Ruby’s Building Blocks

Module 3

Arrays

- Ruby's arrays and hashes are indexed collections. Both store collections of objects, accessible using a key. With arrays, the key is an integer, whereas hashes support any object as a key. Both arrays and hashes grow as needed to hold new elements.
- A new array can be created by using the literal constructor [ ].
- Arrays can contain different types of objects.
- a = [1, "two", 3.0]
- An array can also be created by explicitly calling "Array.new".
- a = Array.new #=> **[ ]**
- Array.new(3) #=> **[nil, nil, nil]**
- Array.new(3, true) #=> **[true, true, true]**
- To build up multi-dimensional arrays a block can be passed.
- a = Array.new(3) {Array.new(3)}
  - #=> **[[nil, nil, nil], [nil, nil, nil], [nil, nil, nil]]**

Accessing Elements

- Elements in an array can be retrieved using the Array[] method.
- It can take a single integer argument (a numeric index), a pair of arguments (start and length) or a range.
- Negative indices start counting from the end, with -1 being the last element.

Accessing Elements Ex:

- arr = [1, 2, 3, 4, 5, 6]
- arr[2] #=> **3**
- arr[100] #=> **nil**
- arr[-3] #=> **4**
- arr[2, 3] #=> **[3, 4, 5]**
- arr[1..4] #=> **[2, 3, 4, 5]**
- arr[1..-3] #=> **[2, 3, 4]**
- Another way to access a particular array element is by using the at method
- arr.at(0) #=> **1**

- a = Array.new a[4] = "4“;
- a[0, 3] = ['a', 'b', 'c']
- a[1..2] = [1, 2]
- a[0, 2] = "?"
- a[0..2] = "A"
- a[-1] = "Z"
- a[1..-1] = nil
- a[1..-1] = []
- a[0, 0] = [1, 2]
- a[3, 0] = "B"

- a = Array.new a[4] = "4"; #=> **[nil, nil, nil, nil, "4"]**
- a[0, 3] = ['a', 'b', 'c'] #=> **["a", "b", "c", nil, "4"]**
- a[1..2] = [1, 2] #=> **["a", 1, 2, nil, "4"]**
- a[0, 2] = "?" #=> **["?", 2, nil, "4"]**
- a[0..2] = "A" #=> **["A", "4"]**
- a[-1] = "Z" #=> **["A", "Z"]**
- a[1..-1] = nil #=> **["A", nil]**
- a[1..-1] = [] #=> **["A"]**
- a[0, 0] = [1, 2] #=> **[[1, 2], "A"]**
- a[3, 0] = "B" #=> **[[1, 2],"A", "B"]**

- arr.fetch(100) #=> **IndexError: index 100 outside of array bounds:**
- arr.first #=> **1**
- arr.last #=> **6**
- To return the first n elements of an array, use take
- arr.take(3) #=> **[1, 2, 3]**
- The drop does the opposite of take, by returning the elements after n elements have been dropped
- arr.drop(3) #=> **[4, 5, 6]**

- To query an array about the number of elements it contains, use length, count or size.
- browsers = ['Chrome', 'Firefox', 'Safari', 'Opera', 'IE']
- browsers.length #=> **5**
- browsers.count #=> **5**
- browsers.empty? #=> **false**
- browsers.include?('Konqueror') #=> **false**

Adding Items to Arrays

- Items can be added to the end of an array by using either push or <<
- arr = [1, 2, 3, 4]
- arr.push(5) #=> **[1, 2, 3, 4, 5]**
- arr << 6 #=> **[1, 2, 3, 4, 5, 6]**
- unshift will add a new item to the beginning of an array.
- arr.unshift(0) #=> **[0, 1, 2, 3, 4, 5, 6]**

- a = ['ant', 'bee', 'cat', 'dog', 'elk']
- a[0] # => "ant"
- a[3] # => "dog"
- # this is the same if you don't wish to add quotes and commas:
- a = %w{ ant bee cat dog elk }
- a[0] # => "ant"
- a[3] # => "dog“
- Ruby hashes are similar to arrays. A hash literal uses braces rather than square brackets. The literal must supply two objects for every entry: one for the key, the other for the value. The key and value are normally separated by =>.
- inst_section = { ‘cello' => 'string‘, 'clarinet' => 'woodwind', 'drum' => 'percussion‘} #Key is String ‘cello’
- p inst_section['cello'] # "string"

- inst_section = {:cello => 'string', :clarinet => 'woodwind', :drum => 'percussion‘} #Key is symbol because we have ‘:’ before the object
- inst_section[:oboe] # => "woodwind"
- inst_section['cello'] # => nil
- inst_section = {cello: 'string', clarinet: 'woodwind‘}
- inst_section[:cello] #another way
- h = { 'dog' => 'canine', 'cat' => 'feline', 'donkey' => 'asinine' }
- h.length # => 3
- h['dog'] # => "canine"
- h['cow'] = 'bovine'
- h[12] = 'dodecine'
- h['cat'] = 99
- h # => {"dog"=>"canine", "cat"=>99, "donkey"=>"asinine", "cow"=>"bovine", 12=>"dodecine"}

Word Frequency: Using Hashes and Arrays

- Calculates the number of times each word occurs in some text.
- Let's start with the method that splits a string into words:

```
def words_from_string(string)
string.downcase.scan(/[\w']+/)
end
```

- This method uses two very useful String methods: downcase returns a lowercase version of a string, and scan returns an array of substrings that match a given pattern. In this case, the pattern is [\w']+, which matches sequences containing "word characters" and single quotes.
- `p words_from_string("But I didn't inhale, he said (emphatically)")`
- produces:
- ["but", "i", "didn't", "inhale", "he", "said", "emphatically"]

Create a hash object using Hash.new(0), the parameter (0 in this case) will be used as the hash’s default value—it will be the value returned if you look up a key that isn’t yet in the hash. Using that, we can write our count_frequency method:

```
def count_frequency(word_list)
    counts = Hash.new(0)
for word in word_list
counts[word] += 1
end
    counts
end
```

- p count_frequency(["sparky", "the", "cat", "sat", "on", "the", "mat"])

produces:

- {"sparky"=>1, "the"=>2, "cat"=>1, "sat"=>1, "on"=>1, "mat"=>1}

Each new word will be added into has as key elements and value of 1 will be assigned unless untill if same word is found back it increments the value of that particular key.

Ex: { ..., "the" => 1, ... }   counts[next_word] += 1   { ..., "the" => 2, ... }

Your Task read from text file and do the same above task in Lab

- With insert you can add a new element to an array at any position.
- arr.insert(3, 'apple') #=> **[0, 1, 2, 'apple', 3, 4, 5, 6]**

Removing Items from an Array:

- arr = [1, 2, 3, 4, 5, 6]
- arr.pop #=> **6** arr #=> **[1, 2, 3, 4, 5]**
- To retrieve and at the same time remove the first item, use shift :
- arr.shift #=> **1** arr #=> **[2, 3, 4, 5]**
- arr.delete_at(2) #=> **4** arr #=> **[2, 3, 5]**
- To delete a particular element anywhere in an array, use delete:
- arr = [1, 2, 2, 3]
- arr.delete(2) #=> **2** arr #=> **[1,3]**

- A useful method if you need to remove nil values from an array is compact:
- arr = ['foo', 0, nil, 'bar', 7, 'baz', nil]
- arr.compact!   **#=> ['foo', 0, 'bar', 7, 'baz']**
- arr   **#=> ['foo', 0, 'bar', 7, 'baz']**
- To remove duplicate elements from an array:
- arr = [2, 5, 6, 556, 6, 6, 8, 9, 0, 123, 556]
- arr.uniq   **#=> [2, 5, 6, 556, 8, 9, 0, 123]**
- Set Intersection :
- [1, 1, 3, 5] & [3, 2, 1] **#=> [1, 3]**
- Concatenation: [1, 2, 3] + [4, 5] **#=> [1, 2, 3, 4, 5]**
- Difference: [1, 1, 2, 2, 3, 3, 4, 5] - [1, 2, 4]
- **#=> [3, 3, 5]**

- a = ["a", "b", "c", "d", "e"]
- a.clear **#=> [ ]**
- **bsearch {|x| block } → elem**
- By using binary search, finds a value from this array which meets the given condition in block. #O(log n).
- ary = [0, 4, 7, 10, 12]
- ary.bsearch {|x| x >= 4 } **#=> 4**
- ary.bsearch {|x| x >= 6 } **#=> 7**
- **Returns the first element which matches the criteria.**

- Arrays with blocks: Array.new(4) {|i| i.to_s } **#=> ["0", "1", "2", "3"]**
- Array({:a => "a", :b => "b"}) **#=> [[:a, "a"], [:b, "b"]]**
- arr = [1, 2, 3, 4, 5]
- arr.each {|a| print a -= 10, " "}
- **# prints: -9 -8 -7 -6 -5**
- **#=> [1, 2, 3, 4, 5] #original array remain unchanged**
- <u>reverse_each</u>:
- words = %w[first second third fourth fifth sixth]
- str = ""
- words.reverse_each {|word| str += "#{word} "}
- p str #=> **"sixth fifth fourth third second first “**

