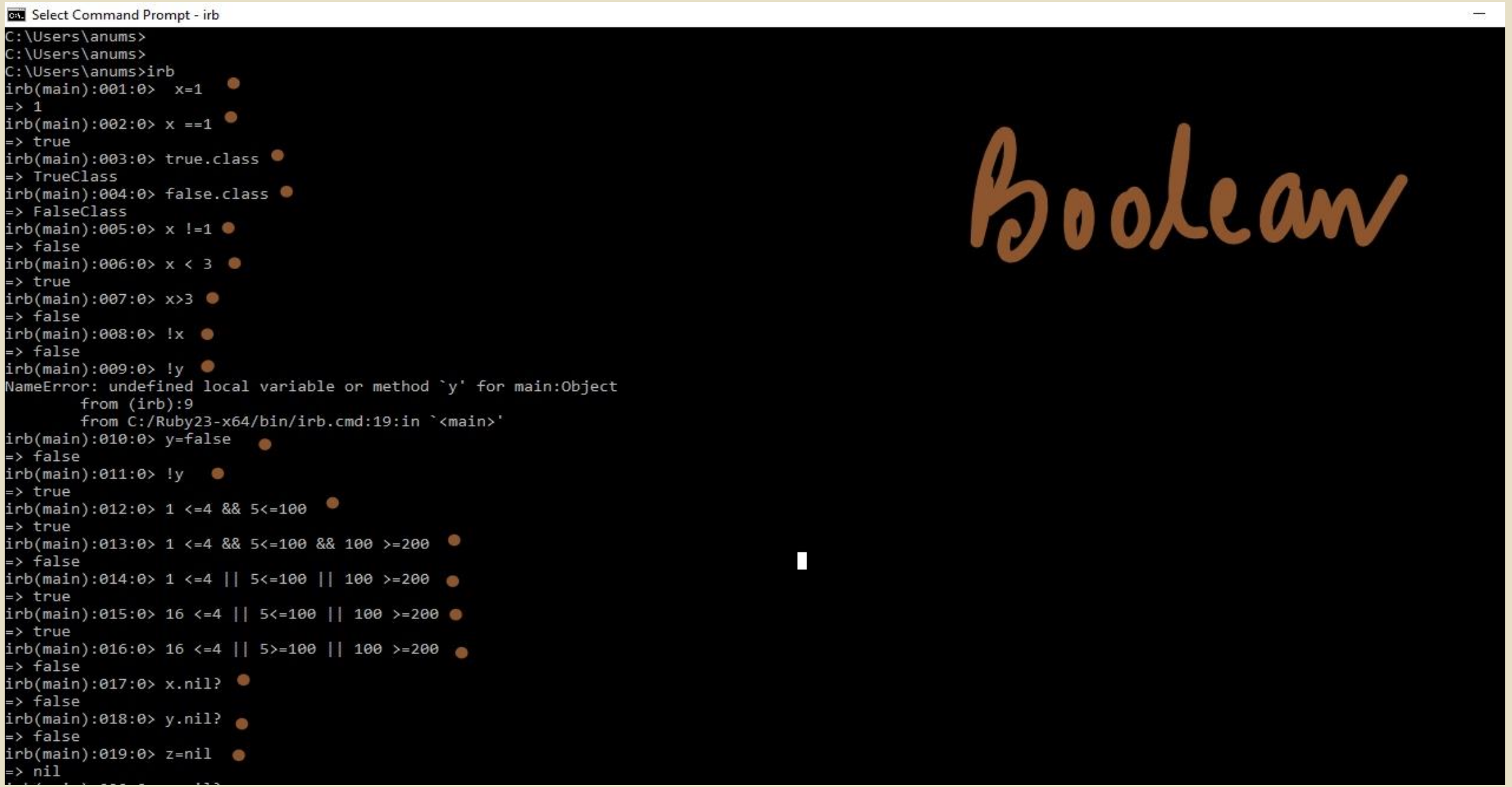
- Use arrays when the order matters
- Use hashes when label is matter

