プログラミング言語　第十一回

担当：篠沢 佳久
櫻井　彰人

平成20年 7月 7日

本日の内容

- 関数（メソッド）
  - 関数の定義
  - 関数の呼び出し
  - 実引数と仮引数
- 関数とサブルーチン

関数とは

関数の定義と呼び出し

関数*とは何か

- （普通のプログラム言語では）
  面倒な計算を（どこかで）やってくれるもの
  例：
  - 正弦関数：Math.sin(x)
  - 対数関数：Math.log(x)

```
x = Math.sin(Math.log(3.0))
puts( x )
```

*Ruby においてはメソッドとも呼びます

同じ処理の繰り返し①

同じ処理

```
x = 3; y = 5
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and #{y}" )

x = 10; y = 15
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and

x = -5; y = -9
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and #{y}" )
```

二つの整数値の大きい値を求める

```
C:¥Ruby>ruby sample.rb
5 is the maximum of 3 and 5
15 is the maximum of 10 and 15
-5 is the maximum of -5 and -9
```


ループを利用しての解決方法

二次元配列

```
a= [
    [ 3,5 ],
    [ 10,15 ],
    [ -5,-9]
]

3.times{ |i|
    x = a[ i ][ 0 ]
    y = a[ i ][ 1 ]
    if x >= y then
        max = x
    else
        max = y
    end
    puts( "#{max} is the maximum of #{x} and #{y}" )
}
```

```
C:¥Ruby>ruby sample.rb
5 is the maximum of 3 and 5
15 is the maximum of 10 and 15
-5 is the maximum of -5 and -9
```

同じ処理の繰り返し②

```
x = 3; y = 5
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and #{y}" )

# 何かの処理

x = 10; y = 15
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and #{y}" )

# 何かの処理
```

```
x = -5; y = -9
if x >= y then
    max = x
else
    max = y
end
puts( "#{max} is the maximum of #{x} and #{y}" )
```

別の処理が入ってしまってループ化できない場合もある

関数の定義は誰でもできる

- 2数の最大値を値とする関数 max(x,y) を定義してみる

```
def max( x, y )
  if  x >= y then
    return x
  else
    return y
  end
end

i = 3; j = 5
x = max( i, j )
puts( "#{x} is the maximum of #{i} and #{j}" )
```

仮引数

実引数

```
5 is the maximum of 3 and 5
```


関数の定義

```
def  関数名( 仮引数1, 仮引数2,  ･･･ ,仮引数n )
     実行したい処理
     return 返す値
end
```

```
def max( x, y )
  if  x >= y then
    return x
  else
    return y
  end
end
```

```
def average( x, y )
    z = (x+y)/2
    return z
end
```


関数の呼び出し

関数名(実引数1, 実引数2, ･･･, 実引数n)

```
i = 3; j = 5
x = max( i, j )
puts( "#{x} is the maximum of #{i} and #{j}" )
```

```
a = 9; b = 3
x = average( a, b )
puts( "#{x} is the average of #{a} and #{b}" )
```


引数の受け渡し

```
i = 3; j = 5
x = max( i, j )
```

i=3, j=5 として関数に渡される

```
def max( x, y )
  if  x >= y then
    return x
  else
    return y
  end
end
```

x=3, y=5 として処理する

必ず呼び出す関数の仮引数と一致する順番に実引数を記述する

関数の戻り値

```
def max( x, y )
  if  x >= y then
    return x
  else
    return y
  end
end
```

x=3, y=5 なので y の値を返す

```
i = 3; j = 5
x = max( i, j )
```

関数の戻り値は5なのでxには5が代入される

return 変数名
return 値
関数中において,値を戻す

関数の定義と呼び出し①

```
def max( x, y )
  if  x >= y then
    return x
  else
    return y
  end
end

i = 3; j = 5
x = max( i, j )
puts( "#{x} is the maximum of #{i} and #{j}" )

i = 10; j = 15
x = max( i, j )
puts( "#{x} is the maximum of #{i} and #{j}" )

i = 5; j = -9
x = max( i, j )
puts( "#{x} is the maximum of #{i} and #{j}" )
```

関数の定義

関数の呼び出し

関数の定義と呼び出し②

```
def f( x, y )
  z = x*x+y*y
  return z
end

(0..9).each{ |x|
   (0..9).each{ |y|
        print( " x = " , x , " y= " , y , " f= " , f( x , y ) , "¥n" )
   }
}
```

