1. Introduction

Fig. 1.6 Photo by Alejandro Piñero Amerio on Unsplash

Note

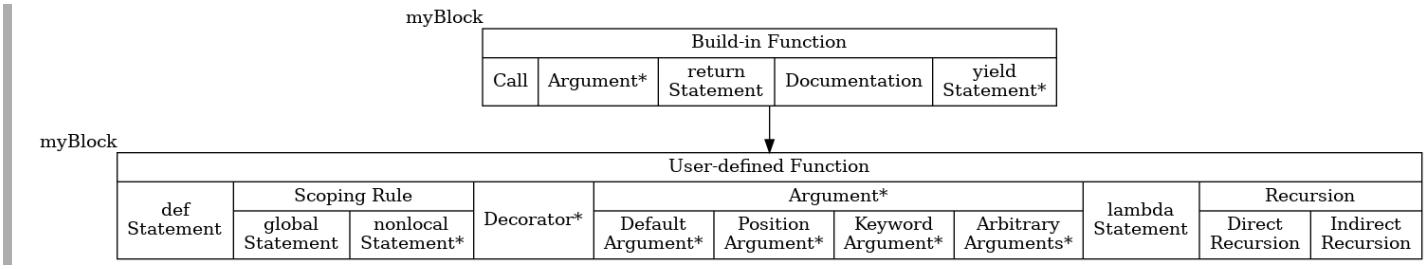
Outline (v20220501)

1. Overview
    a. return
    b. yield
    c. return vs. yield
2. Examples: return
    a. Ex1: printIt()
    b. Ex2: printIt() with Parameter
    c. Ex3: sum()
    d. Ex4: getTime()
    e. Ex5a: 1-Page Cal.: Chi
    f. Ex5b: 1-Page Cal.: Eng
    g. Ex5c: 1-Page Cal.: Function
    h. Ex5d: 1-Page Perpetual Cal.
3. Examples: yield
    a. Ex1: Generator Function
    b. Ex2: Infinite Sequence
    c. Ex3: Coroutine

Note

Roadmap

1. This topic: Function



2. Course: Python 1
3. Subject: Programming
4. Field
    a. Software Engineering (SE)
    b. Computer Science and Information Engineering (CSIE)
    c. Electrical/Electronics Engineering (EE)

1.1. Overview

1. 函數是經過組織且可重複使用的程式碼，是能用來實現單一或是相關聯的程式碼。
2. 巧妙的運用函數可以提高程式碼的重複利用率，避免一樣的事情卻需要重複寫好幾次的程式碼來執行。
3. 這跟迴圈的概念有點像，都是在重複利用程式碼；不同的地方在於函數是在需要時才呼叫使用。
4. Python 裡面已經內定好許多好用的函數，例如input()、print() 等等，不過這並不代表普羅不能自己定義一個函數來使用。
5. 畢竟Python 所內定的函數是針對大多數使用者都會使用到的功能，如果想要客製化的功能，還是必須要自己動手做。

Note

Mathematical Functions

```
y = f(x) = x + 5
y = f(1) = 1 + 5 = 6
y = f(6) = 6 + 5 = 11

y = f(x, y) = x - y + 5
y = f(1, 0) = 1 - 0 + 5 = 6
y = f(6, 4) = 6 - 4 + 5 = 7

y = f() = x + 5
y = f() = 0 + 5 = 5
y = f() = 0 + 5 = 5

solve f(x, y) = xy = 1 to get (x, y) ==> (1, 1), (2, 1/2), (3, 1/3), …
```

6. Syntax

```
def FunctionName([parameters]):
    function_suite
    [return[[(]expression[)]]]
```

7. 要自己定義一個函數，有以下幾個規定：

    a. 定義函數以def 關鍵字開頭，後面接想要的函數名稱和( )。

    b. 任何傳入參數和引數必須放在( ) 中間。

    c. 函數內容以：作為開始，同樣的，與for 迴圈及if 判斷式一樣要縮排。

1.1.1. return

1. return [expression] 結束函數，為選擇性的回傳一個值給呼叫此函數的程式碼。
2. 若是return 沒有後續的敘述，則相當於此函數不回傳任何東西，也就是所謂的回傳None。
3. If function must return value(s), return keyword has to add.
4. If function need NOT return value, return keyword can be omit.