- The map method can be used to create a new array based on the original array, but with the values modified by the supplied block.
- arr.map {|a| 2*a} #=> **[2, 4, 6, 8, 10]**
- arr #=> **[1, 2, 3, 4, 5]**
- arr.map! {|a| a**2} #=> **[1, 4, 9, 16, 25]**
- arr #=> **[1, 4, 9, 16, 25]**
- Non-destructive Selection
- arr = [1, 2, 3, 4, 5, 6]
- arr.select {|a| a > 3} #=> **[4, 5, 6]**
- arr.reject {|a| a < 3} #=> **[3, 4, 5, 6]**
- arr.drop_while {|a| a < 4} #=> **[4, 5, 6]**
- arr #=> **[1, 2, 3, 4, 5, 6]**
- Destructive Selection
- Original array elements will be changed **select! and reject!** are the corresponding destructive methods

- [1, 2, 3] * 3 #=> **[1, 2, 3, 1, 2, 3, 1, 2, 3]**
- [1, 2, 3] * "," #=> **“1,2,3”**

- **Collect:**
- a = ["a", "b", "c", "d"]
- a.collect {|x| x + "!"}
- |x| for each element in block, the value gets returned from index 0 until end and that will be added with "!".
- **#=> ["a!", "b!", "c!", "d!"]**
- **count {|item| block} → int:**
- ary = [1, 2, 4, 2]
- ary.count {|x| x%2 == 0} **#=> 3**
- **Each and Collect:**
- **a=[2,3,4,5]**
- **a.collect{|x|} # It needs a return variable**
- **=> [nil, nil, nil, nil]**
- **a.each{|x|} #each method** doesn't modify the array itself, no return value.
- **=>[2,3,4,5]**

- Fill :

- a = ["a", "b", "c", "d"]
- a.fill("x") #=> **["x", "x", "x", "x"]**
- a.fill("z", 2, 2) #=> **["x", "x", "z", "z"]**
- a.fill("y", 0..1) #=> **["y", "y", "z", "z"]**
- a.fill {|i| i*i} #=> **[0, 1, 4, 9]**

- **flatten → new_ary:**
- Returns a new array that is a one-dimensional flattening of self (recursively).
- s = [1, 2, 3] **#=> [1, 2, 3]**
- t = [4, 5, 6, [7, 8]] **#=> [4, 5, 6, [7, 8]]**
- a = [s, t, 9, 10] **#=> [[1, 2, 3], [4, 5, 6, [7, 8]], 9, 10]**
- a.flatten **#=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

- **hash → integer**
- A hash function is a function that takes an input (in this case, an array) and returns a fixed-size output, usually a numeric value, called a hash code or hash value.
- arr = [1, 2, 3]
- arr.hash   #=> 1831741227259963022
- The hash method generates a unique hash code based on the contents of the array.
- If two arrays have the same contents, they will have the same hash code.
- However, if the contents of the array change, the hash code will also change.

- **slice(index) → obj**
- **slice(start, length) → new_ary or nil**
- **slice(range) → new_ary or nil**
- **a.slice(1)   => "b“**
- **a.slice(1..3)   => ["b", "c", "d"]**
- **a.slice(1,3)=> ["b", "c", "d"]**
- **a.slice(1,2)=> ["b", "c"]**
- **a.slice(-3,3)   => ["c", "d", "e"]**

sort → new_ary

- ary = ["d", "a", "e", "c", "b"]
- ary.sort   **#=> ["a", "b", "c", "d", "e"]**

- **take(n) → new_ary:**
- a = [1, 2, 3, 4, 5, 0]
- a.take(3) **#=> [1, 2, 3]**
- **take_while {|obj| block} → new_ary**
- a = [1, 2, 3, 4, 5, 0]
- a.take_while {|i| i < 3} **#=> [1, 2]**
- **transpose → new_ary**
- a = [[1,2], [3,4], [5,6]]
- a.transpose **#=> [[1, 3, 5], [2, 4, 6]]**

Ranges

- (1...5).to_a # => [1, 2, 3, 4]
- (1..10).step(2).to_a # => [1, 3, 5, 7, 9]

- (Date.new(2024, 1, 1)..Date.new(2024, 1, 5)).to_a
- # [2024-01-01, 2024-01-02, 2024-01-03, 2024-01-04, 2024-01-05]

- ('a'..'f').include?('c') # => true
- ('a'..'f').cover?('z')   # => false

Numbers

- require 'bigdecimal'
- a = BigDecimal("0.1") + BigDecimal("0.2")
- b = BigDecimal("0.3")
- a == b # => true

- c1 = Complex(2, 3) # 2 + 3i
- c2 = Complex(1, -1) # 1 - i
- c1 + c2 # => Complex(3, 2)

- Math.sqrt(16)   # => 4.0
- Math.log(10)   # => 2.302585092994046
- Math.sin(Math::PI / 2) # => 1.0

- rand # => random float between 0.0 and 1.0
- rand(100) # => random integer between 0 and 99

String

- A String object holds and manipulates an arbitrary sequence of bytes, typically representing characters.
- "Ho! " * 3 #=> **"Ho! Ho! Ho! “**
- **str + other_str → new_str:**
- str1 = "Hello, "
- str2 = "world!"
- result = str1 + str2
- puts result #=> "Hello, world!“
- **Interpolation:** #{}
- name = "Alice" puts "Hello, #{name}!" #=> "Hello, Alice!"

- num = 3.14159
- str = sprintf("The value of pi is %.2f", num)
- puts str
- **Substring manipulation:**
- str = "Hello, world!“
- puts str[0]
- puts str[7, 5]
- puts str[7..11]
- **String case manipulation**
- str = "Hello, World!“
- puts str.upcase
- puts str.downcase
- puts str.capitalize
- puts str.swapcase

- **String formatting**
- num = 3.14159
- str = sprintf("The value of pi is %.2f", num)
- puts str   #=> "The value of pi is 3.14“
- **Substring manipulation:**
- str = "Hello, world!“
- puts str[0] #=> "H"
- puts str[7, 5] #=> "world"
- puts str[7..11] #=> "world“
- **String case manipulation**
- str = "Hello, World!“
- puts str.upcase   #=> "HELLO, WORLD!"
- puts str.downcase   #=> "hello, world!"
- puts str.capitalize   #=> "Hello, world!"
- puts str.swapcase   #=> "hELLO, wORLD!"

- **String trimming:**
- str = " hello, world "
- trimmed_str = str.strip
- puts trimmed_str #=> "hello, world”

Containers

- Container is a general term used to refer to data structures that hold and organize multiple values. There are several built-in container types in Ruby, such as arrays and hashes, which allow you to store and manipulate collections of data.
- **Arrays:** An array is an ordered collection of objects, which can be of any data type. Arrays are represented by square brackets ([]) and can be created by listing the elements inside the brackets, separated by commas.
- **Hashes:** unordered collection of key-value pairs, where each key is unique. Hashes are represented by curly braces ({})
- hash2 = { 1 => "one", 2 => "two", 3 => "three" }
- puts hash2[2] #two
- **Sets:** A set is an unordered collection of unique elements.
- set1 = Set.new([1, 2, 3, 4, 5])
- set2 = Set.new([3, 4, 5, 6, 7])
- puts set1 | set2 # {1, 2, 3, 4, 5, 6, 7}

- require 'set'
- set1 = Set.new([1, 2, 3])
- set2 = Set.new([3, 4, 5])
- set1.union(set2) # => #{1, 2, 3, 4, 5}
- set1.intersection(set2) #{3}

- set1.difference(set2) # {1, 2}
- set1.subset?(set2) # => false

Stack and Queue

- stack = []
- stack.push(1)
- stack.push(2)
- stack.pop # => 2
- stack # => [1]

```
require 'thread'
queue = Queue.new
queue.push(1)
queue.push(2)
queue.pop # => 1
```

Implementing Stack with Class:

```
class Stack
  def initialize
    @elements = []
  end

  def push(element)
    @elements.push(element)
  end

  def pop
    @elements.pop
  end

  def top
    @elements.last
  end

  def empty?
    @elements.empty?
  end
end

stack = Stack.new
stack.push(1)
stack.push(2)
stack.pop # => 2
```

