PROGRAMMING FOR BUSINESS COMPUTING 商管程式設計

Strings

Hsin-Min Lu
盧信銘
台大資管系

The String Data Type

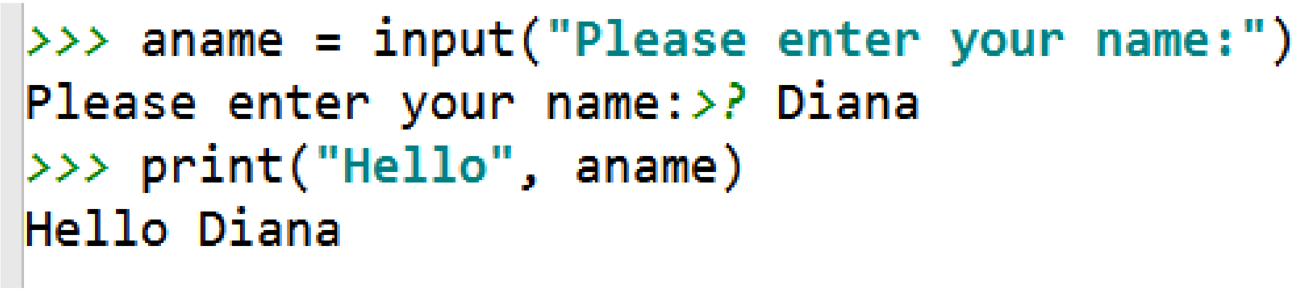
- Processing text data is an important task for PC users.
  - Think about the time you spent on using word processors such as MS words.
  - A large portion of online interactions are posting text messages.
- In Python, text is represented in by the *string* data type.
- A string is a sequence of characters enclosed within quotation marks (") or apostrophes ('). ☼

The String Data Type (Cont'd.)

- `>>> str1="Hello"`
- `>>> str2='ntu'`
- `>>> print(str1, str2)`
- `Hello ntu`
- `>>> type(str1)`
- `<class 'str'>`
- `>>> type(str2)`
- `<class 'str'>`
  ☼

The String Data Type (Cont'd.)

- We have encountered the input() function before.
- input() takes user input string and return it to the caller.

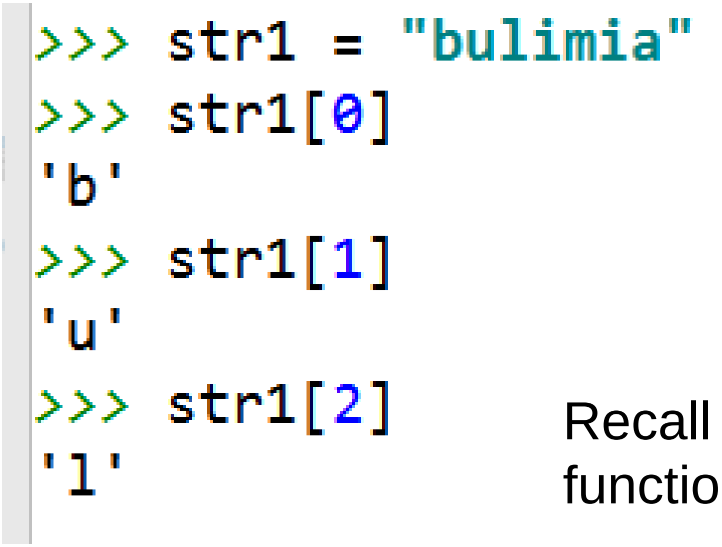


```
>>> aname = input("Please enter your name:")
Please enter your name:>? Diana
>>> print("Hello", aname)
Hello Diana
```

- A string is a sequence of characters.
- Access the individual characters in a string through *indexing*.
  - From left to right.
  - Starting from 0.

