

Python 數據分析與視覺化篇

Dr. Steve Lai

1

Python 基礎操作

Steve Lai (賴昭榮)

- 台灣大學資訊工程學系博士
- 台灣金融研訓院2022菁英講座
- 麥錫森智能科技 創辦人兼CEO
 - ◆ 自然語言與金融資料分析
 - ◆ Plusform 為旗下線上教學品牌
 - 金融研訓院-芬課斯平台 Python 與 Java 課程
 - Hahow iOS UIKit 課程
 - 專長：Mobile Application (Android、iOS) ，資料分析與探勘，機器學習與深度學習



Steve Lai

1-1

數值與型態

(Value and Type)

數值 (Value) 與型態 (Type)

- 在 Python 中的基本數值分為
 - 整數 (integer)
 - 浮點數 (float-point number)
 - 字串 (string)
 - 布林值 (boolean)
- 不會改變的數值可稱為常數值 (Constant)
- 使用內建函數 `type()` 可在 Python 中查看型態

整數 (integer)

- 就如同數學中的整數
 - E.g., 123, 1024

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

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

浮點數 (float-point number)

- 因為電腦無法精確表示數字中之有理數 (rational number) ，只能以近似值來標示，此稱為浮點數。
 - E.g., 12.3, 10.24

```
>>> type(12.3)
```

```
<class 'float'>
```

字串 (string)

- 在 Python 中，文字的表示型態只有字串型態，不像其他語言還有字元 (character) 型態。
- 字串有三種表示方法：
 - 單引號，e.g., 'Hello, World!'
 - 雙引號，e.g., "Hello, World!"
 - 三個雙引號或是三個單引號所表示的多行字串，e.g.,

"""Hello, World!
I'm Steve"""

```
>>> type("Hello, World!")  
<class 'str'>
```

布林值 (boolean value)

- 表達真或假、對或錯的值
 - 在 Python 中，真表示為 `True`，假表示為 `False` (注意：大小寫有差別)
 - 不同程式語言中，大小寫及表示法或有不同

```
>>> type(True)
```

```
<class 'bool'>
```

1-2

變數與其使用情境

(Variables)

變數 (Variable)

- 在程式執行過程中，需要將數值暫時保留做為後續運行時使用。
- 變數：就是將數值存放起來並且給予變數的名稱，在程式運行過程中可再透過這個名稱存取這個數值。

x = 12.24

x 12.24

y = 1024

y 1024

變數 (Variable)

- 在程式執行過程中，需要將數值暫時保留做為後續運行時使用。
- 變數：就是將數值存放起來並且給予變數的名稱，在程式運行過程中可再透過這個名稱存取這個數值。

x = 12.24

x 12.24

y = 1024

y 1024 200

y = 200

Python 的變數命名規則

- 必須是文字或是底線（_, underscore）開頭
- 只能包含文字、底線與數字
- 大小寫有區別（case sensitive）

Good: spam eggs spam23 _speed

Bad: 23spam #sign var.12

Different: spam Spam SPAM

如何善用變數名稱

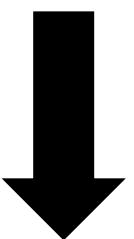
- 變數名稱：雖然只要遵循規則，都可隨意使用變數名稱，然而選擇好的變數名稱使用能有以下好處
 - 幫助程式設計師記憶
 - 程式設計過程中減少錯誤
 - 協同合作更能瞭解變數用途與內容

如何善用變數名稱

```
xs23sdf = 35.0
df43jkl = 12.50
xs23sdf = xs23sdf * df43jkl
print(xs23sdf)
```

如何善用變數名稱

```
xs23sdf = 35.0  
df43jkl = 12.50  
xs23sdf = xs23sdf * df43jkl  
print(xs23sdf)
```



```
a = 35.0  
b = 12.50  
c = a * b  
print(c)
```

如何善用變數名稱

```
xs23sdf = 35.0  
df43jkl = 12.50  
xs23sdf = xs23sdf * df43jkl  
print(xs23sdf)
```



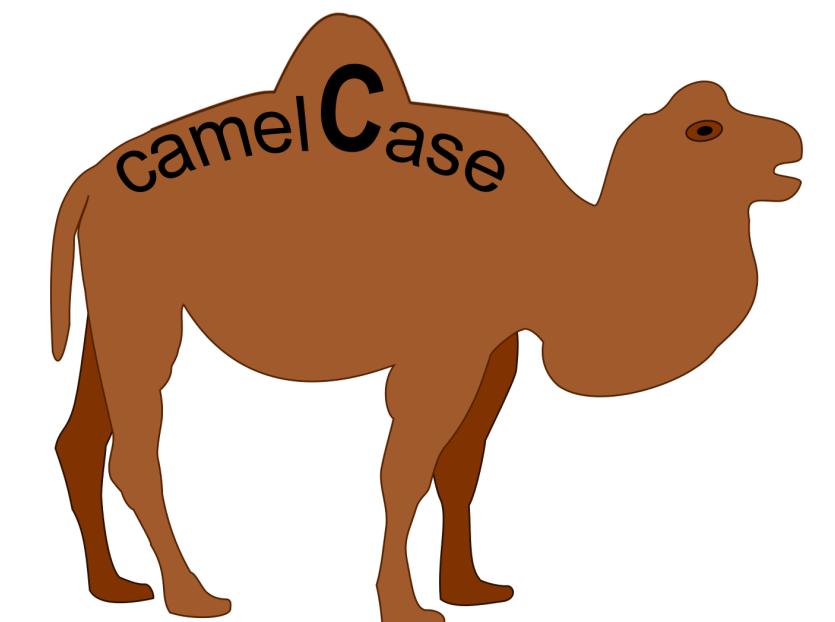
```
a = 35.0  
b = 12.50  
c = a * b  
print(c)
```



```
days = 35.0  
interest = 12.50  
balance = days * interest  
print(balance)
```

變數命名習慣

- 取有意義的名稱，e.g., height, balance
- 可用複合字，e.g., my_bank_no, customerNo
- 兩種複合字的命名方法：
 - 用底線隔開，又可稱為蛇形命名法（snake case），Python社群慣用
 - 用大寫隔開，又稱駝峰式命名（camel case），Java社群慣用
- 一般變數開頭習慣用小寫字母（大寫命名另有含義）

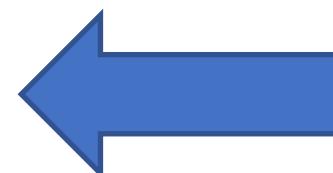


From wiki

程式敘述 (Statement)

- 程式敘述為程式語言中最小的執行單位
 - 單一程式敘述可能有一個或是多個程式行為

```
x = 5  
x = x + 5  
print(x)
```



指定敘述 (Assignment statement)

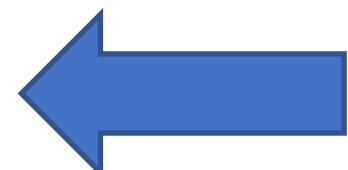
包含運算的指定敘述 (Assignment with expression)

列印函數敘述 (Print statement)

指定敘述 (Assignment statement)

- 指定敘述：用來表示將數值指定給變數的語法
 - 將等號 (=) 右邊的數值指定給等號 (=) 左邊的變數
 - 單一等號在程式語言中多作為指定敘述
 - 當等號右邊包含其他敘述時，將會先完成等號右邊的結果，再將結果指定於等號左邊

`x = 5`



將整數5指定給變數x

`x = x + 5`

將x中的5加上，再將結果指定給變數x

`print(x)`

將x的內容列印在畫面上

1-3

運算敘述

(Expression Statements)

數值運算

- 使用電腦中的符號表達數值運算
 - 星號 (*) 代表「乘法」
 - 兩個星號 (**) 代表「次方」
 - 百分比 (%) 代表「計算餘數」

Operator	Operation
+	加法 (Addition)
-	減法 (Subtraction)
*	乘法 (Multiplication)
/	除法 (Division)
**	次方 (Power)
%	餘數 (Remainder)

運算優先順序

- 與數學中的運算優先權相同
 - 刮號最先
 - 指數運算
 - 先乘除（餘數相當於除法）
 - 後加減
 - 由左至右
- 然而為了容易閱讀
 - 善用刮號
 - 盡量簡化運算
 - 將過長的算式分散成多行敘述

Parenthesis
Power
Multiplication
Addition
Left to Right



不同數值間的運算與轉換

- 數值之間可互相運算
 - 必須知道運算結果的型態
 - 可透過type()內建函數瞭解型態
 - 不同型態之間不一定能夠互相運算，其結果可能有所不同

使用者輸入

- 使用input()可讓程式停下來等待輸入
 - 在程式執行過程中讓操作者或使用者輸入值
 - 使用input()所產生的值其型態為字串

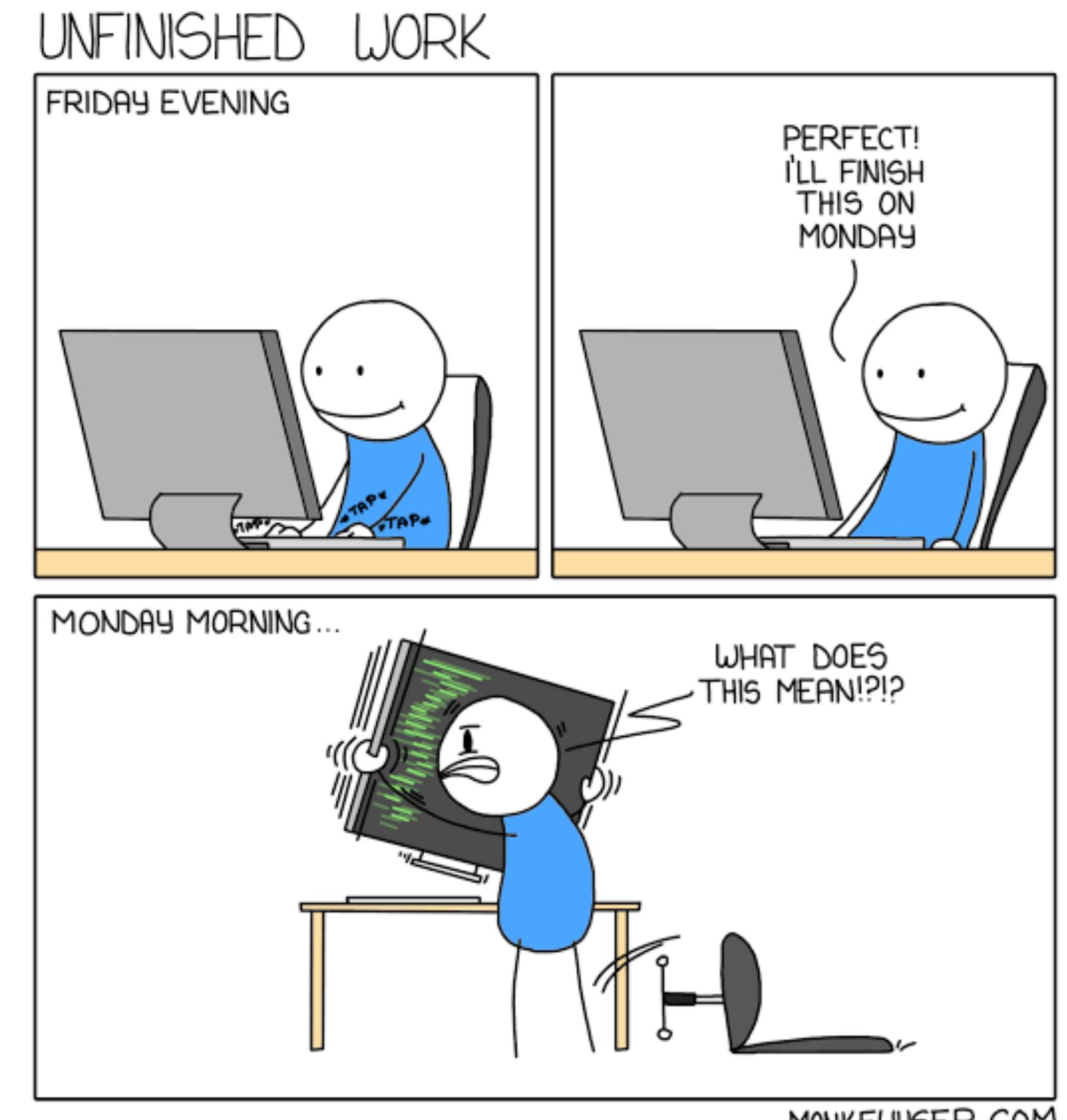
```
nam = input('Who are you? ')
print('Welcome', nam)
```

數值型態轉換 (Type Converting)

- `input()`所輸入的值必須經過轉換才能夠進行數值運算
 - `int()`：轉換值為整數，e.g., `int(" 123 ")`，將"123" 轉換為整數123
 - `float()`：轉換值為數，e.g., `int(" 12 . 3 ")`，將"12 . 3" 轉換為浮點數12.3
 - `str()`：轉換值為字串
 - `bool()`：轉換值為布林值

註解 (Comments)

- 在 Python 中，在 # 後面的文字皆不會執行，稱之為註解
- 註解的使用時機與用途
 - 描述單行程式碼行為
 - 描述函數行為
 - 描述程式區塊行為
 - 提醒自己程式目的
 - 與其他人合作
- 良好的註解將會提升程式閱讀性與維護性



案例練習

- 輸入身高 (cm) 體重 (kg) 後計算BMI

關係運算 (Relational Operator)

- 數值之間的關係運算，會產生對或錯的結果，其型態為布林值
 - True or False
 - E.g., $12 \geq 12$, 結果為 True
 - E.g., $12 > 12$, 結果為 False

Python	Meaning
<	Less than
\leq	Less than or Equal to
$=$	Equal to
\geq	Greater than or Equal to
>	Greater than
\neq	Not equal

邏輯運算 (Logical operation)

- 布林值之間的運算
- and : 兩者必須同時為真才為真
- or : 一者為真即為真
- not : 真轉為假，假轉為真
- Truth table:

Value1	Value2	and
T	T	T
T	F	F
F	T	F
F	F	F

Value1	Value2	or
T	T	T
T	F	T
F	T	T
F	F	F

Value1	not
T	F
F	T

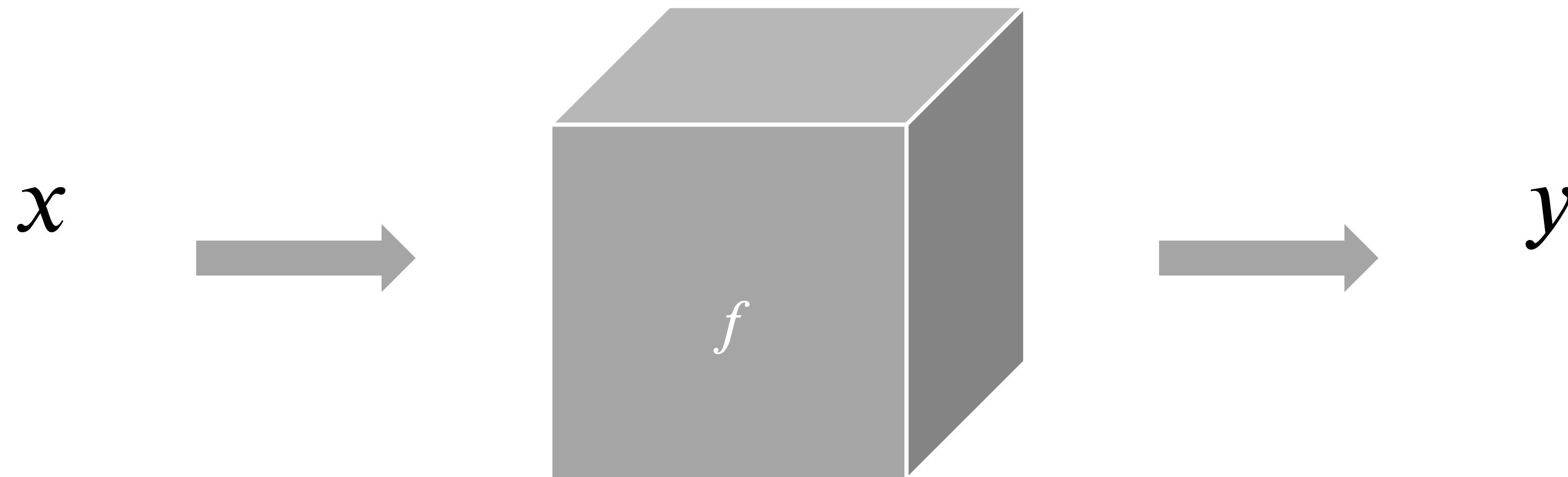
1-4

函數

(Functions)

函數 (Function)

- 回想數學中所學的函數： $y = f(x)$
 - x ：代表輸入至函數的值， y ：代表經過函數運作後的輸出值
 - f ：函數的名稱，代表某個函數
 - 數學函數可想像為對輸入值處理的一個算式



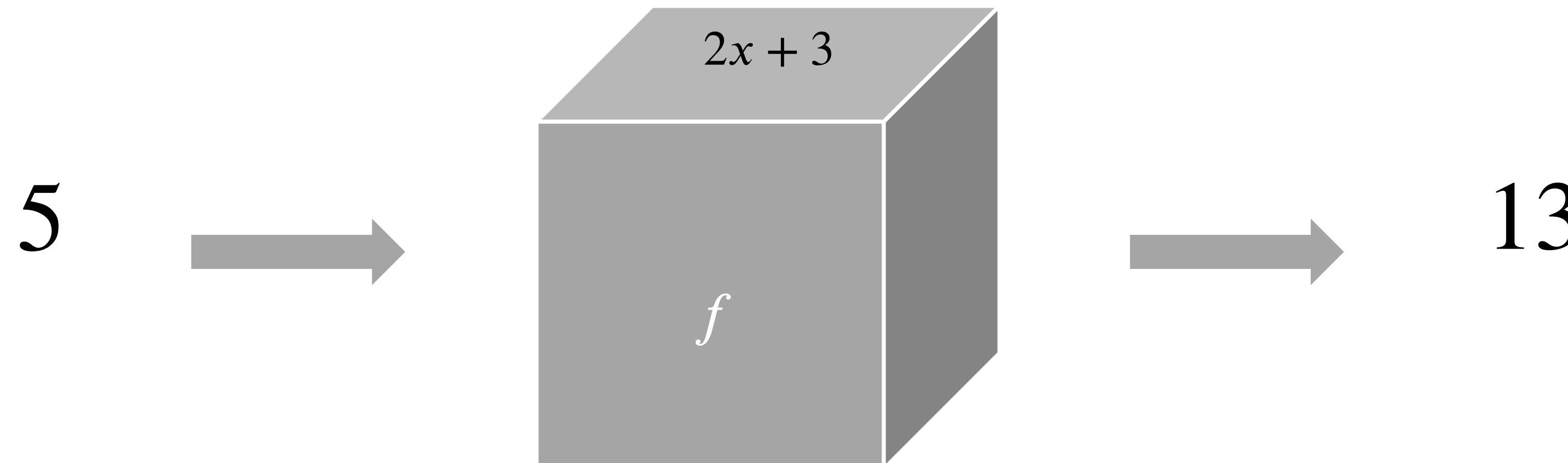
函數 (Function)

- E.g., 假設 f 的算式為對輸入乘上2後加上3，則 $f(x) = 2x + 3$

- 當 $x = 5$

- 將 x 的實際值5輸入，則 $f(5) = 2 * 5 + 3 = 13$

- 輸出的實際結果 $y = 13$



Python中的函數

- 內建函數 (built-in functions, BIF)
 - Python已經事先定義好並可直接使用的函數
 - 前述所介紹過的 `print()`, `type()`, `input()`, `int()`, `float()`, `str()`等皆屬於內建函數
 - 內建函數的名稱請視為保留字元，避免當成變數使用（可作為變數，但將會使得內建函數失效而產生錯誤訊息）
 - 後續課程陸續會使用到不同內建函數
- 內建函數列表
 - <https://docs.python.org/3/library/functions.html>

BIF print() in More Detail

- 官方文件中built-in function：print的部分定義
- `print(*obj, sep=' ', end='\n')`
 - *obj：要列印的值，前面*代表可以輸入多個值，每個值以逗號相隔
 - sep與end為關鍵字參數，在等號後面加上要輸入的值
 - sep代表列印出的值中間的間隔符號，預設值為一個空格的字串
 - end代表列印完後的結果，預設值為換行

```
print("Hello", 1024, "World!", sep="||", end='~~\n')  
Output: Hello||1024||World!~~
```

Python中的自訂函數

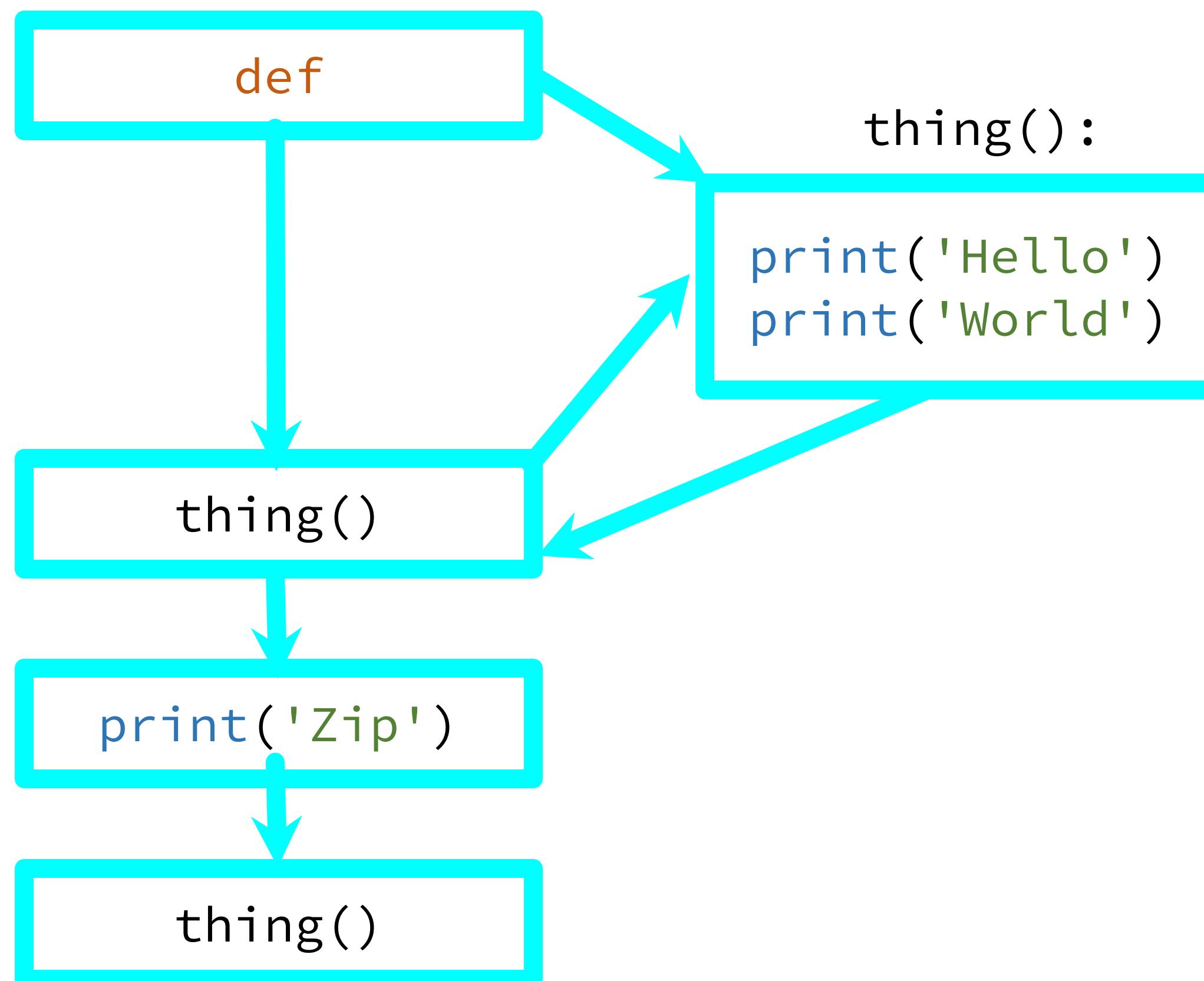
```
def function_name (parameters):
```

function code block
(函數的程式區塊)

Python中的自訂函數

- def 為自訂函數的關鍵字，後面接上函數名稱
- 刮號與冒號不能少，輸入的參數可以沒有，可以一個或多個
- 函數也可以有一個或多個輸出值，也可以沒有輸出值
- 除了輸入輸出之外，函數中也可以有其他影響值的內容或是其他行為，稱為函數的副作用（side-effect）
- 函數的程式區塊（coding block）必須要縮排（indentation）
 - Python以縮排來區分程式的區塊

Python中的自訂函數



```
def thing():
    print('Hello')
    print('World')
```

Output:

```
thing()
print('Zip')
thing()
```

Hello
Fun
Zip
Hello
Fun

函數的使用時機與優點

- 函數定義之後，可在需要的時候呼叫使用
 - 可儲存且重複執行函數中所定義的行為
- 使用函數的時機
 - 一連串重複執行的程式執行步驟
 - 完成某個特定目標的程式執行步驟
- 使用函數的好處
 - 可在需要時呼叫執行函數
 - 可重複使用
 - 當執行步驟又更動時，只需要更動函數內容，也更容易找尋錯誤

參數 (Parameter)

- 參數，或稱形式參數，是在定義函數的時候，將輸入值存放的變數，並在函數執行時使用

```
def greeting(name):  
    print('Hello')  
    print(name)
```

- name即為parameter

引數 (Argument)

- 引數，或稱實際參數，為呼叫函數時的輸入的值
- 在不同的時候使用不同的引數可讓函數完成不同的工作

```
length = len('Hello world')
```

- 
- 'Hello world' 即為 argument

回傳值 (Return Value)

- 函數結束執行時會回到呼叫的地方繼續執行
 - 可使用return關鍵字，如無回傳值也可省略
 - 如有回傳值，接在 return 後面

```
def greeting(name):  
    print('Hello')  
    print(name)  
    return
```

```
def add5(num):  
    result = num + 5  
    return result
```

1-5

模組

(Module)

腳本與檔案

- 單行Python的程式碼，稱之為程式敘述（statement）
 - Python可以單獨執行一行程式碼，也是直譯式語言（interpreted language）特性之一
- 當程式碼越來越多行，會使用整合開發環境（integrated development environment, IDE）將程式碼撰寫並儲存在檔案之中，並一次執行這個檔案，也稱之為腳本（script）
 - Python檔案副檔名需為 .py
 - 本系列教學 IDE 將會使用 PyCharm

模組 (module)

- 將Python檔案導入到其他 Python 的執行環境，那這個被使用檔案將稱之為模組
 - 引用模組使用 `import`
- Python 內建許多標準模組，例如：
 - 數學相關模組 `math`
 - 處理特殊檔案格式模組 `json`, `csv`
- Python 可透過 `pip` 等指令下載社群中豐富的模組
 - 減少開發時間
 - 資料處理分析相關模組 `pandas`
 - 資料視覺化模組 `matplotlib`

1-6

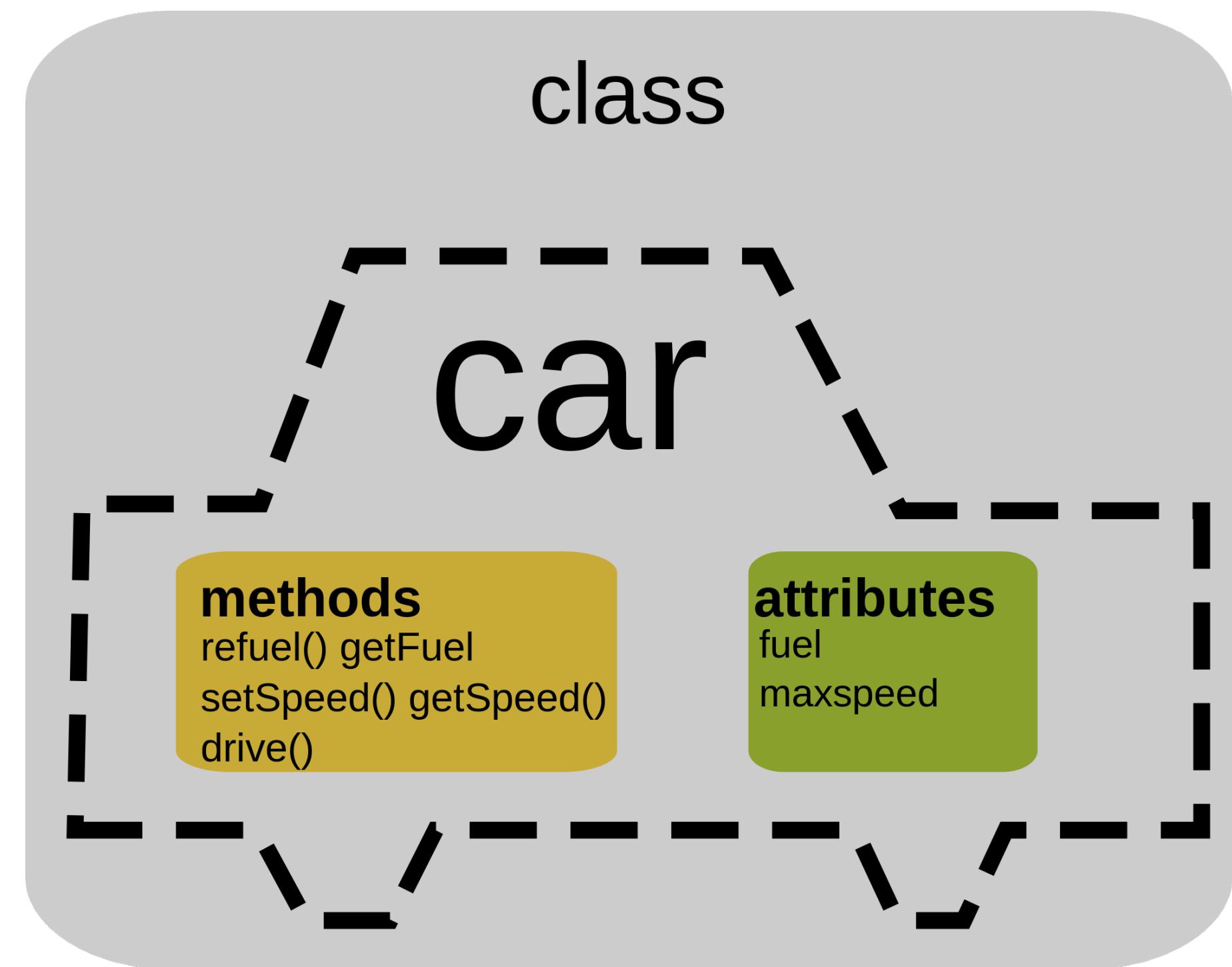
物件導向
(Object-Oriented)