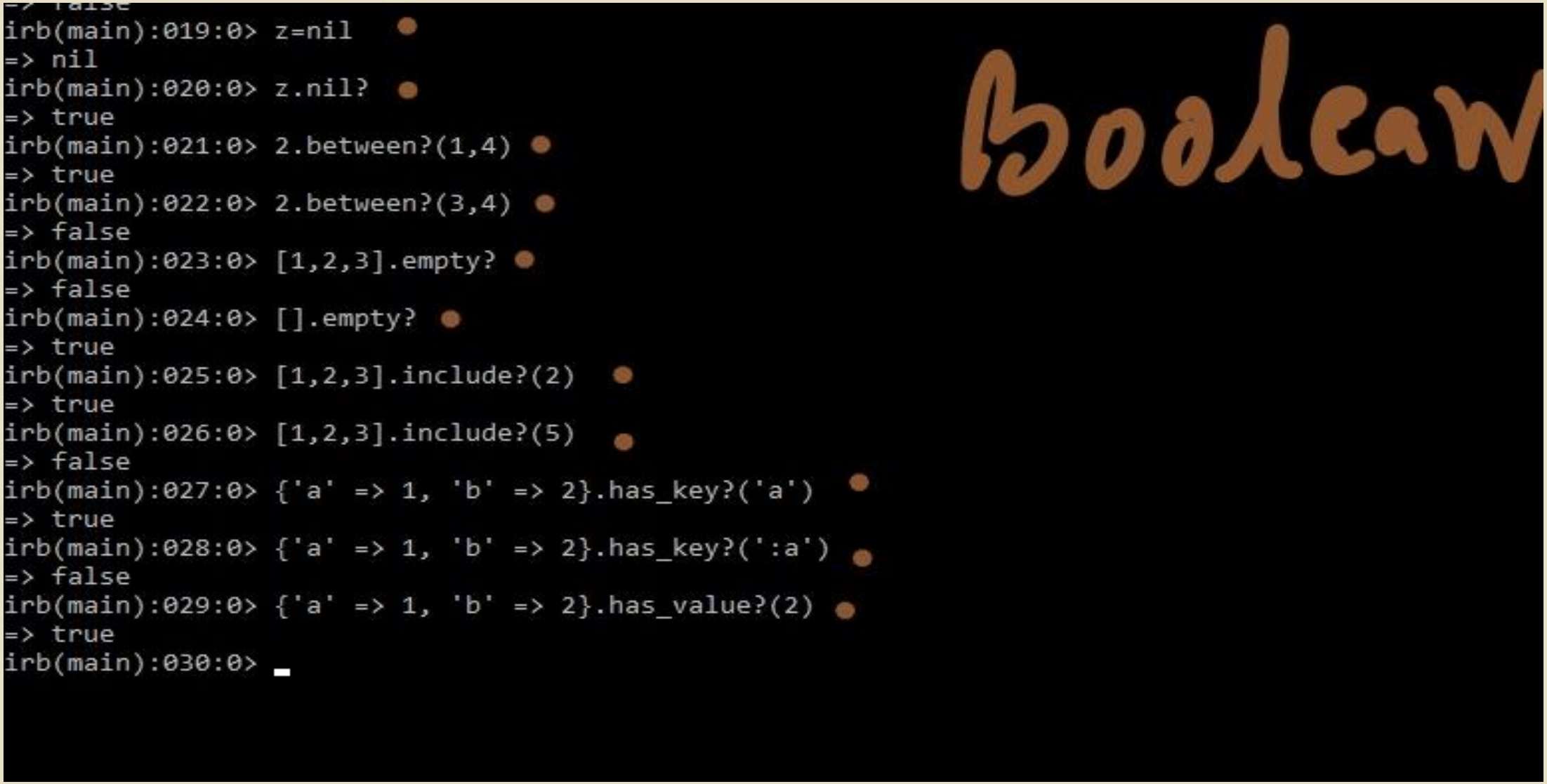
Symbols- is a label used to identify a piece of data AND only stored in memory one time



Boolean(true/false)- comparison and logic operators

Equal	==
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=
Not	!
Not equal	!=
AND	&&
OR	||


Select Command Prompt - irb
C:\Users\anums>
C:\Users\anums>
C:\Users\anums>irb
irb(main):001:0> x=1
=> 1
irb(main):002:0> x ==1
=> true
irb(main):003:0> true.class
=> TrueClass
irb(main):004:0> false.class
=> FalseClass
irb(main):005:0> x !=1
=> false
irb(main):006:0> x < 3
=> true
irb(main):007:0> x>3
=> false
irb(main):008:0> !x
=> false
irb(main):009:0> !y
NameError: undefined local variable or method `y' for main:Object
from (irb):9
from C:/Ruby23-x64/bin/irb.cmd:19:in `<main>'
irb(main):010:0> y=false
=> false
irb(main):011:0> !y
=> true
irb(main):012:0> 1 <=4 && 5<=100
=> true
irb(main):013:0> 1 <=4 && 5<=100 && 100 >=200
=> false
irb(main):014:0> 1 <=4 || 5<=100 || 100 >=200
=> true
irb(main):015:0> 16 <=4 || 5<=100 || 100 >=200
=> true
irb(main):016:0> 16 <=4 || 5>=100 || 100 >=200
=> false
irb(main):017:0> x.nil?
=> false
irb(main):018:0> y.nil?
=> false
irb(main):019:0> z=nil
=> nil
Boolean


