Python: String (字串)

Cheng-Te Li (李政德)

chengte@mail.ncku.edu.tw

Dept. of Statistics, NCKU

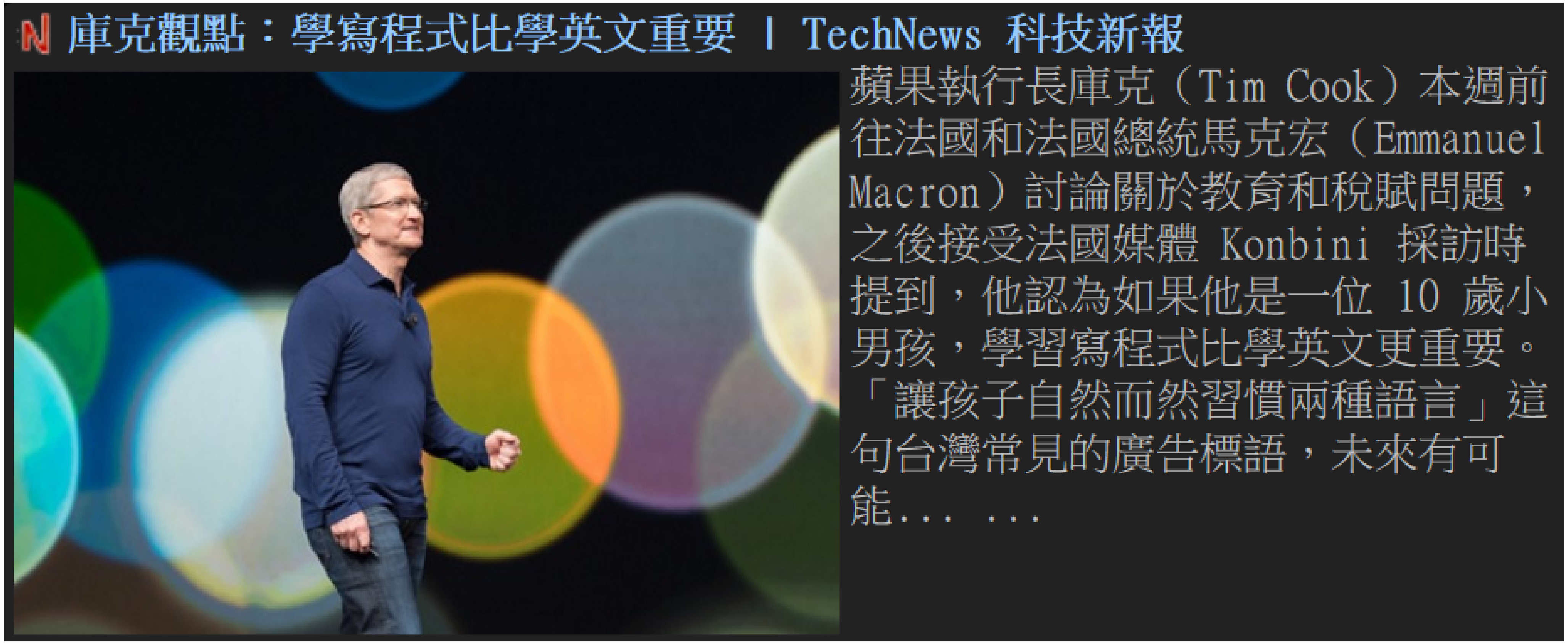
Course Info (0307)

- 0307 will be a pre-recorded video, put at Moodle by 3/7
- Homework 2 (HW2) is announced next week
  - You have 3 weeks for HW2
  - Cover Lecture 2, 3, 4 (string, conditionals, and while loop)
  - Do HW1 as soon as possible
- Concurrent teaching
  - Lecture 2: Strings
  - Lecture 3: Conditionals (if-else)
  - Lecture 4: while loops
- Programming Exam 1 (PE1) will be at **3/12** (Tue)
  - PE1 covers Lecture 1 and HW1

Course Info: Programming Exams

- 3/12 (Tuesday): Programming Exam 1 (第1次上機程式考試)
- For **every student**
- Open book, slides, and paper-based materials
- NO Internet (except for submission)
  - No smart phones
  - No ChatGPT
  - No LINE/messangers etc.
- You can use either PC here or your own laptop
- Time: **08:10-09:00am**
  - Submit your code whenever you finish the exam
- 更多詳細考試規定寫在moodle對應區塊

學程式比學英文重要?



庫克觀點：學寫程式比學英文重要 | TechNews 科技新報

蘋果執行長庫克（Tim Cook）本週前往法國和法國總統馬克宏（Emmanuel Macron）討論關於教育和稅賦問題，之後接受法國媒體 Konbini 採訪時提到，他認為如果他是一位 10 歲小男孩，學習寫程式比學英文更重要。「讓孩子自然而然習慣兩種語言」這句台灣常見的廣告標語，未來有可能... ...

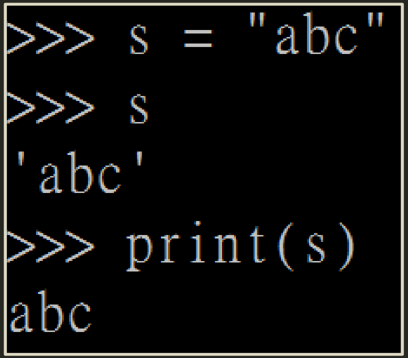
http://technews.tw/2017/10/11/coding-is-important-than-english/

https://9to5mac.com/2017/10/10/tim-cook-interview-video/

String (字串)

- A programmer works more on strings than numbers
- A string is a sequence of characters (字元)
- The characters in strings are enclosed by single quotes (') or double quotes (")
  - "hello", 'happy birthday'
- You can have single quotes inside double-quoted strings, or double quotes inside single-quoted strings

```
"she said: 'I like Python'"
'the result of "1+2" is "3"'
s = "hello" # the result of interpreter includes quotes
print(s)    # the result of print() removes quotes
            # print() for human beings to see
```



```
>>> s = "abc"
>>> s
'abc'
>>> print(s)
abc
```

Empty Strings (空字串)

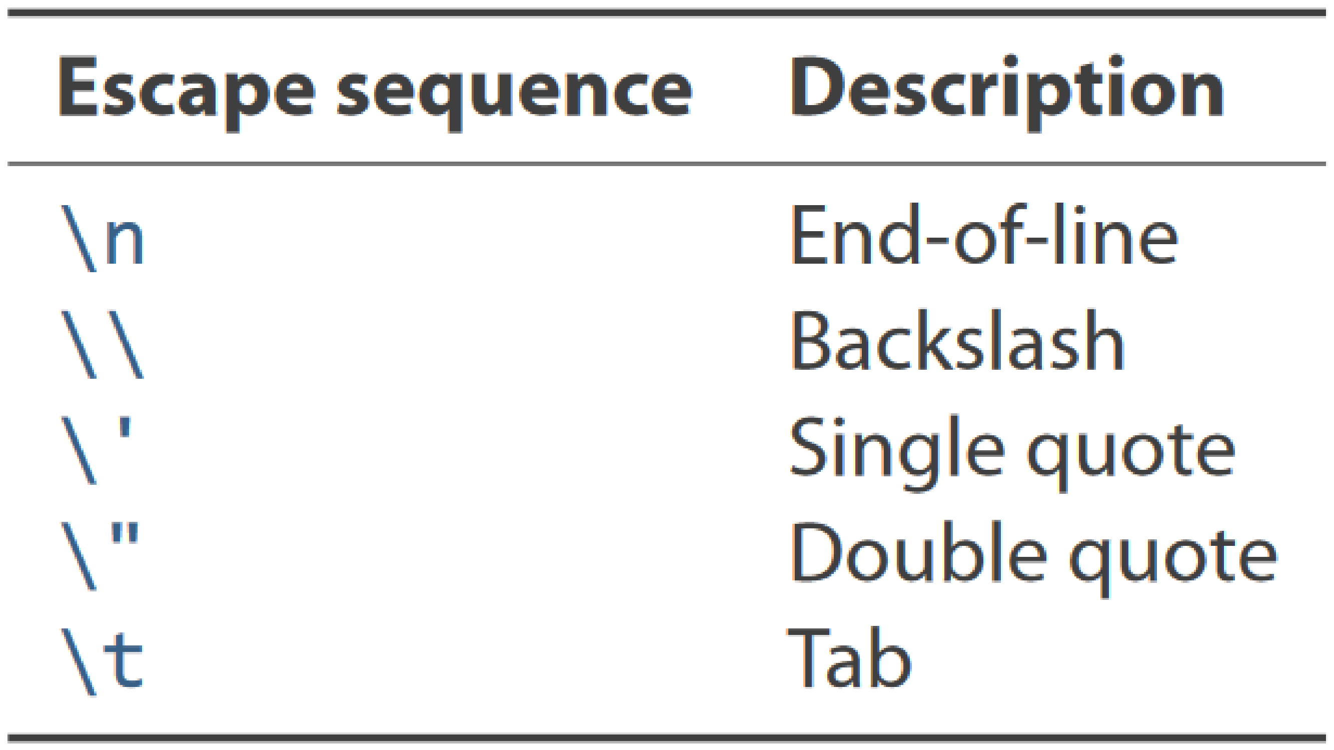
- The empty string has no characters at all but is valid
- When need an empty string?
  - Sometimes you need to build a string from other strings

```
''  # empty string
""  # empty string
s = ""
s += "The year of today is: "
year = 2017
s += str(year)  # convert to string type
print(s)        # The year of today is: 2017
```

```
>>> str(98.6)
'98.6'
>>> str(1.0e4)
'10000.0'
>>> str(True)
'True'
```