関数の定義

関数の呼び出し

関数の定義と呼び出し③

```
def f( x, y, z )
  w = x*x+y*y+z*z
  return w
end

(0..9).each{ |x|
   (0..9).each{ |y|
        (0..9).each{ |z|
        print( " x = " , x , " y= " , y , " z= " , z , " f= " , f( x , y , z ) , "¥n" )
        }
   }
}
```

関数の定義

関数の呼び出し

2数の最小値を値とする関数 min(x,y)

関数の定義

```
def min( x, y)
  if  x <= y then
    return x
  else
    return y
  end
end
```

関数の呼び出し

```
i = 3; j = 5
x = min( i, j )
puts( "#{x} is the minimum of #{i} and #{j}" )
```


3数の最大値を値とする関数 max(x,y,z)

関数の定義

```
def max( x, y, z )
  if ( x >= y ) then
    if ( y >= z ) then
      return x
    else
      if ( x >= z ) then
        return x
      else
        return z
      end
    end
```

```
else
  if ( x >= z ) then
    return y
  else
    if ( y >= z ) then
      return y
    else
      return z
    end
  end
end
```

関数の呼び出し

```
i = 3; j = 5; k = 7
x = max( i, j, k )
puts( "#{x} is the maximum of #{i}, #{j}, and #{k}" )
```


関数が複数個定義されている場合

- 複数個あっても大丈夫

```
def max( x, y)
  if  x >= y then
    return x
  else
    return y
  end
end
def min( x, y)
  if  x <= y then
    return x
  else
    return y
  end
end
i = 3; j = 5
puts( "#{max( i, j )} is the maximum of #{i} and #{j}" )
puts( "#{min( i, j )} is the minimum of #{i} and #{j}" )
```

```
5 is the maximum of 3 and 5
3 is the minimum of 3 and 5
```

関数は関数を使えるか？①

- 勿論、使える

```
def max2( x, y)
  if  x >= y then
    return x
  else
    return y
  end
end
```

```
5 is the maximum of 3 and 5
7 is the maximum of 3, 5, and 7
```

```
def max3( x, y, z )
  return max2( x, max2( y, z) )
end

i = 3; j = 5; k = 7
puts( "#{max2( i, j )} is the maximum of #{i} and #{j}" )
puts( "#{max3( i, j, k )} is the maximum of #{i}, #{j}, and #{k}" )
```


関数は関数を使えるか？①

i = 3; j = 5; k = 7
max3(i, j, k)

↓

max2(3, max2(5, 7))

↓

max2(3, 7)

↓

戻り値は7

関数は関数を使えるか？②

- 勿論、使える

```
def max2( x, y)
  if  x >= y then
    return x
  else
    return y
  end
end
```

```
5 is the maximum of 3 and 5
9 is the maximum of 3, 5, 7, and 9
```

```
def max4( x, y, z, u )
  return max2( max2( x, y) , max2( z, u) )
end

i = 3; j = 5; k = 7; l = 9
puts( "#{max2( i, j )} is the maximum of #{i} and #{j}" )
puts( "#{max4( i, j, k, l )} is the maximum of
#{i}, #{j}, #{k}, and #{l}" )
```


関数は関数を使えるか？②

i = 3; j = 5; k = 7; l = 9
max4(i, j, k, l)

↓

max2(max2(3, 5) , max2(7, 9))

↓

max2(5 , 9)

↓

戻り値は9

(注意!) 変数に値を戻すことはできない①

```
def f( x , y )
   x = x + 1
   y = y + 1
end

a = 3
b = 5
print( " a = " , a , " " , " b= " , b , "¥n" )
f( a , b )
print( " a = " , a , " " , " b= " , b , "¥n" )
```

f(a,b) を実行した後でも変数の値は変化しない

```
C:¥Ruby>ruby sample.rb
 a = 3  b= 5
 a = 3  b= 5
```


(注意!) 変数に値を戻すことはできない②

```
def f( x , y )
   x = x + 1
   y = y + 1
end

x = 3
y = 5
print( " x = " , x , " " , " y= " , y , "¥n" )
f( x , y )
print( " x = " , x , " " , " y= " , y , "¥n" )
```

f(x,y) を実行した後でも変数の値は変化しない

```
C:¥Ruby>ruby sample.rb
 x = 3  y= 5
 x = 3  y= 5
```

