

Rubyレクチャー

資料場所

- <https://github.com/rubylecture/Rubylecture>

Description

Short description of this repository

Website

Website for this repository (optional)

[Save](#) or [Cancel](#)

2 commits

1 branch

0 releases

1 contributor



Branch: master

Rubylecture / +



snake090 rubylecture

Latest commit 48d46fc 2 hours ago



README.md

Initial commit

2 hours ago



Rubyレクチャー.pdf

rubylecture

2 hours ago



Rubyレクチャー.pptx

rubylecture

2 hours ago



accessor.rb

rubylecture

2 hours ago



car.rb

rubylecture

2 hours ago



classmember.rb

rubylecture

2 hours ago



module.rb

rubylecture

2 hours ago



README.md

Rubylecture

<> Code

Issues

0

Pull requests

0

Wiki

Pulse

Graphs

Settings

HTTPS clone URL

<https://github.com>



You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).

Clone in Desktop

Download ZIP

環境設定

- <http://rubyinstaller.org/downloads/> にアクセス










Downloads



RubyInstallers

Archives»

Not sure what version to download? Please read the right column for recommendations.

-  [Ruby 2.2.3](#)
-  [Ruby 2.2.3 \(x64\)](#)
-  [Ruby 2.1.7](#)
-  [Ruby 2.1.7 \(x64\)](#)
-  [Ruby 2.0.0-p647](#)
-  [Ruby 2.0.0-p647 \(x64\)](#)
-  [Ruby 1.9.3-p551](#)

Other Useful Downloads

7-ZIP ARCHIVES

-  [7-Zip 9.20](#)

WHICH VERSION TO DOWNLOAD?

If you don't know what version to install and you're getting started with Ruby, we recommend you use Ruby **2.1.X** installers. These are the most stable language and a extensive list of packages (gems) that are compatible and updated.

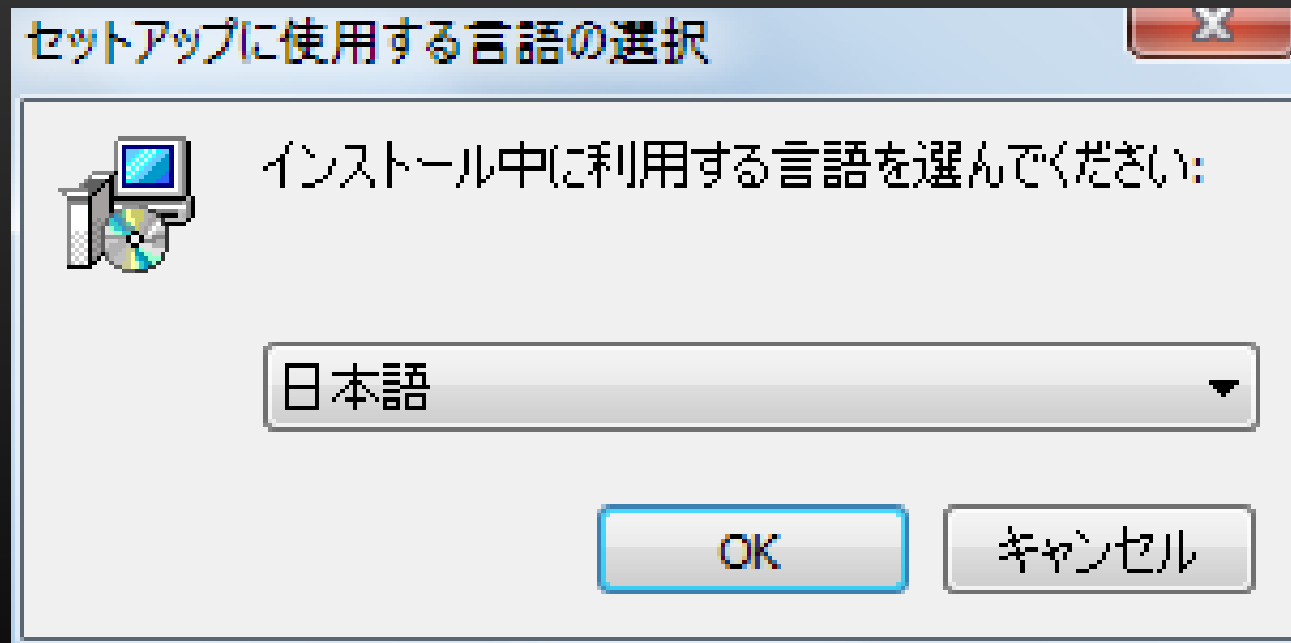
However, not all packages (gems) are maintained. Some older packages may not be compatible with newer versions of Ruby and RubyGems.