Escape with \ (用\來轉義)

- By preceding a character with a backslash (\), you can escape the meaning of character within strings to have special effects/meanings

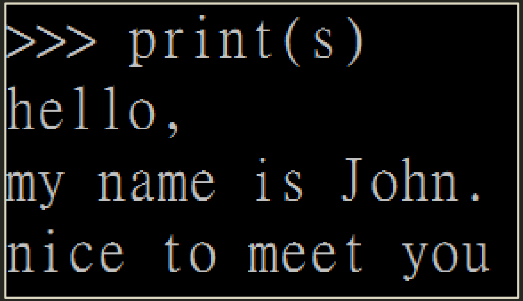


Escape sequence	Description
`\n`	End-of-line
`\\`	Backslash
`\'`	Single quote
`\"`	Double quote
`\t`	Tab

換行

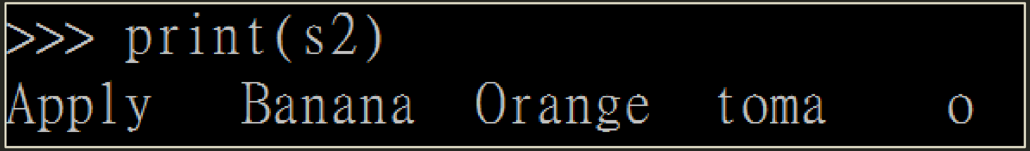
Escape Sequences

```python
s = "hello,\nmy name is John.\nnice to meet you"
```



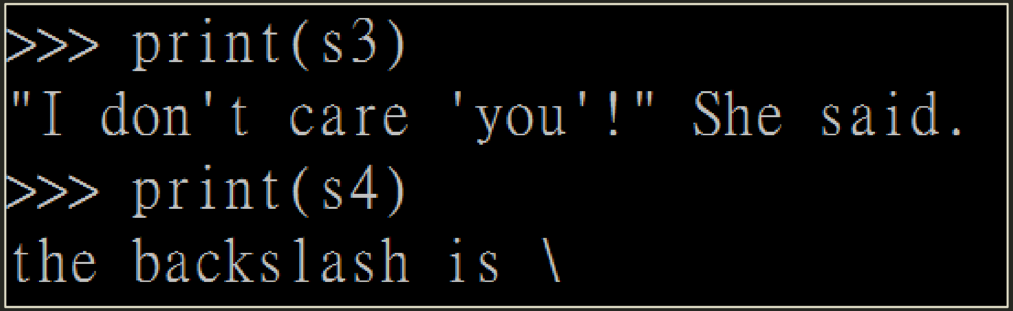
```
>>> print(s)
hello,
my name is John.
nice to meet you
```

```python
s2 = "Apply\tBanana\tOrange\ttoma\to"
```



```
>>> print(s2)
Apply   Banana  Orange  toma    o
```

```python
s3 = "\"I don't care \'you\'!\" She said."
s4 = "the backslash is \\"
```



```
>>> print(s3)
"I don't care 'you'!" She said.
>>> print(s4)
the backslash is \
```

Concatenation (串接)

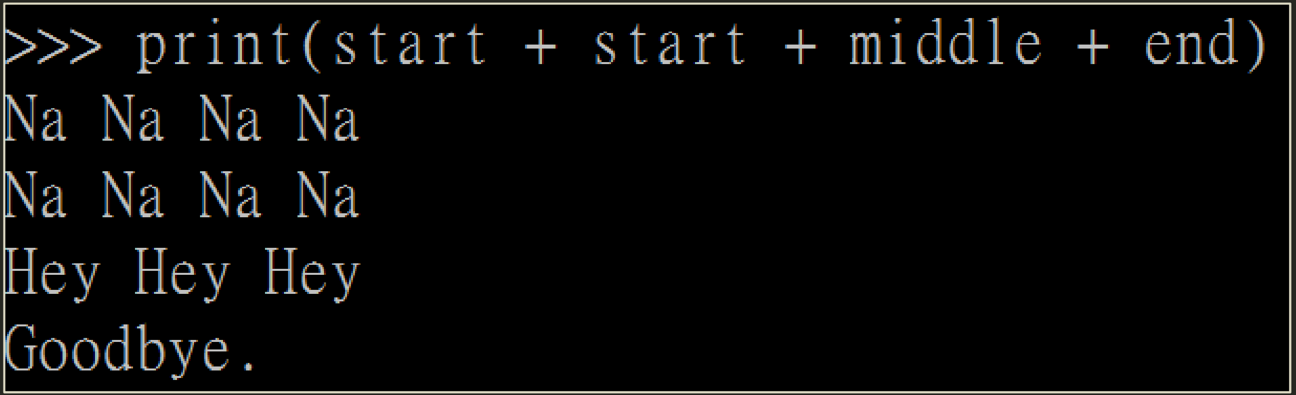
- When the + operator is applied to strings, it means "concatenation"

```
a = "hello"
b = a + "there"
print(b)                    # hellothere
b += "!"
print(b)                    # hellothere!
c = a + " " + "there!"
print(c)                    # hello there!
```

Duplication (複製)

- When the * operator is applied to strings, it means “duplication”

```
start = 'Na ' * 4 + '\n'
middle = 'Hey ' * 3 + '\n'
end = 'Goodbye.'
print(start + start + middle + end)
```



```
>>> print(start + start + middle + end)
Na Na Na Na
Na Na Na Na
Hey Hey Hey
Goodbye.
```

Contain? (包含)

- The **in** operator is used to check if another string is contained in a string container
  - Returns **True** if the element appears in the container, **False** otherwise

```
vowels = "aeiou"
"a" in vowels            # True
"k" in vowels            # False
"aiu" in vowels          # False
"io" in vowels           # True
not "aiu" in vowels      # True
```

Look into Strings

- We can get at any single character in a string using an index specified in square brackets [index]
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed

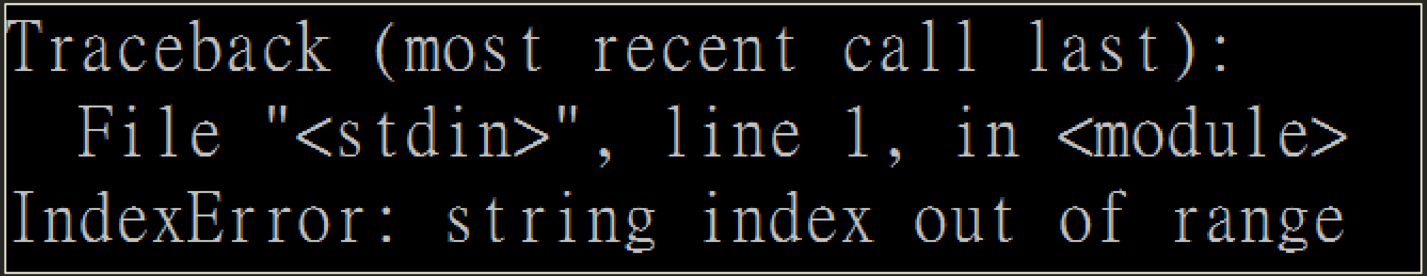
	b	a	n	a	n	a
index	0	1	2	3	4	5

```
fruit = 'banana'
letter = fruit[5]
print(letter)        # a
letter = fruit[0]
print(letter)        # b
i = 3
w = fruit[i-1]
print(w)             # n
```

A Character Too Far

- You will get a Python error if you attempt to index beyond the end of a string
- So be careful when constructing index values and slices

```
s = "banana"
print(s[10])  # what will happen?
```



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Slicing Strings

- You can extract a substring (a subsequence of a string) from a string using the **slice** technique

Slice with `[start:end:step]`

[:] extracts the entire sequence from start to end

[*start* :] specifies from the *start* offset to the end

[: *end*] specifies from the beginning to the *end* offset minus 1

[*start* : *end*] indicates from the *start* offset to the *end* offset minus 1

[*start* : *end* : *step*] extracts from the *start* offset to the *end* offset minus 1, skipping characters by *step*

Slicing Strings

	i		l	o	v	e		b	a	n	a	n	a
index	0	1	2	3	4	5	6	7	8	9	10	11	12

```
line = "i love banana"
print(line[4])          # v
print(line[:])          # i love banana
print(line[2:5])        # lov
print(line[7:12])       # banan
print(line[2:])         # love banana
print(line[:6])         # i love
print(line[2:10:2])     # lv a
print(line[1::3])       #  vba
print(line[::4])        # ivaa
```

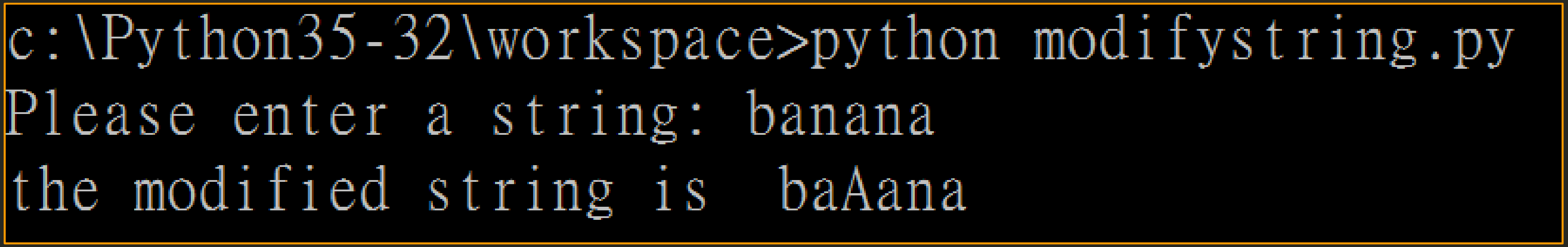
Slicing Strings

	i		l	o	v	e		b	a	n	a	n	a	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
line = "i love banana!"
print(line[-1])          # !
print(line[:-7])         # i love
print(line[-4:-10])      #
print(line[-4:])         # ana!
print(line[-4:3])        #
print(line[3:-4])        # ove ban
print(line[-14])         # i
print(line[::-1])        # !ananab evol i
print(line[-4:-10:-1])   # anab e
print(line[-4:3:-1])     # anab ev
print(line[-2:-12:-2])   # aaa v
```

In-class Exercise: Modify a String

- Write a program (modifystring.py) that:
  - Gets a string from the user
  - Modifies that string so that position 3 is an A
  - Prints the modified string



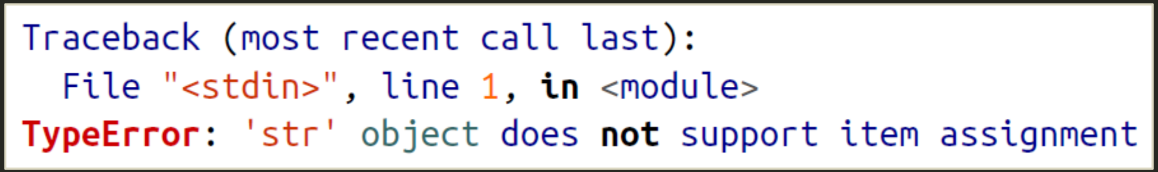
```
c:\Python35-32\workspace>python modifystring.py
Please enter a string: banana
the modified string is  baAana
```

```python
my_string = input("please enter a string: ")
my_string = my_string[0:2] + "A" + my_string[3:]
print("the modified string is ", my_string)
```

String is Immutable (不可變的)

- You cannot insert a character directly into one or change the character at a specific index
- But you can use some string functions such as `replace()` or the **slice** technique to change it

```
name = "Henny"
name[0] = "P"               # what will happen?
```



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
name.replace("H", "P")  # Penny (replace “H” with “P”)
print(name)             # Henny
name2 = name.replace("H", "P")
print(name2)            # Penny
name3 = "P" + name[1:]  # Penny (using the slice technique)
print(name3)            # Penny
```

Built-in Functions for Strings

```
string.function(arguments)
```

- `len()`
- `startswith()`
- `endswith()`
- `find()`
- `rfind()`
- `count()`
- `isalnum()`
- `split()`
- `join()`
- `strip()`
- `capitalize()`
- `title()`
- `upper()`
- `lower()`
- `replace()`

Built-in Functions for Strings

- **`len()`**: counts characters in a string
- `startswith()`: check if a string **starts** with another string
- `endswith()`: check if a string **ends** with another string

```
question = "Blue Tuesday!? "
len(question)                     # 15
empty = ""
len(empty)                        # 0
question.startswith("Blue")       # True
question.startswith("Blur")       # False
question.endswith("Monday")       # False
question.endswith("Tuesday!?")    # False
question.endswith("Tuesday!? ")   # True
```

Built-in Functions for Strings

- **find()**: find the offset of the **first** occurrence of another string
- rfind(): find the offset of the **last** occurrence of another string
- **count()**: count the frequency of another string in the string
- isalnum(): are all of the characters in the string letters or numbers?

```
question = "Is today Monday or Tuesday?"
day1 = "Monday"
day2 = "Tuesday"
question.find(day1)          # 9
question.find(day2)          # 19
question.find("Wednesday") # -1
question.rfind("day")        # 23
question.count("day")        # 3
question.count("month")      # 0
question.isalnum()           # False
```

If the substring is not found, find() returns **-1**

Remember that string position starts at zero

Built-in Functions for Strings

- The `replace()` function is like a "search and replace" operation in a word processor
- `replace()`: **Replace** all occurrences of the search string with the replacement string

```
greet = "Hello Bob"
new_greet = greet.replace("Bob", "Jane")
print(new_greet)                             # Hello Jane
new_greet = greet.replace("o", "X")
print(new_greet)                             # HellX BXb
new_greet = greet.replace(" ", "\t")
print(new_greet)                             # Hello   Bob
new_greet = greet.replace("G", "K")
print(new_greet)                             # Hello Bob
```

