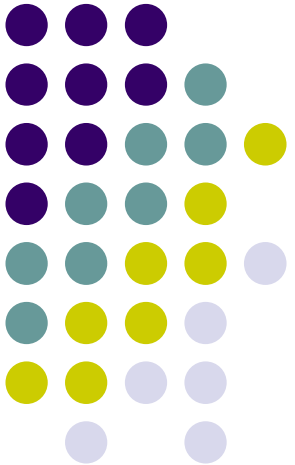
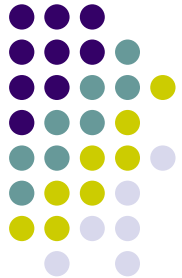
Python程式設計入門
數值型別

葉難

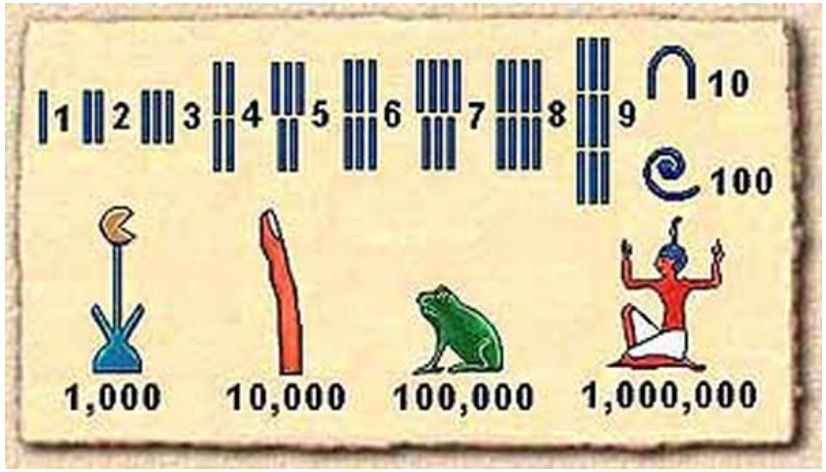


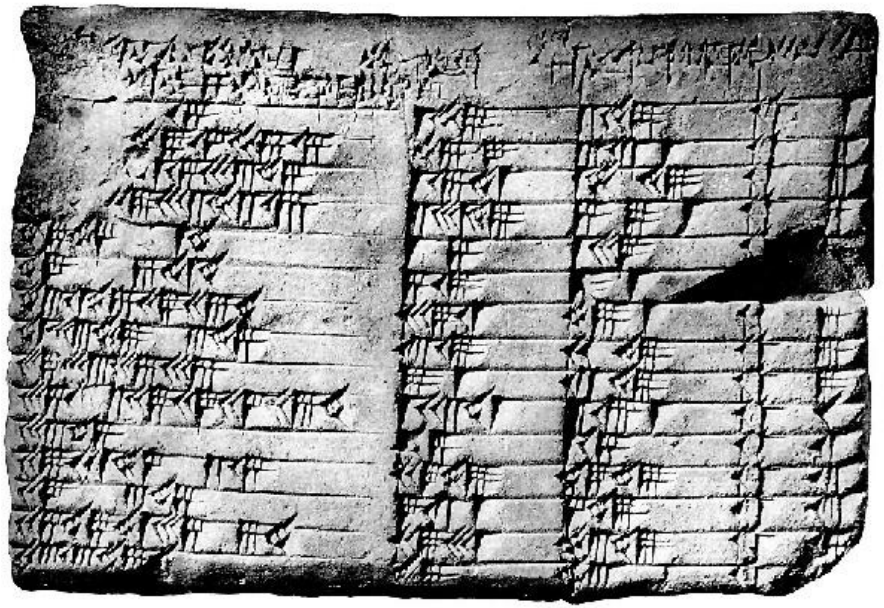
大綱



- 數值型別、表示法
- int、float、bool（布林型別）
- complex（複數）、Decimal（十進位數）、Fraction（分數）
- 位元運算