物件 (Objects)

- 日常生活中能夠觸碰、感受或是製造的物品皆是物件
 - 遊戲、汽車、手機皆是物件
- 程式中的物件
 - 雖不能觸碰或感受，但與日常生活的物件相似
 - 都可以做某些事情，或是對它們做某些事
- 正式的說，一個物件收集了資料和它所相關的行為
 - 資料用變數表達，稱為屬性 (attributes)
 - 行為用函數表達，稱為方法 (methods)
 - 相對於函數收集了一連串相關的行為，物件進一步的把這些行為所需要的資料也包裝在一起

物件的範例

- 車輛
 - 屬性（名詞表示的資料）
 - 車型、顏色、廠牌、.....
 - 方法（動詞表示的行為）
 - 前進、後退、.....
- 銀行帳戶
 - 屬性（名詞表示的資料）
 - 帳戶號碼、所有人、餘額
 - 方法（動詞表示的行為）
 - 領錢、存錢



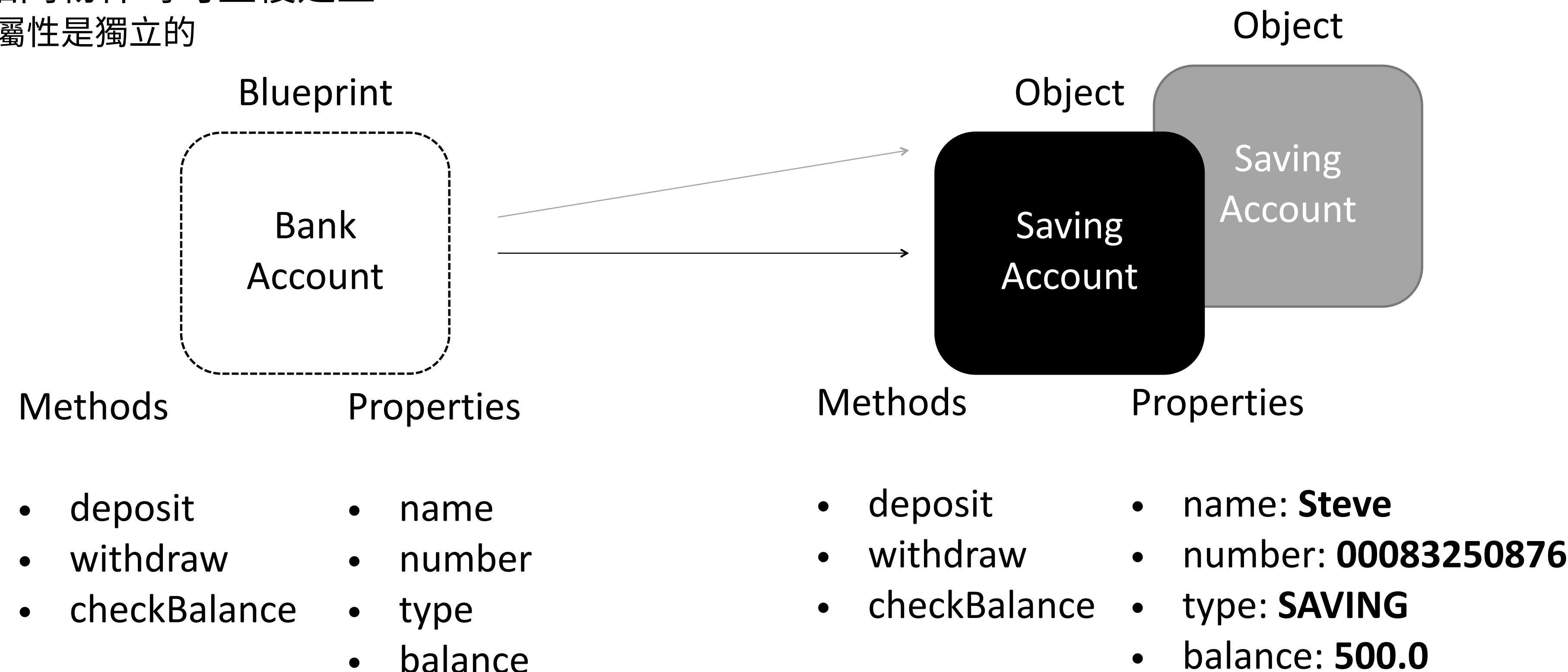
source: <https://gist.github.com/tigarcia>

類別 (Classes)

- 蘋果、橘子都是物件，轎車與卡車也都是物件，它們有相似之處，但也有不同之處
 - 蘋果與橘子都是水果的類別
 - 轎車與卡車也都是交通工具的類別
- 類別描述一些相關的物件，是物件的藍圖

類別 (Classes)

- 在程式中，類別描述該類別建立的物件之屬性與方法
- 執行時用這類別的描述建立物件
- 當需要多個相同物件時可重複建立
 - 每個物件的屬性是獨立的



自訂類別

```
class class_name :
```

```
    class code block  
(函數的程式區塊)
```

自訂類別範例

```
class BankAccount:  
  
    def __init__(self):  
        self.name = ""  
        self.number = ""  
        self.balance = 0  
  
    def withdraw(self, money):  
        self.balance -= money  
  
    def deposit(self, money):  
        self.balance += money
```

BankAccount 為類別名稱
開頭習慣為大寫

- `__init__` 是 Python 類別中初始化的特殊方法
- Python 中有許多特殊的方法
- `name`, `number`, `balance` 都是 `BankAccount` 的屬性

- `self` 為類別方法一定會帶入的參數
- `withdraw` 和 `deposit` 為 `BankAccount` 的方法

類別與物件的使用

```
account_steve = BankAccount()  
account_steve.deposit(100)  
print(account_steve.balance)
```

```
account_andy = BankAccount()  
account_andy.deposit(2000)  
print(account_andy.balance)
```

Output:

100
2000

account_steve 與
account_andy 為兩個獨立的
物件，雖然有同樣的屬性與方
法，但其屬性的內容皆為獨立

可以想成python是這樣執行：

BankAccount.deposit(account_steve, 100)

account_steve 相當於帶入
self 這個參數

同樣是BankAccount，因此可
以從同一個類別重複建立物件

1-7

條件判斷

(Conditional Execution)

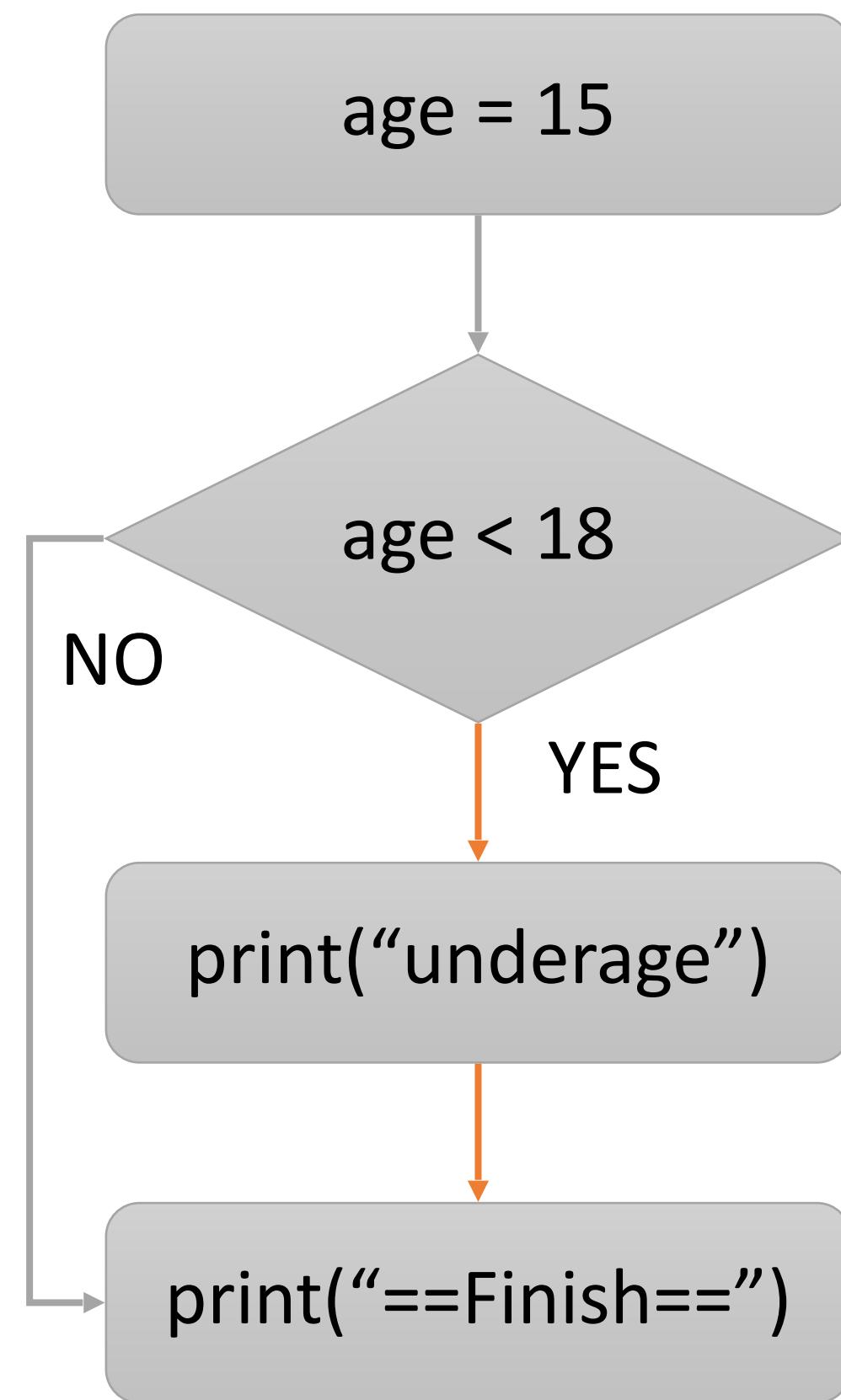
條件判斷語法-if

- if 為條件判斷關鍵字
- 後面會接著一個產生結果為 True or False 的運算
- 如果為True才執行 ”True” 部分的敘述，False則不執行

if some condition holds :

the “True” part statements
(條件成立時執行)

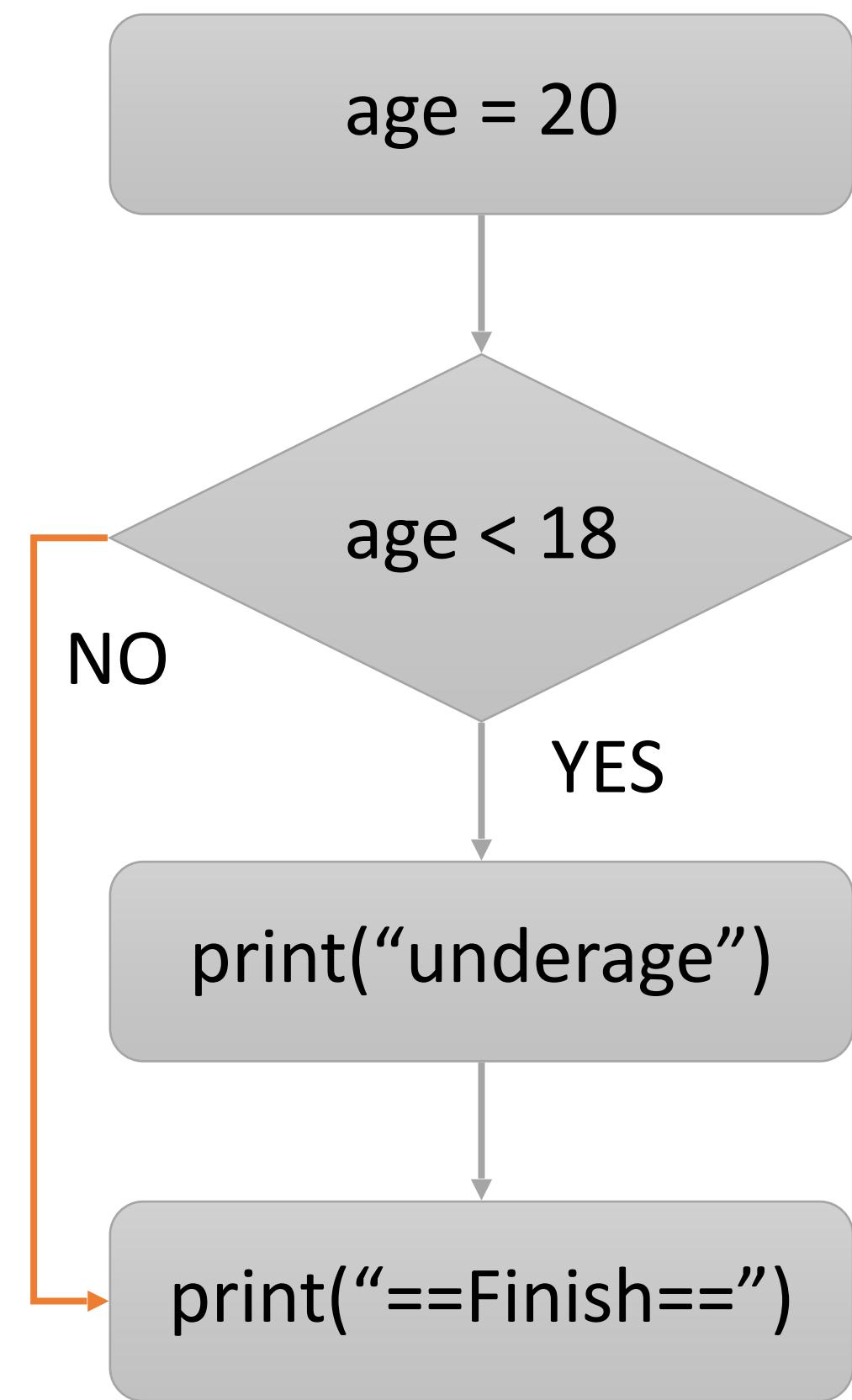
條件判斷語法-if 範例 (1)



```
age = 15  
if age < 18:  
    print("underage")  
print("==Finish==")
```

Output:
underage
==Finish==

條件判斷語法-if 範例 (2)



```
age = 20
if age < 18:
    print("underage")
print("==Finish==")
```

Output:
==Finish==

條件判斷語法-if_else

- 當條件成立時，執行”True” part
- 條件不成立時，執行”False” part

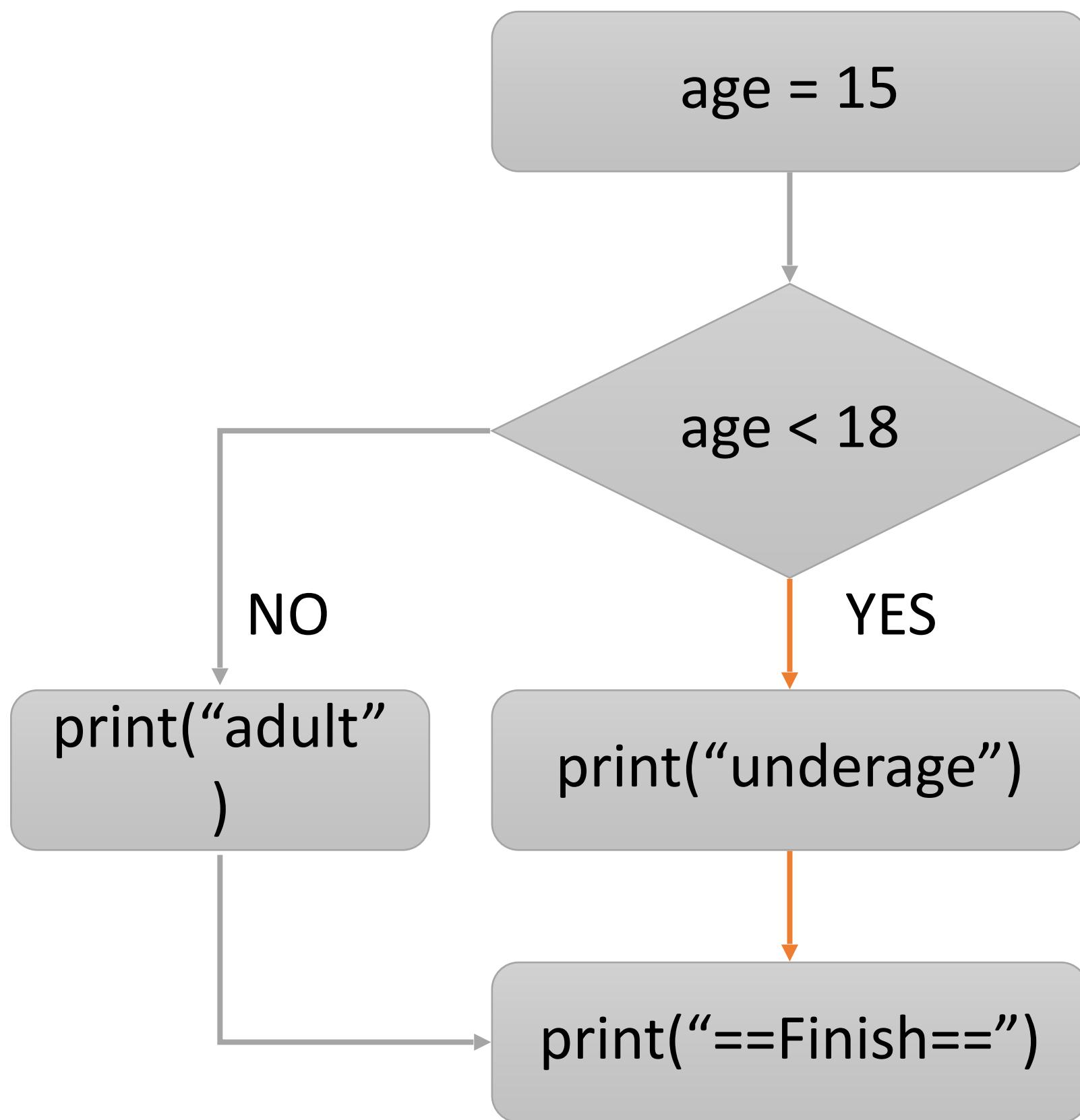
```
if some condition holds :
```

```
    the “True” part statements  
    (條件成立時執行)
```

```
else :
```

```
    the “False” part statements  
    (條件不成立時執行)
```

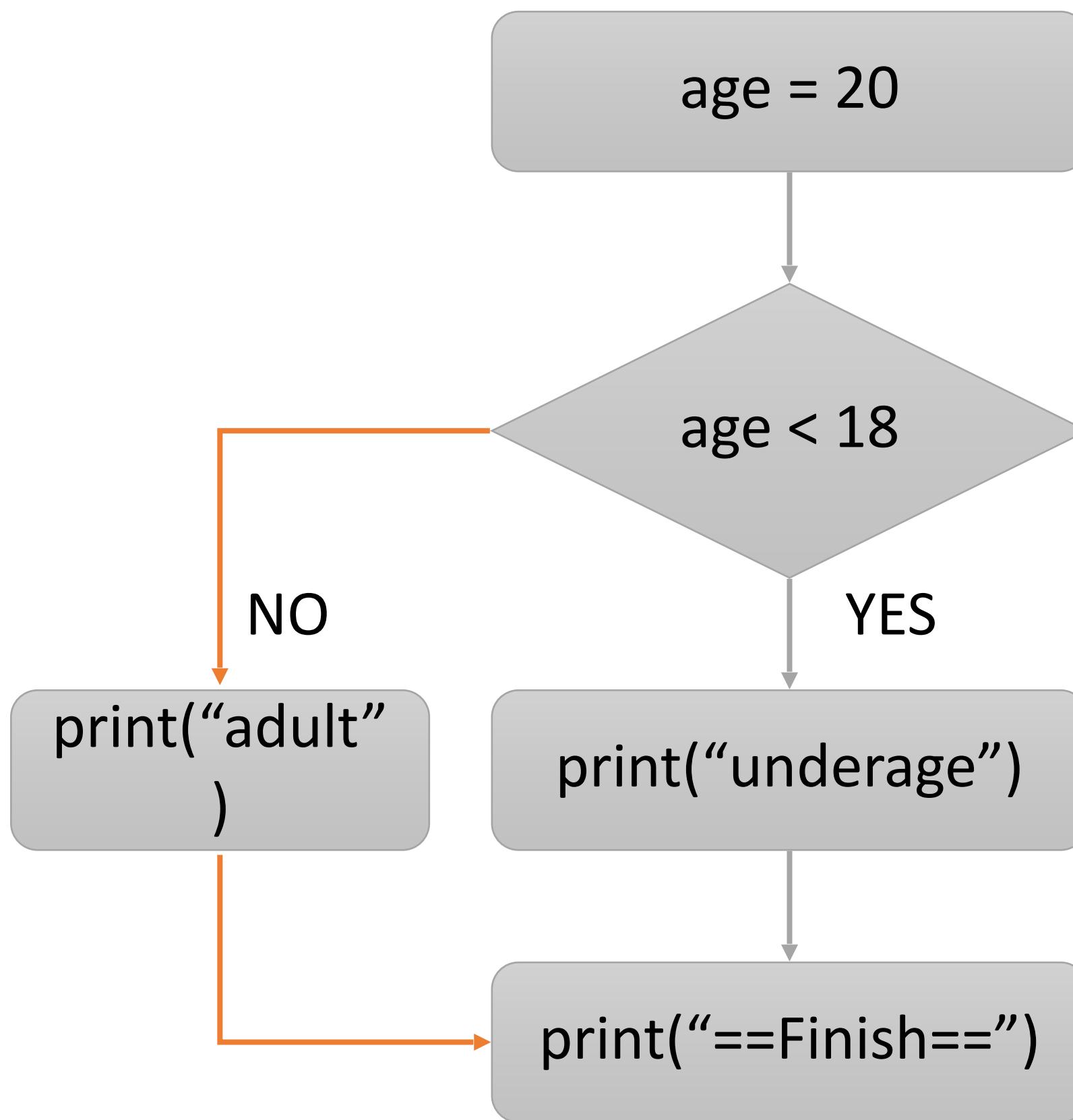
條件判斷語法- if_else 範例 (1)



```
age = 15
if age < 18:
    print("underage")
else:
    print("adult")
print("==Finish==")
```

Output:
underage
==Finish==

條件判斷語法- if_else 範例 (2)



```
age = 20
if age < 18:
    print("underage")
else:
    print("adult")
print("==Finish==")
```

Output:
adult
==Finish==

條件判斷語法-if elif else

- 只會有一個成立

if some condition holds :

the “True” part statements
(條件成立時執行)

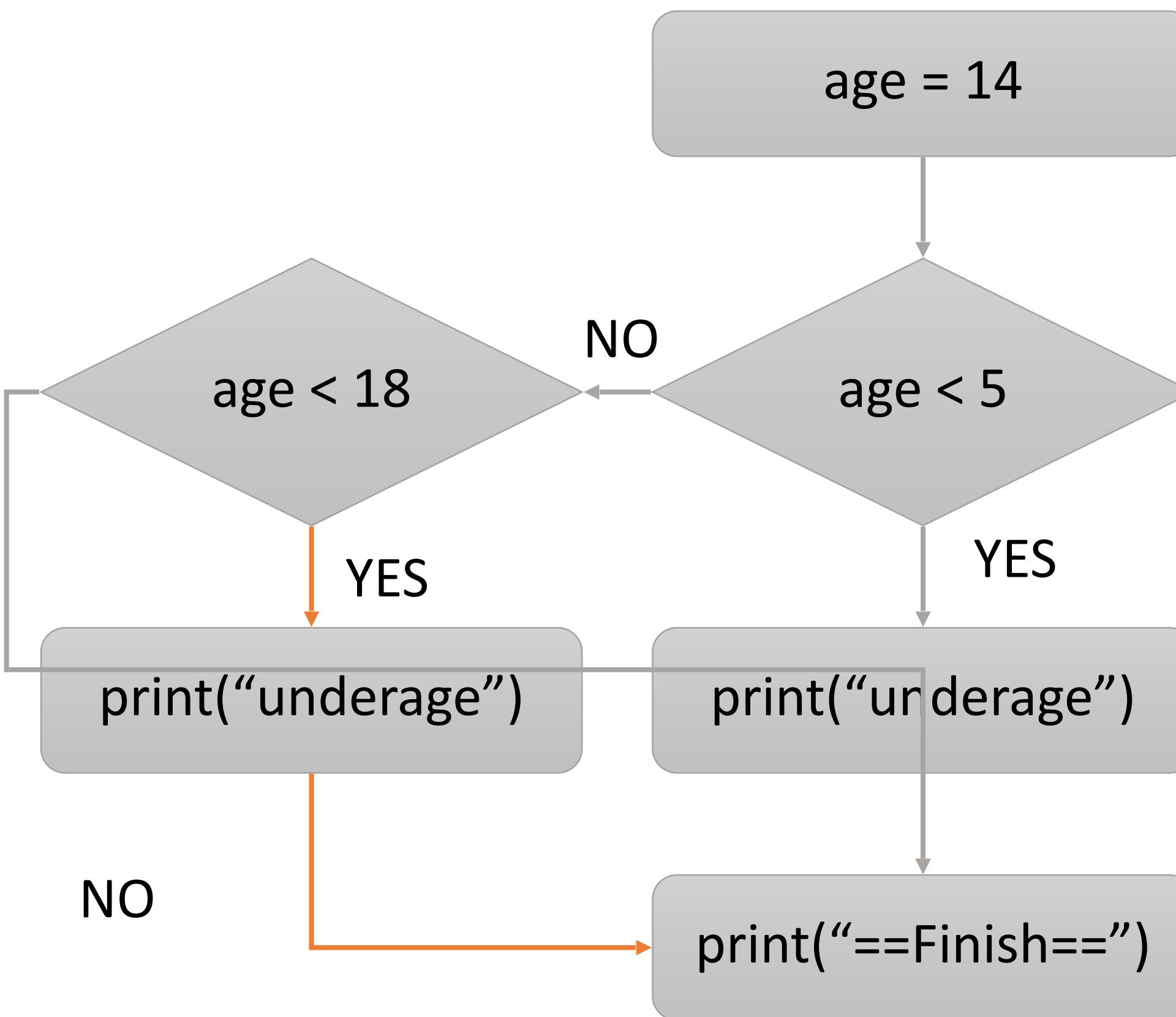
elif some condition holds :

the elif “True” part statements
(條件成立時執行)

else :

the “False” part statements
(條件不成立時執行)

條件判斷語法- if_else 範例 (1)



```
age = 14
if age < 5:
    print("child")
elif age < 18:
    print("underage")
else:
    print("adult")
print("==Finish==")
```

Output:
underage
==Finish==

錯誤偵測-try_except

- Python 在產生錯誤的時候會出現error訊息，並停止程式的執行
- try_expect 語法可以判斷是否產生錯誤，並且找尋對應錯誤來執行
- 將可能會產生錯誤的程式敘述放置於try的程式區塊下，如產生錯誤會跳到對應的expect
- 常見的幾個錯誤類別：
 - ArithmeticError: 計算錯誤
 - NameError: 為定義的錯誤
 - ImportError: 引用模組錯誤
 - Exception: 所有錯誤的最高類別

錯誤偵測-try_except 語法

try some condition holds :

 code statements

except some error class :

 code statements

錯誤偵測-try_except 範例

```
try:  
    print(1 / 0)  
except TypeError:  
    print('型別發生錯誤')  
except NameError:  
    print('使用沒有被定義的對象')  
except Exception as e:  
    print(type(e))  
print('hello')
```

Output:

<class 'ZeroDivisionError'>
hello

1-8

重複執行—迴圈

(Loops)

