第二章　基本資料型態

OOP with Ruby

本章內容

- Ruby 的標準程式結構
- 數字型態資料
- 字串型態資料
- 範圍型態資料
- 日期時間型態資料
- 正規表示式
- 進階資料型態

Ruby 的標準程式結構

- There is no "main"
- Functions are defined by using "def"
- Parenthesis not necessarily exists
- What else?

數字型態資料

- Ruby supports integer and floating-point numbers
- Integer can be any length (determined by your free memory)
- Integer class: Fixnum
  - -2^{30}~2^{30}-1 (32-bit environment) or -2^{62}~2^{62}-1 (64-bit environment)
  - Held internally in binary form
- Integer class: Bignum
  - Implemented as variable-length set of short integers
- Ruby automatically convert between two integer classes if needed.

一個簡單測試

```
num = 81
6.times do
  puts "#{num.class}: #{num}"
  num *= num
end
```

- Output

```
Fixnum: 81
Fixnum: 6561
Fixnum: 43046721
Bignum: 1853020188851841
Bignum: 3433683820292512484657849089281
Bignum: 1179018457773858317152087286141251866567821159227584110909696961
```

OOP with Ruby

數字的不同進位表示法

- 123456 => Fixnum 123456
- 0d123456 => Fixnum 123456
- 123_456 => Fixnum 123456 (underscore ignored)
- -543 => Fixnum -543
- 0xaabb => Fixnum 43707 (hexadecimal)
- 0377 => Fixnum 255 (octal start by leading “0”)
- -0b10_1010 => Fixnum -42 (negative binary with underscore ignored)
- 123_456_789_123_456_789 => Bignum 123456789123456789

整數迭代 (Iterator)

- Integer support various iterators
  - 3.times
    - for (int i=0; i<3; i++)
  - 1.upto(5)
    - for (int i=1; i<=5; i++)
  - 99.downto(95)
    - for (int i=99; i>=95; i++)
  - 50.step(80, 5)
    - for (int i=50; i<=80; i+=5)

自動轉換型態問題

- String with only digits will not automatically convert into numbers
  - “34” will not automatically convert to integer 34
  - Integer(“34”) will convert to integer 34
- Number is able to convert into strings by “to_s” method
  - 34.to_s

OOP with Ruby

浮點數型態

- Floating-point class: Float
  - Mapped to native double data type
  - Notation: [integer part].0e[exponent part]
    - E.g. 1.0e5 => 1×10^5
    - Wrong notation: 1.e5, which will invoke “e5” method of Fixnum

OOP with Ruby

浮點數的四捨五入

- Round to integer
  - 3.14159.round => 3
  - -47.7.round => -48
- Round to certain digit
  - eval(sprintf(“%8.3f”, 3.1415926)) => 3.142
  - eval(sprintf(“%8.2f”, 3.1415926)) => 3.14

OOP with Ruby

數字的格式化輸出

- Use printf method like C

```
x = 345.6789
i = 123
printf("x = %6.2f\n", x)  # x = 345.68
printf("x = %9.2e\n", x) # x = 3.457e+002
printf("i = %5d\n", i) # i = _ _123
printf("i = %05d\n", i) # i = 00123
printf("i = %-5d\n", i) # i = 123
```

OOP with Ruby

數字的格式化輸出 (2)

- Store formatted number in a string
  - Use sprintf method like C

```
str = sprintf("%5.1f",x)        # str will be "345.7"
```

  - Use % method: 'format % value'

```
str = "%5.1" % x                 # str will be "345.7"
str = "%6.2, %05d" % [x,i]   # str will be "345.68, 00123"
```

數值交換

- Traditional “swap” problem will cause the use 3rd storage
- In Ruby, swapping two number can be written as:

```
x, y = y, x
```

字串型態資料

- Ruby strings are simply sequences of 8-bit bytes
- Strings are objects of class String
- String can be quoted by a pair of single quote sign or double quote sign

  ‘this is a test’