irb(main):019:0> z=nil
=> nil
irb(main):020:0> z.nil?
=> true
irb(main):021:0> 2.between?(1,4)
=> true
irb(main):022:0> 2.between?(3,4)
=> false
irb(main):023:0> [1,2,3].empty?
=> false
irb(main):024:0> [].empty?
=> true
irb(main):025:0> [1,2,3].include?(2)
=> true
irb(main):026:0> [1,2,3].include?(5)
=> false
irb(main):027:0> {'a' => 1, 'b' => 2}.has_key?('a')
=> true
irb(main):028:0> {'a' => 1, 'b' => 2}.has_key?(':a')
=> false
irb(main):029:0> {'a' => 1, 'b' => 2}.has_value?(2)
=> true
irb(main):030:0>
Boolean

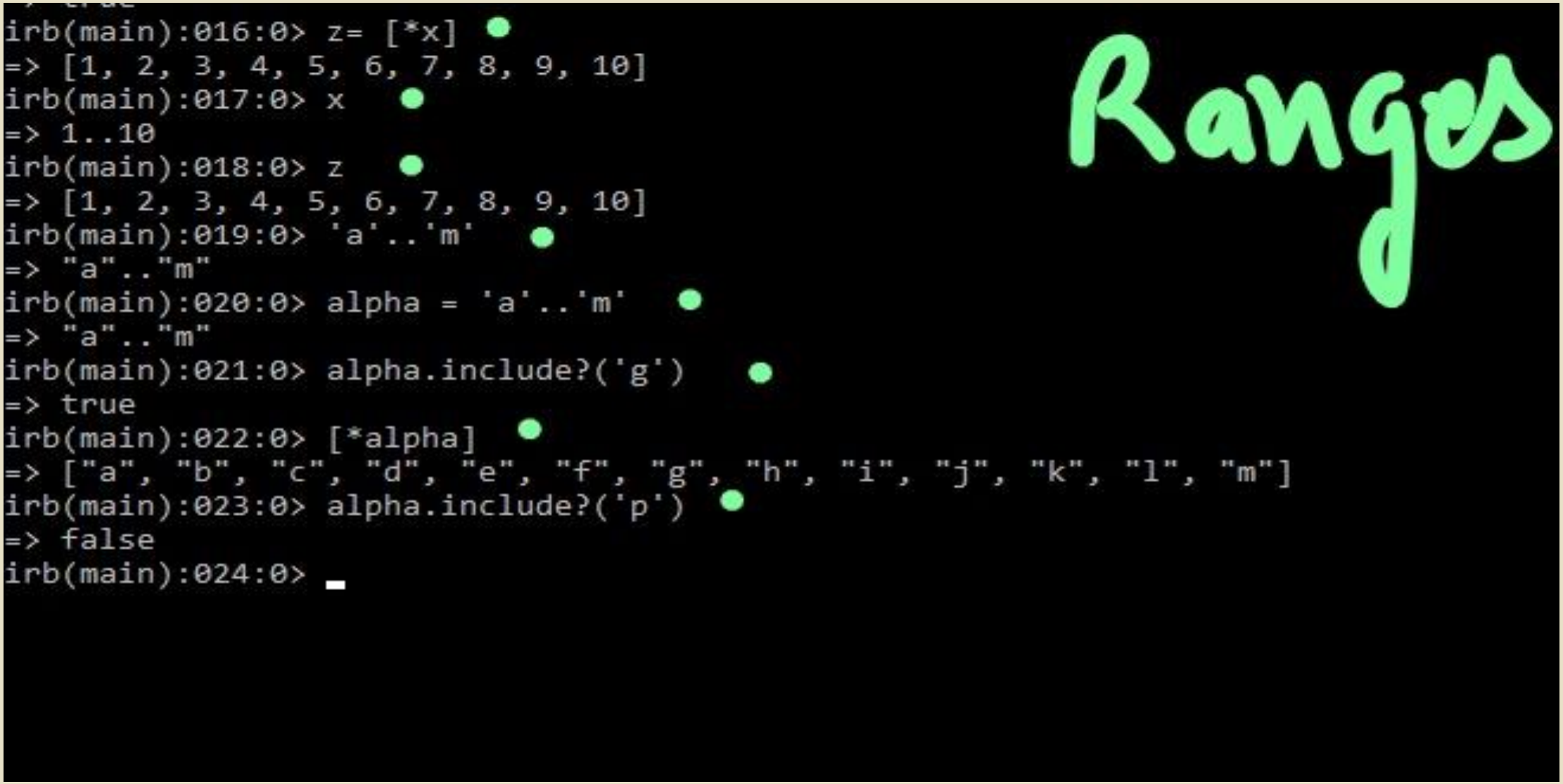
Ranges

- Inclusive range= 1...5 so it includes 1,2,3,4,5
- exclusive range = 1...5 so it includes 2,3,4

Command Prompt - irb

```
C:\Users\anums>
C:\Users\anums>irb
irb(main):001:0> 1..10
=> 1..10
irb(main):002:0> x= 1..10
=> 1..10
irb(main):003:0> x.class
=> Range
irb(main):004:0> 1..10.class
ArgumentError: bad value for range
        from (irb):4
        from C:/Ruby23-x64/bin/irb.cmd:19:in `<main>'
irb(main):005:0> (1..10).class
=> Range
irb(main):006:0> x.begin
=> 1
irb(main):007:0> x.end
=> 10
irb(main):008:0> x.first
=> 1
irb(main):009:0> x.last
=> 10
irb(main):010:0> y=1..10
=> 1..10
irb(main):011:0> y.begin
=> 1
irb(main):012:0> y.end
=> 10
irb(main):013:0> x.include?(1)
=> true
irb(main):014:0> y.include?(1)
=> true
irb(main):015:0> y.include?(10)
=> true
irb(main):016:0> z= [*x]
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
irb(main):017:0> x
