基礎プログラミングII

第5回　変数のスコープ・クラス設計（変数のスコープ規則）

メディア情報コース
平居　悠（ひらい　ゆたか）

到達目標

プログラミングを用いた実践的なデータ処理と情報システムの構築

- 定式化された処理を**関数の形**で記述し利用することができるようになる
- 再帰などの**アルゴリズム**を理解し問題に適用できるようになる
- 基礎的な**CGI**の仕組みの理解を通して**Webインターフェース**を設計できるようになる
- 多様な**社会事象への適用**を設計できるようになる
- 現実社会の課題に対応する**情報システム**を設計・作成できるようになる
- **生成AI**を学習の道具として利用できるようになる

前回

第１回　生成AIを効果的に利用した学習法
第２回　メソッド定義と効果的活用（関数的処理）
第３回　メソッド定義と効果的活用（データ集合処理）
第４回　メソッド定義と効果的活用（再起的アルゴリズム）
第５回　変数のスコープ・クラス設計（変数のスコープ規則）
第６回　変数のスコープ・クラス設計（クラス定義）
第７回　CGIと情報システム（CGIの基本要素）
第８回　CGIと情報システム（持続的な値のやり取り）
第９回　CGIと情報システム（実践的処理）
第１０回　チーム作品作成
第１１回　チーム作品の構築・trr試験
第１２回　チーム作品発表・評価
第１３回　合同成果発表会
第１４回　期末試験

前回の目標

再帰的な方法を用いてプログラムを書けるようになる。

前回学んだこと

1. 再帰とは
2. 再帰処理による並べ換え

再帰的定義

何かの定義にそれ自身が登場するもの

再帰を利用したプログラミング

例題：自然数に対する階乗

定義２：再帰的定義

Nの階乗とは

N-1の階乗とNを掛けたものである。ただし、1の階乗は1である。

再帰的メソッドの注意点

- 問題を分割できるとき、分割部分を解く考え方と全体を解く考え方が同じとき、再帰を使える。
- 問題分割を繰り返し、これ以上問題が分割できない場合に結果そのものを返す条件ブロックを入れる（**脱出条件**）
- 分割した問題を再帰的に同じメソッドを呼んで得られた結果を合成したものを最後の結果として返す

クイックソートの手順

大きな箱Xに任意個のデータが入っているとする。

1. Xの中のデータが１個以下なら並べ換え完了。そうでなければ次に進む。
2. Xの中から、任意のデータを１個抜き取り、その値をkとする。
3. kを抜き取ったあとの中のデータを順次見ていき、kより小さかったら左側に置く、そうでなければ右側に置くことをデータがなくなるまで繰り返す。
4. 左側に寄せたものをクイックソートする。
5. 右側に寄せたものをクイックソートする。
6. 左右のかたまりの中間にkを置いて並べ換え完了。

今日の問い

- 再帰とは何か？
  - 何かの中身にそれ自身が現れること。
- 再帰処理を用いて並べ換えを行うにはどのようにすれば良いか？
  - 配列を２つに分け、それぞれを再帰的に並べ換える。

タイピング練習スケジュール

第２回　　ホームポジション
第３回　　ローマ字
第４回　　英語初級
第５回　　日本国憲法
第６回　　ホームポジション
第７回　　ローマ字
第８回　　英語初級
第９回　　日本国憲法
第１０回　日本国憲法
第１１回　日本国憲法（trr試験、合格点：200点）

タイピングの練習 (日本国憲法)

1. ブラウザを起動し、https://www.koeki-prj.org/trr/ に繋ぐ。
2. 学籍番号（Cは大文字、省略なし８桁）を入力する。
3. Koeki MAILに届いたパスコードをPasscode: 欄に入力する。

今回

第１回　生成AIを効果的に利用した学習法
第２回　メソッド定義と効果的活用（関数的処理）
第３回　メソッド定義と効果的活用（データ集合処理）
第４回　メソッド定義と効果的活用（再起的アルゴリズム）
第５回　変数のスコープ・クラス設計（変数のスコープ規則）
第６回　変数のスコープ・クラス設計（クラス定義）
第７回　CGIと情報システム（CGIの基本要素）
第８回　CGIと情報システム（持続的な値のやり取り）
第９回　CGIと情報システム（実践的処理）
第１０回　チーム作品作成
第１１回　チーム作品の構築・trr試験
第１２回　チーム作品発表・評価
第１３回　合同成果発表会
第１４回　期末試験