Built-in Functions for Strings

- **split()** : **Break** a string into a list of smaller strings based on some separator

```
line = "Hillary Clinton clarified her misleading statements."
line2 = line.split(" ")
print(line2)
# ['Hillary', 'Clinton', 'clarified', 'her', 'misleading',
'statements.']
myDate = "2016-09-27"
myDate.split("-")    # ['2016', '09', '27']
myDate.split(0)      # error
myDate.split("0")    # ['2', '16-', '9-27']
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Built-in Functions for Strings

- `join()`: Collapse a list of strings into a single string
- `strip()`: Remove a given string from both ends of a string
- `lstrip()` and `rstrip()`: Remove whitespace at the left or right

```
presidents = ['Bush', 'Clinton', 'Bush', 'Obama']
"->".join(presidents)              # 'Bush->Clinton->Bush->Obama'
line = "... **I feel so sorry ~~~ ..."
line = line.strip(".")
print("\'" + line + "\'")          # ' **I feel so sorry ~~~ '
line = line.strip()
print("\'" + line + "\'")          # '**I feel so sorry ~~~'
line = line.lstrip("*")
print("\'" + line + "\'")          # 'I feel so sorry ~~~'
line = line.rstrip("~")
print("\'" + line + "\'")          # 'I feel so sorry '
```

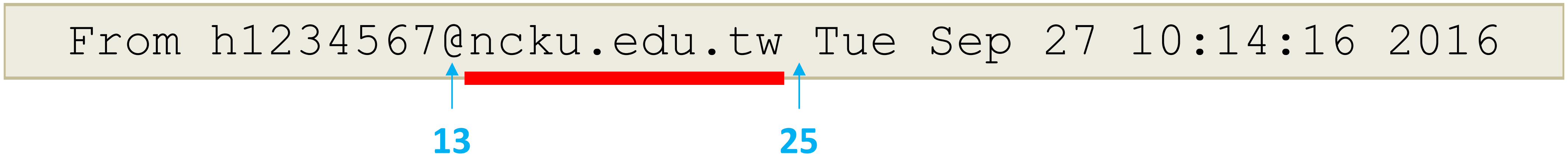
Built-in Functions for Strings

- `capitalize()`: Capitalize the first word
- `title()`: Capitalize all the words
- `upper()`: Convert all characters to uppercase
- `lower()`: Convert all characters to lowercase

```python
line = "i feel so sorry..."
line2 = line.capitalize()
print(line2)                        # 'I feel so sorry...'
line3 = line.title()
print(line3)                        # 'I Feel So Sorry...'
line4 = line.upper()
print(line4)                        # 'I FEEL SO SORRY...'
line5 = line.lower()
print(line5)                        # 'i feel so sorry...'
```

Example: Extract a Substring

- Write a program that parses and extracts the host name of a email message



```
From h1234567@ncku.edu.tw Tue Sep 27 10:14:16 2016
```

```python
data = 'From h1234567@ncku.edu.tw Tue Sep 27 10:14:16 2016'
atpos = data.find('@')
print(atpos)                    # 13
data = data[atpos:]             # '@ncku.edu.tw Tue Sep 27 10:14:16 2016'
sppos = data.find(' ')
print(sppos)                    # 12
host = data[1: sppos]
print(host)                     # ncku.edu.tw
```

In-class Exercise: Modify a Message

- Write a program (modifystring2.py) that modifies a email message using `find()` and slicing

```
From h1234567@ncku.edu.tw Tue Sep 27 10:14:16 2016
```

↓

```
Tue Sep 27 10:14:16 2016 From h7654321@ncku.edu.tw
```

```python
data = 'From h1234567@ncku.edu.tw Tue Sep 27 10:14:16 2016'
pos1 = data.find('h')
pos2 = data.find('@')
data = data[:pos1+1] + "7654321" + data[pos2:]
pos3 = data.find("Tue")
modified_data = data[pos3:] + " " + data[:pos3]
print(modified_data)
```

Formatting String

- One of Python’s coolest features is the string format operator **%**
  - The left of the **%**: place a string (the format string)
  - The right of the **%**: place the values you want to format

```
"FORMAT String" % (value1, value2, ...)
```

```
print("My name is %s and height is %d cm!" % ("John", 170))
# My name is John and height is 170 cm!
format = "My name is %s and height is %d cm!"
values = ("John", 170)
print(format % values)
```

Formatting String

```
print("My name is %s " % ('Zara'))
# %s : string  --> My name is Zara
print("My weight is %d kg and my height is %d cm!" % (61, 165))
# %d : decimal integer  --> My weight is 21 kg!
print("Now is %d degree, my name has %d characters" % (-21, len("Zara")))
# %c : character  --> Now is -21 degree, My name has 4 characters
print("The temperature is %f degrees precisely" % (-21.34))
# %f : floating point real number
# --> The temperature is -21.340000 degrees precisely
print("My BMI is %f " % ((50/(170/100)**2)))
# you also can use expression as the value --> My BMI is 17.301038
print("%d big numbers: %e %e" % (2, 123456789, 999999999999999999999))
# %e : exponential notation (with lowercase 'e')
# --> big number 1.234568e+08 1.000000e+21
print("another big number %E" % (987654321))
# %E : exponential notation (with UPPERcase 'E')
# --> another big number 9.876543E+08
```

Type Codes for Formatting

Format Symbol	Meaning
%s	String
%c	Character
%d	Decimal Integer
%f	Decimal Floating point number
%e	Exponential Floating point number (“e”)
%E	Exponential Floating point number (“E”)
%g	Decimal or exponential Floating point

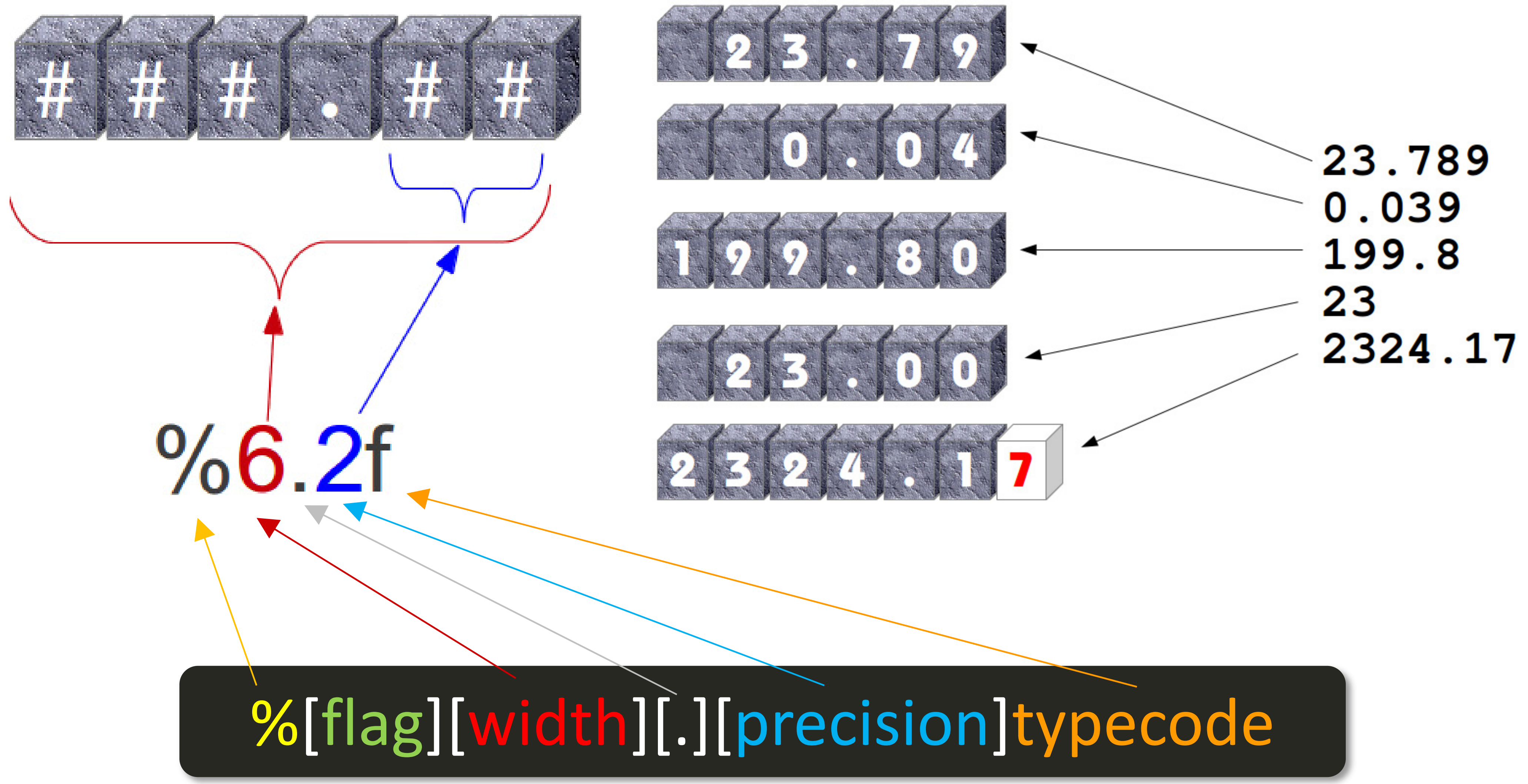
%g chooses formats by number content (it’s formally defined to use exponential format %e if the exponent is less than −4 or not less than precision, and decimal format %f otherwise, with a default total digits precision of 6)

The General Structure of Formatting