=> 1..10
irb(main):018:0> z
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
irb(main):019:0> 'a'..'m'
=> "a".."m"
irb(main):020:0> alpha = 'a'..'m'
=> "a".."m"
```

Ranges


irb(main):016:0> z= [*x]
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
irb(main):017:0> x
=> 1..10
irb(main):018:0> z
=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
irb(main):019:0> 'a'..'m'
=> "a".."m"
irb(main):020:0> alpha = 'a'..'m'
=> "a".."m"
irb(main):021:0> alpha.include?('g')
=> true
irb(main):022:0> [*alpha]
=> ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m"]
irb(main):023:0> alpha.include?('p')
=> false
irb(main):024:0>

Constants:

- not true objects
- points to object.
- The constant are constant
- Different from variables
- Declare constant in capital letter , not in small letters
- TEST=10

Command Prompt - irb

```
C:\Users\anums>
C:\Users\anums>
C:\Users\anums>irb
irb(main):001:0> test=1
=> 1
irb(main):002:0> TEST=2
=> 2
irb(main):003:0> test
=> 1
irb(main):004:0> TEST
=> 2
irb(main):005:0> Hello = 10
=> 10
irb(main):006:0> test =100
=> 100
irb(main):007:0> TEST=100
(irb):7: warning: already initialized constant TEST
(irb):2: warning: previous definition of TEST was here
=> 100
irb(main):008:0> TEST
=> 100
irb(main):009:0> Hello =20
(irb):9: warning: already initialized constant Hello
(irb):5: warning: previous definition of Hello was here
=> 20
irb(main):010:0> Hello
=> 20
irb(main):011:0>
```