今回の目標

クラスを用いたプログラムを読めるようになる。

今回学ぶこと

1. 変数のスコープ
2. オブジェクト指向

今回の問い

- 変数のスコープとは何か？
- クラスを定義するにはどのようにすればよいか？

今回学ぶこと

1. 変数のスコープ
2. オブジェクト指向

変数のスコープ

ある場所で設定した変数が有効な範囲

例題設定

次のような簡単なお財布プログラムを考える。

1. 最初の所持金は1000円
2. ユーザからの入力を繰り返し、その数を所持金に足し続ける（マイナスなら減る）。

最も簡単なプログラム

```
#! /usr/bin/env ruby
fund = 1000
while true
   printf(“残高%d円です。何円出しますか（負の数は入金, 0で終了): ”, fund)
   inout = gets.to_i
   break if inout == 0. #ifを文の後ろにつけると1行で書ける
   fund -= inout
end
```

実行例

```
% ./inout.rb
残高1000円です。何円出しますか (負の数は入金, 0で終了): 30
残高970円です。何円出しますか (負の数は入金, 0で終了): 50
残高920円です。何円出しますか (負の数は入金, 0で終了):-100
残高1020円です。何円出しますか (負の数は入金, 0で終了):[C-d]
```

メソッドを使った場合

右のプログラムを実行するとエラーになる。どこが間違っているか？

```
#! /usr/bin/env ruby
fund = 1000

def purse(withdraw)
   fund -= withdraw
end

while true
   printf("残高%d円です。何円出しますか(負の数は入金, 0で終了): ", fund)
   inout = gets.to_i
   break if inout == 0.
   purse(inout)
end
```

```
#! /usr/bin/env ruby
fund = 1000                          #この変数fundは

def purse(withdraw)                  #メソッドpurse内からは見えない
  fund -= withdraw
end

while true
  printf("残高%d円です。何円出しますか（負の数は入金, 0で終了): ", fund)
  inout = gets.to_i
  break if inout == 0.
  purse(inout)
end
```

改良案

1. 重要な変数の値をメソッドの引数と返却値でやり取りする
2. クラス定義を用いて変数と手続きをカプセル化する

```
#! /usr/bin/env ruby
fund = 1000

def purse(total, withdraw)   # 今の総額と引き出し額を受け取る
  total -= withdraw          # その分だけ総額から引く
end

while true
  printf("残高%d円です。何円出しますか（負の数は入金, 0で終了"): ", fund)
  inout = gets.to_i
  break if inout == 0.
  fund = purse(fund, inout) # 総額と支出をpurseに渡し、結果を受け取る
end
```

今回の問い

- 変数のスコープとは何か？
  - ある場所で設定した変数が有効な範囲
- クラスを定義するにはどのようにすればよいか？

授業内課題

次のプログラムは変数のスコープが外れていてうまく動かない。「変数を引数で渡す形にする」ことで、これを修正せよ。

http://roy.e.koeki-u.ac.jp/~yuuji/2025/pf2/scopeOOP/varerr.rb

```
#! /usr/bin/env ruby

waribiki = 1000

def mikiri(price)
   newprice = price*waribiki
   printf("%d円のところ%d円！\n",
price, newprice.to_i)
end

print("元の値段: ")
   x = gets.to_i
   mikiri(x)
```