    However, there are several types for this kind of return as listed as below.

```
                # return without anything here
return          # return with a keyword
return()        # return with a tuple (), NOT None
return(None)    # return with a None [Recommended]
```

5. Inside a group of programmers,

    we recommend use the last one as your code for clear description as well as let other programmers without guessing the semantics of your codes.

6. Example: Return None

    a. Code

```
def myFun1():
    pass

def myFun2():
    pass
    return

def myFun3():
    return()

def myFun4():
    return(None)

Ret1 = myFun1()
Ret2 = myFun2()
Ret3 = myFun3()
Ret4 = myFun4()

print(Ret1, type(Ret1))     # return None
print(Ret2, type(Ret2))     # return None
```

```
print(Ret3, type(Ret3))     # return a tuple, NOT None
print(Ret4, type(Ret4))     # return None [Recommended]
```

b. Output

```
None <class 'NoneType'>
None <class 'NoneType'>
() <class 'tuple'>
None <class 'NoneType'>
```

1.1.2. yield

1. The yield keyword in Python is used to **create a generator function**.
2. A **generator function** is defined like a normal function but whenever it needs to generate a value, it does so with the yield keyword rather than return.
3. Goal: to save more memory.
4. Code+Output

> Code | Output
>
> a. When [myGenerator()] is called, it returns a generator object without actually executing the function.
>
> b. Then the for loop calls [next()] on the generator object to get the next value from the generator until there are no more values left at which point a [StopIteration] exception is raised automatically.

```
def myGenerator():
    yield 1
    yield 2
    yield 3

if __name__ == '__main__':
    for v in myGenerator():
        print(v)
```

5. The key points include

   a. Generators compute values on demand, not upfront like functions.

   b. The yield keyword pauses execution and sends a value out of the generator.

   c. Calling a generator function returns a generator object.

   d. Calling next() on the generator executes up to the next yield statement.

   e. Generators have state that persists between yields.

6. Some common uses of generators with yield.

   a. Defining iterators that compute large sets of results lazily

   b. Implementing coroutines to produce or consume data streams

   c. Replacing callbacks for asynchronous programming

7. Summary

a. yield creates generator functions that return an iterator which computes values lazily by yielding them one by one.

b. This allows for iterators that can compute large sets of values without consuming large amounts of memory.

1.1.3. return vs. yield

1. Here's a comparison of return vs yield in Python functions along with their advantages, disadvantages and usage.

Item	return	yield
Goal	1. Returns a value from a function and exits. 2. No way to resume execution of the function. 3. Useful for general functions that compute single return values.	1. Returns a value from a generator function. 2. Pauses execution of the function, preserving state. 3. Can resume execution from last yield point later. 4. Allows defining iterators and generators simply.
Plus	1. Simple to use for general functions. 2. Familiar syntax and semantics for most programmers.	1. Pauses execution between values automatically. 2. Preserves state between calls. 3. Can act as an iterator and generator. 4. Simpler concurrency and data streaming.
Minus	1. Can't resume execution of the function later. 2. Not suitable for iterators, generators, coroutines.	1. Overhead of preserving state and resuming later. 2. More complex semantics.
Usage	Normal functions that compute single return values.	Generators, iterators, coroutines that produce series of values.

2. Summary

a. return gives a simple function exit

b. yield offers more power for advanced generators, iterators and coroutines in Python.

