Python 教程學習筆記

通過之前的學習，我們已經對 Python 了解了一些。現在，根據官網文檔我們繼續補充一些 Python 的知識。

代碼流的控制

type

```
print(type(1))
```

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

```
print(type('a'))
```

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

type 函數很有用，來打印對象的類型。

range

range 函數是非常很有用的。

```
for i in range(5):
  print(i, end = ' ')
```

```
0 1 2 3 4
```

```
for i in range(2, 6, 2):
  print(i, end = ' ')
```

```
2 4
```

看 range 函數的定義。

```
class range(Sequence[int]):
    start: int
    stop: int
    step: int
```

可見是一個類。

```
print(range(5))
```

```
range(0, 5)
```

而不是：

```
[0,1,2,3,4]
```

繼續。

```
print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```

為什麼。看 `list` 的定義。

```
class list(MutableSequence[_T], Generic[_T]):
```

list的定義是 `list(MutableSequence[_T], Generic[_T]):`。而 `range`的定義是 `class range(Sequence[int])`。list 繼承了 `MutableSequence`。range 繼承了 `Sequence`。

繼續往下找是這樣的。

```
Sequence = _alias(collections.abc.Sequence, 1)
MutableSequence = _alias(collections.abc.MutableSequence, 1)
```

這裡我們不明白它倆的關係。但大概我們知道了為什麼可以這樣寫 `list(range(5))`。

函數參數

來看函數的補充知識。

```
def fn(a = 3):
  print(a)

fn()
```

```
3
```

這是給參數一個默認值。

```
def fn(end: int, start = 1):
  i = start
  s = 0
  while i < end:
    s += i
    i += 1
  return s

print(fn(10))
```

```
45
```

end 是必須要有的參數。注意到要把必須要有的參數寫在最前面。

```
def fn(start = 1, end: int):

    def fn(start = 1, end: int):
                                ^
SyntaxError: non-default argument follows default argument
```

注意到 end 是 non-default argument。start 是 default argument。意思是說，非默認參數跟在了默認參數後面。就是說必須把非默認參數放在所有默認參數前面。start 是默認參數，即是不傳遞的話，默認已經有值了。

```
def fn(a, /, b):
  print(a + b)

fn(1, 3)
```

這裡/來把參數類型分隔。有兩種形式的傳遞參數方式。一種是靠位置來傳遞，一種是靠指定關鍵詞來傳遞。

```
def fn(a, /, b):
  print(a + b)

fn(a=1, 3)
```

```
    fn(a=1, 3)
              ^
SyntaxError: positional argument follows keyword argument
```

這樣寫就不行。`a=1` 表示這是靠關鍵詞來傳遞參數。這把它當做了一個關鍵詞參數。而 b 則是位置參數。

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

注意這裡定義函數時，用上/和 * 已經隱含了各參數的傳遞類型。所以得按規則傳遞。

```
def fn(a, /, b):
  print(a + b)

fn(1, b=3)
```

上面這樣就沒報錯。

```
def fn(a, /, b, *, c):
  print(a + b + c)

fn(1, 3, 4)
```

```
    fn(1, 3, 4)
TypeError: fn() takes 2 positional arguments but 3 were given
```

`fn` 只能接收 2 個位置參數，但是給了 3 個。

```
def fn(a, /, b, *, c):
  print(a + b + c)

fn(a = 1, b=3, c=4)
```

```
    fn(a = 1, b=3, c=4)
TypeError: fn() got some positional-only arguments passed as keyword arguments: 'a'
```

fn 有一些參數只能靠位置傳遞的現在則是用關鍵詞來傳遞。

映射形式的參數

```
def fn(**kwds):
  print(kwds)

fn(**{'a': 1})
```

```
{'a': 1}
```

```
def fn(**kwds):
  print(kwds['a'])

d = {'a': 1}
fn(**d)
```

```
1
```

可見 `**` 就是把參數展開。

```
def fn(a, **kwds):
  print(kwds['a'])

d = {'a': 1}
fn(1, **d)
```

```
TypeError: fn() got multiple values for argument 'a'
```

像 `fn(1, **d)` 這樣調用函數時，展開就是 `fn(a=1, a=1)`。所以會出錯。

```
def fn(**kwds):
  print(kwds['a'])

d = {'a': 1}
fn(d)
```

```
TypeError: fn() takes 0 positional arguments but 1 was given
```

如果像 `fn(d)` 這樣調用函數，會被當成是位置參數，而不是展開成關鍵詞參數。

