類別設計與變數的有效範圍

設計攝氏溫度類別，具備轉換華氏溫度的功能

```
class Celsius:
    def __init__(self,temperature = 0.0):
        self.set_temperature(temperature)
    def to_fahrenheit(self):
        return (self.get_temperature()*1.8) + 32
    def get_temperature(self):
        return self.__temperature

    def set_temperature(self,value):
        self.__temperature = value

human = Celsius(36.0)
print("C:",human.get_temperature())
print("F:",human.to_fahrenheit())
```

使用python內建初始化方法

在初始化方法中，輸入參數為temperature，使用set_temperature方法，設定雙底線__temperature的內容

```
C: 36.0
F: 96.8
```

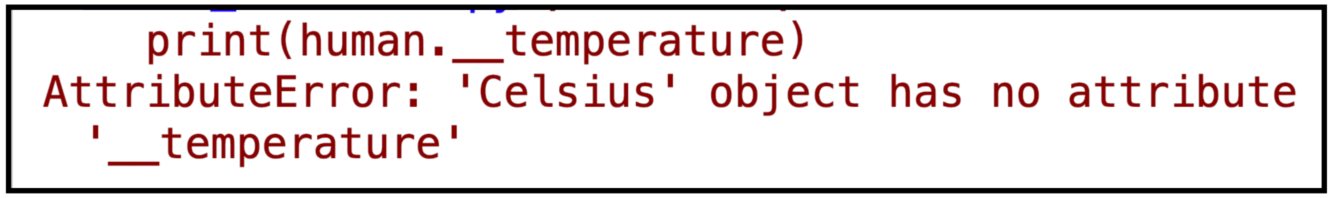
```python
class Celsius:
    def __init__(self,temperature = 0.0):
        self.set_temperature(temperature)
    def to_fahrenheit(self):
        return (self.get_temperature()*1.8) + 32
    def get_temperature(self):
        return self.__temperature

    def set_temperature(self,value):
        self.__temperature = value

human = Celsius(36.0)
print("C:",human.get_temperature())
print("F:",human.to_fahrenheit())
print(human.__temperature)
```

在類別內，使用雙底線__temperature變數儲存攝氏溫度，只限於類別內的函數使用

使用輸入參數設定private變數，只能透過set_temperature方法設定私有變數


```
    print(human.__temperature)
AttributeError: 'Celsius' object has no attribute
  '__temperature'
```

```
class Celsius:
    def __init__(self,temperature = 0.0):
        self.set_temperature(temperature)
    def to_fahrenheit(self):
        return (self.get_temperature()*1.8) + 32
    def get_temperature(self):
        return self.__temperature

    def set_temperature(self,value):
        self.__temperature = value

human = Celsius(36.0)
print("C:",human.get_temperature())
print("F:",human.to_fahrenheit())
```