1.2. Examples: return

1.2.1. Ex1: printIt()

1. Source

Listing 1.2.1.2 /src/Function/p0810Function.py

```
'''
Created on 20150918日
@author: cph
'''
# 開始定義函數
def printit(str):   # 自定義printit 的函數，傳入參數為str，印出所傳入的str 值
    print(str)      # 說明這個函數的功能，任意函數都可以選擇性的加入字串來說明
```

```
    return          # 沒有敘述式，因此回傳值為none

# 現在可以呼叫剛剛所定義的函數
printit("呼叫printit函數")
printit("測試，再次呼叫printit函數")
```

2. Output

```
呼叫printit函數
測試，再次呼叫printit函數
```

3. 在呼叫函數的時候，函數名稱跟傳入的參數同時決定你所呼叫的是哪一個函數。
4. 如果傳入的參數不對，Python 會出現錯誤，顯示你傳入的參數不符合函數的規定。
5. 另外，一個函數也可能擁有0 個或多個輸入參數。

1.2.2. Ex2: printIt() with Parameter

1. Source

Listing 1.2.2.2 /src/Function/p0811FunctionParameter.py

```
'''
Created on 20150918
@author: cph
'''
# 開始定義函數
def printit(str):   # 自定義printit 的函數，傳入參數為str，印出所傳入的str 值
    print(str)      # 說明這個函數的功能，任意函數都可以選擇性的加入字串來說明
    return          # 沒有敘述式，因此回傳值為none

# 現在可以呼叫剛剛所定義的函數
printit("呼叫printit函數")
printit("測試，再次呼叫printit函數")
printit('test1', )          # 以傳入1參數, but add a comma來測試
printit('test1', 'test2')   # 以傳入2參數來測試
printit()                   # 以不傳入參數來測試
```

2. Output

```
呼叫printit函數
測試，再次呼叫printit函數
test1
Traceback (most recent call last):
  File
"W:\_D\Data.cph\Workspace\Python3\Py8.Doc.WhoNotes.Code.Sphinx.Gitlab.Web\docs\_static\Python3
line 14, in <module>
    printit('test1', 'test2')   # 以傳入2參數來測試
TypeError: printit() takes 1 positional argument but 2 were given
```

3. Return 可以將函數所執行的結果回傳到呼叫函數的地方。
4. 如同上面所提到的，return 可以沒有運算式，但是也可以回傳一到多個值。
5. 當需要回傳多個值的時候，以”,” 分隔所想要回傳的值即可。

1.2.3. Ex3: Sum()

1. Source 1: Fix boundary

Listing 1.2.3.1 /src/Function/p0812FunctionReturn.py

```
'''
Created on 20150918
@author: cph
'''
def Sum():              # 定義sum 函數
    i = 1               # 定義初始值為1
    iSum = 0            # 定義初始總和為0
    while i <= 100:     # 使用while 迴圈加總
        iSum += i
        i += 1
    return(iSum)        # 回傳sum

a = Sum()               # 呼叫sum 函數，並且將其回傳的值給a
print(a)
print(Sum())
```

2. Output 1

```
5050
5050
```

3. Return's expression can be optional in Python.

Note

a. Return's expression

```
# Return expression can be remove when function has nothing to return
def fun(x, y):
    pass

# Return expression can be nothing
def fun(x, y):
    pass
    return()

# Return expression can be 1 variable
def fun(x, y):
    pass
    return(x)

# Return expression can be 2 variables
def fun(x, y):
    pass
    return(x, y)

# Return expression can be an expression
def fun(x, y):
    pass
    return(x + y)

# Return expression can be another anonymous function
def fun(x, y):
    pass
    return(lambda x, y: x + y)
```

4. Source 2: Variable boundary

Listing 1.2.3.2 /src/Function/p0812FunctionReturn2.py

```
'''
@since: 20150918
@author: cph
'''
def sum(iB):            # 定義sum函數
    iA = 1              # 定義初始值為1
    iSum = 0            # 定義初始總和為0
    while iA <= iB:     # 使用while迴圈加總
        iSum += iA
        iA += 1
    return(iSum)        # 回傳iSum

iOut = sum(100)         # 呼叫sum 函數，並且將其回傳的值給iOut
print(iOut)
print(sum(10))
print(sum(10000))
```