The 64-bit versions of Ruby are relatively new on the Windows platform. If you use not all the packages have been updated to be compatible with this version you will require some knowledge about compiling and dependency issues, which might be too complicated if you just want to play with the language.

Users of CPUs older than Intel's Nocona (90nm Pentium 4) who use Ruby 2.0.0 will need to build their own using a different compiler, following these [instructions](#).

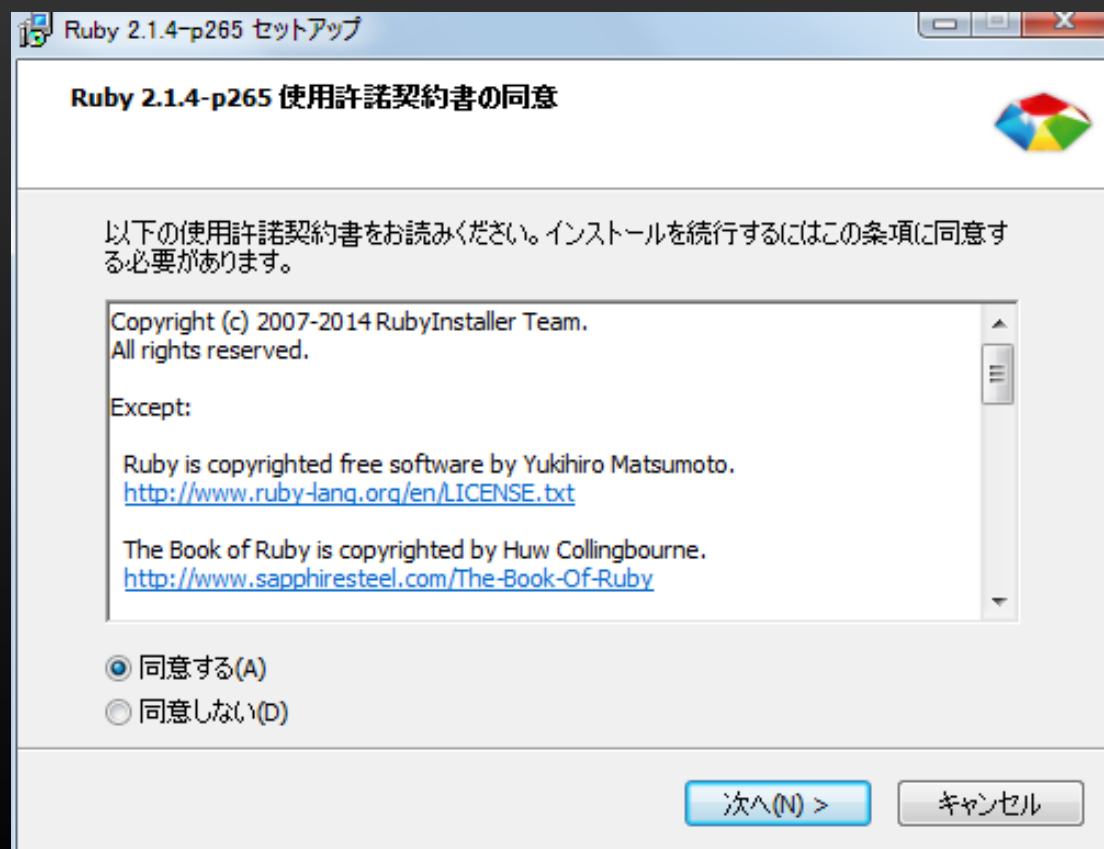
環境設定

- OKクリック



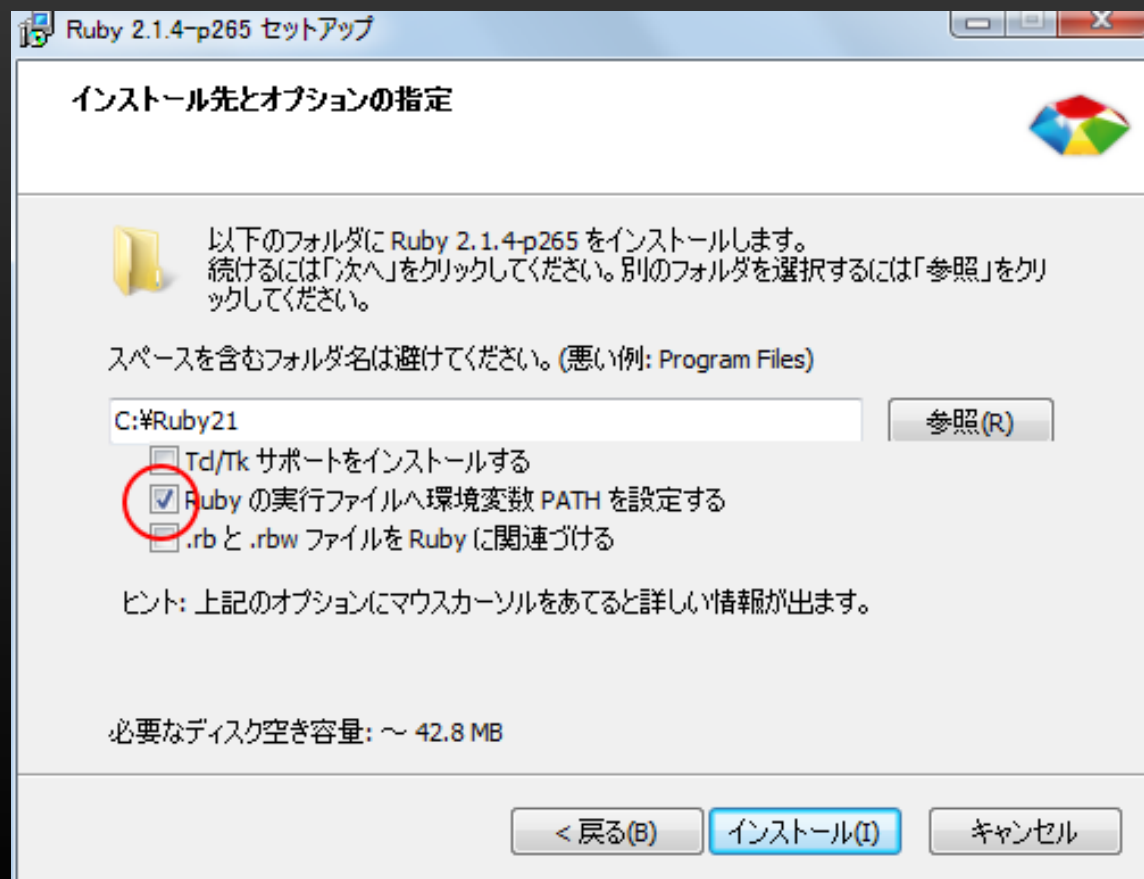
環境設定

- 同意する 次へ



環境設定

- 保存先を選択して、Rubyの実行ファイルへ環境変数PATHを設定する



環境設定

- 次の画面が表示されればインストール完了



Ruby特徴

- インタプリタ言語
 - ソースコードはコンパイルせずに実行することができる
- オブジェクト指向
- 強い動的型付け言語
 - Ruby の変数にはどのような型の値でも代入することができる