轉換為華氏溫度時，先使用get_temperature方法，取得攝氏溫度，再回傳轉換後的華氏溫度數值

回傳private變數

透過to_fahrenheit()方法取得華氏溫度

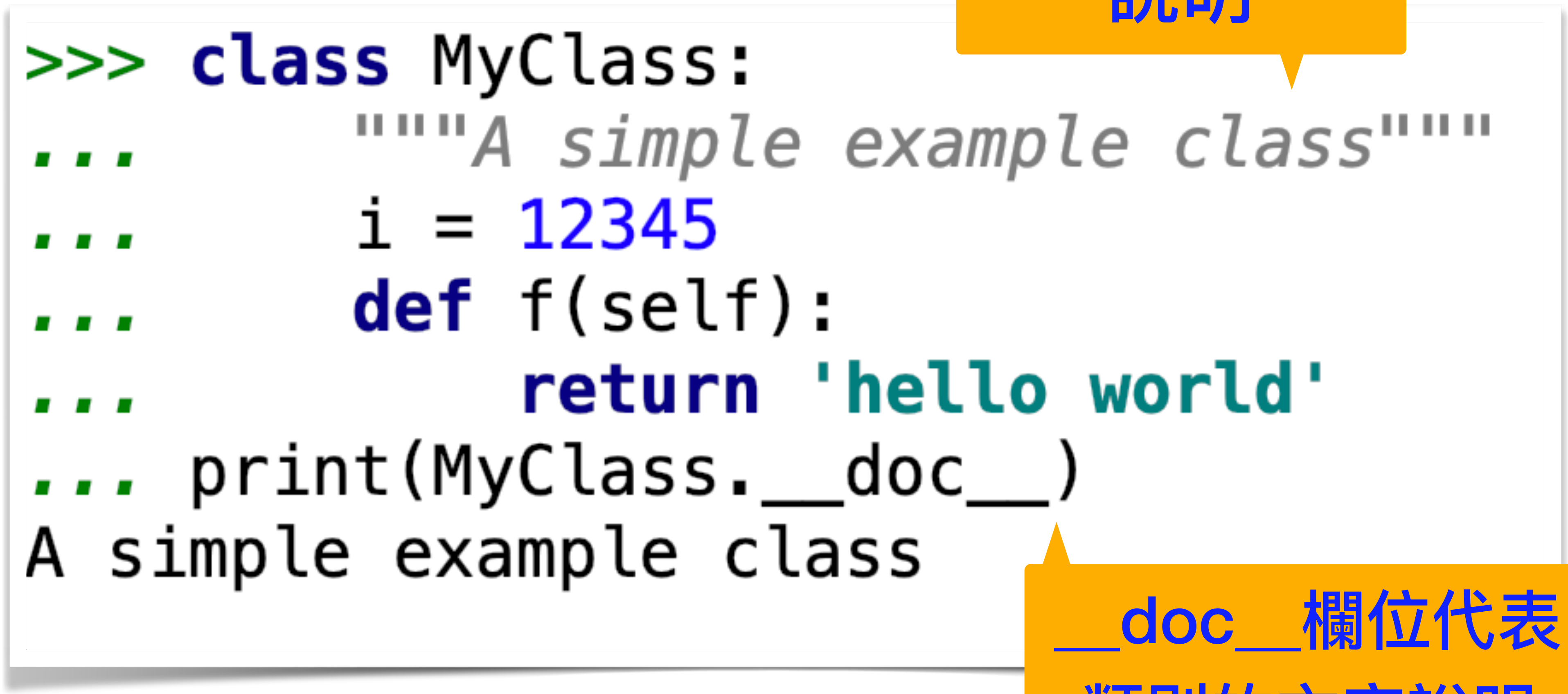
宣告變數human為Celsius類別，並傳入初始化溫度

透過get_temperature()方法取
得攝氏溫度

```
C: 36.0
F: 96.8
```

在類別內的雙底線__temperature變數，為private變數，只限於類別內的方法存取

保留字雙底線__doc__的用法



```
>>> class MyClass:
...     """A simple example class"""
...     i = 12345
...     def f(self):
...         return 'hello world'
... print(MyClass.__doc__)
A simple example class
```

```
>>> class MyClass:
...       """A simple example class"""
...       i = 12345
...       def f(self):
...           return 'hello world'
... x = MyClass()
... print(x.f())
hello world
```

宣告x為型態MyClass的變數，並使用方法f()

建立複數類別

複數類別

```
>>> class Complex:
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
... x = Complex(3.0, -4.5)
... print(x.r, x.i)
3.0 -4.5
```

