Introduction to Julia Programming

Basic programming

by Yueh-Hua Tu

What does Julia looks like?

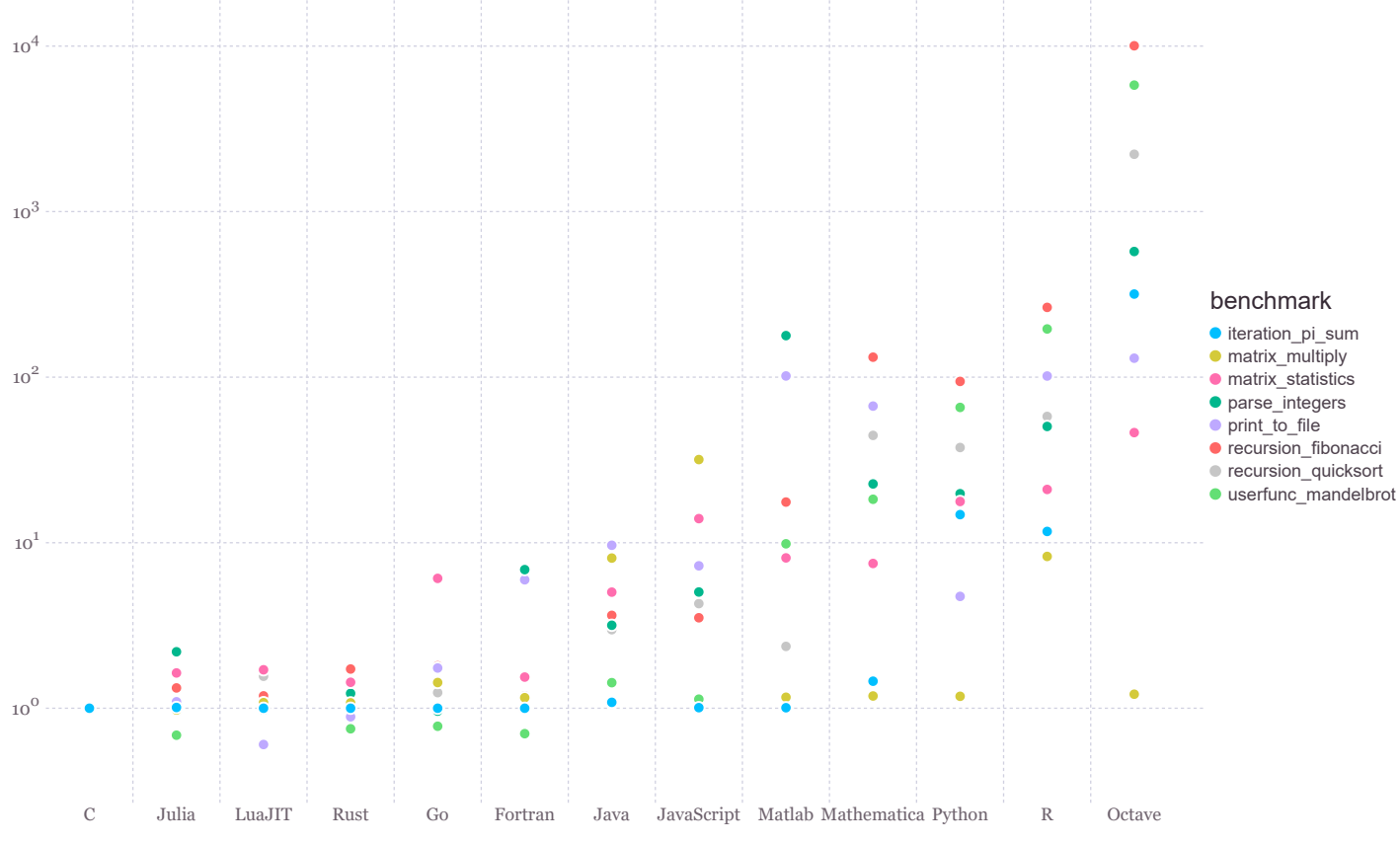
- High performance
- High readibility
- General-purpose language (opposite to domain-specific language, DSL)
- Designed for numerical computing, and for domain experts
- Support multi-paradigm (e.g. object-oriented, functional, metaprogramming)
- Built-in package manager
- Easy concurrency and distributed computing

Why Julia?

