

IT日本語（2）

Ruby

授業の目的と目標

目的

- Rubyの基本的な文法を扱えること
- 日本語で書かれた課題を読み、Rubyを用いて解決できる

目標

- 基本的な文法を学ぶ
- 実際にコードを実行させる

目次

- 1. 標準出力
- 2. コメント
- 3. 演算子
- 4. 変数
- 5. 代入演算子
- 6. 文字列
- 7. シンボル
- 8. 配列
- 9. イテレーション
- 10. ハッシュ
- 11. 条件分岐(if/else/elsif/unless)
- 12. 条件分岐(When/Case)
- 13. 比較演算子
- 14. ループ(for)
- 15. ループ(while)
- 16. メソッド
- 17. ブロック
- 18. Yield (イールド)
- 19. クラスとインスタンス
- 20. Mix-in

コードの実行環境

コードの実行

- ・ https://www.tutorialspoint.com/execute_ruby_online.php

The screenshot shows the CodingGround online Ruby executor interface. The top bar includes the logo, project dropdown, file menu, edit menu, view menu, shutdown button, and help button. The main area has tabs for 'Execute' and 'Share Code', with 'main.rb x' selected. The code editor contains the following Ruby script:

```
1 # Hello World Program in Ruby
2 puts "Hello World!";
3
```

To the right is a sidebar titled 'Tutorials' with links for Ruby, Ruby on Rails, Ruby on Rails 2.1, and Python. At the bottom, there's a terminal window labeled 'Default Term' with the prompt 'sh-4.3\$'.

文法の学習

標準出力

```
print("hello")
p("world")
puts("hello world")
p "hello world"
```

コメント(comment)

```
#これは単行のコメントです。
```

```
=begin
```

```
　これは複数行のコメントです。
```

```
=end
```

えんざんし 演算子(operator)

```
p 1000 + 12
p 20 - 43
p 212 * 5
p 30 / 5
p 72 % 11
p 3 ** 16
```

へんすう 変数(variables)

```
# 英字が "_"(アンダーバー) から始まる
# 英数字からなる文字です。
# 予約後は使えません。

name = "Tran Nam Phuong"
_age = 24
is_student = true
nil = 0

# 定数(変更しないことが期待される変数)
# すべて大文字かアンダーバーで構成する
DEFAULT_VALUE = 123489
DEFAULT_VALUE = 100
```

だいにゅうえんざんし 代入演算子

```
str = "フランジア"  
  
# 多重代入  
  
age, name = 31, "ishida"  
age, name = [31, "ishida"]  
  
# *をつけると複数の値を配列に代入  
  
x,y,z = 1, *[2,3]  
x,y,*z = 1, *[2,3,4]  
  
# 自己代入  
  
my_point = 5; my_point += 10  
fee = nil; fee ||= 10
```

もじれつ 文字列(text)

```
str = "私は日本人です"  
p str + "私は1993年生まれです"  
  
age = 24  
p "私は#{age}歳です"  
  
p "bow!" * 5  
p str[2..4]          #範囲に注意  
  
p "私は大学" + 4 + "年生です"  
p "私は大学%d年生です" % 4  
  
# ヒアドキュメント  
line = <<EOL  
私はベトナム人です  
私は1993年生まれです  
EOL
```

シンボル(symbol)

```
# 下記例は同じ意味  
lang1 = "japanese"  
if lang1 == "japanese"  
  ...  
end  
  
lang2 = :vietnamese  
if lang2 == :vietnamese  
  ...  
end  
  
# 違いは以下  
puts "japanse".object_id  
puts "japanse".object_id  
puts :vietnamese.object_id  
puts :vietnamese.object_id
```

はいれつ 配列(array)

```
arr1 = [0, 2, 4]
p arr1[1]
p arr1.length()
p arr1.length

# 初期化方法
p Array.new(3)
p Array.new(3, 5)

arr2 = Array.new(3) do |i|
  i * 2
end
p arr2
```

