PROGRAMMING BASICS

BUILT-IN OBJECTS

```
"Sponge Bob".class                  #=> String
12.class                            #=> Integer
3.14.class                          #=> Float
["Sponge Bob", 12, 3.14].class      #=> Array
true.class                          #=> TrueClass
false.class                         #=> FalseClass
(1..100).class                      #=> Range
```

STRING

```
"yipi yeah".upcase   #=> "YIPI YEAH"
"Hello" == 'Hello'   #=> true
```

Interpolation

```
'two: #{1 + 1}'      #=> "two: #{1 + 1}"
"two: #{1 + 1}"      #=> "two: 2"
```

Conversion to integer

```
'1984'.class         #=> String
'1984'.to_i          #=> 1984
'1984'.to_i.class    #=> Integer
```

INTEGER

Used to be `Fixnum` in Ruby versions < 2.4.0

```
# Standard arithmetic
1 + 2       #=> 3
2 * 4       #=> 8
# Built-in methods
20.even?    #=> true
20.odd?     #=> false
```

Conversion to string

```
1984.to_s     #=> "1984"
```

FLOAT

```
3.1416.truncate    #=> 3
1.618.round        #=> 2
```

ARRAY

```
['Sponge Bob', 12, 3.14].size       #=> 3
['Huey', 'Dewey', 'Louie'].sort     #=> ["Dewey", "Huey", "Lou
[3, 5, 1].sort                      #=> [1, 3, 5]
```

Shortcut for array of strings

```
%w[Huey Dewey Louie]  #=> ["Huey", "Dewey", "Louie"]
%w(Huey Dewey Louie)  #=> ["Huey", "Dewey", "Louie"]
```

RANGE

```
(1..10).to_a       #=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
(1...10).to_a      #=> [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

SPECIAL VALUES

```
nil
```

```
false
true
```

Tomorrow: we'll use *booleans* to implement logic in our code

RUBY DOC IS YOUR FRIEND!

Want to perform some basic manipulation?

- shuffle an array
- capitalize a word
- split a word into an array of characters
- etc.

DO NOT RE-INVENT THE WHEEL

http://www.ruby-doc.org/core/

VARIABLES

Deeply understand them!

They are an **elementary block of programming**.

WHY VARIABLES?

Store values to re-use them

```
age = 17
puts "You are #{age} years old"

puts "Lucky you, it's your birthday!"
age = age + 1
puts "You are now #{age}"
```

ASSIGNING

```
1  city = “Paris”
2  population = 2000000
```

city	population
“Paris”	2000000

ASSIGNING

```
1  city = “Paris”
2  population = 2000000


3  city_details = “#{city}, France”
```

city	population
“Paris”	2000000

city	population	city_details
“Paris”	2000000	“Paris, France”

REASSIGNING / INCREMENTING

```
1  city = “Paris”
2  population = 2000000


3  city_details = “#{city}, France”


4  population = population + 30000
```

city	population
“Paris”	2000000

city	population	city_details
“Paris”	2000000	“Paris, France”

city	population	city_details
“Paris”	2030000	“Paris, France”

METHODS

PROBLEM

```
puts "Hello John!"
puts "Hello Paul!"
puts "Hello Ringo!"
```

We want to change `Hello` with `Hi` and `!` with a full stop `.`

SOLUTION: A METHOD

Factoring your code

```
def say_hi(name)
  return "Hi #{name}."
end

puts say_hi("Alex")      # => "Hi Alex."
puts say_hi("Edward")    # => "Hi Edward."
```

Don't Repeat Yourself (DRY)

WHY METHODS?

Apply some ruby code to **dynamic inputs**, over and over again

```ruby
def full_name(first_name, last_name)
  name = first_name.capitalize + " " + last_name.capitalize
  return name
end

puts full_name("boris", "paillard")   # => "Boris Paillard"
puts full_name("seb", "saunier")      # => "Seb Saunier"
```

COMBINE VARIABLES AND METHODS

```
def max(x, y)
  if x > y
    return x
  else
    return y
  end
end

first_number = 2
second_number = 5
largest_number = max(first_number, second_number)
puts largest_number   # => 5
```

PARAMETERS VS. ARGUMENTS

```
def new_population(population, births)
  return population + births
end
```

- `population` and `births` are **parameters**

```
puts new_population(2000000, 300)
```

- `2000000` and `300` are **arguments**
- the parameters are going to take the values of the arguments

METHOD WITH NO ARGUMENTS

Some methods may not need arguments

```
def tomorrow
  tomorrow_date = Date.today + 1
  return tomorrow_date.strftime("%B %d")
end

puts tomorrow   # => "October 4"
```

THE RETURN CONVENTION

A method returns the **last statement executed.**

```
def add(x, y)
  return x + y
end

# is the same as

def add(x, y)
  x + y
end
```

CONVENTIONS

Methods and variables in **snake_case**

```
good_method_or_variable_name
```

CONVENTIONS

a method ending with a `?` returns `true` or `false`

```
42.even? #=> true
42.odd?  #=> false
```

a method ending with a `!` is destructive or dangerous

```
text = 'Hello'

text.upcase
#=> "HELLO"

text
#=> "Hello"

text.upcase!
#=> "HELLO"
```

HAPPY HACKING!