Information Science
6: Repetition and Recursion I: Examples

Naonori Kakimura
垣村尚徳
kakimura@global.c.u-tokyo.ac.jp

PEAK
Programs in English at Komaba

Remark: Substitute for isrb

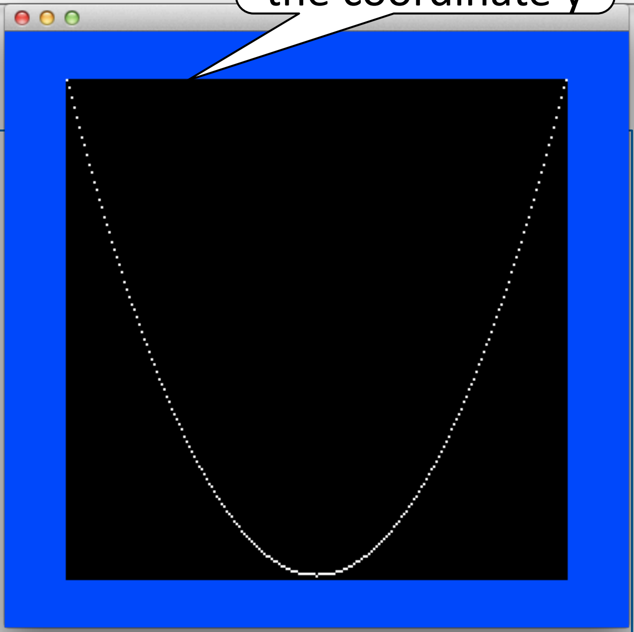
- Project Jupyter
  - https://try.jupyter.org/
- choose "new" -> "Ruby“
  - There, you can use "irb-like" environment
- Function "show" can be downloaded from
  - http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/site/?RubyShowMethod
    - from the website of Prof. Chiba
  - Can be loaded in a similar way to irb

Today's Contents

- Remarks on Animation
  - Quadratic function
  - Circle
- Recursion
  - Concept
    - Ex. Summation
  - Examples
    - Sum of divisors
    - Greatest divisors
- Exercises

Review: Sample Animation


```
def quadratic(s)
    t = 2*s + 1
    image = make2d(t, t)
    u = 1.0*(t - 2)/s**2
    for x in 0 .. (t - 1)
        y = -u*(x - s)**2 + t - 2
        image[y][x] = 1
        show(image)
    end
    image
end
```