配列とテキスト

```
# テキストから配列をつくる  
p "Ruby Java Python".split  
p "100, 212, 345".split(",")
```

```
# 配列からテキストを作る  
p [123,45,67,8,90].join  
p ["I","am","a","Vietnamese"].join(" ")
```

イテレーション(iteration)

```
# 配列の要素を一つづつ処理する

arr = [1,2,3,4,5]
arr.each do |x|
  x *= 10
  p x
end

arr2 = arr.select do |i|
  i % 2 == 0
end
p arr2

arr3 = arr.collect do |i|
  i ** 2
end
p arr3
```

レンジとイテレーション

```
# レンジ
ran1 = 1..10
p ran1.to_a
ran2 = 1...10
p ran2.to_a

# 配列に用いる場合、Forループの様に使える
total = 0
(1..10).to_a.each do |i|
  total += i
end
p total

# 数字以外
chars1 = "a".."f"; p chars1.to_a
chars2 = "G".."L"; p chars2.to_a
```

ハッシュ(hash)

```
# 配列の要素に名前をつけたもの
fruits = { "apple" => 5, "banana" => 3}
p fruits["apple"]
fruits["orange"] = 8

wishlist = Hash.new()

# 配列同様にイテレーションできる
fruits.each do |i,j|
  p "%s: %d" % [i,j]

end

pricelist = fruits.select do |i,j|
  j > 4
end
p pricelist
```

じょうけんぶんき 条件 分岐 (if/else/elsif/unless)

```
score = 75
if score > 90 then
    p "Excellent"
elsif score > 70 then
    p "Good(" + score.to_s + ")"
else
    p "bad"
end
```

```
is_completed = false
unless is_completed
    p "It's happened error"
end
```

```
# if文の入れ替え
is_pass = true
p "It's passed"  if is_pass
```

じょうけんぶんき 条件 分岐 (when/case)

```
p "Please input your question"  
p "[1] What your name"  
p "[2] How old are you"  
p "[3] What do you like food"  
p "[0] Nothing"  
p ">>>"  
input = gets  
  
case input.to_i  
when 1  
  p "I'm Duc"  
when 2  
  p "I'm 23 years old"  
when 3  
  p "I love all of Vietnam food"  
else  
  p "I did't understand"  
end
```

gets (標準入力)
ユーザーが自由に値を入れることができる

ひかくえんざんし 比較演算子

(==/!=/=/>/</>=/<=/<=)

```
p 1 == 1
p 2 != 3
p 10 > 5
p 12 <= 100
p "A" < "a"
```

```
# UFO演算子
p 10 <=> 1
p 1 <=> 1
p 1 <=> 10
```

ループ(for)

```
# 基本的なループ
```

```
10.times do |i|
  print (i ** 2).to_s + ", "
end
```

```
# レンジを使うループ
```

```
for i in 1..5
  print i.to_s + ", "
end
```

```
for i in 1...5
  print i.to_s + ", "
end
```

ループ(while)

```
times = 0
total = 1.0
while total < 2
    total *= 1.03
    times += 1
end
p "%d回(%f)" % [times, total]
```

```
times = 0
total = 1.0
until total < 0.05
    total /= 2
    times += 1
end
p "%d回(%f)" % [times, total]
```

課題1

課題1

2から100までの素数(Prime number)を表示しなさい (例) 2,3,5,7,11,...

```
"2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,  
83, 89, 97"
```

問題1

問題1

K58 IT日本語(2) #2-2 Ruby Aクラス

指示があるまで問題を解かないでください。
制限時間は6分間です

次へ

- Google Formを使って課題を出します
メール、またはチャットワークでURLを共有します
- 指示があるまで問題を解かないでください。
- 制限時間は6分間です

メソッド

```
def func1(apple, orange)
  r = apple * 100
  r += orange * 80
  return r
end
p func1(3, 5)      # <= func1 3, 5
```

```
def func2(apple, orange, basket=1)
  r = apple * 100
  r += orange * 80
  r += basket * 300
end
p func2(2, 4)      # <= func2 2,4
```

```
def func3(name, *favorite)
  favorite
end
p func3("ishida", "Ruby", "PHP", "Javascript")
```

