Information Science
3:Conditions, Repetitions, Arrays

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

PEAK
Programs in English at Komaba

Class Cancelation

- **Class on 26 Dec. will be cancelled**

- Planning to have a Q&A session on
  - **13 Jan, Friday, P2**
    - No lectures
    - No need to attend if you have no question

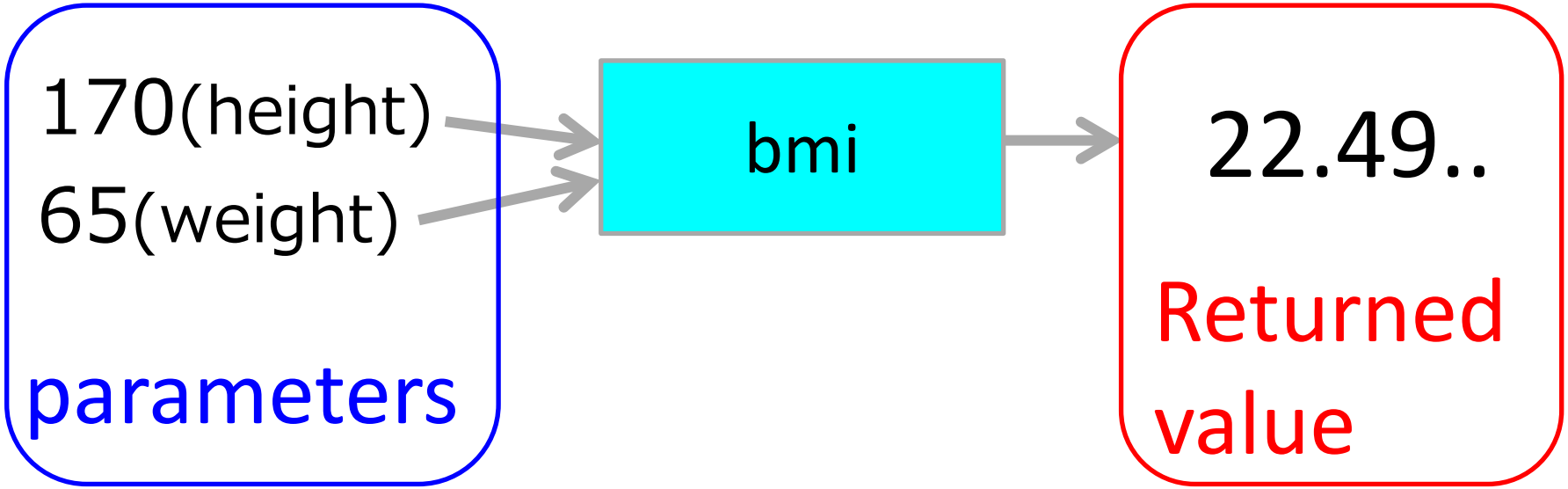
Today's contents

- Review of last week
  - Defining functions
  - Local variables

- Today and Next week
  - Fundamental components in programs
    - IF, FOR, WHILE
    - Arrays (next week)

Functions

- Give a name to some action, and do not see inside



Ex. body-mass index

- a measure for human body shape
- Body mass[kg]÷ (height[m]* height[m])
  - ratio btw weight and height: implying fat or slim

Review: Two Ways of Defining Functions

1. Typing in directly in irb

- Easy to define, but necessary to write every time

```
irb(main):003:0> def bmi(height , weight)
irb(main):004:1>     weight / (height/100.0) ** 2
irb(main):005:1> end
```

```
def  [Func Name](parameters)
...     the details
end
```

Review: Two Ways of Defining Functions

2. Loading from a file

recommended

- prepare another file containing functions

```
# BMI of a person with height (cm) and weight (kg)
def bmi(height , weight )
  weight / ( height /100.0) ** 2
end
```