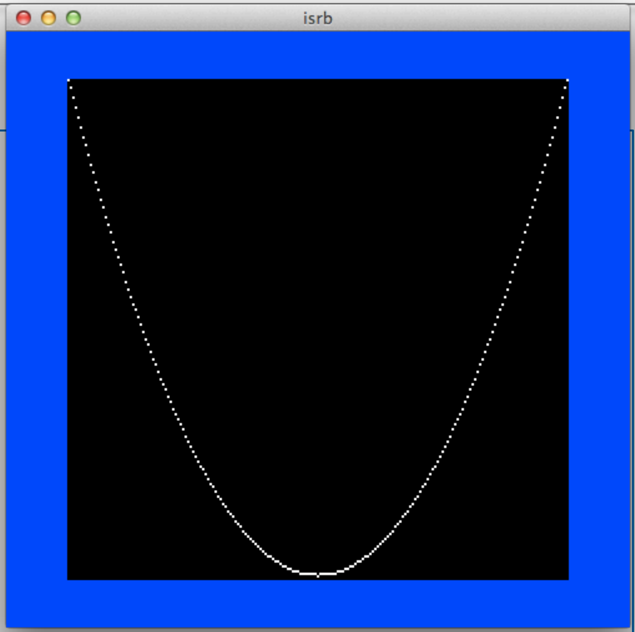
x computes
the coordinate y

Each coordinate x determines
the corresponding y

Redraw the image
each time you add a point

animation.rb

Review: Sample Animation



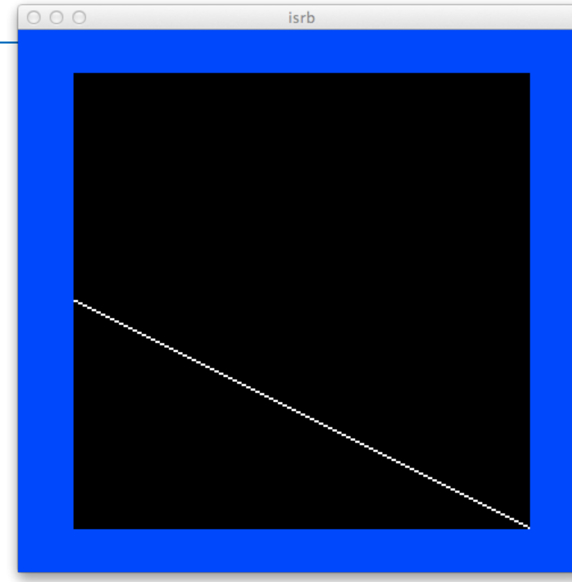

```ruby
def quadratic(s)
    t = 2*s + 1
    image = make2d(t, t)
    u = 1.0*(t - 2)/s**2
    for x in 0 .. (t - 1)
        y = -u*(x - s)**2 + t - 2
        image[y][x] = 1
        show(image)
    end
    image
end
```

You can change the values to observe how it works

animation.rb

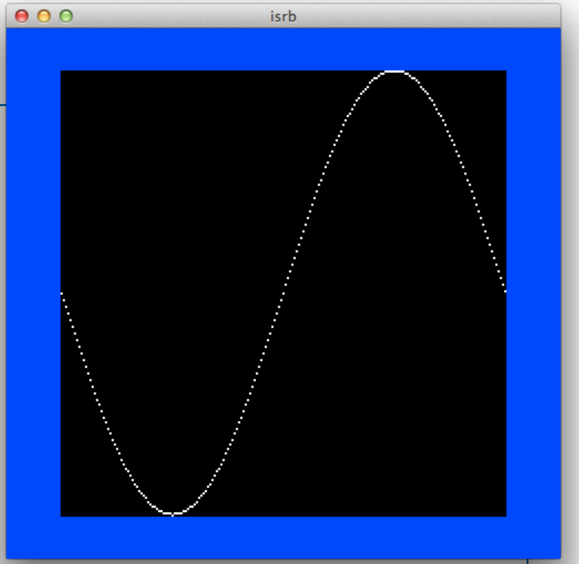
Same Framework: Line


```
def line(s)
  t = 2*s + 1
  image = make2d(t, t)
  for x in 0 .. (t - 1)
    y = 0.5*x + s
    image[y.to_i][x] = 1
    show(image)
  end
  image
end
```



".to_i" makes the value integer

Same Framework: Sin Curve




```
def sincurve(s)
    t = 2*s + 1
    image = make2d(t, t)
    for x in 0 .. (t - 1)
        y = s*sin(x*PI/s) + s
        image[y.to_i][x] = 1
        show(image)
    end
    image
end
```

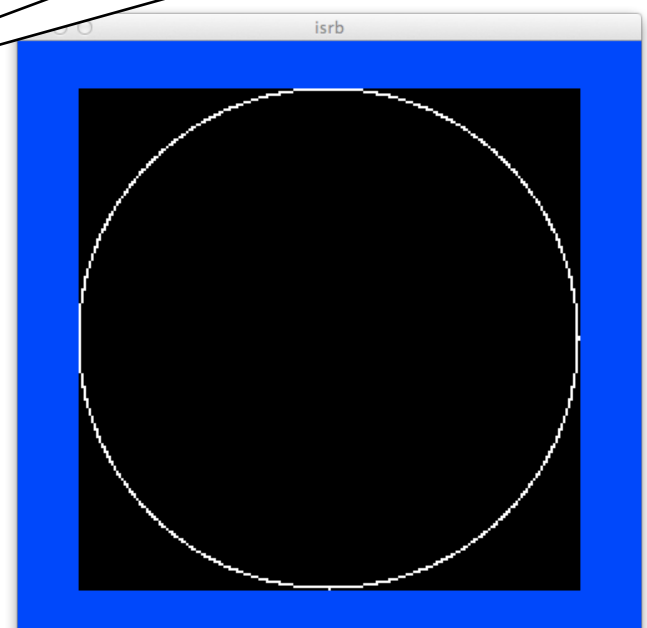
".to_i" makes the value integer

LW's Exercise 2: Animation of Circle

n points on the circle

```
def circle(s, n)
    t = 2*s + 1
    image = make2d(t, t)
    for p in 0 .. (n - 1)
        theta = p*2*PI/n
        y = (Fill in this part)
        x = (Fill in this part)
        image[y.to_i][x.to_i] = 1
        show(image)
    end
    image
end
```