5. Output 2

```
5050
55
50005000
```

1.2.4. Ex4: getTime()

1. 因為每次要看時間都需要重新施展一長串的咒語，於是菲絲恩將這些咒語寫到了魔法書內。
2. 以後要看時間時，只需要拿著魔法書並呼喊名稱就可以自動施展這些魔法了。
3. Source

Listing 1.2.4.2 /src/Function/p08e2Function.py

```
'''
@since: 20150918
@author: cph
@note:
    Question:
        因為每次要看時間都需要重新施展一長串的咒語，
        於是菲絲恩將這些咒語寫到了魔法書內。以後要看時間時，
        只需要拿著魔法書並呼喊名稱就可以自動施展這些魔法了。
'''
import time                     # 匯入time 模組

def getTime(sTimeFormat):    # 定義一個名為getTime 的函數
    tmTimeStamp = time.time()
    # 將資料回傳
    return(time.strftime(sTimeFormat, time.localtime(tmTimeStamp)))

print("完整顯示: " + getTime('%Y-%m-%d %H:%M:%S'))
print("只顯示日期: " + getTime('%Y-%m-%d'))
print("只顯示時間: " + getTime('%H:%M:%S'))
```

4. Output

```
完整顯示: 2014-06-21 17:35:58
只顯示日期: 2014-06-21
只顯示時間: 17:35:58
```

1.2.5. Ex5a: 1-Page Cal.: Chi

1. Question: Please output an 1-page calendar for 2021. Such as shown as below.

Note that we focus on the calendar content, not headers.

2. Code+Output

Cal2021.py | Output

a. Note that the format might be distortion and hard to adjust it.

```
            2021 Single Page Calendar

                  2    6    9    4    1    5    8
                  3        12    7   10
                 11
  1  8 15 22 29 周一 周二 周三 周四 周五 周六 周日
  2  9 16 23 30 周二 周三 周四 周五 周六 周日 周一
```

```
 3 10 17 24 31 周三 周四 周五 周六 周日 周一 周二
 4 11 18 25    周四 周五 周六 周日 周一 周二 周三
 5 12 19 26    周五 周六 周日 周一 周二 周三 周四
 6 13 20 27    周六 周日 周一 周二 周三 周四 周五
 7 14 21 28    周日 周一 周二 周三 周四 周五 周六
```

1.2.6. Ex5b: 1-Page Cal.: Eng

1. Question: Please output an 1-page calendar for 2021. Note that we change Chinese characters into English ones.
2. Code+Output

Cal2021.py | Output

a. Note that the format adjustment becomes simpler.

```
      2021 Single Page Calendar

                 2  6  9  4  1  5  8
                 3    12  7 10
                11
 1  8 15 22 29  W1 W2 W3 W4 W5 W6 W7
 2  9 16 23 30  W2 W3 W4 W5 W6 W7 W1
 3 10 17 24 31  W3 W4 W5 W6 W7 W1 W2
 4 11 18 25     W4 W5 W6 W7 W1 W2 W3
 5 12 19 26     W5 W6 W7 W1 W2 W3 W4
 6 13 20 27     W6 W7 W1 W2 W3 W4 W5
 7 14 21 28     W7 W1 W2 W3 W4 W5 W6
```

1.2.7. Ex5c: 1-Page Cal.: Function

1. Question: Please output an 1-page calendar for 2021 with functions. Note that we switch the axis from left sdie to right side. Also, we set function with optional arguments for showing frame or not.
2. Code+Output