Better to have a blank for visibility

bmi.rb

```
irb(main):001:0> load("./bmi.rb”)
```

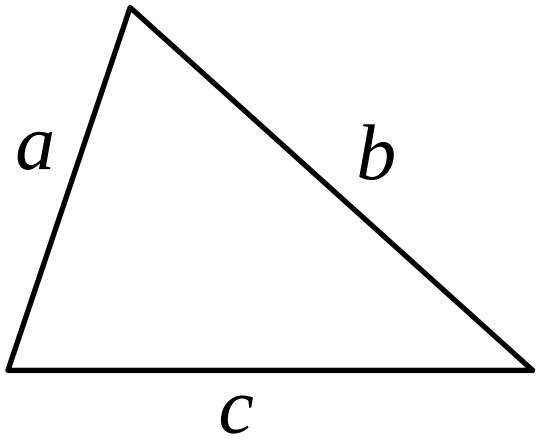
* Need to move to the directory having the file

Local Variable: Variables in a Function

Ex. A function that computes the area of a triangle whose sides have lengths a, b, and c.

- Use Heron's formula

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$



$$s = \frac{a+b+c}{2}$$

Cf) Why does the formula hold?

Useful to use the variable s to express computation

To increase readability of the expression

Local Variable

Ex. A function that computes the area of a triangle whose sides have lengths a, b, and c.

Parameters a,b,c are also local variables.
→ We cannot invoke a,b,c outside

```
def heron(a, b, c)
    s = 0.5*(a+b+c)
   sqrt(s * (s-a) * (s-b) * (s-c))
end
```

s is called a local variable

heron.rb

A local variable is valid only inside the function

Place ”include(Math)” outside of def…end

Note: How to Change Directory in irb

- When using irb
  - pwd = Dir.pwd()
  - cd = Dir.chdir()
    - Dir.chdir(“Desktop")
    - Dir.chdir("Desktop/allcode")
  - ls = Dir.entries(".")

```
irb(main):084:0> Dir.pwd
"/home01/0000000000“
```

Today's contents

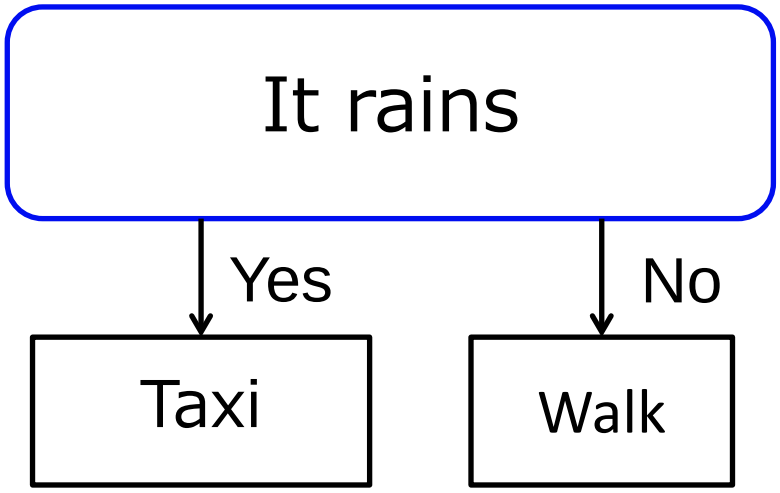
- Review of last week
  - Defining functions
  - Local variables

- Today and Next week
  - Fundamental components in programs
    - IF
    - Operators
    - FOR, WHILE
    - Arrays (next week)

Conditional Processing

- Branching procedure according to a condition

```
if “condition”
    ”do something when the cond is satisfied"
else
   “do something when the cond is not satisfied"
end
```




```
if “it rains”
    “take a taxi"
else
    “walk”
end
```

Conditional Processing in Ruby Programs

Function max(x,y)
  returning the max of x and y

```
def max(x,y)
    if y < x
        x
    else
        y
    end
end
```

max.rb

```
irb(main):003:0> load("./ max.rb")
=> true
irb(main):004:0> max(123, 456)
=> 456
irb(main):005:0> max(max(12, 34), max(56, 78))
=> 78
```

