PYTHON 介紹

Prof. Chia-Yu Lin

Yuan Ze University

2022 Spring

We Use Python 3

- https://www.python.org/downloads/

怎麼學

- 1. https://www.python.org/doc/
- 2. Google “python tutorial”

遇到問題時

- Google
- Stackoverflow
- Ask other people to google for you(?)

空白鍵 (Whitespace)

- 空白鍵在Python中是有意義的，尤其是縮白或是換行
- 用「換行」來結束一行程式碼 (Not a semicolon ”;” like in C++ or Java.)
- 在Pythong中沒有使用 { } 來代表一個block
- 改用一致的縮排 (Tab or Space)
- 冒號通常會出現在一個新的block的開始 (如function and class definitions.)

```
for train, test in kfold.split(dataset):
    X_train= dataset.iloc[train]
    Y_train = label.iloc[train]
```

註解 (Comments)

- 開始註解用 “#” ，剩下都忽略
- “ “ “ “ “ “ 是換行的註解
- 用註解來說明一個你定義的新函式
- The development environment, debugger, and other tools use it: it’s good style to include one.

```
def my_function(x, y):
    """This is the docstring. This
    function does blah blah blah."""
    # The code would go here...
```

Understanding Basic Concept

Output

```
print('Hi, my name is', 'Simon')
```

```
print('Now processing',file_name,'...')
```

變數 (Variable)

- 在Python中不需要先宣告變數
- Python 會根據初始值來定義變數的型態
  - int
  - Float
  - str

```
iv = 10
fv = 12.3
cv = 3 + 5j
sv = 'hello python'
bv = True
nv = None

print(iv, fv, cv, sv, bv)
print(type(iv))
print(type(fv))
print(type(cv))
print(type(sv))
print(type(bv))
print(nv)
print(isinstance(sv, str))
```

User Input

- “input” will return the external message.

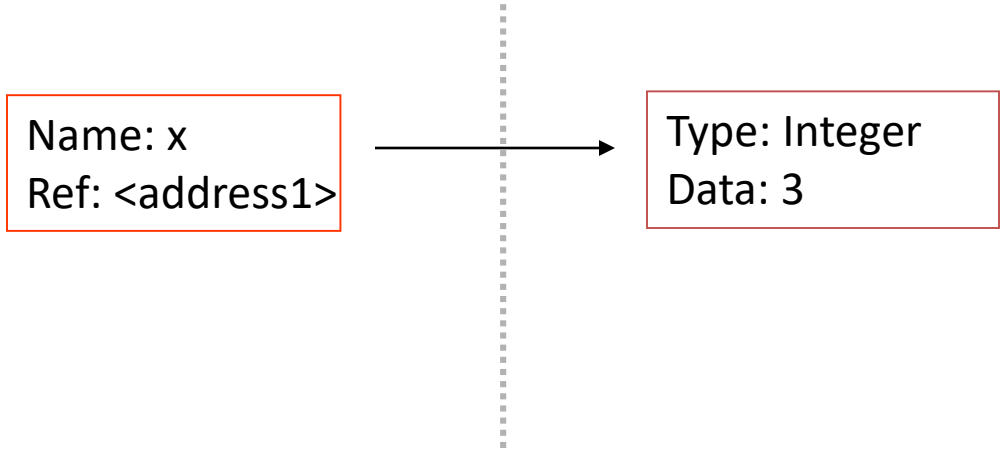
```
name = input('Hello, what is your name?  ')
print('Hi, ', name)
```

Understanding Assignment

Assignment

- For simple built-in datatypes (integers, floats, strings), assignment behaves as you would expect:

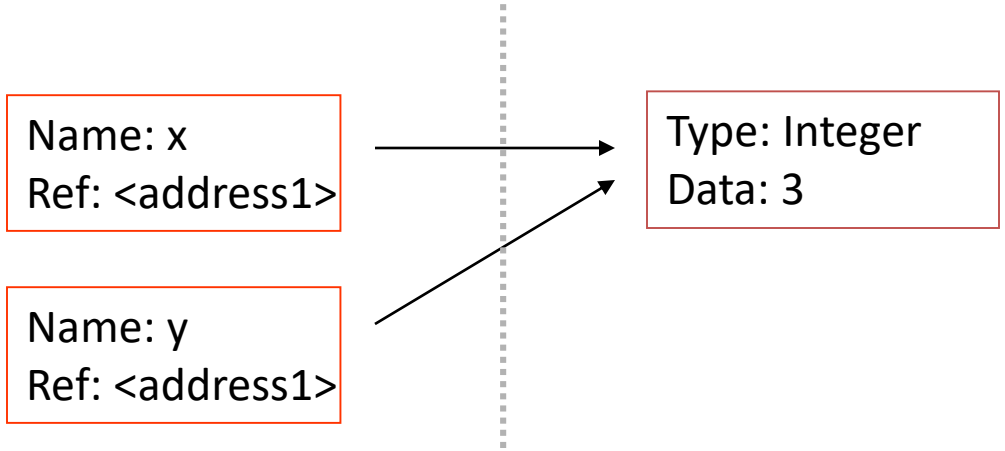
```
>>> x = 3          # Creates 3, name x refers to 3
>>> y = x          # Creates name y, refers to 3.
>>> y = 4          # Creates ref for 4. Changes y.
>>> print x        # No effect on x, still ref 3.
3
```



Assignment

- For simple built-in datatypes (integers, floats, strings), assignment behaves as you would expect:

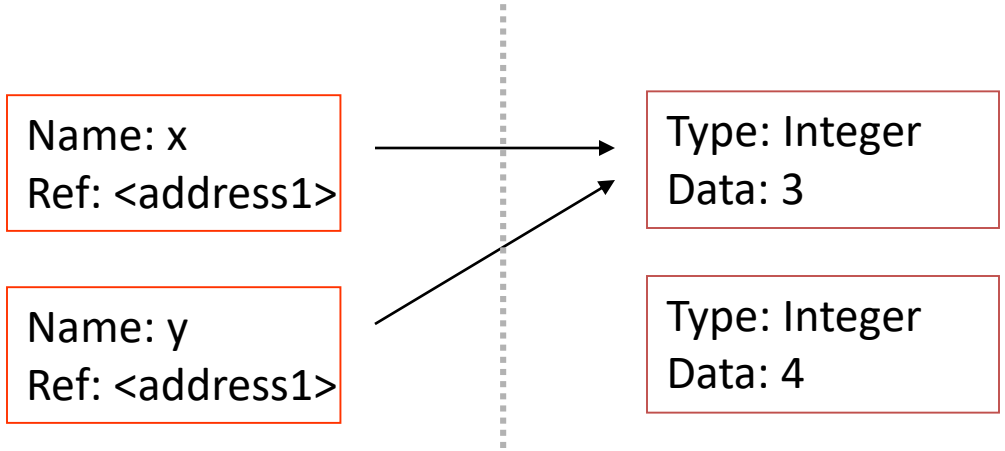
```
>>> x = 3          # Creates 3, name x refers to 3
>>> y = x          # Creates name y, refers to 3.
>>> y = 4          # Creates ref for 4. Changes y.
>>> print x        # No effect on x, still ref 3.
3
```



Assignment

- So, for simple built-in datatypes (integers, floats, strings), assignment behaves as you would expect:

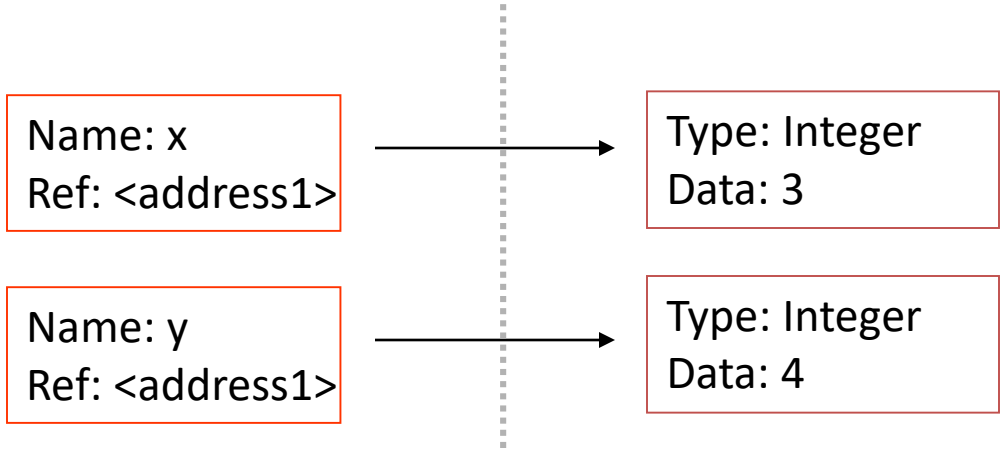
```
>>> x = 3          # Creates 3, name x refers to 3
>>> y = x          # Creates name y, refers to 3.
>>> y = 4          # Creates ref for 4. Changes y.
>>> print x        # No effect on x, still ref 3.
3
```