今回学ぶこと

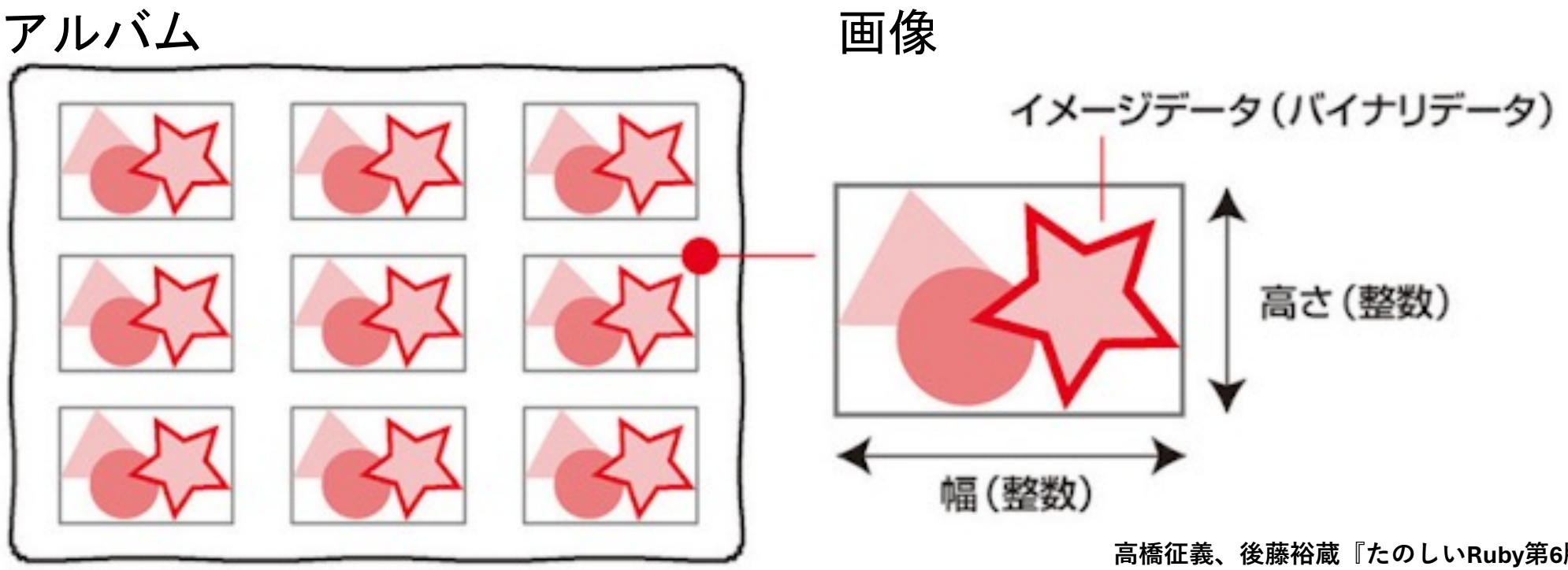
1. 変数のスコープ
2. オブジェクト指向

オブジェクトとは

プログラムの処理の対象（データ）
データとデータを操作するための手続きをまとめたもの

オブジェクト指向プログラミング

ひとまとまりにしたデータをそれぞれオブジェクトとして扱う



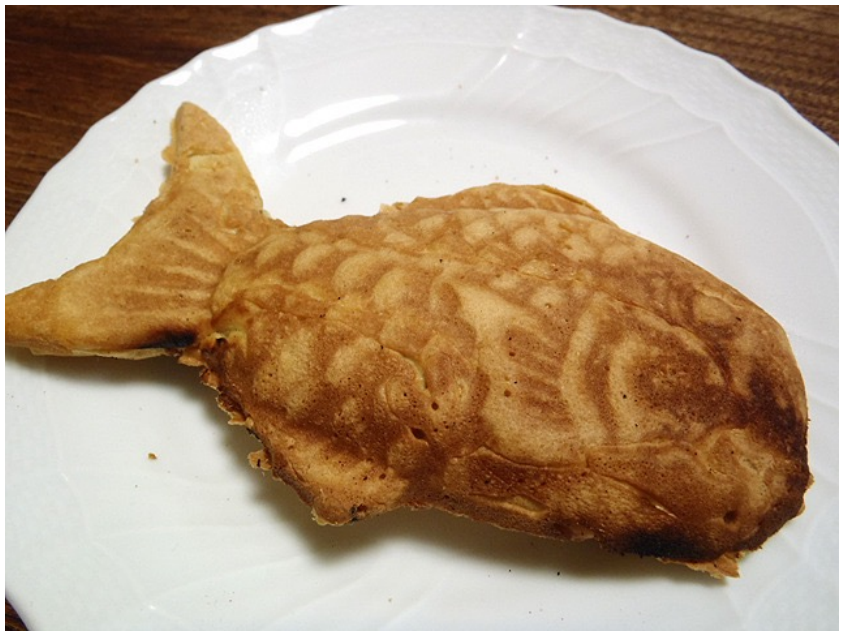
高橋征義、後藤裕蔵『たのしいRuby第6版』

クラスとインスタンス

クラス：ある性質の集合体を設計したもの
インスタンス：実際に動くものとして生み出されたもの

型（クラス）

たい焼き（インスタンス）



https://dailyportalz.jp/kiji/120217153625

お財布プログラムの例題

財布にまつわるお金のやり取りは以下のようにまとめられる。

- 財布に入れるものはお金である。
- 財布は使い始めてから使うのをやめるまでの命である。
- 使い始めるときは最初にいくらか入れた状態で始まる。
- 財布に関する操作は「お金を入れる」と「出す」である。
- 入っている額を超えない範囲で出すことができる。

お財布プログラム：クラスの作成

- 「お財布」を表すクラス名をPurseとする。
- そこに属するのは「中に入る金額」と「入れる操作」と「出す操作」。
- それらを順に変数:fund、メソッド:add、メソッド:subとする。
- 最初にお財布を導入するときの初期金額を設定できる。

クラス定義方法

```
class クラス名
  クラスの定義
end
```

initializeメソッド

```
def initialize(start=0)
  @fund = start
end
```

@で始まる変数はインスタンス変数といい、クラス内で
全てのメソッドから参照できる。

「お財布」を表すクラス

クラス定義も定義しただけでは何も起きず、メソッドと同様、プログラム内で利用する必要がある。

purse.rb

```
#! /usr/bin/env ruby
class Purse
 def initialize(start=0)
 @fund = start
 end
 def add(amount)
  if amount < 0
   STDERR.printf("マイナスはだめよ(%d)\n", amount)
  else
   @fund += amount
   printf("%d円入れました。\n", amount)
  end
 end
 def sub(amount)
  if amount > @fund