Branching into Three Cases: Program 1

Function sgn(x), that returns the sign of a given num x

```
def sgn(x)
    if x < 0
        -1
    else
        if 0 < x
            1          # not(x<0) and 0<x
        else
            0          # not(x<0) and not(0<x)
        end
    end
end
```

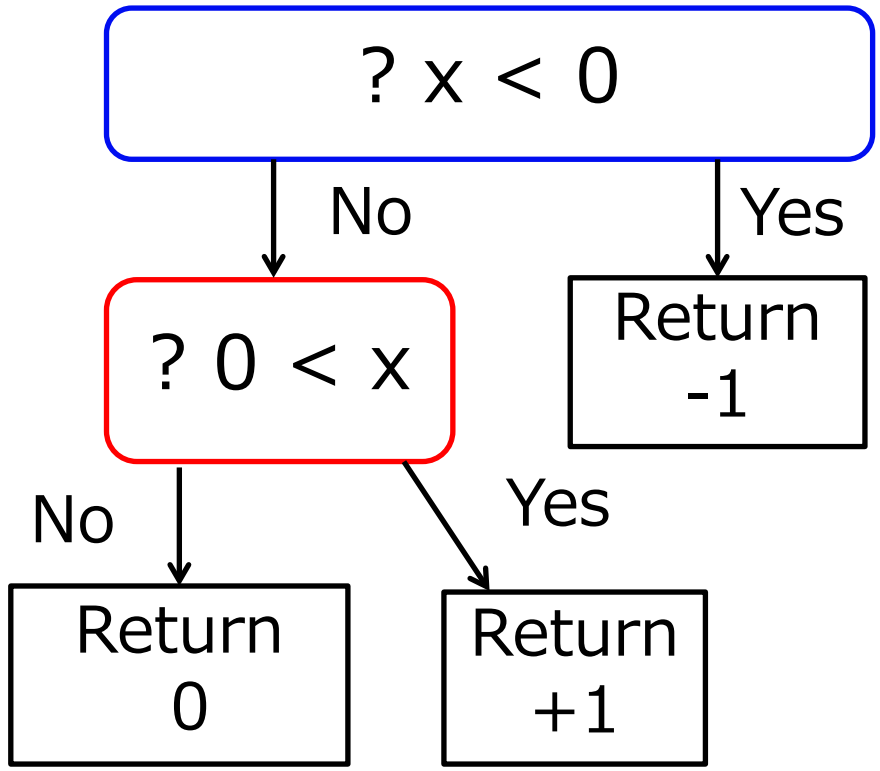
i.e., x is nonnegative

sign.rb

Branching into Three Cases: Program 1

Function sgn(x), that returns the sign of a given num x

```
def sgn(x)
    if x < 0
        -1
    else
        if 0 < x
            1
        else
            0
        end
    end
end
```