表達數字



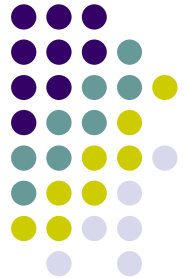


I	V	X	L	C	D	M
1	5	10	50	100	500	1000

42 = 50 - 10 + 1 + 1 = XLII

2015 = 1000 + 1000 + 10 + 5 = MMXV

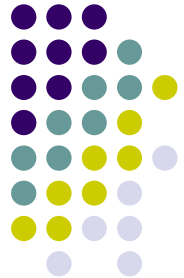
int，10、16、8、2進位表示法



- Python的型別int，無窮精確度
- 16進位：0x或0X
- 8進位：0o或0O，注意數字0與字母O
- 2進位：0b或0B

```
>>> 99, 0b1100011, 0x63, 0o143
(99, 99, 99, 99)
>>> 99, 0B1100011, 0X63, 0O143
(99, 99, 99, 99)
```

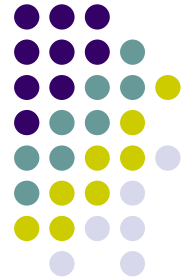
2.x版有int與long



- int是固定精確度的整數型別
- 超過界限後，會自動轉成long（無窮精確度）

```
>>> import sys
>>> sys.maxint      # int最大值
2147483647
>>> 2 ** 33, 1 - 2 ** 33   # 超過
(8589934592L, -8589934591L)
>>> 3L, -3L                # 附加L表示
(3L, -3L)
>>> type(2 ** 33)
<type 'long'>
```

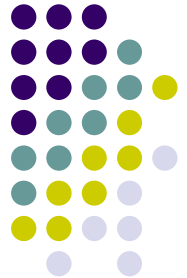
內建函式int（也是建構式）



- 除了可接受數值，也可接受字串，並指定何種進位

```
>>> int(10), int(0b1010), int(0x0a)
(10, 10, 10)
>>> int('99'), int('99', 10), int(99.99)
(99, 99, 99)
>>> int('0b11', 2), int('11', 2)    # 2進位
(3, 3)
>>> int('0xFF', 16), int('FF', 16) # 16進位
(255, 255)
>>> int('0b11', 0), int('0xFF', 0) # 自動判斷
(3, 255)
```

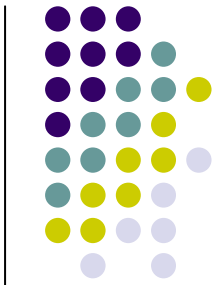
float的基本認知



- 底層硬體採二進位浮點數格式，通常是IEEE-754
- 無法精確表示某些數字，如「0.1」！
- 事實上，很多很多很多數字都無法精確表示

```
>>> 1.0 / 3                    # 1 / 3 也可以
0.3333333333333333
>>> 0.1                        # Shell好心（雞婆），
0.1                            # 弄成容易閱讀的樣子
>>> 0.1 + 0.2
0.30000000000000004
```

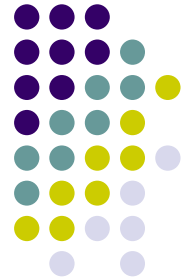
問題：0.1加10次



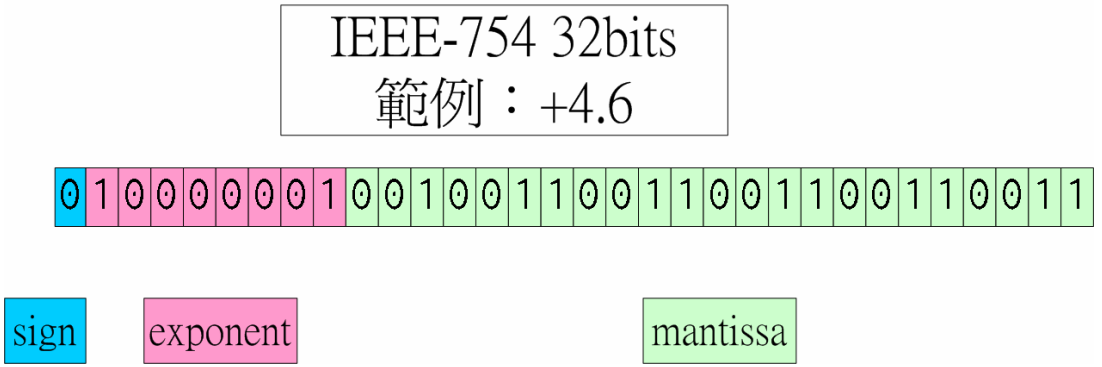
- 差異雖小，累積後就可能出問題
- 數學實數無限多個；然而電腦記憶體有限，無法完美表達

```
>>> x = 0
>>> for _ in range(10):     # 慣例：以「_」代表
...     x += 0.1            # 不需使用的名稱
...
>>> x
0.9999999999999999          # !
```