String Indexing

b	u	l	i	m	i	a
0	1	2	3	4	5	6



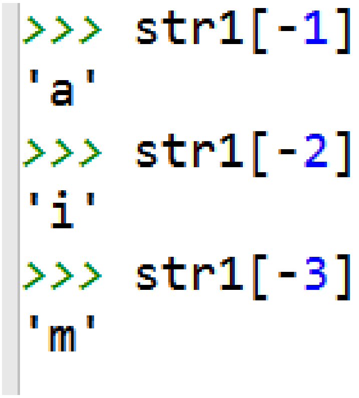
```
>>> str1 = "bulimia"
>>> str1[0]
'b'
>>> str1[1]
'u'
>>> str1[2]
'l'
```

Recall the way to invoke a function is function_name()

String Indexing (Cont'd.)

b	u	l	i	m	i	a
0	1	2	3	4	5	6

- In a string of *n* characters, the last character is at position *n-1*.
- Index from the right to left using negative indexes.



```
>>> str1[-1]
'a'
>>> str1[-2]
'i'
>>> str1[-3]
'm'
```

Slicing Strings

- Slicing: access a contiguous sequence of characters from a string.
- Syntax: `<string>[<start>:<end>]`
  - Both start and end are ints
- Beginning at position start and runs up to **but doesn’t include** the position end.

b	u	l	i	m	i	a
0	1	2	3	4	5	6

```
>>> str1[3:5]
'im'
>>> str1[2:6]
'limi'
>>> str1[2:8]
'limia'
>>> str1[2:10]
'limia'
>>> str1[2:]
'limia'
>>> str1[:5]
'bulim'
```

Some String Operations

- Can we put two strings together into a longer string?
- *Concatenation* “glues” two strings together (+).
- *Repetition* builds up a string by multiple concatenations of a string with itself (*).

```
>>> "spam" + "eggs"
'spameggs'
>>> "Spam" + "And" + "Eggs"
'SpamAndEggs'
>>> 3 * "spam"
'spamspamspam'
>>> "spam" * 5
'spamspamspamspamspam'
>>> (3 * "spam") + ("eggs" * 5)
'spamspamspameggseggseggseggseggs‘
>>> len("career")
6
```

The String Data Type

- The function *len* will return the length of a string.

```
a1="career"
print(len(a1))

for ch in a1:
    print("Get a character:", ch)
```

- Output:

```
6
Get a character: c
Get a character: a
Get a character: r
Get a character: e
Get a character: e
Get a character: r
```

String Operations

Operator	Meaning
+	Concatenation
*	Repetition
<string>[]	Indexing
<string>[:]	Slicing
len(<string>)	Length
for <var> in <string>	Iteration through characters

Strings, Lists, and Sequences

- Strings and lists are quite similar.
- Both are a special kind of *sequence.*
- There are some common operations that can be applied to both types.
- Some examples:

```
>>> [1,2] + [3,4]
[1, 2, 3, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'
>>> grades[2:4]
['C', 'D']
>>> len(grades)
5
```

Strings, Lists, and Sequences

- Strings are always sequences of characters, but *lists* can be sequences of arbitrary values.
- Lists can have numbers, strings, or both!

```
myList = [1, "Spam ", 4, "U"]
```

☼

Mutable and Immutable, Again

- Lists are *mutable*, ➔they can be changed.
- Strings can **not** be changed.

```
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = "p"

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in -toplevel-
    myString[2] = "p"
TypeError: object doesn't support item assignment
```

Example: Converting Date Format

- Two commonly used date format is **yyyymmdd** and **ddmmyy**.
    - yyyymmdd: 20141203, 19990212
    - ddmmyy: 03122014, 12021999

```
def ymd2dmy(dstr):
    """Convert date format from ymd to dmy
       E.g. 20150312 to 12032015"""
    y1 = dstr[0:4]
    m1 = dstr[4:6]
    d1 = dstr[6:8]
    return d1 + m1 + y1
```