```
%[flag][width][.][precision]typecode
```

- **Flag**: specify things like left justification (−), numeric sign (+), a blank before positive numbers, and zero fills (0)
- **Width**: give a total minimum field width for the text
- **Precision**:
  Set the number of digits (precision) to display after a decimal point for floating point numbers
- **Typecode**: d, s, f, e, g

Formatting String



%[flag][width][.][precision]typecode

Formatting String

```
x = 1234
myformat = "integers: %d | %-6d | %06d"
print(myformat % (x, x, x))
% integers: 1234 | 1234   | 001234

x, y = 1.23456789, 123456789
print(x)
myformat = "output: %e | %E | %f | %g"
print(myformat % (x, x, x, x))
% output: 1.234568e+00 | 1.234568E+00 | 1.234568 | 1.23457
print(myformat % (y, y, y, y))
% output: 1.234568e+08 | 1.234568E+08 | 123456789.000000 | 1.23457e+08

myformat = "output: %-6.2f | %05.2f | %+06.1f"
print(myformat % (x, x, x))
% output: 1.23   | 01.23 | +001.2
```

Python Formatting Code

Strings

We consider the string “Hello, world!”.

Formatting code	**Hello, world!**
%s	'Hello, world'
%20s	' Hello, world!'
%-20s	'Hello, world! '
%3s	'Hello, world'

Integers

We consider the integers 12,345 and -12,345.

Formatting code	**12,345**	**-12,345**
%d	'12345'	'-12345'
%20d	' 12345'	' -12345'
%-20d	'12345 '	'-12345 '
%020d	'00000000000000012345'	'-0000000000000012345'
%+d	'+12345'	'-12345'
%+20d	' +12345'	' -12345'
%+-20d	'+12345 '	'-12345 '
%+020d	'+0000000000000012345'	'-0000000000000012345'
%3d	'12345'	'-12345'

Floating point numbers

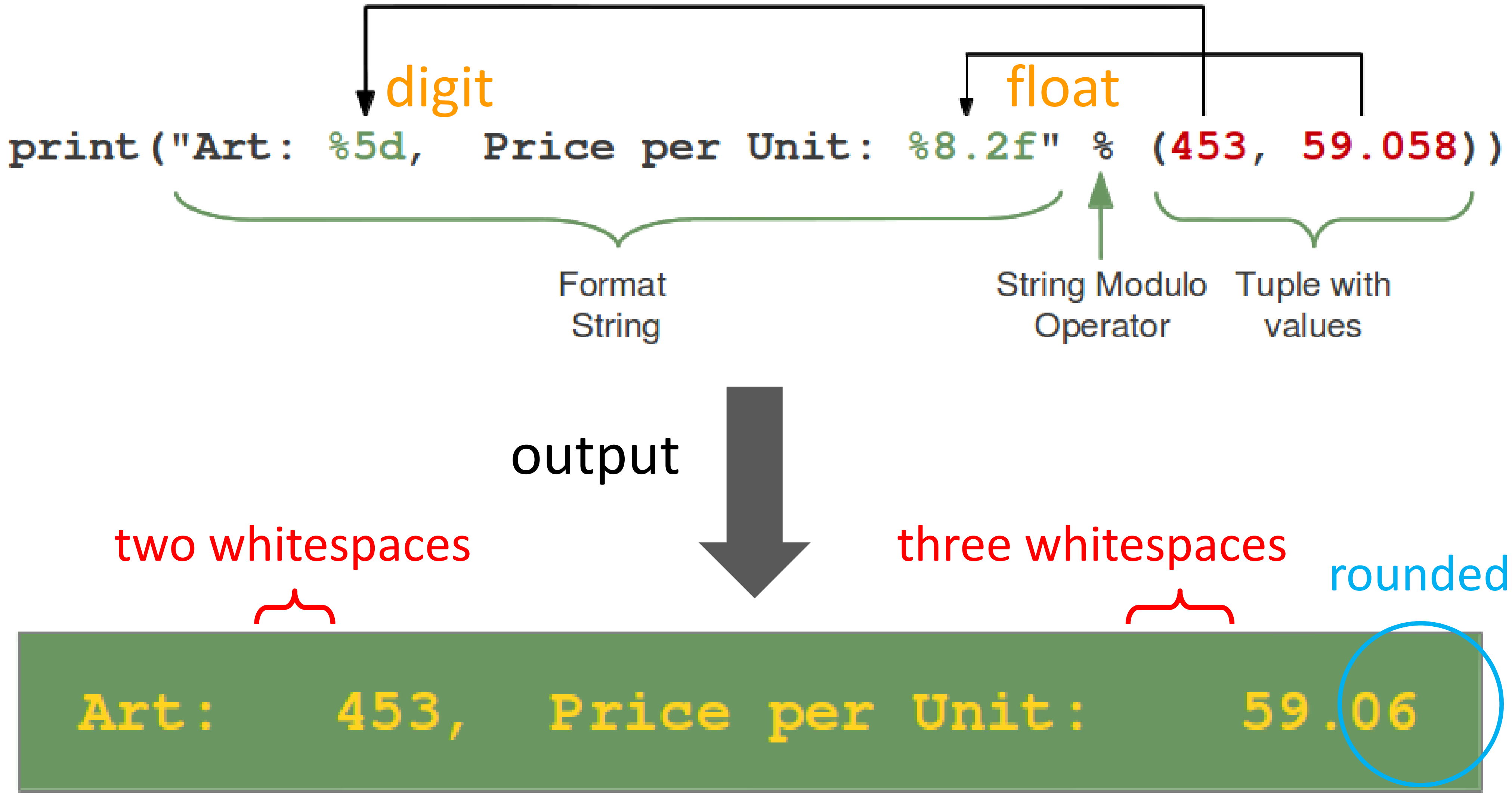
We consider the floating point numbers 12·34567 and -12·34.

The `%f` formatting code presents data in decimal notation. The %e code does it in exponential form.

Formatting code	**12·34567**	**-12·34**
%f	'12.345670'	'-12.340000'
%20f	' 12.345670'	' -12.340000'
%-20f	'12.345670 '	'-12.340000 '
%020f	'0000000000012.345670'	'-000000000012.340000'
%+f	'+12.345670'	'-12.340000'
%+20f	' +12.345670'	' -12.340000'
%+-20f	'+12.345670 '	'-12.340000 '
%+020f	'+000000000012.345670'	'-000000000012.340000'
%.4f	'12.3457'	'-12.3400'
%20.4f	' 12.3457'	' -12.3400'
%-20.4f	'12.3457 '	'-12.3400 '
%020.4f	'0000000000000012.3457'	'-000000000000012.3400'
%+.4f	'+12.3457'	'-12.3400'
%+20.4f	' +12.3457'	' -12.3400'
%+-20.4f	'+12.3457 '	'-12.3400 '
%+020.4f	'+000000000000012.3457'	'-000000000000012.3400'

Formatting code	**12·34567**	**-12·34**
%e	'1.234567e+01'	'-1.234000e+01'
%20e	' 1.234567e+01'	' -1.234000e+01'
%-20e	'1.234567e+01 '	'-1.234000e+01 '
%020e	'000000001.234567e+01'	'-00000001.234000e+01'
%+e	'+1.234567e+01'	'-1.234000e+01'
%+20e	' +1.234567e+01'	' -1.234000e+01'
%+-20e	'+1.234567e+01 '	'-1.234000e+01 '
%+020e	'+00000001.234567e+01'	'-00000001.234000e+01'
%.4e	'1.2346e+01'	'-1.2340e+01'
%20.4e	' 1.2346e+01'	' -1.2340e+01'
%-20.4e	'1.2346e+01 '	'-1.2340e+01 '
%020.4e	'00000000001.2346e+01'	'-0000000001.2340e+01'
%+.4e	'+1.2346e+01'	'-1.2340e+01'
%+20.4e	' +1.2346e+01'	' -1.2340e+01'
%+-20.4e	'+1.2346e+01 '	'-1.2340e+01 '
%+020.4e	'+0000000001.2346e+01'	'-0000000001.2340e+01'

Formatting String



