

# Ruby 講義

## 第4回 変数、条件分岐、繰り返し メソッド、require

五十嵐邦明

twitter : igaiga555

<http://www.facebook.com/kuniaki.igarashi>



2013.5.2 at 一橋大学  
ニフティ株式会社寄附講義  
社会科学における情報技術と  
コンテンツ作成III



# 五十嵐邦明

講師

株式会社万葉



twitter: igaiga555

<https://github.com/igaiga/>

<http://www.facebook.com/kuniaki.igarashi>



# 濱崎 健吾

Teaching Assistant  
fluxflex, inc(米国法人)



twitter: hmsk

<https://github.com/hmsk/>

<http://www.facebook.com/hamachang>

# 先週の復習

# 3つの世界

## Ruby(irb)

Ruby語が通じる世界  
1行ずつコードを実行

## Ruby(ファイル)

Ruby語が通じる世界  
ファイルにコードを書いて実行

## Shell

OS語が通じる世界  
ターミナルの中



**puts("Hello, World!")**

**オブジェクト**

**String(文字列)オブジェクト**

# puts("Hello, World!")



教科書  
p.8,9

## メソッド 引数

メソッド：手続き、命令

引数：メソッドに渡すデータ

"Hello, World!" はString(文字列)オブジェクトであり、  
この場合はメソッドへ渡しているなので引数でもあります。

また、引数の前後についているカッコ ( ) は(原則)省略可能で、  
( の代わりにスペースで書くこともできます。

つまり、以下の2文は同じ意味です。

```
puts("Hello, World!")  
puts "Hello, World!"
```

# irb (Interactive RuByの略)

irb はrubyコードを1行ずつ実行する環境です。

## 起動方法

Mac : ターミナル

Win : "Command Prompt with Ruby and Rails"

Linux(Ubuntu) : 端末 (※分からない場合は次のページ参照)

※ ↑ どれも今後はターミナルと呼びます。(ターミナルの和訳が端末)  
起動して以下のように入力してEnterを押してください。

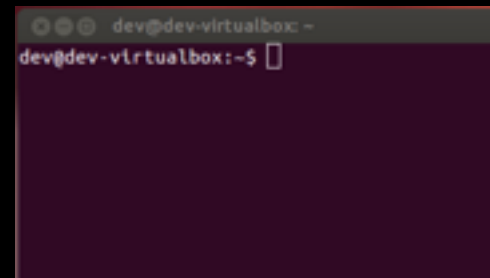
**\$ irb**

※先頭の \$ はターミナルであることを表すマークです。  
(\$と、その次のスペースは入力不要です。irbと打てばOK。)  
今後、ターミナルで打つコマンドは同じ書式  
(紫の背景色、\$ マーク)で書きます。