  “that is a joke”

- Escape sequence: start by a backslash sign “\”
  - Single quote string can represent only \’ and \\ escape sequence
  - Double quote string allows you put more escape sequence, even embed variable evaluation or a piece of program code

字串型態資料　(2)

- Escape sequence examples

  \" – double quote

  \\ – backslash

  \a – beep sound

  \b – delete key

  \r – carriage return

  \n – new line

  \s – space character

  \t – Tab character

OOP with Ruby

字串測試

```
puts "Hello\t\tworld"
puts "Hello\b\b\b\b\bGoodbye world"
puts "Hello\rStart over world"
puts "1. Hello\n2. World“
```

- Output

```
Hello                world
Goodbye world
Start over world
1. Hello
2. World
```

字串的計算

```
puts "Seconds/day: #{24*60*60}"
puts "#{'Ho! '*3}Merry Christmas!"
puts "This is line #$.“
```

- Output

```
Seconds/day: 86400
Ho! Ho! Ho! Merry Christmas!
This is line 3
```

OOP with Ruby

子字串操作

- String can be operated as a zero-based index character array

```
str = "Humpty Dumpty"
sub1 = str[7,4]
sub2 = str[7,99]
sub3 = str[10,-4]
```

- Output

```
“Dump”
“Dumpty” (overrunning is OK)
nil (empty thing, length is negative)
```

OOP with Ruby

子字串操作　(2)

```
str1 = "Alice"
sub1 = str1[-3,3]
str2 = "Through the Looking-Glass"
sub3 = str2[-13,4]
```

- Output

```
“ice”
“Look”
```

  - A negative index counts backward from the end of the string

OOP with Ruby

範圍型態資料

- Ruby uses ranges to implement three features: sequences, conditions, and intervals
- Ranges as sequences
  - “..” and “…” operators
  - “..” creates an inclusive range
  - “…” creates a range that excludes the specified high value
  - 1..10 means 1,2,3,4,5,6,7,8,9,10
  - ‘a’..’d’ means ‘a’,’b’,’c’,’d’
  - 4…7 means 4,5,6

OOP with Ruby

範圍型態資料 (2)

- Ranges as intervals
  - (1..10) === 5 # range test, true
  - (1..10) === 15 # false
  - (‘a’..’j’) === ‘c’ # true
  - (‘a’..’j’) === ‘z’ # false

OOP with Ruby

日期時間型態資料

- Greenwich Mean Time (GMT) is an old term not really in official use anymore
- New global standard is Coordinated Universal Time (or UTC)
- GMT and UTC are virtually the same thing
- In Ruby, the Time class is used for most operations
  - A time is typically stored internally as a number of seconds from a specific point in time (called the *epoch*), which was midnight January 1, 1970 GMT.

OOP with Ruby

時間操作

- Current time

```
Time.new
Time.now
```

- Working with Specific Times (Post-epoch): the mktime method will create a new Time object based on the parameters passed to it

```
t1 = Time.mktime(2001) # January 1, 2001 at 0:00:00
t2 = Time.mktime(2001,3)
t3 = Time.mktime(2001,3,15)
t4 = Time.mktime(2001,3,15,21)
t5 = Time.mktime(2001,3,15,21,30)
t6 = Time.mktime(2001,3,15,21,30,15) # March 15, 2001 9:30:15 pm
```

時間操作 (2)

- mktime assumes the local time zone
  - Time.local is a synonym for Time.mktime
- Time.gm method assumes GMT (or UTC)

```
t8 = Time.gm(2001,3,15,21,30,15) # March 15, 2001 9:30:15 pm UTC
```

  - Time.utc is a synonym for Time.gm

時間操作 (3)

- All above methods can take an alternative set of parameters.
  - The instance method to_a (which converts a time to an array of relevant values) returns a set of values in this order: seconds, minutes, hours, day, month, year, day of week (0..6), day of year (1..366), daylight saving (true or false), and time zone (as a string).

```
t0 = Time.local(0,15,3,20,11,1979,2,324,false,"GMT-8:00")
```