Balanced Parentheses Check Using Stack

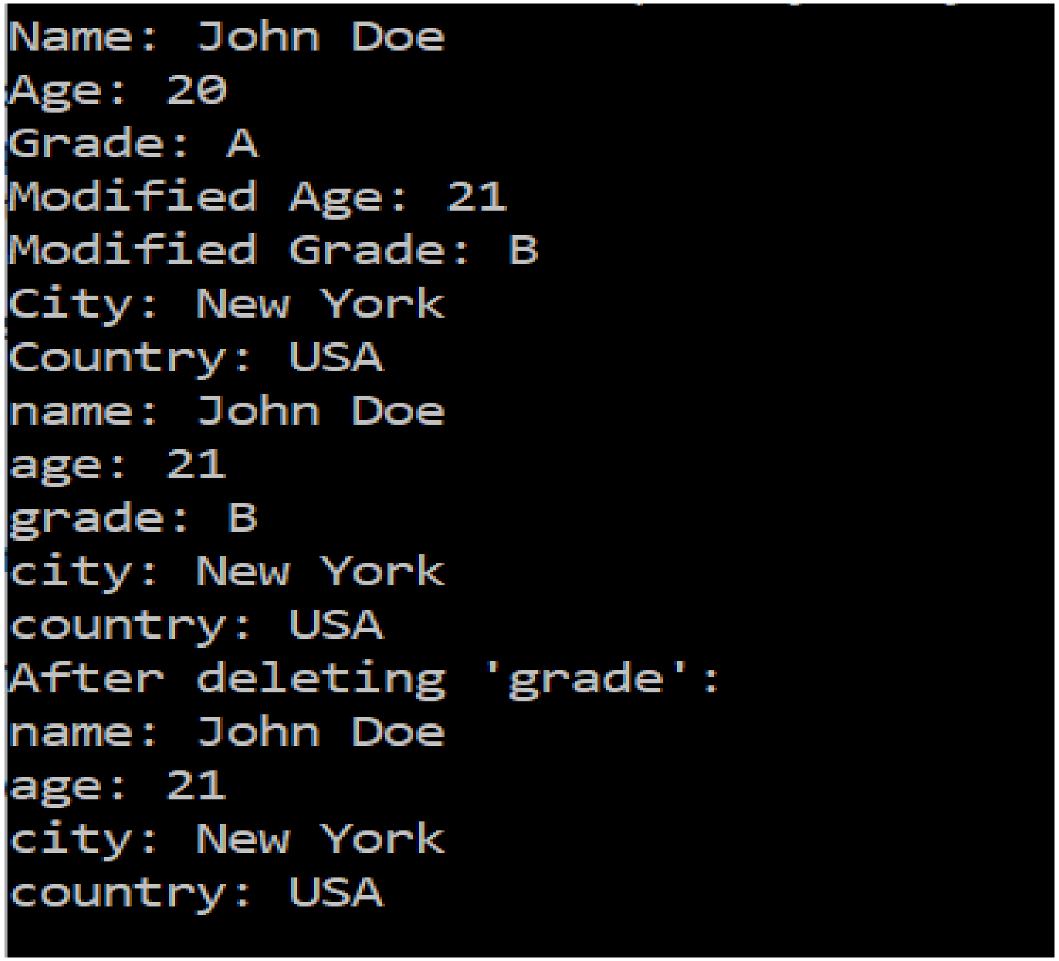
```
def balanced_parentheses?(string)
  stack = [ ]
  pairs = { '(' => ')', '{' => '}', '[' => ']' } #hash key and value

  string.each_char do |char|
    if pairs.keys.include?(char)
      stack.push(char)
    elsif pairs.values.include?(char)
      return false if stack.empty? || pairs[stack.pop] != char
    end
  end

  stack.empty?
end

puts balanced_parentheses?("({[]})") # => true
puts balanced_parentheses?("({[})")  # => false
```

Try

- expression = ['2', '3', '+', '4', '*']
- # Equivalent to (2 + 3) * 4
- puts evaluate_postfix(expression) # => 20

```ruby
# Creating a hash
student = {
  "name" => "John Doe",
  "age" => 20,
  "grade" => "A"
}

# Accessing hash values
puts "Name: #{student["name"]}"
puts "Age: #{student["age"]}"
puts "Grade: #{student["grade"]}"

# Modifying hash values
student["age"] = 21
student["grade"] = "B"

puts "Modified Age: #{student["age"]}"
puts "Modified Grade:
    #{student["grade"]}"

# Adding new key-value pairs
student["city"] = "New York"
student["country"] = "USA"

puts "City: #{student["city"]}"
puts "Country: #{student["country"]}"

# Iterating over hash
student.each do |key, value|
  puts "#{key}: #{value}"
end

# Removing a key-value pair
student.delete("grade")

puts "After deleting 'grade':"
student.each do |key, value|
  puts "#{key}: #{value}"
end
```

O/P



```
Name: John Doe
Age: 20
Grade: A
Modified Age: 21
Modified Grade: B
City: New York
Country: USA
name: John Doe
age: 21
grade: B
city: New York
country: USA
After deleting 'grade':
name: John Doe
age: 21
city: New York
country: USA
```

Min Stack

```
class MinStack
 def initialize
  @stack = []
  @min_stack = []
 end

 def push(x)
  @stack.push(x)
  if @min_stack.empty? || x <= @min_stack.last
   @min_stack.push(x)
  end
 end

 def pop
  return if @stack.empty?
  popped = @stack.pop
  @min_stack.pop if popped == @min_stack.last
 end

 def get_min
   @min_stack.last
  end
end

# Usage
min_stack = MinStack.new
min_stack.push(3)
min_stack.push(5)
puts min_stack.get_min # Output: 3
min_stack.push(2)
min_stack.push(1)
puts min_stack.get_min # Output: 1
min_stack.pop
puts min_stack.get_min # Output: 2
```

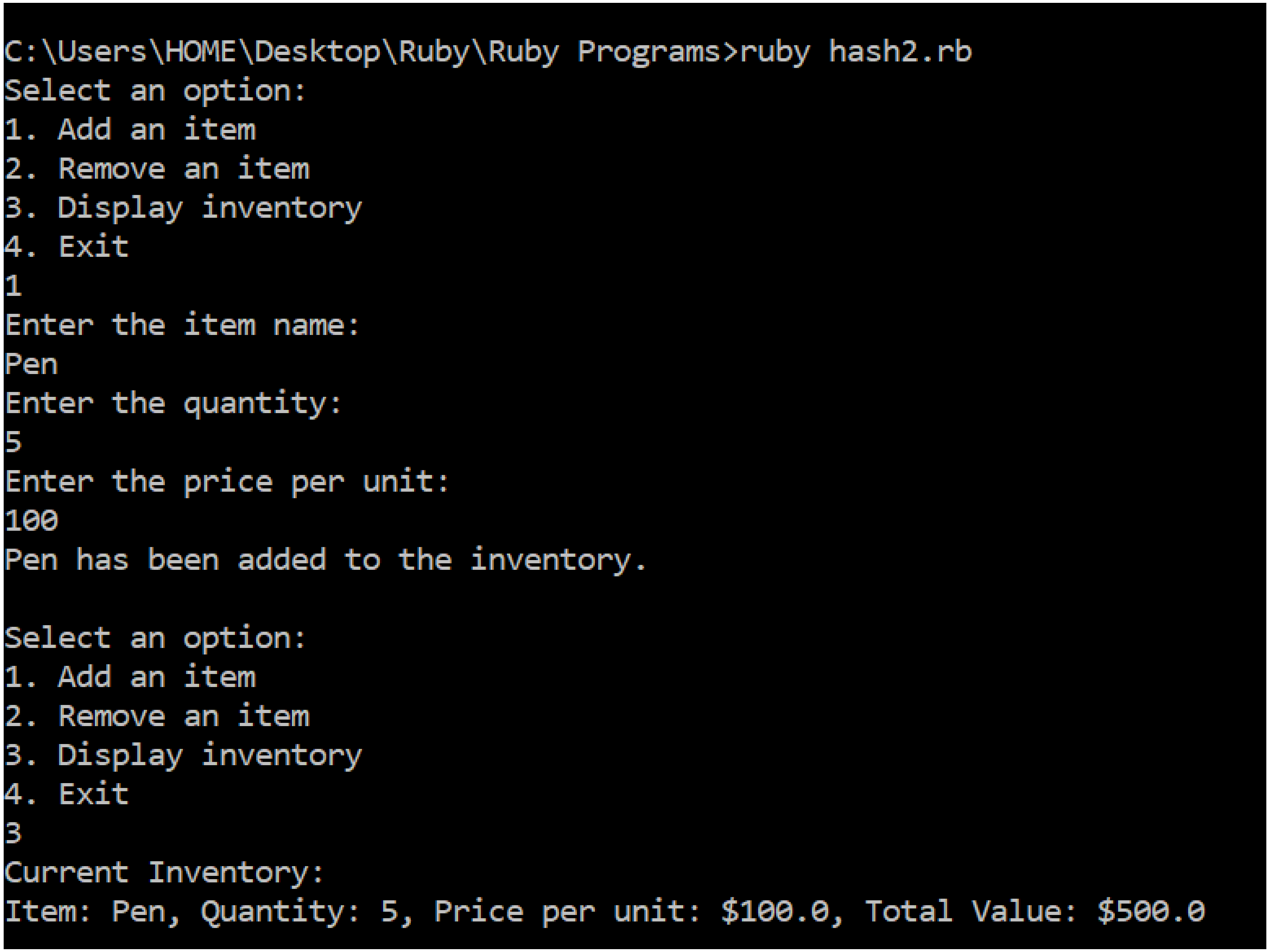
```
# Inventory Management System # Create an empty inventory hash

inventory = {}
# Function to add an item to the inventory
def add_item(inventory)
  puts "Enter the item name:"
  name = gets.chomp
#Chomp Used to remove the trailing newline character (\n) from a string.
  puts "Enter the quantity:"
  quantity = gets.chomp.to_i

  puts "Enter the price per unit:"
  price = gets.chomp.to_f

  inventory[name] = { quantity: quantity, price: price }

  puts "#{name} has been added to the inventory."
end
```

```
# Function to remove an item from the inventory

def remove_item(inventory)
  puts "Enter the item name to remove:"
  name = gets.chomp

  if inventory.key?(name)
    inventory.delete(name)
    puts "#{name} has been removed from the inventory."
  else
    puts "#{name} is not found in the inventory."
  end
end
```

```
# Function to display the current inventory

def display_inventory(inventory)
  if inventory.empty?
    puts "The inventory is empty."
  else
    puts "Current Inventory:“
# Iterate over each key-value pair in the inventory hash.
    inventory.each do |name, data|
# accesses the value of the :quantity key within the data
      quantity = data[:quantity]
      price = data[:price]
      total_value = quantity * price
      puts "Item: #{name}, Quantity: #{quantity}, Price per unit: $#{price}, Total
     Value: $#{total_value}"
    end
  end
end
```

```ruby
# Main program loop
loop do
  puts "Select an option:"
  puts "1. Add an item"
  puts "2. Remove an item"
  puts "3. Display inventory"
  puts "4. Exit"

  option = gets.chomp.to_i

  case option
  when 1
    add_item(inventory)
  when 2
    remove_item(inventory)
  when 3
    display_inventory(inventory)
  when 4
    break
  else
    puts "Invalid option. Please try again."
  end

  puts "\n"
end

puts "Exiting the program. Goodbye!"
```