Assignment

- So, for simple built-in datatypes (integers, floats, strings), assignment behaves as you would expect:

```
>>> x = 3          # Creates 3, name x refers to 3
>>> y = x          # Creates name y, refers to 3.
>>> y = 4          # Creates ref for 4. Changes y.
>>> print x        # No effect on x, still ref 3.
3
```



Assignment

- For other data types (lists, dictionaries, user-defined types), assignment works differently.
  - These datatypes are **"mutable."**
  - When we change these data, we do it *in place.*
  - We don't copy them into a new memory address each time.
  - If we type y=x and then modify y, both x and y are changed!
  - We'll talk more about "mutability" later.

immutable

```
>>> x = 3
>>> y = x
>>> y = 4
>>> print x
3
```

mutable

x = some mutable object
y = x
make a change to y
look at x
x will be changed as well

Multiple Assignment

- You can also assign to multiple names at the same time.

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

Understanding Operator

Mathematical Operator

運算子	功能
`x + y`	X加Y
`x - y`	X減Y
`x * y`	X乘Y
`x / y`	X除以Y
`x // y`	X除以Y，只取整數解
`x % y`	求X除以Y的餘數
`x ** y`	X的Y次方

Comparison Operator

運算子	效果
`x < y`	X是否小於Y
`x <= y`	X是否小於等於Y
`x > y`	X是否大於Y
`x >= y`	X是否大於等於Y
`x == y`	X是否等於Y
`x != y`	X是否不等於Y

Boolean

運算子	效果
`a or b`	A或B其中一個條件成立就回傳True
`a and b`	A或B兩個條件都成立才回傳True
`not A`	如果A為True，則回傳False，反之則回傳True

Understanding Container

Range

- Store the variables in specific range.
- The content cannot be modified once range is created.

- Range(stop)
  - stop：停止點
- Range(start, stop)
  - start：起始點
  - stop：停止點
- Range(start, stop, step)
  - start：起始點
  - stop：停止點
  - step：間隔

```
r1 = range(10)
r2 = range(5, 50, 5)

print(type(r1))
print(r1)
print(r2)
```

Store 0~9

- 若沒有給起始值，將預設為0
- 若沒有給間隔，將預設為1
- 遇到停止點後，創造的過程就會終止，因此Range中的數字將不會包含停止點

Tuple

- Store a set of data.
- Data in tuple can be different types.
- The content cannot be modified once tuple is created.

```
t1 = 10, 20
# it can hold different types of data
t2 = 10, 'hello world'

print(type(t1))
print(t1)
print(t2)
```

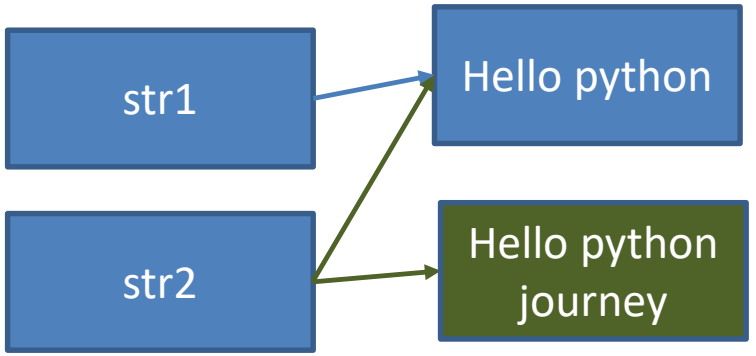
List

- Array of Python
- The type in List can be different.

```
arr1 = [1, 2, 3]
arr2 = [10, 'hello world', 8.7]
arr1[0] = [1, 2, 3]

print(type(arr1))
print(arr1)
print(arr2)
```