Converting Date Format

```
def ymd2dmy(dstr):
    y1 = dstr[0:4]
    m1 = dstr[4:6]
    d1 = dstr[6:8]
    return d1 + m1 + y1
```

- Output:

```
>>> d1 = "20150512"
>>> d2 = ymd2dmy(d1)
>>> print("Converted date is", d2)
Converted date is 12052015
>>>
>>> d1 = "20171123"
>>> d2 = ymd2dmy(d1)
>>> print("Converted date is", d2)
Converted date is 23112017
```

Example: Validating Taiwan ID String

- Taiwan ID number of a string of length 10.
- First digit must be a upper case letter (between A to Z).
- Second digit must be either 1 or 2.
- The remaining digits are numbers.
- Example ID string: A123456789.

- Use a simple checksum rule to validate whether an ID is valid or not.
- According to this rule, A123456789 is valid, but A123456788 is not.
- We are going to see how to validate Taiwan ID. ☼

Length and the First Digit

- Use len() to check length

```
>>> str1="A123456789"
>>> len(str1)
10
```

- How to validate the first digit?
- As mentioned before, a string is a sequence of characters.
- Each character is stored using some sort of internal encoding.
- Traditional, English characters are stored using the ASCII system (American Standard Code for Information Interchange).

ASCII System

- 0 – 127 are used to represent the characters typically found on American keyboards.
  - 65 – 90 are “A” – “Z”
  - 97 – 122 are “a” – “z”
  - 48 – 57 are “0” – “9”
- The others are punctuation and *control codes* used to coordinate the sending and receiving of information. ☼

☼

Finding Internal Codes

- The *ord* function returns the numeric (ordinal) code of a single character.
- The *chr* function converts a numeric code to the corresponding character.

```
>>> ord("A")
65
>>> ord("a")
97
>>> chr(97)
'a'
>>> chr(65)
'A'
```

Checking the First Digit

- Note that the internal codes are arranged so that upper case letters are occupied in a continuous chunk of code range
- A → 65, B → 66, C → 67, …, Z → 90.
- We can use this characteristic to validate the first digit.
- The first internal encoding of the first digit need to be between 65 and 90.

Checking the First Digit

```
>>> idstr = "A123456789"
>>> code1 = ord(idstr[0])
>>> if (code1 < 65 or code1 > 90):
...     print("not valid")
... else:
...     print("valid")
...
valid
>>>
>>> idstr = "b123456789"
>>> code1 = ord(idstr[0])
>>> if (code1 < 65 or code1 > 90):
...     print("not valid")
... else:
...     print("valid")
...
not valid
```

Validation Rules for Taiwan ID

- 1. Map the first digit to a two-digit number.
  - E.g. A → 10, B → 11, C → 12, D → 13, .. Z → 33
  - Note: not in the order of A to Z.
- 2. Attach the two-digit number to the remaining 9-digit ID.
- 3. Compute a checksum by multiplying the digit at each position to a weight: [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
- 4. Sum over all results, divide the sum by 10 and compute the remainder.
- 5. If the remainder is 0, then it is valid. Otherwise, this is a invalid ID.

Mapping Table

A	10	H	17	O	35	V	29
B	11	I	34	P	23	W	32
C	12	J	18	Q	24	X	30
D	13	K	19	R	25	Y	31
E	14	L	20	S	26	Z	33
F	15	M	21	T	27		
G	16	N	22	U	28		

Example

- ID: A123456789
- Convert ‘A’ to ‘10’
- New ID: 10123456789
- Apply the weight: [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
- ➔ 1*1 + 0*9 + 1*8 + 2*7 + 3*6 + 4*5 + 5*4 + 6*3 + 7*2 + 8*1 + 9*1 = 130
- 130 / 10 = 13, remainder = 0
- ➔ Valid ID.