変数宣言

```
# -*- coding: shift_jis -*-
```

```
#変数宣言
```

```
x=5.0
```

```
y=9.0
```

```
mean=(x+y)/2
```

```
puts("#{x}と#{y}の平均は#{mean}である.")
```

変数宣言

- `# -*- coding: shift_jis -*-` 文字コード指定
- Rubyは変数宣言(int,float等)を宣言せずに使うことができる
- `puts` で出力することができる
- 変数は`#{変数}`のようにして出力する

入出力

```
# -*- coding: shift_jis -*-
```

```
print '整数入力'
```

```
x=gets.chomp.to_f#get:キー入力 chomp:改行コード取り除く to_f:float型
```

```
print "#{x}¥n"
```

```
print'文字列入力'
```

```
mozi=gets.chomp
```

```
print(mozi,"¥n")
```

入出力

- print puts 違い print改行なし puts改行あり

- シングルクォート ダブルクォート違い

シングルクォート 式展開されない

ダブルクォート 式展開される

例 `print "#{x}¥n"` `print '#{x}¥n'`

出力 10 `#{x}`

- gets キーボードから文字入力を受け,string型で取得
- chomp 改行コードを取り除く
- to_f float型に変換 to_i:int型

Times文

```
# -*- coding: shift_jis -*-
```

```
10.times do |i|  
  puts "abc"  
  puts "#{i}回目の繰り返しです.¥n"  
end
```

Times文

- 使い方

ループ回数.times do i	ループ回数.times { i
処理内容	処理内容
end	}

i=0から始まりiが一つずつ増える

ループ文には他にもwhile,for,each等ある

配列

```
# -*- coding: shift_jis -*-  
  
names=["ando","ito","ueno"]  
names.size.times{|i|  
    puts names[i]  
}
```

配列

- 宣言方法: 配列名=["要素","要素"]
要素ない場合: 配列名=Array.new(確保する配列数)
- 配列名.size 確保した配列数を返す

問題

- 1～20までの数字を配列に格納して、1～20の合計と平均をもとめる

メソッド

```
# -*- coding: shift_jis -*-
```

```
def keisan(*number)
  sum=0.0
  number.size.times{ |i|
    sum=sum+number[i]
  }
  puts sum
  return sum/number.size
end
```

```
number=(1..20).to_a
puts keisan(*number)
```

メソッド

- 使い方

```
def メソッド名(引数) *引数は先頭大文字 ×  
  処理内容  
end
```

宣言せずに好きな型を帰すことができる

*変数名 : 可変長引数 好きなサイズの配列を渡すことができる

注意: 一番最後の引数につける

クラス

```
class Car
  def initialize(carname) #コンストラクタ
    @name = carname #@ インスタンス変数
  end

  def dispName #メソッド
    print(@name)
  end
end

car = Car.new("crown") #インスタンス化
car.dispName
```

クラス

- 使い方

class クラス名 *クラス名は最初大文字

処理内容

end

- コンストラクタ インスタンス化した時に自動で呼ばれるメソッド

def initialize(引数)

処理名

end

クラス

- インスタンス変数
- 宣言方法 :@変数名
- クラス内で全メソッドで共通して使用することが出来る
- クラス内のどこかのメソッドで宣言すれば使うことができる
- クラス外からのアクセスはできない

アクセサ

```
# -*- coding: shift_jis -*-
```

```
class Car
```

```
  attr_accessor:name #変数の値の読み取り・書き換えを行う
```

```
  attr_reader:price #変数の値の読み取りだけを行う
```

```
  def initialize(carname,price)
```

```
    @name = carname
```

```
    @price = price
```

```
  end
```

アクセサ続き

```
def dispName
  puts("name:#{@name},price:#{@price}")
end
end
car = Car.new("crown",100)
car.dispName
car.name="lexus"
#car.price=80 エラー
car.dispName
```