O/P



```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby hash2.rb
Select an option:
1. Add an item
2. Remove an item
3. Display inventory
4. Exit
1
Enter the item name:
Pen
Enter the quantity:
5
Enter the price per unit:
100
Pen has been added to the inventory.

Select an option:
1. Add an item
2. Remove an item
3. Display inventory
4. Exit
3
Current Inventory:
Item: Pen, Quantity: 5, Price per unit: $100.0, Total Value: $500.0
```

First non-repeating character in a given string

```
def first_non_repeating_character(string)
  # Step 1: Convert the string into an array of characters
  chars = string.chars

  # Step 2: Create a hash to store the count of each character
  char_count = Hash.new(0)

  # Step 3: Populate the hash with the frequency of each character
  chars.each do |char|
    char_count[char] += 1
  end
chars.each do |char|
    return char if char_count[char] == 1
  end
  # Step 5: Return nil if no non-repeating character is found
  nil
End
# Test the function with different inputs
puts first_non_repeating_character("swiss")    # Output: "w"
puts first_non_repeating_character("ruby")     # Output: "r"
puts first_non_repeating_character("aabbcc")   # Output: nil
```

There are, in total, 21 match sticks to start the game. First, we ask the user to pick 1 or 2 or 3 or 4 matches per pick. Once the user makes their pick, the computer makes a choice (the same rules apply to the computer, i.e., it can pick either 1 or 2 or 3 or 4 matches per pick). The trick is that the computer's choice is always five minus the user's pick. For example, if the computers pick is variable c and the user pick is stored in variable p, then: $c = 5 - p$; Who wins the game?

Files and Directories
Module 4

Class IO

- Class IO is the basis for input and output in Ruby.
- Class File is the only class in the Ruby core that is a subclass of IO.
- Class StringIO provides an IO-like stream that handles a String.
- To work with a file, you can use the File class to create an input or output object that represents the file. For example, to open a file named example.txt in read mode and read its contents, you can use the following code:

- file = File.open("example.txt", "r")
- # create input object for file
- contents = file.read   # read contents of file
- puts contents   # print contents to console
- file.close   # close file

Mode	Meaning
r	Read-only, starts at beginning of file (default mode).
r+	Read/write, starts at beginning of file.
w	Write-only, truncates (reduce) an existing file to zero length or creates a new file for writing.
w+	Read/write, truncates existing file to zero length or creates a new file for readingand writing.
a	Write-only, starts at end of file if file exists; otherwise, creates a new file for writing.
a+	Read/write, starts at end of file if file exists; otherwise, creates a new file for readingand writing.

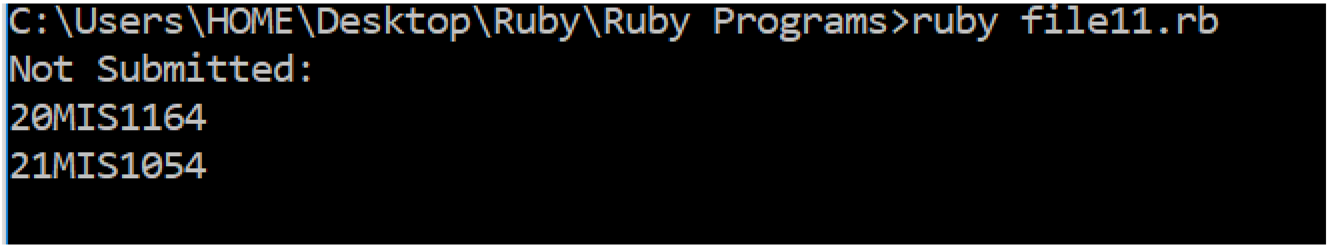
```
File.open("testfile", "r") do |file|
#... process the file
end # << file automatically closed here
```

Reads a line from standard input:

```
while line = gets #read the file line by line.
puts line
end
% ruby copy.rb:
– These are lines
– These are lines
– that I am typing
– that I am typing
– ^D
```

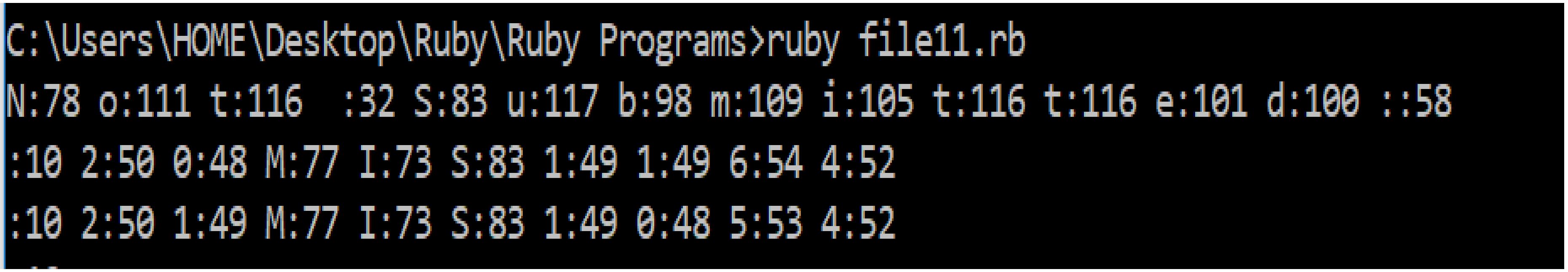
```
% Passing file name as argument:
% ruby copy.rb testfile
This is line one
This is line two
This is line three
And so on...
```

- Finally, we can explicitly open the file and read from it:
- File.open(" Rubyda1studentnames.txt") do |file|
- while line = file.gets
- puts line
- end
- end



```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby file11.rb
Not Submitted:
20MIS1164
21MIS1054
```

- The chr method converts an integer to the corresponding ASCII character:
- File.open("testfile") do |file|
- file.each_byte {|ch| print "#{ch.chr}:#{ch} " }
- end #For example, if the file contains the letter A, ch would be 65 (the ASCII value for A), and ch.chr would be 'A'



```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby file11.rb
N:78 o:111 t:116  :32 S:83 u:117 b:98 m:109 i:105 t:116 t:116 e:101 d:100 ::58
:10 2:50 0:48 M:77 I:73 S:83 1:49 1:49 6:54 4:52
:10 2:50 1:49 M:77 I:73 S:83 1:49 0:48 5:53 4:52
```

Get the original newlines visible using String#dump

```
File.open("testfile") do |file|
file.each_line {|line| puts "Got #{line.dump}" }
end
```

```
Got "This is line one\n"
Got "This is line two\n"
Got "This is line three\n"
```

Iterator with the autoclosing block feature:

IO.foreach("testfile") {|line| puts line }

- IO.foreach method takes the name of an I/O source, opens it for reading, calls the iterator once for every line in the file, and then closes the file automatically.