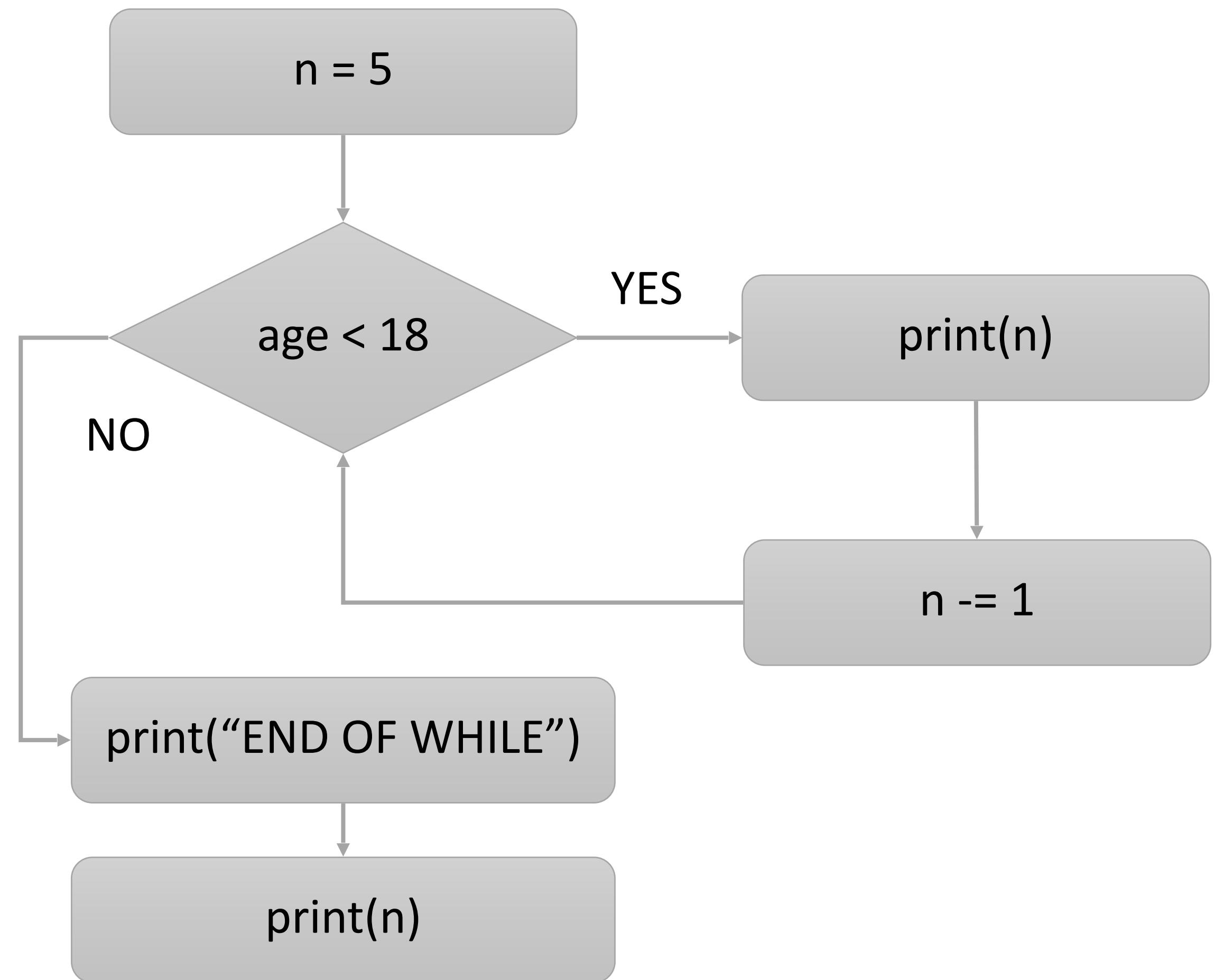
未定義的迴圈 - while loop

- while 條件成立時執行程式區塊
- 執行結束後會重複判斷條件是否成立
- 需依據當時條件是否成立判斷是否執行，因此稱為未定的的迴圈

```
while some condition holds :
```

```
    code block
```

while loop 範例



```
n = 5
while n > 0:
    print(n)
    n -= 1
print("END OF WHILE")
print(n)
```

Output:

5
4
3
2
1
END OF WHILE
0

中斷迴圈 (Break)

```
while True:  
    line = input('> ')  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```

```
> hello there  
hello there  
> finished  
finished  
> done  
Done!
```

中斷迴圈 (Break)

```
while True:  
    line = input('> ')  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```

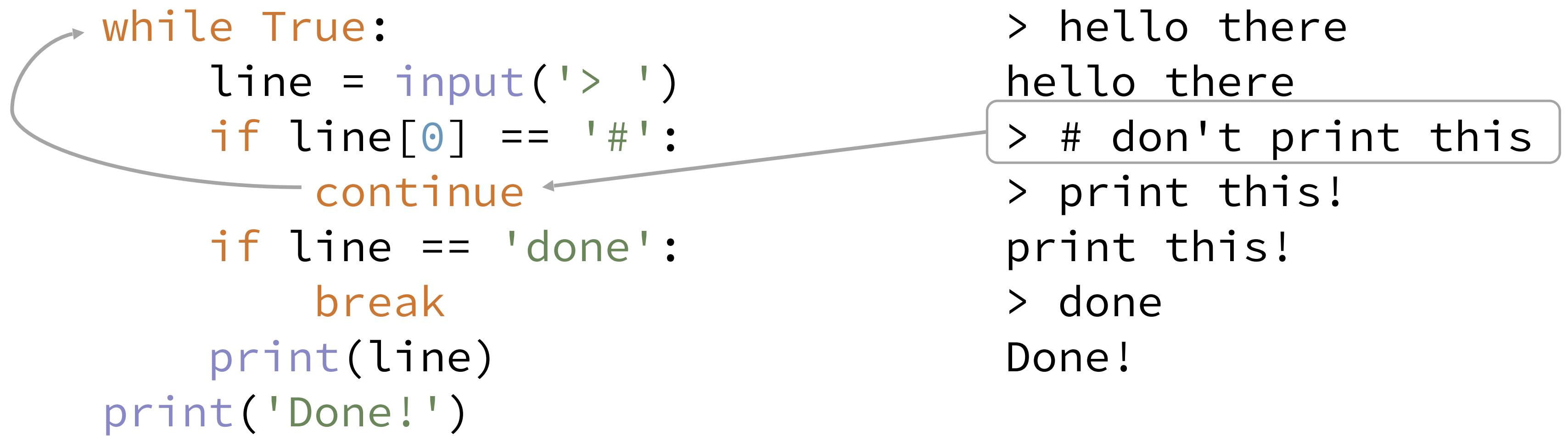
```
> hello there  
hello there  
> finished  
finished  
> done  
Done!
```

結束疊代 (Finish an Iterations with continue)

```
while True:  
    line = input('> ')  
    if line[0] == '#':  
        continue  
    if line == 'done':  
        break  
    print(line)  
print('Done!')
```

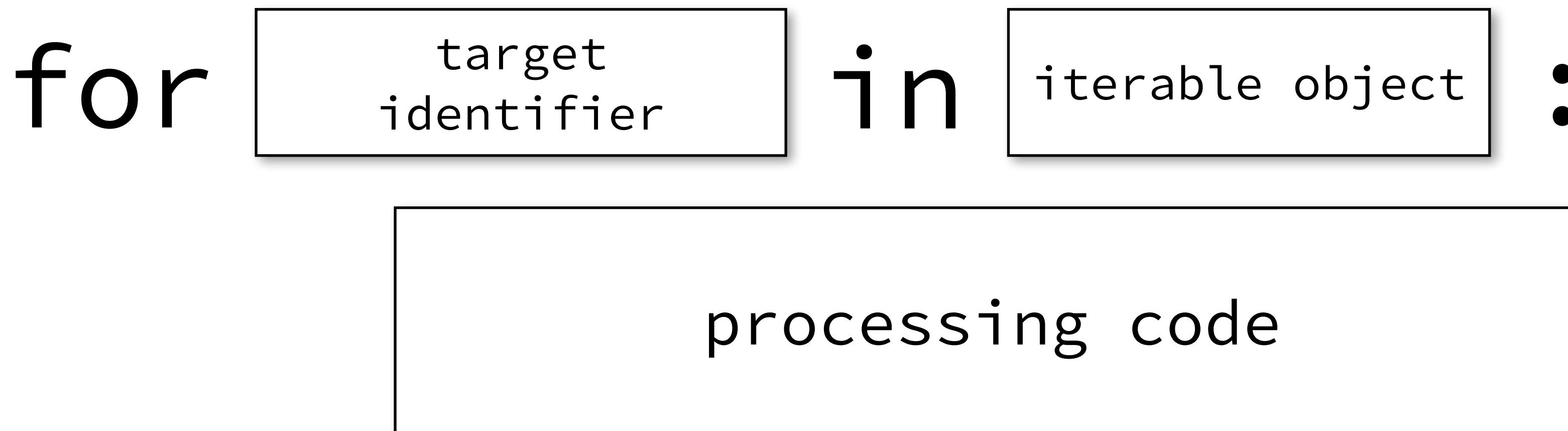
> hello there
hello there
> # don't print this
> print this!
print this!
> done
Done!

結束疊代 (Finish an Iterations with continue)



已定義的迴圈-for loop

- for 後面接上的是標示符號，這個符號會在迴圈中使用
- in 後面接上的是可疊代展開的物件



for loop 範例

Output:

```
for i in "Hello":  
    print(i)  
print("END OF FOR")  
print(i)
```

The diagram illustrates the execution flow of the provided Python code. It starts with the assignment of the string "Hello" to the variable i. This is followed by a loop where the character at index 0 (H) is printed. Then, the string "END OF FOR" is printed. Finally, the character at index 4 (o) is printed again. Arrows connect the code lines to their respective outputs.

H
e
l
l
o
END OF FOR
o

1-9

字元與編碼

(Character and Coding)

字元 (Character)

- 單一的字母或符號稱為字元
- 電腦系統無法直接儲存字元，而是利用編碼表去對應所表示的字元
- 最早的字元編碼系統為ASCII code (Ref: <https://zh.wikipedia.org/wiki/ASCII>)
 - A 編碼為 65
 - a 編碼為 97
 - 可利用BIF `ord()` 取得字元編碼
 - 也可利用BIF `chr()` 將編碼轉換為對應字元
- Python使用Unicode為編碼，相容於ASCII，可表達各國文字與符號

1-10

字串

(String)

表示法

- 被單引號('...) 或是雙引號("...")所包起來的文字皆為字串
- 一次表達多行字串可使用'''...''' 或是 """..."

字符串的運算

- 使用”+”將兩個字符串合併
 - `print("Pyt" + "hon")`, Output: Python
 - 只有兩個字符串允許使用合併運算
- 使用”*”將字符串複製
 - `print("Python" + "!" * 5)`, Output: Python!!!!

字符串中的子字符串

- 字符串使用index的方式取得当中的子字符串

```
greeting = "Hello"  
print(greeting[0])  
print(greeting[3])  
print(greeting[-4])
```

Output:

H
l
e

String:	H	e	I	I	o
index	0	1	2	3	4
:	-5	-4	-3	-2	-1

字符串中的子字符串

- 字串使用slicing的方式取得當中的子字串
- [0:2]代表 0 開始到 2 (但不包含2的位置)

```
greeting = "Hello"
```

```
print(greeting[0:2])
```

```
print(greeting[3:4])
```

```
print(greeting[-4:-2])
```

Output:

He

l

el

String:	H	e	I	I	o
index	0	1	2	3	4
:	-5	-4	-3	-2	-1

字串中的子字串

- 使用超過範圍的index會產生IndexError
- 嘗試更改字串會產生TypeError

```
greeting = "Hello"  
print(greeting[6])  
greeting[3] = "A"  
greeting[-4:-2] = "A"
```

Output:

IndexError
TypeError
TypeError

String:	H	e	I	I	o
index	0	1	2	3	4
:	-5	-4	-3	-2	-1

字串的方法 (The Methods of String)

- 所有的型態在Python中皆為物件
- 字串提供許多方法，可使用BIF `dir(str)`查詢，列舉一些查用的方法與其功能
 - `str.count(sub[, start[, end]])`
 - 回傳str中出現幾個sub，start與end為選項參數，用來表達其範圍
 - `str.endswith(suffix[, start[, end]])`
 - 計算str是否是suffix結尾，並回傳True or False
 - `str.startswith(prefix[, start[, end]])`
 - 計算str是否是suffix開頭，並回傳True or False
- 參閱 <https://docs.python.org/3/library/stdtypes.html#string-methods>

1-11

Lists

(串列)

資料結構 (Data Structure)

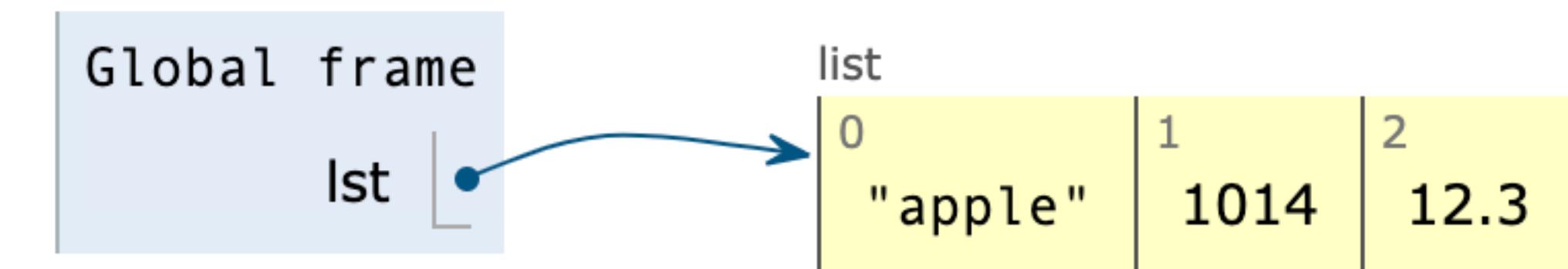
- 單一的數值對於處理大量的資料有其限制
- 以某種形式將資料收集擺放的結構即為資料結構
 - 不同的資料結構有著不同的特性
 - 會根據不同的演算法 (Algorithm) 選擇不同的資料結構
 - 一系列完成某個目的或是解決某個特定問題的步驟集合就是演算法
- 串列即為一種資料結構

串列 (List)

- 將不同資料以順序性的方式擺放的資料結構
- 有眾多資料，且順序是重要的時候使用

```
lst = ["apple", 1014, 12.3]
```

- 使用中括號，將這些值用逗號隔開擺放
- List中的每個資料稱為元素 (element)

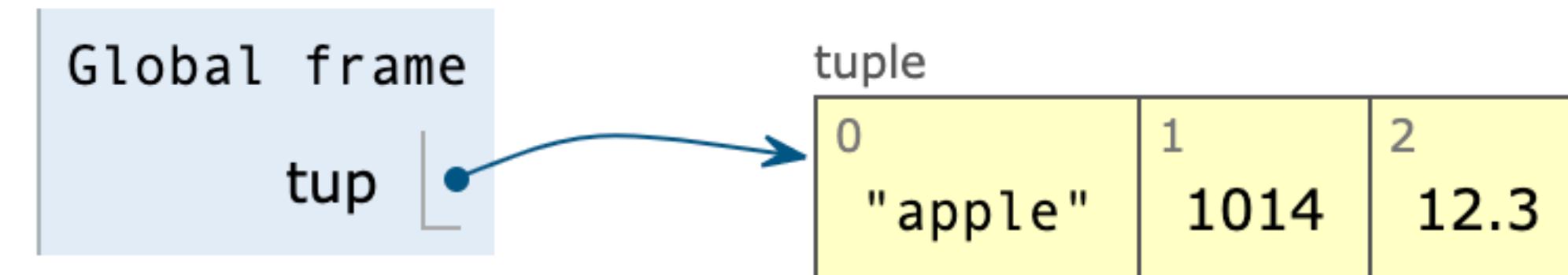


元組 (Tuple)

- 將不同資料以順序性的方式擺放的資料結構
- 然而與 list 最大不同在於 tuple一旦建立之後就不能更改其內容
- 資料產生後如不常更動使用tuple將更有效率

```
tup = ("apple", 1014, 12.3)
```

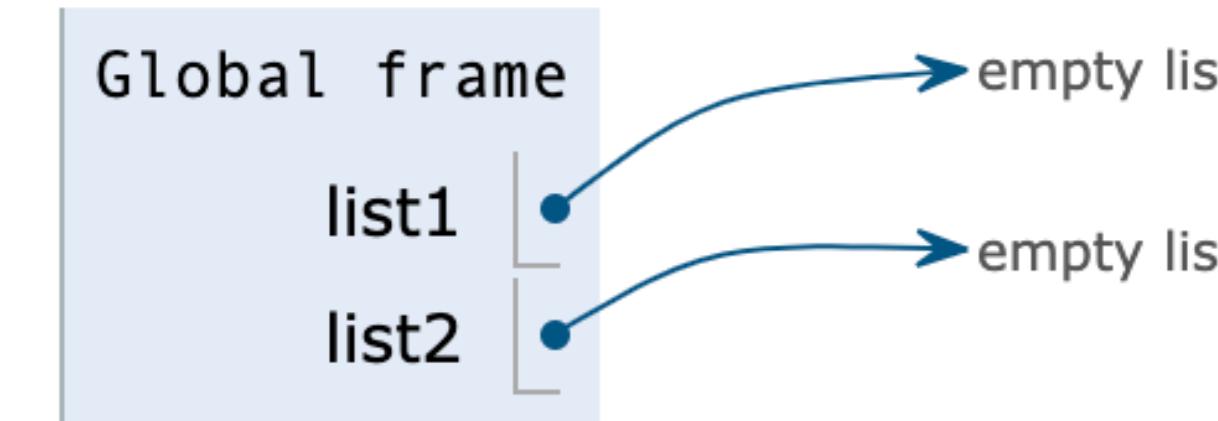
- 使用小括號，將這些值用逗號隔開擺放



List基本操作

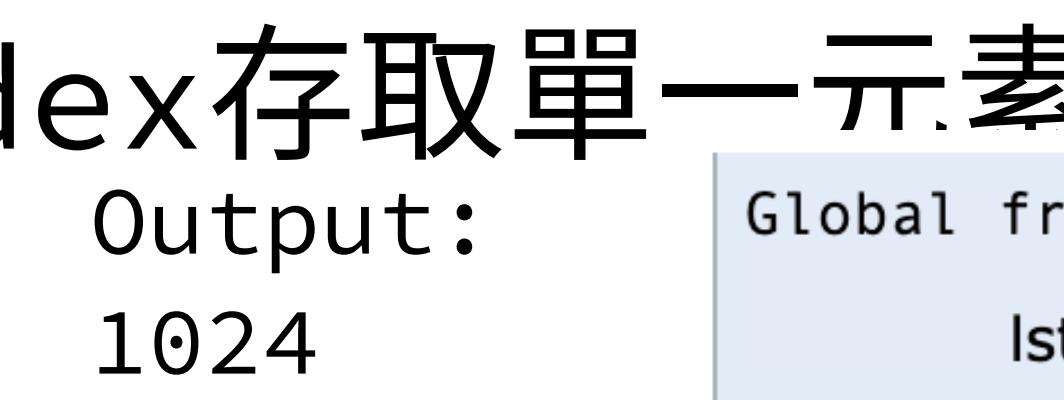
- 產生一個空的list

- `list1 = []` or
`list2 = list()`



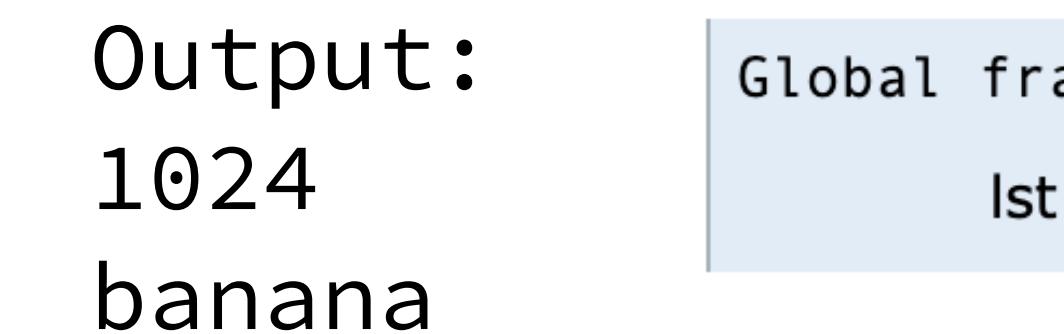
- 與字串相同，list也是使用index存取單一元素

```
lst = ["apple", 1014, 12.3]
print(lst[1])
```



- list產生之後可以更改其內容

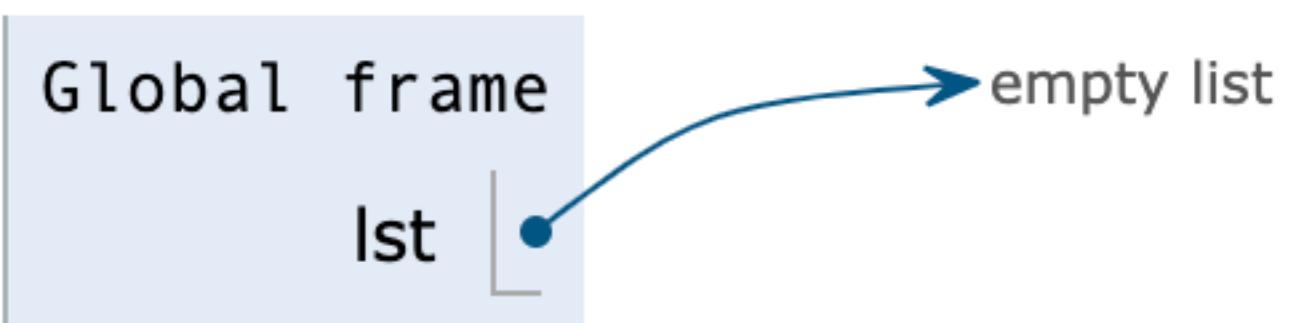
```
lst = ["apple", 1014, 12.3]
print(lst[1])
lst[1] = "banana"
print(lst[1])
```



List 的方法

- `list.append(x)`
 - 將x放置到list的最後

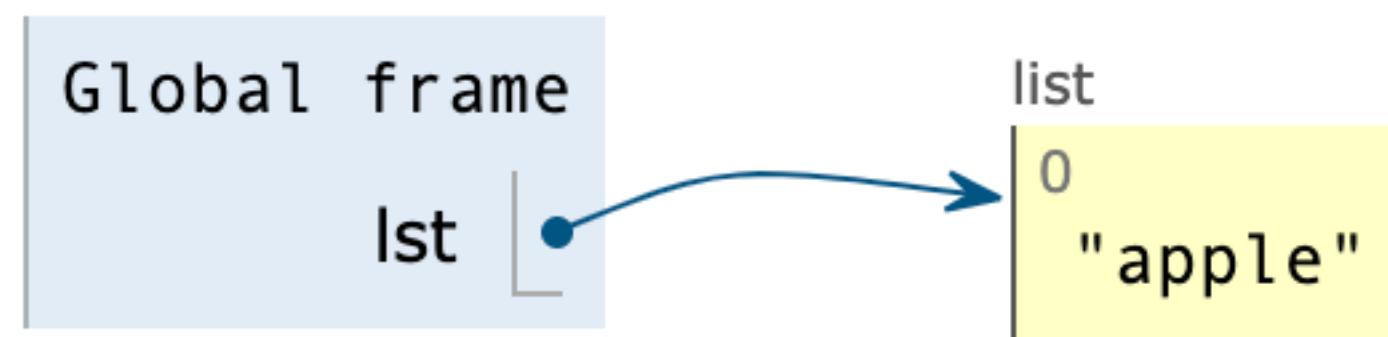
```
lst = list()
```



List 的方法

- `list.append(x)`
 - 將x放置到list的最後

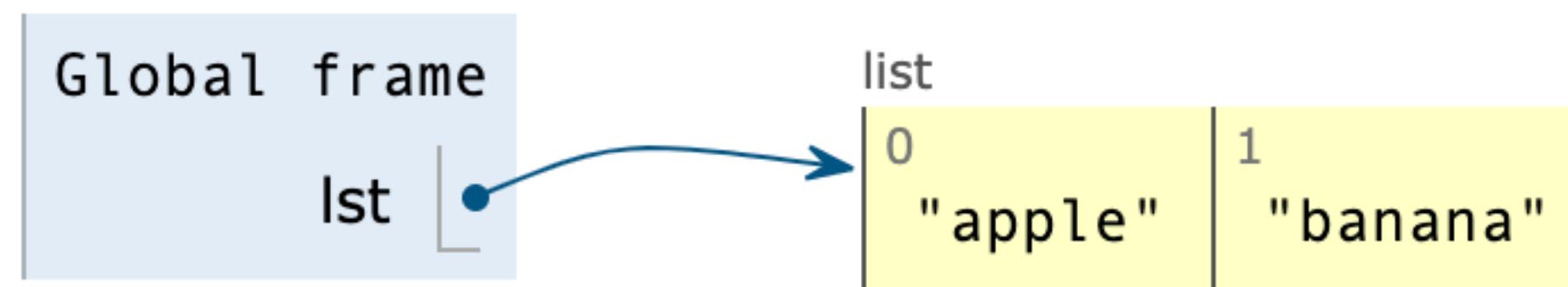
```
lst = list()  
lst.append("apple")
```



List 的方法

- `list.append(x)`
 - 將x放置到list的最後

```
lst = list()  
lst.append("apple")  
lst.append("banana")
```



List 的方法

- `list.insert(i, x)`
 - 將x放置到index i 之前

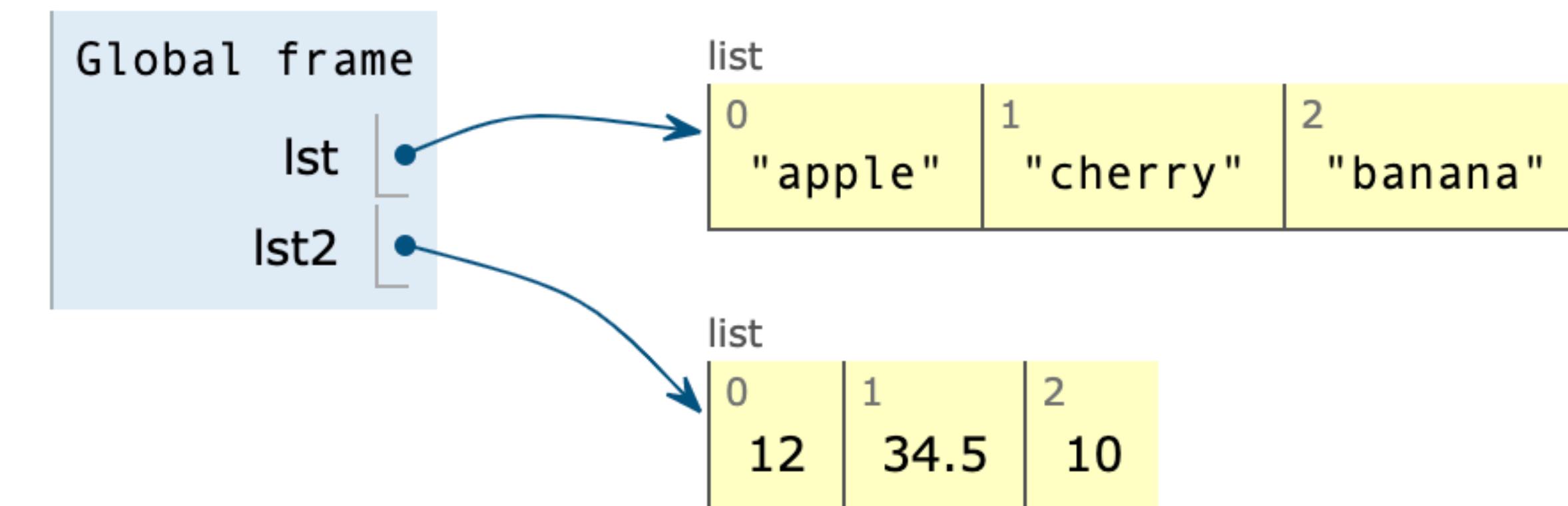
```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")
```



List 的方法

- `list.extend(iterable)`
 - 將 iterable 的資料展開並資序append至list

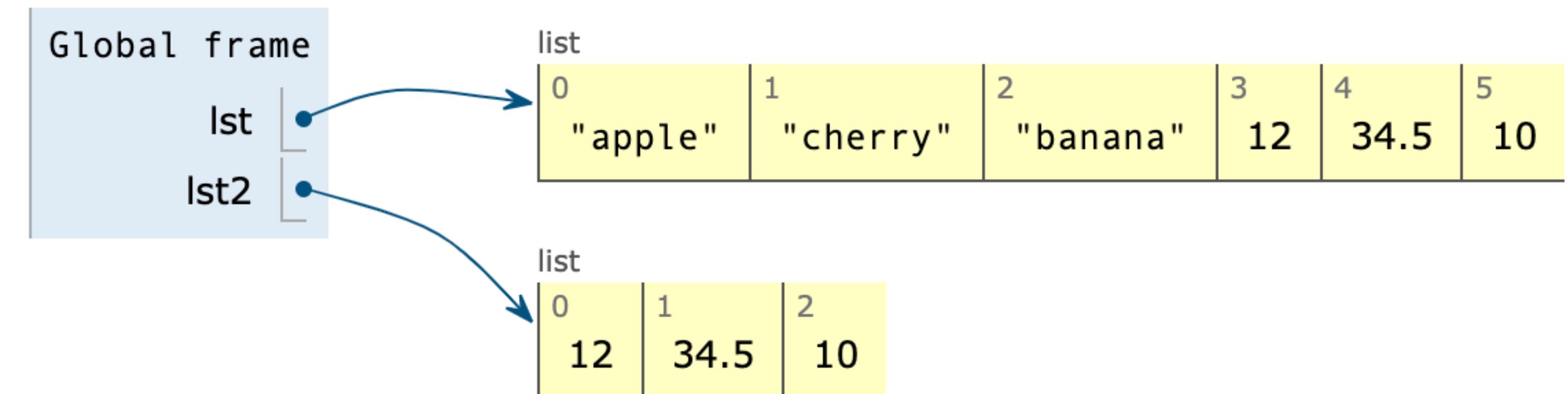
```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]
```



List 的方法

- `list.extend(iterable)`
 - 將 iterable 的資料展開並資料 append 至 list

```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)
```

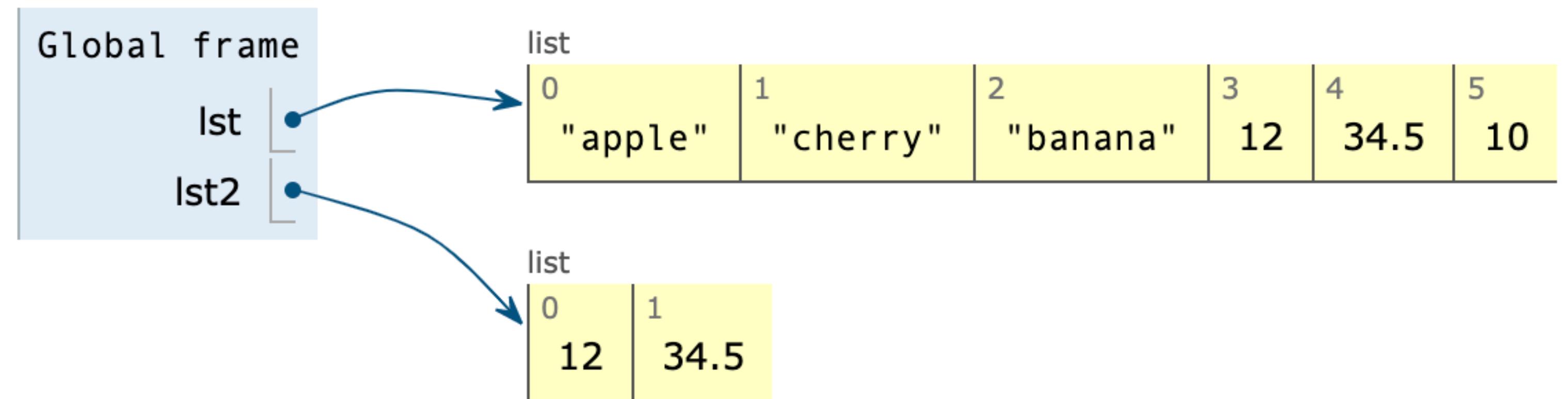


List 的方法

- `list.pop([i])`

- 將 `list` 的最後一個元素取出並回傳，如有給 `i` 則取出 `index i` 元素並回傳

```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)  
lst2.pop()
```