".to_i" makes the value integer

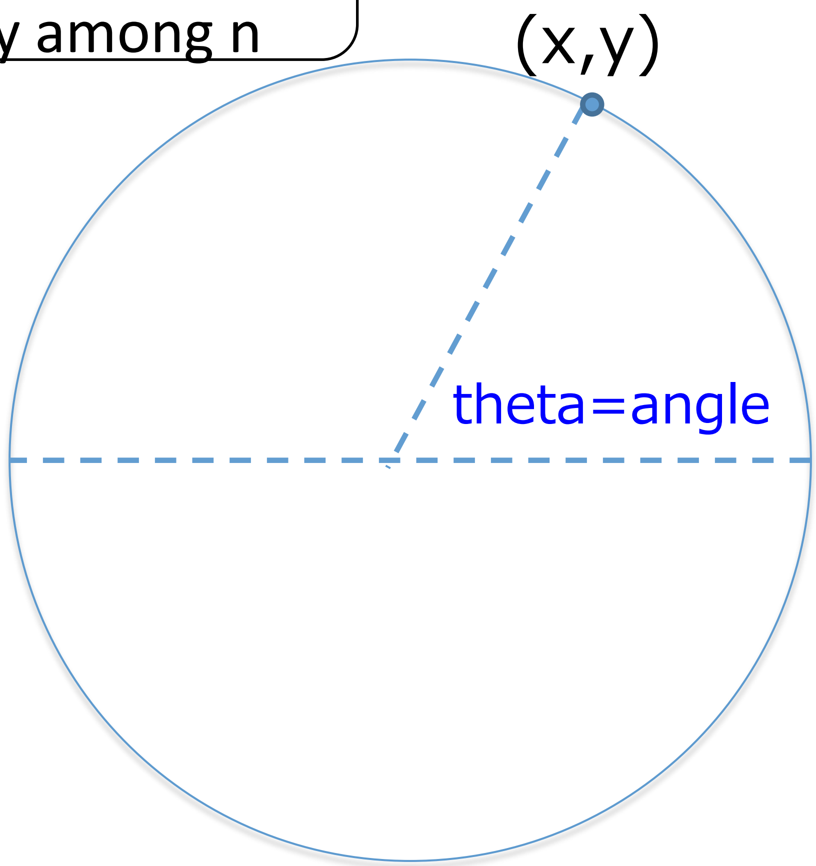


Hint: use sin &cos

LW's Exercise 2: Animation of Circle

Divide the angle(perimeter) equally among n

```
def circle(s, n)
    t = 2*s + 1
    image = make2d(t, t)
    for p in 0 .. (n - 1)
        theta = p*2*PI/n
        y = (Fill in this part)
        x = (Fill in this part)
        image[y.to_i][x.to_i] = 1
        show(image)
    end
    image
end
```



Hint: use sin & cos

You can make completely different program if you want

Today's Contents

- Remarks on Animation
  - Quadratic function
  - Circle

- Recursion
  - Concept
    - Ex. Summation
  - Examples
    - Sum of divisors
    - Greatest divisors
- Exercises

Ex1: Simple Problem

- Compute 1+2+3+⋯+n

  - Ruby cannot understand "⋯"

    irb(main):004:0> 1 + 2 + 3 + ⋯ + n

- Method 1: Repeat adding from 1 to n
- Method 2: Using recursive relation

Ex1: Simple Problem

- Compute $1+2+3+\cdots+n$

- Method 1: Repeat adding from 1 to n

```
s = 0
for i in 1..n
    s = s + i
end
```

Define SUM using Repetition


```
def sum_loop(n)
    s = 0
    for i in 1..n
        s = s + i
    end
    s
end

            sum_loop.rb *
```

```
irb(main):004:0> sum_loop(3)
=> 6
irb(main):005:0> sum_loop(10)
=> 55
irb(main):006:0> 1+2+3+4+5+
irb(main):007:0* 6+7+8+9+10
=> 55
```

Recursive Definition

- $sum(n) = 1+2+\cdots+n$
  - $sum(1)=1$
  - $sum(2)=1+2$
  - $sum(3)=1+2+3$
  - $\cdots$

- This can be viewed as "$sum(n)=sum(n-1)+n$"
  - Another type of definition
    - Called recursion: Definition using definition itself

Another Definition of Summation

- Recursive definition

$$\mathrm{sum}(n) = \begin{cases} \mathrm{sum}(n-1) + n & (n \geq 2) \\ 1 & (n = 1) \end{cases}$$