```
def fn(a, / , **kwds):
  print(kwds['a'])

d = {'a': 1}
fn(1, **d)
```

這樣卻行。說明位置參數和映射形式的參數可以同名。

```
def fn(a, / , a):
  print(a)

d = {'a': 1}
fn(1, **d)
```

```
SyntaxError: duplicate argument 'a' in function definition
```

這樣就出錯了。注意這幾種情況的微妙關係。

```
def fn(a, / , **kwds):
  print(kwds['a'])

fn(1, **[1,2])
```

```
TypeError: __main__.fn() argument after ** must be a mapping, not list
```

`**` 後面必須跟著映射。

可迭代類型的參數

```
def fn(*kwds):
  print(kwds)

fn(*[1,2])
```

```
(1, 2)
```

```
def fn(*kwds):
  print(kwds)

fn(*1)
```

```
TypeError: __main__.fn() argument after * must be an iterable, not int
```

`*` 必須跟著 `iterable`。

```
def fn(a, *kwds):
  print(type(kwds))

fn(1, *[1])
```

```
<class 'tuple'>
```

打印一下類型。這也是為什麼上面輸出 `(1,2)`，而不是 `[1,2]`。

```
def fn(*kwds):
  print(kwds)

fn(1, *[1])
```

```
(1, 1)
```

注意到這裡調用 `fn(1, *[1])` 時，就把參數展開了，成了 `fn(1,1)`。然後在 `fn(*kwds)` 解析的時候，`kwds` 又把 `1,1` 變成了元組 `(1,1)`。

```
def concat(*args, sep='/'):
  return sep.join(args)

print(concat('a','b','c', sep=','))
```

```
a,b,c
```

Lambda 表達式

lambda 就是把函數當做變量來保存。還記得在「解謎計算機科學」一文中所說的嗎。

```
def incrementor(n):
  return lambda x: x + n

f = incrementor(2)
print(f(3))
```

```
5
```

再看一個例子。

```
pairs = [(1, 4), (2,1), (0, 3)]

pairs.sort(key = lambda pair: pair[1])

print(pairs)
```

```
[(2, 1), (0, 3), (1, 4)]
```

```
pairs = [(1, 4), (2,1), (0, 3)]

pairs.sort(key = lambda pair: pair[0])

print(pairs)
```

```
[(0, 3), (1, 4), (2, 1)]
```

pair[0] 時，就按第一個數排序。pair[1] 時，就按第二個數排序。

文檔註釋

```
def add():
  """add something
  """
  pass
```

```
print(add.__doc__)
```

```
add something
```

函數簽名

```
def add(a:int, b:int) -> int:
  print(add.__annotations__)
  return a+b

add(1, 2)
```

```
{'a': <class 'int'>, 'b': <class 'int'>, 'return': <class 'int'>}
```

數據結構

列表

```
a = [1,2,3,4]

a.append(5)
print(a)   # [1, 2, 3, 4, 5]

a[len(a):] = [6]
print(a)   # [1, 2, 3, 4, 5, 6]

a[3:] = [6]
print(a)   # [1, 2, 3, 6]

a.insert(0, -1)
print(a)   # [-1, 1, 2, 3, 6]

a.remove(1)
print(a)   # [-1, 2, 3, 6]

a.pop()
```

```
print(a)  # [-1, 2, 3]

a.clear()
print(a)  # []

a[:] = [1, 2]
print(a.count(1)) # 1

a.reverse()
print(a)   # [2, 1]

b = a.copy()
a[0] = 10
print(b)   # [2, 1]
print(a)   # [10, 1]

b = a
a[0] = 3
print(b)  # [3, 1]
print(a)  # [3, 1]
```

列表構造

```
print(3 ** 2)   # 9
print(3 ** 3)   # 27
```

先來學一種運算，`**`。這表示次方的意思。

```
sq = []
for x in range(10):
  sq.append(x ** 2)

print(sq)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

接著試試用 `map`。

```python
a = map(lambda x:x, range(10))
print(a)
# <map object at 0x103bb0550>
print(list(a))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

sq = map(lambda x: x ** 2, range(10))
print(list(sq))
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

sq = [x ** 2 for x in range(10)]
print(sq)
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

可見 `for` 是很靈活的。