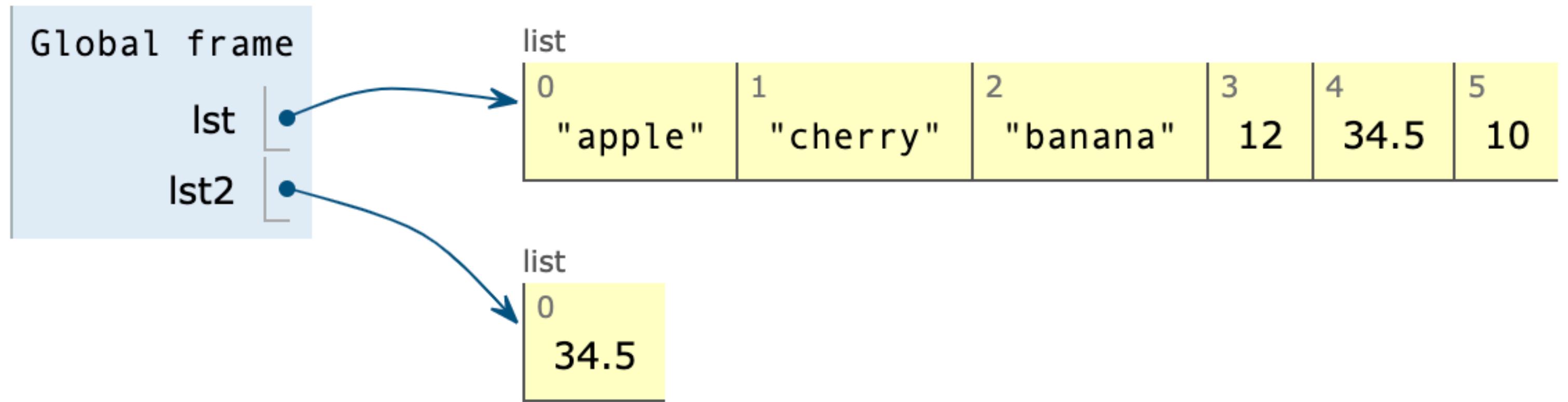


List 的方法

- `list.pop([i])`

- 將 `list` 的最後一個元素取出並回傳，如有給 `i` 則取出 `index i` 元素並回傳

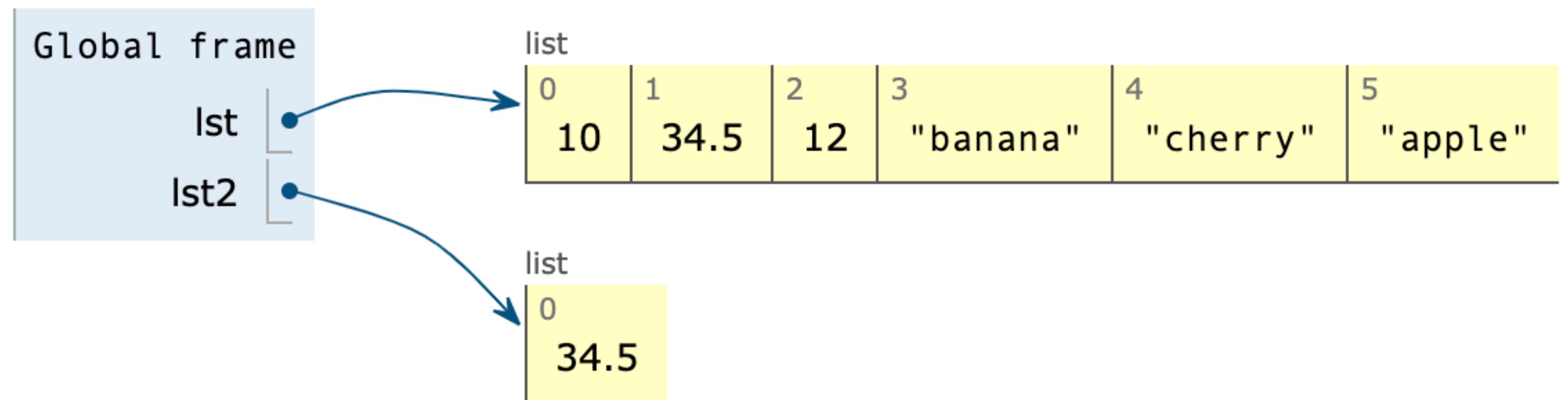
```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)  
lst2.pop()  
lst2.pop(0)
```



List 的方法

- `list.reverse()`
 - 將 list 前後翻轉

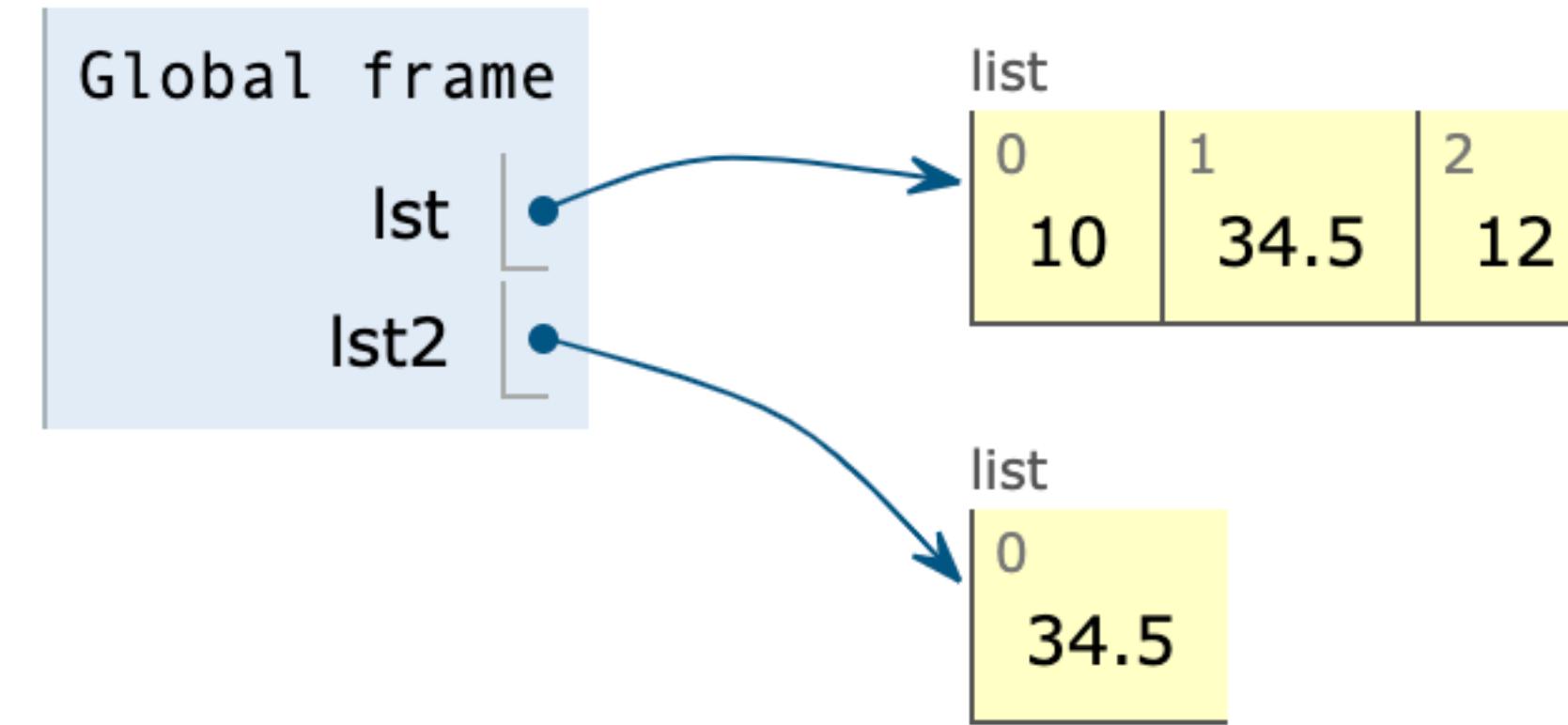
```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)  
lst2.pop()  
lst2.pop(0)  
lst.reverse()
```



List 的方法

- `list.remove(x)`
 - 將list中的x元素移除

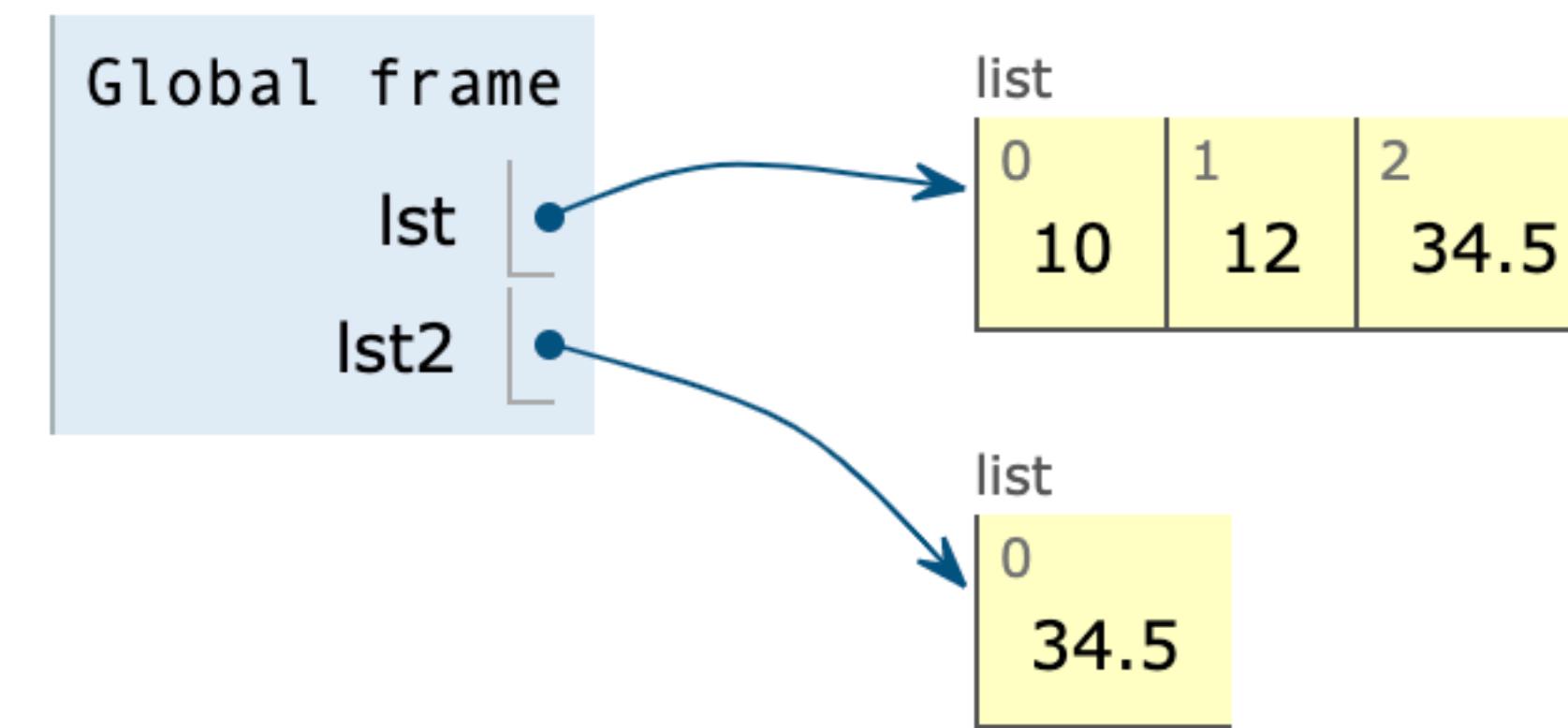
```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)  
lst2.pop()  
lst2.pop(0)  
lst.reverse()  
lst.remove("apple")  
lst.remove("banana")  
lst.remove("cherry")
```



List 的方法

- `list.sort()`
 - 將 list 依照大小排序

```
lst = list()  
lst.append("apple")  
lst.append("banana")  
lst.insert(1, "cherry")  
lst2 = [12, 34.5, 10]  
lst.extend(lst2)  
lst2.pop()  
lst2.pop(0)  
lst.reverse()  
lst.remove("apple")  
lst.remove("banana")  
lst.remove("cherry")  
lst.sort()
```

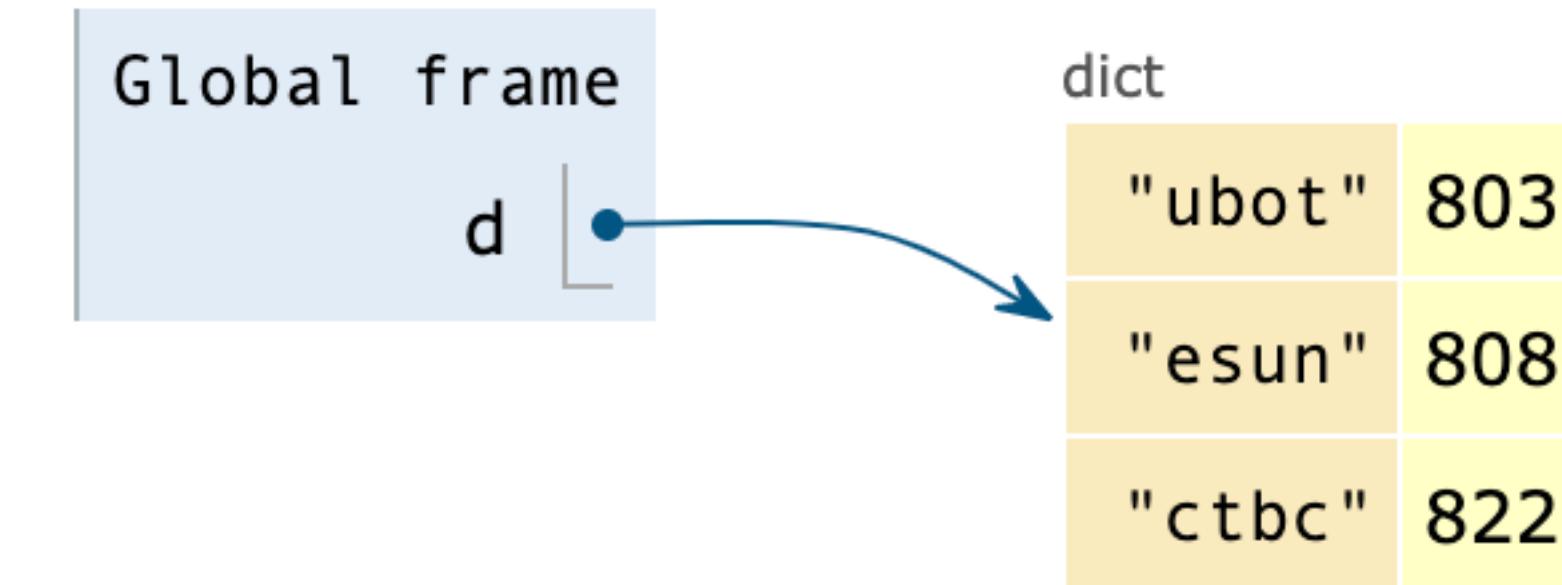


1-12
Dictionarys
(字典)

字典 (Dictionary)

- 收集資料的資料結構，然後其沒有一定順序
- 存取資料的方式是使用key

```
d = dict()  
d['ubot'] = 803  
d['esun'] = 808  
d['ctbc'] = 822
```

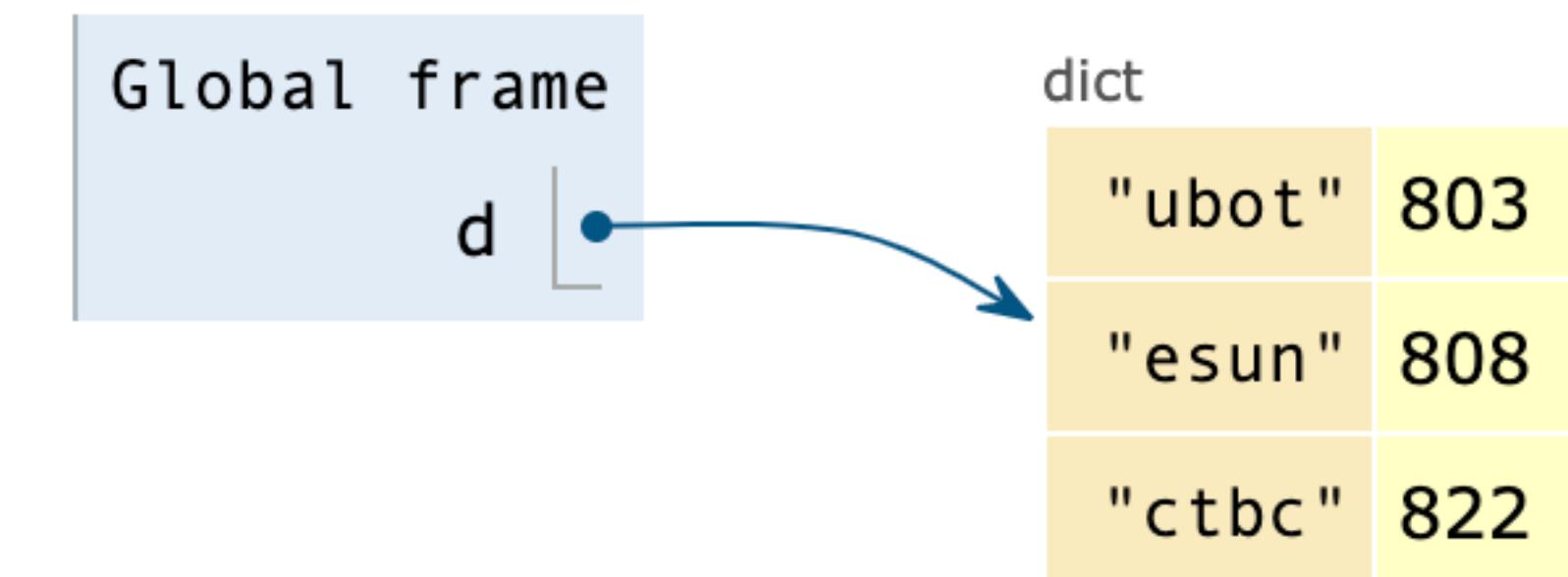


- 產生空字典後，在將資料加入
- 每一組資料稱為entry

字典 (Dictionary)

- 收集資料的資料結構，然後其沒有一定順序
- 存取資料的方式是使用key
- Key不能重複，產生之後也不能更改

d = Value可以更改
d = {"ubot": 803, "esun": 808, "ctbc": 822}



- 也可使用大括號產生字典
- 每一組entry以 key: value並用逗號隔開

Dictionary 基本操作

```
d = {"ubot": 803, "esun": 808, "ctbc": 822}  
print(d["esun"])  
print(d["post"])
```

- Key不存在時會產生錯誤

Output:

808

KeyError

Dictionary的方法

- `dict.get(key, [def])`
 - Key存在時回傳value，不存在時回傳None
 - 如有def參數，則回傳def

```
d = {"ubot": 803, "esun": 808, "ctbc": 822}  
print(d["esun"])  
print(d.get("post"))  
print(d.get("post", 0))
```

Output:
808
None
0

Dictionary的方法

- `dict.keys()`
 - 回傳所有keys的list
- `dict.values()`
 - 回傳所有values的list
- `dict.items()`
 - 將key-value的entry包裝成元組後集合成list回傳

Dictionary 的方法

```
d = {"ubot": 803, "esun": 808, "ctbc": 822}  
for k in d.keys():  
    print(k)  
print("====")  
for v in d.values():  
    print(v)  
print("====")  
for k, v in d.items():  
    print("K:", k, ", V:", v)
```

Output:

```
ubot  
esun  
ctbc  
=====  
803  
808  
822  
=====  
K: ubot , V: 803  
K: esun , V: 808  
K: ctbc , V: 822
```

Level Test :)

- 計算0~100，2 至 9分別的倍數的總和
- Advanced: 去除5開頭的數字

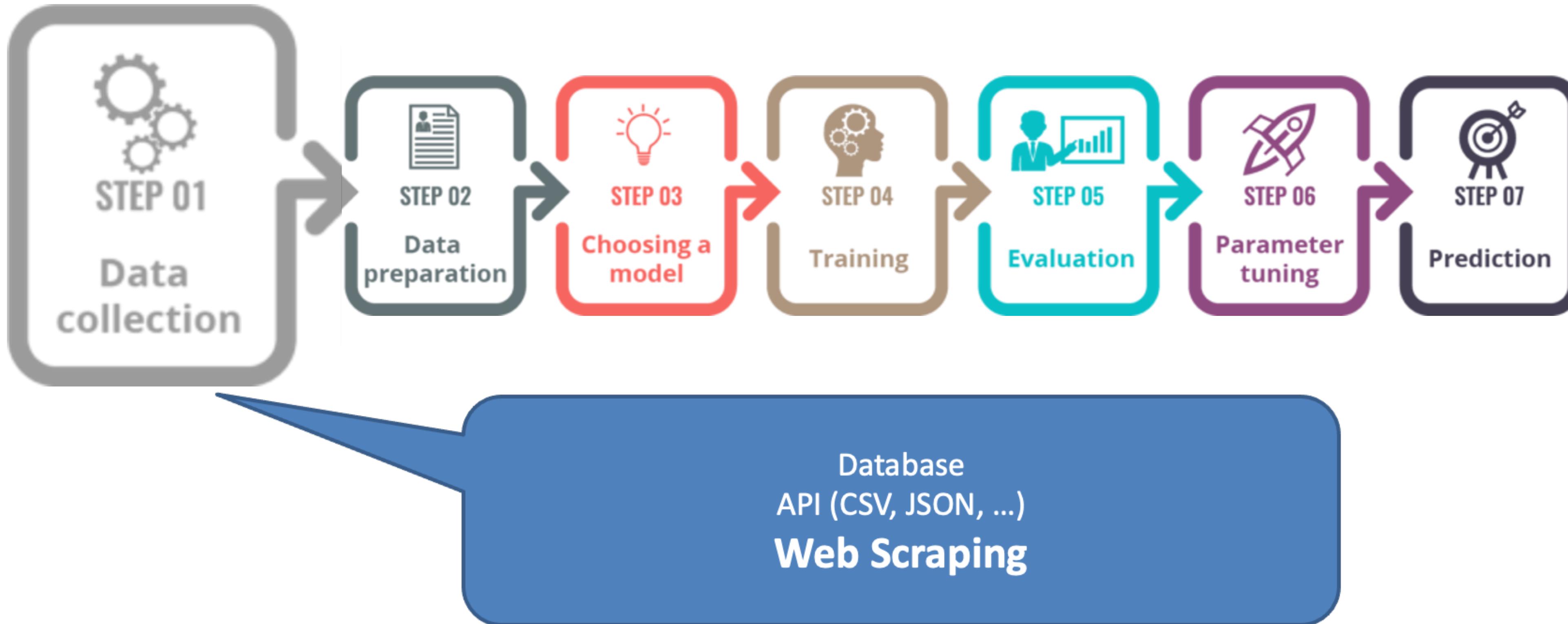
2-1

資料分析

Data Analysis

- 為了探索有用的資訊，瞭解資訊所代表的意義用以幫助決策的流程
 - 包含審查 (inspecting) 、清理 (cleansing) 、轉換 (transforming) 和資料模型化 (modeling data)
- 資料分析可分為幾種型態
 - 敘述統計 (descriptive statistics)
 - 平均、最大值、最小值等資料表現出來的統計資訊
 - 探索式資料分析 (**exploratory data analysis, EDA**)
 - 利用統計與視覺化等方法，瞭解資料樣貌與背後所代表的意義
 - 確定性資料分析 (**confirmatory data analysis, CDA**)
 - 利用推論統計的方法，檢定資料的所代表的意義是否符合

From Data to AI



[Google Cloud Platform](#)

<https://www.youtube.com/watch?v=nKW8Ndu7Mjw>

The 7 steps of machine learning (AI Adventures)

<https://medium.com/dataseries/7-steps-to-machine-learning-how-to-prepare-for-an-automated-future-78c7918cb35d>

.

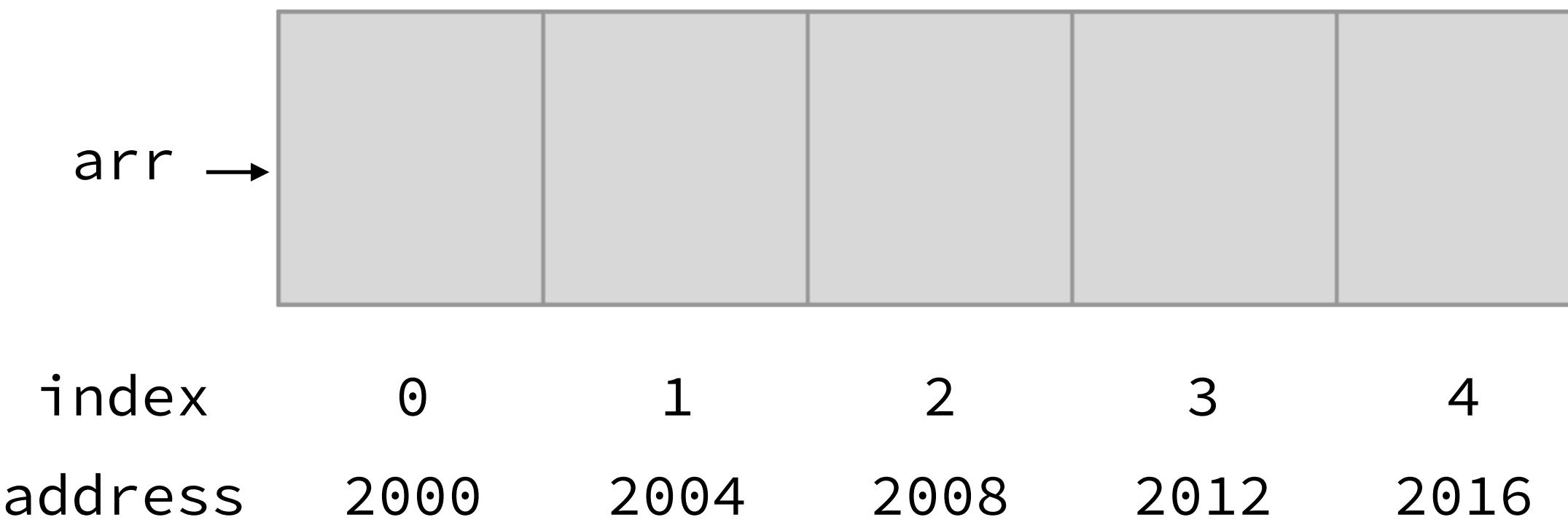
2-2



Module NumPy

- NumPy是一個支援科學運算的模組
 - 主要的物件為多維度的陣列 (multidimensional array, ndarray)
 - 可以快速進行多種運算，像是數學、邏輯、維度轉換、排序等基礎運算，又或是線性代數、統計運算與隨機模擬等

Array and List



ndarray object

- NumPy array建立時必須固定長度 (list可以改變長度) ，增加或刪減資料取得代之的是回傳一個新的 ndarray
- NumPy array的元素必須都是同一種資料型態 (例如都是整數或是都是浮點數)
- NumPy array適合對於大量的資料進行數學運算，比起python內建的資料結構可以用更少的語法達成且更為快速
- 有許多的模組建構於NumPy之上，例如資料分析的pandas，機器學習的SciPy等模組

Live Coding

- Import numpy as np
- 運算數度的比較範例

```
1 my_arr = np.arange(1000000)
2 my_list = list(range(1000000))
```

```
1 %%time
2 my_list3 = []
3 for _ in range(10):
4     for i in range(1000000):
5         my_list3.append(i * 2)

CPU times: user 983 ms, sys: 78.7 ms, total: 1.06 s
Wall time: 1.07 s
```

```
1 %%time
2 for _ in range(10):
3     my_list2 = [x * 2 for x in my_list]

CPU times: user 482 ms, sys: 93.7 ms, total: 576 ms
Wall time: 577 ms
```

```
1 %%time
2 for _ in range(10):
3     my_arr2 = my_arr * 2

CPU times: user 14.2 ms, sys: 12.5 ms, total: 26.7 ms
Wall time: 25.5 ms
```

建立 NumPy ndarray

Function	Description
<code>array</code>	將輸入的資料 (list, tuple,...) 建立一個新的ndarray
<code>arange</code>	與內建函數range一樣，但是是回傳ndarray，且能夠使用浮點數
<code>linspace</code>	再給定的範圍內建立給定個數（預設為50個）的ndarray
<code>ones</code>	給定個數，建立內容都為1的ndarray
<code>zeros</code>	給定個數，建立內容都為0的ndarray
<code>empty</code>	給定個數，建立內容都為空的ndarray
<code>full</code>	給定維度與值，建立內容都為該值的該維度ndarray
<code>eye, identity</code>	建立一個N*N的單位矩陣（只有對角線為1，其餘為0）

ndarray 基本屬性與型態

In : array1.shape # 用tuple表示的每一個維度的size

out: (3, 3)

In : array1.dtype # array內元素的資料型態

out: dtype('int64')

In : array1.ndim # 維度個數

out: 2

In : array1.size # 元素個數

out: 9

The screenshot shows a Jupyter Notebook interface. On the left, there is a code cell containing Python code to create a 3x3 NumPy array named 'array1'. On the right, there is an output cell showing the array's contents in a grid format.

