

Rubyレクチャー

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

-  [Ruby 2.2.3](#)

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.

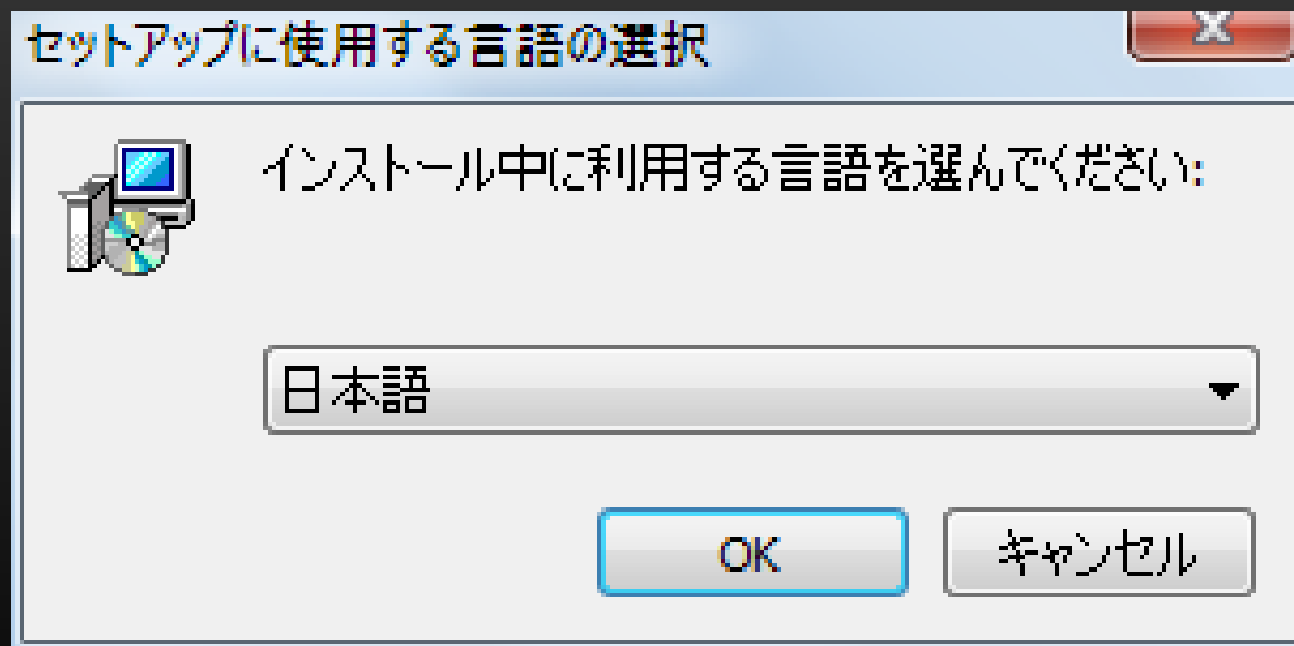
However, not all packages (gems) are maintained. Some older packages may not be compatible with newer versions of Ruby and RubyGems.

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](#).

