Good info from SoloLearn app

Welcome to Ruby!

Ruby is a dynamic, object-oriented, general-purpose programming language.

It is ranked among the top 10 programming languages worldwide. Much of its growth is attributed to the popularity of software written in Ruby, particularly the Ruby on Rails web framework.

A quote from its creator, Yukihiro "Matz" Matsumoto: "Ruby is simple in appearance, but is very complex inside, just like our human body."

Matsumoto has said that Ruby is designed for programmer productivity and fun, following the principles of good user interface design.

In Ruby, everything (even a simple number) is an object. We will learn more about objects in the coming lessons.

Ruby is also completely free. Not only free of charge, but also free to use, copy, modify, and distribute.

// \*\*\*\*\*\* ПРЕВОД \*\*\*\*\*\*\*\*///

Добре дошли в Ruby!

Ruby е динамичен, обектно-ориентиран език за програмиране с общо предназначение.

Класиран е сред 10-те най-добри езика за програмиране в света. Голяма част от растежа му се дължи на популярността на софтуера, написан на Ruby, особено на уеб рамката Ruby on Rails.

Цитат от неговия създател, Юкихиро "Мац" Мацумото: "Рубинът е прост на външен вид, но е много сложен отвътре, точно като нашето човешко тяло."

Мацумото каза, че Ruby е предназначен за производителност и забавление на програмисти, следвайки принципите на добрия дизайн на потребителския интерфейс.

В Ruby всичко (дори просто число) е обект. Ще научим повече за обектите в следващите уроци.

Ruby също е напълно безплатен. Не само безплатно, но и безплатно за използване, копиране, модифициране и разпространение.

------------------------------------------------------------------------------------------------------------------------------------------

In Ruby everything is **an object**

------------------------------------------------------------------------------------------------------------------------------------------

First program:

***puts*** “Hello World” -> This code will output the text "Hello World" to the screen.

All text values (strings) must be enclosed in single or double **quotes**.

Another method that can be used to display output to the screen is ***print.***

This code displays the same output as before, except that the **puts** automatically adds a new line after the output, while **print** does not.

Този код показва същия изход като преди, с изключение на това, че puts автоматично добавя нов ред след изхода, докато print не го прави.

Code:

puts "Hi"

print "there"

print "Ruby"

On the console:

Hi

thereRuby

------------------------------------------------------------

Code to output “My first program” to the screen.

print “My “

print “first “

puts “program”

-------------------------------------------------------------------------------------------------------------

Comments are lines of annotation within Ruby code that are ignored at program runtime.   
In Ruby, the **hashtag symbol** is used to create a single-line comment.

Example:

# printing some text

puts “Hi there”

Output:

Hi there

----------------------------------------------------------

You can also create multi-line comments.   
Everything between the **=begin** and **=end** reserved words are considered a comment:

=begin

   This comment

   spans multiple

   lines

=end

puts "Hello"

Output:

Hello

**Variables**

A **variable** is a named storage location for a value. It is called variable because the information stored in that location can be changed when the program is running.   
To assign a variable a value, use the **equal sign**.

x = 8

This assignment statement declares a variable named **x** and gives it the value 8. The equal sign is called the assignment operator.   
We can later use the variable name to access its value. For example, to output the value stored in the variable, we can use **puts** or **print** and refer to the variable name:

x = 8

puts x

Output:

8

Variable names may consist of alphanumeric characters and the underscore character (\_), but **cannot** begin with a capital letter or a number.

**Constants**

Variables beginning with a capital letter are called **constants**.   
The value of a constant variable cannot be changed once it has been assigned.

Code:

MyNum = 42

# Trying to change the value produces a warning

MyNum = 8

Output:

./Playground/file0.rb:4: warning: already initialized constant MyNum

./Playground/file0.rb:1: warning: previous definition of MyNum was here

*What type of variable is Y?*

x = 8

Y = x

puts Y

*Answer:*

Constant variable

**Data Types**  
  
All variables in Ruby can be of all **data types**.   
Ruby automatically determines data type by the value assigned to the variable.

**For example:**

x = 42 # integer   
y = 1.58 # floating point value   
z = "Hello" # string

You can reassign a different value to a variable at any time.   
To insert the value of a variable into a double quote string (a string is a sequence of characters, such as "Hello"), use the # symbol and curly brackets with the variable name.