Code in cell 1:

```
1 import numpy as np
2
3 array1 = np.array([[0, 1, 2],
4                   [3, 4, 5],
5                   [6, 7, 8]])
```

Output cell 1:

	col 0	col 1	col 2
row 0	0	1	2
row 1	3	4	5
row 2	6	7	8

ndarray 運算

- `array1 + array1`
- `array1 - array1`
- `array1 * array1`
- `array1 ** 2`
- `array * 10`
- `1 / array1`
- `array > 5`

	col 0	col 1	col 2
row 0	0	1	2
row 1	3	4	5
row 2	6	7	8

Live Coding

- 上頁投影片內容

Indexing and Slicing (1)

- `array1 = np.arange(6)`

array1	0	1	2	3	4	5
index	0	1	2	3	4	5

- `array1[4]`
- `array1[2:5]`
- `array1[2:5] = 12`

Indexing and Slicing (1)

- `array1 = np.arange(6)`

- **array1[4]**

array1	0	1	2	3	4	5
index	0	1	2	3	4	5

- `array1[2:5]`

- `array1[2:5] = 12`

Indexing and Slicing (1)

- `array1 = np.arange(6)`

- `array1[4]`

- **`array1[2:5]`**

- `array1[2:5] = 12`

array1	0	1	2	3	4	5
index	0	1	2	3	4	5

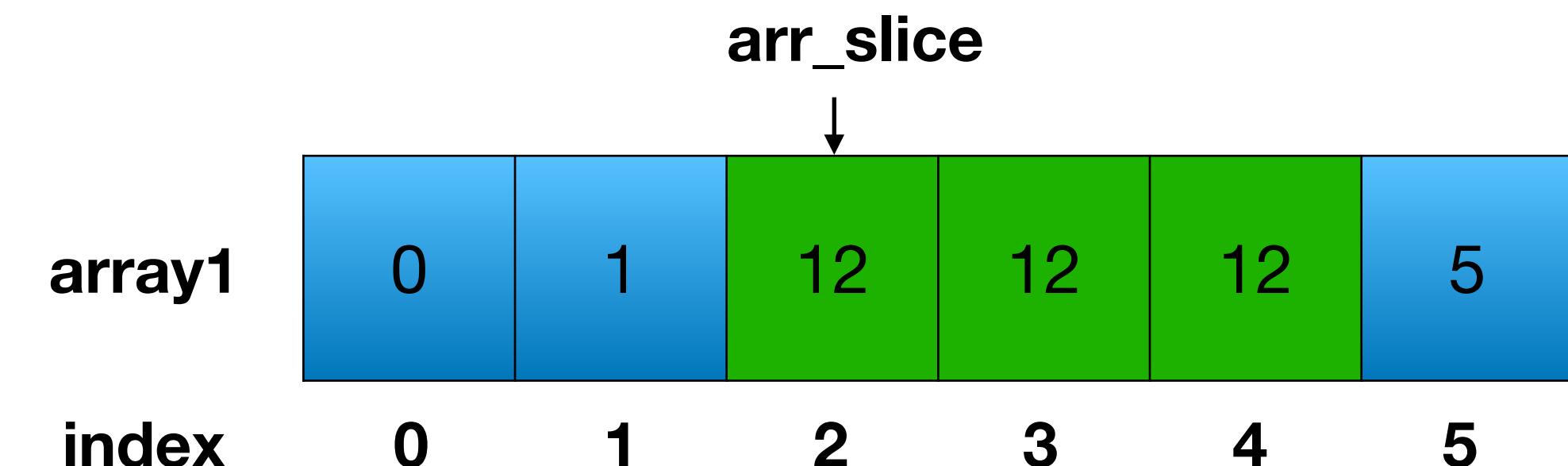
Indexing and Slicing (1)

- `array1 = np.arange(6)`
- `array1[4]`
- `array1[2:5]`
- `array1[2:5] = 12`

array1	0	1	12	12	12	5
index	0	1	2	3	4	5

Indexing and Slicing (2)

- **arr_slice = array1[2:5]**



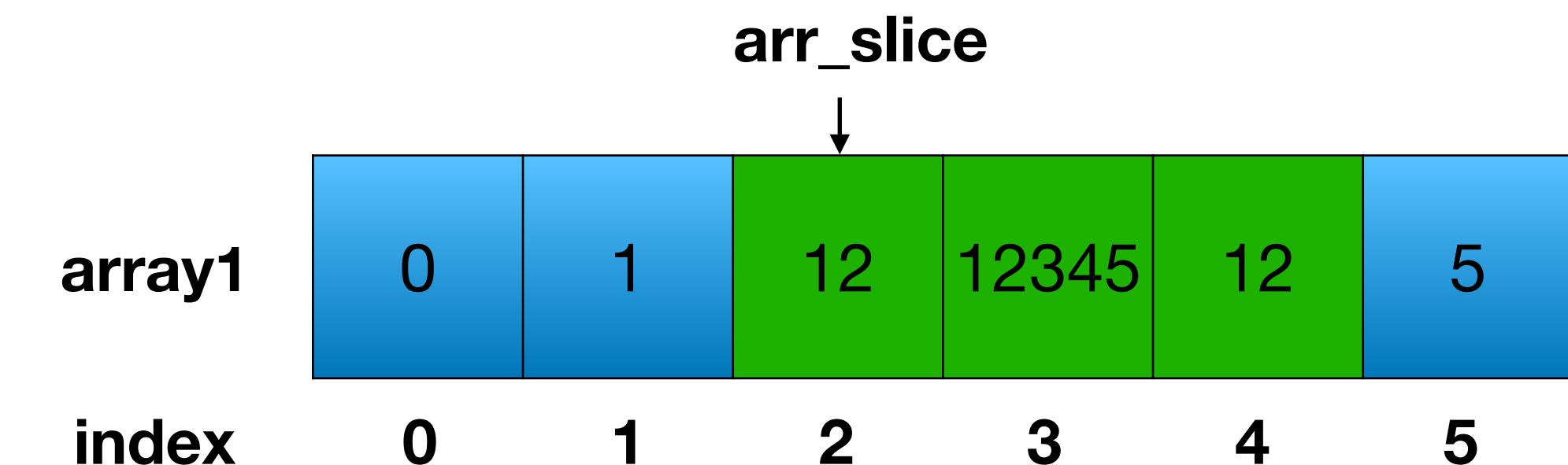
- `arr_slice[1] = 12345`
- `arr_slice[:] = 100`

Indexing and Slicing (2)

- `arr_slice = array1[2:5]`

- **`arr_slice[1] = 12345`**

- `arr_slice[:] = 100`

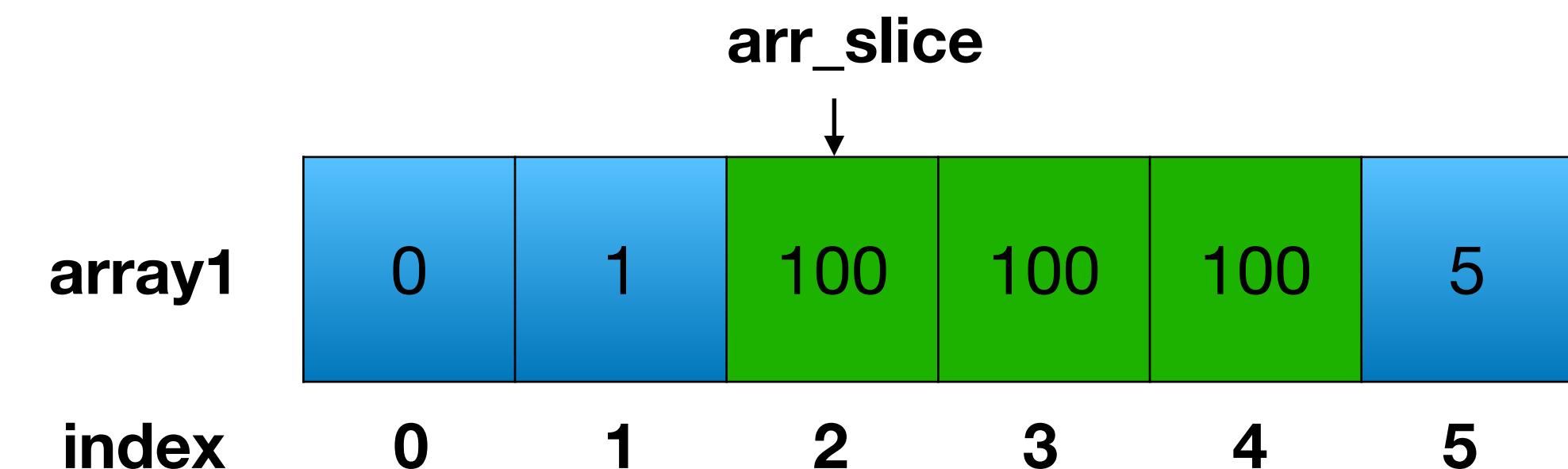


Indexing and Slicing (2)

- `arr_slice = array1[2:5]`

- `arr_slice[1] = 12345`

- `arr_slice[:] = 100`



Indexing and Slicing (3)

- `arr2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])`
- `np.arange(9).reshape((3, 3))`

			axis 1	
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d[2]

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d[0][2]
- arr2d[0, 2]

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- arr2d[:2]

- 選擇前兩 row

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- arr2d[:2, 1:]

- 選擇前兩row和第一col到最後

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- arr2d[1, :2], shape : (2,)
 - 混合indexing和slicing時，維度會下降，
 - arr2d[1:2, :2]一樣位置內容，但shape: (1, 2)

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d[:2, 2], shape:(2,)
- arr2d[:2, 2:], shape:(2, 1)

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

Indexing and Slicing (3)

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- arr2d[:, :1]

- : 代表該維度全部

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d % 2 == 0

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	True	False	True
	row 1	False	True	False
row 2	axis 0	True	False	True

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

			axis 1
			col 0 col 1 col 2
row 0	True	False	True
	False	True	False
	True	False	True

- arr2d[arr2d % 2 == 0]

- out:array([0, 2, 4, 6, 8])

			axis 1
			col 0 col 1 col 2
row 0	0	1	2
	3	4	5
	6	7	8

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- names = np.array(['Steve', 'Joe', 'Steve'])
- names == 'Steve'

Steve	Joe	Steve
-------	-----	-------

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

- arr2d[names == 'Steve']

- 必須跟該維度長度一樣

Steve	Joe	Steve
-------	-----	-------

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d[names == 'Steve', 1:]

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	6	7	8	

Steve	Joe	Steve
-------	-----	-------

Boolean Indexing

- arr2d = np.array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
- arr2d[~(names == 'Steve')]

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8

Steve	Joe	Steve
-------	-----	-------

Boolean Indexing

- arr2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])

- arr2d[(names == 'Steve') | (arr2d % 2 == 0)]

- 邏輯運算 or

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	True	False	True
	row 1	False	True	False
	row 2	True	False	True

Steve	Joe	Steve
-------	-----	-------

- arr2d[(names == 'Steve') | (arr2d % 2 == 0)]

- 邏輯運算 or

- arr2d[(names == 'Steve') | (arr2d % 2 == 0)]

- 邏輯運算 or

- arr2d[(names == 'Steve') | (arr2d % 2 == 0)]

- 邏輯運算 or

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
	row 2	6	7	8

0	1	2
3	4	5
6	7	8

Boolean Indexing

- arr2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	True	False	True
	row 1	False	True	False
row 2	axis 0	True	False	True
	row 1	Steve	Joe	Steve

- arr2d[(names == 'Steve') & (arr2d % 2 == 0)]

		axis 1		
		col 0	col 1	col 2
row 0	axis 0	0	1	2
	row 1	3	4	5
row 2	axis 0	6	7	8
	row 1	Steve	Joe	Steve

Transposing

- `arr = np.arange(15).reshape((3, 5))`

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

0	5	10
1	6	11
2	7	12
3	8	13
4	9	14

- `arr.T` or `arr.transpose()`

Matrix Multiplication

- arr_ma = np.dot(arr, arr.T)
- arr @ arr.T

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

0	5	10
1	6	11
2	7	12
3	8	13
4	9	14

Rule For Matrix Multiplication



$$A \cdot B = AB$$

$m \times n$ $n \times p$ $m \times p$

↓ ↓ ↓

Equal

Dimensions of AB

2 × 2 Matrix Multiplication

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \times \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_1a_2 + b_1c_2 & a_1b_2 + b_1d_2 \\ c_1a_2 + d_1c_2 & c_1b_2 + d_1d_2 \end{bmatrix}$$



30	80	130
80	255	430
130	430	730

Mathematical and Statistical Methods

- arr = np.arange(15).reshape((3, 5))

In :arr.mean()

Out:7.0

In :np.mean(arr)

Out:7.0

In :arr.sum()

Out:105

					axis 1
					0
					1
					2
					3
					4
					5
					6
					7
					8
					9
					10
					11
					12
					13
					14

Mathematical and Statistical Methods

- arr = np.arange(15).reshape((3, 5))

In :arr.mean()

Out:7.0

In :arr.mean(axis=0)

Out:array([5., 6., 7., 8., 9.])

In :arr.mean(axis=1)

Out:array([2., 7., 12.])

					axis 1
					0
					1
					2
					3
					4
					axis 0
					5
					6
					7
					8
					9
					10
					11
					12
					13
					14

Mathematical and Statistical Methods

Function	Description
<code>sum</code>	根據維度的元素總和
<code>mean</code>	算術平均
<code>std, var</code>	標準差與變異數
<code>min, max</code>	給定個數，建立內容都為1的ndarray
<code>cumsum</code>	累積總和，起始值為0
<code>cumprod</code>	累積連乘，起始值為1

2-3



Module pandas

- 為了方便與快速進行資料清理與分析的模組
- 經常與許多數值運算、資料分析、視覺化等工具一起搭配使用
- 採用像是NumPy array的計算方式，最大不同在於使用表格化資料，也可以將不同型態資料整合在一起

Live Coding

- Import pandas as pd

pandas Data Structure - Series

- Series是一個一維的順序性資料，有相同的資料型態，並伴隨著資料index的資料標籤，預設為從0開始的數字

```
1 obj = pd.Series([1, -1, 3, 4])
2 obj
```

0	1
1	-1
2	3
3	4

dtype: int64

- 常使用描述性質的資料標籤的index

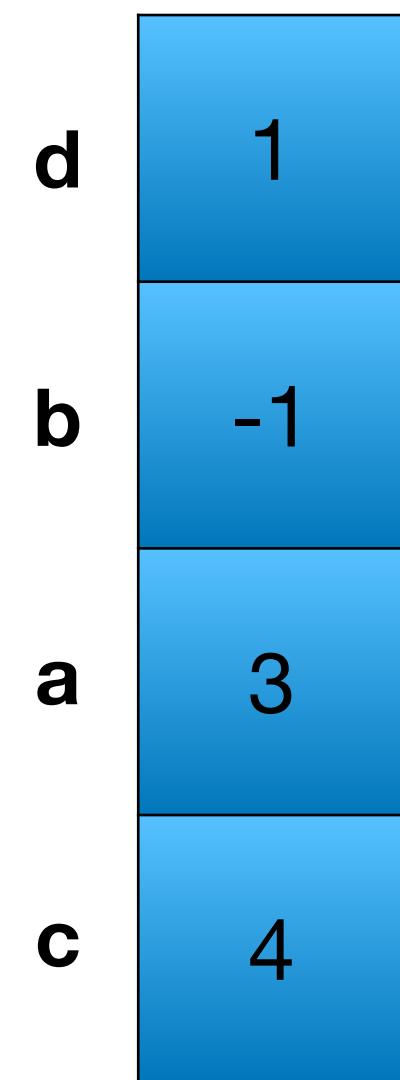
```
1 obj2 = pd.Series([1, -1, 3, 4], index=['d', 'b', 'a', 'c'])
2 obj2
```

d	1
b	-1
a	3
c	4

dtype: int64

Series

- obj2 = pd.Series([1, -1, 3, 4], index=['d', 'b', 'a', 'c'])

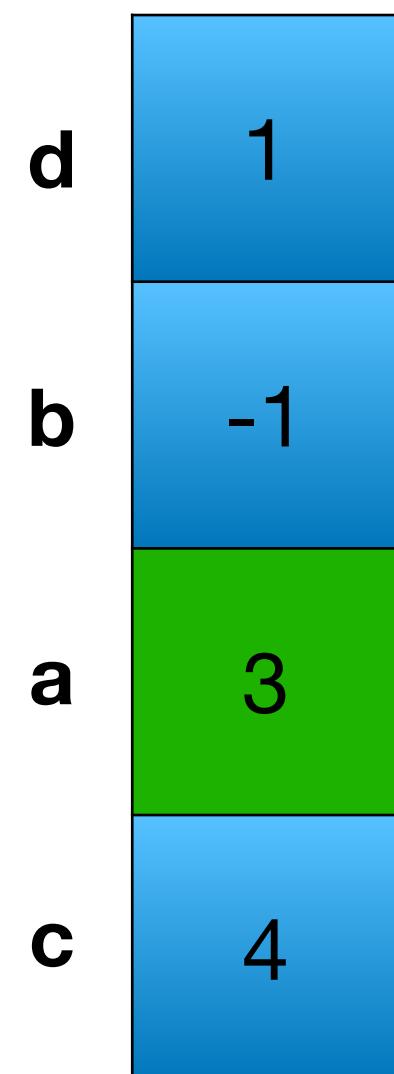


Series

- obj2 = pd.Series([1, -1, 3, 4], index=['d', 'b', 'a', 'c'])

In :obj2['a']

Out:3



Series

- `obj2 = pd.Series([1, -1, 3, 4], index=['d', 'b', 'a', 'c'])`

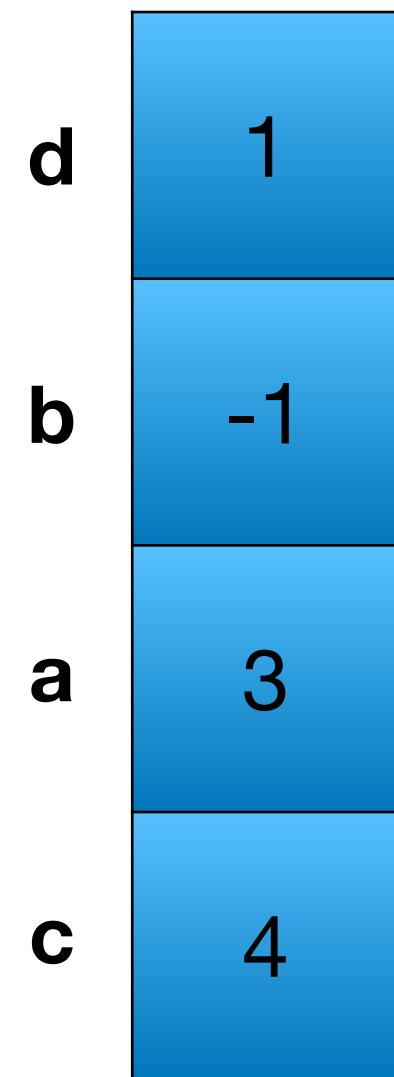
In :`obj2[['c', 'a', 'd']]`

Out: c 4

a 3

d 1

dtype: int64



Live Coding

```
1 obj2[obj2 > 1]
```

```
a    3  
c    4  
dtype: int64
```

```
1 obj2 * 2
```

```
d    2  
b   -2  
a    6  
c    8  
dtype: int64
```

```
1 import numpy as np  
2  
3 np.abs(obj2)
```

```
d    1  
b    1  
a    3  
c    4  
dtype: int64
```

```
1 'a' in obj2
```

True

```
1 'e' in obj2
```

False

Live Coding

- 利用dict建立series
- 判斷資料是否為空

```
1 number = {"004": "台灣銀行", "007": "彰化銀行", "812": "台新銀行"}  
2 obj3 = pd.Series(number)  
3 obj3
```

```
004    台灣銀行  
007    彰化銀行  
812    台新銀行  
dtype: object
```

```
1 banks = ["003", "004", "007", "812"]  
2 obj4 = pd.Series(number, index=banks)
```

```
003    NaN  
004    台灣銀行  
007    彰化銀行  
812    台新銀行  
dtype: object
```

```
1 pd.isna(obj4)
```

```
003    True  
004    False  
007    False  
812    False  
dtype: bool
```

```
1 pd.isnull(obj4)
```

```
003    True  
004    False  
007    False  
812    False  
dtype: bool
```

```
1 pd.notna(obj4)
```

```
003    False  
004    True  
007    True  
812    True  
dtype: bool
```

Live Coding

```
1 obj4.name = "銀行代碼對照表"  
2 obj4.index.name = "代碼"  
3 obj4
```

代碼

```
003     NaN  
004     台灣銀行  
007     彰化銀行  
812     台新銀行
```

```
Name: 銀行代碼對照表, dtype: object
```

Live Coding

- numpy-like operation

```
1 employees = {"004": 100, "007": 80, "812": 95}
2 obj5 = pd.Series(employees, index=banks)
3 obj5
```

```
003      NaN
004    100.0
007     80.0
812    95.0
dtype: float64
```

```
1 obj5 * 2
```

```
003      NaN
004    200.0
007   160.0
812   190.0
dtype: float64
```

```
1 obj5 + obj5
```

```
003      NaN
004    200.0
007   160.0
812   190.0
dtype: float64
```

Live Coding

- Update index

```
1 | obj
```

```
0    1  
1   -1  
2    3  
3    4  
dtype: int64
```

```
1 | obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']  
2 | obj
```

```
Bob      1  
Steve   -1  
Jeff      3  
Ryan      4  
dtype: int64
```

pandas Data Structure - DataFrame

- 將資料以表格形式表達的資料
 - 具有順序性
 - 每一個column代表一種資料，不同column為不同資料型態
 - 每一個column為一個Series
 - 每個row跟column都有index

Live Coding

```
1 data = {"city": ["Taipei", "Taipei", "Taipei", "Taoyuan", "Taoyuan", "Taoyuan"],  
2     "year": [2021, 2020, 2019, 2021, 2020, 2019],  
3     "pop": [2.52, 2.60, 2.64, 2.27, 2.26, 2.24]}  
4 frame = pd.DataFrame(data)  
5 frame
```

	city	year	pop
0	Taipei	2021	2.52
1	Taipei	2020	2.60
2	Taipei	2019	2.64
3	Taoyuan	2021	2.27
4	Taoyuan	2020	2.26
5	Taoyuan	2019	2.24

Live Coding

- 取得前後部分資料，用以大量資料部分資訊
- 更改column順序

```
1 data = {"city": ["Taipei", "Taipei", "Taipei", "Taoyuan", "Taoyuan", "Taoyuan"],  
2     "year": [2021, 2020, 2019, 2021, 2020, 2019],  
3     "pop": [2.52, 2.60, 2.64, 2.27, 2.26, 2.24]}  
4 frame = pd.DataFrame(data)  
5 frame
```

	city	year	pop
0	Taipei	2021	2.52
1	Taipei	2020	2.60
2	Taipei	2019	2.64
3	Taoyuan	2021	2.27
4	Taoyuan	2020	2.26
5	Taoyuan	2019	2.24

```
1 frame.head()
```

	city	year	pop
0	Taipei	2021	2.52
1	Taipei	2020	2.60
2	Taipei	2019	2.64
3	Taoyuan	2021	2.27
4	Taoyuan	2020	2.26

```
1 frame.tail()
```

	city	year	pop
1	Taipei	2020	2.60
2	Taipei	2019	2.64
3	Taoyuan	2021	2.27
4	Taoyuan	2020	2.26
5	Taoyuan	2019	2.24

Live Coding

```
1 frame2 = pd.DataFrame(data, columns=['year', 'city', 'pop', 'debt'],
2                         index=['one', 'two', 'three', 'four', 'five', 'six'])
3 frame2
```

	year	city	pop	debt
one	2021	Taipei	2.52	NaN
two	2020	Taipei	2.60	NaN
three	2019	Taipei	2.64	NaN
four	2021	Taoyuan	2.27	NaN
five	2020	Taoyuan	2.26	NaN
six	2019	Taoyuan	2.24	NaN

Live Coding

- 使用像是dict方式取得該col內容為Series
- Column名稱必須跟python中的變數規則一樣才可以直接使用

```
1 frame2["city"]
```

```
one      Taipei
two      Taipei
three    Taipei
four     Taoyuan
five     Taoyuan
six      Taoyuan
Name: city, dtype: object
```

```
1 frame2.year
```

```
one      2021
two      2020
three    2019
four     2021
five     2020
six      2019
Name: year, dtype: int64
```

```
1 frame2[["city", "pop"]]
```

	city	pop
one	Taipei	2.52
two	Taipei	2.60
three	Taipei	2.64
four	Taoyuan	2.27
five	Taoyuan	2.26
six	Taoyuan	2.24

Live Coding

```
1 frame2["debt"] = np.arange(6.)  
2 frame2
```

	year	city	pop	debt
one	2021	Taipei	2.52	0.0
two	2020	Taipei	2.60	1.0
three	2019	Taipei	2.64	2.0
four	2021	Taoyuan	2.27	3.0
five	2020	Taoyuan	2.26	4.0
six	2019	Taoyuan	2.24	5.0

```
1 frame2["density"] = frame2["pop"] > 2.5
```

```
1 frame2
```

	year	city	pop	debt	density
one	2021	Taipei	2.52	0.0	True
two	2020	Taipei	2.60	1.0	True
three	2019	Taipei	2.64	2.0	True
four	2021	Taoyuan	2.27	3.0	False
five	2020	Taoyuan	2.26	4.0	False
six	2019	Taoyuan	2.24	5.0	False

Live Coding

```
1 frame2
```

	year	city	pop	debt
one	2021	Taipei	2.52	NaN
two	2020	Taipei	2.60	NaN
three	2019	Taipei	2.64	NaN
four	2021	Taoyuan	2.27	NaN
five	2020	Taoyuan	2.26	NaN
six	2019	Taoyuan	2.24	NaN

```
1 frame2.set_index("year")
```

year	city	pop	debt
2021	Taipei	2.52	NaN
2020	Taipei	2.60	NaN
2019	Taipei	2.64	NaN
2021	Taoyuan	2.27	NaN
2020	Taoyuan	2.26	NaN
2019	Taoyuan	2.24	NaN

```
1 frame2.to_numpy()
```

```
array([[2021, 'Taipei', 2.52, nan],  
       [2020, 'Taipei', 2.6, nan],  
       [2019, 'Taipei', 2.64, nan],  
       [2021, 'Taoyuan', 2.27, nan],  
       [2020, 'Taoyuan', 2.26, nan],  
       [2019, 'Taoyuan', 2.24, nan]], dtype=object)
```

Live Coding

```
1 obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])  
2 obj
```

```
d    4.5  
b    7.2  
a   -5.3  
c    3.6  
dtype: float64
```

```
1 obj2 = obj.reindex(["a", "b", "c", "d", "e"])  
2 obj2
```

```
a   -5.3  
b    7.2  
c    3.6  
d    4.5  
e    NaN  
dtype: float64
```

Selection on DataFrame with loc and iloc

Type	Notes
<code>df[val]</code>	選取單一column或一個序列的column; 或是slice選擇row以及boolean array選擇row
<code>df.loc[val]</code>	利用index label選擇單一或是多個row
<code>df.loc[:, val]</code>	利用label選擇單一或是多個column
<code>df.loc[val1, val2]</code>	利用label選rows與columns
<code>df.iloc[where]</code>	利用數字index位置選擇單一row或是多個row
<code>df.iloc[:, where]</code>	利用數字index位置選擇單一column或是多個column
<code>df.iloc[where_i, where_j]</code>	利用數字index位置選擇rows與columns

Live Coding

```
1 obj2
```

```
a -5.3  
b 7.2  
c 3.6  
d 4.5  
e NaN  
dtype: float64
```

```
1 mod2 = np.arange(5) % 2 == 0
```

```
1 obj2[mod2]
```

```
a -5.3  
c 3.6  
e NaN  
dtype: float64
```

```
1 obj2[1:3]
```

```
b 7.2  
c 3.6  
dtype: float64
```

Summarizing and Computing Descriptive Statistics

Method	Description
<code>count</code>	不是NaN的資料個數
<code>describe</code>	計算Series或是DataFrame基本統計資訊
<code>min, max</code>	計算最大最小值
<code>idxmin, idxmax</code>	計算最大最小值的label
<code>quantile</code>	計算四分位數
<code>sum</code>	計算總和

Live Coding

```
1 df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5], [np.nan, np.nan], [0.75, -1.3]],  
2                 index=['a', 'b', 'c', 'd'],  
3                 columns=['one', 'two'])  
4 df
```

	one	two
a	1.40	NaN
b	7.10	-4.5
c	NaN	NaN
d	0.75	-1.3

```
1 df.sum()  
  
one    9.25  
two   -5.80  
dtype: float64
```

```
1 df.sum(axis=0)  
  
one    9.25  
two   -5.80  
dtype: float64
```

```
1 df.sum(axis=1)  
  
a     1.40  
b     2.60  
c     0.00  
d    -0.55  
dtype: float64
```

Live Coding

```
1 obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
```

```
1 obj.unique()
```

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

```
1 obj.value_counts()
```

```
a    3  
c    3  
b    2  
d    1  
dtype: int64
```