   STDERR.printf("そんなに入っていません(残高: %d)\n", @fund)
  else
   @fund -= amount
   printf("%d円出しました。\n", amount)
  end
 end
 def howmuch()                    # 現在の残高を返す
  @fund
 end
end
```

purse.rbを利用するプログラム

```
#! /usr/bin/env ruby
require_relative "purse.rb"                    # 同じディレクトリあるときは require_relative

puts "=== 今日からこの財布を使おう。5000円入れておこう: Purse.new(5000)"
saifu = Purse.new(5000)                        # .newで定義したクラスの実体を生成する。
# 生成するときに initialize メソッドが呼ばれる。

puts "=== 1598円の買い物をしよう。"
puts "=== 2000円出すぞ: saifu.sub(2000)"
saifu.sub(2000)
puts "=== おつりを402円もらった。しまおう: saifu.add(402)"
saifu.add(402)
puts "=== いまいくらあるんだろう: saifu.howmuch"
printf("残り%d円です\n", saifu.howmuch)
```

purse.rbを利用するプログラム

```
#! /usr/bin/env ruby
require_relative "purse.rb"
puts "=== 今日からこの財布を使おう。5000円入れておこう: Purse.new(5000)"
saifu = Purse.new(5000)

puts "=== 1598円の買い物をしよう。"
puts "=== 2000円出すぞ: saifu.sub(2000)"
saifu.sub(2000)
puts "=== おつりを402円もらった。しまおう: saifu.add(402)"
saifu.add(402)
puts "=== いまいくらあるんだろう: saifu.howmuch"
printf("残り%d円です\n", saifu.howmuch)
```

saifu = Purse.new(5000)
定義したPurseクラスの性質を持つインスタンスを一つ生成する(.new)。このとき引数として5000が渡され、それがクラス定義にあるinitializeメソッドに渡される。残高変数@fundに5000が格納された状態でこの財布の利用が始まる。

