# プログラミング言語　第十二回

担当：篠沢 佳久
櫻井　彰人

平成21年 7月 6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

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

```
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



## 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

関数の定義

```
def max( 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 = 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}" )
```



# 関数の定義と呼び出し②

関数の定義

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

関数の呼び出し

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



# 関数の定義と呼び出し③

関数の定義

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

関数の呼び出し

```
(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" )
        }
   }
}
```



# 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" )
```

x=3, y=5の値が渡される

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

f(a,b) を実行した後でも変数の値は変化しない



## (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" )
```

x=3, y=5の値が渡される

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

f(x,y) を実行した後でも変数の値は変化しない

# 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
```



## 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]
9.16666666666667 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



## 関数のまとめ

```
ああだ
y=myMethod(x)
こうだ

myMehod2()
そうだ

myMethod3(i,j)
なんだ

myMethod2()
かんだ
```

```
def myMethod(x)
  あれも
  myMethod2()
  これも
  return x+y*j
end
```

```
def myMethod2()
  なにも
  かも
end
```

```
def myMethod3(a,b)
  なにも
  かも
end
```



## 練習問題

練習問題①～④

## 練習問題①

- m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

- 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③

- 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.

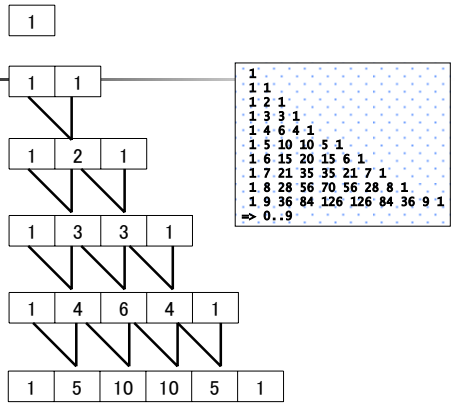
```
def f( x )
        練習問題②の答え
end

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 ) )
```



## 問題④（パスカルの三角形）

- パスカルの3角形を印字するプログラムを書きなさい。ただし、次のプログラムで印字されるように関数を完成してください。







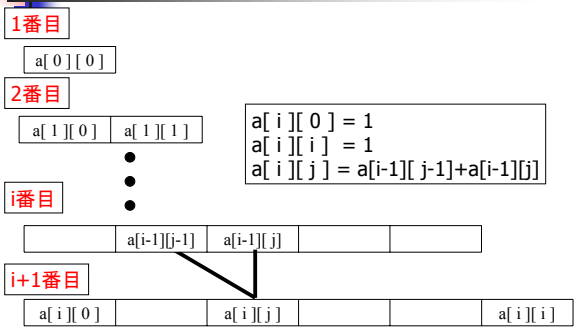

## パスカルの三角形のプログラムの大枠

```
def Pascal( n )
  n = 10 if( n>30 || n<=0 )
  a = Array.new( n )
  a.length.times{ |i|
        a[ i ] = Array.new( n )
  }

  (0..a.length-1).each{ |i|
    (0..a[i].length-1).each{ |j|
      a[ i ][ j ] = 0
    }
  }
```

二次元配列の要素を決めるプログラムを追加（次ページ）

```
(0..a.length-1).each{ |i|
    (0..a.length-1).each{ |j|
        if a[ i ][ j ] != 0 then
                printf( " %3d", a[i][j] )
        else
                print( "    " )
        end
    }
    print( "¥n" )
  }
end

print("Enter n: "); n = gets.chomp.to_i
Pascal( n )
```

## 二次元配列の要素の決め方






## パスカルの三角形

実行結果

```
>ruby report-4.rb
Enter n: 10
  1
  1  1
  1  2  1
  1  3  3  1
  1  4  6  4  1
  1  5 10 10  5  1
  1  6 15 20 15  6  1
  1  7 21 35 35 21  7  1
  1  8 28 56 70 56 28  8  1
  1  9 36 84 126 126 84 36  9  1
```



## 練習問題

- 練習問題①～④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
  - 7/9（木）18時までに提出です
- 来週は講義のまとめの問題（レポート一回分として評価）を行ないます
  - これまでの復習をして来て下さい