こんな風に表示が出ればOKです。

2.0.0p0 :001 >

※irbを終了させるには exit と打ちます。





# 四則演算

```
puts 1 + 2  
puts 2 - 3  
puts 5 * 10  
puts 100 / 4  
puts 2**32
```

**+: 足し算**

**-: 引き算**

**\*: 掛け算**

**/: 割り算**

**\*\* : 累乗**

logとかsinとかもあります。  
知りたい方はこちら。

<http://miyamae.github.io/rubydoc-ja/2.0.0/#!/class/-math.html>

**shellはOSと話をするための世界です。**

**(OS=PCと考えてもらっても大体あっています)**

**良く使うコマンドを簡単に説明します。**

**ls : ファイル一覧を見る**

**pwd : 今いるフォルダ名を表示**

**cd : フォルダ移動**

**mkdir : フォルダ作成**

# Rubyコードをファイルに記述して実行

## 1. エディタを起動します。

インストールしたエディタを起動します。

お勧めエディタ

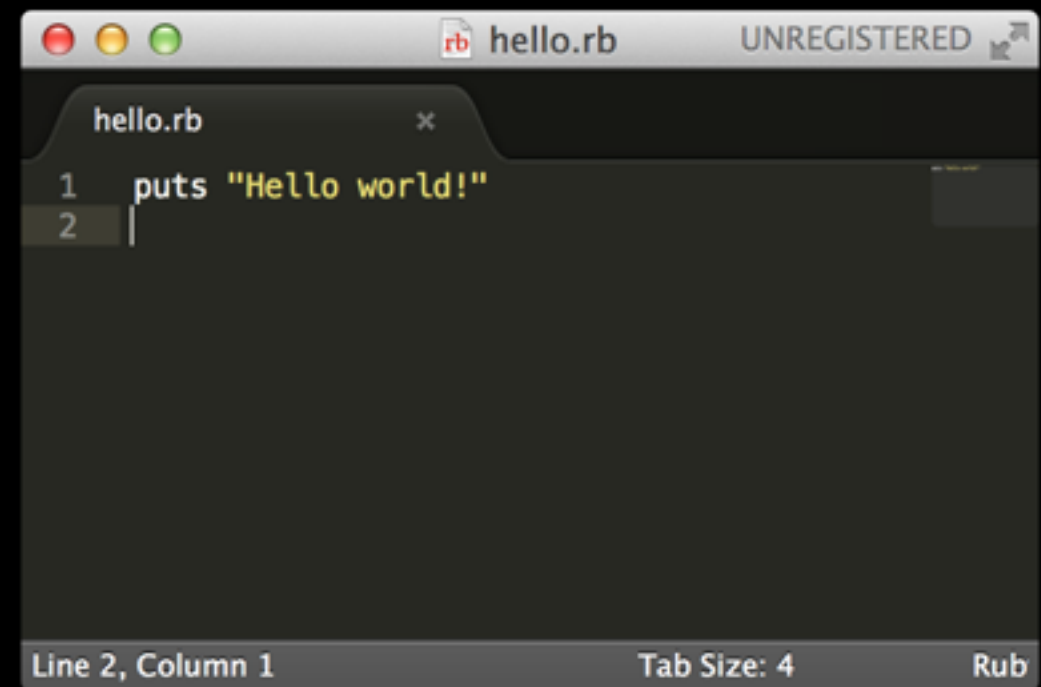
Win, Mac : SublimeText2

Linux(Ubuntu) : gedit

(Ubuntu は次ページに解説あります。)

## 2. プログラムを入力します。

```
puts("Hello world!")
```



## 3. hello.rbという名前で作成します。

手順0. で作ったフォルダの下に保存してください。

※名前は .rb を付ければ、他の名前でもOKです。



# Rubyコードをファイルに記述して実行

4. ターミナルを起動します。

5. `cd` コマンドで`hello.rb` を保存したフォルダへ移動します。

```
$ cd src
```

※`pwd`コマンドを使うと現在のフォルダを確認できます。

```
$ pwd
```

6. `hello.rb` を実行します。

```
$ ruby hello.rb
```

 ※`ruby [ファイル名]` で実行

`Hello world!` と表示されれば成功です。

成功例      `$ ruby hello.rb` ←enter  
                 `Hello world!`

※以下のエラーが出たときは `ls` コマンドでファイル有無を確認します。  
`ruby: No such file or directory -- XXX.rb (LoadError)`

# エラーメッセージは お得な情報を教えてくれる

Rubyが教えてくれたエラーメッセージ

```
helloerror.rb:2:in `<main>': undefined  
method `prin' for main:Object  
(NoMethodError)
```

日本語訳

helloerror.rb というファイルの 2 行目で  
prinなんてメソッドはないので  
そんなメソッドないよエラー が起きたよ

**ここから今週の内容**



# やること

変数

条件分岐

繰り返し

メソッド

require

# 変数



教科書  
p.21

## オブジェクトへのラベル・荷札

```
name = "igarashi"
```

name という名前の変数に  
"igarashi" オブジェクトを代入しています。

変数は代入されたオブジェクトを書いたときと  
同じように振る舞います。

以下の2つは同じ結果になります。

```
puts name
```

=> "igarashi"

```
puts "igarashi"
```

=> "igarashi"

では、この場合は  
どうでしょうか？



# サンプルコード

```
a = "abc"
```

```
b = a
```

```
a.upcase!
```

```
puts a
```

```
puts b
```

※**upcase!** は  
**String**オブジェクトを  
大文字にするメソッド

**a** は **"ABC"** になりますが、  
**b** は どうなるでしょう？

```
a = "abc"
```

```
b = a
```

```
a.upcase!
```

```
puts a  
puts b
```

