Python Function

馬誠佑

2025/03/28

Function

Argument (引數)

- def functionName(para1, para2,...):

  tmp = para1 + para2

  .... Parameter (參數)

  ....

  return a, b, c

Ex:

```
def info( name, age ):
   print ("Name: “ + name)
   print ("Age “ + str(age))
   return

#呼叫info函数
info( age=50, name="miki" )
```

Ex:

```
def do_nothing():
   pass
```

Ex:

```
def agree():
   return True
```

Ex:

```
def echo(anything):
   return anything + ‘ ‘ +anything
```

```
def func():
    x = 100
    print('x in func(): ' + str(x))

x = 0
print('x: ' + str(x))
func()
print('x: ' + str(x))
```

```
x: 0
x in func(): 100
x: 0
```

```
def func():
    x = 100
    print('x in func(): ' +
str(x))
    return x

x = 0
print('x: ' + str(x))
x = func()
print('x: ' + str(x))
```

```
x: 0
x in func(): 100
x: 100
```

```
def test(array):
    array.append('aaaa')

def test2(array):
    array = [] + array
    array.append('aaaa')
    return array


a = [1, 2, 3, 14, 5]
print('Before execute test(), a = ' + str(a))
test(a)
print('After execute test(), a = ' + str(a))
b = test2(array = a)
print(' After execute test(), a = ' + str(a))
print(' After execute test(), b = ' + str(b))
```

```
Before execute test(), a = [1, 2, 3, 14, 5]
After execute test(), a = [1, 2, 3, 14, 5, 'aaaa']
After execute test2(), a = [1, 2, 3, 14, 5, 'aaaa']
After execute test2(), b = [1, 2, 3, 14, 5, 'aaaa', 'aaaa']
```

Positional argument (位置性引數)

- 值會依順序複製到對應的參數

Ex: def menu(wine, entree, dessert)

```
    return {‘wine’:wine, ’entree’:entree, ‘dessert’:dessert }
>>>menu(‘chardonnay’, ‘chicken’, ‘cake’)
{‘wine’:’chardonnay’, ‘entree’:’chicken’, ‘dessert’:’cake’}
```

關鍵字引數

- 用引數對應的參數名稱來指定引數

>>>menu(entree= ‘beef’, dessert= ‘bagel’, wine=‘bordeaux’)

{‘wine’: ’bordeaux’, ‘entree’:’beef’, ‘dessert’:’bagel’}

- 混合使用位置與關鍵字引數 (位置引數一定要在關鍵字引數前面)

>>>menu(‘frontenac’, dessert=‘flan’, entree=‘fish’)

{‘wine’: ‘frontenac’, ’entree’: ‘fish’, ‘dessert’: ‘flan’}

指定預設參數值

```
def menu(wine, entree, dessert='pudding'):
        return {'wine': wine, 'entree':entree, 'dessert':dessert}

>>>menu('chardonnay', 'chicken')
{'wine': 'chardonnay', 'entree':'chicken', 'dessert':'pudding'}

>>>menu('dunkelfelder', 'duck', 'doughnut')
{'wine': 'dunkelfelder', 'entree':' duck', 'dessert':' doughnut'}
```

預設參數計算時機

```
def buggy(arg, result=[]):
        result.append(arg)
        print(result)
>>>buggy(‘a’)
[‘a’]
>>>buggy(‘b’) #預期是[‘b’]
[‘a’, ‘b’]
```

預設的參數值僅在被定義時計算。

```
def buggy(arg, result=None):
        if result is None:
                result = []
        result.append(arg)
        print(result)
>>>buggy(‘a’)
[‘a’]
>>>buggy(‘b’) #預期是[‘b’]
[‘b’]
```

‘*’ in function

```
def print_args(*args):
        print(‘Positional tuple:’, args)
>>>print_args()
Positional tuple: ()
>>>print_args(3, 2, 1, ‘wait!’, ‘uh…’)
Positional tuple: (3, 2, 1, ‘wait!’, ‘uh…’)
```

Function參數裡面的’*’可以將數量不一定的位置引數組成一個參數值tuple

```
def print_more(required1, required2, *args):
        print(‘Need this one’, required1)
        print(‘Need this one too’, required2)
        print(‘All the rest:’, args)
>>>print_more(‘cap’, ‘gloves’, ‘scarf’, ‘monocle’, ‘mustache wax’)
Need this one: cap
Need this one too: gloves
All the rest: (’scarf’, ’monocle’, ‘mustache wax’)
```