```
print("Art: %5d,  Price per Unit: %8.2f" % (453, 59.058))
```

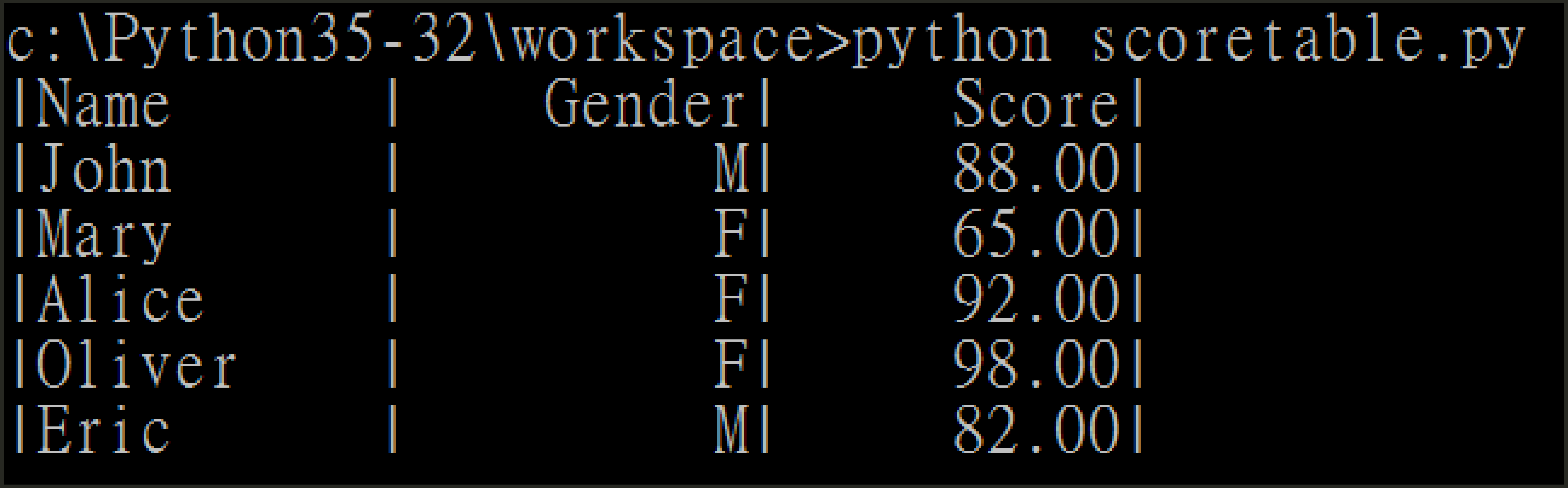
```
Art:   453,  Price per Unit:    59.06
```

Formatted Printing

```
value = 356.08977
s1 = "Price: $ %8.2f" % (value)
print(s1)                      # Price: $   356.09
a, b = 5, 3.1415
s2 = "%03d + %4.2f = %5.2f" % (a, b, a+b)
print(s2)                      # 005 + 3.14 =  8.14
print(a, "+", b, "=", a+b)     # 5 + 3.1415 = 8.1415
myformat = "%5d * %4.2f = %-5.2f \n%5d * %4.2f \t= %5.4f"
print(myformat % (a, b, a*b, 2*a, 2*b, 2*a*b))
%    5 * 3.14 = 15.71
%   10 * 6.28    = 31.4150
myformat = "|%-10s|%10s|%10d|"
print(myformat % ("Johnson", "Math", 90))
print(myformat % ("Tom", "English", 75))
% |Johnson   |      Math|        90|
% |Tom       |   English|        75|
```

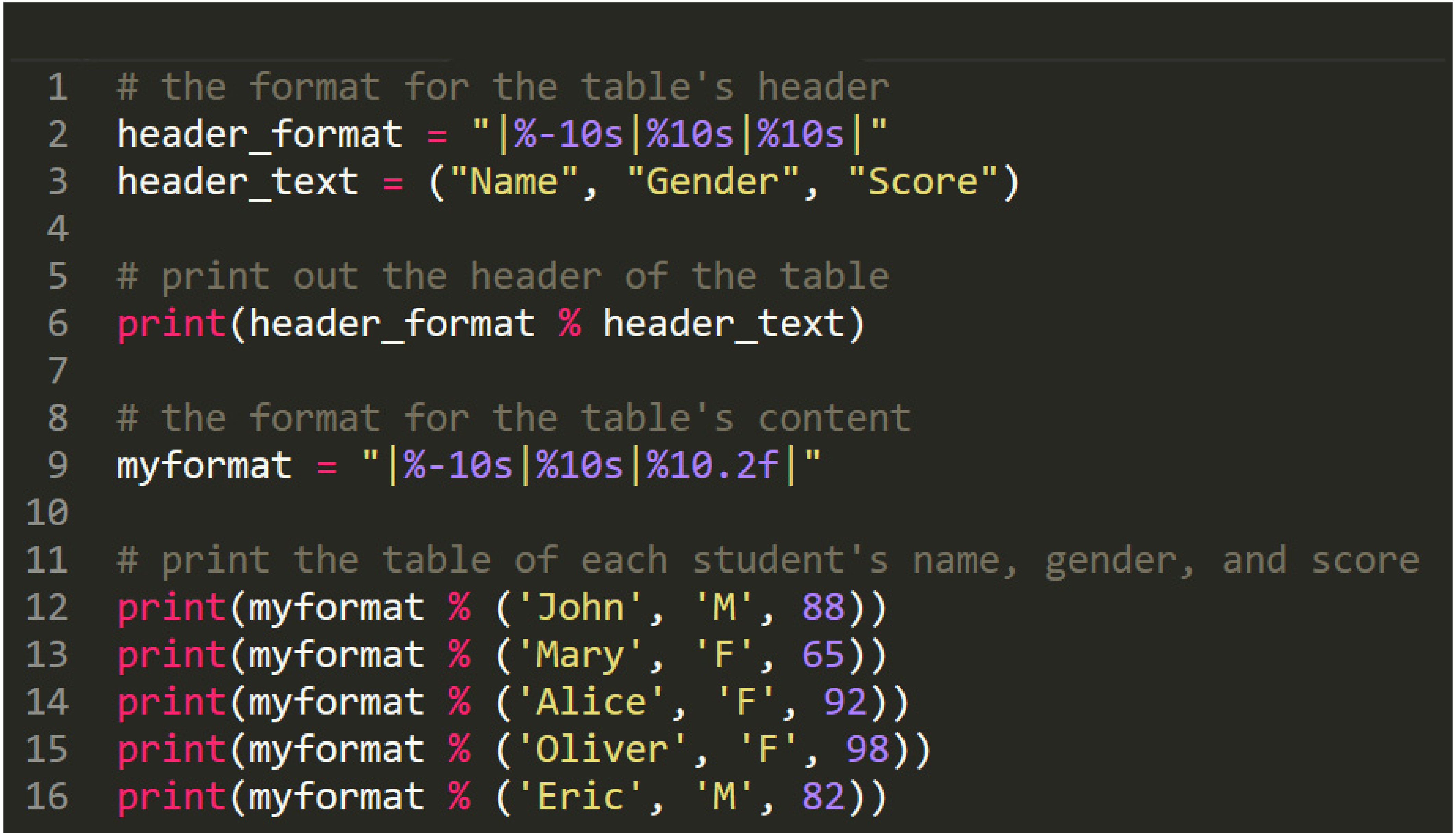
Example: Score Table

- Write a program (scoretable.py) that prints out the three-column table (Name, Gender, Score), in which the first column is left-aligned and the second and third columns are right-aligned