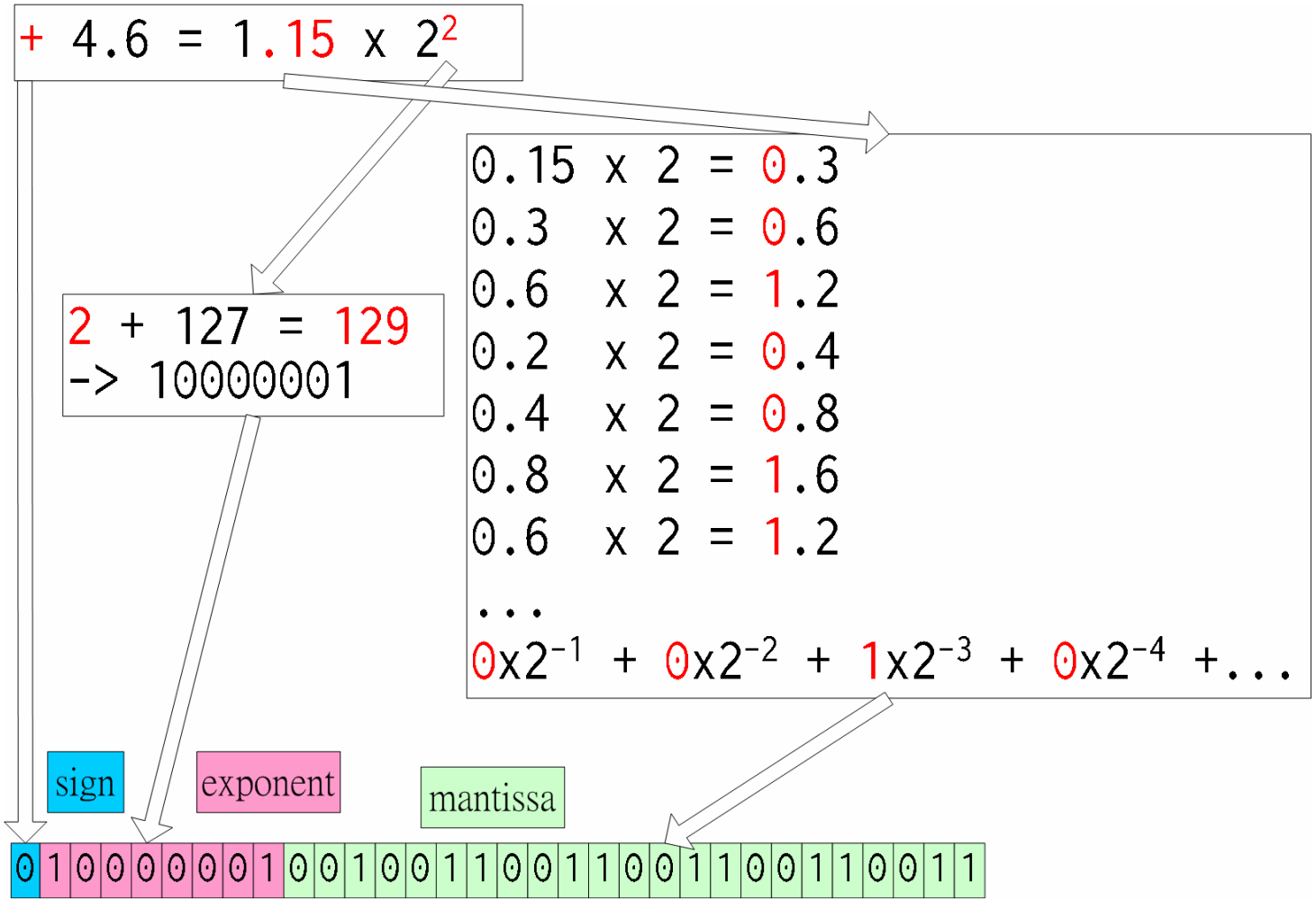
範例：「+4.6」轉IEEE-754格式



- 「+」：「sign」是「0」
- 4.6 / 2 = 2.3、2.3 / 2 = 1.15，所以4.6 -> 1.15 x 2^2
- 使用127作爲bias value，127+2得到129 -> 10000001，這是「exponent」
- 然後不斷乘上2，取出個位數，作爲「mantissa」
- 0.15 x 2 = 0.3 -> 0
- 0.3 x 2 = 0.6 -> 0
- 0.6 x 2 = 1.2 -> 1

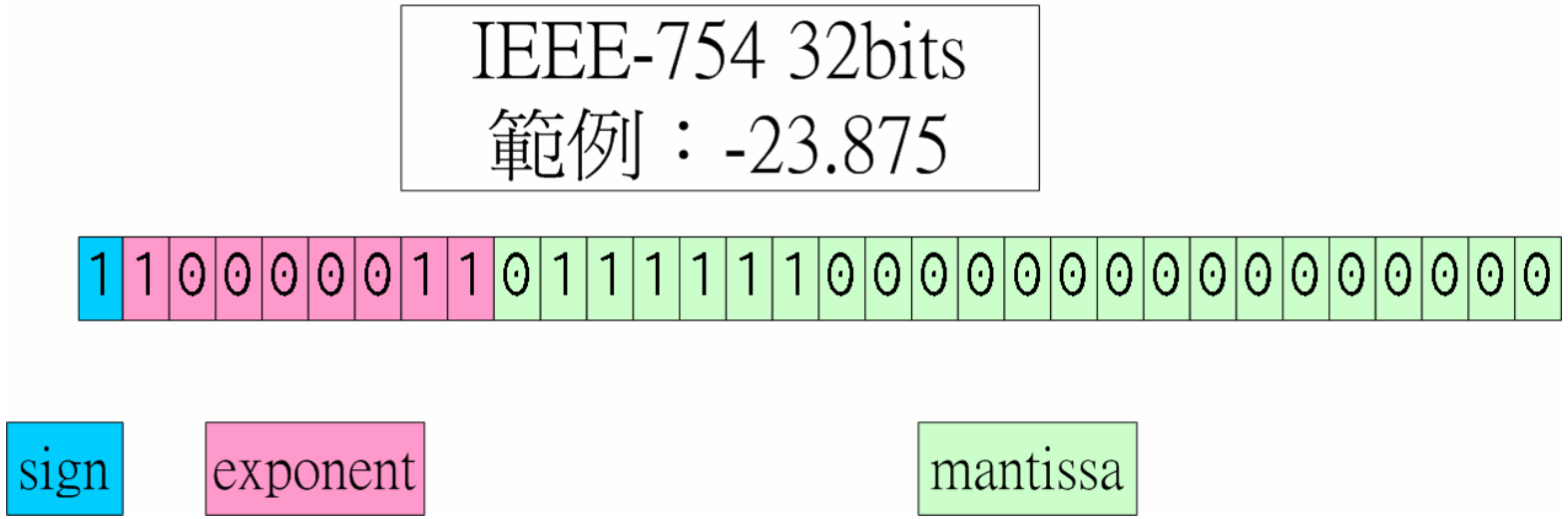


範例：「+4.6」示意圖

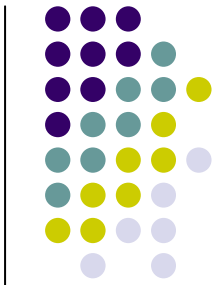

+ 4.6 = 1.15 x 2^2
0.15 x 2 = 0.3
0.3 x 2 = 0.6
0.6 x 2 = 1.2
0.2 x 2 = 0.4
0.4 x 2 = 0.8
0.8 x 2 = 1.6
0.6 x 2 = 1.2
...
0x2^-1 + 0x2^-2 + 1x2^-3 + 0x2^-4 +...
2 + 127 = 129
-> 10000001
sign
exponent
mantissa
0 1 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1

