基礎プログラミングII

第2回　メソッド・繰り返し処理

メソッドとは

何かの「手順」を行い、その結果を返してくれるもの。

(例) printf, gets, to_i

メソッドとは

数学における関数のようなもの

例

$$f(x) = 2x + 1$$

f(4)を呼び出すと2×4+1を計算できる。fに与えた4のことを**引数（ひきすう）**という。

メソッドの定義方法

メソッドを定義するには**def**を用いる。

```
def メソッド名(引数リスト)
  定義本体
end
```

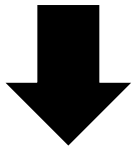
メソッドの定義例

```
def f(x)
   2 * x + 1
end
```

メソッドの定義例

```
def f(x)
```

()内の引数リストはxのみ



「メソッドfを呼ぶときは引数を1つだけ付ける」という意味。メソッド定義の引数リストに書く変数は**仮引数**といい、メソッドを呼んだときに与えた値が代入される。

メソッドの定義例

```
2 * x + 1
```

メソッド定義本体。

printf(“2*%d + 1は %dです\n”, x, 2*x+1)
のようにRubyのメソッドを書いても良い。

メソッドの定義例

```
end
```

メソッド定義の終了を示す。

メソッドの参照と返却値

メソッドを参照するには、参照したい位置で

```
メソッド名(引数)
```

と書く。例えば、先ほど定義したfメソッドを呼ぶには、

```
f(3)
```

などとする。

メソッドの参照と返却値

メソッドを実行するとき最後に実行した文の値がメソッドの実行結果として返される。これを**返却値**という。

引数を２つ以上利用する場合

```
def foo(x, y)
  x + y * 2
end
```

のように引数リストで仮引数をカンマ(,)で区切って列挙する。呼び出す際はfoo(2, 5)のようにする。

引数が不要の場合

```
def bar()
  ...なんらかの内容...
end
```

のように引数リストの括弧内は空にする。

return文

メソッドの中で呼ぶと、メソッドの返却値を指定できる。

```
def daikei(jotei, katei, takasa)
  if jotei < 0 || katei < 0 || takasa < 0
    return nil
  end
  return (jotei+katei)*takasa/2
end
```

jotei, katei, takasaのいずれかが負の場合nilを返しメソッドを抜ける

仕事をするのみのメソッド

```
def hello(who)
  pirntf(“%s さんこんにちは！\n”, who)
end
```

消費税計算

金額を渡すと消費税を計算するメソッド

例：金額250円、税率10%なら

tax (250, 1.10)

とする

```
#! /usr/koeki/bin/ruby
# coding: utf-8

                金額    税率
def tax(price, ratio)
  charge = price*ratio
  charge.to_i
end
```

引数のデフォルト値

特に何も指定しなければ採用される値のことを
デフォルト値という。

```
def tax(price, ratio=1.10)
  charge = price*ratio
  charge.to_i
end
```

ratioの値を省略したときは1.10として計算する。

引数省略時の注意点

引数省略時のデフォルト値は、引数リストの後ろの方からしか使えない。

OK

```
def tax(price=1000, tax=1.1)
def tax(price, tax =1.10)
```

NG

```
def tax(price=1000, tax)
```

定義したメソッドを呼ぶときは引数を順番通り渡す必要があるため、前の方の引数を省略して後ろの引数を指定できない

メソッドに配列を渡す例

配列をメソッドに渡すこともできる。

右の例では、数値が複数入っている配列を受け取り、それらの値の平均値を求める。

```
def average(score)
  sum = 0.0
  score.each do |x|
    sum += x
  end
  sum/score.length
end
```

いろいろな繰り返し (while以外)

- **times**: 一定回数の繰り返し
- **upto, downto**: 数え上げ、カウントダウン
- **each, for**: 要素に対する繰り返し
- **step**: 飛び飛びの繰り返し

times: 決められた回数だけ繰り返す

繰り返したい回数を***N***とすると、以下のように記述する。

```
N.times do
  ...繰り返し本体...
end
```

upto, downto: 整数を数えながらの繰り返し

whileを使った場合 (復習)

```
sum = 0
i = 1
while i <= 10
  sum += i
  i += 1
end

printf(“合計は %d です\n”, sum)
```

uptoを使った場合

```
sum = 0
1.upto(10) do |x|
  sum += x
end

printf(“合計は %d です\n”, sum)
```

each, for: 配列要素すべてに対する繰り返し

配列の各要素1つ1つ取り出しながらの繰り返し

配列.each do |**変数**|
　...上記の**変数**を使った処理...
end

for **変数** in **配列**
　...上記の**変数**を使った処理...
end

配列を昇順にした結果を与える場合

```
配列.sort.each do |変数|
  ...上記の変数を使った処理...
end
```

step: 飛び飛びの繰り返し

1, 3, 5, 7, 9...や10, 20, 30のように数値を一定間隔で変えながら利用したい場合はstepメソッドを用いる。

```
開始値.step(終了値[, 増加値]) do |変数|
  ...変数を利用した処理
end
```

増加値は省略できて、省略した場合は1とみなされる。

まとめ

- メソッドとは何かの「手順」を行い、その結果を返してくれるもの。
- メソッドを定義して計算や仕事をさせることができる。
- 繰り返し処理にはwhile以外にもtimes, upto, each, stepなどがあり、用途に応じて使い分ける。