String



- The content of string cannot be modified.
- Combine string: "+"
- "is" can be used to see whether two string use the same memory.
- Split & join are two important actions in string type.

```
str1 = 'hello python'
str2 = str1
# str2[0] = 'y'
# a = a + b could be written as a += b
str2 += ' journey'
print(str2 is str1)

print(str1)
result = str2.split(' ')
print(result)
result_back = '***'.join(result)
print(result_back)
```

['hello', 'python', 'journey']

Hello***python***journey

Tuples, Lists, and Strings

- Tuples are defined using parentheses (and commas).

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Lists are defined using square brackets (and commas).

```
>>> li = ["abc", 34, 4.34, 23]
```

- Strings are defined using quotes (", ').

```
>>> st = "Hello World"
>>> st = 'Hello World'
```

Tuples, Lists, and Strings

- We can access individual members of a tuple, list, or string using square bracket "array" notation.

```
 tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]      # Second item in the tuple.
'abc'

li = ["abc", 34, 4.34, 23]
>>> li[1]      # Second item in the list.
34

st = "Hello World"
>>> st[1]   # Second character in string.
'e'
```

Looking up an Item

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Positive index: count from the left, starting with 0.

```
>>> t[1]
'abc'
```

Negative lookup: count from right, starting with –1.

```
>>> t[-3]
4.56
```

Slicing: Return Copy of a Subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Return a copy of the container with a subset of the original members. Start copying at the first index, and stop copying before the second index.

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

You can also use negative indices when slicing.

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

Slicing: Return Copy of a Subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Omit the first index to make a copy starting from the beginning of the container.

```
>>> t[:2]
(23, 'abc')
```

Omit the second index to make a copy starting at the first index and going to the end of the container.

```
>>> t[2:]
(4.56, (2,3), 'def')
```

Copying the Whole Container

You can make a copy of the whole tuple using `[:]`.

```
>>> t[:]
   (23, 'abc', 4.56, (2,3), 'def')
```

So, there's a difference between these two lines:

```
>>> list2 = list1    # 2 names refer to 1 ref
                     # Changing one affects both

>>> list2 = list1[:]    # Two copies, two refs
                        # They're independent
```

Get Data from Sequence

- seq[start:stop:step]
- Step default is 1.
- -1 represents the last variable.
- Start default is 0.
- If you do not set ‘stop’, you can get all variables after start.

```
str1 = 'hello world'
arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# mind the stop
arr2 = arr1[0:5]
# -1 represent the last element
arr3 = arr1[0:-1:2]
# you can ignore the args...
arr4 = arr1[:]

print(arr2)
print(arr3)
print(arr4)
print(arr4 is arr1)
print(str1[5:])
# print(arr1[:-1])
```

操作	描述
`x in s`	檢查X是否存在於S這個容器之中
`x not in s`	檢查X是否不存在於S這個容器之中
`s + t`	容器S與容器T的內容相加
`s * n`	三個容器S => `s + s + s`
`len(s)`	取得容器的長度 (裡面有幾個元素的意思)
`min(s)`	取得容器內的最小值 (前提是裡面的元素要能比大小啊!)
`max(s)`	取得容器內的最大值
`s.index(x[,i[,j]])`	X元素在S容器的索引值，如果有給`i, j`就只會在index為`i~j`的範圍找
`s.count(x)`	X這個元素在S這個容器內出現幾次

操作	描述
`s[i] = x`	index為`i`的元素的內容置換為X
`s[i:j] = t`	index從`i`到`j`的元素內容置換為X
`s[i:j:k] = t`	index從`i`到`j`的元素，以`step`為k的方式，將內容置換為X
`del s[i:j]`	把index從`i`到`j`的元素刪除
`del s[i:j:k]`	index從`i`到`j`的元素，以`step`為k的方式刪除元素
`s.append(x)`	將X塞到S容器的最後面
`s.clear()`	將S容器的內容全部刪除(same as del s[:])
`s.copy()`	複製S容器(same as s[:])
`s.extend(t)`	同`s = s + t`
`s.insert(i,x)`	在S容器index為`i`的位置將X插入，原有的元素(們)將會往後移
`s.pop([i])`	將index為`i`的元素取出，並將其移出容器
`s.remove(x)`	刪除第一個找到的X
`s.reverse()`	讓容器的內容順序顛倒

What’ s the difference between tuples and lists?

