Ruby Notes

1 First program

A Ruby program has the form of

```
  print "Hello, Yamaguchi\n"
```

If we "execute" the above program, Ruby executes `print "Hello, Yamaguchi\n"`, and displays "Hello, Yamaguchi". The symbol ¥n (or \n, the same meaning) means the line break. Something between "'s represents a string of letters. If you do not put ", Ruby do not understand it is a string. For example, compare the following two commands:

```
  print "1+2"
  print 1+2
```

You can write many sentences after "print".

```
  print "This ", "is ", "a pen", "."
```

To use a mathematical function, write

```
include(Math)
```

Then we can do, for example,

```
sqrt(3)
log(100)
```

2 Variables

A variable is a kind of container having a value. Assigning a value to a variable can be done as follows.

```
  i = 2
```

When you write i after this assignment, the variable i means 2.

```
  print i*i
```

We can change the value by re-assigning.

```
  i = 3
  print i*i
```

Note that "=" means "assign the right-hand side to the left variable," and not "equal" in mathematics. So $i = 3$ and $3 = i$ are different. For example, if we type

```
a = 1
b = 2
b = a
print "a is ", a, " and b is ", b, "\n"
b = 2
a = b
print "a is ", a, " and b is ", b, "\n"
```

then we can observe the behavior.

It is convenient to write

```
i += 1
# same as i = i + 1
```

to increment i by one. Similarly, we can write `i -= 1`, `i /= 2`, and so on.

`print` is useful when your program does not work well. We can display the value of a variable during the process by inserting `print`.

```
i = 4
print "The variable i is ", i, "\n"
```

3 Function

We can group a series of process, and invoke the group one time.

```
def half(i)
  i/2
end
```

Variables in the brackets () are parameters, aka arguments. If you write `half(5)`, then Ruby executes the above function when i=5, which returns 2. We can also use it in such a way as `n = half(5)`. Then the returned value is assigned to `n`.

```
def repeat(n)
 for i in 1..n
  print "*"
 end
end
```

If you write `repeat(10)`, then Ruby executes the above function when n=10, and print "*" 10 times.

Remarks on the returned value. Usually, the result of the line which is run at the end is the output of a function in Ruby. For example, the last line of `half(i)` is `i/2`, which means that it returns `i/2`. On the other hands, the last line of `repeat(n)` is `end` of the `for` loop, which means that it returns `1..n`.

3.1 How to load a function from a file

```
load("FILE NAME")
```

For example, write

```
load("./bmi.rb")
```

Note that "./" means the current directory. We need to specify where the file is.

4 Array

To define an array, we write

```
a = [1,2,3]
```

It means that a[0] is 1, a[1] is 2, and a[2] is 3. Remark that the index begins from 0. To change a value of some entry, write

```
a[1] = 5
```

The length of array `a` is obtained by

```
a.length() # same as a.length
```

To define an array with size `n`, we write

```
a = Array.new(n)
```

A high-dimensional array is defined by

```
a = [[1,2,3],[0,0,0],[3,4,5]]
```

We can refer to the entries such as

```
a[1][2]
```

If we do

```
show(a)
```

in `isrb`, the array `a` is displayed in another window so that the value of each entry represents the brightness.

5 Strings

Below is an operation to concatenate two strings

```
"1"+"2"
```

The result should be 12. Note that Ruby distinguishes a character and a number. For example, when you execute

```
n=1
print "The variable n is " + n
```

then some error happens, because the variable `n` is regarded as a number. A solution is to write

```
n=1
print "The variable n is " + n.to_s
```

where ".to_s" is a function to convert a number to a string. Or, we can write

```
n="1"
print "The variable n is " + n
```

where now `n` is a string because it is enclosed by "".

The converse is ".to_i"(to integer) or ".to_f"(to real number).

```
ns="111"
print ns.to_i-100, "\n"
```

6 Control Structures

6.1 IF

We can branch the process based on a condition.

```
if CONDITION (returning TRUE/FALSE)
   COMMANDS when the CONDITION holds.
end
```

We can also provide some commands when the condition is not true.

```
if CONDITION (returning TRUE/FALSE)
   COMMANDS when the CONDITION holds.
else
   COMMANDS when the CONDITION does not hold.
end
```

Example. Display something if `year` is divisible by 4. A condition after `if` has to return TRUE/FALSE.

```
year = 2020
if year % 4 == 0
 print "leap year\n"
end
```

We can “nest” branching.

```
if CONDITION (returning TRUE/FALSE)
   COMMANDS when the CONDITION holds.
elsif ANOTHER_CONDITION
   CAMMANDS when the ANOTHER_CONDITION holds
            (and CONDITION does not hold).
elsif YET_ANOTHER_CONDITION
   CAMMANDS when the YET_ANOTHER_CONDITION holds
            (and the first two conditions do not hold).
else
   CAMMANDS when none of the above conditions holds.
end
```

Example of nesting.

```
if i > 0
 print "plus\n"
elsif i == 0
 print "zero\n"
else
 print "minus\n"
end
```

6.1.1 Operators

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

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

6.2 FOR

Repeat the same commands specified number of times. We have one variable changing in each repetition, which can be used to compute.

```
for VARIABLE in START..END
  #VARIABLE changes from START to END one-by-one.
  COMMANDS for VARIABLE=START, START+1,..., END
end
```

Examples:

```
for i in 0..4
  print i," ",i*i,"\n"
end
```

Execute printing i and i*i for i = 0,1,2,3,4.

```
for i in 1..9
 for j in 1..9
  print i*j," "
 end
 print "\n"
end
```

(i,j) changes as (1,1),(1,2),...,(1,9),(2,1),....

6.3 WHILE

Repeat until a specified condition does not hold.

```
while CONDITION
  COMMANDS to do while the CONDITION holds.
end
```

Examples

```
i = 13
while i != 1
 if i%2 == 0
  i /= 2    # meaning that i = i/2
 else
  i = 3*i + 1
 end
 print i,"\n"
end
```