アクセサ

- インスタンス変数はクラス外からアクセスできない
- アクセサを用いることでクラス外から参照、更新することができる
- 使い方 : アクセサ:変数名
- attr_accessor 変数の値の読み取り・書き換えを行う
- attr_reader 変数の値の読み取りだけを行う
- attr_writer 変数の値の書き換えだけを行う

クラス変数 クラスメソッド

```
class Tax
  @@zeiritsu = 0.08 #クラス変数
  def self.zeiritsu= (n)
    @@zeiritsu = n
  end
  def self.priceWithTax(price)
    return (price * (1.0 + @@zeiritsu)).to_i
  end
  def self.tax(price)
    return (price * @@zeiritsu).to_i
  end
end
```

クラス変数 クラスメソッド

```
price = 12300
```

```
puts("価格:" + price.to_s)
```

```
puts("税込:" + Tax.priceWithTax(price).to_s)
```

```
puts("税額:" + Tax.tax(price).to_s)
```

```
Tax.zeiritsu = 0.1
```

```
puts("※消費税が10%にあがると.....")
```

```
puts("税込:" + Tax.priceWithTax(price).to_s)
```

```
puts("税額:" + Tax.tax(price).to_s)
```

クラス変数 クラスメソッド

- クラスメソッド: インスタンス化せずに呼ぶことができるメソッド
- 宣言方法 : `def クラス.メソッド`, `def self.メソッド`
- クラス変数: インスタンス化せずにクラス内に値を保持することができる
- 宣言方法 : `@@変数名`

モジュール

```
# -*- coding: shift_jis -*-
```

```
module SuuchiModule #モジュール  
  def minValue(x, y)  
    if x < y  
      return x  
    else  
      return y  
    end  
  end  
end
```

モジュール続き

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

```
module_function :minValue #モジュール関数ができるように設定する
module_function :maxValue
end
```


モジュール続き

```
class Test
```

```
  include SuuchiModule #モジュールをインクルードする
```

```
  def dispValue(x, y)
```

```
    min = minValue(x, y)
```

```
    print("2つの値", x, "と", y, "の中で小さい値は", min, "です¥n")
```

```
  end
```

```
end
```

```
include SuuchiModule # インクルードすることでモジュール名を省略して関数を使える
```

```
print(minValue(10, 8), "¥n")
```

```
print(maxValue(10, 8), "¥n")
```

```
test = Test.new
```

```
test.dispValue(10, 8)
```

モジュール

- オブジェクトを生成することができない
- 使用方法:「モジュール名.メソッド名」の形式で関数のように実行する
他のクラスの中にインクルードして利用する
- クラスの共通となる機能をモジュールとして定義し、各クラスにインクルードして利用することでコードの再利用性を高めることができる

モジュール(関数)

- 使い方

Module モジュール名 *モジュール名先頭文字大文字
メソッド

module_function:関数名
end

module_function:モジュール内の関数を使えるようにする

include モジュール名: モジュール名なしで宣言できる

includeなし:モジュール名.関数名

Includeあり:関数名

モジュール(クラス)

- 使い方

```
Module モジュール名 *モジュール名先頭文字大文字  
メソッド  
end
```

```
include モジュール名 :クラス内で宣言することで全てのモジュール内  
のメソッドがクラスのメソッド内で使用できる
```

- 以下のCarクラスとモジュールを設計せよ
 - クラス変数: ElementCount 配列の要素数をカウントする
 - インスタンス変数: CarName [5], Price [5]
 - メソッド : AddCar 車名と値段を引数にして、配列の末尾に追加 既に5つ登録されていた場合先頭要素を消去し、ArrayPackを実行し、配列の末尾に要素を追加
 - :DeleteCar 番号の配列の要素を消去し、ArrayPackを実行
 - :ShowCar 0～4の番号を付けて、全てのCarNameとPriceを表示
 - モジュール: ArrayPack :要素を前に詰めた配列を返す
- ループを回しキーボードから1～4の数字を受け以下のメソッドを実行
- 1:車名と値段を入力してAddcar実行 2:番号を入力してDeleteCar 実行
3:ShowCar実行 4:ループ終了