Cal2021.py | Output: Without Frame | Output: With Frame (Harder)

a. 2021 calendar with frame

```
+--------------------------------------+
|      2021 Single Page Calendar       |
+--------------------------------------+
|  2  6  9  4  1  5  8 | Month         |
|  3    12  7 10       |     Weekday   |
| 11                   |           Day |
+--------------------------------------+
| W1 W2 W3 W4 W5 W6 W7 | 1  8 15 22 29 |
| W2 W3 W4 W5 W6 W7 W1 | 2  9 16 23 30 |
| W3 W4 W5 W6 W7 W1 W2 | 3 10 17 24 31 |
| W4 W5 W6 W7 W1 W2 W3 | 4 11 18 25    |
| W5 W6 W7 W1 W2 W3 W4 | 5 12 19 26    |
| W6 W7 W1 W2 W3 W4 W5 | 6 13 20 27    |
| W7 W1 W2 W3 W4 W5 W6 | 7 14 21 28    |
+--------------------------------------+
```

1.2.8. Ex5d: 1-Page Perpetual Cal.

1. Question: Please output an 1-page perpetual calendar. That is, we can input random year and show an 1-page calendar of specific year.
2. Code+Output

PerpetualCal.py | Output

```
Please input year, such as 2021: 1964
+-------------------------------------+
|1964 1-Page Calendar, by CPH, 20210123|
+-------------------------------------+
|  6  9  1 10  5  2  3 | Month         |
|    12  4        8 11 |     Weekday   |
|        7             |           Day |
+-------------------------------------+
| W1 W2 W3 W4 W5 W6 W7 | 1  8 15 22 29 |
| W2 W3 W4 W5 W6 W7 W1 | 2  9 16 23 30 |
| W3 W4 W5 W6 W7 W1 W2 | 3 10 17 24 31 |
| W4 W5 W6 W7 W1 W2 W3 | 4 11 18 25    |
| W5 W6 W7 W1 W2 W3 W4 | 5 12 19 26    |
| W6 W7 W1 W2 W3 W4 W5 | 6 13 20 27    |
| W7 W1 W2 W3 W4 W5 W6 | 7 14 21 28    |
+-------------------------------------+

Please input year, such as 2021: 2050
+-------------------------------------+
|2050 1-Page Calendar, by CPH, 20210123|
+-------------------------------------+
|  8  2  6  9  4  1  5 | Month         |
|     3    12  7 10    |     Weekday   |
|    11                |           Day |
+-------------------------------------+
| W1 W2 W3 W4 W5 W6 W7 | 1  8 15 22 29 |
| W2 W3 W4 W5 W6 W7 W1 | 2  9 16 23 30 |
| W3 W4 W5 W6 W7 W1 W2 | 3 10 17 24 31 |
| W4 W5 W6 W7 W1 W2 W3 | 4 11 18 25    |
| W5 W6 W7 W1 W2 W3 W4 | 5 12 19 26    |
| W6 W7 W1 W2 W3 W4 W5 | 6 13 20 27    |
| W7 W1 W2 W3 W4 W5 W6 | 7 14 21 28    |
+-------------------------------------+

Please input year, such as 2021: 2599
+-------------------------------------+
|2599 1-Page Calendar, by CPH, 20210123|
+-------------------------------------+
|  4  1  5  8  2  6  9 | Month         |
|  7 10        3    12 |     Weekday   |
|             11       |           Day |
+-------------------------------------+
| W1 W2 W3 W4 W5 W6 W7 | 1  8 15 22 29 |
| W2 W3 W4 W5 W6 W7 W1 | 2  9 16 23 30 |
| W3 W4 W5 W6 W7 W1 W2 | 3 10 17 24 31 |
| W4 W5 W6 W7 W1 W2 W3 | 4 11 18 25    |
| W5 W6 W7 W1 W2 W3 W4 | 5 12 19 26    |
| W6 W7 W1 W2 W3 W4 W5 | 6 13 20 27    |
| W7 W1 W2 W3 W4 W5 W6 | 7 14 21 28    |
+-------------------------------------+
```