Tow language problem

科學運算及數值運算需要高效能的運算，這樣的情境同樣也適用資料科學及大數據的運算環境。

Benchmarks



[Julia microbenchmark (https://julialang.org/benchmarks/)](https://julialang.org/benchmarks/)

Traditional Chinese resource

[Julia traditional Chinese docs (https://docs.juliatw.org/latest/)](https://docs.juliatw.org/latest/)

Outline

- Numbers
- Operators
- Control Flow
- String and Operators
- Functions

In [1]:
```
println("Hello World!")
```

```
Hello World!
```

In [2]:
```
print("Hello World!")  # 不會自動換行
```

```
Hello World!
```

In [3]:
```
# 這個叫作單行註解
```

註解是一種執行後，會被編譯器忽略的部份

In [4]:
```
#=
當然你可以寫多行註解
=#
```

Numbers

In [5]:
```
5
```

Out[5]: 5

In [6]:
```
2 + 5
```

Out[6]: 7

In [7]:
```
2.0 + 5
```

Out[7]: 7.0

對於數字的運算來說，他就是一台純粹的計算機

Numbers

數字對於 Julia 來說並不是"同樣"的

數字可以分成：

- 整數（Integer）
- 浮點數（Floating point）

Julia 還有支援

- 有理數（Rational number）
- 複數（Complex number）

Integer

Integer

Type	Signed?	Bit size	Smallest value	Largest value
Int8	T	8	-2^7	2^7-1
Uint8	F	8	0	2^8-1
Int16	T	16	-2^{15}	$2^{15}-1$
Uint16	F	16	0	$2^{16}-1$
Int32	T	32	-2^{31}	$2^{31}-1$
Uint32	F	32	0	$2^{32}-1$
Int64	T	64	-2^{63}	$2^{63}-1$
Uint64	F	64	0	$2^{64}-1$
Int128	T	128	-2^{127}	$2^{127}-1$
Uint128	F	128	0	$2^{128}-1$
Bool		8	false	true

以上表摘錄自官方網站

我要怎麼知道我用的整數是什麼型別？

In [8]:
```
typeof(100)
```

Out[8]:
```
Int64
```

如果沒有特別宣告的話，會依據系統位元數決定

In [9]:
```
Int
```

Out[9]:
```
Int64
```

我能不能宣告其他型別的數字？

In [10]:
```
Int8(10)
```

Out[10]:
```
10
```

電腦的數字儲存方式

電腦儲存數字的方式是用 `01` 的方式儲存及運算

所以一個 `Int8` （8 bit integer）會以

□□□□□□□□

的方式儲存

00000001 就是 1

000000010 就是 2

00000100 就是 4

10000100 則是 -4

整數的上下限

In [11]:
```
typemax(Int8)
```

Out[11]:
```
127
```

他相當於 01111111

In [12]:
```
typemin(Int8)
```

Out[12]:
```
-128
```

他則是 10000000

Overflow!

那如果對最大的數 +1 的話會怎麼樣？

In [13]:
```
typemax(Int64) + 1
```

Out[13]:
```
-9223372036854775808
```

有趣的事情就發生了！

Bitwise operators（位元運算子）

當 x 跟 y 是整數或是布林值...