aはオブジェクト  
"abc"を示  
す変数

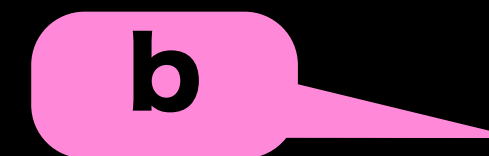
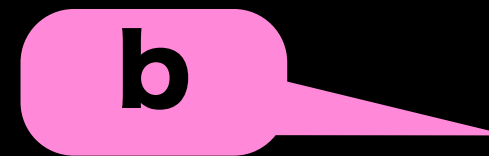
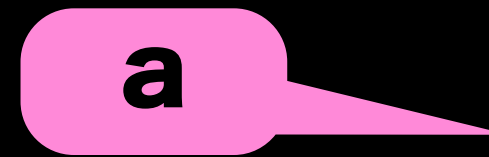
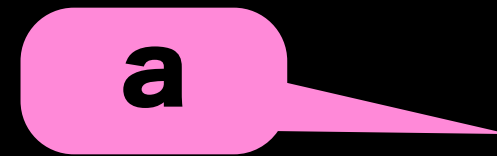
bもaと同じ  
"abc"を示す  
変数

aの指すオブ  
ジェクトを大  
文字にする

a => "ABC"  
b => "ABC"

変数

オブジェクト



# さっきと似てるけどちょっと違うコード

```
a = "abc"
```

```
b = "abc"
```

```
a.upcase!
```

```
puts a
```

```
puts b
```

新しいコード

さっきは→  
こうでした

```
a = "abc"
```

```
b = a
```

```
a.upcase!
```

```
puts a
```

```
puts b
```

さっきのコード



```
a = "abc"
```

```
b = "abc"
```

```
a.upcase!
```

```
puts a
```

```
puts b
```

aはオブジェクト  
"abc"を示  
す変数

bは別のオブジェ  
クト"abc"を指  
す

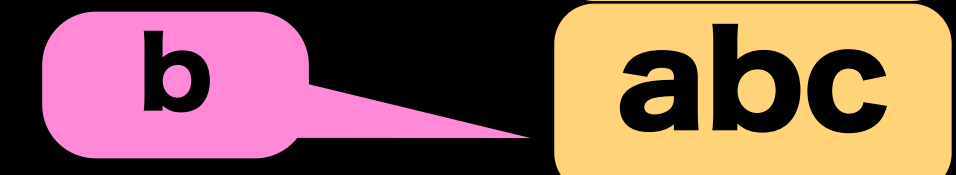
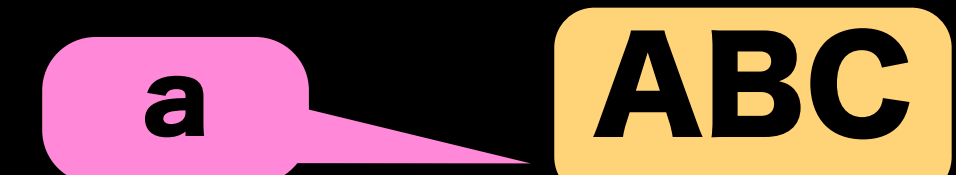
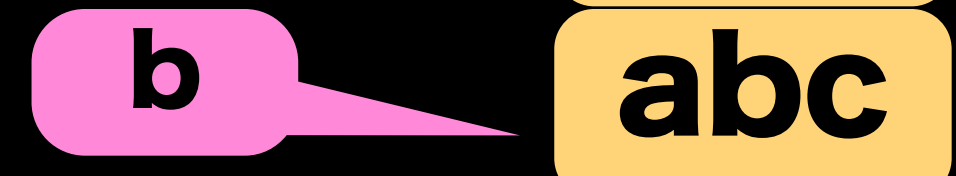
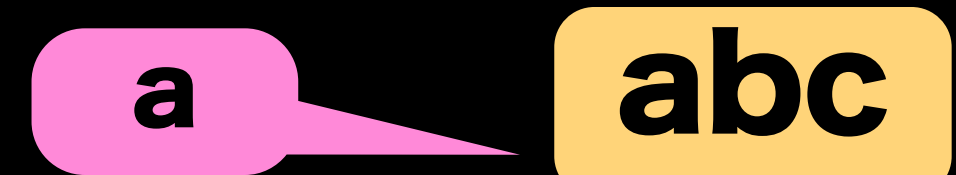
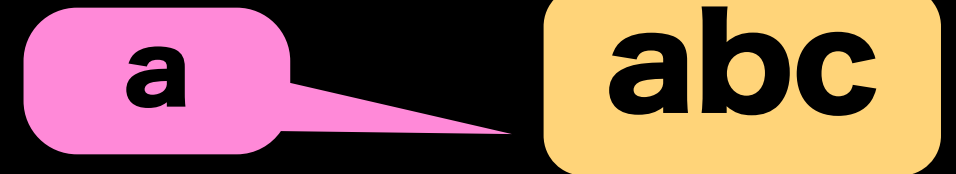
aの指すオブジ  
ェクトを大文  
字にする

a => "ABC"

b => "abc"