時間操作 (4)

- Determining Day of the Week

```
time = Time.now
day = time.to_a[6]    # 2 (meaning Tuesday)
```

  - Equals to

```
day = time.wday       # 2 (meaning Tuesday)
```

- Converting to and from the Epoch

```
epoch = Time.at(0)   # Find the epoch (1 Jan 1970 GMT)
newmil = Time.at(978307200)   # Happy New Millennium! (1 Jan 2001)
now = Time.now       # 16 Nov 2000 17:24:28
sec = now.to_i       # 974424268
```

時區

- Obtaining the Time Zone
  - The accessor zone in the Time class will return a String representation of the time zone name

```
z1 = Time.gm(2000,11,10,22,5,0).zone # "GMT-6:00"

z2 = Time.local(2000,11,10,22,5,0).zone # "GMT-6:00"
```

OOP with Ruby

正規表示式

- Regular expressions are used to match patterns against strings
- Regular expressions are objects of type Regexp
  - They can be created by calling the constructor explicitly or by using the literal forms /pattern/ and %r{pattern}.

```
a = Regexp.new('^\s*[a-z]')    # /^\s*[a-z]/
b = /^\s*[a-z]/        # /^\s*[a-z]/
c = %r{^\s*[a-z]}       # /^\s*[a-z]/
```

字串比對

- you can match it against a string using
  - Regexp#match(string)
  - match operators =~ (positive match)
    - The match operators return the character position at which the match occurred
  - match operators !~ (negative match)

```
name = "Fats Waller"
name =~ /a/       #1
name =~ /z/       # nil
/a/ =~ name       #1
```

比對樣式

- The match operators return the character position at which the match occurred.
  - $& receives the part of the string that was matched by the pattern
  - $` receives the part of the string that preceded the match
  - $' receives the string after the match
- Every regular expression contains a pattern, which is used to match the regular expression against a string
  - Within a pattern, all characters except ., |, (,), [,], {, }, +, \, ^, $, *, and ? match themselves.

比對樣式 (2)

```
def show_regexp(a, re)
  if a =~ re
    "#{$`}<<#{$&}>>#{$'}"
  else
    "no match"
  end
end

show_regexp('very interesting', /t/)  # very in<<t>>eresting
show_regexp('Fats Waller', /a/)       # F<<a>>ts Waller
show_regexp('Fats Waller', /ll/)      # Fats Wa<<ll>>er
show_regexp('Fats Waller', /z/)       # no match
```