The Validation Process in Python

- Mapping the first letter to a two-digit number

```
>>> idstr="A123456789"
>>> code1 = ord(idstr[0])
>>> cmap = [10, 11, 12, 13, 14, 15, 16, 17, \
...         34, 18, 19, 20, 21, 22, 35, 23, 24, \
...         25, 26, 27, 28, 29, 32, 30, 31, 33]
>>> num1 = cmap[code1 - 65]
>>> newid = str(num1) + idstr[1:]
>>> print("newid=", newid)
newid= 10123456789
```

Mapping the First Digit

- cmap is a list that contains 26 elements
- The first element is for letter A, the second element is for letter B, and so on.

```
>>> code1 = ord(idstr[0])
```

- ➔ code1 is the ASCII code of the first digit

```
>>> num1 = cmap[code1 - 65]
```

- ➔ num1 is 0 for A, 1 for B, and so on
- `>>> newid = str(num1) + idstr[1:]`
- ➔ Concatenate the two-digit number with the remaining ID.

Compute the Checksum

```
>>> weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
>>> checksum = 0
>>> for i in range(0, 11):
...     checksum += weight[i] * int(newid[i])
...
>>> remainder = checksum % 10
>>> print("checksum=", checksum)
checksum= 130
>>> print("remainder=", remainder)
remainder= 0
```

Putting Everything Together

- Create a function that return True if the ID is valid, return False otherwise.

```
def verify_twid(idstr):
    """Verify Taiwan ID Number.
       Return True if valid; False otherwise"""
    #check length
    if len(idstr) != 10:
        return False
    #check first letter
    code1 = ord(idstr[0])
    if (code1 < 65 or code1 > 90):
        return False
    #check the remaining letters
    for i in range(1,10):
        code2 = ord(idstr[i])
        if (code2 < 48 or code2 > 57):
            return False
```

```
def verify_twid(idstr):
    #... Continue from previous slide …
    #check the second character
    code2 = ord(idstr[1])
    if (code2 < 49 or code2 > 50):
        return False
    #convert first English character to two-digit number.
    cmap = [10, 11, 12, 13, 14, 15, 16, 17, 34, 18, 19, 20,
21, 22, 35, 23, 24, 25, 26, 27, 28, 29, 32, 30, 31, 33]
    num1 = cmap[code1 - 65]
    newid = str(num1) + idstr[1:]
    weight = [1, 9, 8, 7, 6, 5, 4, 3, 2, 1, 1]
    checksum = 0
    for i in range(0, 11):
        checksum += weight[i] * int(newid[i])
    if checksum % 10 == 0:
        return True
    else:
        return False
```

verify_twid() in Action

```
>>> id1="A123456789"
>>> print(verify_twid(id1))
True
>>> verify_twid("B123456789")
False
>>> verify_twid("C999")
False
>>> verify_twid("123999")
False
>>> verify_twid("Z199999999")
False
>>> verify_twid("Z199999990")
True
```

我要Python講中文

- Python可以講中文
  - 真的嗎?
  - 真的!
- When the computer systems started to become popular in the 1960s, most systems used ASCII encoding.
- ASCII, however, cannot handle eastern languages
  - 中文、日文、韓文等
  - Why? A character is 8 bit long, can encode at most $2^8 - 1 = 255$ unique characters
  - 但繁體中文常用字有3,000以上!
  - 那怎麼辦?
  - ➔ How about use 2 characters to encode a Chinese character?
    - This will allow as to encode $2^{16} - 1 = 65535$ characters. ☼
    - Enough? I guess! ☼

我要Python講中文

- Double-byte (2 bytes = 16 bits) character sounds good.
- But there are a few complications.
  - 各家電腦廠商 (香港、台灣)各自有自家的編碼法，以至於檔案無法互相流通。
  - 大陸用簡體中文耶 (但是早期他們在鐵幕裡)
  - 日本有漢字，跟我們繁體中文有點像，又不是很一樣。
- 1983年資訊工業策進會為五大中文套裝軟體所設計中文共通內碼，稱為Big-5 (大五碼)
- 使用大五碼的軟體在市場上打下一片天地，Big5也成為中文編碼的業界標準。
- Big5為中文世界(台灣、香港)第一個廣為接受的編碼標準
  - 大陸則使用GB2312 ☼ ☼