変数

オブジェクト



# 名付け重要

変数名は分かりやすい名前にしよう

良い例

```
width = 20  
height = 3  
area =  
    width * height
```

悪い例

```
a1 = 20  
a2 = 3  
a3 =  
    a1 * a3
```

# コメント文



教科書  
p.23

コードの中にある実行されない文  
コードの説明を書いたりします。

コメント文書式1：# 以降はコメント文

```
# name = "igarashi"
```

```
width = 2 # 文の途中からでもOK
```

↑の場合、以下の色塗りの部分がコメントになります。

```
# name = "igarashi"
```

```
width = 2 # 文の途中からでもOK
```

# コメント文



教科書  
p.23

コメント文書式2:

**=begin から =end まではコメント文**

```
=begin
```

```
  2012/04/20 作成
```

```
  2013/04/22 改訂更新
```

```
=end
```

```
puts "Hello"
```

コメント文

←この文はコードとして解釈、実行される

# 条件判断：if 文



教科書  
p.25-27

**if 条件**

**条件が成立したときに実行したい処理**

**end**

※教科書には **then** が書いてありますが、省略可能です。  
普通は省略します。私は書いたことないです。

**条件には値が true(真) または false (偽)  
となる式を書くことが一般的**



# 条件判断 == 演算子

```
x = 3 - 2
```

```
if x == 1
```

```
    puts "x is 1"
```

```
end
```

x が 1 と同じか判断し  
x が 1 の時に  
puts が実行されます。

**==** は左辺(x)と右辺(1)が同じかどうか調べて、  
同じならば true、異なれば false になります。


**=**が2個です。**=** が1つだと代入になってしまうので注意。

ちなみに、異なるかを判断する **!=** もあります。

ほかにも **>** , **>=** , **<** , **<=** など使えます。

# インデント (字下げ)

```
if x == 1  
    puts "x is 1"  
end
```



例えばif文中など、こういう風に先頭にスペースを入れて書くことをインデントするといいます。

プログラムの実行には不要なものですが、

**絶対**に入れてください！

無いと人が読めないの・・・

ちなみにスペースの個数には流派がありますが、2個が主流のようです。

# 条件判断 if - else - end

**if 条件**

**条件が成立した時に実行したい処理**

**else**

**条件が不成立の時に実行したい処理**

**end**

**条件が不成立の時に実行したい処理を書くこともできます。**

# if 文は後ろにも書ける

条件成立時に実行したい処理 if 条件

```
if x == 1  
  puts "x is 1"  
end
```

左の文は以下のように1行で書くこともできます。


```
puts "x is 1" if x == 1
```

## 1行で書ける条件

- ・ 実行したい処理が1行だけのとき
- ・ else節を書かないとき

# if文の演習

1. コードを実行した際に true が表示されるように右の空欄(黒部分)を埋めてコードを完成させて実行してください。

```
x = 2  
if   
  puts 'true'  
end
```

2. x が "GW" という文字列のときは Yeah!、それ以外の場合は sigh... と表示させるコードを書いて実行させてください。

# if文の演習

## 解答

**解答は一例です。**

**他にもいろいろな書き方ができるので、**

**「完全一致しないと正解ではない」とは思わずに。**



# if文の演習 解答

1. `x = 2`

```
if x == 2  
  puts 'true'  
end
```

他にもいろいろ書けます

```
x = 2  
if x > 1  
  puts 'true'  
end
```

2. `x` が "GW" という文字列のときは Yeah!、それ以外の場合は sigh... と表示させるコードを書いて実行させてください。

```
x = "GW"  
if x == "GW"  
  puts "Yeah!"  
else  
  puts "sigh..."  
end
```

繰り返し

# 繰り返し : while文



教科書  
p.27

```
while 繰り返し続ける条件  
  繰り返したい処理  
end
```

1から10までの数を  
順番に表示するコード

```
i = 1  
while i <= 10  
  puts i  
  i = i + 1  
end
```

# 繰り返し : times文



教科書  
p.28

```
繰り返す回数.times do  
  繰り返したい処理  
end
```

※doを  
忘れずに!