- メソッドの入力値(parameter)のことを引数という
- メソッドの結果のことを戻り値という

ブロック

ブロックとは処理を独立させたもの

do ~ endと{ ~ }は等価

```
(1..5).each do |counter|
  puts "iteration #{counter}"
end
```

```
(1..5).each { |counter|
  puts "iteration #{counter}"
}
```

ブロックの利用方法

```
a = [ "a", "b", "c", "d" ]  
p a.collect() { |x|  
  x + "!"  
}
```

{|x| x + “!”} は
collectメソッドの引数

```
array = [1, 2, 3, 4, 5]  
doubled = array.map() do |element|  
  element * 2  
end  
puts doubled
```

element * 2 は
mapメソッドの引数

Yield (イールド)

```
def do3times
    yield
    yield
    yield
end

# 上のメソッドは使う人によって自由に変更できる
do3times() {
    puts "I'm a Japanese"
}

do3times() {
    puts "I'm a Vietnamese"
}
```

クラスとインスタンス

```
class Human
  def initialize(name, age = 0)
    @name = name
    @age = age
  end

  def name=(name)
    @name = name
  end

  def name
    @name
  end
end

man = Human.new "ishida", 31
p man
```

- class Xxxxx ~ end の部分がクラス定義
- def ~ endの部分がインスタンスマソッド
- initializeは、コンストラクタマソッド
インスタンスを作られたときに自動的に
呼び出される
- @nameは、インスタンス変数
- インスタンスを作る場合は、Xxxxx.new(引数)

アクセッサー

(attr_accessor, attr_writer, attr_reader)

```
class Human
  attr_accessor :name, :age
  attr_reader :school_name

  def initialize(name, age = 0)
    @name = name
    @age = age
    @school_name = "HUST"
  end
end
```

```
man = Human.new "ishida", 32
man.age = 33
p man
```

```
man.school_name = "Keio University"
```

- ・自動的にインスタンス変数を作成する
- ・attr_xxxxは実際にはメソッド
- ・シンボルを引数に渡す
- ・attr_writerは書き込みだけ許可する
- ・attr_readerは読み込みだけ許可する

インスタンス変数とクラス変数

```
class Student
  private
    @@student_number = 0

  public
    def initialize (name, age=22)
      @name = name
      @age = age
      @@student_number += 1
    end

    def introduce
      p "My name is #{@name} and #{@age} years old"
      p "My student number is #{@@student_number}"
    end
  end
  Student.new("Than Ba Trong").introduce
  Student.new("Nguyen Thi Ngoc" ).introduce
```

- @@はクラス変数

- インスタンス変数の初期化はコンストラクタ内で

カプセル化

```
class Sample
public
def method1; p "this is public"; end

private
def method2; p "this is private"; end

protected
def method3; p "this is protected"; end

public
def method4; method2; end
def method5; self.method2; end
def method6; method3; end
def method7; self.method3; end
end
sample = Sample.new()
```

正しく実行できるのは？

sample.method1

sample.method2

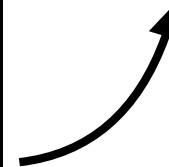
sample.method3

sample.method4

sample.method5

sample.method6

sample.method7



カプセル化

```
class Sample
public
def method1; p "this is public"; end

private
def method2; p "this is private"; end

protected
def method3; p "this is protected"; end

public
def method4; method2; end
def method5; self.method2; end
def method6; method3; end
def method7; self.method3; end
end

sample = Sample.new()
```

- public

外から自由に呼び出すことができる

- private

外から自由に呼び出すことができない

- protected

同じ名前のメソッド（例ではmethod3）を持っているオブジェクトからは自由に呼び出すことができる

継承 (Inheritance)

```
class Hedspi < Student
```

- 継承は" < "を使います

```
  def introduce_jp
```

```
    p "私の名前は#{@name}、#{@age}歳です"
```

```
    p "私の学籍番号は#{@@student_number}です"
```

```
  end
```