我要Python講中文

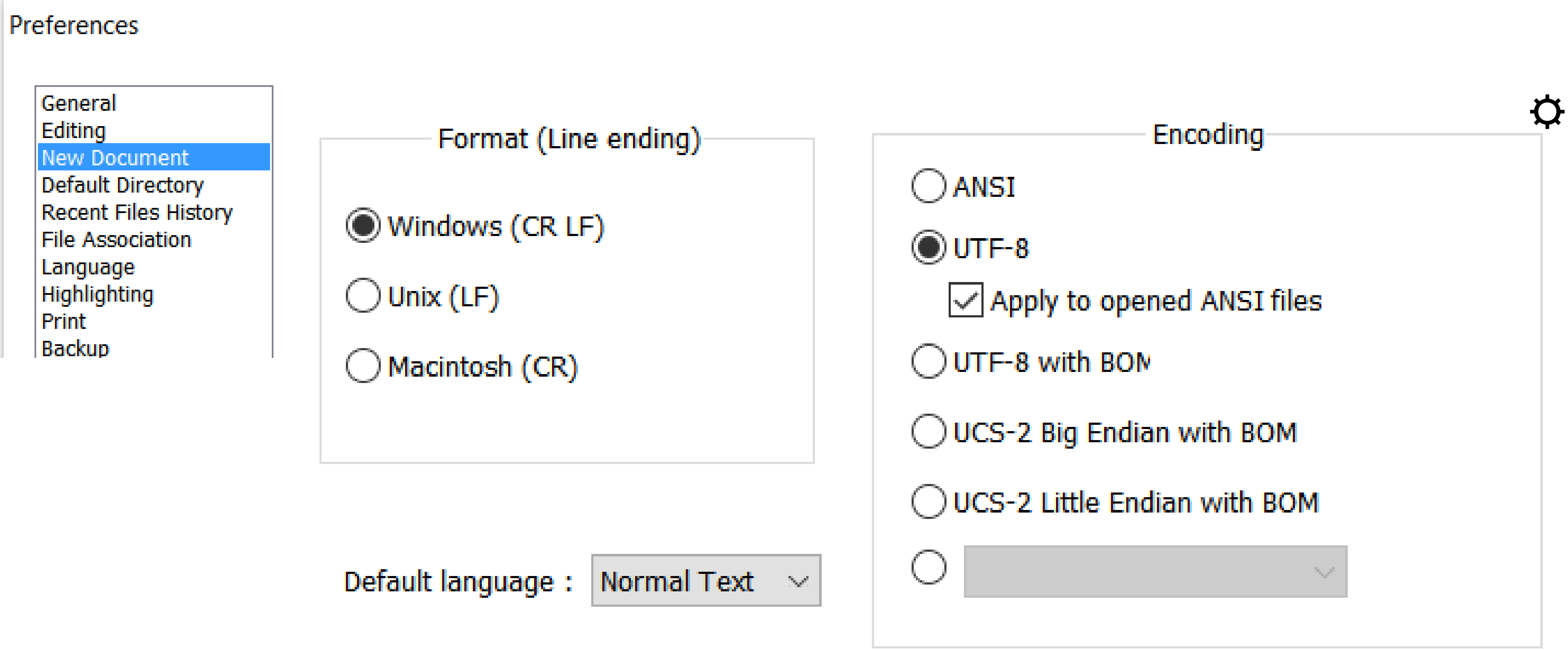
- 其實ASCII只說英文這件事在世界各地都是個問題。
- Unicode (一個非營利組織) 為了解決這個問題，開始發展世界統一的文字編碼。
- 1992年六月收錄20,902中日韓文字。
- 目前大部分的作業系統支援Unicode
  - Windows, Linux, Mac, Andriod, iPhone, etc.
- 常見的Unicode編碼方式有兩種
  - UTF-8 (Linux預設): one, two, or three bytes for a character.
  - UTF-16 (Microsoft Windows預設): one or two bytes for a character.
- You should use UTF-8 in most cases. ☼