- ~x： bitwise not，對每個位元做 $\neg x$ 運算，`11100->00011`
- x & y： bitwise and，對每個位元做 $x \wedge y$ 運算
- x | y： bitwise or，對每個位元做 $x \vee y$ 運算
- x $ y: bitwise xor，對每個位元做 $x \oplus y$ 運算
- x >>> y： 位元上，將 x 的位元右移 y 個位數
- x >> y： 算術上，將 x 的位元右移 y 個位數
- x << y： 將 x 的位元左移 y 個位數

Bitwise operators（位元運算子）

In [14]:
```
Int8(1) << 2  # 將Int8(1)的位元左移2個位數
```

Out[14]: 4

In [15]:
```
Int8(4) >> 2  # 將Int8(4)的位元右移2個位數
```

Out[15]: 1

In [16]:
```
~Int8(4)  # 00000100 -> 11111011
```

Out[16]: -5

Bitwise operators（位元運算子）

In [17]:
```
Int8(4) & Int8(8)  # 00000100 & 00001000 = 00000000
```

Out[17]:
```
0
```

In [18]:
```
Int8(4) | Int8(8)  # 00000100 | 00001000 = 00001100
```

Out[18]:
```
12
```

Floating-Point Numbers

Floating-Point Numbers

In [19]:
```
0.5
```

Out[19]:
```
0.5
```

In [20]:
```
.5
```

Out[20]:
```
0.5
```

In [21]:
```
typeof(0.5)
```

Out[21]:
```
Float64
```

In [22]:
```
2.5e-4
```

Out[22]:
```
0.00025
```

Floating-Point Numbers

Type	Bit size
Float16	16
Float32	32
Float64	64

Infinite value

Float16	Float32	Float64
Inf16	Inf32	Inf
-Inf16	-Inf32	-Inf
NaN16	NaN32	NaN

Infinite value

In [23]:
```
1 / Inf
```

Out[23]:
```
0.0
```

In [24]:
```
1 / 0
```

Out[24]:
```
Inf
```

In [25]:
```
-5 / 0
```

Out[25]:
```
-Inf
```

In [26]:
```
0 / 0
```

Out[26]:
```
NaN
```

Arithmetic operators（算術運算子）

當 x 跟 y 都是數字...

- +x：就是 x 本身
- -x：變號
- x + y,x - y,x * y,x / y：一般四則運算
- div(x, y)：商
- x % y：餘數，也可以用rem(x, y)
- x \ y：跟/的作用一樣
- x ^ y：次方

Arithmetic operators

In [27]:
```
div(123, 50)
```

Out[27]: 2

In [28]:
```
123 % 50
```

Out[28]: 23

Comparison operators（比較運算子）

用在 x 跟 y 都是數值的狀況

- x == y：等於
- x != y, x ≠ y：不等於
- x < y：小於
- x > y：大於
- x <= y, x ≤ y：小於或等於
- x >= y, x ≥ y：大於或等於

Comparison operators（比較運算子）

- +0 會等於 -0，但不大於 -0
- Inf 跟自身相等，會大於其他數字除了 NaN
- -Inf 跟自身相等，會小於其他數字除了 NaN

In [29]:
```
Inf == Inf
```

Out[29]:
```
true
```

In [30]:
```
Inf > NaN
```

Out[30]:
```
false
```

In [31]:
```
Inf == NaN
```

Out[31]:
```
false
```

In [32]:
```
Inf < NaN
```

Out[32]:
```
false
```

變數的宣告及使用

Variables

In [33]:
```
x = 5.0
```

Out[33]: 5.0

In [34]:
```
x
```

Out[34]: 5.0

Naming

變數開頭可以是字母（A-Z or a-z）、底線或是Unicode（要大於 00A0）

運算符號也可以是合法的變數名稱，例如 + ，通常用於指定function（函式）

支援Unicode作為變數名稱

In [35]:
```
δ = 0.00001
```

Out[35]:
```
1.0e-5
```

In [36]:
```
안녕하세요 = "Hello"
```

Out[36]:
```
"Hello"
```

支援類LaTeX語法

打 \delta 之後，按下tab

In [37]:
```
δ
```

Out[37]:
```
1.0e-5
```

\alpha -*tab*- _2 -*tab*

In [38]:
```
α₂
```

```
UndefVarError: α₂ not defined

Stacktrace:
 [1] top-level scope at In[38]:1
```

Unicode Input (https://docs.julialang.org/en/v1/manual/unicode-input/)

內建常數

In [39]:
```
pi
```

Out[39]: π = 3.1415926535897...