設定複數的實部與虛部

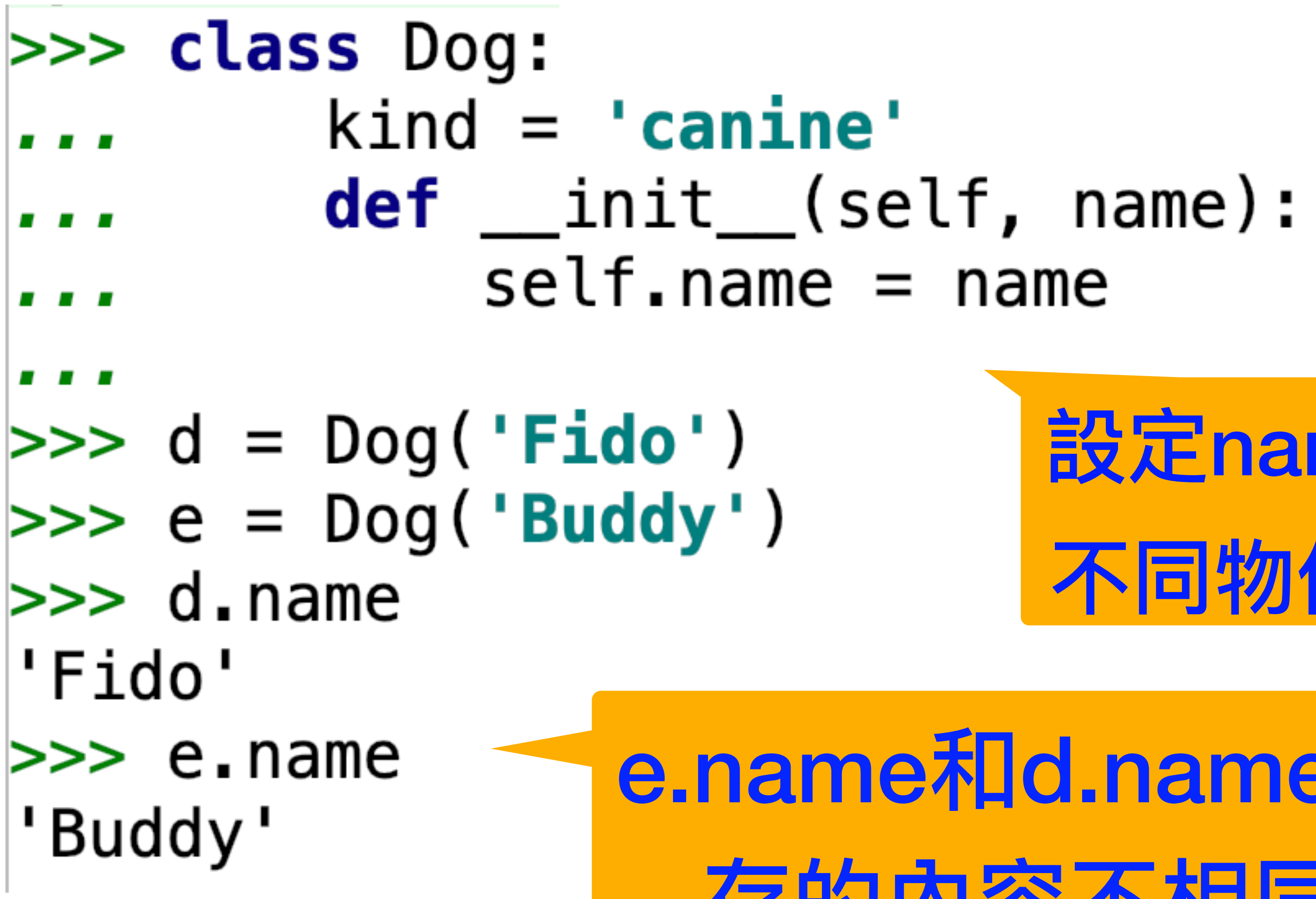
建立狗類別

```
>>> class Dog:
...     kind = 'canine'
...     def __init__(self, name):
...         self.name = name
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind
'canine'
>>> e.kind
'canine'
```

將共有的特性欄位kind設定為字串’canine’

將個別特性欄位name設定為傳入的參數字串

e.kind和d.kind儲存的內容相同，都是字串’canine’