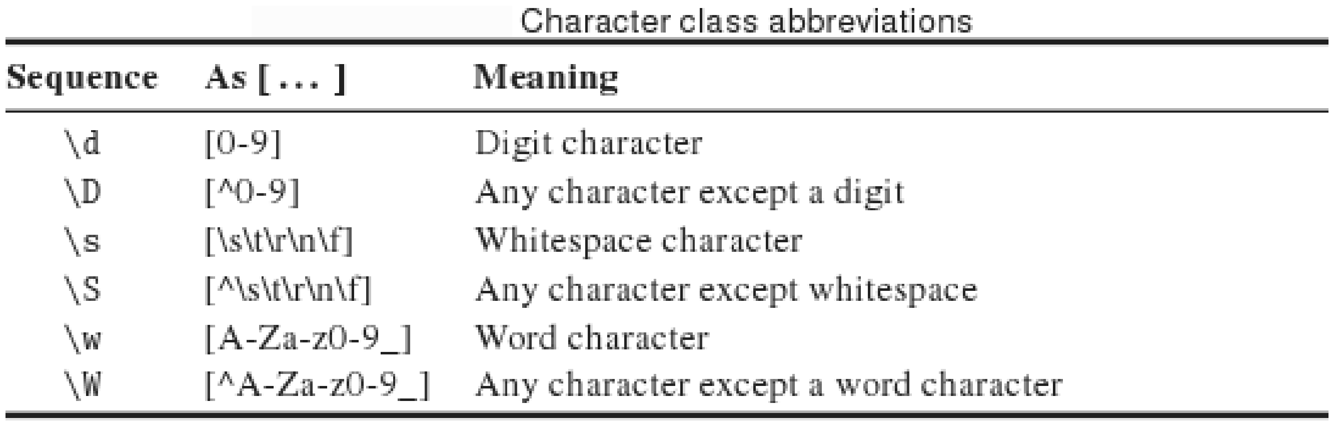
樣式替換

- The methods String#sub and String#gsub look for a good portion of string matching the first argument and replace it with the second argument.
  - String#sub replace once
  - String#gsub replace every occurrence
  - Both String#sub and String#gsub return a new copy of the String containing the substitutions
  - Mutator versions of String#sub! And String#gsub! modify the original string

OOP with Ruby

樣式替換　(2)

```
a = "the quick brown fox"
a.sub(/[aeiou]/, '*')     -> "th* quick brown fox"
a.gsub(/[aeiou]/, '*')   -> "th* q**ck br*wn f*x"
a.sub(/\s\S+/, '')        -> "the brown fox"
a.gsub(/\s\S+/, '')      -> "the"
```

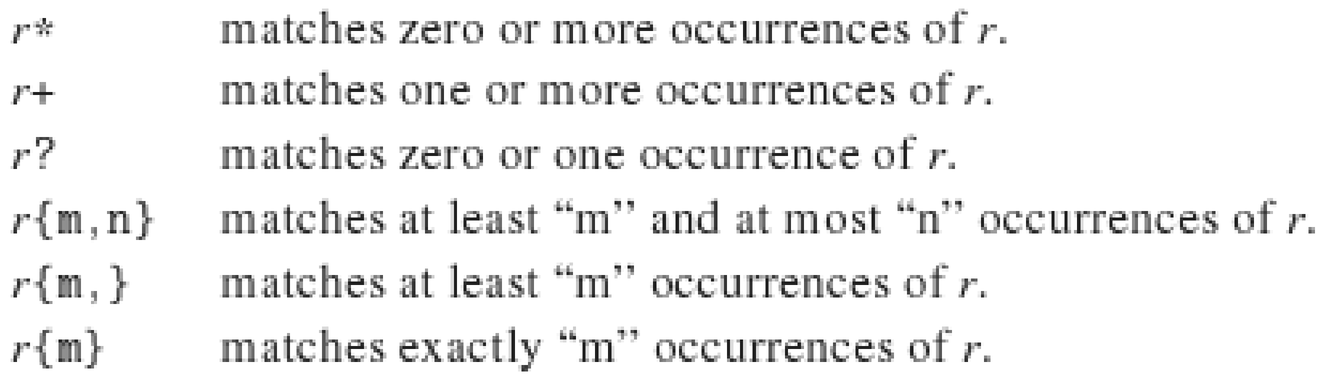


Character class abbreviations

Sequence	As [...]	Meaning
\d	[0-9]	Digit character
\D	[^0-9]	Any character except a digit
\s	[\s\t\r\n\f]	Whitespace character
\S	[^\s\t\r\n\f]	Any character except whitespace
\w	[A-Za-z0-9_]	Word character
\W	[^A-Za-z0-9_]	Any character except a word character

樣式替換 (3)

- Pattern repetition



*r**	matches zero or more occurrences of *r*.
r+	matches one or more occurrences of *r*.
r?	matches zero or one occurrence of *r*.
r{m,n}	matches at least “m” and at most “n” occurrences of *r*.
r{m,}	matches at least “m” occurrences of *r*.
r{m}	matches exactly “m” occurrences of *r*.

進階資料型態

- Stacks and queues
- Linked lists
- Trees
- Graphs

本章回顧

- Ruby classes for this chapter
  - Fixnum, Bignum
  - Integer
  - Float
  - String
  - Time
  - Regexp