Python Speaks Unicode

- Python string support Unicode.
- How to use Unicode (Chinese characters) in your Python scripts.
- 心法: 要告訴Python你的程式是什麼編碼
  - # -*- coding: utf8 -*-
  - (放在第一行，指定UTF8編碼)
  - 或是
  - #!/usr/bin/python
  - # -*- coding: utf8 -*-
  - (放在第二行)

Make Sure Your Text Editor Use UTF-8

- In Notepad++: Settings → Preferences → New Document
- Select UTF-8, and check “Apply to opened ANSI files”



Preferences

General
Editing
New Document
Default Directory
Recent Files History
File Association
Language
Highlighting
Print
Backup

Format (Line ending)

● Windows (CR LF)
○ Unix (LF)
○ Macintosh (CR)

Default language : Normal Text

Encoding

○ ANSI
● UTF-8
☑ Apply to opened ANSI files
○ UTF-8 with BOM
○ UCS-2 Big Endian with BOM
○ UCS-2 Little Endian with BOM
○

中文訊息

- Try the following simple python script.

```
# -*- coding: utf8 -*-

msg=u'中文測試'
print(msg)
```

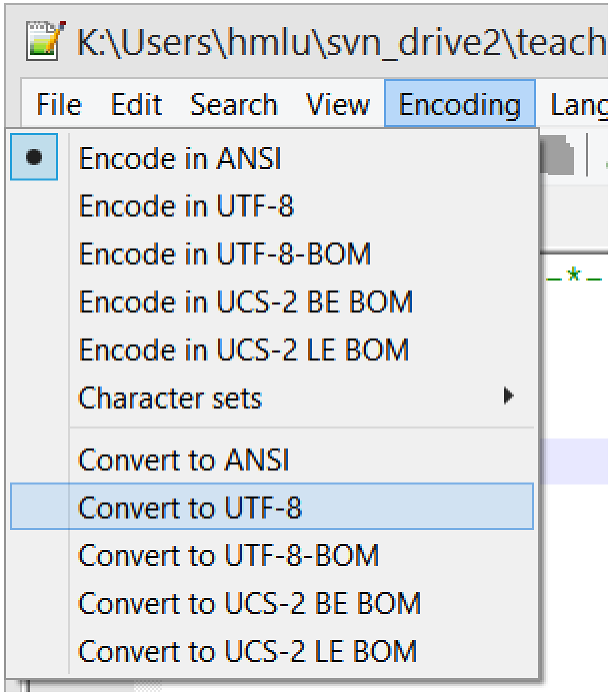
- If you see error message like this, you need to fix the encoding of your file:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "testcmsg1.py", line 2
SyntaxError: 'utf8' codec can't decode byte 0xa4 in position
0: invalid start byte
```

中文訊息

- 如果你的檔案不是UTF-8編碼…
- If you are using Notepad++, goto "Encoding" ➔ "Convert to UTF-8" ➔ save the file
- Try again! You will see:

中文測試



- Need to be very careful about your Chinese encoding ☼

中文訊息

```
msg=u'中文測試'
```

- 字串前面加u表示這個是個Unicode字串。叫Python用適當的解碼方式轉換成Unicode。
- Python Ver. 3.X 可以不用加u。但Python Ver. 2.X如果沒加，則需要後續作encoding處理。
- 看看這個例子:

```
# -*- coding: utf8 -*-
msg=u'中文測試'
print("msg=", msg)
print("len(msg)=", len(msg))
msg2='中文測試'
print("msg2=", msg2)
print("len(msg2)=", len(msg2))
```

Output:

```
msg= 中文測試
len(msg)= 4
msg2= 中文測試
len(msg2)= 4 ☼
```

chr and ord

- We are look at the internal encoding of characters

```
>>> chr(65)
'A'
>>> ord('A')
65
>>> ord('鴨')
40232
>>> chr(40232)
'鴨'
```

- chi(i) returns the character with internal encoding i
- ord(str) returns the internal encoding of str

Getting the internal code of a message

- Suppose you want to pass a secret message to you pal but you do not want other people to easily know what the message is.
- You can to convert the text into internal encoding

```
# -*- coding: utf8 -*-
msg='晚上七點水源星巴克見'
for achr in msg:
    print(ord(achr), end= " ")
print()
```

- Here is the output:

```
26202 19978 19971 40670 27700 28304 26143 24052 20811
35211
```

Getting the internal code of a message

- The code starts with a declaration on the encoding of the program.
- The for loop takes a character one time, and pass it to ord()
- Note the print line:

  ```
  print(ord(achr), end= " ")
  ```

  - What is the purpose of `end= " "`?☼

I want to know what this message is about

- Now you pal get this message, he or she wants to know what this is about
  - `26202 19978 19971 40670 27700 28304 26143 24052 20811 35211`
- Start with a string that contain the code, and split the string by space

```
code='26202 19978 19971 40670 27700 28304 26143 24052 20811 35211'
tmpcode = code.split(' ')
```

- Now the tmpcode contains a list of strings, each a code for a character

```
>>> tmpcode
['26202', '19978', '19971', '40670', '27700', '28304', '26143', '24052', '20811', '35211'] ☼
```

☼

I want to know what this message is about

- We can retrieve the code of each character using its index:

```
>>> tmpcode[0]
'26202'
>>> tmpcode[3]
'40670'
```

- Note that each element is a string.
- We want to use chr() to convert the code into message, one character a time.
- However, chr() takes int as input.
- We can convert string to int by the int() function.

```
>>> int(tmpcode[3])
40670                              ☼
```

I want to know what this message is about

- After getting a character, we need to concatenate them together.
- So we start with a empty unicode string

```
msg = ""
```

- Concatenate the first character to msg:

```
msg = msg + chr(int(tmpcode[0]))
```

- Another way to write this line:

```
msg += chr(int(tmpcode[0]))
```

☼

I want to know what this message is about

- Putting everything together

```
code='26202 19978 19971 40670 27700 28304 26143 24052 20811 35211'
tmpcode = code.split(' ')

msg = ""
for acode in tmpcode:
    msg += chr(int(acode))
print ("msg =", msg)
```

- The output is:

msg = 晚上七點水源星巴克見

☼

Common String Operations

- capitalize(): Capitalize the first character.
- title(): Capitalize the first character of each word.
- upper(): Convert all characters to uppercase.
- replace(old, new): Replace the occurrences of old with new.
- Examples:

```
>>> s = "athletes could not join the parade"
>>> print(s.capitalize())
Athletes could not join the parade
>>> print(s.title())
Athletes Could Not Join The Parade
>>> print(s.upper())
ATHLETES COULD NOT JOIN THE PARADE
>>> print(s.replace("athletes", "guests"))
guests could not join the parade
```

Common String Operations (Cont'd.)

- See Python 3 Document for a list of complete methods. (Section 4.7.1)
- https://docs.python.org/3/library/stdtypes.html#string-methods

```
>>> #count: Return the number of non-overlapping occurrences
>>> s2 = "media and mania"
>>> print(s2.count("ia"))
2
```

```
>>> #in operation
>>> uletter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
>>> 'A' in uletter
True
>>> 'z' in uletter
False
>>> 'AD' in uletter
False
>>> 'MN' in uletter
True
```

Common String Operations (Cont’d.)