```
c:\Python35-32\workspace>python scoretable.py
|Name      |    Gender|     Score|
|John      |         M|     88.00|
|Mary      |         F|     65.00|
|Alice     |         F|     92.00|
|Oliver    |         F|     98.00|
|Eric      |         M|     82.00|
```

Example: Score Table



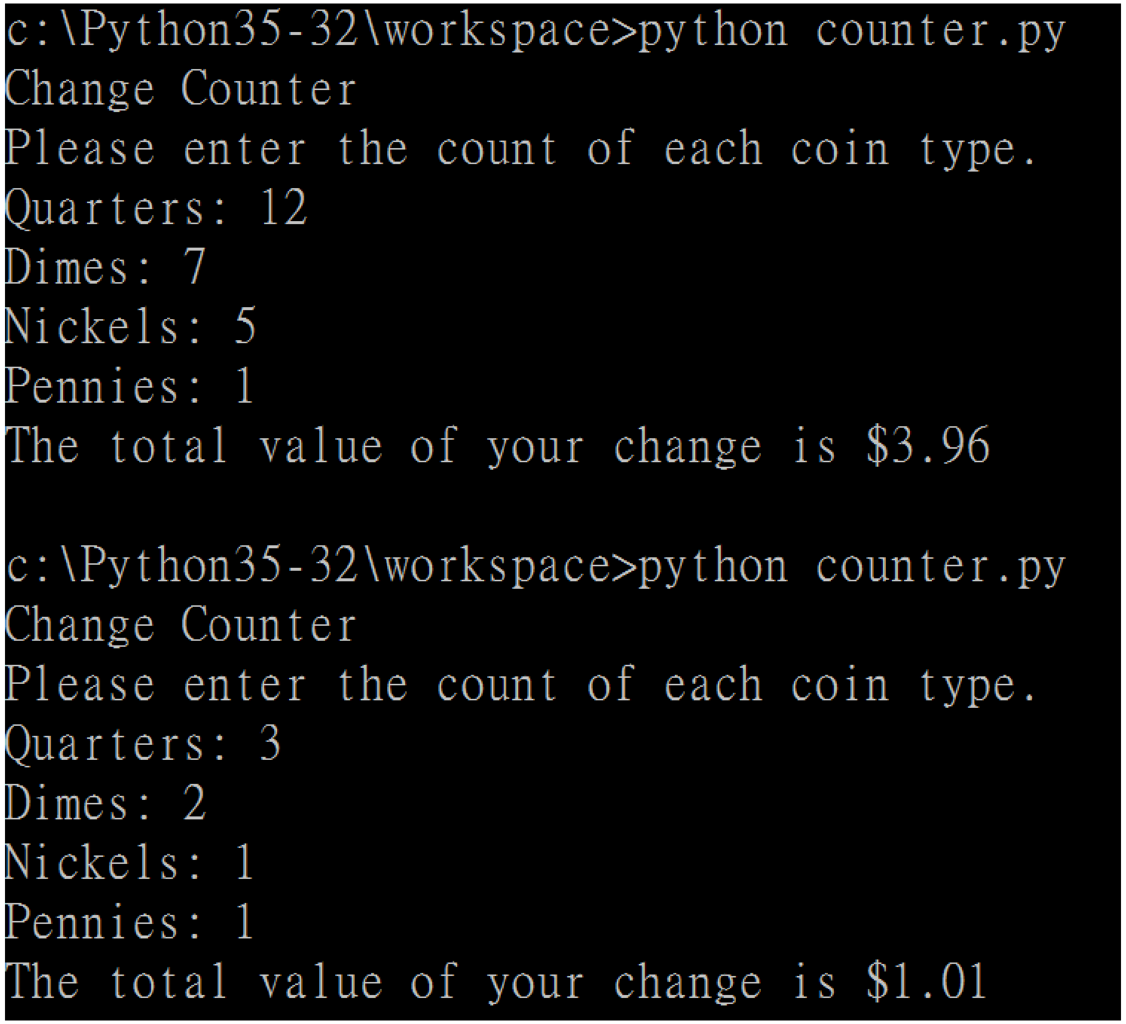
```python
# the format for the table's header
header_format = "|%-10s|%10s|%10s|"
header_text = ("Name", "Gender", "Score")

# print out the header of the table
print(header_format % header_text)

# the format for the table's content
myformat = "|%-10s|%10s|%10.2f|"

# print the table of each student's name, gender, and score
print(myformat % ('John', 'M', 88))
print(myformat % ('Mary', 'F', 65))
print(myformat % ('Alice', 'F', 92))
print(myformat % ('Oliver', 'F', 98))
print(myformat % ('Eric', 'M', 82))
```

In-class Exercise: Change Counter

Write a program (counter.py)

- Allow users to input the numbers of different types of coins
  - Quarter (0.25元=25分)
  - Dim (1角=10分)
  - Nickel (5分)
  - Penny (1分)

  Note: 1元=10角=100分
- Calculate the value of some change in dollars (元)



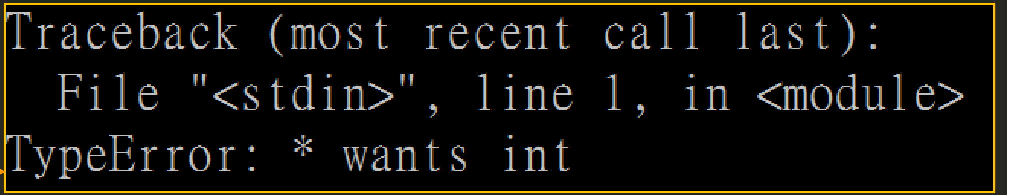
```
c:\Python35-32\workspace>python counter.py
Change Counter
Please enter the count of each coin type.
Quarters: 12
Dimes: 7
Nickels: 5
Pennies: 1
The total value of your change is $3.96

c:\Python35-32\workspace>python counter.py
Change Counter
Please enter the count of each coin type.
Quarters: 3
Dimes: 2
Nickels: 1
Pennies: 1
The total value of your change is $1.01
```

```python
# print welcome information
print("Change Counter")
print("Please enter the count of each coin type.")