In [40]:
```
π
```

Out[40]: π = 3.1415926535897...

In [41]:
```
ℯ
```

Out[41]: e = 2.7182818284590...

命名指南

- 建議命名都用小寫
- 字跟字之間請用底線隔開，像 `lower_case` ，不過不鼓勵使用底線，除非影響到可讀性
- 型別的命名請以大寫開頭，並使用 `CamelCase` 的寫法
- 函式或是 macro 請用小寫加上底線的寫法
- 函式如果會修改到輸入的變數的話，請在函式名稱後加上！，像 `transform!()`

Numeric Literal Coefficients

In [42]:
```
x = 3
```

Out[42]: 3

In [43]:
```
2x^2 - 3x + 1
```

Out[43]: 10

In [44]:
```
(x-1)x
```

Out[44]: 6

Complex numbers and rational numbers

Complex numbers

In [45]:
```
1 + 2im
```

Out[45]:
```
1 + 2im
```

In [46]:
```
(1 + 2im) + (3 - 4im)
```

Out[46]:
```
4 - 2im
```

In [47]:
```
(1 + 2im) * (3 - 4im)
```

Out[47]:
```
11 + 2im
```

In [48]:
```
(-4 + 3im)^(2 + 1im)
```

Out[48]:
```
1.950089719008687 + 0.6515147849624384im
```

In [49]:
```
3 / 5im == 3 / (5*im)
```

Out[49]:
```
true
```

Complex numbers

In [50]:
```
real(1 + 2im)
```

Out[50]:
```
1
```

In [51]:
```
imag(3 + 4im)
```

Out[51]:
```
4
```

In [52]:
```
conj(1 + 2im)
```

Out[52]:
```
1 - 2im
```

In [53]:
```
abs(3 + 4im)  # 複數平面上的向量長度
```

Out[53]:
```
5.0
```

In [54]:
```
angle(3 + 4im)  # 複數平面上的向量夾角
```

Out[54]:
```
0.9272952180016122
```

Complex numbers

In [55]:
```
1 + Inf*im
```

Out[55]:
```
1.0 + Inf*im
```

In [56]:
```
1 + NaN*im
```

Out[56]:
```
1.0 + NaN*im
```

In [57]:
```
(1 + NaN*im)*(3 + 4im)
```

Out[57]:
```
NaN + NaN*im
```

Rational numbers (有理數)