```
>>> #find: Return the lowest index in the string where the given substring is found
>>> s3 = "02-33661184"
>>> s3.find('-')
2
>>>
>>> #is numeric characters
>>> s4 = "1235"
>>> s4.isnumeric()
True
>>> s5 = "1235.2"
>>> s5.isnumeric()
False
```

Common String Operations (Cont'd.)

```
>>> #is upper characters
>>> s6 = "HI"
>>> s6.isupper()
True
>>> s7 = "Hi"
>>> s7.isupper()
False
>>>
>>> #split a string by a given separator string.
>>> s8 = "Not a useful tool."
>>> print(s8.split(" "))
['Not', 'a', 'useful', 'tool.']
>>>
```

Common String Operations (Cont'd.)

```
>>> #remove extra spaces
>>> s9 = " many spalce   "
>>> print(s9.strip())
many spalce
>>>
>>> #remove given characters.
>>> 'www.example.com'.strip('cmowz.')
'example'
```

Formatting Strings

- We often need to provide output in a specific format.
- Give “pretty print”
- For example, output gasoline price using a specific format ($23.4).
- Output stock price with two decimal places (e.g., 32.12).
- Add extra “0” upfront (e.g. ID: 000325).
- Generating reports following a specific format:
  - `Name: Joe Smith               Phone: 02-12345543`
  - `First Contact: 2006-12-32     Age: 40`

String Formatting

- Consider this example: We have a variable that store the price of a product, and we want to output the price with only two decimal places:

```
>>> prc=13.87623
>>> print("Current price: %0.2f" % prc)
Current price: 13.88
```

- For numbers, % means the remainder operation.
- For strings, % is a string formatting operator. ☼

☼

String Formatting

- The formatting specifier has the form: %<width>.<precision><type-char>
- Type-char can be **d**ecimal, **f**loat, **s**tring (decimal is base-10 ints)
- <width> and <precision> are optional.
- <width> tells us how many spaces to use to display the value. 0 means to use as much space as necessary.

```
>>> prc=13.87623
>>> print("Current price: %0.2f" % prc)
Current price: 13.88
```

String Formatting

- If the given <width> is not enough, Python will expand the space until the result fits.
- <precision>: number of places to display after the decimal (for floating point numbers only).

- %0.2f: use as much space as necessary and two decimal places to display a floating point number. ☼

String Formatting

```
>>> "%s同學您好, 您借的書已逾期%d天，請盡速歸還。" % ("王大雄", 55)
'王大雄同學您好, 您借的書已逾期55天，請盡速歸還。'
>>> '整數：%5d[欄位長度為5]' % 7
'整數：    7[欄位長度為5]'
>>> '整數：%10d[欄位長度為10]' % 99
'整數：        99[欄位長度為10]'
>>> '浮點數：%10.5f[欄位長度為10，五位小數點' % 3.1415926
'浮點數：   3.14159[欄位長度為10，五位小數點'
>>> '浮點數：%0.5f[欄位長度為0，五位小數點' % 3.1415926
'浮點數：3.14159[欄位長度為0，五位小數點'
>>> '比較兩個格式：%f 與 %0.20f' % (3.14, 3.14)
'比較兩個格式：3.140000 與 3.14000000000000012434' ☼
```

String Formatting

- Output values are right-justified by default (if the width is wider than needed)
- To left-justify use a negative width (e.g., %-10.3f)
- You may see random digits if showing a float with long decimal places. This is caused by internal representation for float.
☼

Concatenate Strings and Floats

- You can use “+” to concatenate strings.
- Be very careful if you are concatenate string and other data types (e.g. float).

```
>>> value = 3.14
>>> print ("The value is" + value + ".")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: must be str, not float
```

If value is an int or float, Python thinks the + is a mathematical operation, not concatenation, and “.” is not a number! ☼

THANK YOU!

Questions?