3-1

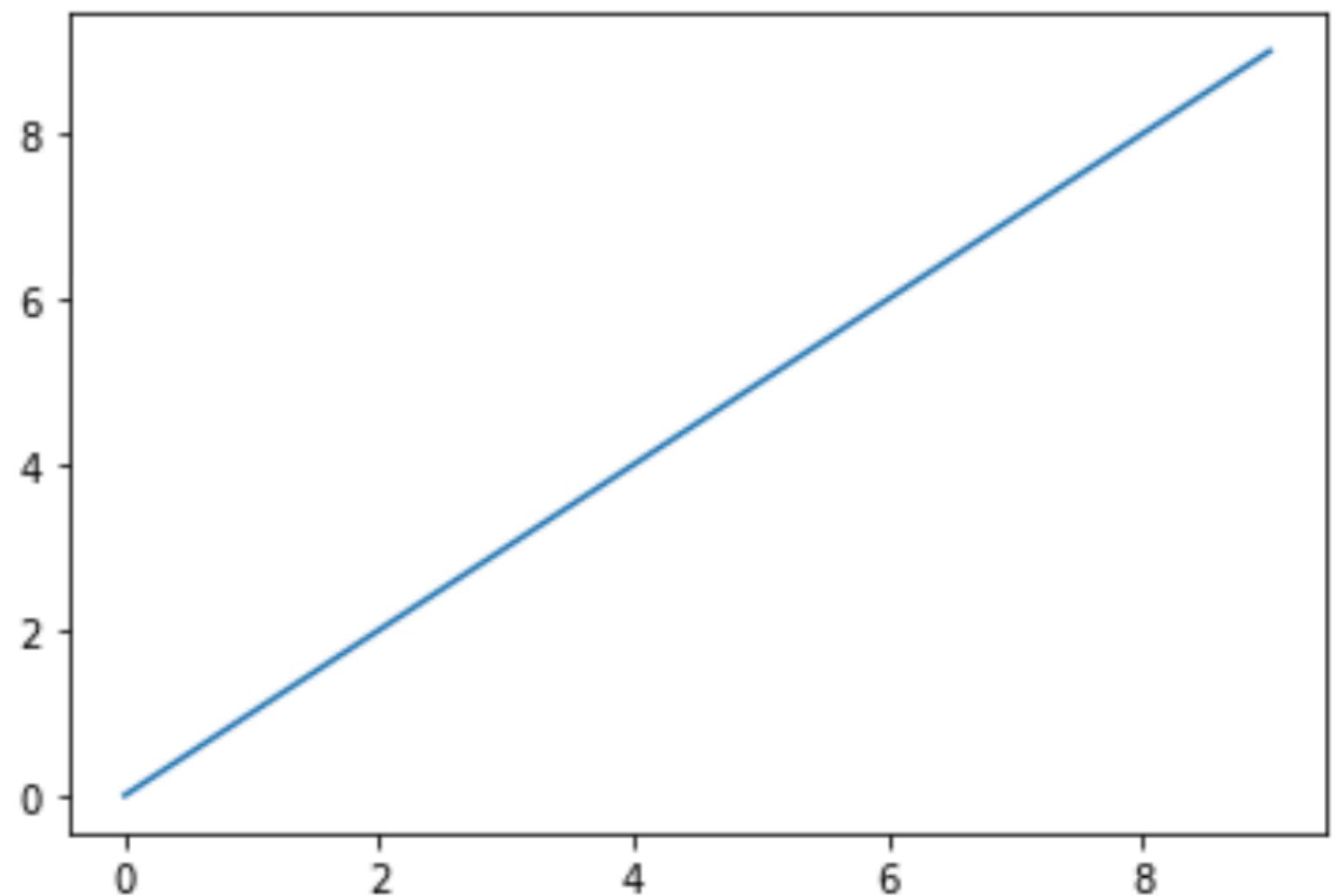
matplotlib

Module matplotlib

- 在探索資料的過程中，利用圖形化可以發現許多資料
 - 例如偏離值或是資料轉換的方法

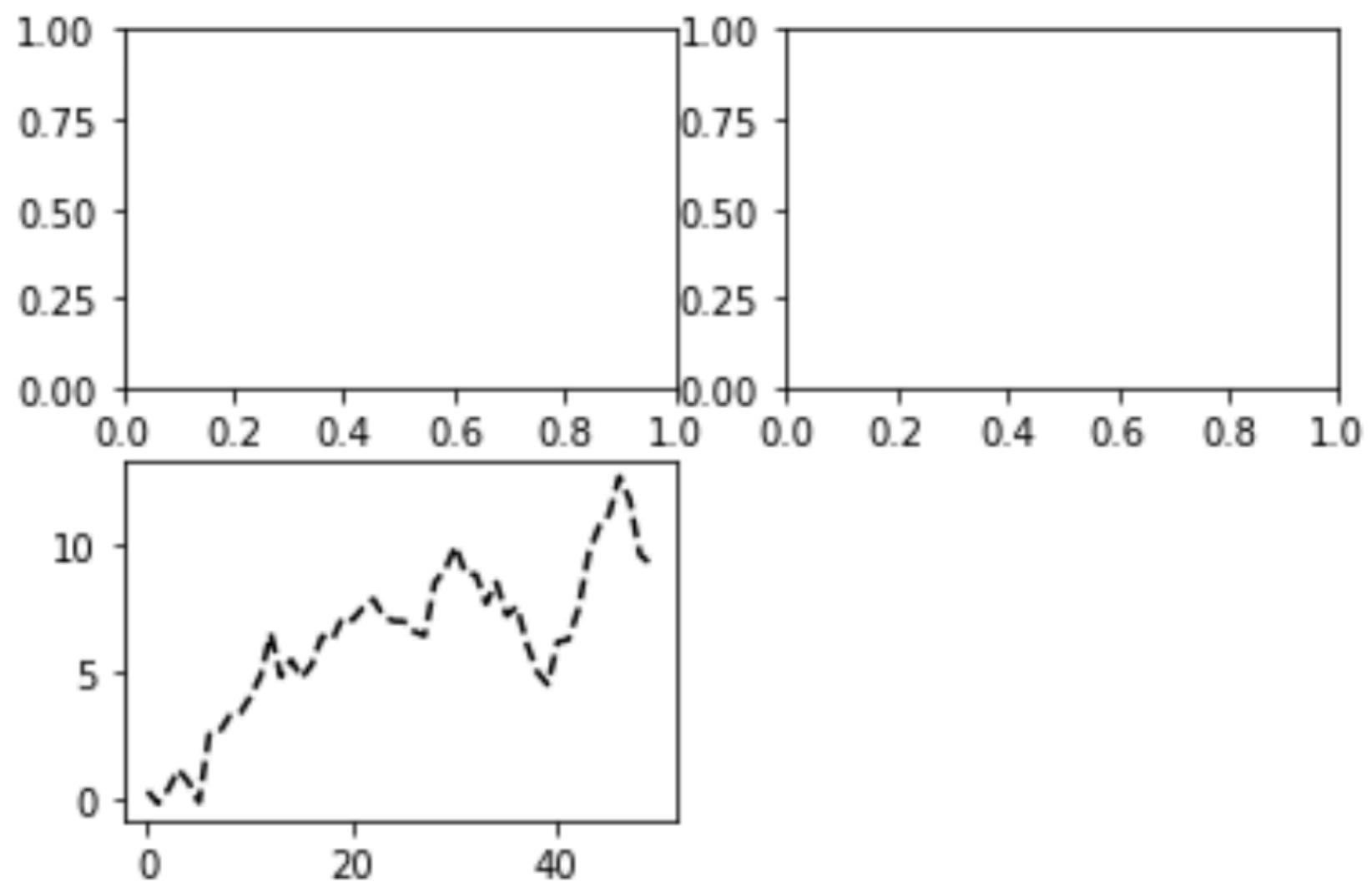
Live Coding

```
1 %matplotlib inline
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 data = np.arange(10)
6 plt.plot(data)
7 plt.show()
```



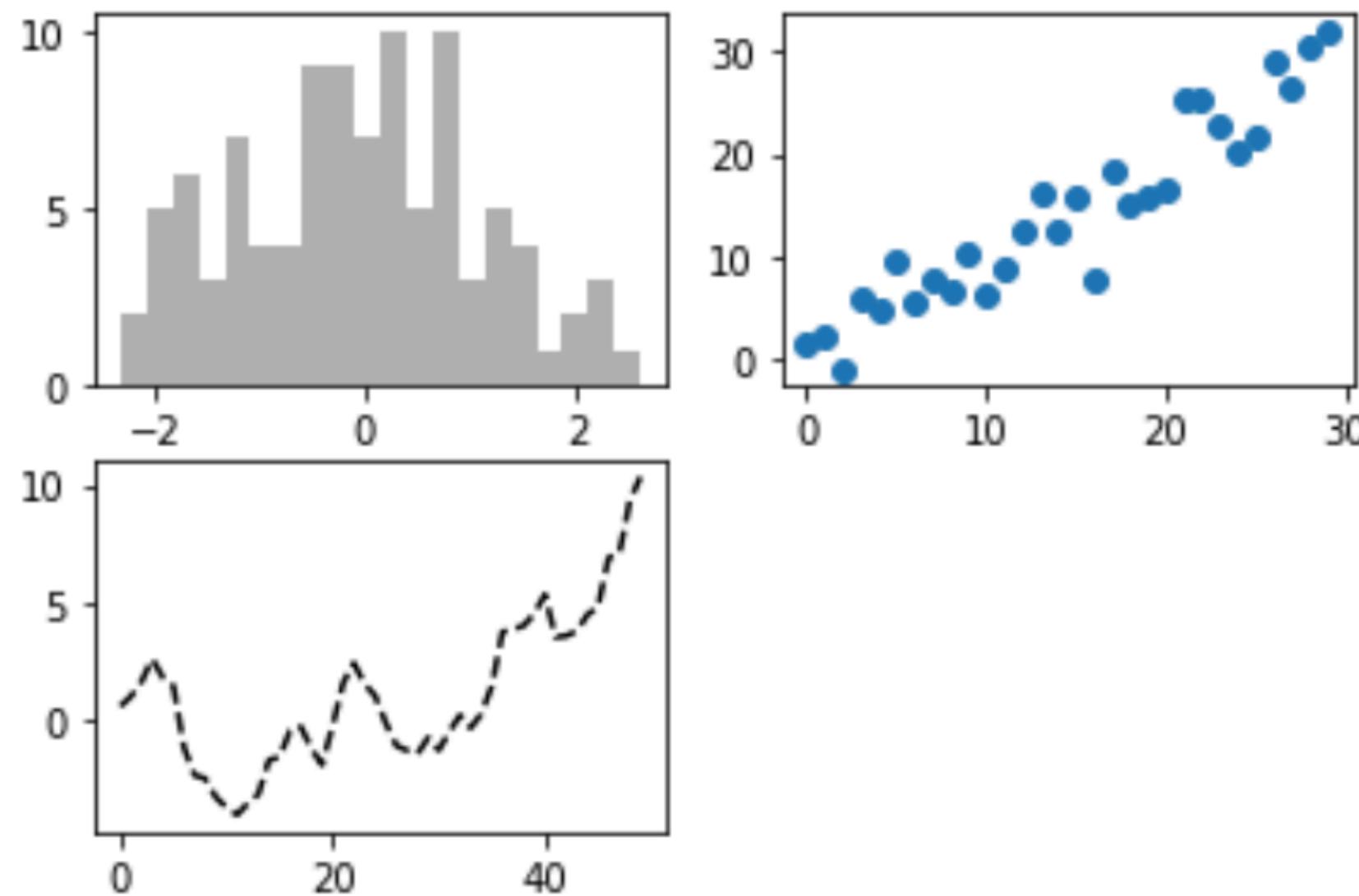
Live Coding

```
1 fig = plt.figure()
2 ax1 = fig.add_subplot(2, 2, 1)
3 ax2 = fig.add_subplot(2, 2, 2)
4 ax3 = fig.add_subplot(2, 2, 3)
5 plt.plot(np.random.standard_normal(50).cumsum(), 'k--')
6 plt.show()
```



Live Coding

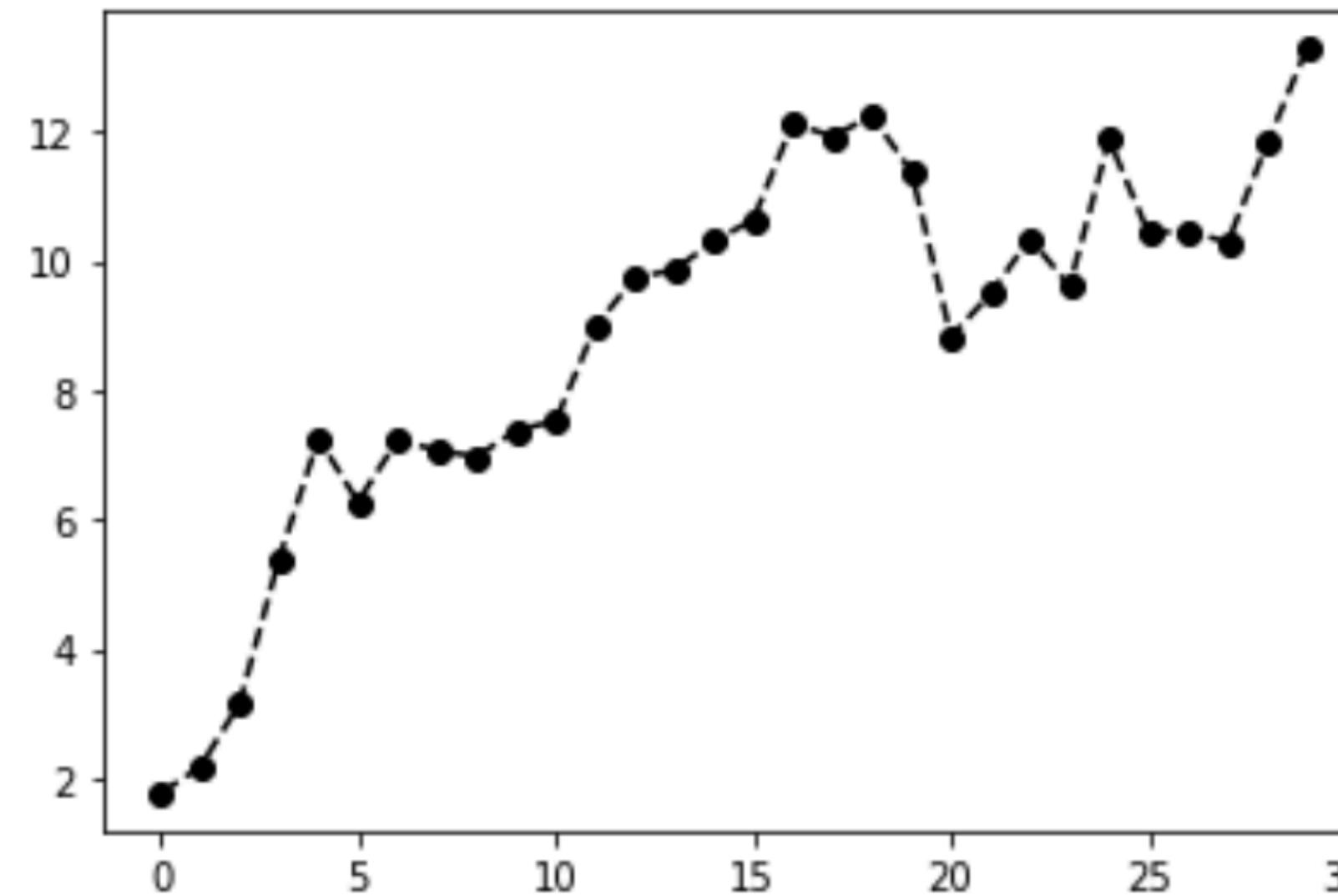
```
1 fig = plt.figure()
2 ax1 = fig.add_subplot(2, 2, 1)
3 ax2 = fig.add_subplot(2, 2, 2)
4 ax3 = fig.add_subplot(2, 2, 3)
5 plt.plot(np.random.standard_normal(50).cumsum(), 'k--')
6 ax1.hist(np.random.standard_normal(100), bins=20, color='k', alpha=0.3)
7 ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.standard_normal(30))
8 plt.show()
```



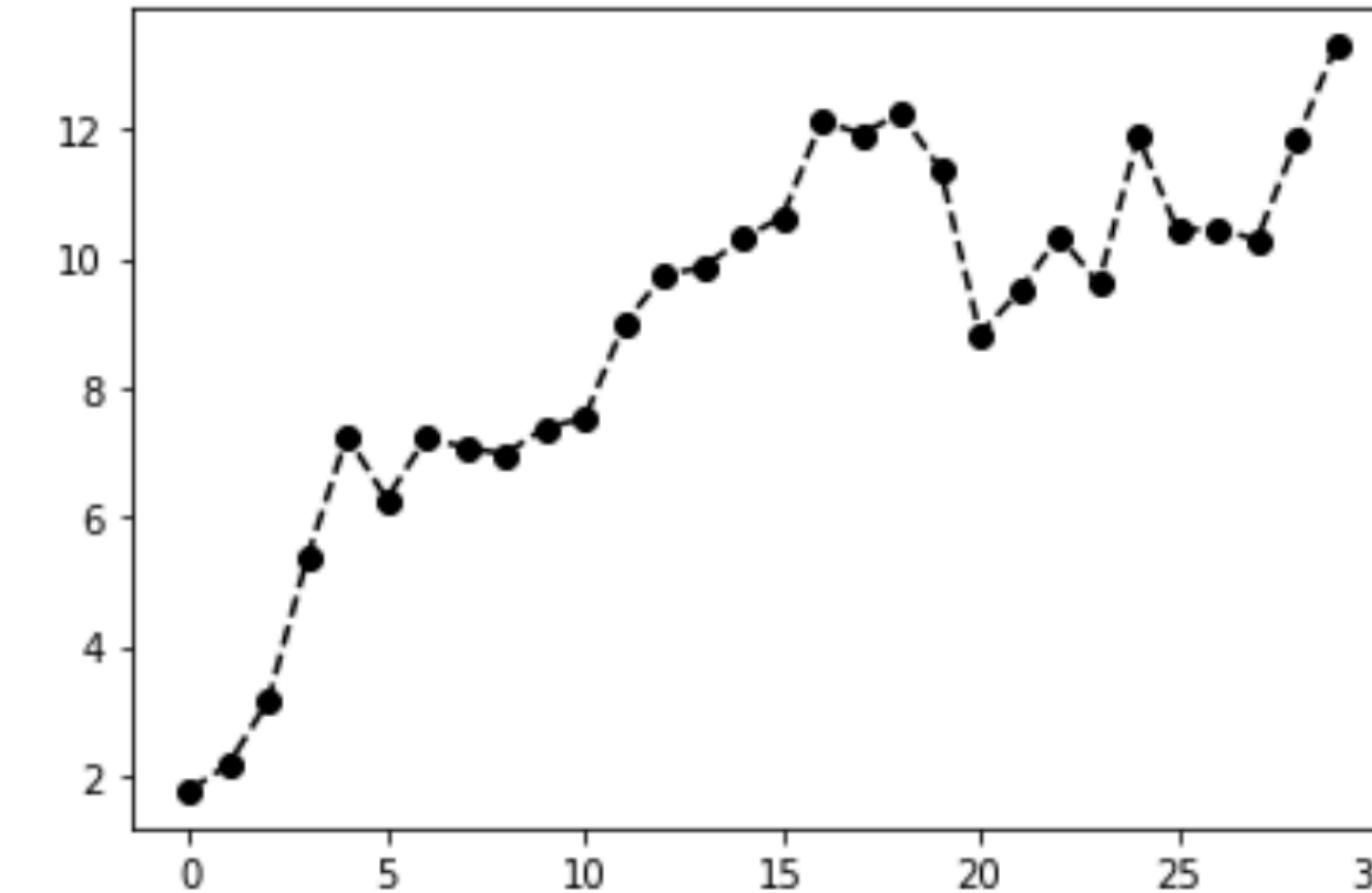
Live Coding

- Style, color

```
1 import numpy.random as nr
2 nr.seed(0)
3
4 plt.plot(nr.randn(30).cumsum(), 'ko--')
5 plt.show()
```

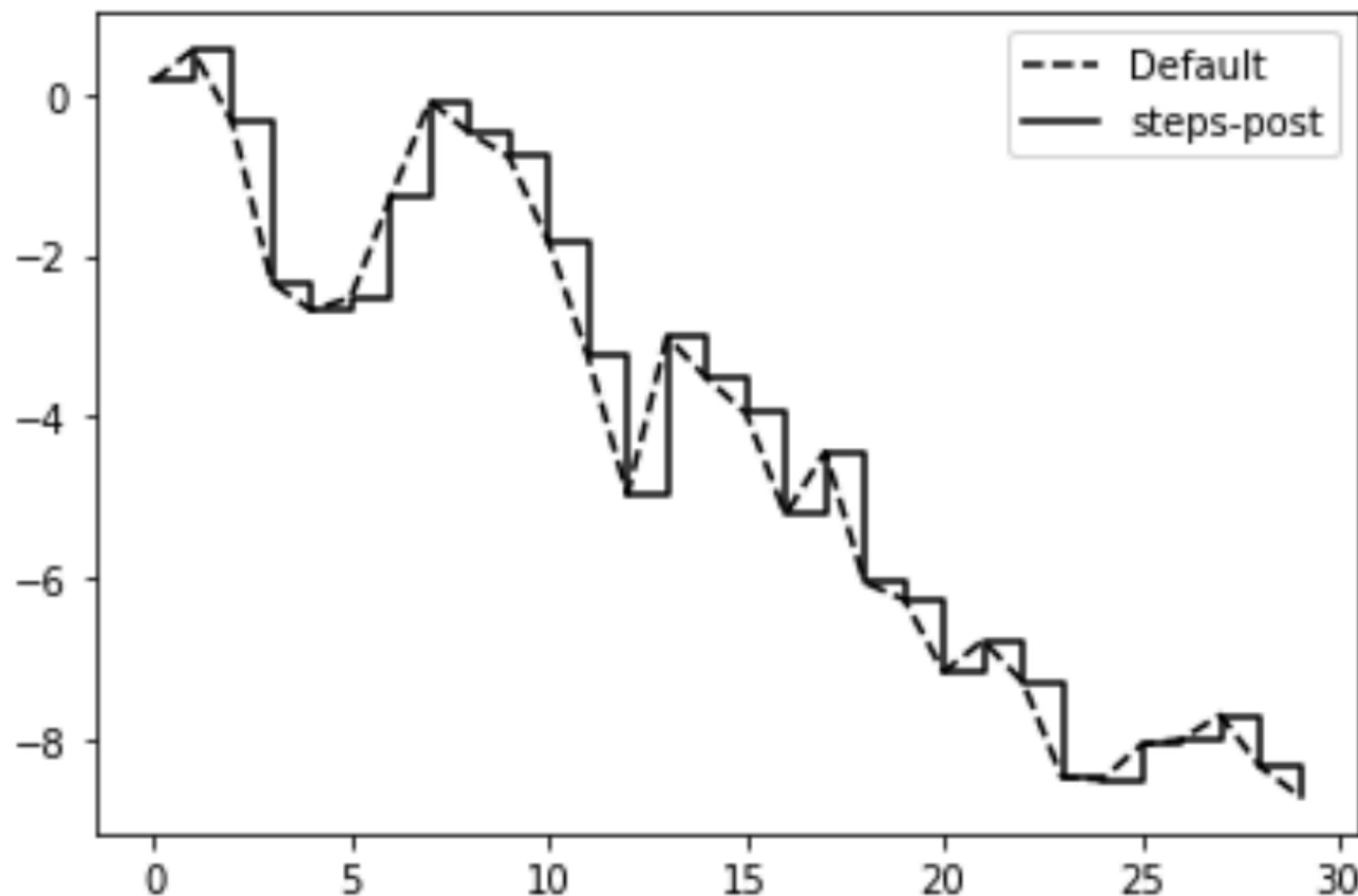


```
1 nr.seed(0)
2
3 plt.plot(nr.randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
4 plt.show()
```



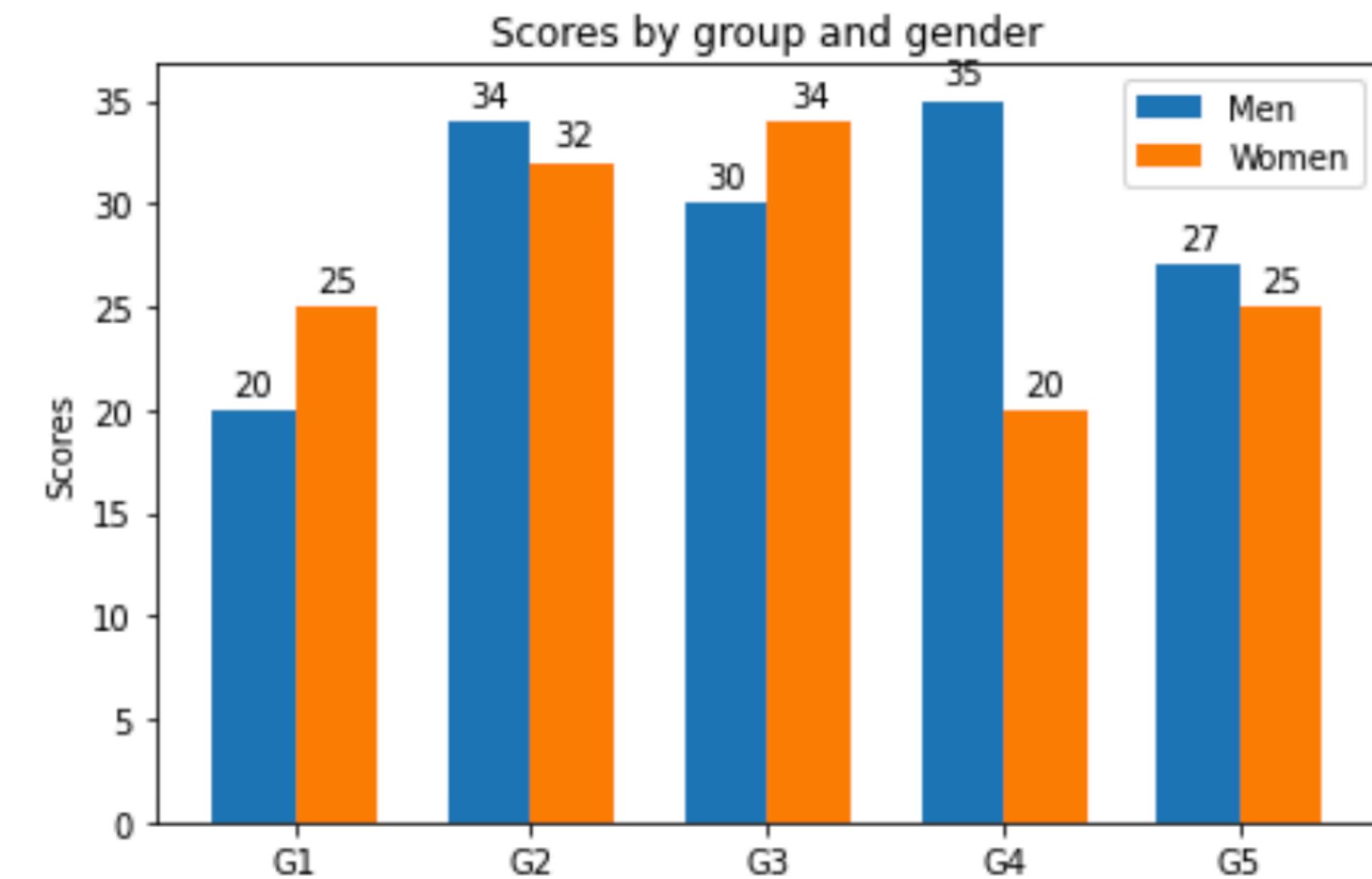
Live Coding

```
1 data = np.random.standard_normal(30).cumsum()  
2 plt.plot(data, 'k--', label='Default')  
3 plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')  
4 plt.legend(loc='best')  
5 plt.show()
```



Live Coding

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 labels = ['G1', 'G2', 'G3', 'G4', 'G5']
5 men_means = [20, 34, 30, 35, 27]
6 women_means = [25, 32, 34, 20, 25]
7
8 x = np.arange(len(labels)) # the label locations
9 width = 0.35 # the width of the bars
10
11 fig, ax = plt.subplots()
12 rects1 = ax.bar(x - width/2, men_means, width, label='Men')
13 rects2 = ax.bar(x + width/2, women_means, width, label='Women')
14
15 # Add some text for labels, title and custom x-axis tick labels, etc.
16 ax.set_ylabel('Scores')
17 ax.set_title('Scores by group and gender')
18 ax.set_xticks(x, labels)
19 ax.legend()
20
21 ax.bar_label(rects1, padding=3)
22 ax.bar_label(rects2, padding=3)
23
24 fig.tight_layout()
25
26 plt.show()
```



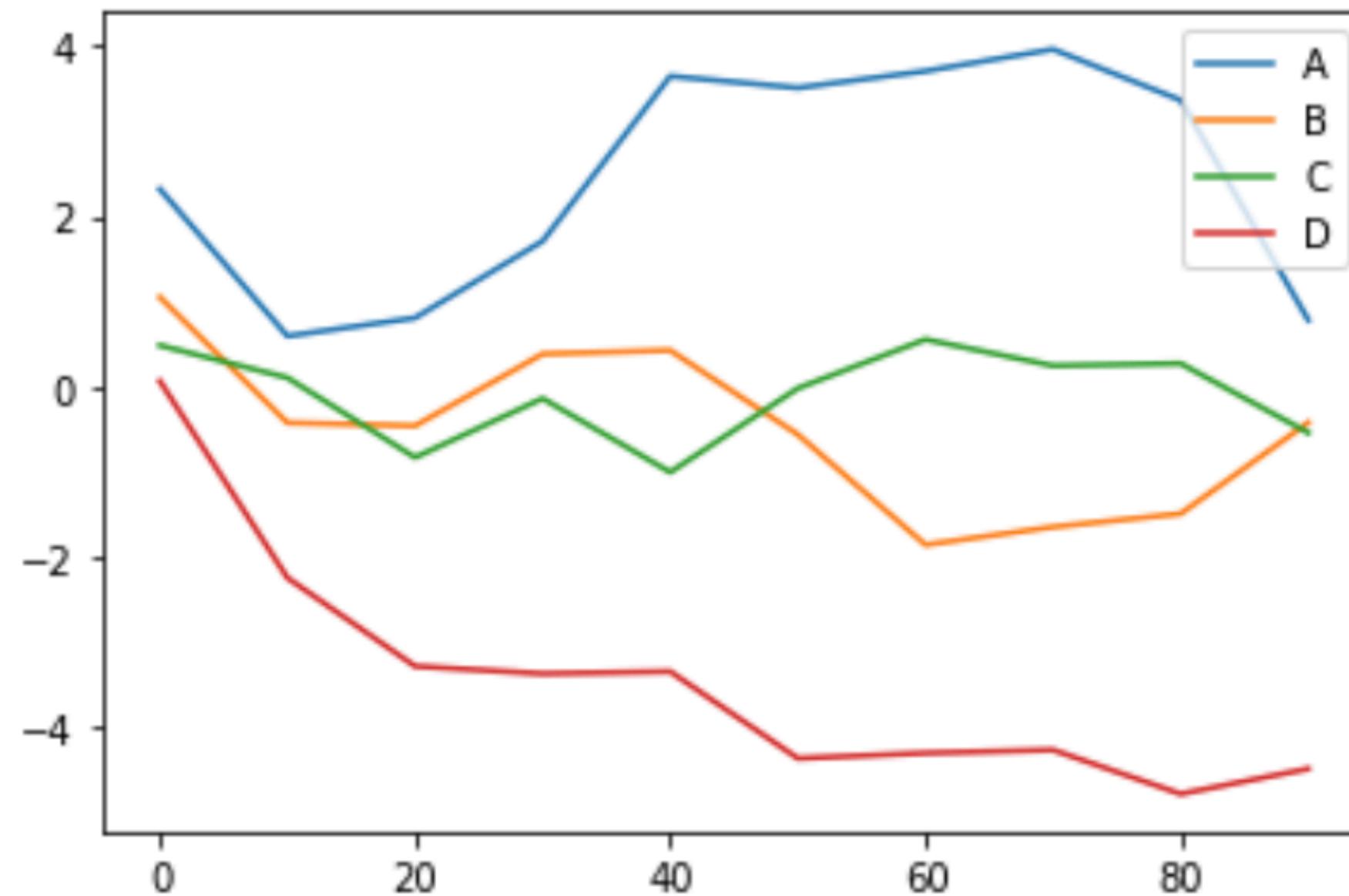
Live Coding

- 線性圖
- 散佈圖
- 直方圖
- 長條圖
- 原餅圖
- 方盒圖

Live Coding

