Python
Lambda Function 匿名函式

臺北科技大學資訊工程系

Lambda

- lambda 參數1, 參數2, ..: 運算式A if 關係運算式 else 運算式B
  - (inline anonymous function)
  - 提供簡易功能 ，只處理一個運算式，回傳一個值。

```
lambda param1, param2, ... : expression

def fun(param1, param2, ...) :
    return expression
```

- 其中expression不能放assignment，=等號。

```
def func(x, y, z):
    return x + y + z
func2 = lambda x,y,z : x+y+z
print(func(1, 2, 3))
print(func2(1, 2, 3))
```

```
Output
6
6
```

Lambda vs Function

- Lambda不需定義名稱，一般函式(Function)需定義名稱。
- Lambda函式只能有一行運算式，一般函式(Function)可以有多行運算式。
- Lambda每一次運算完會自動回傳結果，一般函式(Function)要回傳結果須加上 return 關鍵字。

```
max = lambda m, n: m if m > n else n
print(max(10, 3))  # 顯示 10
```

```
def max(m, n):
    return m if m > n else n

print(max(10, 3))  # 顯示 10
```

- 若 function 不好重複使用，可傳遞參數給lambda 而非一般 function
- lambda式典型用法是搭配filter()和map()內建函式

Exercise

- Output

```
score = int(input('請輸入分數：'))
level = score // 10
{
    10 : lambda: print('Perfect'),
    9  : lambda: print('A'),
    8  : lambda: print('B'),
    7  : lambda: print('C'),
    6  : lambda: print('D')
}.get(level, lambda: print('E'))()
```

Sorted函式

- 基本排序
- 反向排序

```
x = [4, 2, 5, 3, 1]
y = sorted(x)
z = sorted(x, reverse = True)  #反向
print(y)
print(z)
x.sort()
print(x)
```

- 自訂排序鍵值函數
  - 根據第三個元素排序：

```
scores = [
    ('Jane', 'B', 12),
    ('John', 'A', 15),
    ('Dave', 'B', 11)]
print(sorted(scores, key = lambda s: s[2]))
```

```
[('Dave', 'B', 11), ('Jane', 'B', 12), ('John', 'A', 15)]
```

Sorted函式

- 對應排序：使用字典

```
[[1, 'D'], [2, 'H'], [5, 'C'], [7, 'S']]
[[7, 'S'], [5, 'C'], [2, 'H'], [1, 'D']]
[[7, 'S'], [2, 'H'], [1, 'D'], [5, 'C']]
[[5, 'C'], [1, 'D'], [2, 'H'], [7, 'S']]
[[5, 'C'], [1, 'D'], [2, 'H'], [7, 'S']]
[[7, 'S'], [2, 'H'], [1, 'D'], [5, 'C']]
['S3', 'H7', 'D8', 'D10', 'C12']
['S3', 'H7', 'D10', 'D8', 'C12']
```

```
01  data = [[1,'D'],[5,'C'],[2,'H'],[7,'S']]
02  match = {'S':1, 'H':2, 'D':3, 'C':4}
03  poker = 'D8 S3 D10 C12 H7'.split()
04
05  print(sorted(data))
06  print(sorted(data, reverse=True))                    # 逆排序 (大到小)
07  print(sorted(data, key=lambda k:k[1], reverse=True)) #以第1個正排序 (由小到大)
08  print(sorted(data, key=lambda k:(k[1], k[0])))       #先以第1個正排序，再以第0個正排序
09  print(sorted(data, key=lambda k:(k[1], -k[0])))      #先以第1個正排序，再以第0個逆排序
10
11  print(sorted(data, key=lambda k:match.get(k[1])))    #以第1個為key，透過字典對應後的值排序
12
13  print(sorted(poker, key=lambda k:match.get(k[0]))) #以第1個為key，透過字典對應後的值排序
14  print(sorted(poker, key=lambda k:(match.get(k[0]),k[1])))
```

Exercise

- 修改程式
  - 排序用 lambda 指定 key為avg，請將該敘述改以英文成績eng排序

```
def dataProcess(scores):
    grades=[]
    for stu in scores:
        name = stu[0]
        eng, math, phy = int(stu[1]), int(stu[2]), int(stu[3])
        avg = round((eng + math + phy)/3, 2)
        grades.append([name, [eng, math, phy], avg])
    sortedGrade = sorted(grades, key=lambda x: x[-1], reverse= True)
    print (sortedGrade)

dataProcess([['John', 90, 80, 90],['Mary', 100, 90, 85],['Tom', 80, 90, 70]])
```

```
sortedGrade = sorted(grades, key=lambda          ], reverse= True)
```

Nest function or Inner function

- "nested function" or "**inner function**"
  - 定義 function裡面的function。

```
def make_repeater(n):
  return lambda s: s*n
twice = make_repeater(2) #n=2
print (twice('word'))  #s=word
print (twice(5))         #s=5
```

```
Output
wordword
10
```

```
def function1(): # outer function
    print ("Hello from outer function")
    def function2(): # inner function
        print ("Hello from inner function")
    function2()

function1()
```

```
Hello from outer function
Hello from inner function
```

使用lambda實作inner function

```
def num1(x):
    print('x=',x)
    def num2(y):
        print('y=',y)
        return x * y
    return num2
res = num1(10)          #outer
print(res(5))           # inner
print(num1(10)(5))
```

```
x= 10
y= 5
50
```

```
# use lambda for inner function
def num1(x):
    print('x=',x)
    return lambda y: x * y

res = num1(10)          #outer
print(res(5))
print(num1(10)(5))
```

```
x= 10
50
```

- 定義inner function原因: **Encapsulation**

```
def outer_function(x):
    # Hidden from the outer code
    def inner_increment(x):
        return x + 2
    y = inner_increment(x)
    print(x, y)

inner_increment(5)
#outer_function(5)
```

```
NameError: name 'inner_increment' is not defined
```

END