```python
a = [i for i in range(5)]
print(a)
# [0, 1, 2, 3, 4]

a = [i+j for i in range(3) for j in range(3)]
print(a)
# [0, 1, 2, 1, 2, 3, 2, 3, 4]

a = [i for i in range(5) if i % 2 == 0]
print(a)
# [0, 2, 4]

a = [(i,i) for i in range(3)]
print(a)
# [(0, 0), (1, 1), (2, 2)]
```

嵌套列表構造

```python
matrix = [[(i+j*4) for i in range(4)] for j in range(3)]
print(matrix)
# [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

```
t = []
for j in range(3):
  t.append([(i+j*4) for i in range(4)])
print(t)
# [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

注意這兩段代碼的方式。即是說：

```
[[(i+j*4) for i in range(4)] for j in range(3)]
```

也就相當於：

```
for j in range(3):
   [(i+j*4) for i in range(4)]
```

也即相當於：

```
for j in range(3):
  for i in range(4):
      (i+j*4)
```

所以這方便用來做矩陣轉置。

```
matrix = [[(i+j*4) for i in range(4)] for j in range(3)]
print(matrix)
#  [[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]

mt = [[row[j] for row in matrix] for j in range(4)]
print(mt)
# [[0, 4, 8], [1, 5, 9], [2, 6, 10], [3, 7, 11]]

print(list(zip(*matrix)))
[(0, 4, 8), (1, 5, 9), (2, 6, 10), (3, 7, 11)]
```

del

```
a = [1, 2, 3, 4]
```

```
del a[1]
print(a)  # [1, 3, 4]

del a[0:2]
print(a) # [4]

del a
print(a) # NameError: name 'a' is not defined
```

字典

```
ages = {'li': 19, 'wang': 28, 'he' : 7}
for name, age in ages.items():
    print(name, age)

# li 19
# wang 28
# he 7

for name in ages:
    print(name)

# li
# wang
# he

for name, age in ages:
     print(name)

ValueError: too many values to unpack (expected 2)

for i, name in enumerate(['li', 'wang', 'he']):
    print(i, name)

# 0 li
# 1 wang
```

```
# 2 he

print(reversed([1, 2, 3]))
# <list_reverseiterator object at 0x10701ffd0>

print(list(reversed([1, 2, 3])))
# [3, 2, 1]
```

模塊

腳本方式調用模塊

```
import sys

def f(n):
    if n < 2:
        return n
    else:
        return f(n-1) + f(n-2)

if __name__ == "__main__":
    r = f(int(sys.argv[1]))
    print(r)
```

```
% python fib.py 3
2

% python -m fib 5
5
```

dir

```
import fib

print(dir(fib))
```

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec_
```

“ ‘python import builtins print(dir(builtins))

[‘ArithmeticError’ , ‘AssertionError’ , ‘AttributeError’ , ‘BaseException’ , ‘BlockingIOError’ , ‘BrokenPipeError’ , ‘BufferError’ , ‘BytesWarning’ , ‘ChildProcessError’ , ‘ConnectionAbortedError’ , ‘ConnectionError’ , ‘ConnectionRefusedError’ , ‘ConnectionResetError’ , ‘DeprecationWarning’ , ‘EOFError’ , ‘Ellipsis’ , ‘EnvironmentError’ , ‘Exception’ , ‘False’ , ‘FileExistsError’ , ‘FileNotFoundError’ , ‘FloatingPointError’ , ‘FutureWarning’ , ‘GeneratorExit’ , ‘IOError’ , ‘ImportError’ , ‘ImportWarning’ , ‘IndentationError’ , ‘IndexError’ , ‘InterruptedError’ , ‘IsADirectoryError’ , ‘KeyError’ , ‘KeyboardInterrupt’ , ‘LookupError’ , ‘MemoryError’ , ‘ModuleNotFoundError’ , ‘NameError’ , ‘None’ , ‘NotADirectoryError’ , ‘NotImplemented’ , ‘NotImplementedError’ , ‘OSError’ , ‘OverflowError’ , ‘PendingDeprecationWarning’ , ‘PermissionError’ , ‘ProcessLookupError’ , ‘RecursionError’ , ‘ReferenceError’ , ‘ResourceWarning’ , ‘RuntimeError’ , ‘RuntimeWarning’ , ‘StopAsyncIteration’ , ‘StopIteration’ , ‘SyntaxError’ , ‘SyntaxWarning’ , ‘SystemError’ , ‘SystemExit’ , ‘TabError’ , ‘TimeoutError’ , ‘True’ , ‘TypeError’ , ‘UnboundLocalError’ , ‘UnicodeDecodeError’ , ‘UnicodeEncodeError’ , ‘UnicodeError’ , ‘UnicodeTranslateError’ , ‘UnicodeWarning’ , ‘UserWarning’ , ’ Value