Numbers

Integer

An integer is a whole number, like 1, 2, -5, etc. When you operate using only integers, Ruby will give you an Integer answer.

Float

A float is a number with decimal places, like 3.14, 1.5, 3.0, etc. When you operate with Floats Ruby gives you a Float answer

Strings

A group of characters (array of characters)

Concatenation

```
"Hello" + "Adam"
=> "Hello Adam"
```

String multiplication

```
"Hi" * 3
=> "HiHiHi"

"1" * 2
=> "11" # Ruby see's 1 as a string
```

More things you can do with strings

```
"hello".capitalize()
=> "Hello"
"hello".reverse()
=> "olleH"
"hello".next()
=> "Hellp"
"hello".upcase()
=> "HELLO"
"HeLLo".swapcase()
=> "hEllO"
```

Even more things you can do with strings

```
"hello".length()
=> 5
```

Arrays

literal constructor []

```
list = [1, 2, 3, 4, 5]
puts list

=> 1
=> 2
=> 3
=> 4
=> 5
```

explicitly calling ::new

```
list_two = Array.new
list_two.push("item")
list_two.push("another item")
puts list_two

=> "item"
=> "another item"
```

Variables

A variable is a name that Ruby associates with a particular object.

Creating a variable

```
school = "Lighthouse"
x = 10
y = 20.0
```

Working with variables

```
school = "Lighthouse"
city = "Toronto"
x = 10
y = 20.0
z = x + y
=> 30.0
place = school + " " + city
=> "Lighthouse Toronto"
```

Shortcuts

Example	Shortcut	Meaning
var = var + 2	var += 2	Add 2 to **var**
var = var - 3	var -= 3	Subtract 3 from **var**
var = var * 6	var *= 6	Multiply **var** by 6
var = var / 2	var /= 2	Divide **var** by 2
var = var** 3	var **=3	Cube **var**
var = var % 4	var %= 4	**var** modulo 4

Constants VS. Variables

Constants are like variables. Except that you are telling Ruby that their value is supposed to remain fixed. If you try to change the value of a constant Ruby will give you a warning. You define constants just like variables except the first character is capitalized.

```
Fullname = "Adam Dahan"
=> "Adam Dahan"
```

Note: Though Fullname is a "constant" it's value will still change. Being a constant just means that Ruby will give you a warning.

Hashes

```
# old hand
grades = { "Jane Doe" => 10, "Jim Doe" => 6 }
puts

=> { "Jane Doe" => 10, "Jim Doe" => 6 }

# new hand using symbols
options = { font_size: 10, font_family: "Arial" }
puts

=> { font_size: 10, font_family: "Arial" }

# assign values to keys of a hash
grades = Hash.new
grades["score"] = 9
puts grades["score"]

=> 9
```

Making Decisions

if expressions are used for conditional execution. The values false and nil are false, and everything else are true. Notice Ruby uses elsif, not else if nor elif.

Executes code if the conditional is true. If the conditional is not true, code specified in the else clause is executed.

if…else

```
x = 5

if x < 10
  puts "x is less than 10"
else
  puts "x is greater than 10"
end

=> "x is less than 10"
```

if…elsif…else

```
x = 5

if x > 10
  puts "x is greater than 10"
elsif x < 5
  puts "x is less than 5"
else
  puts "x is not greater than 10 or less than 5."
  puts "It must equal 5."
end

=> "x is not greater than 10 or less than 5."
=> "It must equal 5."
```

when (similar to switch in other langs)

```
obj = "hello"
case obj.class
when String
  p "It is a string"
when Fixnum
  p "It is a number"
else
  p "It is not a string"
end

=> "It is a string"
```

Flow Control

Loops in Ruby are used to execute the same block of code a specified number of times.

Loops (.times)

```
4.times do
  puts "Hello"
end

=> "Hello"
=> "Hello"
=> "Hello"
=> "Hello"
```

Loops (.times) – Counting

```
# Counting
count = 0
5.times do
  count += 1
  puts "Count = " + count.to_s
end

=> "Count = 1"
=> "Count = 2"
=> "Count = 3"
=> "Count = 4"
=> "Count = 5"

# Counting backwards
count = 10
10.times do
  count -= 1
  puts count
end

=> 9
=> 8
=> 7
=> 6
=> 5
=> 4
=> 3
=> 2
=> 1
=> 0
```

each loops

```
(0..2).each do |i|
   puts "Value of local variable is #{i}"
end

=> "Value of local variable is 0"
=> "Value of local variable is 1"
```

loop array with each

```
list = Array.new
list.push("item")
list.push("another item")

list.each do |item|
  p item
end
```

while loops

```
i = 0
num = 5

while i < num  do
   puts "Inside the loop i = #{i}"
   i +=1
end

=> "Inside the loop i = 0"
=> "Inside the loop i = 1"
=> "Inside the loop i = 2"
=> "Inside the loop i = 3"
=> "Inside the loop i = 4"
```

Methods

```
def say_name
  puts "Adam"
end

say_name
=> "Adam"

# method with parameter
def say(name)
  puts name
end

say("Adam")
=> "Adam"

# multiple parameters
def say(name, age)
  puts name, age
end

say("Adam", 27)
=> "Adam"
=> 27
```

Classes

```
class Foo
  def self.bar
    puts 'class method'
  end

  def baz
    puts 'instance method'
  end
end

Foo.bar
=> "class method"
Foo.baz
=> undefined method 'baz' for Foo:Class

Foo.new.baz
=> "instance method"
Foo.new.bar
=> undefined method 'bar' for #<Foo:0x1e820>
```

Exceptions

```
# Randon runtime error
begin
  raise "Hello I am a random runtime error"
rescue => e
  p e.message
  p e.backtrace
end

# Rescuing Exceptions Inside Methods
def some_method
  p 'Hello method'
  raise
  p 'Bye method'
rescue
  p 'Rescuing exceptions'
end
some_method

# Raising standard errors
begin
  raise ZeroDivisionError, "zero division error"
rescue ZeroDivisionError => e
  p e.message
  p e.backtrace
end

# Custom errors
class MyCrazyException < Exception
end

raise MyCrazyException, "I am a crazy new exception"
```