Tuples: Immutable

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in -toplevel-
    tu[2] = 3.14
TypeError: object doesn't support item
 assignment
```

You're not allowed to change a tuple *in place* in memory; so, you can't just change one element of it.

But it's always OK to make a fresh tuple and assign its reference to a previously used name.

```
>>> t = (1, 2, 3, 4, 5)
```

Lists: Mutable

```
>>> li = ['abc', 23, 4.34, 23]
>>> li[1] = 45
>>> li
['abc', 45, 4.34, 23]
```

We can change lists *in place*. So, it's ok to change just one element of a list. Name li still points to the same memory reference when we're done.

Operations on Lists Only

```
>>> li = [1, 2, 3, 4, 5]

>>> li.append('a')
>>> li
[1, 2, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>>li
[1, 2, 'i', 3, 4, 5, 'a']
```

Operations on Lists Only

The 'extend' operation is similar to concatenation with the + operator. But while the + creates a fresh list (with a new memory reference) containing copies of the members from the two inputs, the extend operates on list **li** in place.

```
>>> li.extend([9, 8, 7])
>>>li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7]
```

Extend takes a list as an argument. Append takes a singleton.

```
>>> li.append([9, 8, 7])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7, [9, 8, 7]]
```

Operations on Lists Only

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')      # index of first occurrence
1

>>> li.count('b')      # number of occurrences
2

>>> li.remove('b')     # remove first occurrence
>>> li
 ['a', 'c', 'b']
```

Operations on Lists Only

```
>>> li = [5, 2, 6, 8]

>>> li.reverse()     # reverse the list *in place*
>>> li
[8, 6, 2, 5]

>>> li.sort()        # sort the list *in place*
>>> li
[2, 5, 6, 8]

>>> li.sort(some_function)
    # sort in place using user-defined comparison
```

Tuples vs. Lists

- Lists slower but more powerful than tuples.
  - Lists can be modified, and they have lots of handy operations we can perform on them.
  - Tuples are immutable and have fewer features.

- We can always convert between tuples and lists using the list() and tuple() functions.

```
li = list(tu)
tu = tuple(li)
```

Understanding Ifelse

If….else

- Conditions don’ t need to put in ().
- Add “:” after every condition.
- Use “**indentation**” to represent the action if condition is achieved.
- else if in Python is “**elif**”

```
grade = 90

# there's no ()
if grade >= 90:
    print('Excellent!')
elif grade >= 60:
    print('Good enough!')
else:
    print('Loser!')
```

Understanding Loop

for

- For in Python is designed for getting elements from container.

```
arr1 = [2, 4, 6, 8, 10]
str1 = 'hello python'

for i in range(10):
    print(i)
print('***\n')

for i in range(len(arr1)):
    print(arr1[i])
print('***\n')

for i in arr1:
    print(i)
print('***\n')

for i in str1:
    print(i)
print('***\n')

# for i in arr1:
#     i += 1
# print(arr1)
```

If the lens of arr1 is 5, generate 0~4 in range.

C++ vs. Python

C++

```cpp
int getMax(size_t size, int const* array){
    // Find the maximum element in array
    /*
      It is just an example.
      Python has nice build-in function
      called "max()".
    */
    int max = -1 * INT_MAX;
    for (size_t i=0; i<size; ++i) {
        if (array[i] > max)
            max = array[i];
    }
    return max;
}
```

Python

```python
def getMax(array):
    # Find the maximum element in array
    """
      It is just an example.
      Python has nice build-in function
      called "max()".
    """
    max = -1* sys.maxint -1
    for element in array:
        if element > max:
            max = element
    return max
```

Practice

- Leetcode https://leetcode.com/
- **Problem 1313: Decompress Run-Length Encoded List**
- **Problem 1431: Kids With the Greatest Number of Candies**
- **Problem 1480: Running Sum of 1d Array**
- **Problem 1528: Shuffle String**
- **Problem 1672: Richest Customer Wealth**
- **Problem 1512: Number of Good Pairs**

HW

- Capture your pass result of six problems on Leetcode in the word document.