範例： IEEE-754格式逆轉

- 「sign」是「1」，所以是負數「-」
- 「exponent」10000011 = 131，131-127=4
- 「mantissa」2^{-1}x0 + 2^{-2}x1 + 2^{-3}x1...=0.4921875
- -1.492875 x 2^{4} = -23.875



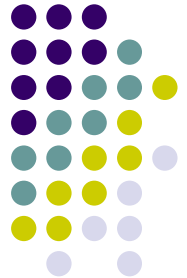
float表示法



- 小數點、科學記號「e」

```
>>> 3.0, 3., 3.14159
(3.0, 3.0, 3.14159)
>>> 5E2, 22e3, 3e8      # 5百，22K，光速
(500.0, 22000.0, 300000000.0)
>>> 6.02e23, 1e-6       # 莫耳數，微米
(6.02e+23, 1e-06)
>>> 5 e 2               # e前後不可有空格
 5 e 2
     ^
SyntaxError: invalid syntax
```

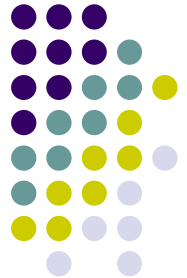
內建函式**float**（也是建構式）



- 僅接受一個參數，可以是數值或字串

```
>>> float(), float(-3), float(0xFF)
(0.0, -3.0, 255.0)
>>> float(3.14), float(-45.56e78)
(3.14, -4.556e+79)
>>> float('3.14'), float('-45.56e78')
(3.14, -4.556e+79)
```

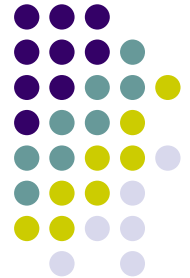
float特殊數值



- 無限大、無限小、非數（Not a Number）
- 病毒效應

```
>>> float('nan'), float('inf'), float('-inf')
(nan, inf, -inf)
>>> 1e309, -1e309          # 太大了、大小了
(inf, -inf)
>>> float('nan') * 2, float('inf') * 2   # 影響後續運算
(nan, inf)
>>> float('nan') == float('nan')    # 非數不等於任何東西
False                               # 包括它自己
>>> import math
>>> math.isnan(float('nan')), math.isinf(float('inf'))
(True, True)
```

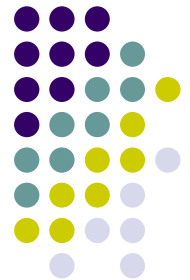
內建函式和模組**math**



- abs、pow、divmod、round
- 模組math：ceil、floor、sqrt、log、trunc

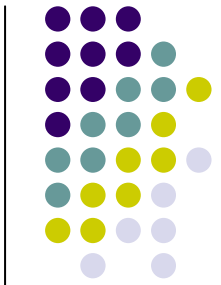
```
>>> round(3.678), round(3.14159, 2)
(4, 3.14)                              # 四捨五入
>>> round(1234.5678, -2)               # 到小數點前面
1200.0
>>> import math
>>> math.sqrt(3**2 + 4**2)             # 畢氏定理
5.0
>>> math.pow(27, 1.0/3)                # 立方根
3.0
```

範例：平方根
牛頓法



- 欲求一數n的平方根，如√2約等於1.414...
- 先隨便猜個答案ans，如1
- 然後算出((n/ans) + ans)/2，作爲下個答案
- 重複上述步驟，直到答案夠好了
- 例：n爲2
  ((2/1) + 1) / 2得到1.5，
  ((2/1.5) + 1.5) / 2得到1.4166666666666665，
  依此類推