Don't forget the base case

- Ex
  - sum(3) = sum(2) + 3
    = (sum(1) + 2) + 3
    = (1 + 2) + 3

SUM Using Recursion

$$\mathrm{sum}(n) = \begin{cases} \mathrm{sum}(n-1) + n & (n \geq 2) \\ 1 & (n = 1) \end{cases}$$

```
def sum(n)
    if n >= 2
        sum(n-1) + n
    else
        1
    end
end
                    sum.rb *
```

```
irb(main):005:0> sum(10)
=> 55
irb(main):006:0> 1+2+3+4+5+
irb(main):007:0* 6+7+8+9+10
=> 55
```

How Recursion Works

- Suppose n=3
  - Rem. Program is performed from top to bottom

```
sum(n)              n=3
    if n >= 2
        sum(n-1) + n
    else
        1
    end
end
                return 6
```

Call sum(2)

```
sum(n)              n=2
    if n >= 2
        sum(n-1) + n
    else
        1
    end
end
                return 3
```

Call sum(1)

```
sum(1)   n=1
    if n >= 2
        sum(n-1)+n
    else
        1
    end
end
 return 1
```

Observation: Print the Process

- Inserting “print” to display variables

```
def sum_loop(n)
  s = 0
  print "sum=", s, "¥n"
  for i in 1..n do
     s = s + i;
     print "sum=", s, "¥n"
  end
  s
end
```

```
def sum(n)
  print "Compute sum(",n,")...¥n"
  if n >= 2
    print "sum(", n, ")=sum("
    print  n-1 ,")+", n, "¥n"
    s = sum(n - 1) + n
  else
    s = 1
  end
  print "sum(", n, ")=", s, "¥n"
  s
end
```

Observation: Print the Process


```
def sum_loop(n)
  s = 0
  print "sum=", s, "¥n"
  for i in 1..n do
     s = s + i;
     print "sum=", s, "¥n"
  end
  s
end
```

```
irb(main):073:0> sum_loop(5)
sum=0
sum=1
sum=3
sum=6
sum=10
sum=15
15
```

Print 3 objects: “sum=”, s, and “¥n”
- Object with “” is letters
- Object without “ ” is a variable
- “¥n” mean “return”(line break)

Observation: Printing the Process


```
irb(main):080:0> sum(4)
Compute sum(4)...
sum(4)=sum(3)+4
Compute sum(3)...
sum(3)=sum(2)+3
Compute sum(2)...
sum(2)=sum(1)+2
Compute sum(1)...
sum(1)=1
sum(2)=3
sum(3)=6
sum(4)=10
10
```

```
def sum(n)
  print "Compute sum(",n,")...¥n"
  if n >= 2
    print "sum(", n, ")=sum("
    print  n-1 ,")+", n, "¥n"
    s = sum(n - 1) + n
  else
    s = 1
  end
  print "sum(", n, ")=", s, "¥n"
  s
end
```

Advantages of Recursion

- Recursion
  - Simpler description
    - no "⋯", no loop
  - Harder to understand its behavior
  - Sometimes it takes much more time to compute than using repetition
    - Next week

$$\mathrm{sum}(n) = \begin{cases} \mathrm{sum}(n-1) + n & (n \geq 2) \\ 1 & (n = 1) \end{cases}$$

Today's Contents

- Remarks on Animation
  - Quadratic function
  - Circle
- Recursion
  - Concept
    - Ex. Summation
  - Examples
    - Sum of divisors
    - Greatest divisors
- Exercises

Ex2. Sum of Divisors by Repetition

- sod(k, n):
  - Computing the sum of the divisors of k in 1⋯n
  - sod(10,9)
    - divisors of 10 in 1..9: 1,2,5 → sum=8
  - sod(11,10)
    - divisors of 11 in 1..10: 1 → sum=1
- Purpose: Make a program
  - Using repetition
  - Using recursion

Past Exercises

- Use divisible(x,y) that decides whether x is divisible by y

```ruby
def divisible(x, y)
  x % y == 0
end
```

divisible.rb

```
irb(main):004:0> divisible(6,2)
=> true
irb(main):005:0> divisible(6,3)
=> true
irb(main):006:0> divisible(5,2)
=> false
```

Ex2. Sum of Divisors by Repetition


```
load ("./divisible.rb”)
def sod_loop(k, n)
  s = 0
  for i in 1..n
    if divisible(k, i)
      s = s + i
    end
  end
  s
end
```

Add i only if k is divisible by i

sod_loop.rb

```
irb(main):004:0> sod_loop(10,9)
=> 8
irb(main):005:0> 5+2+1
=> 8
irb(main):006:0> sod_loop(28,27)
=> 28
irb(main):007:0> 14+7+4+2+1
=> 28
irb(main):008:0> sod_loop(29,28)
=> 1
```

