

UNIT 5



python™

String + Number

張傑帆 Chang, Jie-Fan



OUTLINE

- 數字
- 格式化
- 型態轉換
- 字串
- 字串應用



數字

- 數字在Python裡面可以分成兩大系統：
- 整數與浮點數
- 整數，又可分為
 - `Int`
 - `bool`
- `0`和`False`在意義上是相同的
- `1`和`True`也一樣有相同的意義
- 所以邏輯上來說「`a += 1`」和「`a += True`」是一樣的，只是一般並不會使用後者這樣寫法



數字

- 在電腦中，數值是以二進位儲存，並以十進位表示
- 也可以使用其他表示方式，例如二進位、八進位、十六進位等等，如下所示。

函數	描述
bin(int)	以字串形式傳回int的二進位表示方式
oct(int)	以字串形式傳回int的八進位表示方式
hex(int)	以字串形式傳回int的十六進位表示方式

- 以下的例子便是在說明不同進位系統結果的差異：



數字

01	<code>print(bin(23097351))</code>	# 印出如何用二進位制表示23097531
02	<code>print(oct(23097351))</code>	# 印出如何用八進位制表示23097531
03	<code>print(hex(23097351))</code>	# 印出如何用十六進位制表示23097531
04		
05	<code>print(0b1011000000111000000000111)</code>	# 以0b 開頭表示此數值為二進位制
06	<code>print(0o130070007)</code>	# 以0o 開頭表示此數值為八進位制
07	<code>print(0x1607007)</code>	# 以0x 開頭表示此數值為十六進位制

>>>

0b1011000000111000000000111

0o130070007

0x1607007

23097351

23097351

23097351

>>>

Decimal
Octal
Binary
Hexadecimal



數字

- 在浮點數中，又可分為
 - float
 - complex
 - decimal
- float便是指最常見的浮點數，精確度範圍則視所使用的Python編譯器決定
- 電腦本身是以二進位的方式來儲存數值
因此像0.5這樣的數值都可以被精準地儲存
但是像0.1與0.2這類型無法以 $1/2$ 的次方表示的數值
在儲存上就會有些許誤差，不過這是大部分程式語言都普遍存在的問題。



數字

- **complex**則是用來儲存複數類型的數值資料
- 這裡的複數跟數學的複數指的都是同一個東西
也就是會把數字分成**實部**及**虛部**
並分別以**兩個float**來表示複數的實部與虛部
- 表示方式為實部的**float**以+或-連接虛部的**float**
並在**虛部float**後面加上**j**，例如 **$4.7 + 6j$** 與 **$0.7j$** 等
- 注意，如果**實部為0**則可以直接**忽略**
比如說 **$6j$** 即是實部為0



格式化

- **Python** 有個特別的輸出方式，就是所謂的格式化
- 這跟硬碟的格式化完全不一樣，在這裡指的是在輸出的地方先以「**%**」與其他文字做結合，後面再將要輸出的資訊加以詳述
- 好處是可以在格式化的時候**規定輸出的格式**
- 例如，在**print** 函數的引數中以字串方式表達包含下表中描述內容所欲格式化的類型，然後再於**%()** 中填入欲格式化的變數名稱

```
print('text%x' %(tmp))
```



格式化

類型	描述
%%	在字串中顯示%。
%d	以10進位整數方式輸出。
%f	將浮點數以10進位方式輸出。
%e,%E	將浮點數以10進位及科學記號輸出。
%o	以8進位整數方式輸出。
%x,%X	以16進位整數方式輸出。
%s	使用str()將字串輸出 <code>print(chr(65)) => 'A'</code>
%c	以chr方式輸出。
%r	使用repr()輸出字串。



格式化

■ 示範程式碼

```
01 a = 7134
```

```
02 print('Hex a = %x, Dec a = %d, Oct a = %o'%(a, a, a))
```

■ 執行結果

```
>>>
```

```
Hex a = 1bde, Dec a = 7134, Oct a = 15736
```

```
>>>
```

- 在上面程式碼中，使用三種進位（十六進位、十進位、八進位）之整數方式作為格式化的輸出



格式化

01	<code>import math</code>	# 匯入import math 模組
02	<code>print('PI = %f'%(math.pi))</code>	# 以預設方式輸出math.pi
03	<code>print('PI = %8.4f'%(math.pi))</code>	# 四捨五入到小數點後第四位，
04		# 並設定總長為8，空白補在左邊
05	<code>print('PI = %-8.4f'%(math.pi))</code>	# 四捨五入到小數點後第四位，
06		# 並設定總長為8，空白補在右邊

>>>

PI = 3.141593

PI = 3.1416

PI = 3.1416

>>>



格式化

■ 示範程式碼

```
01 print('%0.3s'%(iverson))    # 取字串的前三位元輸出
02 print('%07.4s'%(iverson))  # 取字串的前四位元輸出，總長為7，補空白在左
03 print('%04.7s'%(iverson))  # 取字串的前七位元輸出，總長為4，補空白在左
```