Ruby と5回表示する

```
5.times do  
  puts "Ruby"  
end
```

メソッド

# メソッド

**puts や print はメソッドだと前に説明しました。  
メソッドは「コード群を機能単位で集めたもの」  
と言えます。**

**puts や print は「表示する」ための機能を集めて提供しているメソッドです。これらはRubyがあらかじめ用意しているメソッドの中の1つです。**

**一方で、メソッドは自分で作ることもできます。  
これを「メソッドを定義する」と言います。  
次のページで定義方法を見ていきましょう。**



# メソッドの定義、呼び出し



教科書  
p.28

メソッド定義には  
**def** を使います

```
def メソッド名  
  メソッドで実行したい処理  
end
```

定義

```
def hello  
  puts "Hello"  
end
```

呼び出し

```
hello()
```

**hello()**を呼ぶと、事前に定義していた**hello**メソッドが呼ばれ実行されます。(ここでは **puts "Hello"**)

**hello()**の**()**は省略可能です。(曖昧にならない限り)

# 引数付きのメソッド

引数の仕組みを使ってメソッドにデータを渡すことができます。

```
def メソッド名(引数)  
  メソッドで実行したい処理  
end
```

定義

```
def hello(word)  
  puts word  
end
```

呼び出し

```
hello("Hi")
```

どのように動作するか次のページで説明します

# 引数付きのメソッド

定義時にメソッド名に続いて(変数名)を書くと、呼び出し時に渡された引数をその変数に代入された状態でメソッドを実行できます。

定義

```
def hello(word)
  puts word
end
```

呼び出し

```
hello("Hi")
```

hello("Hi")と呼び出すと、hello メソッドが呼ばれます。

その際に `word = "Hi"` の代入が行われます。

その後メソッドの中身が実行されます。

`puts word` が実行され、`word` には `"Hi"` が代入されているので、動作としては `"Hi"` が表示されます。

# require



教科書  
p.30-31

メソッド定義などを別のファイルに書き、読み込むことができます。

hello.rb

```
def hello  
  puts "Hello"  
end
```

use\_hello.rb

```
require "./hello"  
hello()
```

↑教科書は require "hello"になっていますが、Ruby1.9.2以降だとエラーになるので "./hello" としてください。 ./ は(shellで)今いるフォルダの意味です。

実行

上記2つのファイルを同じフォルダに置いて以下のコマンド

```
$ ruby use_hello.rb
```

# メソッドの演習

(演習1) Helloを画面に表示する

hello メソッドを定義して

呼び出すコードを書いて実行してください。

hello.rb

```
def hello  
  puts "Hello"  
end  
hello()
```

←最初なので答えを書いておきます。実行してみてください。

```
$ ruby hello.rb
```

# メソッドの演習

(演習2) 以下のような結果になるように  
以下のコードdisplay.rb の※部分に  
display メソッドの定義を書いてください

display.rb

※ここにメソッド定義を書く

```
display("Yes")
```

```
display("Ja")
```

```
$ ruby display.rb
```

ヒント：

引数付きメソッド  
定義を使います

←実行したときに

Yes

Ja

←実行結果はこの2行が表示されてほしい

# メソッドの演習

**(演習3) 演習2で書いたメソッド定義を別のファイルに移動して、require で読み出し、同じ結果になるようにしてください。**



# メソッドの演習

## 解答

# メソッドの演習 解答

(演習1) Helloを画面に表示するhello メソッドを定義して呼び出すコードを書いて実行してください。→演習のページに書いてある通り

(演習2) 以下のような結果になるように以下のコードdisplay.rb の※部分にdisplay メソッドの定義を書いてください

**display.rb**

```
def display(text)
  puts text
end

display("Yes")
display("Ja")
```

```
$ ruby display.rb
```

# メソッドの演習 解答

(演習3) 演習2で書いたメソッド定義を別のファイルに移動して、`require` で読み出し、同じ結果になるようにしてください。

**display.rb**

```
def display(text)
  puts text
end
```

**use\_display.rb**

```
require "./display"
display("Yes")
display("Ja")
```

```
$ ruby use_display.rb
```

今日の

まとめ

# 変数



教科書  
p.21

## オブジェクトへのラベル・荷札

```
name = "igarashi"
```

name という名前の変数に  
"igarashi" オブジェクトを代入しています。

変数は代入されたオブジェクトを書いたときと  
同じように振る舞います。

以下の2つは同じ結果になります。

```
puts name
```

=> "igarashi"

```
puts "igarashi"
```

=> "igarashi"