Constants

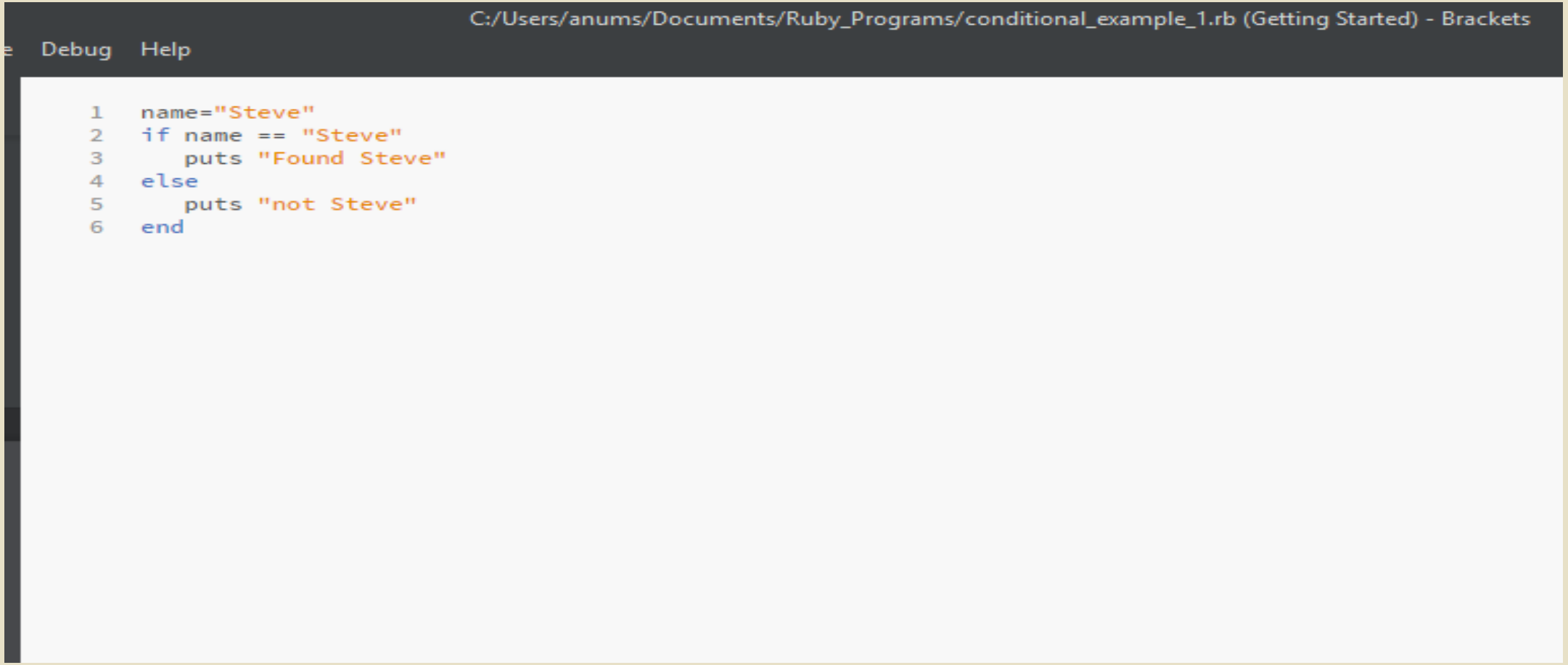
Control Statements

Conditionals

- Provide the action in Ruby programming

- ❑if , elsif and else
- ❑unless
- ❑case
- ❑ternary
- ❑or/or-equals

if and else statement example



if elsif and else example

C:/Users/anums/Documents/Ruby_Programs/conditional_example_2.rb (Getting Started) - Brackets

ebug Help

```ruby
# example of conditional statements
#x=56 first execution
x=17  # seconf execution
if x<=10
  puts "less than and equal to 10"
elsif x >=20
  puts "greater than and equal to 20"
else
puts "numbers are between 11 and 19"
end

```

unless

```
1   =begin
2   syntax for unless:
3
4   unless boolean
5   ...
6   end
7
8   =end
9   x = 1
10  unless x == 2
11    puts "x is not 2"
12  end
```

case

```
=begin
syntax for unless:

case test_value
when value
..
when value
..
else
..
end

=end

x=1
case
when x == 0
  puts "x is 0"
when x == 1
  puts "x is 1"
when x == 2
  puts "x is 2"
else
  puts "x is not 0, 1, or 2"
end
```

Ternary Operator

```
1   =begin
2   ternary operator:syntax
3
4   boolean ? code1 : code2
5
6   =end
7
8   x=1
9   puts x==1? "one" : "not one"
```

or/or equals

```
1   =begin
2   or/or-equals operator:syntax
3
4   unless x
5   x=y
6   end
7   is same as
8   x || = y
9   it means if x has a value then leave it alone
10  but if not , then we will set x=y
11  =end
12
13  x=1
14  y= nil
15  z=2
16
17  puts "example1"
18  x=y || z
19  puts "the value of x is #{x}"
20  puts "the value of y is #{y}"
21  puts "the value of z is #{z}"
22
23  puts "example2"
24  x ||= y
25  puts "the value of x is #{x}"
26  puts "the value of y is #{y}"
27
28
```