■ 執行結果

```
>>>
ive
    iver
iverson
>>>
```



格式化

- `%r` 是個比較特殊的格式化。
- 使用 `repr()` 這個函數來輸出，回傳物件的字串表達形式，就是不管後面接的是什麼，都是以字串來表示它，例如：

```
01 test = '%r %r %r'
02 print(test%(1, 2, 3))
03 print(test%('dad', 'mom', 'son'))
04 print(test%(True, False, 'Neither'))
05 print((repr(0), repr("1")))
```

```
>>>
```

```
1 2 3
```

```
'dad' 'mom' 'son'
```

```
True False 'Neither'
```

```
0 '1'
```

```
>>>
```



格式化

函數	描述
<code>str.format(*args, **kwargs)</code>	進行格式化字串運算

- 在3.x 版中，格式化的方式又多加了一種

```
01 a = '{}+{}={}'
```

```
02 print(a)
```

輸出a 原本的樣子

```
03 print(a.format(7, 9, 7 + 9))
```

以7,9,7+9 來替換掉三組{}

```
04
```

```
05 b = '{city}, {country}'
```

```
06 print(b)
```

輸出b 原本的樣子

```
07 print(b.format(city = "Taipei", country = "Taiwan"))
```

以Taipei 替換掉city

```
08
```

以Taiwan 替換掉country

```
10 c = '{1}+{2}={0}'
```

```
11 print(c)
```

輸出b 原本的樣子

```
12 print(c.format(3 + 4, 3, 4))
```

以3 + 4,3,4 依序替換掉{0},{1},{2}

```
>>>
```

```
{1}+{2}={0}
```

```
7+9=16
```

```
{city},{country}
```

```
Taipei, Taiwan
```

```
{1}+{2}={0}
```

```
3+4=7
```

```
>>>
```



小練習

- 輸出以下結果：
- 分別使用 `print(%)` 或 `format()` 來完成

```
>>>
```

```
姓名:帳單之門
```

```
年齡:99
```

```
性別:male
```

```
>>>
```



課堂練習二

- 請試做「兩數相加」題目並依格式輸出

$$5.10 + 2.30 = 7.40$$

- 請試做「九九乘法表」題目並依格式輸出

1*1=	1	1*2=	2	1*3=	3	1*4=	4	1*5=	5	1*6=	6	1*7=	7	1*8=	8
2*1=	2	2*2=	4	2*3=	6	2*4=	8	2*5=	10	2*6=	12	2*7=	14	2*8=	16
3*1=	3	3*2=	6	3*3=	9	3*4=	12	3*5=	15	3*6=	18	3*7=	21	3*8=	24
4*1=	4	4*2=	8	4*3=	12	4*4=	16	4*5=	20	4*6=	24	4*7=	28	4*8=	32
5*1=	5	5*2=	10	5*3=	15	5*4=	20	5*5=	25	5*6=	30	5*7=	35	5*8=	40



字串與格式化

- 補0

- Ex:

```
>>> print("會員編號:%(#)08d" % {"#" : 123456})
```

```
會員編號:00123456
```

- Ex:

```
>>> print("%8.2f" % (123.456))
```

```
123.46
```

- Ex:

```
>>> money=987.98
```

```
>>> print("$%*.2f" % (7, money))
```

```
$ 987.98
```



字串與格式化

- 輸出格式不只接受單純的數字和字串型態的變數
也可帶入整個字典型態變數
- 其字典索引須為字串 **str** 型態
- Ex:

```
>>> name={"game":"xbox", "apple":"iphone", "camera":"nikon"}  
>>> print("%(apple)s, %(camera)s, %(game)s" %name)  
iphone, nikon, xbox
```



字串與格式化

- 旗標(Flag)指定格式化的字串變數

- Ex:

```
>>> print("會員編號1:%d, 會員編號2:%d" % (10, 20))
```

會員編號1:10, 會員編號2:20

```
>>> print("會員編號2:%(#2)d, 會員編號1:%(#1)d"  
%{"#1":10, "#2":20})
```

會員編號2:20, 會員編號1:10

```
>>> print("會員編號1:%(num1)d, 會員編號2:%(num2)d"  
%{"num1":10, "num2":20})
```

會員編號1:10, 會員編號2:20

- 數字型態的必須先用str()函數轉換成字串

```
>>> a=1
```

```
>>> b=2
```

```
>>> ("會員編號1:%(1)d, 會員編號2:%(2)d" % {str(a):10, str(b):20})
```



小練習

- 請建立一dictionary 名為 short 其key為
 - NTU, TW, 112, CSIE, Train
- 所對應的value為
 - 台灣大學, 台灣, 台大ip, 資工系, 訓練班
- 令使用者可自由輸入列印指令:

■ Ex:

```
print("%(NTU)s" %short)
台灣大學
>>>
```

```
print("%(TW)s的%(NTU)s有個%(CSIE)s, %(Train)s最棒了, 112是%(112)s" %short)
```

```
台灣的台灣大學有個資工系, 訓練班最棒了....
```