戻り値のない関数（サブルーチン）

戻り値のない関数

- 「戻り値のない関数」は関数とは言い難いが、大目にみてください。
  - サブルーチンとか副プログラムと言うのが正しい。
- 仕事だけする。例えば、印字のみ。

```
def sayHello( n )
  n.times{ |i|
    i.times{
      print( " " )
    }
    puts( "Hello!" )
  }
end

sayHello( 4 )
```

```
Hello!
 Hello!
  Hello!
   Hello!
=> 4
```


サブルーチンと関数

- 違いは、
  - 関数：戻り値あり
  - サブルーチン：戻り値がない
- その結果、使い方が違う。
  - 関数: 式の中
  - サブルーチン: 一つの文として
- 本講義では区別なしに用いる

戻り値のない関数の例①

```
def max( x, y )
  if  x >= y then
    z = x
  else
    z = y
  end
  puts( "#{z} is the maximum of #{x} and #{y}" )
end

i = 3; j = 5
max( i, j )

i = 9; j = 15
max( i, j )

i = -5; j = -7
max( i, j )
```

```
C:¥Ruby>ruby sample.rb
5 is the maximum of 3 and 5
15 is the maximum of 9 and 15
-5 is the maximum of -5 and -7
```

戻り値のない関数の例②

```
def max( x, y )
  if  x >= y then
    z = x
  else
    z = y
  end
  puts( "#{z} is the maximum of #{x} and #{y}" )
end

i = 3; j = 5
k = max( i, j )
print( " k= " , k , "¥n" )
```

```
C:¥Ruby>ruby sample.rb
5 is the maximum of 3 and 5
k= nil
```

変数kには何も代入されない

戻り値のない関数の例③

戻り値のある関数

```
def f( x )
    if x % 2 == 0 then
        return true
    else
        return false
    end
end

n = 15
if f( n ) == true then
    print( n , "は偶数です¥n" )
else
    print( n , "は奇数です¥n" )
end
```

戻り値のない関数

```
def f( x )
    if x % 2 == 0 then
        print( x , "は偶数です¥n" )
    else
        print( x , "は奇数です¥n" )
    end
end

n = 15
f( n )
```

```
C:¥Ruby>ruby sample.rb
15は奇数です
```

引数も戻り値もない関数

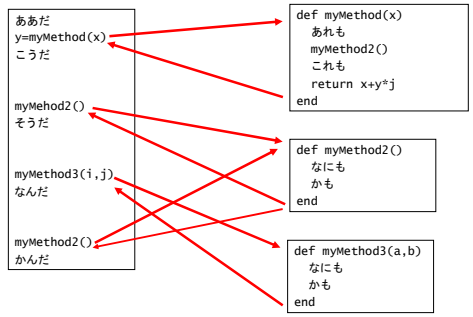
```
def max()
  x = 3
  y = 5
  if  x >= y then
    z = x
  else
    z = y
  end
  puts( "#{z} is the maximum of #{x} and #{y}" )
end

max()
```

引数がない

```
C:¥Ruby>ruby sample.rb
5 is the maximum of 3 and 5
```


関数のまとめ



関数の例

配列を引数として受け取る関数
配列を戻り値として返す関数

配列を引数として受け取る関数①（平均を求める関数）

配列を引数として受け取る

```
def average( a )
  s=0.0
  a.each{|x|
       s += x
  }
  return s/a.length
end
```

整数値を返す

```
def average( a )
  s=0.0
  (0..a.length-1).each{|i|
    s += a[i]
  end
  return s/a.length
end
```

整数値を返す

```
x = [1,2,3,4,5,6,7,8,9,10]
print( "#{average( x )} is the average of: "); p( x )
```

配列を引数として渡す

```
5.5 is the average of: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

配列を引数として受け取る関数②（分散を求める関数）

```
def average( a )
  s=0.0
  a.each{|x|
       s += x
  }
  return s/a.length
end

def var( a )
  ave = average( a )
  s = 0.0
  a.each{ |x|
    s += ( x - ave ) * ( x - ave )
  }
  return s /(a.length-1)
end

x = [1,2,3,4,5,6,7,8,9,10]
print( "#{average( x )} is the average of: "); p( x )
print( "#{var( x )} is the variance of: "); p( x )
```

配列を引数として受け取る

```
C:¥Ruby>ruby sample.rb
5.5 is the average of: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
9.166666666666667 is the variance of: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

配列を引数として渡す

配列を引数として受け取る関数③内積を求める関数

```
def inner_product( a , b )
  s = 0
  a.length.times{ |i|
    s += a[ i ] * b[ i ]
  }
  return s
end

x = [1,2,3,4,5]
y= [6,7,8,9,10]
print( inner_product( x , y ) , "¥n" )
```