In [58]:
```
2//3
```

Out[58]: 2//3

In [59]:
```
-6//12  # 會自動約分
```

Out[59]: -1//2

In [60]:
```
5//-20  # 自動調整負號
```

Out[60]: -1//4

In [61]:
```
numerator(2//10)  # 約分後的分子（numerator）
```

Out[61]: 1

In [62]:
```
denominator(7//14)  # 約分後的分母（denominator）
```

Out[62]: 2

Rational numbers

In [63]:
```
10//15 == 8//12
```

Out[63]:
```
true
```

In [64]:
```
2//4 + 1//7
```

Out[64]:
```
9//14
```

In [65]:
```
3//10 * 6//9
```

Out[65]:
```
1//5
```

In [66]:
```
float(3//4)  # 轉成浮點數
```

Out[66]:
```
0.75
```

Rational numbers

In [67]:
```
1//(1 + 2im)  # 分母實數化
```

Out[67]:
```
1//5 - 2//5*im
```

In [68]:
```
5//0  # 可以接受分母為0
```

Out[68]:
```
1//0
```

In [69]:
```
3//10 + 1  # 與整數運算
```

Out[69]:
```
13//10
```

In [70]:
```
3//10 - 0.8  # 與浮點數運算
```

Out[70]:
```
-0.5
```

In [71]:
```
2//10 * (3 + 4im)  # 與複數運算
```

Out[71]:
```
3//5 + 4//5*im
```

Boolean

In [72]:
```
true
```

Out[72]:
```
true
```

In [73]:
```
typeof(false)
```

Out[73]:
```
Bool
```

Negation

- !x：true變成false，false變成true

In [74]:
```
!true
```

Out[74]:
```
false
```

Bitwise operators

In [75]:
```
~false
```

Out[75]:
```
true
```

In [76]:
```
true & false
```

Out[76]:
```
false
```

In [77]:
```
true | false
```

Out[77]:
```
true
```

Updating operators（更新運算子）

In [78]:
```
x = 5
y = 0
```

Out[78]:
```
0
```

In [79]:
```
y += 2x
```

Out[79]:
```
10
```

Updating operators

- `+=`：`x += y`等同於`x = x + y`，以下類推
- `-=`
- `*=`
- `/=`
- `\=`
- `%=`
- `^=`
- `&=`
- `|=`
- $\veebar$= ($\veebar$: \xor-tab)
- `>>>=`
- `>>=`
- `<<=`

Control Flow

條件判斷

```
if <判斷式>
    <運算>
end
```

In [80]:

```
x = 0
y = 5

if x < y
    println("x is less than y")
end
```

```
x is less than y
```

條件判斷

In [81]:
```
x = 10
y = 5

if x < y
  println("x is less than y")
end
```

In [82]:
```
if x < y
  println("x is less than y")
else
  println("x is not less to y")
end
```

```
x is not less to y
```

條件判斷

In [83]:
```
if x < y
    println("x is less than y")
elseif x > y
    println("x is greater than y")
else
    println("x is equal to y")
end
```

```
x is greater than y
```

非布林值是不能當作判斷式的

In [84]:
```
if 1  # 數字不會自動轉成布林值
    print("true")
end
```

```
TypeError: non-boolean (Int64) used in boolean context

Stacktrace:
 [1] top-level scope at In[84]:1
```

短路邏輯

In [85]:
```
if 3 > 5 && 5 == 5
    println("This is not going to be printed.")
end
```