範例：平方根
my_sqrt.py



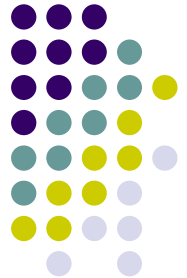
```
diff = 0.00001

def is_ok(n, ans):
    return abs(ans**2 - n) < diff

def get_better(n, ans):
    return ((float(n) / ans) + ans) / 2

def my_sqrt(n):
    ans = 1                          # 起初先隨便猜1
    while not is_ok(n, ans):         # 夠不夠好
        ans = get_better(n, ans)     # 逼近
    return ans
```

範例：平方根
my_sqrt_m.py



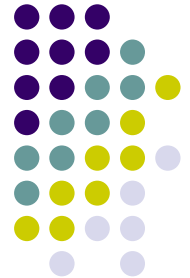
```
def my_sqrt(n, diff=0.00001):
    def is_ok(n, ans, diff):
        return abs(ans**2 - n) < diff

    def get_better(n, ans):
        return ((float(n) / ans) + ans) / 2

    ans = 1
    while not is_ok(n, ans, diff):
        ans = get_better(n, ans)

    return ans
```

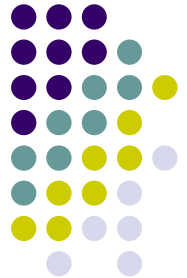
布林型別bool



- 型別bool代表真假值，Python已內建名稱True與False
- 其底層實作爲數字1和0，但最好不要這麼用

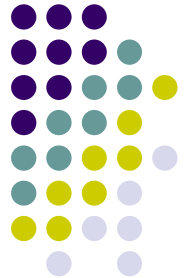
```
>>> type(True), type(False)
(<class 'bool'>, <class 'bool'>)
>>> True is 1, True == 1
(False, True)
>>> sum([True, False, True, False])
2
```

真假值一般化



- 除了True與False，其他物件也可作爲真假值
- None、0、0.0、空字串「"」、空list或空tuple（或空的容器），會被當做「False」（假）
- 其餘皆會被當做「True」（真）

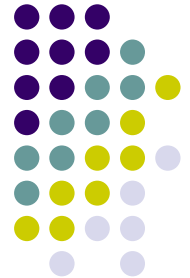
型別complex（複數）



- 模組cmath，含有與複數相關的函式

```
>>> import cmath
>>> x = 4+5j
>>> abs(x)
6.403124237432848 5
>>> cmath.phase(x)          # 輻角（弧度）
0.8960553845713439
>>> cmath.polar(x)          # 極座標
(6.403124237432848 5, 0.8960553845713439)
>>> cmath.sqrt(x)           # 平方根
(2.2806933416652985+1.0961578895015195j)
```

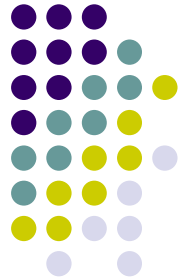
模組decimal的型別Decimal



- 某些領域產業，如金融銀行，不能接受float的特性，小差異會造成大問題
- Decimal可提供適當精確度，指定有效位數

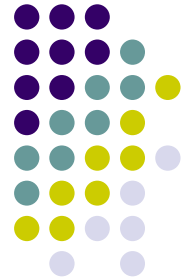
```
>>> from decimal import Decimal
>>> Decimal(0.1)      # 注意
Decimal('0.1000000000000000055511151231257827021181583404541015625')
>>> Decimal('0.1')    # 使用字串
Decimal('0.1')
>>> Decimal('3.14') * 3 * 3
Decimal('28.26')
```

Decimal的運算



```
>>> from decimal import Decimal as D
>>> D('7.62') * D('5.45')
Decimal('41.5290')
>>> li = [D('0.11'), D('0.23'), D('0.15')]
>>> max(li), sum(li)
(Decimal('0.23'), Decimal('0.49'))
>>> D('1.5').sqrt(), D('1.5').exp()
(Decimal('1.22474487139158904909864203735'),
  Decimal('4.48168907033806482260205546'))
```