Ex2. Recursive definition

- sod(k, n)
  - Compute sod(k, n-1) and
  - Do something depending on divisibility of n

$$\mathrm{sod}(k, n) = \begin{cases} \mathrm{sod}(k, n-1) + n & \text{(n≧2 \& k is divisible by n)} \\ \mathrm{sod}(k, n-1) + 0 & \text{(n≧2 \& k is not divisible by n)} \\ 1 & (n = 1) \end{cases}$$

Ex2. Sum of Divisors by Recursion


```
load ("./divisible.rb”)
def sod(k, n)
    if n >= 2
        if divisible(k, n)
            sod(k, n-1) + n
        else
            sod(k, n-1)
        end
    else
        1
    end
end
```

sod.rb *

```
irb(main):006:0> sod(28,27)
=> 28
irb(main):007:0> 14+7+4+2+1
=> 28
irb(main):008:0> sod(29,28)
=> 1
```

Today’s Contents

- Remarks on Animation
  - Quadratic function
  - Circle
- Recursion
  - Concept
    - Ex. Summation
  - Examples
    - Sum of divisors
    - Greatest divisors
- Exercises

Ex3. The Greatest Divisor of Integers

gd(k,n): Find the maximum number x such that k is divisible by x, and x is at most n

- Example
  - gd(10, 6): Greatest divisor of 10 in 1···6 is 5
  - gd(12, 11): of 12 in 1···11 is 6
  - gd(11, 7): of 11 in 1···7 is 1

Ex3. Define by repetition

gd(k,n): Find the maximum number x such that k is divisible by x, and x is at most n

```
load ("./divisible.rb”)
def gd_loop(k, n)
    while !divisible(k,n)
        n = n - 1
    end
    n
end

                gd_loop.rb *
```

```
def divisible(k, n)
    k % n == 0
end
```

Check k is divisible by n
Check k is divisible by n=n-1
Check k is divisible by n=n-2
…
If yes, return the current value

Ex3. Define by repetition

gd(k,n): Find the maximum number x such that k is divisible by x, and x is at most n

```
load ("./divisible.rb”)
def gd_loop(k, n)
    while !divisible(k,n)
        n = n - 1
    end
    n
end
```

gd_loop.rb *

```
irb(main):004:0> gd_loop(6,5)
=> 3
irb(main):005:0> gd_loop(98,30)
=> 14
irb(main):006:0> gd_loop(98,97)
=> 49
irb(main):007:0> gd_loop(47,46)
=> 1
```

Ex3. Recursive Definition

$$\mathrm{gd}(k,n) = \begin{cases} n & \text{(k is divisible by n)} \\ \mathrm{gd}(k,n-1) & \text{(k is not divisible by n)} \end{cases}$$

Ex3. Define by recursion


```ruby
load ("./divisible.rb")
def gd(k,n)
    if divisible(k, n)
        n
    else
        gd(k, n-1)
    end
end
```

gd.rb *

```
irb(main):005:0> gd(98,30)
=> 14
irb(main):006:0> gd(98,97)
=> 49
irb(main):007:0> gd(47,46)
=> 1
```

Today’s Contents

- Remarks on Animation
  - Quadratic function
  - Circle

- Recursion
  - Concept
    - Ex. Summation
  - Examples
    - Sum of divisors
    - Greatest divisors
- Exercises

Exercise1: Factorial of n

- Consider two ways to compute $n!=1\times\ldots\times n$
  - factorial_loop(n) that computes $n!$ using repetition
  - factorial(n) that computes $n!$ using recursion

Exercise 2:

- We want to compute the sum of multiples of p between 1 and n (p≦n)
  - Make the function sump_loop(p,n) using repetition
  - Make the function sump(p,n) using recursion

Exercise 3: Perfect Numbers

- Perfect Number:
  - Number n satisfying sod(n,n-1)=n
    - 6 = 1 + 2 + 3
    - 28 = 1 + 2 + 4 + 7 + 14
    - …

- Make the function that finds the biggest perfect number less than n, using the function sod
  - Then find the biggest perfect number less than 1000

- (If you have time) make two programs
  - Using loop
  - Using recursion

(Optional) If you have time

- See the Exercise PDF file and solve
  - Exercise 5-9(a)
  - Exercise 5-3(d),4(d)
  - And any other problems

- Past exam 2015, 1

Next Week

- More Recursion
  - Combination number
  - Sierpinski triangle
  - Tower of Hanoi

- Deadline of Today’s Exercises
  - **Nov 16, 23:59**