‘**’ in function

```
def print_kwargs(**kwargs):
    print('Keyword arguments', kwargs)

print_kwargs()
print_kwargs(wine='Sangria', dessert='Panna Cotta', entree='mutton', drink='soda')
```

✓ 0.0s

```
Keyword arguments {}
Keyword arguments {'wine': 'Sangria', 'dessert': 'Panna Cotta', 'entree': 'mutton', 'drink': 'soda'}
```

Function 參數裡面使用’**’來將關鍵字引數組成dictionary

引數的順序是：

- 必須的位置引數
- 選用的位置引數 *args
- 選用的關鍵字引數 **kwargs

純關鍵字引數（Only supported by Python3）

```python
def print_data(data,*,start = 0, stop = 100):
    for value in data[start:stop]:
        print(value)
data = ['a','b','c','d','e','f']
print_data(data)
print('==========================')
print_data(data, start = 2)
print('==========================')
print_data(data, stop = 4)
```

✓ 0.0s

```
a
b
c
d
e
f
==========================
c
d
e
f
==========================
a
b
c
d
```

‘*’代表接下來的start, end 必須用具名引數來提供

可變與不可變引數

```
outside = ['one','two','three']
def test(arg):
    arg[1] = 'terrible!'

print(outside)
test(outside)
print(outside)
```

✓ 0.0s

```
['one', 'two', 'three']
['one', 'terrible!', 'three']
```

Docstrings

```
def func(anything):
    'docstring describes what this function does....'
    '''long description of what this function
    does ....'''
    return anything

help(func)
def func2(anything):
    '''long description of what this function
    does ....'''
    return anything
help(func2)
```

✓ 0.0s

```
Help on function func in module __main__:

func(anything)
    docstring describes what this function does....

Help on function func2 in module __main__:

func2(anything)
    long description of what this function
    does ....
```

Function as argument

```
def answer():
    print(42)

answer()
def run_something(func):
    func()

run_something(answer)
```

✓ 0.0s

```
42
42
```

```
def add(arg1, arg2):
    print(arg1+arg2)

def run_something_with_args(func, arg1, arg2):
    func(arg1,arg2)

run_something_with_args(add, 6, 4)
```

✓ 0.0s

```
10
```

内部函式

```
def outer(a,b):
    def inner(c,d):
        return c + d
    return inner(a,b)

print(outer(5,10))
```

✓ 0.0s

```
15
```

```
def knights(saying):
    def inner(quote):
        return f'we are the knights who say: {quote}'
    return inner(saying)

print(knights('ho ha!'))
```

✓ 0.0s

```
we are the knights who say: ho ha!
```

Closure

```
def knights2(saying):
    def inner2():
        return f'we are the knights who say: {saying}'
    return inner2

a = knights2('quack!')
b = knights2('huff!!')
print(type(a))
print(type(b))
print(a)
print(b)
print(a())
print(b())
```

✓ 0.0s

```
<class 'function'>
<class 'function'>
<function knights2.<locals>.inner2 at 0x106fb58b0>
<function knights2.<locals>.inner2 at 0x106fb50d0>
we are the knights who say: quack!
we are the knights who say: huff!!
```

內部函式可以當成 closure來使用，它是用其他的函式動態生成的函式，可以更改和記得在函式外面建立的變數的值

匿名函式：lambda

```
def edit_story(words, func):
    for word in words:
        print(func(word))

stairs = ['thud', 'meow', 'thud', 'hiss']

def enliven(word):
    return word.capitalize() + '!'

edit_story(stairs, enliven)

edit_story(stairs, lambda word: word.capitalize() + '!')
```

✓ 0.0s

```
Thud!
Meow!
Thud!
Hiss!
Thud!
Meow!
Thud!
Hiss!
```

```
people =[('john', 30), ('jane', 25), ('dave', 45)]
print(sorted(people, key=lambda x: x[1]))
```

✓ 0.0s

```
[('jane', 25), ('john', 30), ('dave', 45)]
```

Generator (產生器)

- 可迭代可能很大的序列且不需要在記憶體中一次建立或儲存整個序列。
- range(start,end)
- 每次呼叫迭代產生器時，它都會記住上次被呼叫時的位置，並且回傳下一個值。一般函式不會記住之前的呼叫，且永遠都用同一個狀態從它的第一行開始執行。

Generator Function

```
def my_range(start=0, end=10, step=1):
    num = start
    while num < end:
        yield num
        num += step

print(my_range)
ranger = my_range(1,5)
print(ranger)

for i in ranger:
    print(i)

for i in ranger:
    print('2:',i)
```