# let the user input coins
quarters = input("Quarters: ")
dimes = input("Dimes: ")
nickels = input("Nickels: ")
pennies = input("Pennies: ")

# comput the total
total = float(quarters) * 25 + float(dimes) * 10 \
        + float(nickels) * 5 + float(pennies)

# print out the result
my_format = "The total value of your change is $%d.%02d"
print(my_format % (total/100, total%100))

# another method
print("The total value of your change is $%.2f" % (total/100))

# this will get an error
#print("The total value of your change is $%.2f" % total/100)
```

Formatted Printing

- You can use an * (asterisk) as the width or precision (or both). In that case, the number will be read from the tuple argument

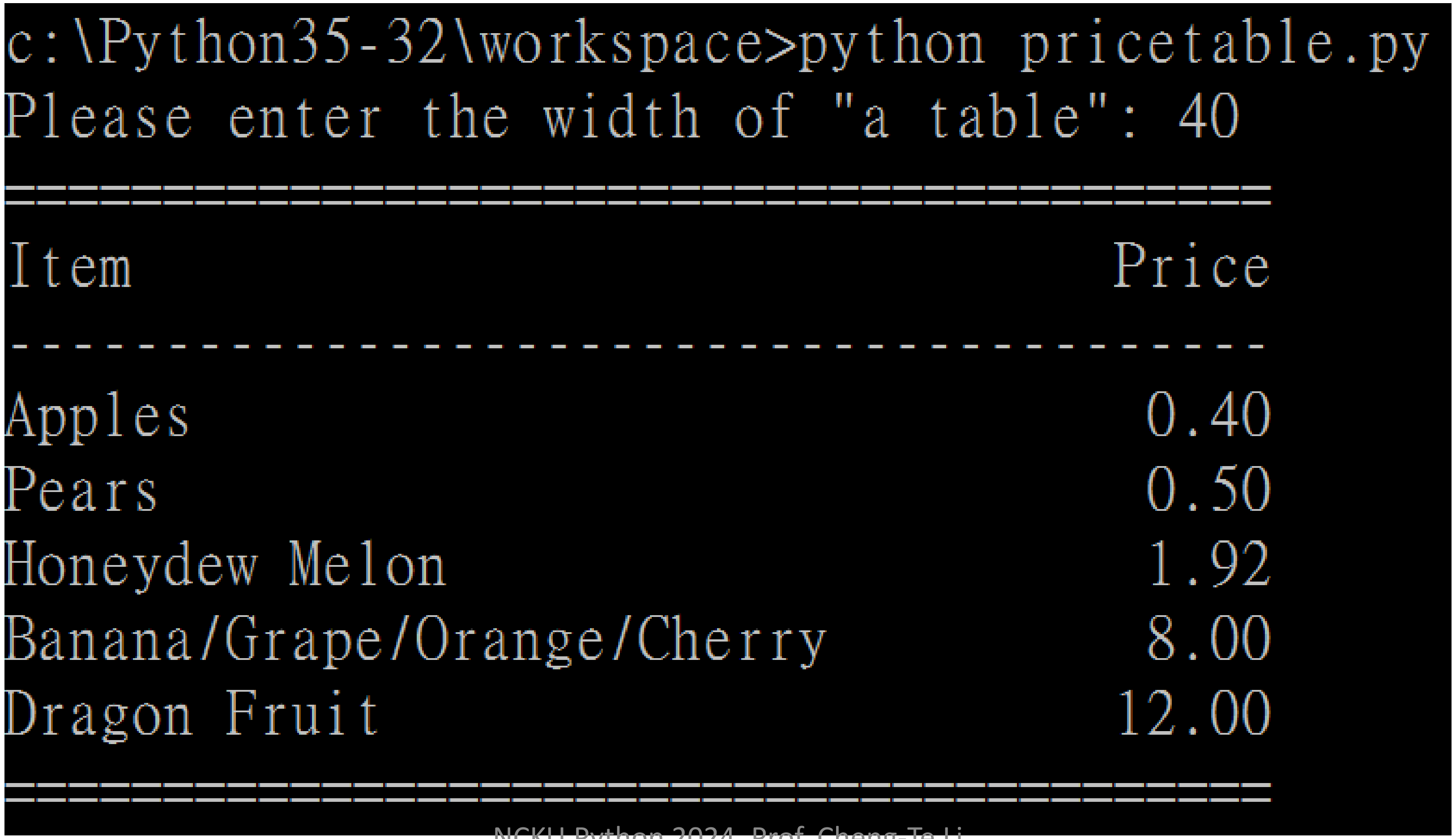
```
value = "Hello World"
str = "%10.5s" % (value)
#     Hello
str = "%*.*s" % (10, 7, value)
print(str)
#   Hello W
weight = 12.3456789
myformat = "%*.*f"
print(myformat % (weight))
print(myformat % (6, 3, weight))
#12.346
```



```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: * wants int
```

In-class Exercise: Print a Table

- Write a program (pricetable.py)
  - Allow the user to input the width of a table
  - Create and print out the table for the prices of fruits



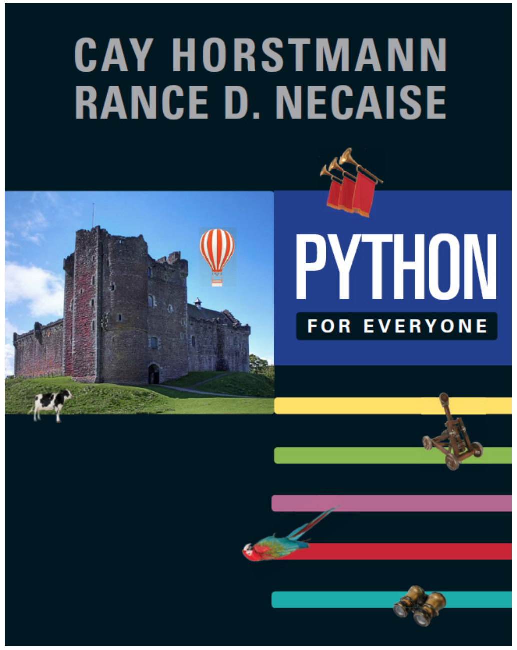
```
c:\Python35-32\workspace>python pricetable.py
Please enter the width of "a table": 40
========================================
Item                               Price
----------------------------------------
Apples                              0.40
Pears                               0.50
Honeydew Melon                      1.92
Banana/Grape/Orange/Cherry          8.00
Dragon Fruit                       12.00
========================================
```

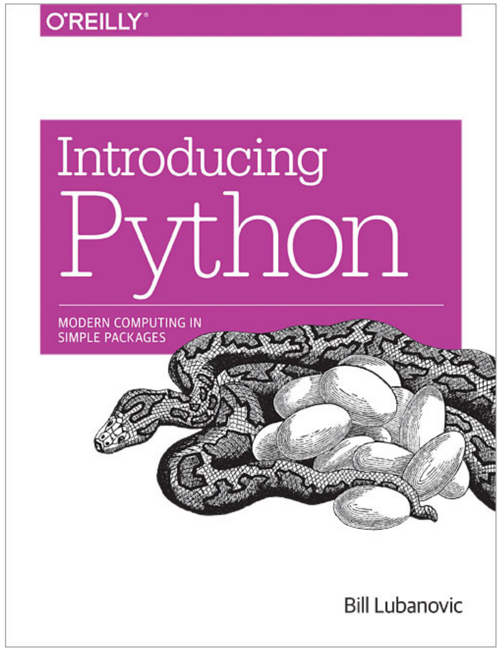
```python
# print a formatted price list with a given width
width = int(input("Please enter the width of \"a table\": "))
price_width = 10                   # the width of price
item_width = width - price_width   # the width of item

# the format for the table's header
header_format = "%-*s%*s"
header_text = (item_width, "Item", price_width, "Price")

# print out the header of the table
print("=" * width)
print(header_format % header_text)

# the format for the table's content
item_format = "%-*s%*.2f"
print("-" * width)

# print out the table of fruit prices
print(item_format % (item_width, 'Apples', price_width, 0.4))
print(item_format % (item_width, 'Pears', price_width, 0.5))
print(item_format % (item_width, 'Honeydew Melon', price_width, 1.92))
print(item_format % (item_width, 'Banana/Grape/Orange/Cherry', price_width, 8))
print(item_format % (item_width, 'Dragon Fruit', price_width, 12))

print("=" * width)                 # end of the table
```

Summary

- String Data Type
- Escape Sequence
- Concatenation, Duplication, Contain?
- Indexing Strings
- Slicing Strings
- Built-Function for String
  - len(), startswith(), endswith(), find(), rfind()
  - count(), isalnum(), strip(), lstrip(), rstrip()
  - split(), join(), replace()
  - capitalize(), title(), upper(), lower()
- Formatting String

Suggested Reading





P.48 – P.64

P.27 – P.39

P.152 – P.156