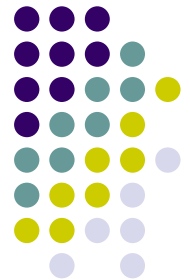
算術環境：精確度、捨入規則等



- getcontext()可取得算術環境，含多項設定

```
>>> decimal.getcontext().prec        # 精確度
28
>>> decimal.getcontext().prec = 4    # 改爲4
>>> D('1.2') / D('3.4')
Decimal('0.3529')
>>> D('1.2341') + D('0.0005')        # 捨入
Decimal('1.235')
>>> decimal.getcontext().rounding =
  decimal.ROUND_FLOOR                # 地板捨入法
>>> D('1.2341') + D('0.0005')
Decimal('1.234')
```

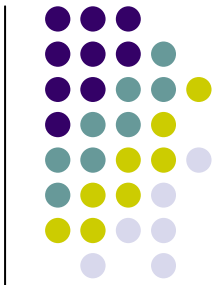
模組fractions的型別Fraction



- 三分之一「1/3」具有無限位數，float與Decimal都無法完美表示

```
>>> from fractions import *
>>> Fraction(2, 6)              # 分子、分母
Fraction(1, 3)                  # 自動約分
>>> F = Fraction                # 別名
>>> F('-10/30')
Fraction(-1, 3)
>>> Fraction('1.414')           # 字串
Fraction(707, 500)
>>> Fraction(1.414)             # float
Fraction(6368089873101881, 4503599627370496)
```

Fraction基本操作



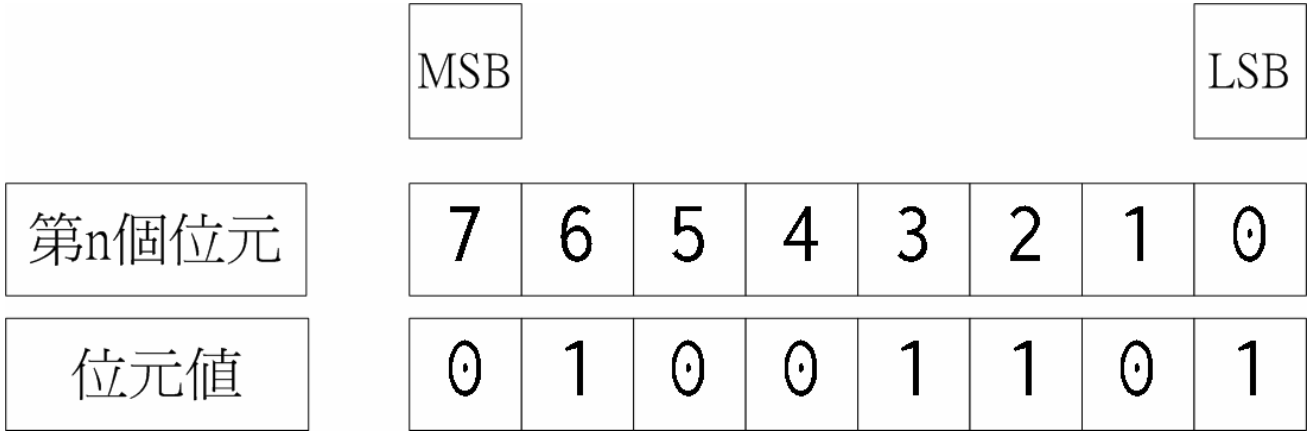
- 基本操作

```
>>> F()                                 # 一分之零
Fraction(0, 1)
>>> F(F(1, 3), F(2, 5))                 # 傳入Fraction
Fraction(5, 6)
>>> F(1.1, 2)
TypeError: both arguments should be Rational instances
>>> abs(F(1, 2) - F(5, 6))
Fraction(1, 3)
>>> F('3.1415926535897932').limit_denominator()
Fraction(3126535, 995207)                 # 限定分母位數
```

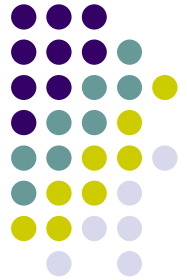
位元（bit-wise）運算

- 僅對整數型別int有意義
- 把int值的內容，當做一連串有順序的位元（bit）

```
>>> bin(5), 1*(2**2) + 0*(2**1)+ 1*(2**0)
('0b101', 5)
>>> bin(77)
'0b1001101'
```