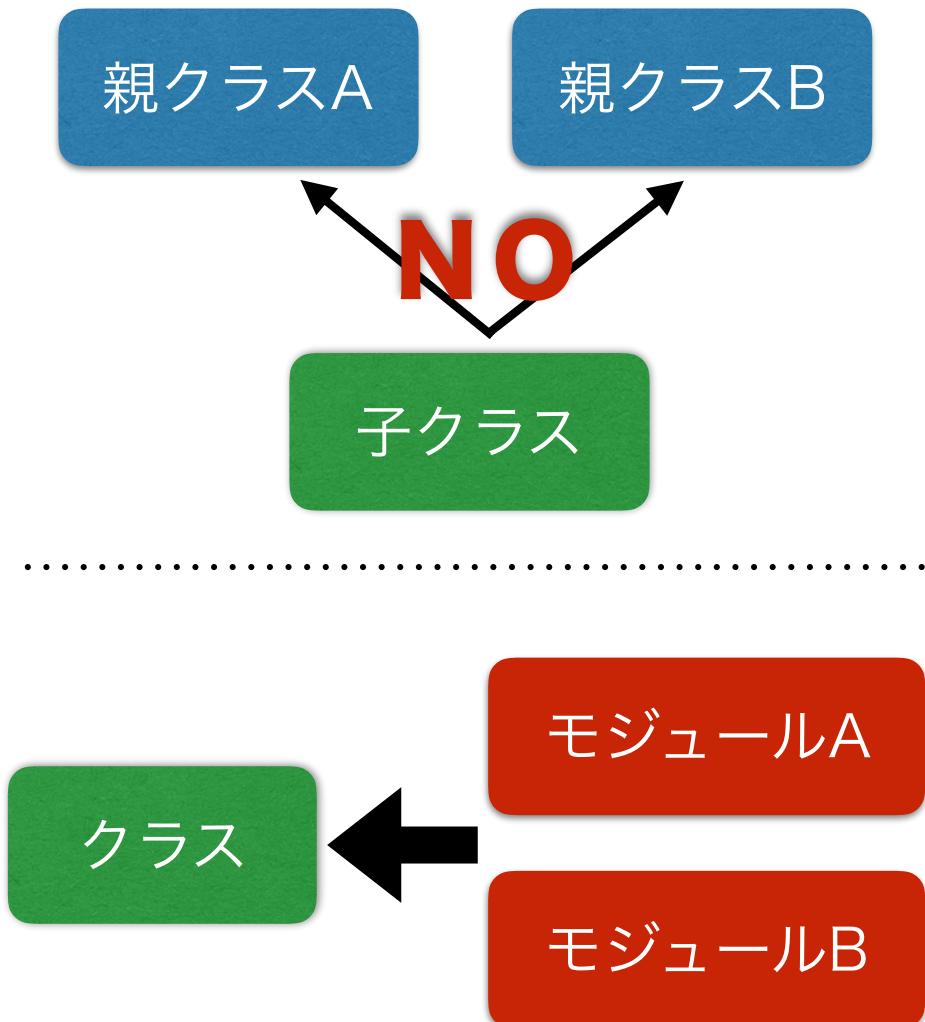
```
end
```

```
son = Hedspi.new("Nguyen Duc Son", 23)
```

```
son.introduce_jp
```

Mix-in

- Rubyは1つの親クラスしか引き継ぐことが出来ない（単純継承）
- そのため、機能を様々なクラスに含める(include)するための構造としてミックス イン **Mix-in**がある
- これにより、簡単に機能拡張ができる