Output:

```
C:\Users\anums\Documents\Ruby_Programs>ruby conditional_example_1.rb
Found Steve

C:\Users\anums\Documents\Ruby_Programs>ruby conditional_example_2.rb
numbers are between 11 and 19

C:\Users\anums\Documents\Ruby_Programs>ruby unless_example.rb
x is not 2

C:\Users\anums\Documents\Ruby_Programs>ruby case_example.rb
x is 1

C:\Users\anums\Documents\Ruby_Programs>ruby ternary_example.rb
one

C:\Users\anums\Documents\Ruby_Programs>ruby or-equal-example.rb
example1
the value of x is 2
the value of y is
the value of z is 2
example2
the value of x is 2
the value of y is

C:\Users\anums\Documents\Ruby_Programs>
```

Loops

- Loop do : just like for loop
- Break : terminate the whole loop
- Next: jump to next loop
- Redo: redo this loop
- Retry: start the whole loop over
- While: while condition is true, loop over
- Until : if not

break

```ruby
x=0
loop do      # like for loop
  x += 2     # increment by 2
  break if x >= 20      # terminate from loop if x>=20
  puts x     # print the values of x
end
```

next

```
1   x=0
2   loop do
3     x += 2
4     break if x >= 20
5     next if x == 6
6     puts x
7   end
```

while

```
1   x = 0
2   while x < 20
3     x += 2
4     puts x
5   end
```

output

```
C:\Users\anums\Documents\Ruby_Programs>ruby break_example.rb
2
4
6
8
10
12
14
16
18

C:\Users\anums\Documents\Ruby_Programs>ruby next_example.rb
2
4
8
10
12
14
16
18

C:\Users\anums\Documents\Ruby_Programs>ruby while_example.rb
2
4
6
8
10
12
14
16
18
20
```

Iterators

- 1.upto(5) {puts "Hello"}
- 5.downto(1) { puts "Hello"}
- (1..5).each { puts "Hello" }

```ruby
1.upto(5) do |num|
  puts "Hello " + num.to_s
end
```

```ruby
fruits = ['banana', 'apple', 'pear']
# => ["banana", "apple", "pear"]
fruits.each do |fruit|
  puts fruit.capitalize
end

# another syntax
for fruit in fruits
  puts fruit.capitalize
end
```

```
20

C:\Users\anums\Documents\Ruby_Programs>ruby iterator_example1.rb
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5

C:\Users\anums\Documents\Ruby_Programs>ruby iterator_example2.rb
Banana
Apple
Pear
Banana
Apple
Pear

C:\Users\anums\Documents\Ruby_Programs>
```

Code Blocks

Code-Blocks

- It is block of code that we wanted to executed each time through the loop.
- Each iteration and that block of code is defined between "do" and "end".
- So everything between "do" and "end" is code block .

Example:

```
5.times do
    puts "Welcome"
end
```

Code Block Examples

```ruby
# code block examples

# example 1
5.times do
    puts "Hello"
end

# example 2
5.times { puts "Hello"}

# example 3
1.upto(5) do |i|
   puts "Hello" + i.to_s
end

# example 4
array= [1,2,3,4,5]
array.each {|num| puts num * 20 }
```

Common methods that use in code block

- Find
- Merge
- Collect
- Sort
- Inject

Code Block : Find

- Methods:
  - find/detect
  - find_all/select
  - any?
  - all?
  - delete_if

```ruby
# find example

 # each number will puts into |i| as it iteratees through set. if i=5 then i
puts (1..10).find {|i| i==5}

# find return the first value only .
puts (1..10).find {|i| i % 3 ==0}

# detect is just like find . detect is return the single object
puts (1..10).detect {|i| i% 3 == 0}

#detect is return nil
puts (1..10).find {|i| i==20 }

# find_all return the result in array (returns all the objects that match)
puts (1..10).find_all {|i| i % 3 ==0}

# this gives an empty array
puts (1..10).find_all {|i| i % 30 ==0}

# select is just like find_all
puts (1..10).select {|i| (1..10).include?(i * 3) }

# are there is any in the set . it will return a boolean
puts (1..10).any? {|i| i % 3 ==0}

# are all of them meets this requirement. return in boolean true/false
puts (1..10).all? {|i| i % 3 ==0}

# delete the values from array if it is match
puts [*1..10].delete_if {|i| i % 3 ==0}
```