✓ 0.0s

```
<function my_range at 0x106dfe940>
<generator object my_range at 0x12044a5f0>
1
2
3
4
```

產生器只能執行一次。List, set, string, dictionary都會被放在記憶體中，但是產生器可以動態產生它的值，並且透過迭代一次送出一個值。Generator不會記得它們，所以你無法重新啟動或備份產生器。

```
t = (pair for pair in zip(['a','b'],[1,2]))
print(t)
for things in t:
    print(things)
```

✓ 0.0s

```
<generator object <genexpr> at 0x120461190>
('a', 1)
('b', 2)
```

Decorator (裝飾器)

```
def document_it(func):
    def new_function(*args, **kwargs):
        print('Running function:', func.__name__)
        print('Positional arguments:', args)
        print('Keyword arguments:', kwargs)
        result = func(*args, **kwargs)
        print('Result:', result)
        return result
    return new_function

def add_ints(a,b):
    return a + b

print(add_ints(4,6))

cooler_add_ints = document_it(add_ints)
cooler_add_ints(4,6)
```

✓ 0.0s

```
10
Running function: add_ints
Positional arguments: (4, 6)
Keyword arguments: {}
Result: 10
```

- Decorator是一種函式，它會接收一個函式，並且回傳另一個函式。

```
def document_it(func):
    def new_function(*args, **kwargs):
        print('Running function:', func.__name__)
        print('Positional arguments:', args)
        print('Keyword arguments:', kwargs)
        result = func(*args, **kwargs)
        print('Result:', result)
        return result
    return new_function

@document_it
def add_ints(a,b):
    return a + b

print(add_ints(4,6))
```

✓ 0.0s

```
Running function: add_ints
Positional arguments: (4, 6)
Keyword arguments: {}
Result: 10
10
```

Decorator的順序

```python
def document_it(func):
    def new_function(*args, **kwargs):
        print('Running function:', func.__name__)
        print('Positional arguments:', args)
        print('Keyword arguments:', kwargs)
        result = func(*args, **kwargs)
        print('Result:', result)
        return result
    return new_function

def square_it(func):
    def new_func(*args, **kwargs):
        result = func(*args, **kwargs)
        return result * result
    return new_func
```

```python
@document_it
@square_it
def add_ints(a,b):
    return a + b

add_ints(4,6)

@square_it
@document_it
def add_ints(a,b):
    return a + b

add_ints(4,6)
```

✓ 0.0s

```
Running function: new_func
Positional arguments: (4, 6)
Keyword arguments: {}
Result: 100
Running function: add_ints
Positional arguments: (4, 6)
Keyword arguments: {}
Result: 10

100
```

Namespace & scope-1

```
animal = 'wombat'
def print_global():
    print('inside print_global:', animal)

print('at top level:', animal)
print_global()
```

✓ 0.0s

```
at top level: wombat
inside print_global: wombat
```

```
def change_and_print_global():
    print('inside change_and_print_global:', animal)
    animal = 'bat'
    print('after the change:', animal)

change_and_print_global()
```

⊗ 0.2s

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Cell In[44], line 6
      3     animal = 'bat'
      4     print('after the change:', animal)
----> 6 change_and_print_global()

Cell In[44], line 2
      1 def change_and_print_global():
----> 2     print('inside change_and_print_global:', animal)
      3     animal = 'bat'
      4     print('after the change:', animal)

UnboundLocalError: local variable 'animal' referenced before assignment
```

Namespace & scope-2

```
def chang_local():
    animal = 'bat'
    print('inside change_local:', animal)

chang_local()
print(animal)
```

✓ 0.0s

```
inside change_local: bat
wombat
```

```
animal = 'wombat'
def change_and_print_global():
    global animal
    animal = 'bat'
    print('inside change_and_print_global:', animal)

change_and_print_global()
print(animal)
```

✓ 0.0s

```
inside change_and_print_global: bat
bat
```

Python有兩個函式可讓你讀取名稱空間的內容：

- locals() 會回傳一個字典，裡面有區域名稱空間的內容。
- Globals()會回傳一個字典，裡面有全域名稱空間的內容。

Recursion（遞迴）

```python
def flatten(matrix):
    for item in matrix:
        if isinstance(item, list):
            for subitem in flatten(item):
                yield subitem
        else:
            yield item

m = [1,2,[3,[4,5],6],[[7,8,9],[]]]
print(flatten(m))
print(list(flatten(m)))
```

✓ 0.0s