- 在 a && b 裡，如果 a 為 false，就不需要考慮 b 了
- 在 a || b 裡，如果 a 為 true，就不需要考慮 b 了

連續比較

In [86]:
```
1 < 2 < 3
```

Out[86]:
```
true
```

比較特殊値

In [87]:
```
isfinite(5)
```

Out[87]:
```
true
```

In [88]:
```
isinf(Inf)
```

Out[88]:
```
true
```

In [89]:
```
isnan(NaN)
```

Out[89]:
```
true
```

迴圈

```
while <持續條件>
    <運算>
end
```

In [90]:
```
a = 0
i = 1
while i <= 100
    a += i
    i += 1
end
a
```

Out[90]:
```
5050
```

廻圏

```
for i = 1:100
    <運算>
end
```

In [91]:
```
a = 0
for i = 1:100
    a += i
end
a
```

Out[91]:
```
5050
```

Characters and Strings

Characters

字元是組成字串的基本單元

In [92]:
```
'A'
```

Out[92]:
```
'A': ASCII/Unicode U+0041 (category Lu: Letter, uppercase)
```

In [93]:
```
'a'
```

Out[93]:
```
'a': ASCII/Unicode U+0061 (category Ll: Letter, lowercase)
```

字元用單引號，字串用雙引號

In [94]:
```
typeof('A')
```

Out[94]:
```
Char
```

In [95]:
```
typeof("A")
```

Out[95]:
```
String
```

字元其實是用相對應的整數表示的

In [96]:
```
Int('A')
```

Out[96]:
```
65
```

In [97]:
```
Char(65)
```

Out[97]:
```
'A': ASCII/Unicode U+0041 (category Lu: Letter, uppercase)
```

In [98]:
```
Int('B')
```

Out[98]:
```
66
```

字元能適用加法嗎？

In [99]:
```
'A' + 1
```

Out[99]:
```
'B': ASCII/Unicode U+0042 (category Lu: Letter, uppercase)
```

In [100]:
```
'C' - 2
```

Out[100]:
```
'A': ASCII/Unicode U+0041 (category Lu: Letter, uppercase)
```

字元可以比較大小嗎？

In [101]:
```
'C' > 'A'
```

Out[101]:
```
true
```

In [102]:
```
'a' > 'A'
```

Out[102]:
```
true
```

In [103]:
```
Int('a')
```

Out[103]:
```
97
```

In [104]:
```
'a' - 'A'
```

Out[104]:
```
32
```

Strings

In [105]:
```
x = "Hello World!"
```

Out[105]:
```
"Hello World!"
```

In [106]:
```
"""Hello World!"""
```

Out[106]:
```
"Hello World!"
```

In [107]:
```
"""Hello
World
!
"""
```

Out[107]:
```
"Hello\nWorld\n!\n"
```

Indexing

In [108]:
```
x[1]
```

Out[108]:
```
'H': ASCII/Unicode U+0048 (category Lu: Letter, uppercase)
```

In [109]:
```
x[end-1]
```

Out[109]:
```
'd': ASCII/Unicode U+0064 (category Ll: Letter, lowercase)
```

In [110]:
```
x[3:5]
```

Out[110]:
```
"llo"
```

Unicode and UTF-8

In [111]:
```
s = "\u2200 x \U2203 y"
```

Out[111]:
```
"∀ x ∃ y"
```

In [112]:
```
s[1]
```

Out[112]:
```
'∀': Unicode U+2200 (category Sm: Symbol, math)
```

In [113]:
```
s[2]
```

```
StringIndexError("∀ x ∃ y", 2)

Stacktrace:
 [1] string_index_err(::String, ::Int64) at ./strings/string.jl:12
 [2] getindex_continued(::String, ::Int64, ::UInt32) at ./strings/string.jl:220
 [3] getindex(::String, ::Int64) at ./strings/string.jl:213
 [4] top-level scope at In[113]:1
```

用來告訴你下一個index

In [114]:
```
nextind(s, 1)
```

Out[114]:
```
4
```

In [115]:
```
s[4]
```

Out[115]:
```
' ': ASCII/Unicode U+0020 (category Zs: Separator, space)
```

Operators

In [116]:
```
length("123456")
```

Out[116]: 6

Interpolation

In [117]:
```
x = "Today"
y = "Sunday"
string(x, " is ", y)
```

Out[117]:
```
"Today is Sunday"
```

In [118]:
```
"$x is $y"
```

Out[118]:
```
"Today is Sunday"
```

In [119]:
```
"1 + 2 = $(1 + 2)"
```

Out[119]:
```
"1 + 2 = 3"
```

Equivalent

In [120]:
```
"1 + 2 = 3" == "1 + 2 = $(1 + 2)"
```

Out[120]:
```
true
```

Contains substring

In [121]:
```
occursin("banana", "na")
```

Out[121]:
```
false
```

Repeat

In [122]:
```
repeat(x, 10)
```

Out[122]:
```
"TodayTodayTodayTodayTodayTodayTodayTodayTodayToday"
```

Join strings

In [123]:
```
join(["apples", "bananas", "pineapples"], ", ", " and ")
```

Out[123]:
```
"apples, bananas and pineapples"
```

Functions

What is function?