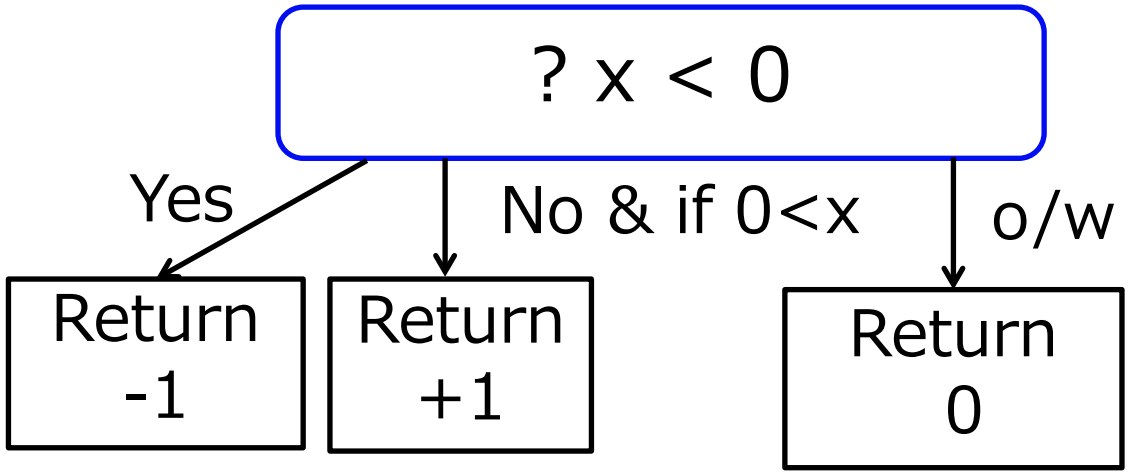
sign.rb

Branching into Three Cases: Program 2

Function sgn(x), that returns the sign of a given num x

```
def sgn(x)
    if x < 0
        -1
    elsif 0 < x
        1
    else
        0
    end
end
```

We can use "elsif",
which is the same as "else if"
(Only one "end" is necessary)



Cf) Bad Programs: Try to Execute It


```
def max_error(x, y)
  if x > y
    x
  end
  if y >= x
    y
  end
end
```

(>= means ≧)

max_error(1,2) returns 2 correctly,
but max_error(2,1) returns nil

Rem. The returned value of a function is the value computed at the end

- In max_error(2,1), the final computation is

(y(=1) >= x(=2)) at the 2nd IF condition, which returns “nil”

Cf) Bad Programs: Try to Execute It


```
def max_error2(x, y)
  if x > y
    return x
  end
  if y >= x
    return y
  end
end
```

Add “return”
to specify the returned value

Rem. The returned value of a function is the value computed at the end

Today's contents

- Review of last week
  - Defining functions
  - Local variables

- Today and Next week
  - Fundamental components in programs
    - IF
    - Operators to compare two values
    - FOR, WHILE
    - Arrays (next week)

Comparisons Returns True/False

- Operators to compare two values
  - It returns TRUE/FALSE

notation	Math	meaning
x > y	$>$	x is greater than y
x >= y	$\geq$	x is equal to or greater than y
x == y	$=$	x is equal to y (NOT x=y)
x < y	$<$	x is smaller than y
x <= y	$\leq$	x is equal to or smaller than y
x != y	$\neq$	x is not equal to y

operators different from standard math expression

Testing equality is "==."
"=" means "assignment"
Ex. a=8

Logical Operation Giving True/False


```
irb(main):005:0> x = 3
=> 3
irb(main):003:0> x == 2
=> false
irb(main):004:0> 1 < x
=> true
```

Assigning a value (not returning true/false)

Decide if x is equal to 2

Decide if x is greater than 1

"=" and "==" are different

Example: Testing Evenness


```
def is_even(x)
  x%2 == 0
end
```

- The equation returns False or True

```
irb(main):004:0> is_even(20)
=> true
irb(main):005:0> is_even(11)
=> false
```

Combination of Conditions

notation	meaning
`x > y || x == 0`	x > y or x == 0
`x < y && y < z`	x < y and y < z
`!(x < y && y < z)`	NOT (x < y and y < z)

Example of &&

- Returns x if x is btwn -1～1, and 0 otherwise

```
def btwn(x)
   if -1 <= x && x <= 1
     x
   else
     0
   end
end
```

x is between -1 and 1

Exercise: Which is True?

Supposing that x is 7, y is 5, and z is 3

1. x < y
2. z == y
3. z <= x

```
irb(main):003:0> x=7
irb(main):003:0> y=5
irb(main):003:0> z=3
irb(main):003:0> x<y
=> false
```

Exercise: Which is True?