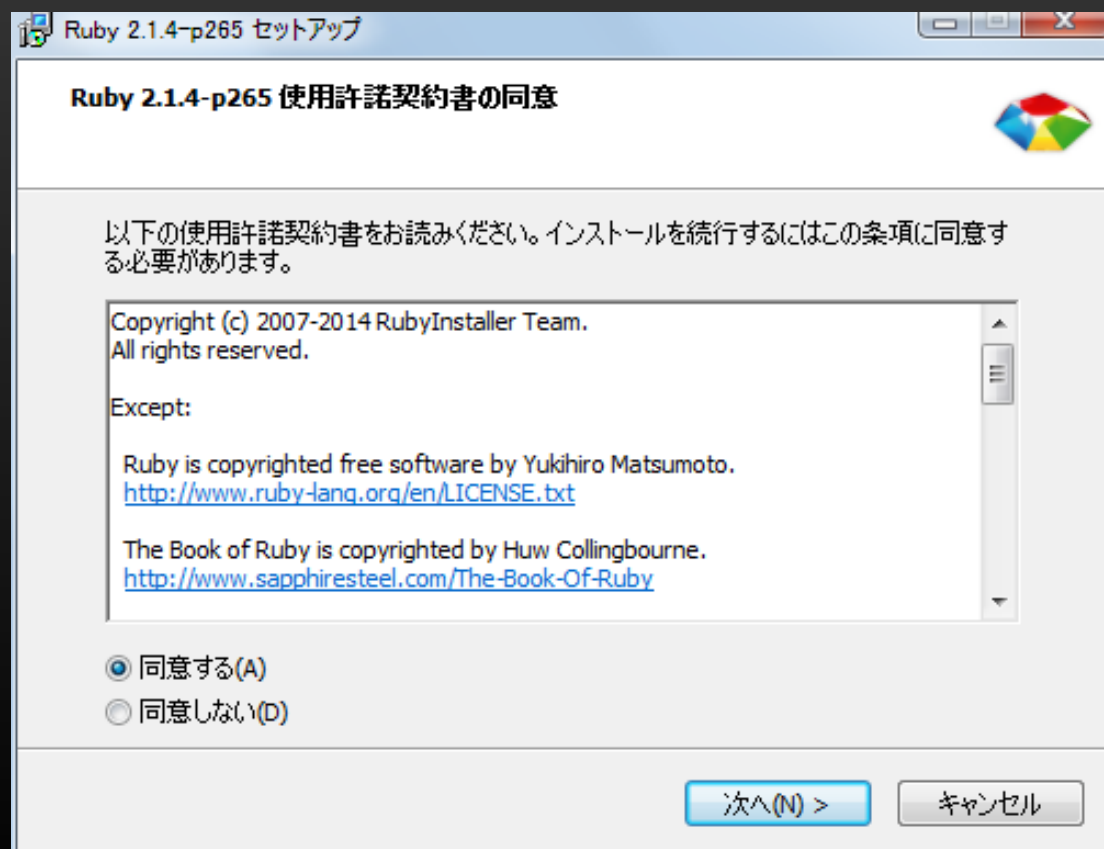
環境設定

- OKクリック



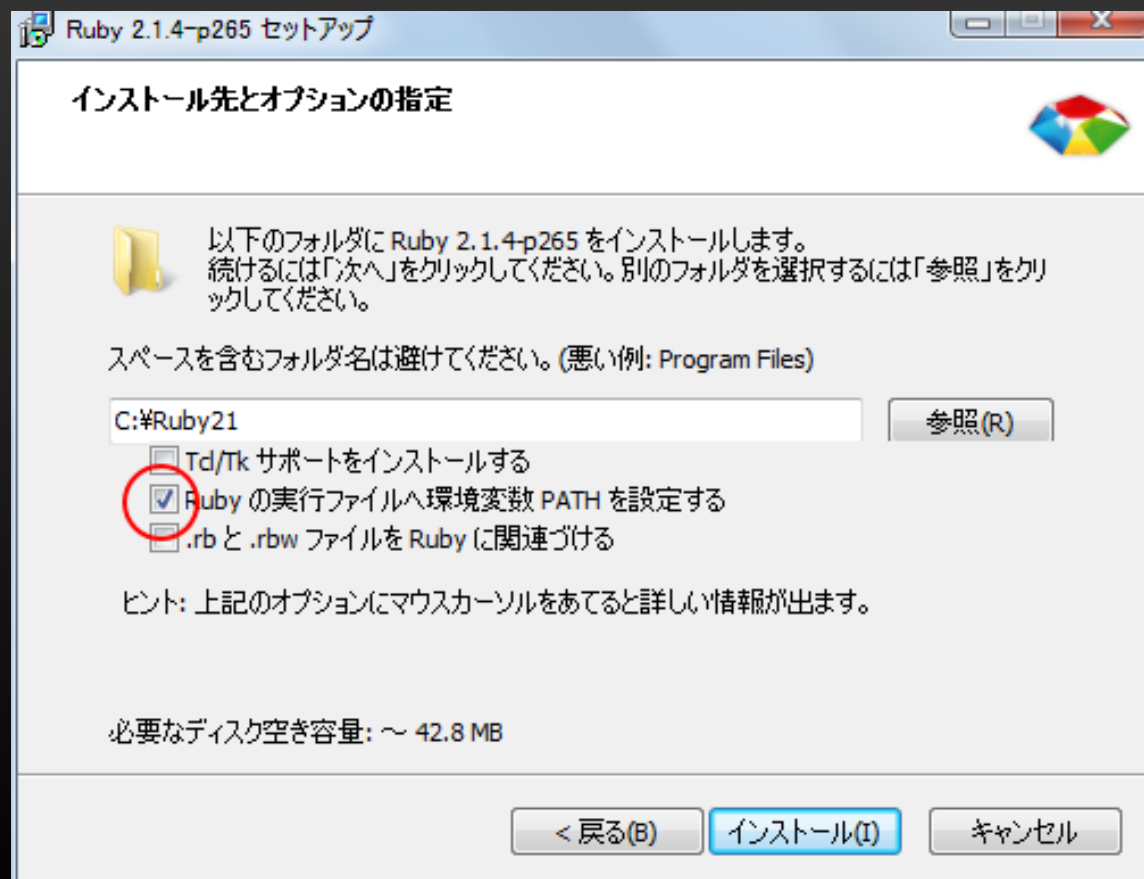
環境設定

- 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.")
```

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"
```

Times文

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

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

配列

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

```
# Ruby Test
```

```
names=["ando","ito","ueno"]
```

```
names.size.times{|i|
```

```
    puts names[i]
```

```
}
```

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

解答

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

```
number=(1..20).to_a
```

```
sum=0.0
```

```
number.size.times{ |i|
```

```
    sum=sum+number[i]
```

```
}
```

```
puts sum
```

```
puts sum/number.size
```

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
```

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
- 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

- 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

モジュール

- オブジェクトを生成することができない
- 利用方法:「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
- 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

- 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