Mix-inの利用

```
class Phone
  @phone_number = "123-456-789"

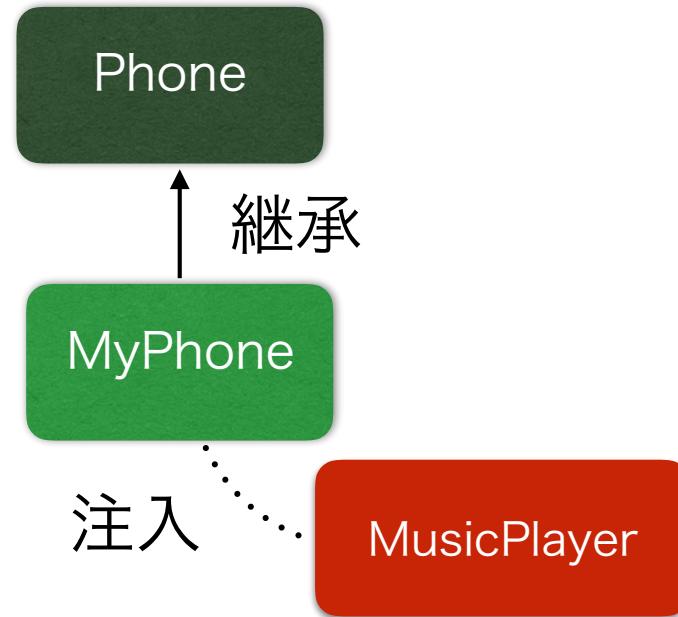
  def tell(tell_number)
    p "Calling to #{tell_number}"
  end
end

module MusicPlayer
  def play_music
    puts "Listening to the music!"
  end
end

class MyPhone < Phone
  include MusicPlayer
end

MyPhone.new().play_music
```

- モジュールはModuleを使って定義する
- クラスに機能を追加する場合は、includeを使う



課題2

課題2

コーヒーの自動販売機をつくる

1. コーヒーは2000vnd

2. saleメソッド

1. 引数は、入力した金額（数字だけ）

2. 引数が2000以上ならば「コーヒー購入」と表示

また、お釣りを表示する

→ 例) 30,000を入力したら、お釣りは10,000vnd

3. 引数が2000未満ならば「お金が足りません」と表示

入力されたお金をそのまま返す

3.infoメソッドで

コーヒーの残量・売上金額を表示

例) コーヒーの残りは3個、売上は12000vndです



課題2

```
# class VendingMachine
  def initialize
    @stock = 10 # コーヒーの残りの数
    @amount = 0 # 支払われた金額
  end

  # コーヒーを購入する
  # 引数1: payment 入力した金額
  # 戻り値: お釣り
  def sale(payment)
    ?
    payment
  end
```

```
# 自動販売機内の情報を表示する
# 引数: なし
# 戻り値: なし
def info()
  # 表示例:
  # "コーヒーの残りは3個、売上は...vndです"
  ?
end

v = VendingMachine.new

p v.sale(10000)
# => お金が足りません
# => 10000

p v.sale(30000)
# => コーヒー購入
# => 10000

v.info
# "コーヒーの残りは9個、売上は20000vndです"
```

課題2

コーヒーの自動販売機をつくる

1. コーヒーは2000vnd

2. saleメソッド

1. 引数は、入力した金額（数字だけ）

2. 引数が2000以上ならば「コーヒー購入」と表示
また、お釣りを表示する

→ 例) 30,000を入力したら、お釣りは10,000vnd

3. 引数が2000未満ならば「お金が足りません」と表示
入力されたお金をそのまま返す

3. infoメソッドで

コーヒーの残量・売上金額を表示

例) コーヒーの残りは3個、売上は12000vndです

```
> ruby -Ku subject2.rb
```

```
お金が足りません
```

```
10000
```

```
コーヒー 購入
```

```
10000
```

```
"コーヒーの残りは9個、売上は20000vndです"
```

課題2

問題2

K58 IT日本語(2) #2-2 Ruby Aクラス

指示があるまで問題を解かないでください。
制限時間は10分間です

次へ

- Google Formを使って課題を出します
メール、またはチャットワークでURLを共有します
- 指示があるまで問題を解かないでください。
- 制限時間は10分間です



今日の授業は終わりです
来週も元気出会いましょう！

Hẹn gặp lại vào tuần tới