1.3. Examples: yield

1.3.1. Ex1: Generator Function

1. Yield can be used to create a generator function that produces a sequence of values.
2. Code+Output

> Code | Output
>
> **Listing 1.3.1.1 /src/Function/yield/Ex1.py**

```
'''
author: CPH
since:  20230731
'''
def count(iMax):
    iNum = 1
    while iNum <= iMax:
        yield iNum
        iNum += 1

if __name__ == '__main__':
    for n in count(5):
        print(n)
```

1.3.2. Ex2: Infinite Sequence

1. Yield can produce an infinite sequence by looping forever:
2. Code+Output

> Code | Output
>
> **Listing 1.3.2.1 /src/Function/yield/Ex2.py**

```
'''
author: CPH
since:  20230731
'''
def infinite():
    iNum = 0
    while (1):
        yield iNum
        iNum += 1

if __name__ == '__main__':
    for i in infinite():
        print(i, end=' ')
        if i >= 10:
            break
```

1.3.3. Ex3: Coroutine

1. Yield allows bidirectional communication in coroutines:

2. Code+Output

Code | Output

Listing 1.3.3.1 /src/Function/yield/Ex3.py

```
'''
author: CPH
since:  20230731
'''
def echo():
    while 1:
        sRet = yield
        print(sRet)

if __name__ == '__main__':
    e = echo()
    next(e)             # prime the coroutine
    e.send("Hello,")
    e.send("Python...")
```

3. Summary,

a. yield is useful for generators, iterators, and coroutines in Python.

b. It pauses execution while preserving state between calls.

Note

1. Start: 20170719
2. System Environment

Listing 1.3.3.2 requirements.txt

```
sphinx==7.1.2                           # Sphinx
graphviz>=0.20.1                        # Graphviz
sphinxbootstrap4theme>=0.6.0            # Theme: Bootstrap
sphinx-material>=0.0.35                 # Theme: Material
sphinxcontrib-plantuml>=0.25            # PlantUML
sphinxcontrib.bibtex>=2.5.0             # Bibliography
sphinx-autorun>=1.1.1                   # ExecCode: pycon
sphinx-execute-code-python3>=0.3        # ExecCode
btd.sphinx.inheritance-diagram>=2.3.1   # Diagram
sphinx-copybutton>=0.5.1                # Copy button
sphinx_code_tabs>=0.5.3                 # Tabs
sphinx-immaterial>=0.11.3               # Tabs

#-----------------------------------------------------
#-- Library Upgrade Error by Library Itself
#   >> It needs to fix by library owner
#   >> After fixed, we need to try it later
#-----------------------------------------------------
pydantic==1.10.10                       # 2.0: sphinx compiler error, 20230701

#-----------------------------------------------------
#-- Minor Extension
#-----------------------------------------------------
sphinxcontrib.httpdomain>=1.8.1         # HTTP API

#sphinxcontrib-blockdiag>=3.0.0         # Diagram: block
#sphinxcontrib-actdiag>=3.0.0           # Diagram: activity
#sphinxcontrib-nwdiag>=2.0.0            # Diagram: network
#sphinxcontrib-seqdiag>=3.0.0           # Diagram: sequence

#-----------------------------------------------------
#-- Still Wait For Upgrading Version
#-----------------------------------------------------

#-----------------------------------------------------
#-- Still Under Testing
#-----------------------------------------------------
#numpy>=1.24.2                          # Figure: numpy

#-----------------------------------------------------
#-- NOT Workable
#-----------------------------------------------------
#sphinxcontrib.jsdemo==0.1.4    # ExecCode: Need replace add_js_file()
#jupyter-sphinx==0.4.0          # ExecCode: Need gcc compiler
#sphinxcontrib.slide==1.0.0     # Slide: Slideshare
#hieroglyph==2.1.0              # Slide: make slides
#matplotlib>=3.7.1              # Plot: Need Python >= v3.8
#manim==0.17.2                  # Diagram: scipy, numpy need gcc
#sphinx_diagrams==0.4.0         # Diagram: Need GKE access
#sphinx-tabs>=3.4.1                     # Tabs: Conflict w/ sphinx-material
```