**For example:**

age = 42

puts "He is #{age} years old"

Output:

He is 42 years old

It is good practice to use the same data type in a single variable all over your application. It helps you to avoid mistakes.

**Doing Math**  
  
Math is an important part of programming. Ruby supports the following **arithmetic operators**:

x = 5

y = 2

#Addition

puts x+y

#outputs 7

#Subtraction

puts x-y

#outputs 3

#Multiplication

puts x\*y

#outputs 10

#Division

puts x/y

#outputs 2

When you divide two integer values, the result will be an integer, as shown in the above example. If you want to have a floating point result, one operand must be a floating point value:   
x = 5.0   
y = 2   
puts x/y # outputs 2.5

***What is the output of this code?***

a = 25

b = 8

x = a/b

puts x + b

***Output:***

11

**Modulus Operator**

The **modulus** operator, represented by the percentage symbol (%), represents the remainder of a division operation.   
**For example:**

**x = 9**

**y = 5**

**puts x%y**

**Output: 4**

**------------------------------------------------------------------**

9 divided by 5 is 1 with a remainder of 4

**------------------------------------------------------------------**

***What is the value of z?***

a = 13

b = a – 8

z = a%b

***Answer***: 3

**------------------------------------------------------------------**

**Exponent Operator**

The \*\* represents the **exponent** operator for raising a number to a power to perform exponentiation.   
**For example:**

a = 2

b = 5

puts a\*\*b

# this raises 2 to the power of 5 and outputs 32

-------------------------------------------------------------------------------

All operators can also be used with floating point values.

***What is the result of 3\*\*2 ?***

***Answer:*** 9

**Shorthand Assignment Operators**

All of the arithmetic operators have corresponding shorthand forms for assignment.   
For example, a = a + 8 can be written as a += 8.   
The same applies to the other operators:

x += y # x=x+y   
x -= y # x=x-y   
x \*= y # x=x\*y   
x /= y # x=x/y   
x %= y # x=x%y   
x \*\*= y # x=x\*\*y

These are called **self-assignment** operators, as they perform an assignment and an arithmetic operation at the same time.

***What is the output of this code?***

a = 8

a /= 3

puts 4%a

***Output: 0***

**Parallel Assignment**

Ruby also supports parallel assignment of variables. This enables multiple variables to be initialized with a single line of code.   
**For example:**

x = 10   
y = 20   
z = 30

may be more quickly initialized using parallel assignment:

x, y, z = 10, 20, 30

puts x, y, z

Output:

10

20

30

Parallel assignment is also useful for swapping the values held in two variables:   
a, b = b, a

Swap the values of the variables using a single statement:

x = 10

y = 42

x, y = y, x

----------------------------------------------------------------------------------------------------

**Operator Precedence – Приоритет на оператора**   
  
Ruby evaluates a mathematical expression using an order of operations that is based on operator precedence. Exponentiation has the highest precedence followed by multiplication, division, and modulus from left to right, and then addition and subtraction from left to right.   
You can change the order of operations by using parentheses.   
**For example:**

x = (3+2) \* 4

puts x

output: 20

What is the output of this code?

5-2\*3+3

Output: 2

**Strings**

As mentioned in the previous lessons, a **string** is text between single or double quotation marks.   
However, some characters can't be directly included in a string. For instance, single quotes can't be directly included in a single quote string, because this would designate the end of the string. Characters like these can be included in a string by using an **escape sequence**, which is indicated by a **backslash (\)**

text = 'Ruby\'s syntax is fun'

puts text

Outputs: Ruby’s syntax is fun

A string formed with double quotation marks can also include the **\n** escape sequence, which represents a new line.

Text = “Hello \n World”

puts text

Outputs:

Hello

World

Only the \' and \\ escape sequences can be used with single quote strings.

text = ‘Welcome’

puts text

**String Interpolation**

You can embed any Ruby expression inside a double quote string using **#{ }**, just as you did with variable names. Ruby evaluates the placeholders and replaces them with values:

a = 5

b = 2

puts “The sum is #{a+b}”

# outputs “The sum is 7”

Note that there is no space between the hash mark (#) and the opening curly brace ({). In case of a space, it will be interpreted as literal text.

a = 15

b = 3

res = “The result is #{a / b}”

puts res

**Concatenation**  
  
Strings can be joined using the + in a process called **concatenation**.   
When concatenating strings, it doesn't matter whether they've been created with single or double quotes.

a = “Hi “

b = ‘there’

puts a+b

# output: Hi there

Even if your strings contain numbers, they are still added as strings rather than integers. Adding a string to a number produces an error, because even though they might look similar, they are two different entities: "1" is a string, whereas 1 is an integer.

**‘7’ + 2 produces an error**

**Repeating a String**

Strings can be repeated using the \* and an integer value.   
The order of the string and the integer does matter: the string has to come first.   
**For example:**

a = "abc"

puts a\*3

puts '5'\*4

**Output:**

abcabcabc

5555

**Strings can't be multiplied by other strings.**

**Puts ‘7’\*2 --🡪 outputs: 77**

**Input**  
  
To get input from the user in Ruby, you use the **gets** method, which returns what the user types as a string. To store the input for later use, you can assign the return value to a variable.   
**For example:**

**x = gets**

**puts x**

**Output: Каквото му въведа, това ще изведе на конзолата**

**gets** gets a line of text, including the new line at the end. If you do not want to include the new line, use the **gets.chomp** method:

puts "Enter your name"

name = gets.chomp

puts "Welcome, #{name}"

Output:

Enter your name

Welcome, Tsvety

The value of the input is a string. To convert it to an integer, you can use the **gets.to\_i** method.

temp = gets

puts “You entered #{temp}”

***What is the output of this code?***

temp = “7”

temp += “2”

puts temp

**Answer: 72**

x = 42

x += 7

puts x

**comment this text:**

=begin

Text

Text

Text

=end

name = “Amy”

msg = “Hi, my name is #{name}”

puts msg

***What is the output?***

a= 4

puts a\*\*2

Answer: 16

***Module 2: Control Structures***

**Booleans**

In Ruby, there are two Boolean values: **true** and **false**.

isOnline = true   
userIsAdmin = false

Another value that you will often encounter is **nil**. It shows the absence of value.

A Boolean can be: true and false

If you try to evaluate a value other than **true** or **false** as a Boolean, Ruby will automatically treat it as a Boolean.   
When this is done, a non-Boolean value that evaluates to true is called "**truthy**" and a non-Boolean value that evaluates to false is called "**falsey**".   
  
In Ruby only **false** and **nil** are falsey. Everything else is truthy (even 0 is truthy).

Only **true** and **false** are Booleans. **nil** is not a Boolean. 0 is not a Boolean. The string "Hello" is not a Boolean. However, in a context where a Boolean is expected, Ruby evaluates them as Boolean (**truthy** and **falsey**).

As a Boolean, the string “false” is: truthly

**Comparison**

A Boolean comparison using the = = operator returns **true** when two operands are equal, and **false** when they are not:

a = 5

b = 8

puts a == b

puts a == 5

Output:

false

true

Be careful not to confuse assignment (one equals sign) with comparison (two equals signs).

***What is the output of this code?***

puts “58” = = 5

***Answer:*** false

Another comparison operator, the **not equal operator** (!=), evaluates to true if the items being compared aren't equal, and false if they are.   
**For example:**

**a = 8**

**b = 7**

**puts a!= b**

**Output: true**

***What is the output of this code?***

a = 7

b = a/2

puts b == (a - 4)

***Output: true***

Ruby also has operators that determine whether one value is greater than or less than another. These operators are > and < respectively. Similarly, the greater than or equal to, and less than or equal to operators are >= and <=.   
**For example:**

puts 12 > 8 #true

puts 5 < 2 #false

puts 5 >= 5.0 #true

puts 3 <= 6 #true

There is also the **.eql?** method, which results in true only if both arguments have the **same type and equal values**.   
**For example:**

puts 3 == 3.0

puts 3.eql?(3.0)

Output:

true

false

3.eql?(3.0) is **false** because 3 is an integer and 3.0 is a float.

Greater than and less than operators can also be used to compare strings **lexicographically** (the alphabetical order of words is based on the alphabetical order of their component letters).

***What is the output of this code?***

puts 7 > 7.0

***Answer: false***

**if Statements**  
  
You can use an **if** expression to execute code when a certain condition holds.   
If a conditional expression evaluates to **true**, the **if** code is carried out. Otherwise, the code is ignored.   
**For example:**

**a = 42**

**if a > 7**

**puts “Yes” output: Yes**

**end**

The condition a > 7 is evaluated. When it is **true**, the statements inside the **if** are executed and the program outputs Yes.   
You can have multiple statements inside a single **if** expression.

The **end** keyword is required to indicate the end of the **if**.