```
1 df = pd.DataFrame(np.random.standard_normal((10, 4)).cumsum(0),  
2                   columns=['A', 'B', 'C', 'D'],  
3                   index=np.arange(0, 100, 10))
```

```
1 df.plot()  
2 plt.show()
```

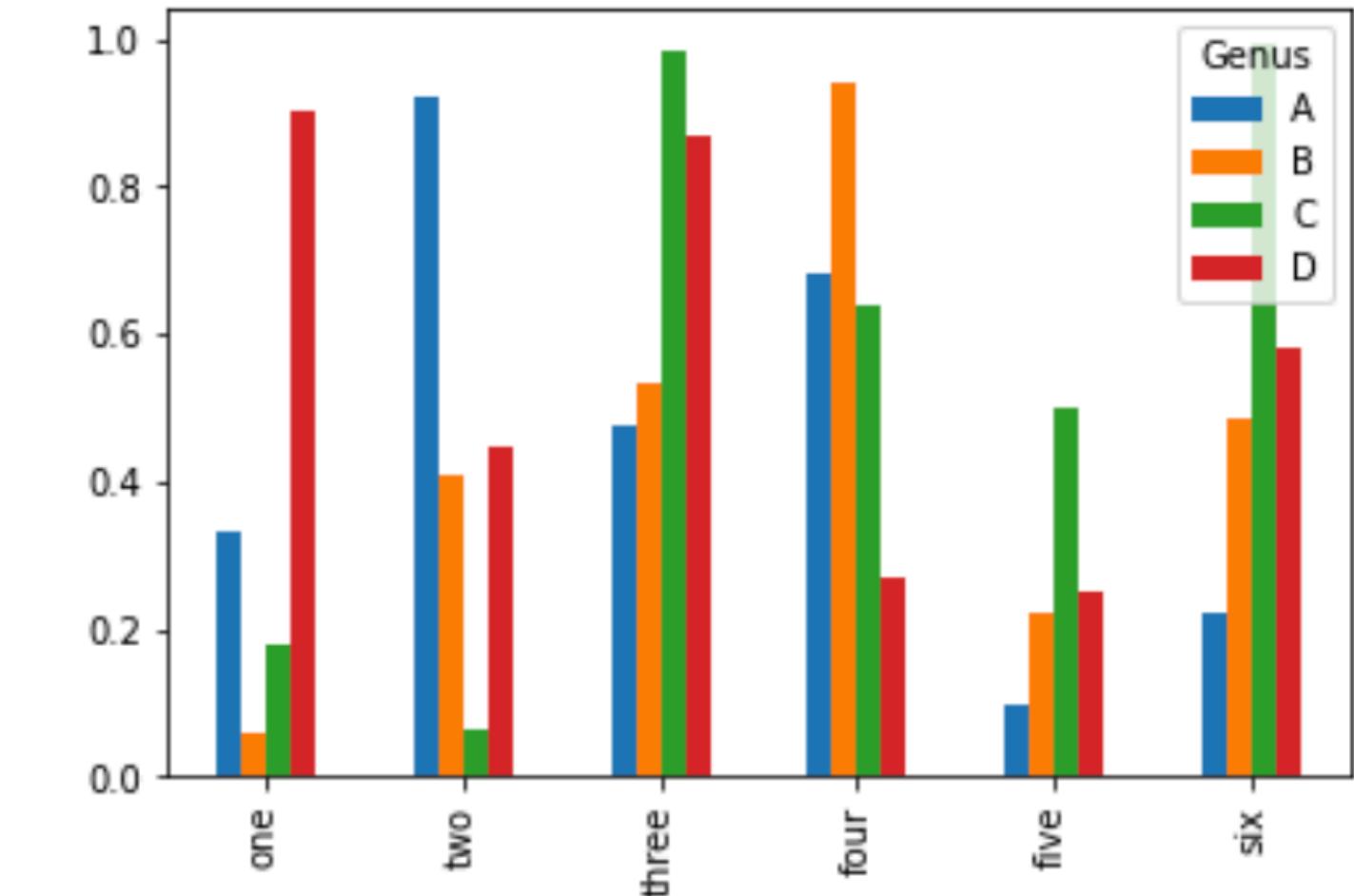


Live Coding

```
1 df = pd.DataFrame(np.random.uniform(size=(6, 4)),
2                     index=['one', 'two', 'three', 'four', 'five', 'six'],
3                     columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
4 df
```

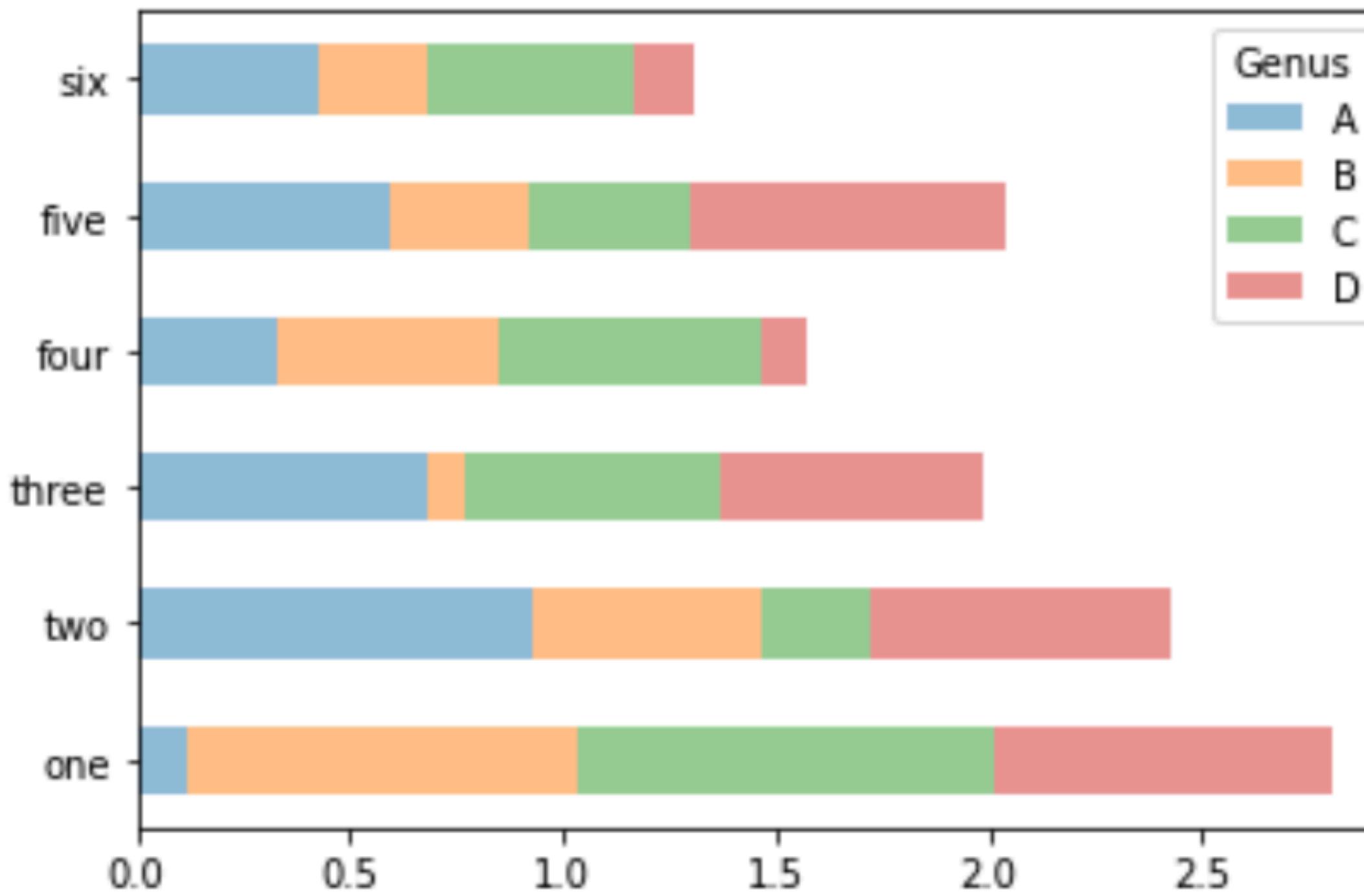
Genus	A	B	C	D
one	0.122428	0.916558	0.974458	0.795963
two	0.925785	0.542340	0.255921	0.702348
three	0.685924	0.082897	0.600044	0.619525
four	0.326964	0.521508	0.615677	0.111046
five	0.592162	0.327074	0.382839	0.733764
six	0.430131	0.253924	0.487058	0.139975

```
1 df.plot.bar()
2 plt.show()
```



Live Coding

```
1 df.plot.barh(stacked=True, alpha=0.5)  
2 plt.show()
```

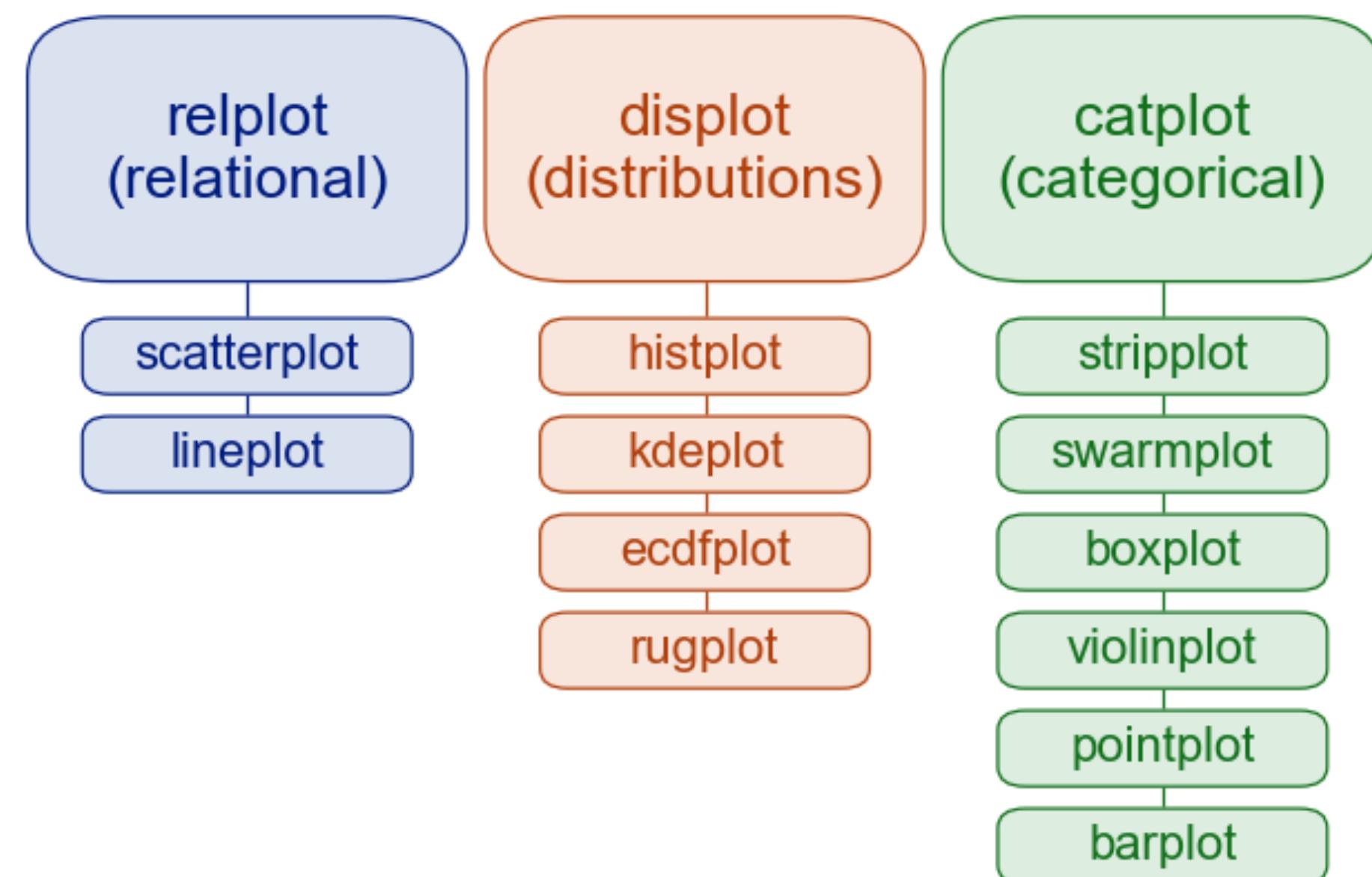


3-2

seaborn

Module seaborn

- 以matplotlib為基底所建構的Python 資料視覺化模組
 - 提供更方便使用的介面與較好的統計圖形呈現



Live Coding

- 利用小費資料進行幾種圖形的呈現與變化

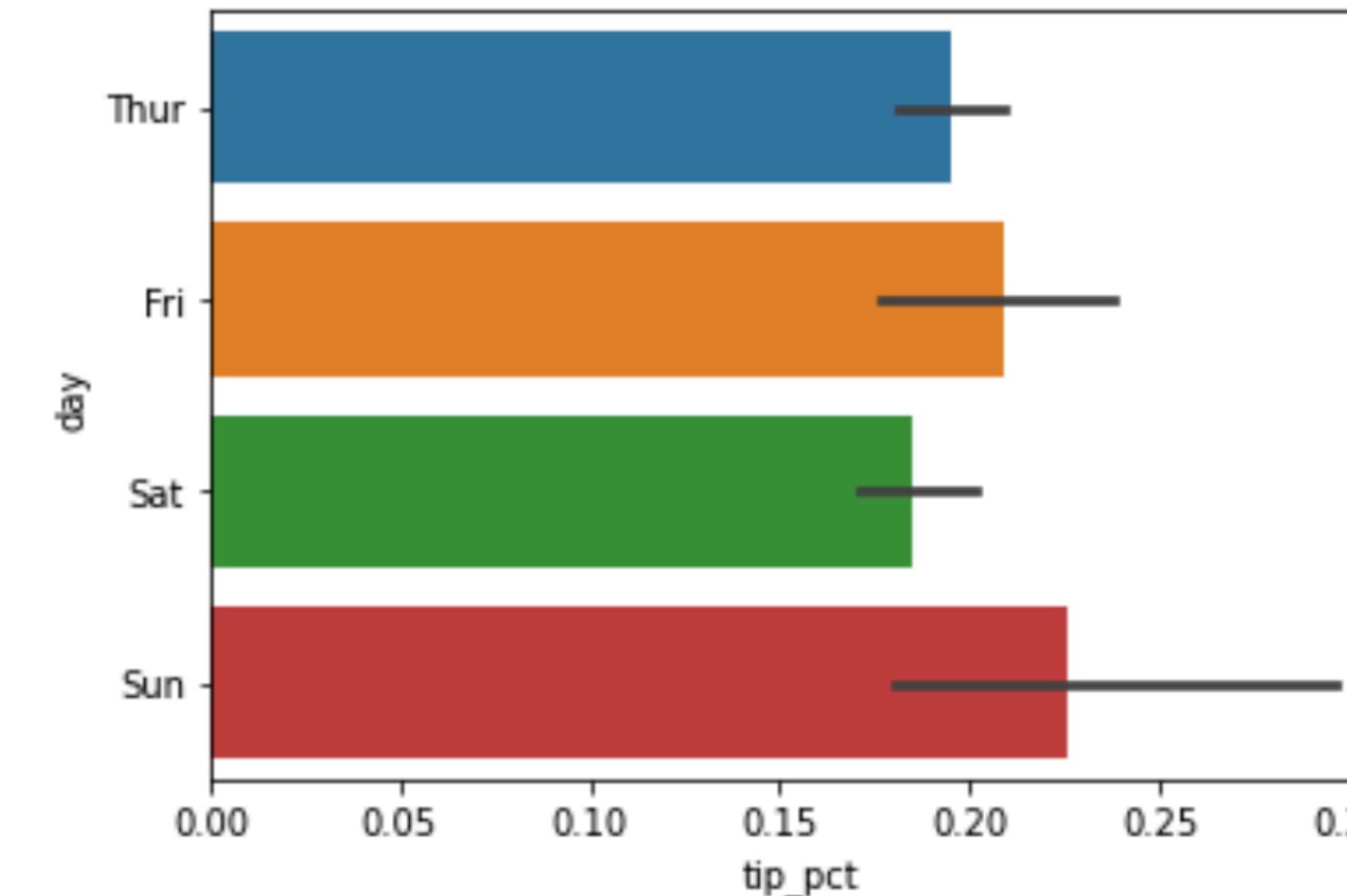
```
1 import seaborn as sns  
2  
3 tips = sns.load_dataset("tips")  
4 tips
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

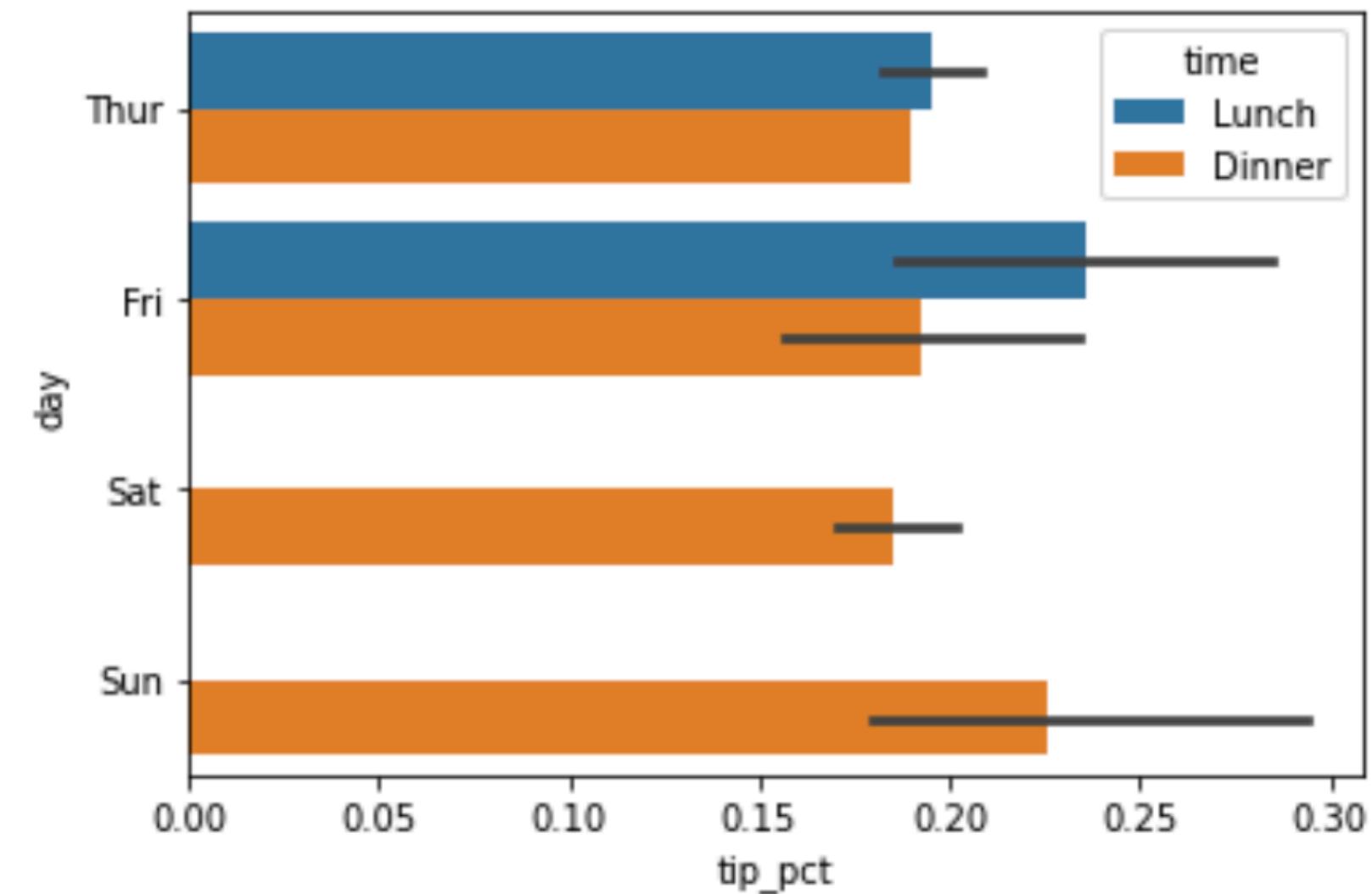
```
1 tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])  
1 sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

<AxesSubplot:xlabel='tip_pct', ylabel='day'>



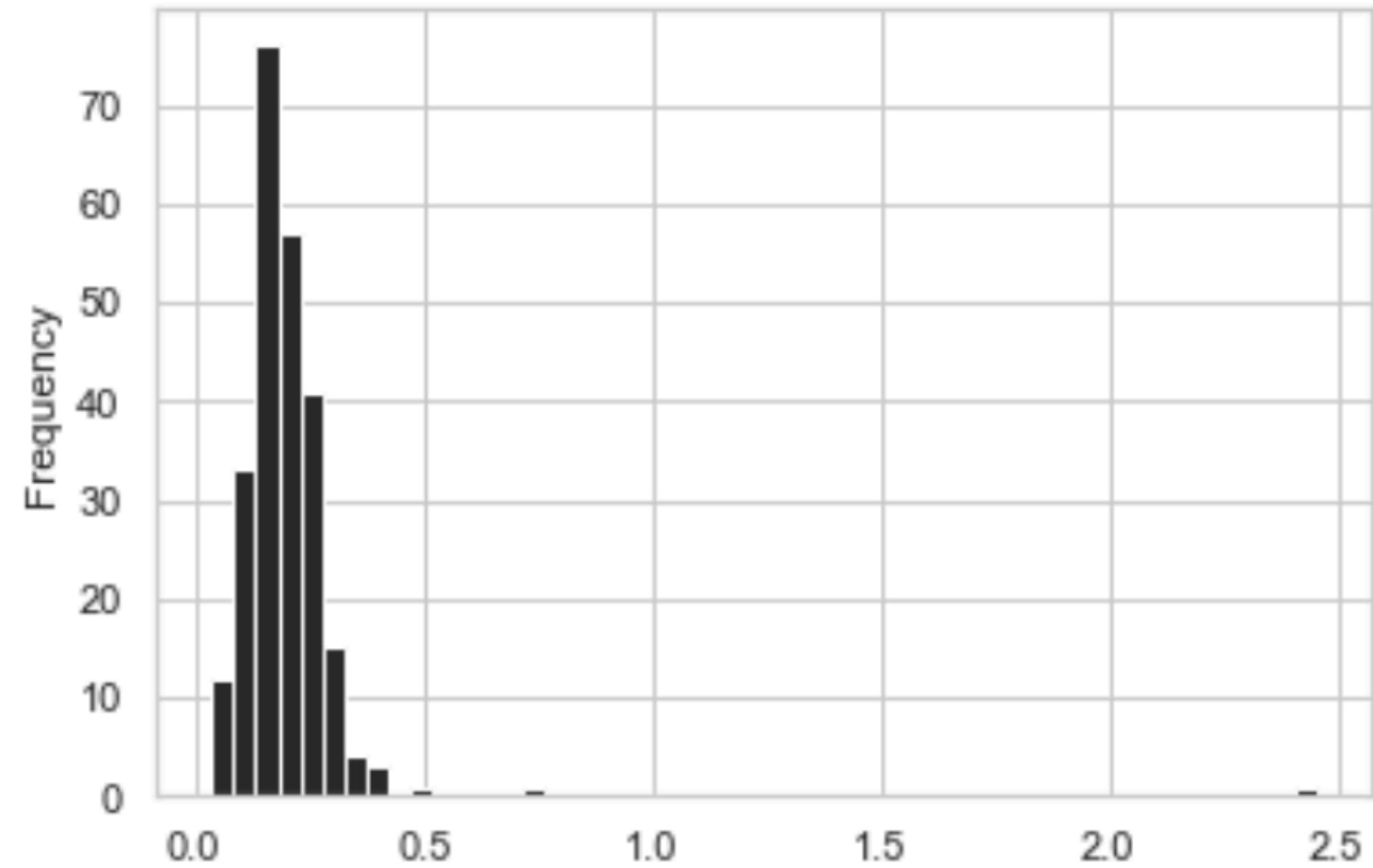
Live Coding

```
1 sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')  
2 plt.show()
```

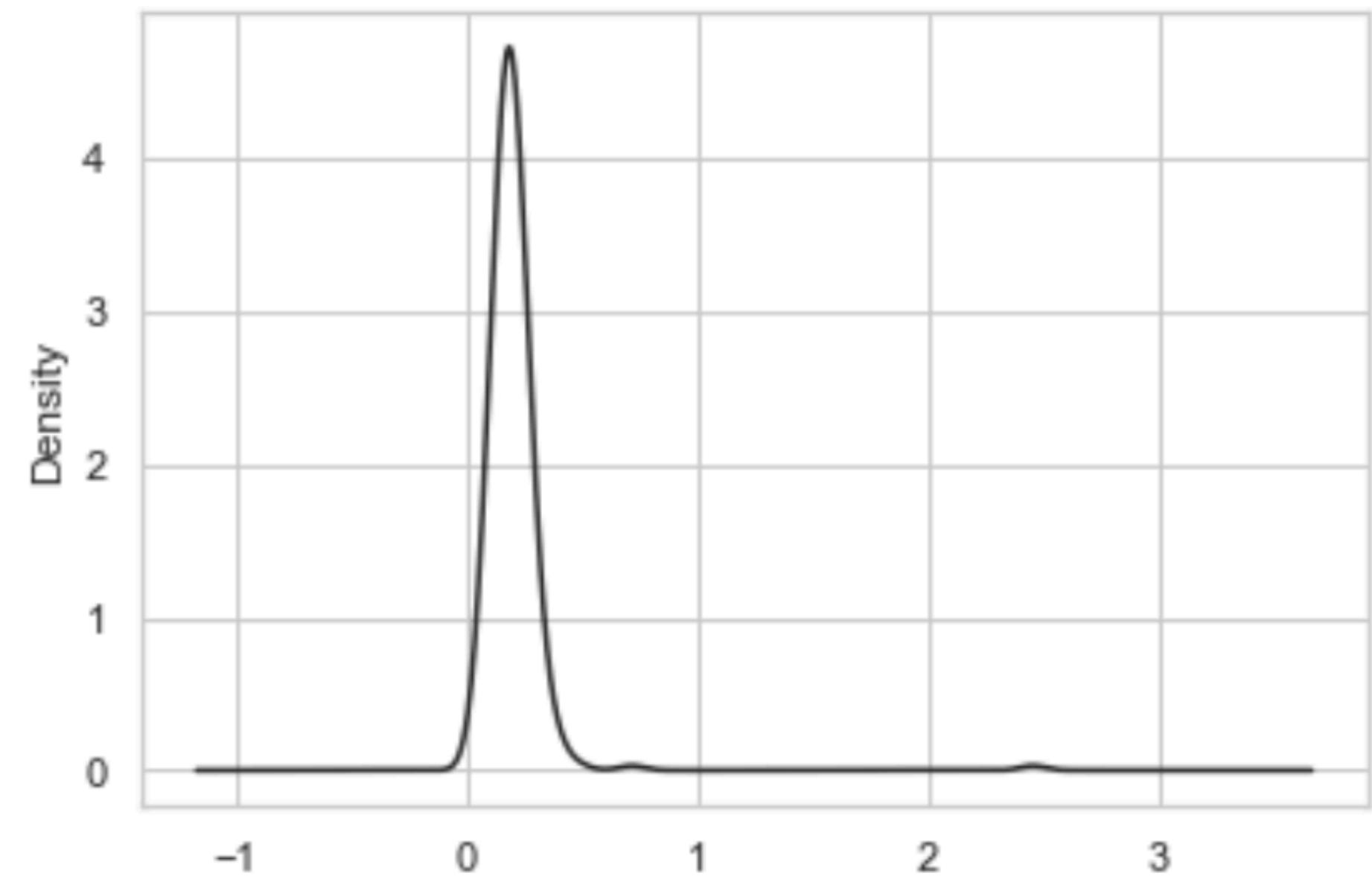


Live Coding