Supposing that x is 7, y is 5, and z is 3

1. x < y => false
2. z == y => false
3. z <= x => true

```
irb(main):003:0> x=7
irb(main):003:0> y=5
irb(main):003:0> z=3
irb(main):003:0> x<y
=> false
```

Exercise: Which is True? (On ITC-LMS)

Supposing that x is 7, y is 5, and z is 3

1. x < y
2. y != z
3. z > x
4. z == x

Expect the output before using Ruby

Exercise: Which is True? (On ITC-LMS)

Supposing that x is 7, y is 5, and z is 3

1. x > y && z == x
2. x > y || z == x
3. !(x > y)

(continued)

Expect the output before using Ruby

1. x(=7) > y(=5) and z(=3)==x(=7) => false

true false

Exercise: Which is True? (On ITC-LMS)

Supposing that x is 7, y is 5, and z is 3

4. x < y && y != z
5. x <= y || y == z
6. !(x < y && y == z)

Today's contents

- Review of last week
  - Defining functions
  - Local variables
- Today and Next week
  - Fundamental components in programs
    - IF
    - Operators
    - FOR, WHILE
    - Arrays (next week)
- Exercise

Repetitive Processing: WHILE

- “While ～, repeat the following”
  - While the condition “～” holds, do the commands between “while” and “end”

```
while CONDITION
    “COMMANDS"
end
```

Easy to understand if we put an indent(space) at the head of line

```
x = 5
while x >= 1
   print x
   print "hello!¥n"
   x = x - 1
end
```

OUTPUT:

```
5 hello!
4 hello!
3 hello!
2 hello!
1 hello!
```

¥n means line break

Repetitive Processing: FOR

- For each i in some range, do the following

Ex. Repeat 100 times

```
for i in 1..100
    “COMMANDS"
end
```

i is a variable
(OK to have another name)
i changes from 1 to 100
during the repetition

```
for j in 1..5
   print j
   print "hello!¥n"
end
```

OUTPUT:
1 Hello!
2 Hello!
3 Hello!
4 Hello!
5 Hello!

Today's contents

- Review of last week
  - Defining functions
  - Local variables
- Today and Next week
  - Fundamental components in programs
    - IF
    - Operators
    - FOR, WHILE
    - Arrays (next week)
- Exercise

How to submit exercises

- Exercises specified in the subsequent slides
  - Submit a text file(.txt) containing all of your Ruby codes
    - Through ITC-LMS (Assignments)
  - A sample text file for attachment can be found in LMS
  - You can write a question or comments if you have

```
Date:
Student ID:
Name:
======================
Ex)
2-1 (a)

def log_3(n)
    ....
end
======================
2-1 (b)
```

Separate each code with =, -, % etc

Copy&Paste your code (not results)

Exercises for Conditions

- Choose true statements in slides 24-26
- Solve at least 5 problems out of the following.
  - You can solve all or other problems in addition
- **3-1** Define a function *abs(x)* that computes the absolute value of a value x.
- **3-3(a)** Define a function *divisible(x,y)* that decides if x is divisible by y.
- **3-3(c)** Check a leap year
- **3-5(a)(b)** Compute a quadratic function
- **3-5(c)** Compute the median of the 3 numbers
- **3-6** logic functions
- **3-7** Triangle detection
- **3-8** random number
- **3-10** using && and ||

Recap. Exercises from Last Week

- Make a file heron.rb, and confirm it works
  - Try heron(3,4,5), heron(3,4,7), heron(3,4,8), etc.
    - consider why we have sometimes an error
  - Check Problem 2-8 (Problem 8 in Section 2)

- **Solve for submission**
  - 2-2 logarithm
  - 2-4 computing the area of an equilateral triangle

  - If you have time
    - 2-3 (a)(b) and 2-7

Remarks on Distributed Exercises

- Some are required to submit in exercise session
  - I will specify some of them during the session
- For the other exercises, if you submit (a copy of) your file, then I will check it.
  - confirm it works well before submitting
  - It may be included in the final evaluation
  - But it is optional
  - NO sample answers

Remarks on Previous Exercises

- Before submission,
  - check whether your program works correctly
    - By executing the program with specified parameters
      - Need careful writing
  - Common mistakes
    - Ex. 1/2, that would return 0
    - Ex. 2x, that would return error
  - Consider why it works or why it does not
    - Though you can discuss with your friends

Deadline of Submission

- **10/26 (Wed) 23:59**

- Next session: 10/24
  - Continuation of today's topic: Array