```
<generator object flatten at 0x123094cf0>
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
def flatten(matrix):
    for item in matrix:
        if isinstance(item, list):
            yield from flatten(item)
        else:
            yield item

print(flatten(m))
print(list(flatten(m)))
```

✓ 0.0s

```
<generator object flatten at 0x123097d60>
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Error handle: try and except

```
try:
        .....
except <ExceptionType1>:
        <handler1>
...
except <ExceptionTypeN>:
        <handlerN>
except:
        <handlerExcept> #有發生上面沒定義的error時執行
else:
        <process_else> #沒有任何error發生時執行
finally:
        <process_finally> #一定會執行
```

```
short_list = [1,2,3]
while True:
    value = input('Position [q to quit]? ')
    if value == 'q':
        break
    try:
        position = int(value)
        print(short_list[position])
    except IndexError as err:
        print('Bad index:', position)
    except Exception as other:
        print('Something else errors:', other)
```

✓ 42.0s

```
2
1
3
Bad index: 3
Something else errors: invalid literal for int() with base 10: 'two'
```

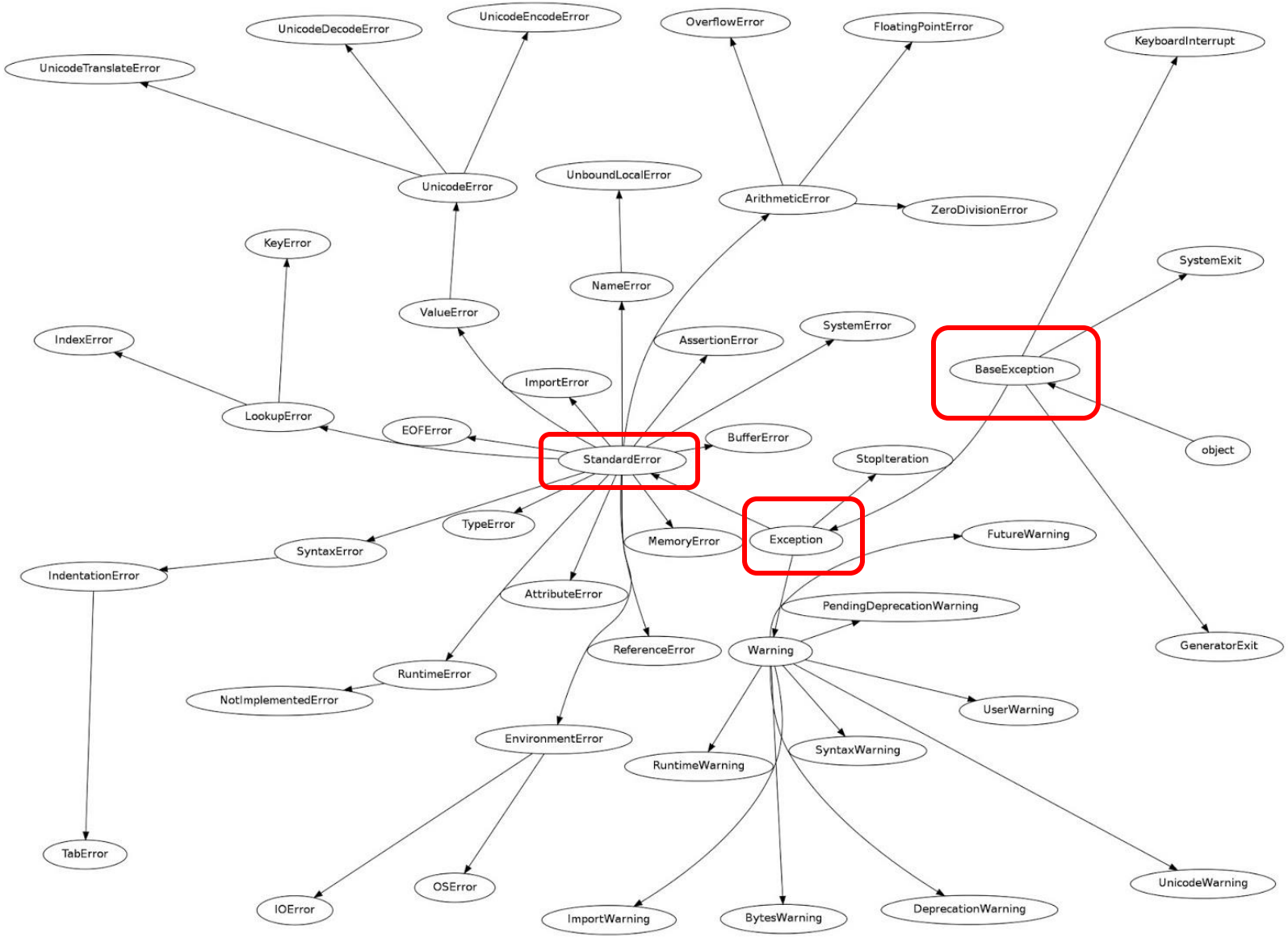
```
short_list = [1,2,3]
position = 5
try:
    short_list[position]
except:
    print('Need a position between 0 and', len(short_list) - 1, 'but got', position)
```

✓ 0.0s

```
Need a position between 0 and 2 but got 5
```

Built-in Exception

亦可定義你自己的ERROR TYPE!



HW5

1. 定義一個稱為good()的函式，用它回傳串列[‘Harry’, ‘Ron’, ‘Hermione’]
2. 定義一個稱為get_odds()的產生器函式，用它回傳range(n)的奇數。使用for loop來找到並印出第三個回傳值。
3. 定義一個稱為test的裝飾器，用它在一個函式被呼叫時印出’start’，在那個函式結束時印出’end’，並計算該函式執行的時間並印出（sec）。

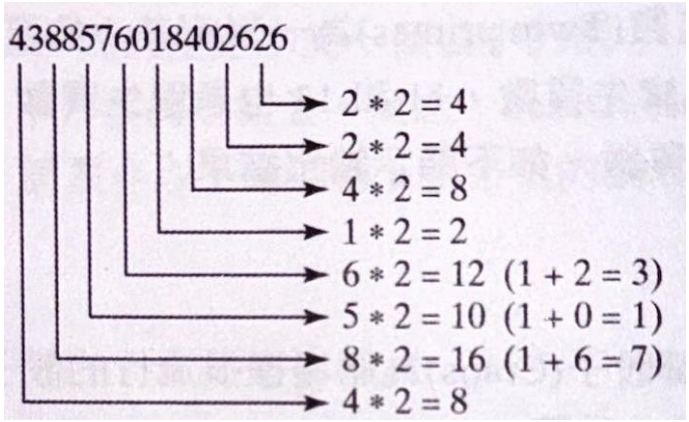
HW5

4.（財務金融：信用卡號碼驗證）信用卡號碼選循特定形式。信用卡號碼必須含有13 到16位數。起始位數必須是：

- 4代表 Visa cards
- 5代表 Master cards
- 37代表 American Express cards
- 6代表 Discover cards

1954年，IBM的Hans Luhn 提出一個驗證信用卡號碼的演算法。此算法用在判斷輸入的卡號是否正確，或在信用卡是否正確被掃描上很有用。信用卡號依此來產生有效驗證，這種作法稱作 Luhn check或 Mod 10 check，說明如下（為了解釋，這裡以卡號 4388576018402626 為例）：

1. 由右到左，每隔一位數（偶數位置）便取其兩倍數字。如果該位數的兩倍數字為兩位數，便將該兩位數相加，以取單一位數數字。



2. 接著將步驟1所有的單位數數字相加。
   4+4+8+2+3+1+7+8=37
3. 再由右到左，將出現於卡號奇數位置的所有位數相加。
   6+6+0+8+0+7+8+3=38
4. 將步驟2與步驟3的結果相加。
   37+38=75
5. 如果步驟4的結果能被10 整除，那麼卡號便是有效的；反之，則是無效。

比方說，卡號 4388576018402626 是無效的，但卡號 4388576018410707是有效的。

請撰寫一程式，提示使用者輸入信用卡號為整數。顯示該號碼是否有效。請使用以下函式設計您的程式：

```
# Return true if the card number is valid
def isValid(number) :

# Get the result from Step 2
def sumfDoubleEvenPlace(number) :

# Return this number if it is a single digit, otherwise, return
# the sum of the two digits
def getDigit (number) :
```

```
# Return sum of odd place digits in “number”
def sumOfOddPlace (number):

# Return true if the digit d is a prefix for “number”
def prefixMatched (number, d):

# Return the number of digits in d
def getSized(d):
```

HW5

5.（迴文整數）請撰寫帶有以下標頭的函式：

```
# Return the reversal of an integer, e.g. reverse(456) returns
# 654
def reverse(number):

# Return true if number is a palindrome
def isPalindrome (number):
```

請使用 reverse函式實作 isPalindrome。當某數字反過來與本身相同時，該數字便是迴文整數（palindrome integer）。請撰寫一測試程式，提示使用者輸入一個整數，接著回報該整數是否為迴文。