Retrieve an entire file into a string or into an array of lines:

# read into string

- str = IO.read("testfile")
- str.length # => 66
- str[0, 30] # => "This is line one\nThis is line "

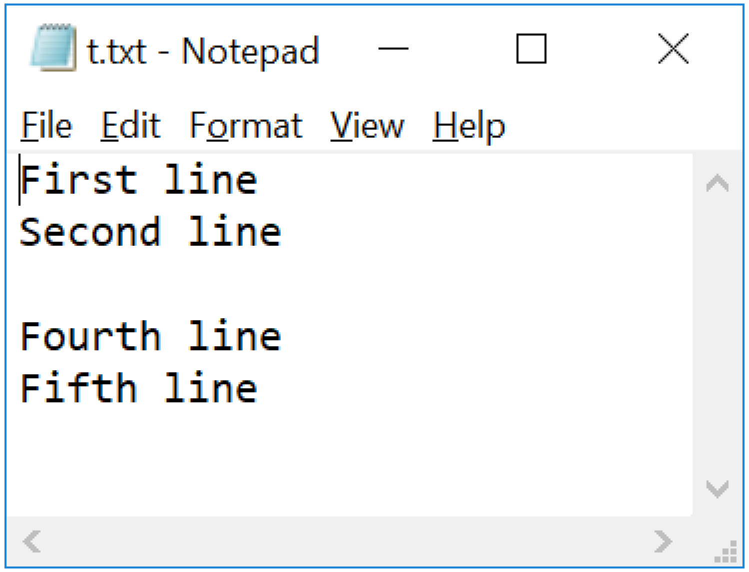
# read into an array

- arr = IO.readlines("testfile")
- arr.length # => 4
- arr[0] # => "This is line one\n"

Write to a File

- text = <<~EOT
-   First line
-   Second line

-   Fourth line
-   Fifth line
- EOT
- File.write('t.txt', text)



```
First line
Second line

Fourth line
Fifth line
```

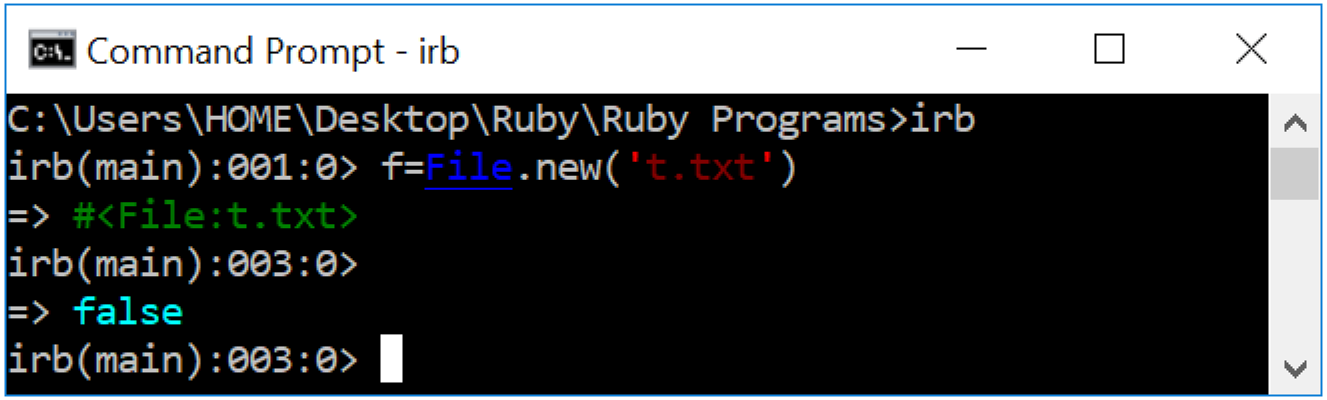
Writing to files

- # Note the "w", which opens the file for writing
- File.open("output.txt", "w") do |file|
- file.puts "Hello"
- file.puts "1 + 2 = #{1+2}"
- end
- # Now read the file in and print its contents to STDOUT
- puts File.read("output.txt")
- produces:
- Hello
- 1+ 2 = 3

```
require 'stringio'
ip = StringIO.new("now is\n the time\n tolearn\n Ruby!")
op = StringIO.new("", "w")
ip.each_line do |line|
op.puts line.reverse
end
op.string # => “\n si won \n \n emit eht \n \n nrael ot \n !ybuR \n”
```

End-of-Stream

- You can query whether a stream is positioned at its end: “.eof?”
- f = File.new('t.txt')
- f.eof? # => false
- f.tell # => 0 #tells the current poition

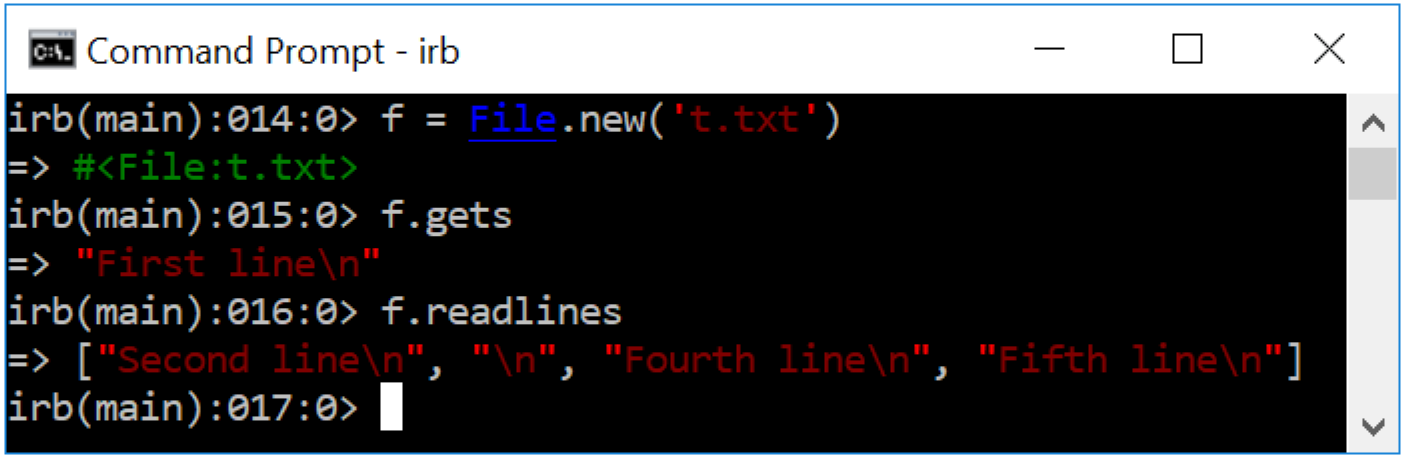


```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>irb
irb(main):001:0> f=File.new('t.txt')
=> #<File:t.txt>
irb(main):003:0>
=> false
irb(main):003:0>
```

- f.seek(0, :END) # => 0 # Repositions to stream end.
- Repositions the stream to its end plus the given offset. [end + 0 = end].
- f = File.open('t.txt')
- f.tell # => 0
- f.seek(0, :END) # => 0
- f.tell # => 52
- f.seek(-20, :END) # => 0
- f.tell # => 32
- f.seek(-40, :END) # => 0
- f.tell # => 12
- f.close

Line IO

- You can read an IO stream line-by-line using:
- Gets: Returns the next line.
- Readlines: Returns all remaining lines in an array.



```
irb(main):014:0> f = File.new('t.txt')
=> #<File:t.txt>
irb(main):015:0> f.gets
=> "First line\n"
irb(main):016:0> f.readlines
=> ["Second line\n", "\n", "Fourth line\n", "Fifth line\n"]
irb(main):017:0>
```

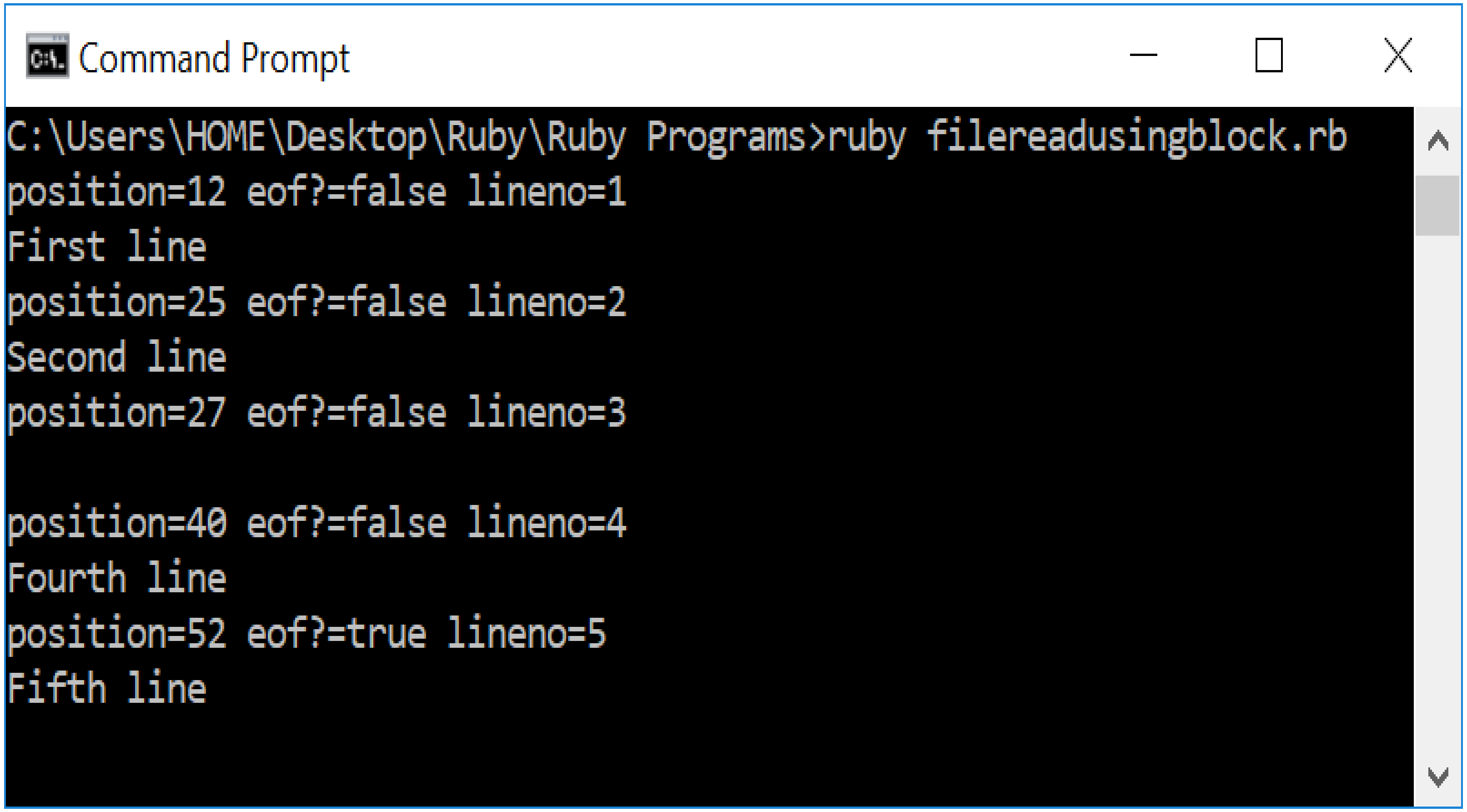
Line Number

- A new stream is initially has line number zero (and position zero); method rewind resets the line number (and position) to zero.
- f = File.new('t.txt')
- f.lineno   # => 0
- f.gets# => "First line\n"
- f.lineno   # => 1
- f.rewind
- f.lineno   # => 0
- f.close

```
File.open('t.txt') do |f| #f – File Object
  f.each_line do |line| #each line in object f
                    p                    "position=#{f.pos}eof?
    =#{f.eof?}lineno=#{f.lineno}"
#p similar to puts #
    puts line
  end
end
```

# ‘do’ keyword begins a block, which takes a single argument f representing the file object that was opened.

```
File.open(‘t.txt’) do |f|
# code that operates on the file object f
f.each_line do |line|
# code that operates with the value inside line
end
```



```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby filereadusingblock.rb
position=12 eof?=false lineno=1
First line
position=25 eof?=false lineno=2
Second line
position=27 eof?=false lineno=3

position=40 eof?=false lineno=4
Fourth line
position=52 eof?=true lineno=5
Fifth line
```

Read and Create

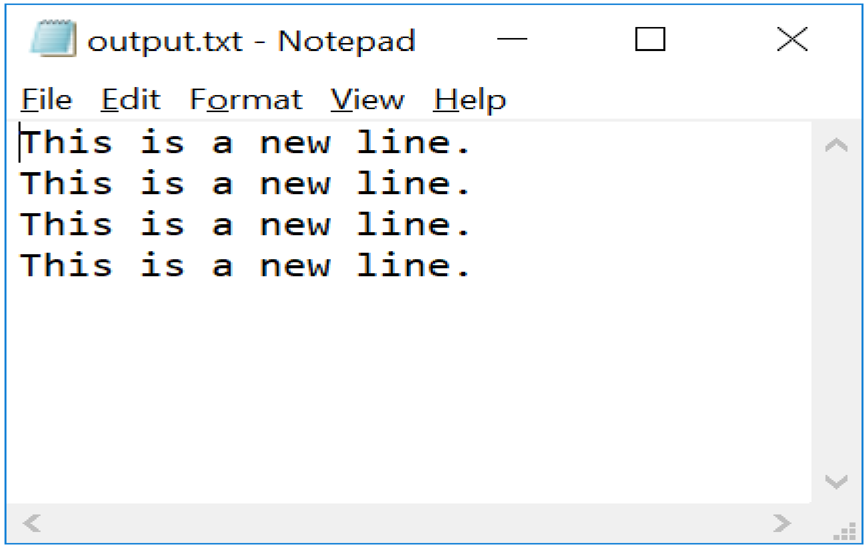
- File.read('t.txt')
- # => "First line\nSecond line\n\nThird line\nFourth line\n"
- File.read('t.txt').size # => 47
- Create: File.new('we.txt','wx') #File Name, Write and Create.
- file = File.open("output.txt", "w")
- # create output object for file
- file.write("Hello, world!") # write string to file
- file.close # close file

Directory

- dir =Dir.new("C:/Users/HOME/Desktop/Ruby/")
- entries = dir.entries
- [".", "..", "Book1.xlsx", "Module 1 and 2.pptx", "Module 3.pptx", "Rubby Syllabus.pdf", "Ruby Programs", "~$Module 3.pptx"]

File Operations

- Append:
- File.open("output.txt", "a") do |file| file.puts("This is a new line.")
- end
- #I ran for 4 times using irb.



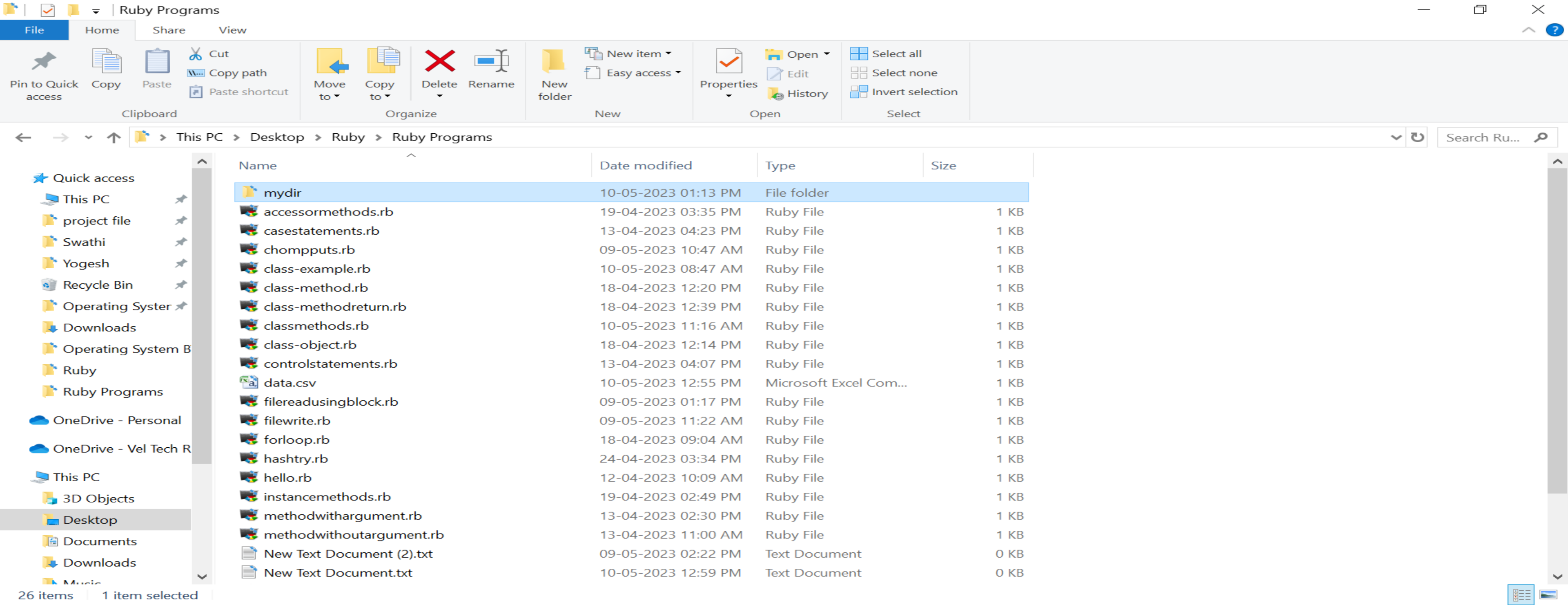
```
This is a new line.
This is a new line.
This is a new line.
This is a new line.
```

- Checking if a file exists:

- if File.exist?(“output.txt")
- puts "The file exists."
- else
- puts "The file does not exist."
- end
- #O/P: The file exists

- <u>Renaming a file:</u>
- File.rename("oldname.txt", "newname.txt")
- <u>Deleting a file:</u>
- File.delete("filename.txt")
- <u>Copying a file:</u>
- FileUtils.cp("sourcefile.txt", "destinationfile.txt")
- <u>Reading a CSV file and processing its contents:</u>
- require 'csv‘
- CSV.foreach("data.csv") do |row|
- puts row[0] # prints the first column of each row
- end

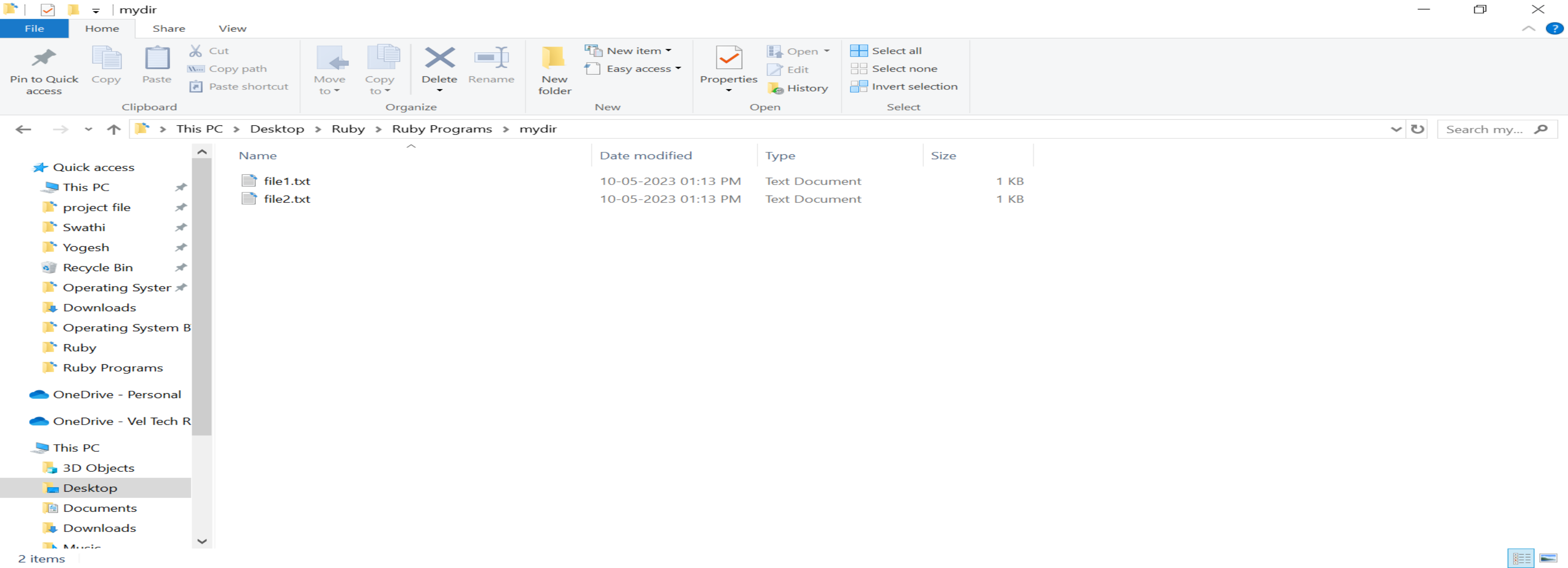
- <u>Creating a directory and writing files to it</u>
- Dir.mkdir("mydir")
  File.open("mydir/file1.txt", "w") do |file|
  file.write("This is file 1.")
- end
- File.open("mydir/file2.txt", "w") do |file|
  file.write("This is file 2.")
- end



This PC > Desktop > Ruby > Ruby Programs

Name	Date modified	Type	Size
mydir	10-05-2023 01:13 PM	File folder	
accessormethods.rb	19-04-2023 03:35 PM	Ruby File	1 KB
casestatements.rb	13-04-2023 04:23 PM	Ruby File	1 KB
chompputs.rb	09-05-2023 10:47 AM	Ruby File	1 KB
class-example.rb	10-05-2023 08:47 AM	Ruby File	1 KB
class-method.rb	18-04-2023 12:20 PM	Ruby File	1 KB
class-methodreturn.rb	18-04-2023 12:39 PM	Ruby File	1 KB
classmethods.rb	10-05-2023 11:16 AM	Ruby File	1 KB
class-object.rb	18-04-2023 12:14 PM	Ruby File	1 KB
controlstatements.rb	13-04-2023 04:07 PM	Ruby File	1 KB
data.csv	10-05-2023 12:55 PM	Microsoft Excel Com...	1 KB
filereadusingblock.rb	09-05-2023 01:17 PM	Ruby File	1 KB
filewrite.rb	09-05-2023 11:22 AM	Ruby File	1 KB
forloop.rb	18-04-2023 09:04 AM	Ruby File	1 KB
hashtry.rb	24-04-2023 03:34 PM	Ruby File	1 KB
hello.rb	12-04-2023 10:09 AM	Ruby File	1 KB
instancemethods.rb	19-04-2023 02:49 PM	Ruby File	1 KB
methodwithargument.rb	13-04-2023 02:30 PM	Ruby File	1 KB
methodwithoutargument.rb	13-04-2023 11:00 AM	Ruby File	1 KB
New Text Document (2).txt	09-05-2023 02:22 PM	Text Document	0 KB
New Text Document.txt	10-05-2023 12:59 PM	Text Document	0 KB

26 items 1 item selected



This PC > Desktop > Ruby > Ruby Programs > mydir

Name	Date modified	Type	Size
file1.txt	10-05-2023 01:13 PM	Text Document	1 KB
file2.txt	10-05-2023 01:13 PM	Text Document	1 KB

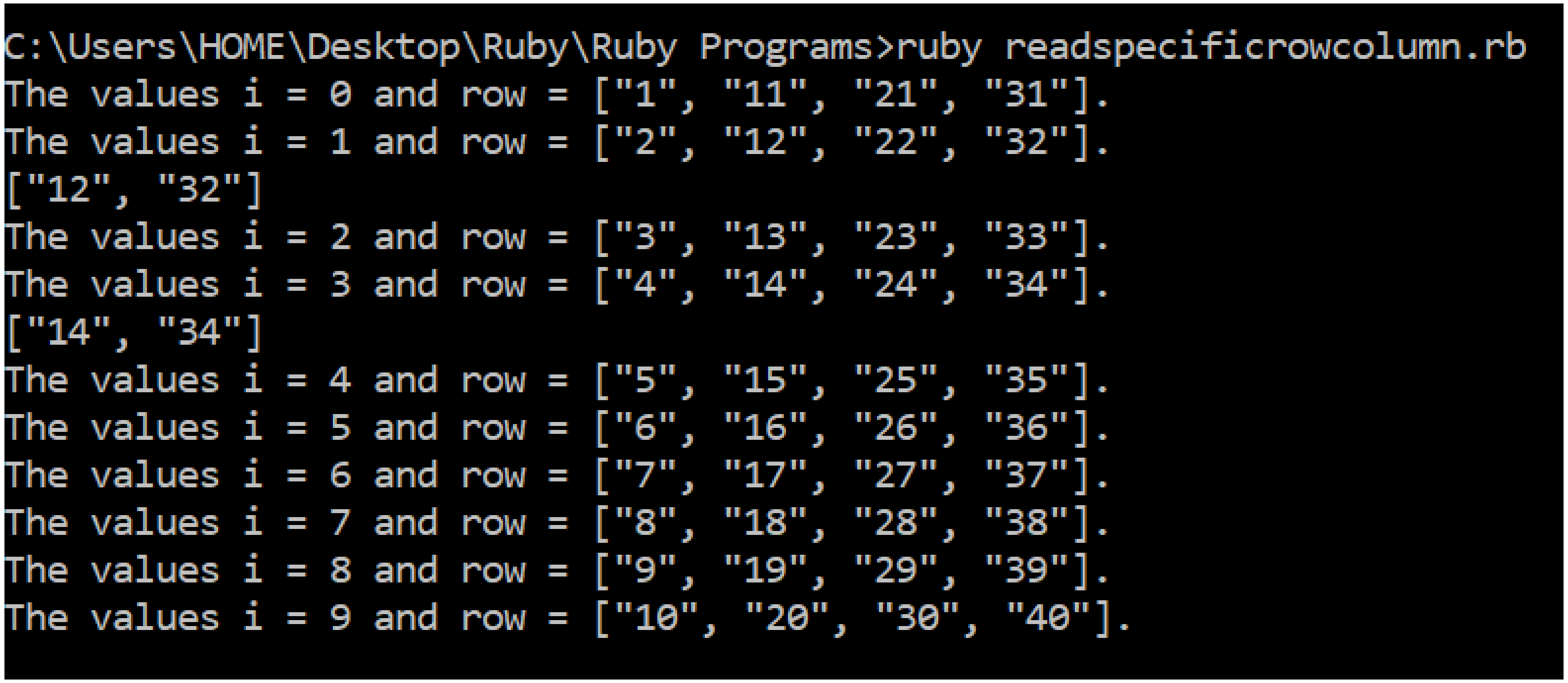
2 items

Read Specific rows and columns

- require 'csv'
- #We need values from 1st and 3rd row further in those rows we need only 1st and 3rd columns data.
- row_indices = [1, 3]
- column_indices = [1, 3]

```
• CSV.foreach("data.csv").with_index do |row, i|
• puts "The values i = #{i} and row = #{row}."
•   if row_indices.include?(i)
•     selected_columns = row.values_at(*column_indices)
•     puts selected_columns.inspect
•   end
• end
```

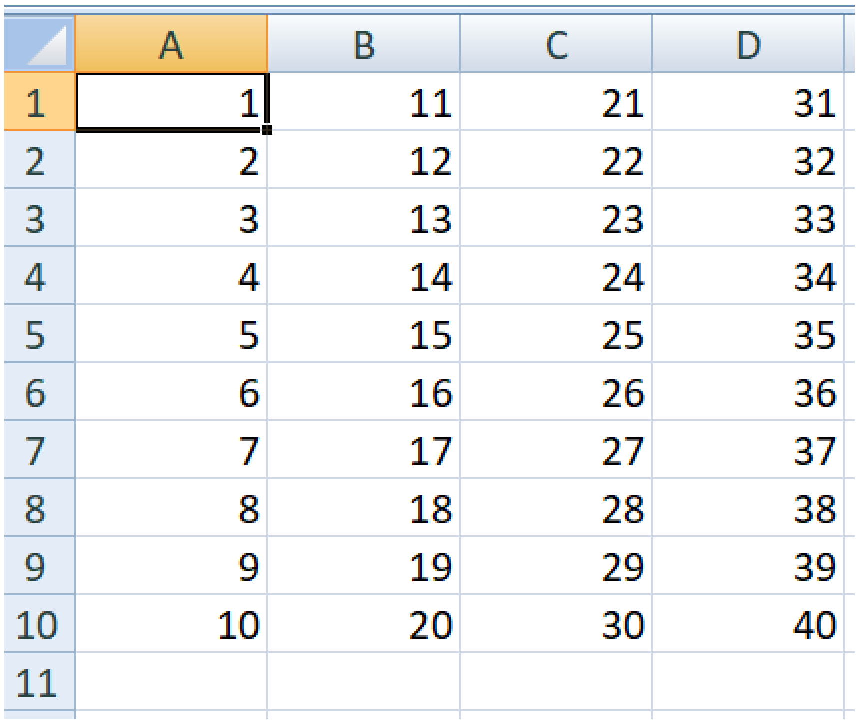
- #row and column indices start at 0.
- #.with_index - can be useful when you need to perform operations based on the position of each element, such as selecting specific rows or columns from a file.
- |row, i| - represents each row and i is index from 0,1, until last row 9.



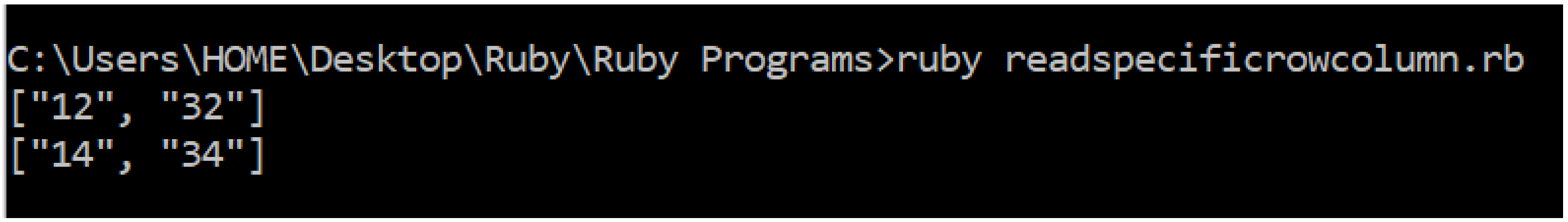
```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby readspecificrowcolumn.rb
The values i = 0 and row = ["1", "11", "21", "31"].
The values i = 1 and row = ["2", "12", "22", "32"].
["12", "32"]
The values i = 2 and row = ["3", "13", "23", "33"].
The values i = 3 and row = ["4", "14", "24", "34"].
["14", "34"]
The values i = 4 and row = ["5", "15", "25", "35"].
The values i = 5 and row = ["6", "16", "26", "36"].
The values i = 6 and row = ["7", "17", "27", "37"].
The values i = 7 and row = ["8", "18", "28", "38"].
The values i = 8 and row = ["9", "19", "29", "39"].
The values i = 9 and row = ["10", "20", "30", "40"].
```

- Note that the values_at method is used to select specific columns from a row, and the * operator is used to convert the column_indices array into arguments to the method.

- .inspect – method prints in string format, if this is not given integer value will be returned.



	A	B	C	D
1	1	11	21	31
2	2	12	22	32
3	3	13	23	33
4	4	14	24	34
5	5	15	25	35
6	6	16	26	36
7	7	17	27	37
8	8	18	28	38
9	9	19	29	39
10	10	20	30	40
11				



```
C:\Users\HOME\Desktop\Ruby\Ruby Programs>ruby readspecificrowcolumn.rb
["12", "32"]
["14", "34"]
```

Encrypt and Decrypt

```
require 'openssl‘
# Encryption
def encrypt(plaintext, key, iv)
  cipher = OpenSSL::Cipher.new('AES-256-CBC')
  cipher.encrypt
  cipher.key = key
  cipher.iv = iv

  encrypted = cipher.update(plaintext) + cipher.final
  return encrypted
end
```

Encryption Process

- The encrypt method takes three parameters: plaintext (the message to be encrypted), key (the encryption key), and iv (the initialization vector).
- A new cipher object is created using the AES-256-CBC algorithm.
- The cipher object is set to encryption mode using cipher.encrypt.
- The encryption key (key) and initialization vector (iv) are set for the cipher object.
- The update method is used to encrypt the plaintext. The result is stored in the encrypted variable.
- Finally, cipher.final is called to finalize the encryption process, and the encrypted variable is returned.

```
# Decryption
def decrypt(ciphertext, key, iv)
  cipher = OpenSSL::Cipher.new('AES-256-CBC')
  cipher.decrypt
  cipher.key = key
  cipher.iv = iv

  decrypted = cipher.update(ciphertext) + cipher.final
  return decrypted
end
```

```
# Main
plaintext = 'Hello, World!'
key = OpenSSL::Random.random_bytes(32) # 256-bit key
iv = OpenSSL::Random.random_bytes(16) # 128-bit IV

# Encryption
encrypted = encrypt(plaintext, key, iv)
puts "Encrypted: #{encrypted}"

# Decryption
decrypted = decrypt(encrypted, key, iv)
puts "Decrypted: #{decrypted}"
```

Compress a File

```
require 'zlib'

def compress_file(input_file, output_file)
  Zlib::GzipWriter.open(output_file) do |gzip|
    File.open(input_file, 'rb') do |input|
      while (data = input.read(4096))
#input.read(4096) reads up to 4096 bytes of data from the input source
    and assigns it to the variable data.
        gzip.write(data)
      end
    end
  end
end
```

Decompress a File

```
def decompress_file(input_file, output_file)
  Zlib::GzipReader.open(input_file) do |gzip|
    File.open(output_file, 'wb') do |output|
      while (data = gzip.read(4096))
        output.write(data)
      end
    end
  end
end
        # Main:
        compress_file('input.txt', 'compressed.txt.gz')
        decompress_file('compressed.txt.gz', 'decompressed.txt')
```

Search and Replace

```
def search_and_replace(input_file, output_file, search_pattern,
    replace_pattern)
  File.open(output_file, 'w') do |output|
    File.foreach(input_file) do |line|
      output.puts line.gsub(search_pattern, replace_pattern)
    end
  end
end

# Example usage:
search_and_replace('input.txt', 'output.txt', /hello/, 'hi')
```