purse.rbを利用するプログラム

```
#! /usr/bin/env ruby
require_relative "purse.rb"
puts "=== 今日からこの財布を使おう。5000円入れておこう: Purse.new(5000)"
saifu = Purse.new(5000)

puts "=== 1598円の買い物をしよう。"
puts "=== 2000円出すぞ: saifu.sub(2000)"
saifu.sub(2000)
puts "=== おつりを402円もらった。しまおう: saifu.add(402)"
saifu.add(402)
puts "=== いまいくらあるんだろう: saifu.howmuch"
printf("残り%d円です\n", saifu.howmuch)
```

saifu.sub(2000)
saifu変数にはPurseクラスから生まれたオブジェクトが入っている。このオブジェクト内にあるsubメソッドを呼ぶ。

purse.rbを利用するプログラム

```
#! /usr/bin/env ruby
require_relative "purse.rb"
puts "=== 今日からこの財布を使おう。5000円入れておこう: Purse.new(5000)"
saifu = Purse.new(5000)

puts "=== 1598円の買い物をしよう。"
puts "=== 2000円出すぞ: saifu.sub(2000)"
saifu.sub(2000)
puts "=== おつりを402円もらった。しまおう: saifu.add(402)"
saifu.add(402)
puts "=== いまいくらあるんだろう: saifu.howmuch"
printf("残り%d円です\n", saifu.howmuch)
```

saifu.add(402)
同様にオブジェクト内にあるaddメソッドを呼ぶ

複数のインスタンス（オブジェクト）

一つのクラスから生まれるインスタンスは全て違う個体で、独立した情報を持ち、それに従った動きをする。

```
chiharu_saifu = Purse.new(5000) # 千春さんが財布を新しくして5000円から開始
hibiki_saifu  = Purse.new(8000) # 響さんが財布を新しくして8000円から開始
yu_saifu      = Purse.new(1000) # 優さんが財布を新しくして1000円から開始
```

複数のインスタンス（オブジェクト）

chiharu_saifu	
@fund変数	5000 (残高)
addメソッド	入れる処理
subメソッド	出す処理
howmuch メソッド	残高を返す処理

hibiki_saifu	
@fund変数	8000 (残高)
addメソッド	入れる処理
subメソッド	出す処理
howmuch メソッド	残高を返す処理

yu_saifu	
@fund変数	1000 (残高)
addメソッド	入れる処理
subメソッド	出す処理
howmuch メソッド	残高を返す処理

変数の隠蔽

クラス定義(classからendまで)の内側で使用された変数はその外側からは一切見えなくなること。プログラム同士が同じ変数名やメソッド名（まとめて**シンボル**という）で競合して異常動作する可能性をなくせる。

カプセル化

シンボルを隠蔽してプログラムの一定の部分の独立性を高めること。複数人、あるいは将来別のプログラムから利用することを見越したプログラムを作るときには欠かせない。

継承

既に定義されたクラスの基本機能は踏襲しつつ、機能や性質を追加すること。

継承方法

```
class クラス名 < スーパークラス名
    クラスの定義
end
```

「お財布」にカードを入れる機能を追加

purse-with-card.rb

```
require_relative "purse.rb”

class PurseWithCard < Purse
 def initialize(money=0, card=[])
  super(monkey)
  @card = card
 end
 def putinCard(card)
  @card << card
  printf("%s をしまいました。\n", card)
 end
 def drawCard(card)
  @card.delete(card)
  printf("%s を取り出しました。\n", card)
 end
 def myCards()
  @card.join(",")
 end
end
```

今回の問い

- 変数のスコープとは何か？
  - ある場所で設定した変数が有効な範囲
- **クラスを定義するにはどのようにすればよいか？**
  - **class クラス名で始め、endで閉じる。**

今回学んだこと

1. 変数のスコープ
2. オブジェクト指向

今回の目標

クラスを用いたプログラムを読めるようになる。

課題

中間試験の問題を完全にして提出せよ。試験で正解を得た場合は別の問題で。

s4基礎プロII(F)の「#05 pf2水 課題 (復習中間試験)」に指示どおり書き込む。提出後修正して良い。その場合は「編集」リンクから書き換えること。

締切：**11月1日 (土)**

事前課題

quiz ruby-cgiを3回以上行い、最も早くクリアした記録の結果をs4基礎プロII(F)の「#06 pf2 水 事前課題 (quiz cgi)」に書き込む。

締切：**11月10日 (月)**