- Hint: 可用 eval() 來執行使用者所輸入的指令



型態轉換

- 有時候，你的資料並不是你所想要使用的型態
- 這時候就需要**型態轉換**
- 利用**轉型函數**來將資訊轉換成特定的資料格式

函數	描述
int(x[,base])	將x轉換成整數，base是轉換的基數，欲設為十進位。
float(x)	將x轉換成浮點數。
str(x)	將x轉換為字串。
tuple(s)	將s轉換為tuple。
list(s)	將s轉換為list。
chr(x)	將x轉換為chr。

- **x** 表示物件，**s** 是表示字串



型態轉換

■ 示範程式碼

```
01 str1 = '64' #str1 中不能有超過要轉換的進位中不會出現的數字
02 print(int(str1)) # 預設為十進位
03 print(int(str1, 8)) # 將基數改為八進位
04 print(int(str1, 16)) # 將基數改為十六進位
```

■ 執行結果

```
>>>
64
52
100
>>>
```



01 #coding=UTF-8

02

03 TMP= input("Input:") # 輸入一個數字

04 print(type(TMP)) # 輸出TMP 的型別

05

06 TMP= int(TMP) # 把TMP 轉換為int 型別

07 print(type(TMP)) # 輸出TMP 的型別

08

09 TMP=float(TMP) # 把TMP 轉換為float 型別

10 print(type(TMP)) # 輸出TMP 的型別

>>>

Input:123

<class 'str'>

<class 'int'>

<class 'float'>

>>>



轉換成字串 STRING

- **Python**有內建的**str()** function 可以將任何資料型式轉換成字串 **string**

```
>>> "Hello " + str(2)
```

```
'Hello 2'
```

```
>>> a=123
```

```
>>> type(123)
```

```
<class 'int'>
```

```
>>> b = str(a)
```

```
>>> type(b)
```

```
<class 'str'>
```



字串

- 字串是程式語言中相當重要的一部分，經常被使用到
 - 傳達資訊給使用者
 - 獲取使用者的輸入資料時
- **Python** 內建函數庫，有多種函數可以直接使用
- 不只是字串有函數庫，其他資料型別也有類似的函數庫
- 有索引值（**index**），會回傳所代表的字元
- 值得一提的是，**index** 值也可以是負數

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]	s[11]
H	e	l	l	o	,		W	o	r	l	d
s[-12]	s[-11]	s[-10]	s[-9]	s[-8]	s[-7]	s[-6]	s[-5]	s[-4]	s[-3]	s[-2]	s[-1]



字串

- 定義字串的方式可以使用一對單引號或雙引號括起來
- 或使用三個單（雙）引號前後括住的字串還可以任意換行，例如：

```
string = """ 使用三個單(雙) 引號  
前後括住的字串可以任意換行"""
```

- 字串具有順序性，可以從中間取出或插入部分資料。
- 使用[] 運算子，形式是str [start : end : step]
- 這三個選項屬性，可以自由選擇是否使用其預設值為：
 - start 為0
 - end 為len(str)，也就是str 的最後一個index 值再加上1
 - step 為1
- print(s)、print(s[:])、 print(s[0:len(s):1]) 結果會一樣



字串型態的切片(SLICE)

■ Ex:

```
>>> toast="PYTHONSLICE"
```

```
>>> toast[-5]
```

```
'S'
```

... -3 -2 -1



```
>>> toast[-5:]
```

```
'SLICE'
```

```
>>> toast[:6]
```

```
'PYTHON'
```

```
>>> toast[:6]+toast[-5:]
```

```
'PYTHONSLICE'
```

■ Ex:

```
>>> 'PYTHONSLICE'[:6]==toast[:6]
```

```
True
```



字串

>>>

Hello, World

llo, World

Hello, The Beautiful World

Hello, World

Hlo ol

dlroW ,olleH

drW,le

>>>

01 s = 'Hello, World'

02

03 print(s) # 直接印出s

04 print(s[2:]) # 設定為[2:]，代表從index = 2 的字元到最後一個字元

05 print(s[0:7] + 'The Beautiful' + s[6:12])
利用+ 運算子將多個字串連接起來

07 print(s[::1]) # 設定為[::1]，表示從第0 個字元到最後一個字元，
以1 為間隔取值

08 print(s[::2]) # 設定為[::2]，表示從第0 個字元到最後一個字元，
以2 為間隔取值

09 print(s[::-1]) # 設定為[::-1]，會將整個字串顛倒輸出

10 print(s[-1:0:-2]) # 設定為[-1:0:-2]，從-1 開始往回到0，以-2 間隔
取值



字串

- 而最後一次輸出則是 `s[-1:0:-2]`，這可能有些難理解，以下將以圖形說明之：

表 4-2 str 的 index 比對表

H	e	l	l	o	,		W	o	r	l	d
s[-12]	s[-11]	s[-10]	s[-9]	s[-8]	s[-7]	s[-6]	s[-5]	s[-4]	s[-3]	s[-2]	s[-1]