Code Block : Merge

- It is used for hashes only

```ruby
# merge example

h1 = { "a" => 111, "b" => 222 }

h2 = { "b" => 111, "c" => 222 }

puts h1.merge(h2)

puts h2.merge(h1)

puts h1.merge(h2) { |key,old,new| new }

puts h1.merge(h2) { |key,old,new| old }

puts h1.merge(h2) { |key,old,new| old * 2 }

h1.merge(h2) do |key, old,new|
     if old<new
          puts old
     else
          puts new
     end
end

puts h1.merge(h2) {|k,o,n| o < n ? o : n}

h1.merge!(h2)
```

Code Block: Collect

- Collect has a synonym which is Map.
- Collect or map method really work the best with :
  - Arrays
  - Hashes
  - Ranges

Collect method example

```ruby
# collect method example

array =[1,2,3,4,5]
puts array

# example1
array.collect {|i| i + 1}
puts array

# example2
array.collect {|i| 1 * 40 }
puts array

# example3
puts ['apple', 'banana', 'orange'].map {|fruit| fruit.capitalize }

# example4
puts ['apple', 'banana', 'orange'].map {|fruit| fruit.capitalize if fruit =='banana' }
```

```ruby
# example 5
puts (1..20).collect {|num| num * 20 }

# example6
puts hash = {"a" => 111, "b" => 222, "c" => 333 }
puts hash.map {|k,v| k }

puts hash.map {|k,v| v * 20 }

puts hash.map {|k,v| "#{k}: #{v * 20}" }
```

Code Block: Sort (compare)

	Value 1 <=> value 2	
-1	Less Than	Moves "Left"
0	Equal	Stays
1	More Than	Moves "Right"

Sort Example

```ruby
# sort example
puts 1<=> 2

puts 2 <=> 1

puts 2 <=> 2

#example 2
puts array =[3,1,5,2,4]

puts "after sorting: array look like this"
puts array.sort {|v1,v2| v1 <=> v2 }
# it can be done like:
array.sort

puts   " reverse sort"

array.sort {|v1,v2| v2 <=> v1}
# it can done by:
array.sort.reverse
```

```ruby
# example 3
fruits =['banana', 'apple', 'orange', 'pear']
puts fruits.sort {|fruit1,fruit2| fruit1.length <=> fruit2.length }
puts fruits.sort {|fruit1,fruit2| fruit2.length <=> fruit1.length }

# you can do this too: fruits.sort

# example 4 sort by length
puts fruits.sort_by {|fruit| fruit.length }

puts fruits.sort_by {|fruit| fruit.reverse }

# example 5
puts fruits

puts fruits.sort! {|fruit1,fruit2| fruit1.length <=> fruit2.length }

puts fruits
```

```ruby
# example 6

puts hash ={"c" => 222, "a" => 555, "d" => 111, "b" => 333}

puts hash.to_a

# it will sort by keys
puts hash.sort {|item1,item2| item1[0] <=> item2[0] }

# it will sort by values
puts hash.sort {|item1,item2| item1[1] <=> item2[1] }
```

Code Block : Inject Method

- It is accumulator.
- It accumulates the values
- And storing it in "memo".
- Example : (1..10).inject {|memo, n| memo +n }

Inject Example

```ruby
# inject examples

array = [*1..10]
puts array

#example 1
sum = array.inject {|memo, n| memo + n }
puts sum

#example 2
sum = array.inject(100) {|memo, n| memo + n}
puts sum

#example 3
product = array.inject {|memo, n| memo * n}
puts product

#example 4
product = array.inject(2) {|memo, n| memo * n}
puts product
```

```ruby
#example 5
sum = array.inject {|memo, n| puts memo + n; memo }
puts sum

#example 6
fruits = ['apple', 'pear', 'banana', 'plum']
puts fruits

longest_word =fruits.inject do |memo,fruit|
   if memo.length > fruit.length
         memo
   else
         fruit
   end
end
puts "longest_word is: #{longest_word}"

#example 7
menu =["Home", "History", "Products", "Services", "Contact Us"]
puts menu.inject(10) {|memo, item| memo + item.length}
```

Methods

Defining and calling methods

```ruby
# method example

# defining the method using def and end
def welcome
    puts "Hello World!"
end

# calling method with method name
welcome

# example 2
def add
    puts 1 + 1
end

# calling add method
add
```

```ruby
# example 3

def longest_word
    words= ['apple','pear','banana','plum']
    longest_word = words.inject do |memo,word|
        memo.length > word.length ? memo : word
    end
    puts longest_word
end

#calling longest_word
longest_word


#example 4
# methods names have questions marks in them
# useful for tests and boolens
def over_five?
    value =3
    puts value > 5 ? 'Over 5' : 'Not over 5'
end

# calling method
over_five?
```

Variable scope in methods

```ruby
# variable scope examples

# global scope
value=7

def over_five?
    value =3 # scope of value=3 is within this method not outside of method
    puts "inside the method: #{value}"
    puts value > 5 ? 'Over 5' : 'Not over 5'
end

puts "outside the method/block: #{value}"
# calling method
over_five?
```

```ruby
# example 2
def longest_word
    words= ['apple','pear','banana','plum']
    longest_word = words.inject do |memo,word|
        memo.length > word.length ? memo : word
    end
    puts longest_word
end

#calling longest_word
longest_word


# let's print the longest_word value from the method
#  this one : longest_word = words.inject do |memo,word|

puts longest_word

# the result will same
# because both method name and local variable inside method is same
# betetr to use different names (method name and variable name)
```

output

```
C:\Users\...\Ruby_Programs>ruby method_scope.rb
outside the method/block: 7
inside the method: 3
Not over 5
banana
banana
```

Arguments

- Arguments are a comma separated list of values that are passed into methods.
- Values are passed in when that are called.

```ruby
# methods with arguments  typically use parenthesis
# methods  without arguments typically  donot.
# Parenthesis are optional on both cases

def welcome(name)
    puts"Hello #{name}"
end

# calling method
welcome("World")
welcome("Mary")
welcome "Fred"
```

```ruby
# add method with two arguments
def add(n1, n2)
    puts n1 + n2
end

#example
def over_five?(value)
    puts value > 5 ? 'Over 5' : 'Not over 5'
end

# calling over_five? method with one parameter
over_five?(4)

# calling add method with parameters
add(2,4)
```

```ruby
# example
fruits= ['apple','pear','banana','plum']

def longest_word(words)
    longest_word = words.inject do |memo,word|
        memo.length > word.length ? memo : word
    end
    puts longest_word
end

#calling longest_word
longest_word(fruits)
```

Arguments default values

```ruby
# default arguments example

def welcome(name="World")
    puts"Hello #{name}"
end

# calling method
welcome
welcome("Mary")

# example 2

def add(n1=0, n2=0)
    puts n1 + n2
end

add(3,7)
add
add(4)
```

```
Hello World
Hello Mary
10
0
4
```

Return Value

- Methods have a default return value: the last operation 's return value.
- Return will both return a value and exit the method.
- Returning the value and using puts outside a method can provide more flexibility than using puts inside.
- Return is specially used in conditional statements.
- Methods can return only one object, use an array to return more.

Return value example 1

```ruby
# Default return value is the last operation's return value
# (unless you explicitly return before it)

# example 1
def welcome(name="World")
    puts "let's try"
    return "Hello #{name}!"
    2 + 2
end

welcome
returned_value = welcome("John")
puts returned_value
```

```
let's try
let's try
Hello John!
```

```ruby
# example 2

# methods return only one value
# return multiple values as array
def add_and_subtract(n1=0, n2=0)
    add = n1 + n2
    sub = n1 - n2
    return [add, sub]
end

result = add_and_subtract(2, 2)
puts result[0]
puts result[1]
add, sub = add_and_subtract(8, 3)

# or you can write it like this:
#  [add, sub] = add_and_subtract(8, 3)
```

```ruby
# example 3

# Returning a value instead of outputting it from
# inside a method can give you greater flexiblity.
def longest_word(words=[])
  longest_word = words.inject do |memo,word|
    memo.length > word.length ? memo : word
  end
  return longest_word
end


fruits = ['apple', 'pear', 'banana', 'plum']
puts longest_word(fruits).length
```

```ruby
# example 4

# Return doesn't have to be at the end
# and there can be more than one.
# Useful for conditional statements.
def over_five?(value=nil)
    return "Exactly 5" if value.to_i == 5
    if value.to_i > 5
        return "Over 5"
    else
        return "Under 5"
    end
end

puts over_five?(112 / 18)
```

Operators are also methods

- Common operators in Ruby are methods too

Operators	methods
8 -2	8.-(2)
8 * 2	8.*(2)
8/2	8./2
8 ** 2	8.**(2)
Array <<4	Array.<<4
Array[2]	Array.[](2)
Array[2]='X'	Array.[]=(2,'X')