```
1 sns.set(style="whitegrid")
2 sns.set_palette('Greys_r')
3
4 tips['tip_pct'].plot.hist(bins=50)
5 plt.show()
```

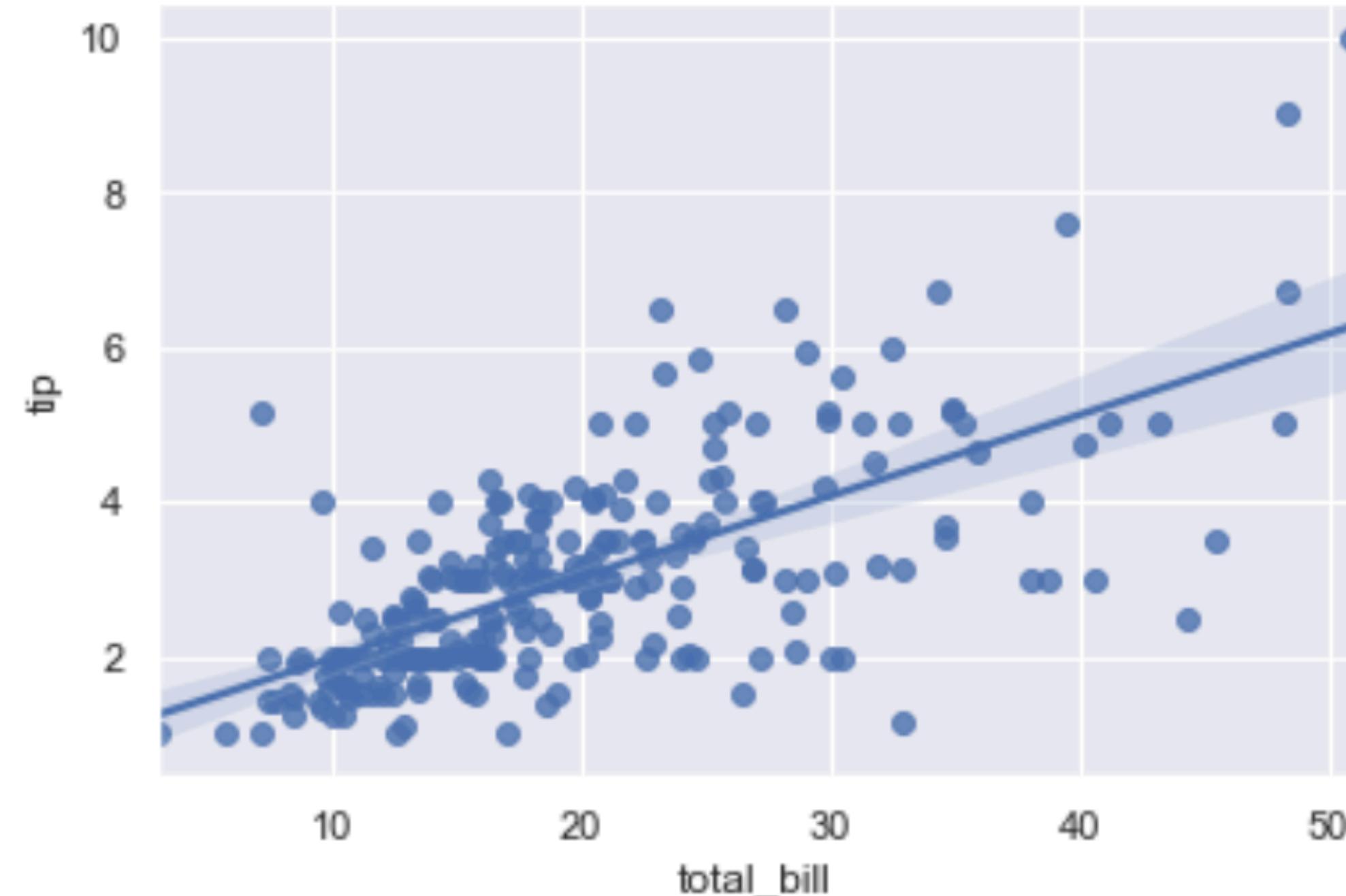


```
1 tips['tip_pct'].plot.density()
2 plt.show()
```

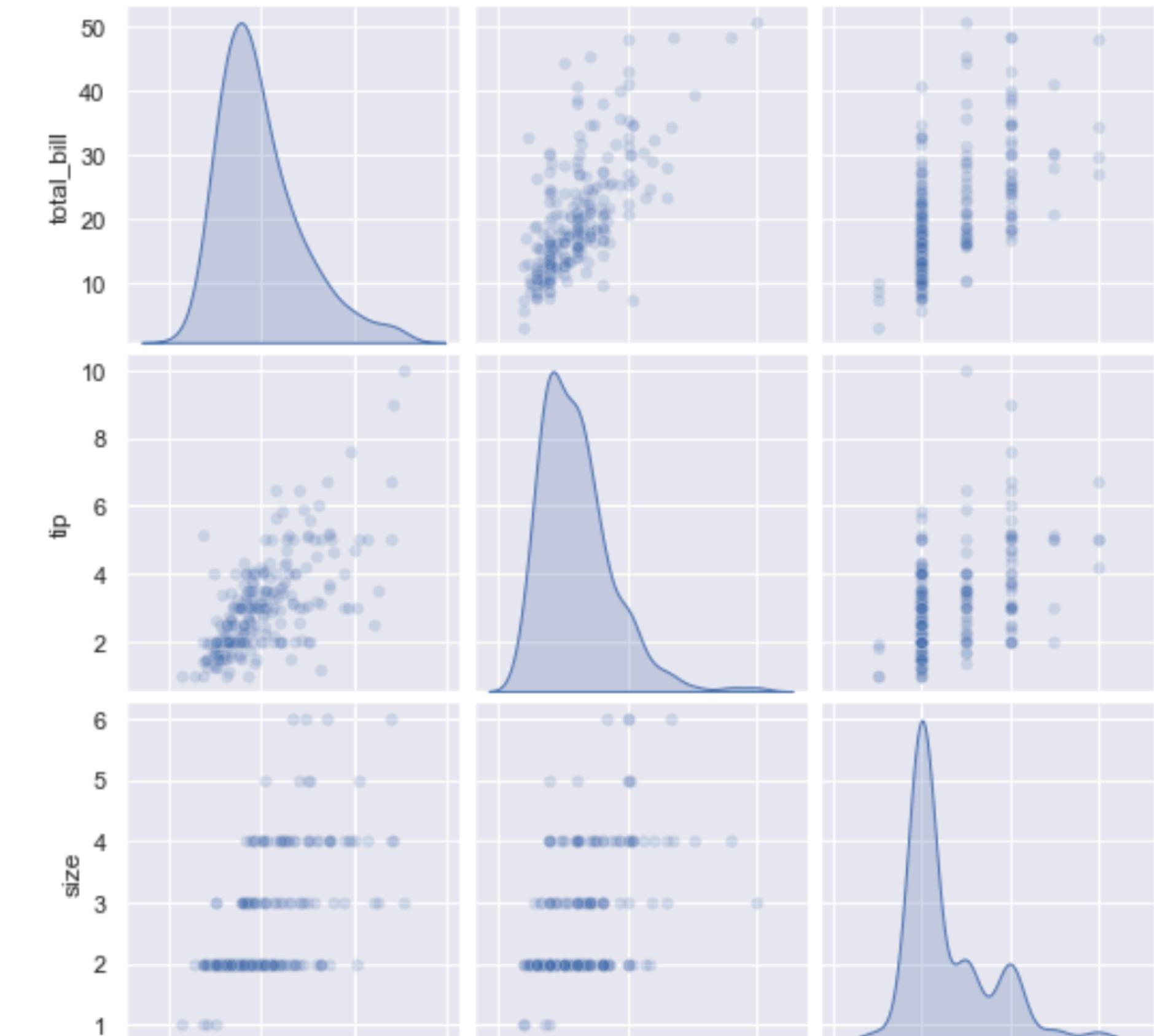


Live Coding

```
1 sns.set_theme(style="darkgrid")
2
3 tips = sns.load_dataset("tips")
4 sns.regplot(x="total_bill", y="tip", data=tips)
5 plt.show()
```

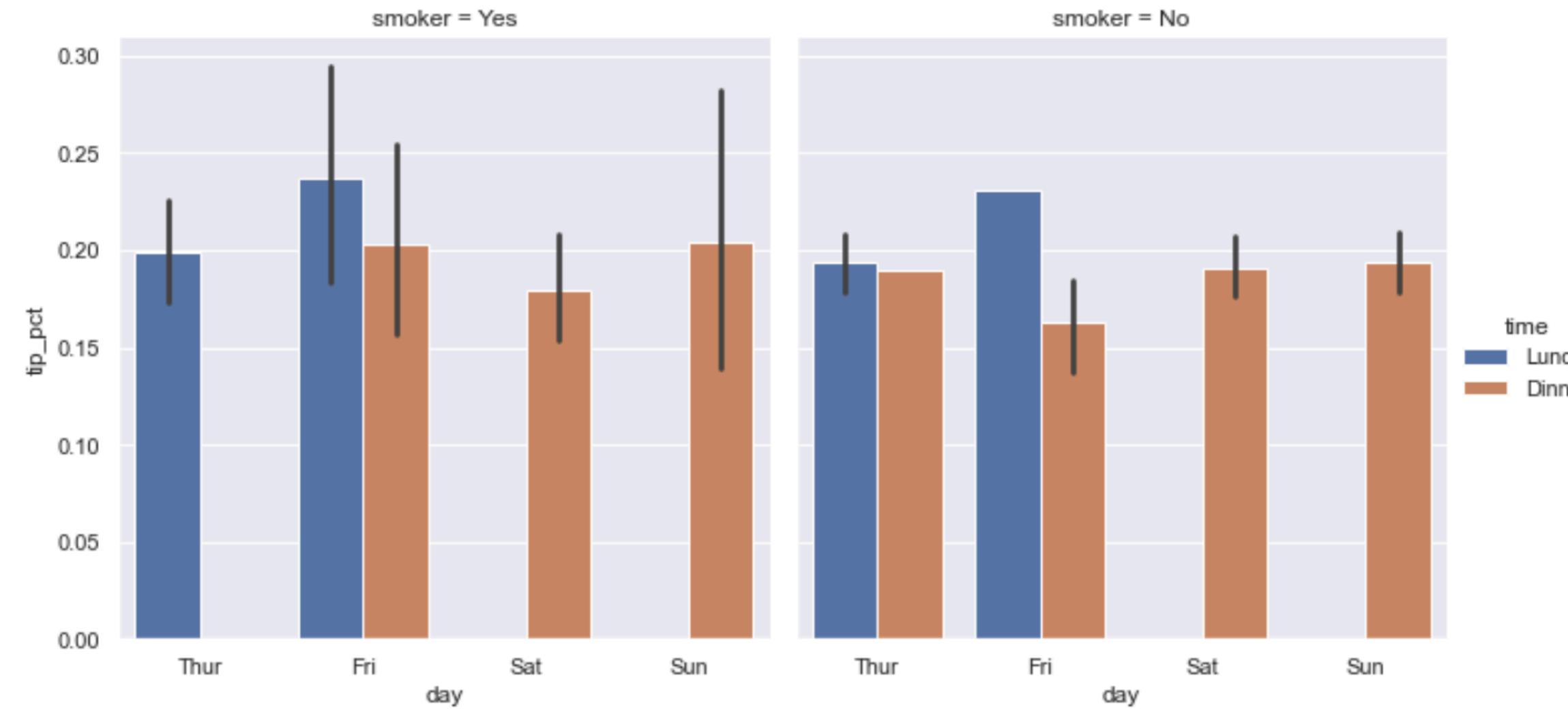


```
1 sns.pairplot(tips, diag_kind="kde", plot_kws={"alpha": 0.2})
2 plt.show()
```

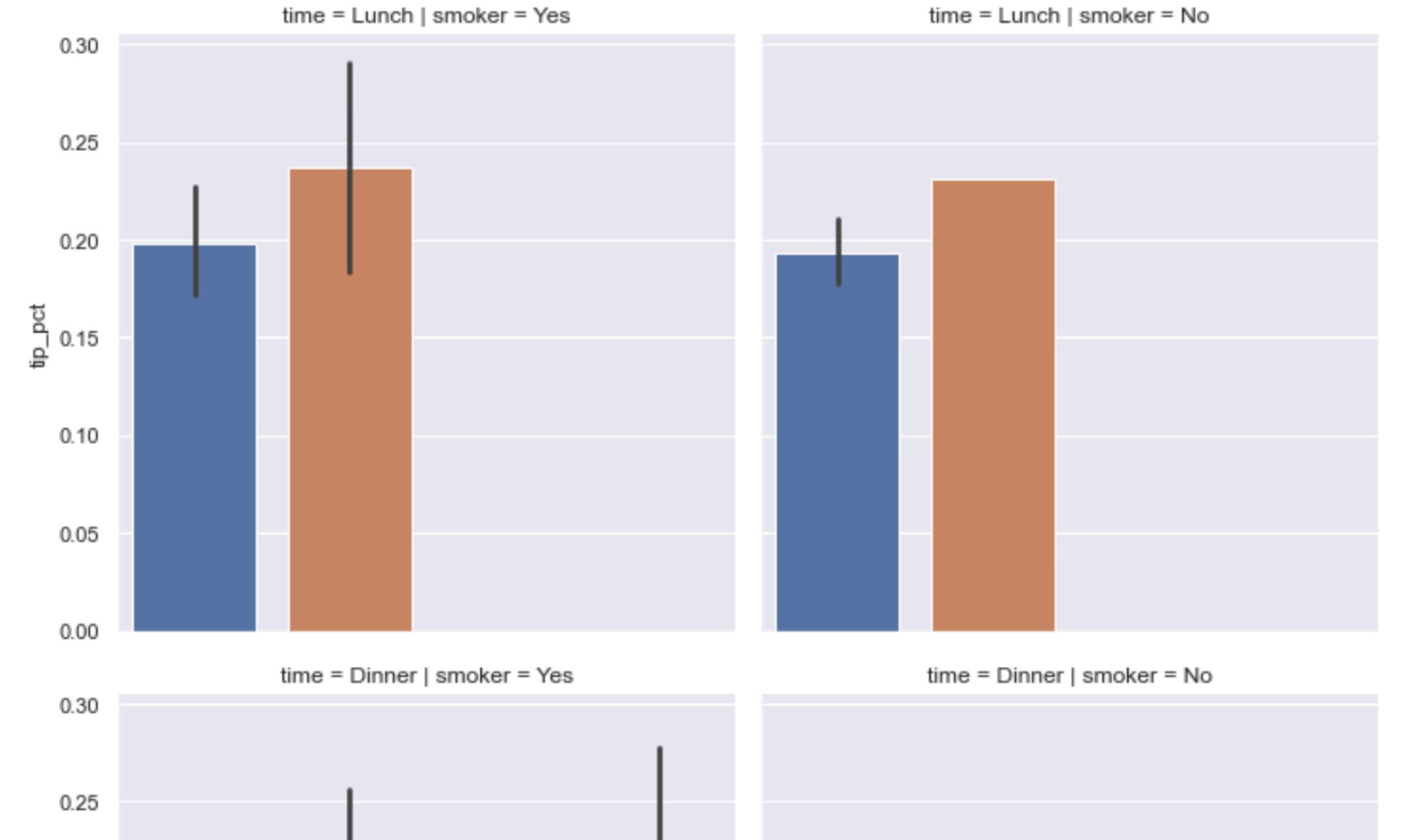


Live Coding

```
1 sns.catplot(x='day', y='tip_pct', hue='time', col='smoker',
2               kind='bar', data=tips[tips.tip_pct < 1])
3 plt.show()
```

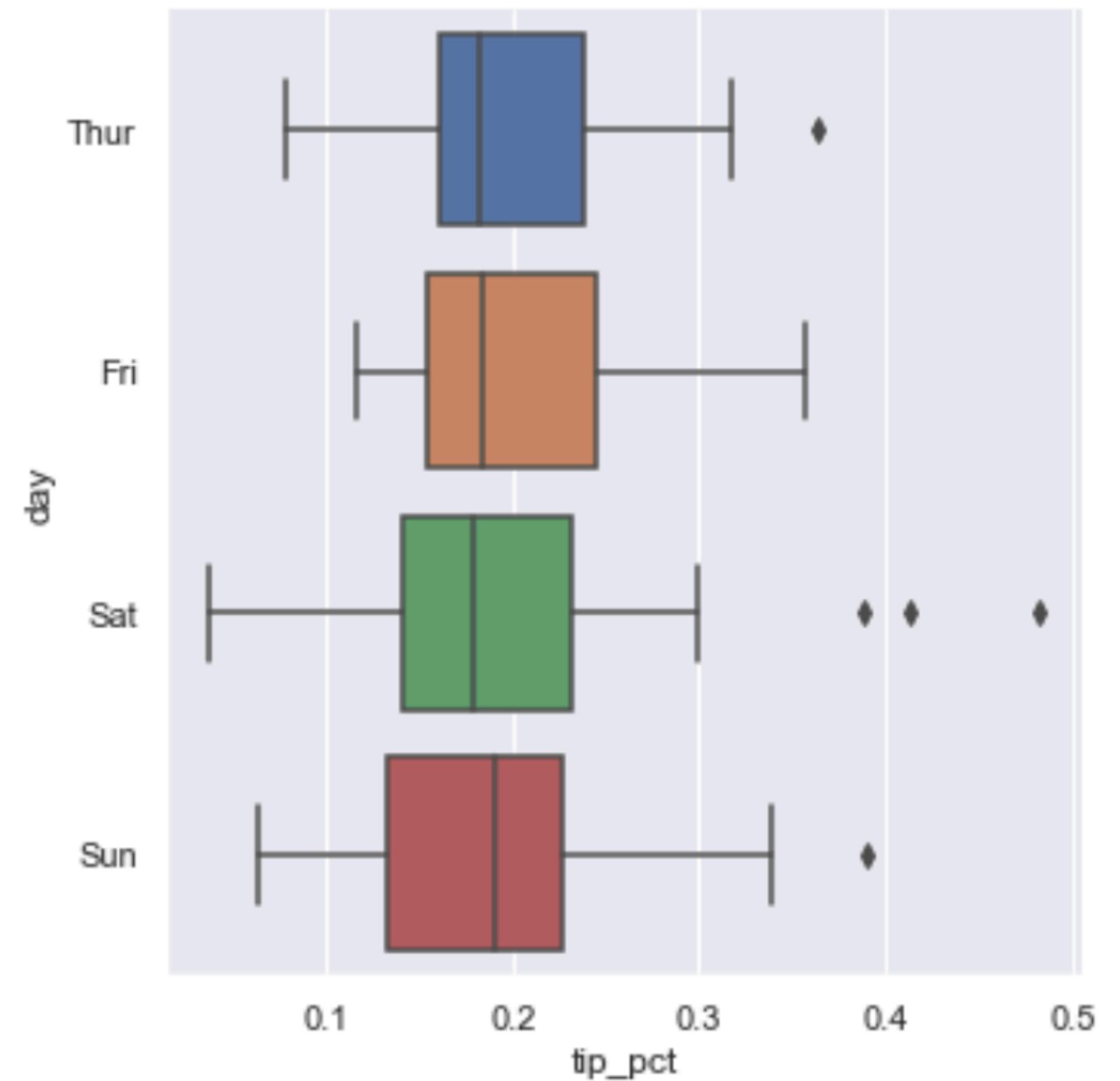


```
1 sns.catplot(x='day', y='tip_pct', row='time',
2               col='smoker',
3               kind='bar', data=tips[tips.tip_pct < 1])
4 plt.show()
```



Live Coding

```
1 sns.catplot(x='tip_pct', y='day', kind='box',
2               data=tips[tips.tip_pct < 0.5])
3 plt.show()
```



3-3

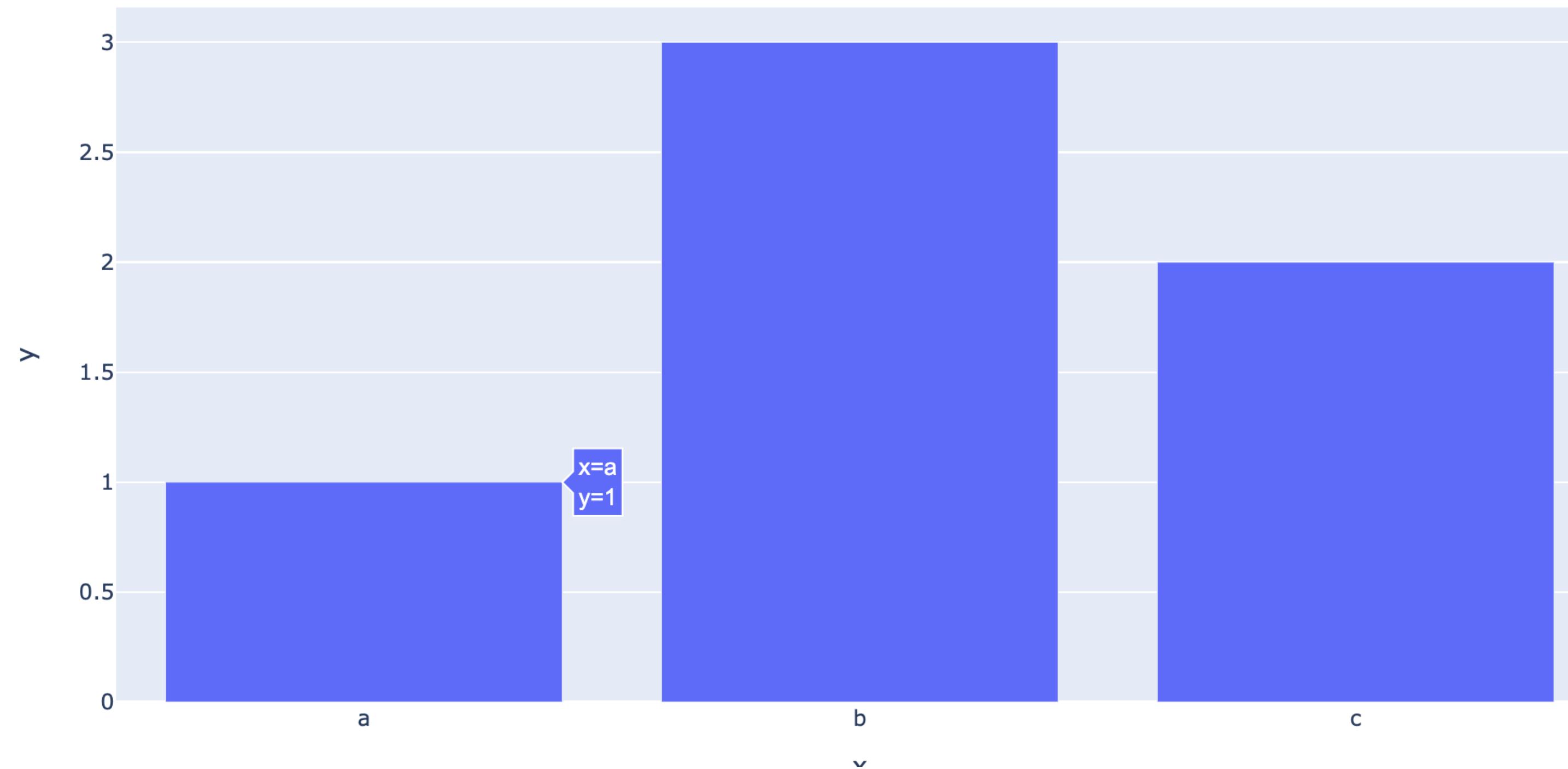
Plotly

Module `plotly`

- 可以繪製呈現在網頁上的互動式圖形模組
 - 會將最後結果儲存成html

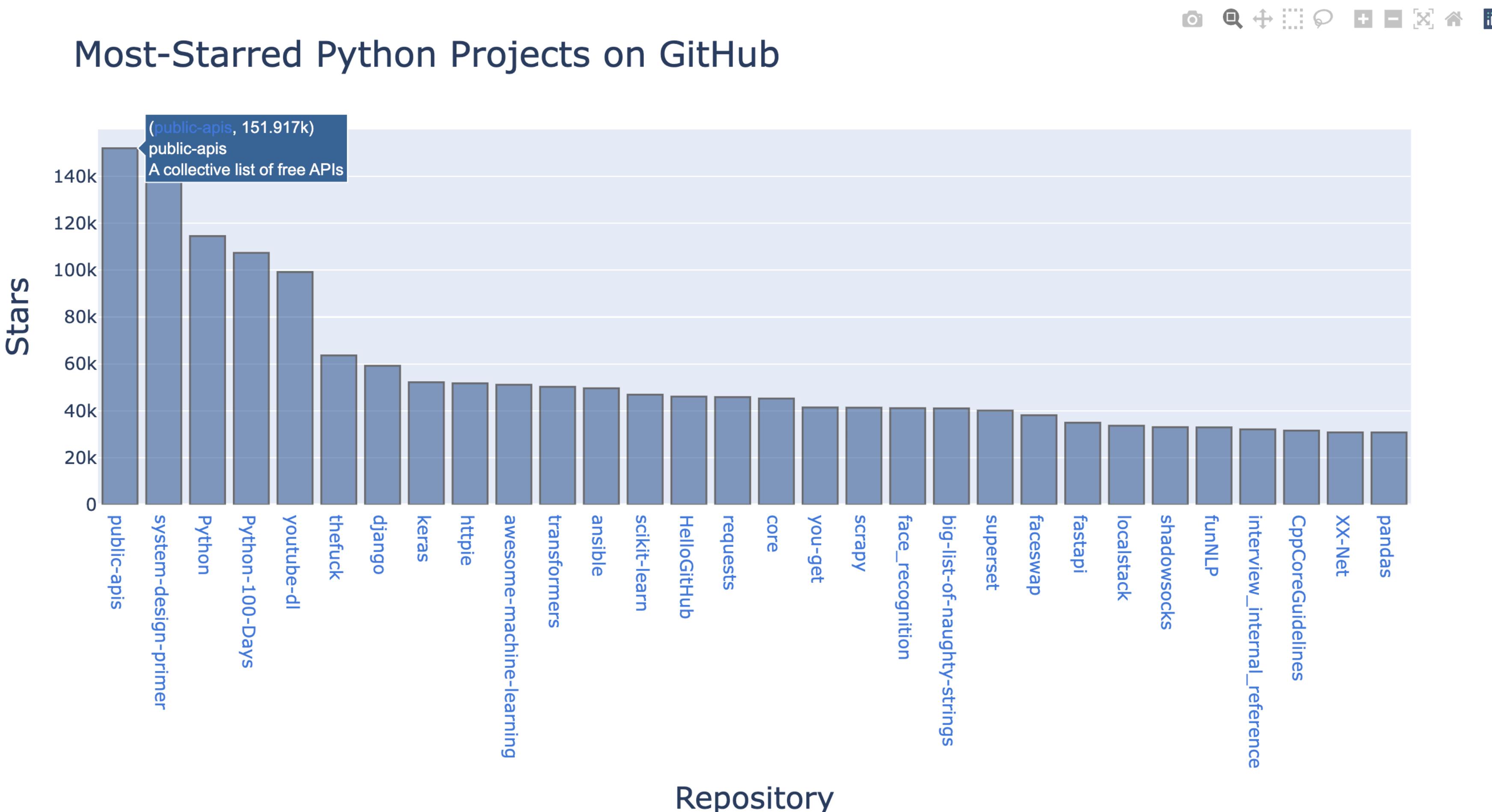
Live Coding

```
1 import plotly.express as px  
2 fig = px.bar(x=["a", "b", "c"], y=[1, 3, 2])  
3 fig.show()
```



Live Coding

- 下一頁為範例程式碼



Live Coding

```
1 import requests
2
3 from plotly.graph_objs import Bar
4 from plotly import offline
5
6 language = "python"
7 url = "https://api.github.com/search/repositories?q=language:{}&sort=stars".format(language)
8 # headers = {'Accept': "application/json; charset=utf-8"}
9 r = requests.get(url)
10 print(r.text)
11 response_dict = r.json()
12 repo_dicts = response_dict['items']
13 repo_links, stars, labels = [], [], []
14 for repo_dict in repo_dicts:
15     repo_name = repo_dict['name']
16     repo_url = repo_dict['html_url']
17     stars.append(repo_dict['stargazers_count'])
18     repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
19     repo_links.append(repo_link)
20
21     owner = repo_dict['owner']['login']
22     description = repo_dict['description']
23     label = f"{owner}<br />{description}"
24     labels.append(label)
25
26 data = [
27     {
28         'type': 'bar',
29         'x': repo_links,
30         'y': stars,
31         'hovertext': labels,
32         'marker': {
33             'color': 'rgb(60, 100, 150)',
34             'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}
35         },
36         'opacity': 0.6,
37     }
38 ]
39 my_layout = {
40     'title': 'Most-Starred Python Projects on GitHub',
41     'titlefont': {'size': 28},
42     'xaxis': {
43         'title': 'Repository',
44         'titlefont': {'size': 24},
45         'tickfont': {'size': 14},
46     },
47     'yaxis': {
48         'title': 'Stars',
49         'titlefont': {'size': 24},
50         'tickfont': {'size': 14},
51     }
52 }
53 fig = {'data': data, 'layout': my_layout}
54 offline.plot(fig, filename='python_repos.html')
```

4

探索式資料分析實務操作

Elements of Structured Data

- 瞭解資料型態 (Data Types)

1. Numeric

- ◆ Continuous

- ◆ Discrete

2. Categorical

- ◆ Binary

- ◆ Ordinal

Estimates of Location

- 瞭解資料樣貌
- 平均、中位數、四分位數等

Estimates of Variability

- 瞭解資料的變化
- 標準差、變異數等

Exploring the Data Distribution

- 瞭解資料的分布
- 利用散佈圖等圖形瞭解資料分布狀況

Exploring Binary and Categorical Data

- 瞭解類別資料樣貌
- 利用眾數或直方圖原餅圖等圖形瞭解資料樣貌

Correlation

- 瞭解類別資料之間的相關性
- 利用散佈圖或是關係矩陣等瞭解資料彼此之間的關係

The Process of Data Preparation

1. Handling Missing Data: Filtering out/in missing data

2. Data Transformation:

- Removing Duplicates
- Transforming Data Using a Function or Mapping
- Replacing Values

3. Extension Data Types

4. String Manipulation

Kaggle

- 資料分析競賽平台
 - <https://www.kaggle.com/>
- 金融經典案例
 - 鐵達尼號生存預測
 - 房價預測、信用評比

信用風險分析

- 使用案例進行探索式資料分析
- <https://www.kaggle.com/competitions/home-credit-default-risk>

Source Code Reference

- Github
 - <https://github.com/lzrong0203/python0913>