`s[-1:0:-2] = s[::-2] == "drW,le"`

元組型態的切片

■ Ex:

```
>>> toast = "PYTHONSLICE"
```

```
>>> tuples = toast[0:3],toast[3:6],toast[6:9],toast[9:11]
```

```
>>> tuples #不加()則自動組成tuple
```

```
('PYT', 'HON', 'SLI', 'CE')
```

```
>>> tuples[0]
```

```
'PYT'
```

```
>>> tuples[2:4]
```

```
('SLI', 'CE')
```

```
>>> tuples[1][0]
```

```
'H'
```

■ Ex:

```
>>>
```

```
("Sunday","Monday","Tuesday","Wednesday","Thursday","  
Friday","Saturday")[3]
```

```
'Wednesday'
```



字典型態的切片

- Ex:

```
>>>
```

```
days={1:"Sunday",2:"Monday",3:"Tuesday",4:"Wednesday",5:"Thursday",6:"Friday",7:"Saturday"}
```

```
>>> days[2],days[3],days[4]
```

```
('Monday', 'Tuesday', 'Wednesday')
```

- 數字型態無法切片

如: `a = 12345` 無法用 `a[2]` 取出 3

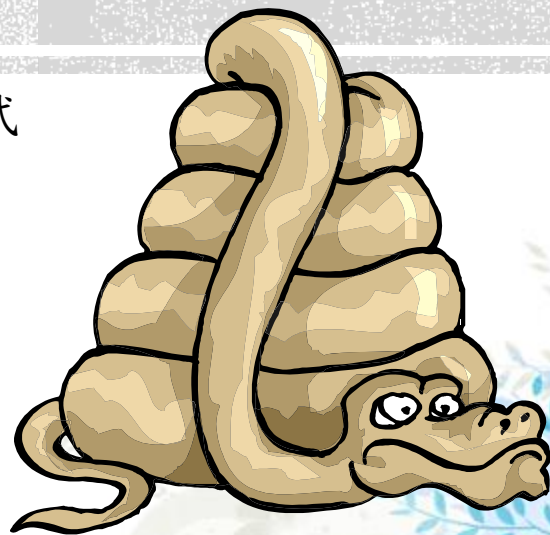
可用數學的方法取出或

先用 `str()` 函數轉換成字串





字串轉換與常用函式



字串函數使用

- **string.strip([chars])**

- 將string字串變數裡的左右兩邊的空白字元刪除掉
- chars引數不為None時會決定string.strip()函數要刪除的字元

- **string.swapcase()**

- 將string字串裡的字母大小寫互轉



字串函數使用

- `string.lstrip(s[, chars])`
- 將s字串變數內左邊的多餘空白字元去掉，chars引數必須傳入字串型態
- chars決定string.lstrip()函數要去掉x字串變數內的那些字元，預設只會刪去空白字元
- Python 2.2版是不能使用chars引數
- Ex:

```
>>> text = "aaaaa bbbbbb aaa ccccc "
```

```
>>> text.lstrip(" a")
```

```
'bbbbbb aaa ccccc'
```

```
>>> text.lstrip(" ab")
```

```
'cccc'
```



字串函數使用

- `string.capitalize()` 函數
- 變數第一個字轉變為大寫
- Ex:

```
>>> print("how are you?".capitalize())  
How are you?
```



字串函數使用

- `string.title()`
 - 將字串內所有為[a-z]的單字第一個字元轉換成大寫
- `string.translate(map)`
 - 將 `string` 中的字元以 `map` 中配對的字元轉換
 - 搭配 `str.maketrans(from, to)`
- Ex:

```
intab = "aeiou"  
outtab = "12345"  
trantab = str.maketrans(intab, outtab)  
  
str = "this is string example....wow!!!"  
print str.translate(trantab);
```

th3s 3s str3ng 2lpl2....w4w!!!



字串函數使用

- `string.upper()`

- 將string字串變數內的字母從小寫轉換為大寫

- `string.lower()`

- 將string內的字元從大寫字母轉換為小寫字母

- `string.zfill(width)`

- 將string變數內的字串前面補0，直到string變數的長度等於width引數設定的長度



小練習

- 輸入一字串將其中小寫字元轉成大寫字元
- 例如：輸入 **abCdE123** -> 輸出 **ABCDE123**
- 提示：使用 **str.upper()**



字串函數使用

- `str.endswith(suffix[, start[, end]])`
- 判斷字串內是否有符合`suffix`引數的值
- Ex:

```
>>> images="xbox.gif, iphone.jpg"
```

```
>>> images.endswith(".jpg")
```

```
True
```

```
>>> images.endswith(".gif",0, 8)
```

```
True
```

```
>>> images.endswith(".gif")
```

```
False
```



字串函數使用

- **str.startswith(prefix[, start[, end]])**
 - 判斷傳入的prefix字串字元是否為開始字元
- **str.istitle()**
 - 判斷字串變數裡的第一個字是否為大寫
 - 如果宣告一句英文句子，句子裡的每一個單字都會判斷
- **str.isupper()**
 - 判斷字串變數內的所有字母是否為大寫
 - 不會理會特殊字元