當有些程式需要不斷被重複使用，只需要更改一部份程式碼即可

這些程式碼就可以被**抽出來（abstract）**，成為 function

讓這部份程式可以有更**廣泛的（generic）**用處，而不是**狹隘而特定的（specific）**

In [124]:
```
function f(x, y)
    return x + y
end
```

Out[124]:
```
f (generic function with 1 method)
```

In [125]:
```
f(1, 2)
```

Out[125]:
```
3
```

當你呼叫函式 f(1, 2) 的時候， x=1 與 y=2 會被傳送給 f 。

函式就會進行後續的運算，並把運算結果透過 return 進行回傳。

當函數被呼叫，記憶體會空出一塊空間給函式，是函式的運算空間。

In [126]:
```
f(f(1, 2), 3)
```

Out[126]: 6

當以上函式被呼叫，最內部的函式 f(1, 2) 會先被運算，等運算結果回傳之後，才運算外層的函式 f(3, 3) 。

另一種函式定義方式

短小輕巧的函式在Julia很常見

In [127]:
```
h(x, y) = x + y
```

Out[127]:
```
h (generic function with 1 method)
```

In [128]:
```
h(1, 2)
```

Out[128]:
```
3
```

Specify input and output datatype

In [129]:
```
function g(x::Int64, y::Int64)::Int64
    return x + y
end
```

Out[129]:
```
g (generic function with 1 method)
```

In [130]:
```
g(1, 2)
```

Out[130]:
```
3
```

In [131]:
```
g(1.2, 2.3)
```

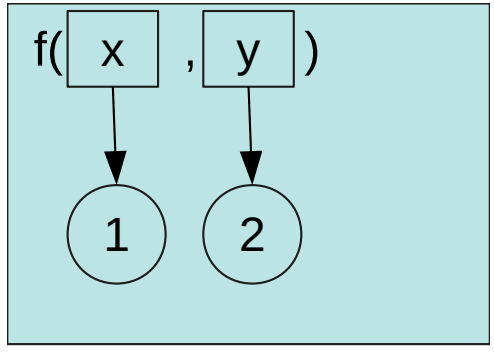
```
MethodError: no method matching g(::Float64, ::Float64)

Stacktrace:
 [1] top-level scope at In[131]:1
```

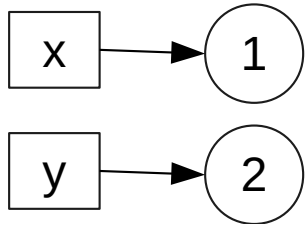
Argument passing

Call-by-value

複製一份變數的值到函式中

e.g. C, primitive values in Java

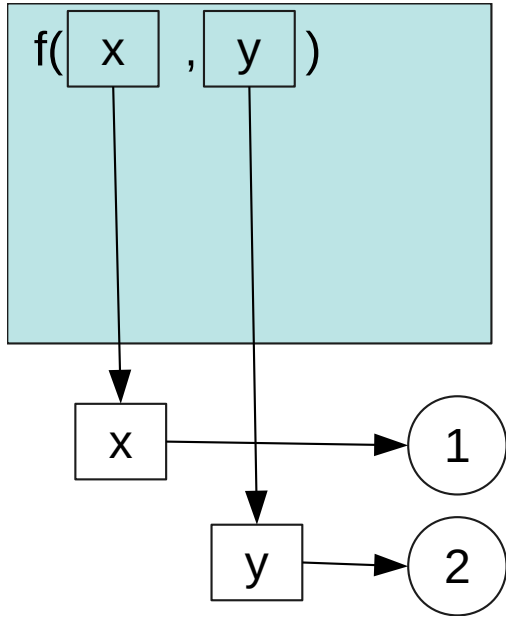




Call-by-reference

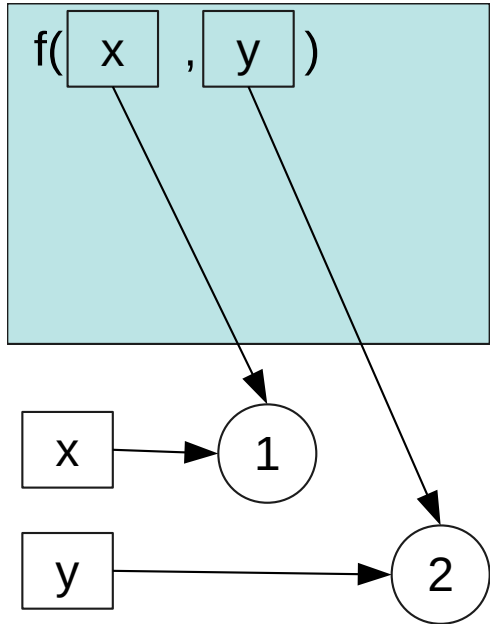
在函式中製造一個參考（reference），參考指向變數

e.g. Python, object in Java



Pass-by-sharing

傳參數時，並不會複製一份給函式，但是參數本身會作為一個新的變數**綁定（bind）**到原本值的位址



如何驗證以上的行為？

In [132]:
```
println(objectid(1))
```

```
11967854120867199718
```

In [133]:
```
x = 1
println(objectid(x))
```

```
11967854120867199718
```

In [134]:
```
function sharing(x)
    println(objectid(x))
    x = 2
    println(objectid(x))
end
```