二補數表示法

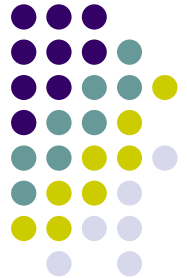


- 若是8位元空間，可表示(2**7 - 1)到-(2**7)的值
- 正數的最高有效位元（MSB）是0
- 負數的最高有效位元（MSB）是1
- 「零」只有一個，「0000 0000」
- 加減法很簡單
- 若是32位元， (2**31 - 1)到-(2**31)的值

例：二補數表示法、8位元

十進位	二進位	說明
127	0111 1111	最大，等於(2 ** 7) - 1
126	0111 1110	
...	...	...
3	0000 0011	
2	0000 0010	
1	0000 0001	
0	0000 0000	零
-1	1111 1111	
-2	1111 1110	
-3	1111 1101	
...	...	...
-126	1000 0010	
-127	1000 0001	
-128	1000 0000	最小，等於-(2 ** 7)

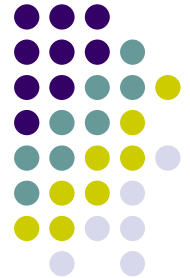
Python的int



- 可表達無窮位數（如果記憶體足夠的話）
- int的二進位表示法，請想像成：
- 正數左邊有無限多個0，負數左邊有無限多個1
- 例「3」：...0000000000011
- 例「-3」：...1111111111101
- 另外請注意，bin()的表示法不一樣

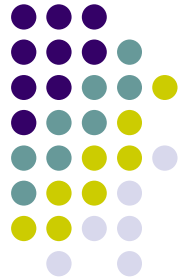
```
>>> bin(3), bin(-3)
('0b11', '-0b11')
```

位元運算子
「 ~、<<、>>、 &、^、|」



- ~x：位元運算「NOT」， 0變1、1變0
- x << n：往左位移，每個位元往左位移n次，右邊新位元填入0
- x >> n：往右位移，每個位元往右位移n次
- x & y：位元運算「AND」，兩者的位元都是1才會是1，否則爲0
- x ^ y：位元運算「XOR」，兩者的位元若不同則爲1，否則爲0
- x | y：位元運算「OR」，兩者的位元都是0才會是0，否則爲1

範例

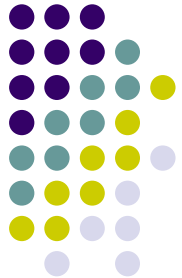


```
>>> x = 17; bin(x), bin(~x)
('0b10001', '-0b10010')
>>> bin(x << 2), x << 2
('0b1000100', 68)
>>> bin(x >> 2), x >> 2
('0b100', 4)
>>> bin(x & 0b11010), x & 0b11010
('0b10000', 16)
>>> bin(x ^ 0b11010), x ^ 0b11010
('0b1011', 11)
```

問題：

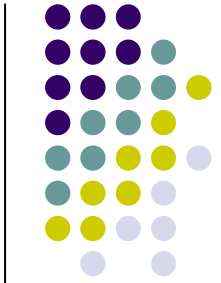
- 使用位元運算子，判斷某int物件是否爲奇數、偶數
- set(x, n)：把第n個位元設爲1
- clear(x, n)：把第n個位元設爲0
- toggle(x, n)：切換第n個位元的值，也就是0、1互換
- get(x, n)：取得第n個位元的值
- get_byte(x, n)：取得第n個byte的值
- getbits(x, p, n) ：從第p個位元向右，取出n個位元
  例：getbits(0b10101010, 5, 3)是0b101

範例：冪集合
powerset.py



```
def powerset(li):
    ps = []
    for n in range(0, 2**len(li)):
        sub = []
        x = n
        for i in range(len(li)):
            if x & 0x1:
                sub.append(li[i])
            x >>= 1
        ps.append(sub)
    return ps
```

問題：**float**精確度



- 請問結果是True還是False？

```
>>> x = (1 << 53) + 1
>>> x + 1.0 < x
```

？

Q&A