字串函數使用

- `str.islower()`

- 判斷字串變數內的字元是否全部都是小寫

- `str.isalnum()`

- 判斷該變數裡的內容是否為[\[a-z\]](#)、[\[A-Z\]](#)與[\[0-9\]](#)的字元
- 不可以判別多行宣告

- `str.isalpha()`

- 與`str.isalnum()`的差異在於這個函數只接受字串內有英文字母



字串函數使用

- `str.isdigit()`
 - 判斷字串內是否為數字
- `str.isspace()`
 - 判斷字串變數是否為空白字元



課堂練習

- 使用者可以輸入任意數字 n
- 當輸入的 n 不為數字，提示使用者輸入型態錯誤，並且重新讓使用者繼續輸入
- 若輸入的值為數字，將其`print`至螢幕上
- `ex.`
- `n=100`



字串

- **str** 本身有幾種常用的函數，在這裡列表說明：

`str.split([sep=None, maxsplit=-1])`

以sep分割成子字串，回傳儲存子字串的list，maxsplit為子字串最多的數量。

`str.count(sub[, start[,end]])`

計算 sub 出現的次數並以int型態回傳，start 為起始計算索引值，end 為結束索引值

`str.find(sub[, start[,end]])`

回傳 sub 第一次出現的索引值，若sub不在其中則回傳-1。start值與end值定義尋找的範圍



字串

`str.index(sub[, start[,end]])`

回傳 `sub` 第一次出現的索引值，若`sub`不在其中則回傳 `ValueError`。 `start`值與`end`值定義尋找的範圍。

`str.replace(old, new[,count])`

將 `str` 中的`old` 子字串以 `new`子字串代換。

如果有給定`count`值，則只有被給定的`count`值所代表的`old`子字串會被替換掉。

- 在上表 `[]` 中的文字是可以自由選擇要不要加上，如果需要裡面的功能，就把需要的`[]`內容加上，反之則可以省略



字串

- **str.split([sep=None, maxsplit=-1])**
 - 將字串作分割，裡面有兩個選項屬性可以使用
 - **sep** 就是分割器，sep 的值就是想拿來作分割器的值。
 - 如果沒有給定sep 值或是將sep 值給予None 值，則會把空白字元當作分割器，並且在回傳時自動把所有空白字元都刪除掉
 - **maxsplit** 是最多分割幾次，預設為-1。
如果有給定非負整數的值（例如給定2），則會分割成最多2+1 個子字串



01 # 故意在各單字之間放置兩個半型空白

02 string = 'apple is a kind of fruit'

03

04 print(string.split(sep=' ')) # 使用半型空白當作分割器

05 print(string.split()) # 不設定分割器，會自動以空白: >>>

06 # 並在回傳時捨棄子字串中的空 ['apple', 'is', 'a', 'kind', 'of', 'fruit']

07 print(string.split(sep=None)) # 跟不設定分割器的: ['apple', 'is', 'a', 'kind', 'of', 'fruit']

08 print(string.split(maxsplit=0)) # 設定最多只分割0 ['apple', 'is', 'a', 'kind', 'of', 'fruit']

09 # 因此不會執行分割 ['apple is a kind of fruit']

10 print(string.split(maxsplit=2)) # 設定最多只分割2 ['apple', 'is', 'a', 'kind', 'of fruit']

11 # 因此會回傳3 個子字串 >>>

12 print(string.split(sep=None, maxsplit=4)) # 設定最多只分割4 次，

13 # 因此會回傳5 個子字串



字串函數使用

- `str.splitlines(keepends)`
- 將字串進行分割
- 以 “\n”和 “\r”作為分割的區隔字元
- 以序列型態回傳
- **Keepends**引數預設**False**，設為**True**會連同脫逸字元一併回傳



字串函數使用

- `str.partition(sep)`
- 將字串做分割，但只會分割第一個符合`sep`引數的字元，形成3-tuple
- Python 2.5板新增的功能
- Ex:

```
>>> "C:\\|D:\\|E:\\|G:\\".partition('|')
('C:\\', '|', 'D:\\|E:\\|G:\\')
>>> "C:\\|D:\\|E:\\|G:\\".partition('|')[0]
'C:\\'
>>> "C:\\|D:\\|E:\\|G:\\".partition('|')[1]
'|'
>>> "C:\\|D:\\|E:\\|G:\\".partition('|')[-1]
'D:\\|E:\\|G:\\'
```



字串

- **str.count(sub[, start[, end]])**
 - 計算str 字串中sub 子字串出現的次數
 - **sub** 是要搜尋的字串，一定要給值
 - **start** 從給定的index值開始計數
 - **end** 計數的動作到給定的index值即停止
不包含給定的index的位置
- ※不能只給定end 值而不給定start 值。



字串

```
>>> for i in range(len(s)):  
      print(i, " ", s[i])
```

```
01 s = 'A clear conscience laughs at false accusation.'  
   # 只要問心無愧，無端的指責可以一笑置之。  
02 print(s.count('a'))      # 在s字串中統計小寫字母a出現的次數  
03 print(s.count('a', 6))   # 從index = 6之後開始統計  
04 print(s.count('a', 0, 36)) # 從index = 0之後開始統計到index = 36  
05                             # 不包含36喔~~~試試 35
```

```
>>>
```

```
6
```

```
5
```

```
5
```

```
>>>
```



字串

- **str.find(sub[, start[, end]])**
- 在**str** 裡面找尋第一次出現**sub** 子字串的**index** 值
- 也就是找**sub**在**str**裡的位置
- 找不到回傳-1
- 跟**str.count**一樣也有兩個選擇性選項屬性
 - **start** 從給定的**index**值開始計數
 - **end** 計數的動作到給定的**index**值即停止
不包含給定的**index**的位置



字串

```
01 s = 'A friend in need is a friend indeed.'  
02                                     # 患難見真情  
03 print(s.find('friend'))           # 找尋s 中friend 子字串出現的index  
04 print(s.find('friend', 3))        # 給定start 值為3  
05 print(s.find('friend', 3, 22))    # 給定start 值為3 且end 值為22
```

```
>>>
```

```
2
```

```
22
```

```
-1
```

```
>>>
```



字串

- **str.index(sub[, start[, end]])**
- 跟str.find 幾乎一模一樣，選項屬性也相同
- 唯一的差異點就是str.index 在搜尋範圍中沒有找到子字串的時候不會回傳-1，而是回報一個「ValueError」的錯誤



字串

```
01 s = 'An uncut gem goes not sparkle. '  
02                                     # 玉不琢，不成器  
03 print(s.index('o'))                 # 使用index() 在s 裡面尋找'o'，  
04                                     # 並回傳'o' 第一次出現的index 值  
05 print(s.index('t', 15))             # 給定start 值來尋找't'  
06 print(s.index('a', 5, 15))          # 給定start 值及end 值，在其中尋找'a'
```

```
>>>
```

```
14
```

```
20
```

Traceback (most recent call last):

File "C:\Users\John\Desktop\Google 雲端硬碟\Python Wizard\examples\Ch4\ex04_06.py", line 7, in <module>

print(s.index('a',5,15)) # 給定start 值及end 值，在其中尋找'a'

ValueError: substring not found

```
>>>
```



字串

- `index()` 跟 `find()` 的功能差別在於找不到子字串時回傳的資料不相同
- `find()` 會回傳 `-1`，因此使用 `find()` 來找尋子字串不需要處理 `Error` 訊息
- 使用 `index()` 就要準備 `Error` 訊息的處理方式
- 有關 `Error` 的錯誤處理，在後面章節教的例外處理會有詳細說明



字串

- **str.replace(old, new[, count])**
- 將舊的子字串old替換成新的子字串new
有一個選擇性選項屬性：count
- count 屬性如果有被給定值(例如在此先設為2)
str 裡面前兩次出現的old 將會被new 子字串取代
- 第三次(如果有)以後出現old 子字串將不會被取代



```
01 s = 'A penny saved is a penny earned. '
02 # 儲蓄方為賺錢之道
04 print(s.replace('n', 'l'))          # 使用replace() 將'n' 用'l' 取代
05 print(s)                            # replace() 並不會真的更改s 所參照物件的值
06 print(s.replace('penny', 'dollar')) # replace() 將penny 以dollar 取代
07 print(s.replace('penny', 'dollar', 1)) # 給定count 值為1，
08                                     # 故第二次出現的penny 並不會被取代
```

```
>>>
```

```
A pelly saved is a pelly earled.
```

```
A penny saved is a penny earned.
```

```
A dollar saved is a dollar earned.
```

```
A dollar saved is a penny earned.
```

```
>>>
```



字串

■ 範例1題目：

1. 令 `line1 = "Welcome to Python World Game."`
請使用 `str.split` 以 空格 做 分割。
2. 令 `line2 = "Can you can a can as a canner can can a can?"`
請使用 `str.count` 算出有幾個 'a'。
3. 令 `line3 = "Few free fruit flies fly from flames."`
請使用 `str.find` 或 `str.index` 找出 "fly" 第一次出現的位子。
4. 令 `line4 = " Czngxrxtulxtizns, yzu hxve pxssed the czmpetitizn."`
請使用 `str.replace` 把 'z' 替換成 'o'，以及把 'x' 替換成 'a'。



答案



課堂練習

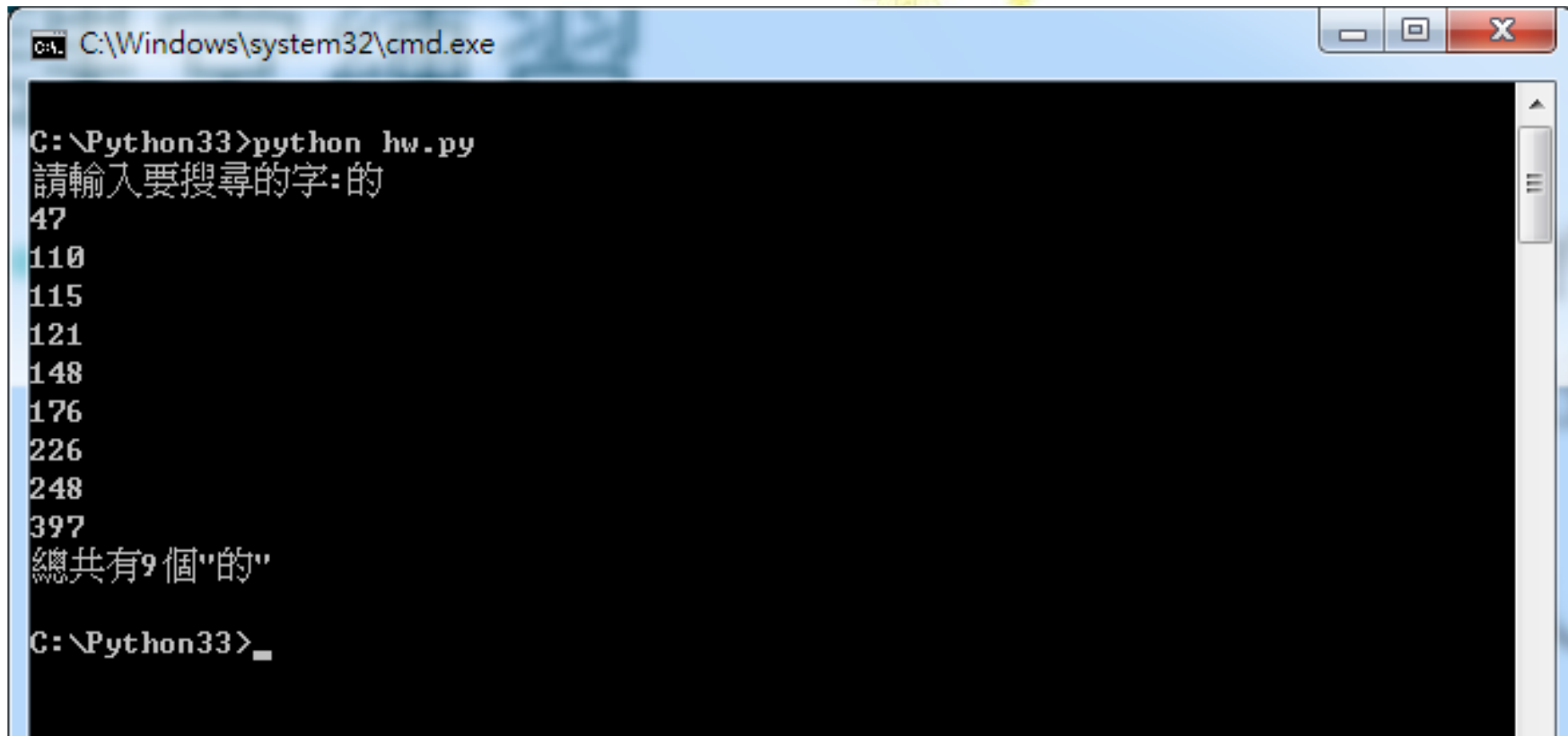
- 題目：在特定的文章(長恨歌)字串中搜尋輸入的字串

ex: '的'

- 練習1: 印出 總共有9個"的" (若輸入的字為"的")
- 練習2: 若有符合的字串，將其索引值印出
(全部印出，並非印出第一個符合的索引值)



範例圖



A screenshot of a Windows command prompt window. The title bar shows the path `C:\Windows\system32\cmd.exe`. The command prompt shows the following text:

```
C:\Python33>python hw.py
請輸入要搜尋的字:的
47
110
115
121
148
176
226
248
397
總共有9個"的"
C:\Python33>_
```



回家作業

- 1. 找出文章中出現最多的字

- Hint: 可用 `for c in text:`

- 2. 計算文章中「扣掉標點符號」後的字數

- Hint: 可用 `split` 或 `replace`



常用方法 JOIN 與 SPLIT

- Join 可將 list 裡的許多字串轉換成為單一的字串

<字串分隔符號>.join(<某list>)

```
>>> ";" .join( ["abc", "def", "ghi"] )  
"abc;def;ghi"
```

- Split 可將一字串依分割符號切割成許多子字串並存進一list中

<字串>.split(<切割符號>)

```
>>> "abc;def;ghi" .split( ";" )  
["abc", "def", "ghi"]
```



SPLIT & JOIN 與 LIST 的解析

- Split 與 join 可以與 list 配合，將所有的字首轉大寫，組成以下敘述：

```
>>> ".join( [s.capitalize() for s in "this is a test ".split( )] )
```

```
'This Is A Test'
```

```
>>> # For clarification:
```

```
>>> "this is a test" .split( )
```

```
['this', 'is', 'a', 'test']
```

```
>>> [s.capitalize() for s in "this is a test" .split()]
```

```
['This', 'Is', 'A', 'Test']
```



方法	描述
<code>str.capitalize()</code>	回傳將 <code>str</code> 改成首字母大寫，其餘字母小寫的字串
<code>str.center(width[, fillchar])</code>	回傳一個將 <code>str</code> 設置字串中央，長度 <code>width</code> 的新字串， <code>fillchar</code> 為填充字元，預設為空格
<code>str.count(sub[, start[, end]])</code>	計算 <code>sub</code> 出現的次數， <code>start</code> 為起始計算索引值， <code>end</code> 為結束索引值
<code>str.encode(encoding="utf-8", errors="strict")</code>	回傳 <code>encoding</code> 版本的 <code>bytes</code> 物件
<code>str.endswith(suffix[, start[, end]])</code>	判斷 <code>str</code> 是否以 <code>suffix</code> 結尾
<code>str.expandtabs([tabsize])</code>	將 <code>tab</code> 符號以 <code>tabsize</code> 的空格數替換
<code>str.find(sub[, start[, end]])</code>	回傳 <code>sub</code> 第一次出現的索引值
<code>str.format(*args, **kwargs)</code>	進行格式化字串運算
<code>str.index(sub[, start[, end]])</code>	回傳 <code>sub</code> 第一次出現的索引值
<code>str.isalnum()</code>	判斷字串中的字元是否至少一個是字母或數字
<code>str.isalpha()</code>	判斷字串中的字元是否至少一個是字母
<code>str.isdecimal()</code>	判斷字串中所有字元是否是十進位數字
<code>str.isdigit()</code>	判斷字串中所有字元是否是數字
<code>str.isidentifier()</code>	判斷字串是否可作為合法的識別字
<code>str.islower()</code>	判斷字串中所有字母字元是否都是小寫字母
<code>str.isnumeric()</code>	判斷字串中所有字元是否是數字
<code>str.isprintable()</code>	判斷字串中所有字元是否都屬於可見字元



方法	描述
<code>str.isspace()</code>	判斷字串是否為空格字元
<code>str.istitle()</code>	判斷字串是否適合當作標題
<code>str.isupper()</code>	判斷字串中所有字母字元是否都是大寫字母
<code>str.join(iterable)</code>	回傳將 <code>str</code> 連結 <code>iterable</code> 各元素的字串
<code>str.ljust(width[, fillchar])</code>	回傳將 <code>str</code> 在寬度 <code>width</code> 向左對齊的字串， <code>fillchar</code> 為填充字元，預設為空格
<code>str.lower()</code>	將 <code>str</code> 的英文字母都改成小寫
<code>str.lstrip([chars])</code>	回傳將 <code>str</code> 左邊具有 <code>chars</code> 字元去除的拷貝版本， <code>chars</code> 預設為空格符號
<code>static str.maketrans(x[, y[, z]])</code>	回傳 <code>x</code> 與 <code>y</code> 配對的 Unicode 編碼字典，若有提供 <code>z</code> ， <code>z</code> 中的字元會跟 <code>None</code> 配對
<code>str.partition(sep)</code>	以 <code>sep</code> 分割 <code>str</code> 為三個部份，結果回傳具有三個子字串的序對
<code>str.replace(old, new[, count])</code>	將 <code>str</code> 中的 <code>old</code> 子字串以 <code>new</code> 代換
<code>str.rfind(sub[, start[, end]])</code>	尋找最右邊的 <code>sub</code> ，也就是索引值最大的 <code>sub</code>
<code>str.rindex(sub[, start[, end]])</code>	尋找最右邊的 <code>sub</code> ，也就是索引值最大的 <code>sub</code>
<code>str.rjust(width[, fillchar])</code>	回傳將 <code>str</code> 在寬度 <code>width</code> 向右對齊的字串， <code>fillchar</code> 為填充字元，預設為空格



方法	描述
<code>str.rpartition(sep)</code>	以 <code>sep</code> 從最右端分割 <code>str</code> 為三個部份，結果回傳具有三個子字串的序對
<code>str.rsplit([sep[, maxsplit]])</code>	將 <code>str</code> 從最右端以 <code>sep</code> 分割成子字串，回傳儲存子字串的串列， <code>maxsplit</code> 為子字串最多的數量
<code>str.rstrip([chars])</code>	從 <code>str</code> 的最右端中移除 <code>chars</code> 字元，預設為空白字元
<code>str.split([sep[, maxsplit]])</code>	將 <code>str</code> 以 <code>sep</code> 分割成子字串，回傳儲存子字串的串列， <code>maxsplit</code> 為子字串最多的數量
<code>str.splitlines([keepends])</code>	將 <code>str</code> 以新行符號分割成子字串，回傳儲存子字串的串列
<code>str.startswith(prefix[, start[, end]])</code>	判斷 <code>str</code> 是否以 <code>prefix</code> 開頭
<code>str.strip([chars])</code>	從 <code>str</code> 中移除 <code>chars</code> 字元，預設為空白字元
<code>str.swapcase()</code>	將 <code>str</code> 中的英文字母進行大小寫轉換
<code>str.title()</code>	將 <code>str</code> 轉換成作為標題的字串
<code>str.translate(map)</code>	將 <code>str</code> 中的字元以 <code>map</code> 中配對的字元轉換
<code>str.upper()</code>	將 <code>str</code> 的英文字母都改成大寫
<code>str.zfill(width)</code>	回傳以 <code>0</code> 塞滿 <code>width</code> 的新字串
<code>str.rpartition(sep)</code>	以 <code>sep</code> 從最右端分割 <code>str</code> 為三個部份，結果回傳具有三個子字串的序對