Out[134]:
```
sharing (generic function with 1 method)
```

In [135]:
```
sharing(x)
```

```
11967854120867199718
5352850025288631388
```

In [136]:
```
x
```

Out[136]:
```
1
```

Operators are functions

In [137]:
```
1 + 2 + 3 + 4 + 5 + 6
```

Out[137]: 21

In [138]:
```
+(1, 2, 3, 4, 5, 6)
```

Out[138]: 21

Anonymous functions

In [139]:
```
a = () -> println("Calling function a.")
```

Out[139]:
```
#3 (generic function with 1 method)
```

In [140]:
```
a()
```

```
Calling function a.
```

In [141]:
```
b = x -> println(x)
```

Out[141]:
```
#5 (generic function with 1 method)
```

In [142]:
```
b(5)
```

```
5
```

In [143]:
```
c = (x, y) -> x + y
```

Out[143]:
```
#7 (generic function with 1 method)
```

In [144]:
```
c(2, 3)
```

Out[144]:
```
5
```

Tuples

In [145]:
```
x = (1, 2, 3)
```

Out[145]:
```
(1, 2, 3)
```

In [146]:
```
x[1]
```

Out[146]:
```
1
```

In [147]:
```
x[2:3]
```

Out[147]:
```
(2, 3)
```

Tuple is immutable

In [148]:
```
objectid(x)
```

Out[148]:
```
0xedbeb2cfeb9b46ff
```

In [149]:
```
objectid(x[2:3])
```

Out[149]:
```
0x1d867d66d42a6657
```

Unpacking

In [150]:
```
a, b, c = x
```

Out[150]:
```
(1, 2, 3)
```

In [151]:
```
a
```

Out[151]:
```
1
```

In [152]:
```
b
```

Out[152]:
```
2
```

In [153]:
```
c
```

Out[153]:
```
3
```

Swap

In [154]:
```
b, a = a, b
```

Out[154]: (1, 2)

In [155]:
```
a
```

Out[155]: 2

In [156]:
```
b
```

Out[156]: 1

Tuple is the data structure that pass arguments to function

In [157]:
```
h(1, 2)
```

Out[157]:
```
3
```

`return` keyword

In [158]:
```
function sumproduct(x, y, z)
    return (x + y) * z
end
```

Out[158]:
```
sumproduct (generic function with 1 method)
```

In [159]:
```
function sumproduct(x, y, z)
    (x + y) * z
end
```

Out[159]:
```
sumproduct (generic function with 1 method)
```

Multiple return values

In [160]:
```
function shuffle_(x, y, z)
    (y, z, x)
end
```

Out[160]:
```
shuffle_ (generic function with 1 method)
```

Argument destruction

In [161]:
```
x = [1, 2, 3]
shuffle_(x...)
```

Out[161]:
```
(2, 3, 1)
```

等價於 `shuffle_(1, 2, 3)`

Vectorizing functions

適用 operators 跟 functions

In [162]:
```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

In [163]:
```
x .^ 2
```

Out[163]:
```
10-element Array{Int64,1}:
   1
   4
   9
  16
  25
  36
  49
  64
  81
 100
```

User-defined function

In [164]:
```
f(x) = 3x
```

Out[164]:
```
f (generic function with 2 methods)
```

In [165]:
```
f.(x)
```

Out[165]:
```
10-element Array{Int64,1}:
  3
  6
  9
 12
 15
 18
 21
 24
 27
 30
```

Q & A

In [ ]: