Ruby programming language

This handout aims to give only a brief introduction to Ruby enough to help you understand the Rails web application framework.

Rails is build on the top of ruby. Ruby was created by Yukihiro Matsumoto, and people just refer to him as Matz online. His reason to create ruby.

"I wanted to create a scripting language that was more powerful than Perl, more object oriented than Python,"

"Maximize programmer efficiency, in other words, the productivity of the programmer."

"I hope to see Ruby help every programmer in the world to be productive and to enjoy programming and to be happy."

That is the primary purpose of the ruby programming language. We can execute and test ruby programming language following three ways

1. Run terminal and type **$ irb**
2. Save your code in a file with extension *.rb*
3. Type **$ rails console**

Message display and calculator

Type irb or rails console on terminal and apply the following commands.

```
> "Hello World"
> puts "Hello world"
```

```
> 3 + 2
> 3 * 2
> 3**2
```

Comments

we use # sign for the ruby comments.

```
# working on ruby programming language
> 17 + 10 # integer addition
```

0.0.1 Strings

We have already seen some string and related operations on sting in section variables. We can create strings either with single or double quotes. We can insert arbitrary ruby expression using string interpolation. Typing variables inside message quotes called the string interpolation

```
> "Hello ruby "
> ""  #Any empty string
```

Concatenation

Concatenating string in Ruby programming.

```
> "Hello " + "Ruby"
```

Variables

There is no datatypes for variables in ruby programming. We can easily define variable by typing variable name

```
> a = 3 * 2
> b = "message"
> first_name = "Ali"
> last_name = "Ahmed"
```

Some operation with strings

```
> first_name +  " " + last_name
```

Interpolation

Build-in functions

Lets try some build-in function with ruby

```
> Math.sqrt(81)
> Math.cos(2)
```

Strings, symbols and regular expressions

0.0.2 Objects and Message Passing

```
> "#{first_name} #{last_name}"  # string interpolation
> "360 degrees =#{2 * Math :: PI } radians "
```

Some operations on strings

```
> "foobar".length
> "foobar".empty
> "foobar".reverse
> name = "Ahmed Anwar"
> name[0]
> name[3..5]
> "Hi " + name[0..4]
> "37".to_s
```

Symbols

Regular expressions

Control structure and iteration

Everything in ruby is an expression and evaluates to some thing

Conditional Execution

The basic if condition in ruby

```
if expression
  code
end
```

where code is executed if and only if the conditional expression evaluates to something other than `false` or `nil`. and if - else.

```
if expression
  code
else
  code
end
```

and if-elsif

```
if expression
  code
elsif
  code
else
  code
end
```

```
  a = 5
  if a > 10
    puts "a is greater than"
  else
    puts "a is less than or equal to 10"
  end
```

one more example

```
>> s = "foobar"
>> if s.empty?
>>  "The string is empty"
>> else
>>  "The string is nonempty"
>> end
>> "The string is nonempty"
```

Iteration

1. The for / in loop over an enumerated collection

```
for var in collection do
  body
end
```

```
> ary = [19, 11, 32]
> for a in ary do
>  puts a
> end
```

2. While loop

```
while condition do
  body
end
```

Collections, Blocks and iterators

Ruby contains a number of collection classes – these are classes that can be used to store collections of data. The most important collection classes in Ruby are **Array** and **Hash**. An example previously experienced.

```
def index
  @users = User.all
end
```

Arrays

Arrays hold collections of object references that can be indexed (zero-based).

```
> a = [33.3, "hi" , 2]
> a[0]
```

Some operations on arrays

```
> a[1..2].sort
> a.include?
> a.reverse
> a.length
> a.first
> a.last
> a << 5
> a[-1]
```

Hashes

A Hash is an associative array (key-value pairs) where the key and value may be any objects, separated by the =>symbol. You index into a hash using keys.

```
phone = { 'home' => 1, 'mobile' => 2, 'work' => 3}
```

```
phone = {:home => 1, :mobile => 2, :work => 3}
> phone[:home] # => 1
> phone.key(1) # => : home
> phone.key?(:home) # => true
> phone.value?(1) # => true
```

Nested collection

```
> ary = [[ " red " , " green " , " blue " ], [1,2,3], [ " Alpha " , "
  Beta " , " Gamma " ]]
> ary[2][1] # => " Beta "
```

Blocks

iterators

```
for each item in the collection # this is psuedocode
  yield (item)
end
```

```
> a = [33.3, " hi " , 2]
> a.each {|element| puts element}
33.3
"hi"
2
=> [33.3, " hi " , 2]
```