# 名付け重要

変数名は分かりやすい名前にしよう

良い例

```
width = 20  
height = 3  
area =  
    width * height
```

悪い例

```
a1 = 20  
a2 = 3  
a3 =  
    a1 * a3
```

# コメント文



教科書  
p.23

コードの中にある実行されない文  
コードの説明を書いたりします。

コメント文書式1：# 以降はコメント文

```
# name = "igarashi"
```

```
width = 2 # 文の途中からでもOK
```

↑の場合、以下の色塗りの部分がコメントになります。

```
# name = "igarashi"
```

```
width = 2 # 文の途中からでもOK
```



# 条件判断：if 文



教科書  
p.25-27

**if 条件**

**条件が成立したときに実行したい処理**

**end**

※教科書には **then** が書いてありますが、省略可能です。  
普通は省略します。私は書いたことないです。

**条件には値が true(真) または false (偽)  
となる式を書くことが一般的**

# 条件判断 == 演算子

```
x = 3 - 2
```

```
if x == 1
```

```
    puts "x is 1"
```

```
end
```

x が 1 と同じか判断し  
x が 1 の時に  
puts が実行されます。

**== は左辺(x)と右辺(1)が同じかどうか調べて、  
同じならば true、異なれば false になります。**


**=が2個です。= が1つだと代入になってしまうので注意。**

**ちなみに、異なるかを判断する != もあります。**

**ほかにも > , >= , < , <= など使えます。**

# インデント (字下げ)

```
if x == 1  
    puts "x is 1"  
end
```



例えばif文中など、こういう風に先頭にスペースを入れて書くことをインデントするといいます。

プログラムの実行には不要なものですが、

**絶対**に入れてください！

無いと人が読めないの・・・

ちなみにスペースの個数には流派がありますが、2個が主流のようです。

# 条件判断 if - else - end

**if 条件**

**条件が成立した時に実行したい処理**

**else**

**条件が不成立の時に実行したい処理**

**end**

**条件が不成立の時に実行したい処理を書くこともできます。**

# 繰り返し : while文



教科書  
p.27

```
while 繰り返し続ける条件  
  繰り返したい処理  
end
```

1から10までの数を  
順番に表示するコード

```
i = 1  
while i <= 10  
  puts i  
  i = i + 1  
end
```

# 繰り返し : times文



教科書  
p.28

```
繰り返す回数.times do  
  繰り返したい処理  
end
```

※doを  
忘れずに!

Ruby と5回表示する

```
5.times do  
  puts "Ruby"  
end
```

# メソッドの定義、呼び出し



教科書  
p.28

メソッド定義には  
**def** を使います

```
def メソッド名  
  メソッドで実行したい処理  
end
```

**定義**

```
def hello  
  puts "Hello"  
end
```

**呼び出し**

```
hello()
```

**hello()**を呼ぶと、事前に定義していたhelloメソッドが呼ばれ実行されます。(ここでは puts "Hello")

**hello()**の()**は省略可能です。(曖昧にならない限り)**

# 引数付きのメソッド

引数の仕組みを使ってメソッドにデータを渡すことができます。

```
def メソッド名(引数)  
  メソッドで実行したい処理  
end
```

定義

```
def hello(word)  
  puts word  
end
```

呼び出し

```
hello("Hi")
```

どのように動作するか次のページで説明します



# require



教科書  
p.30-31

メソッド定義などを別のファイルに書き、読み込むことができます。

hello.rb

```
def hello  
  puts "Hello"  
end
```

use\_hello.rb

```
require "./hello"  
hello()
```

↑教科書は require "hello"になっていますが、Ruby1.9.2以降だとエラーになるので "./hello" としてください。 ./ は(shellで)今いるフォルダの意味です。

実行

上記2つのファイルを同じフォルダに置いて以下のコマンド

```
$ ruby use_hello.rb
```



# 参考資料

# 講義資料置き場

過去の資料がDLできます。

<https://github.com/igaiga/hitotsubashi-ruby-2013>

# 雑談・質問用facebookグループ

<https://www.facebook.com/groups/hitotsubashi.rb>

- 加入/非加入は自由です
- 加入/非加入は成績に関係しません
- 参加者一覧は公開されます
- 参加者はスタッフ(講師・TA)と昨年、今年を受講者です
- 書き込みは参加者のみ見えます
- 希望者はアクセスして参加申請してください
- 雑談、質問、議論など何でも気にせずどうぞ～
- 質問に答えられる人は答えてあげてください
- 講師陣もお答えします
- 入ったら軽く自己紹介おねがいします