配列を引数として受け取る

配列を引数として渡す

配列を戻り値として返す関数①

```
def add( a , b )
  c = []
  a.length.times{ |i|
    c[ i ] = a[ i ] + b[ i ]
  }
  return c
end

x = [1,2,3,4,5]
y= [6,7,8,9,10]
z = add( x , y )
p z
```

配列を戻り値として返す

ベクトル x と y との和

```
C:¥Ruby>ruby sample.rb
[7, 9, 11, 13, 15]
```


配列を戻り値として返す関数②

```
def min_max( a )
  max = a[ 0 ]
  min = a[ 0 ]
  (1..a.length-1).each{ |i|
    if max < a[ i ] then
          max = a[ i ]
    end
    if min > a[ i ] then
          min = a[ i ]
    end
  }
  c = []
  c[ 0 ] = min
  c[ 1 ] = max
  return c
end

x = [1,2,5,3,4]
z = min_max( x )
p z
```

最小値を c[0]
最大値を c[1]
に格納し,配列cを戻り値として返す

```
C:¥Ruby>ruby sample.rb
[1, 5]
```


二次元配列を引数として受け取る関数①（二次元配列の要素の平均値を求める）

```
def Average( x )
  s = 0.0
  count = 0
  (0..x.length-1).each{ |i|
    (0..x[i].length-1).each{ |j|
      s += x[i][j]
      count += 1
    }
  }
  return s/count
end

d = [
    [1,2,3],
    [4,5,6],
    [7,8,9]
   ]
a = Average( d )
print( a , "¥n" )
```

二次元配列を引数として受け取る

```
C:¥Ruby>ruby sample.rb
5.0
```

二次元配列を引数として渡す

二次元配列を引数として受け取る関数②（二次元配列中の各行の平均を求める）

```
def getAverage( x )
  y = Array.new( x.length )
  (0..x.length-1).each{ |i|
    s = 0.0
    (0..x[i].length-1).each{ |j|
      s +=  x[i][j]
    }
    y[i] = s/x.length;
  }
  return y
end

d = [
    [1,2,3],
    [4,5,6],
    [7,8,9]
   ]
a = getAverage( d )
(0..d.length-1).each{ |i|
  print( "#{a[i]} is the average of: ")
  p( d[i] )
}
```

二次元配列を引数として受け取る

一次元配列を戻り値として返す

```
C:¥Ruby>ruby sample.rb
2.0 is the average of: [1, 2, 3]
5.0 is the average of: [4, 5, 6]
8.0 is the average of: [7, 8, 9]
```

二次元配列を引数として渡す

二次元配列を戻り値として返す関数（行列のスカラー倍）

```
def Mul( x , y )
  z = []
  (0..x.length-1).each{ |i|
    z[ i ] = []
    (0..x[i].length-1).each{ |j|
      z[ i ][ j ] = y * x[i][j]
    }
  }
  return z
end

c = [
    [1,2,3],
    [4,5,6],
    [7,8,9]
   ]
d = 3
a = Mul( c , d )
p a
```

x 二次元配列
y 整数値

二次元配列を戻り値として返す

```
C:¥Ruby>ruby sample.rb
[[3, 6, 9], [12, 15, 18], [21, 24, 27]]
```

c 二次元配列
d 整数値

変数に値を戻すことはできない

```
def getAverage( x, y )
  s = 0.0
  (0..x.length-1).each{ |j|
      s += x[j]
  }
  y = s/x.length;
end

d = [1,2,3]
a = 0.0
getAverage( d, a )
print( "#{a} is??? the average of: "); p( d )
```

配列dの要素の平均値を変数aに入れて欲しい…

```
0.0 is??? the average of: [1, 2, 3]
```

でも配列に結果を戻すことはできる

```
def  getAverage( x,  y )
  (0..x.length-1).each{ |i|
    s = 0.0
    (0..x[i].length-1).each{ |j|
      s = s + x[i][j]
    }
    y[i] = s/x.length
  }
end

d = [ [1,2,3], [4,5,6], [7,8,9] ]
a = Array.new( d.length )

getAverage( d, a )

(0..d.length-1).each{ |i|
  print( "#{a[i]} is the average of: ")
  p( d[i] )
}
```

二次元配列dの各行ごとの要素の平均値を配列aに入れる

```
2.0 is the average of: [1, 2, 3]
5.0 is the average of: [4, 5, 6]
8.0 is the average of: [7, 8, 9]
=> 0..2
```


練習問題

練習問題①～⑤

練習問題①

- p円を利率r%でn年間預金した場合,戻り金を求める関数を書きなさい.

```
def f( m , r , n )
    関数の定義をしなさい
end

m = 10000
r = 0.05
n = 2
printf( "%d円を利率%3.2fで%d年間預けた時の戻り金額は%d円です¥n" ,
    m , r , n , f( m , r , n ) )
```


練習問題②

- 練習問題①の関数を用いて，m = 10000，r = 0.05 の時，戻り金が2倍になる年数を求めるプログラムを書きなさい

```
def f( m , r , n )
    練習問題①の答え
end

m = 10000
r = 0.05
    プログラムを記述しなさい
print( "2倍になるのは" , n , "年後です¥n" )
```


練習問題③

- 正の整数xの階乗を求める関数を記述しなさい.

```
def f( x )
        関数の定義をしなさい
end

x = 6
print( x , " の階乗は" , f( x ) , "です¥n" )
```


練習問題④

- 練習問題②の関数を用いて，コンビネーション（${}_nC_r$）（n>0,n>=r）を求める関数を記述しなさい.

```
def f( x )
        練習問題②の答え
end

def comb( n , r )
        関数の定義をしなさい
end

n = 10
r = 5
printf( "%dC%dは%dです¥n" , n , r , comb( n , r ) )
```

練習問題⑤

行列(2次元配列)の加算、乗算、転置を行う関数(add, multiply, transpose)を書きなさい。行列(2次元配列)を行列らしく印字する(印字の一行には行列の一行を印字する)、戻し値のない関数を書きなさい。そして、それらを利用して、計算、印字する例題プログラムを書き、実行しなさい。

```
add( transpose(a), multiply( a, b ) ) )
```

検証は、(本当はこれでは不十分なのだが)極端な場合(単位行列とか対称行列とか)やそれらの組み合わせ、乱数などを用いて行う。

※1 計算結果を関数の戻り値として返す。メモリの使用効率は非常に悪い(従って、実際に使うのは、小規模の場合を除き、お奨めできない)のですが、まあ、ご勘弁。

練習問題⑤の大枠

```
def add( x, y )
  a = Array.new( x.length ).fill{ |i| Array.new( x[i].length ) }
# ここで、加算
  return a
end
def multiply( x, y )
  a = Array.new( x.length).fill{ |i| Array.new( x[i].length ) }
# ここで乗算
  return a
end
def transpose( x )
  a = Array.new( x.length).fill{ |i| Array.new( x[i].length ) }
# ここで転置
 return a
end
def printMatrix( x )
# ここで印字
end
```

x と同じ行数，列数の二次元配列を作成

練習問題⑤の大枠

前ページの続き

```
a = [[1,2,3],[4,5,6],[7,8,9]]
b = [[10,20,30],[40,50,60],[70,80,90]]
printMatrix( add( transpose(a), multiply(a,b) ) )
```

実行結果

```
C:¥Ruby>ruby sample.rb
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
[[300.0, 360.0, 420.0], [660.0, 810.0, 960.0], [1020.0, 1260.0, 1500.0]]
[[301.0, 364.0, 427.0], [662.0, 815.0, 968.0], [1023.0, 1266.0, 1509.0]]
301.0 364.0 427.0
662.0 815.0 968.0
1023.0 1266.0 1509.0
```


配列の初期値の設定方法

- Ruby には次のような方法がある

```
irb(main):001:0> a = Array.new(5).fill(0)
=> [0, 0, 0, 0, 0]
irb(main):002:0> a = Array.new(5).fill(0.0)
=> [0.0, 0.0, 0.0, 0.0, 0.0]
irb(main):003:0> a = Array.new(5).fill( "a" )
⇒["a", "a", "a", "a", "a"]
irb(main):005:0> a = Array.new(3).fill{ Array.new(3) }
=> [[nil, nil, nil], [nil, nil, nil], [nil, nil, nil]]
```

3行3列の二次元配列を作成

練習問題

- 練習問題①～⑤を(できるだけ)(頑張って)行ないなさい。
- プログラムと実行結果をワープロに貼り付けて、keio.jp から提出して下さい。

連絡事項

- 第三回レポート
  - 来週(7/14)講義開始前までに提出です
- 来週は講義のまとめの練習問題を行ないます
  - レポート一回分の評価としますので必ず出席して下さい＆これまでの復習をして来て下さい