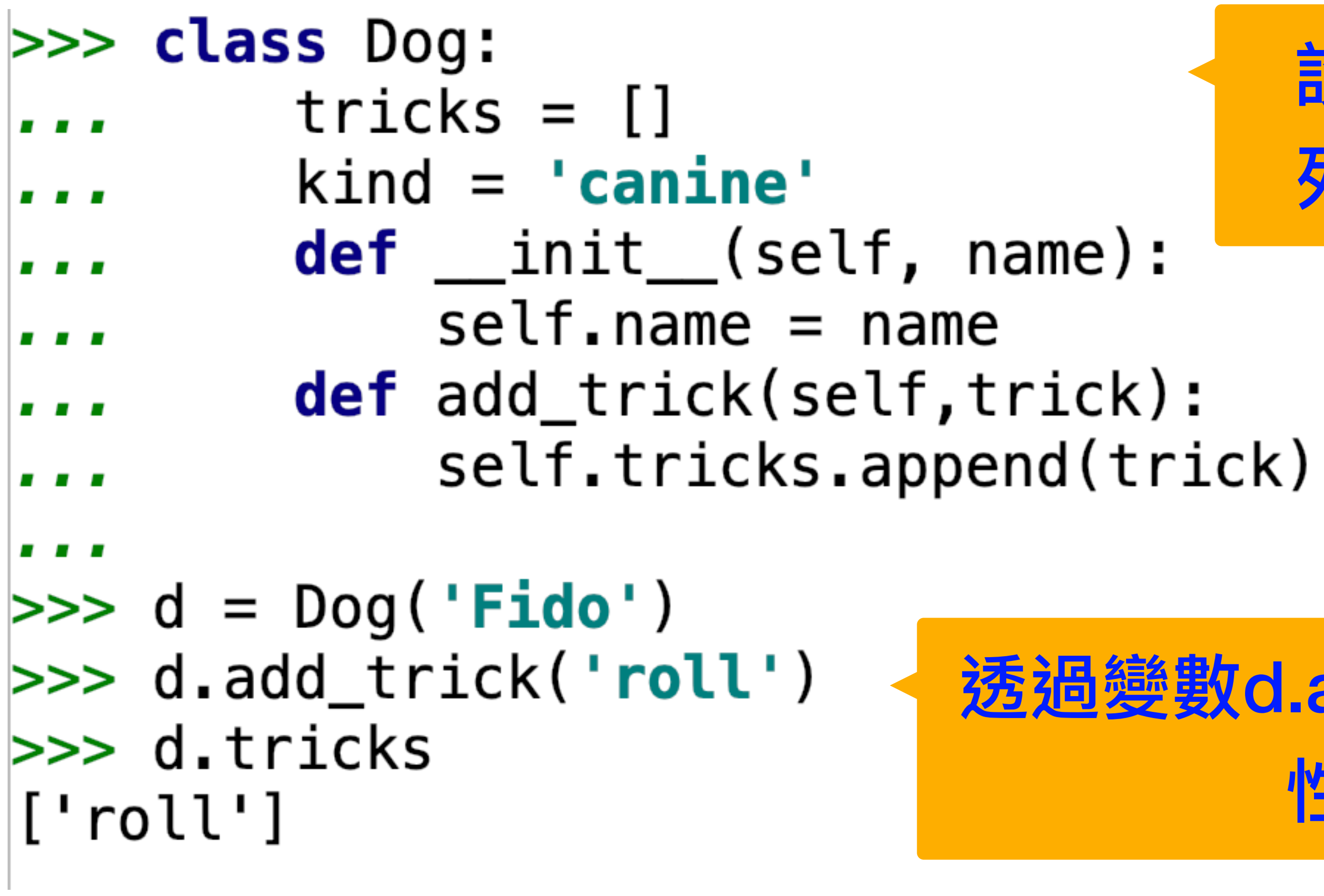


```
>>> class Dog:
...     kind = 'canine'
...     def __init__(self, name):
...         self.name = name
...
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.name
'Fido'
>>> e.name
'Buddy'
```

設定name欄位為name，不同物件個別使用的欄位

e.name和d.name儲存的內容不相同

增加trick特性欄位



```
>>> class Dog:
...     tricks = []
...     kind = 'canine'
...     def __init__(self, name):
...         self.name = name
...     def add_trick(self,trick):
...         self.tricks.append(trick)
...
>>> d = Dog('Fido')
>>> d.add_trick('roll')
>>> d.tricks
['roll']
```

設定trick屬性為空串列，共通使用的欄位

透過變數d.add_trick()改變屬性tricks

```
>>> class Dog:
...     tricks = []
...     kind = 'canine'
...
...     def __init__(self, name):
...         self.name = name
...
...     def add_trick(self, trick):
...         self.tricks.append(trick)
...
...
... d = Dog('Fido')
... d.add_trick('roll')
>>> e = Dog('Buddy')
>>> e.add_trick('play dead')
>>> d.tricks
['roll', 'play dead']
```

透過變數
e.add_trick()改變

變數d的屬性

```
class Dog:

    def __init__(self, name):
        self.name = name
        self.tricks = []    # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']
```

透過變數 e.add_trick()改變屬性tricks

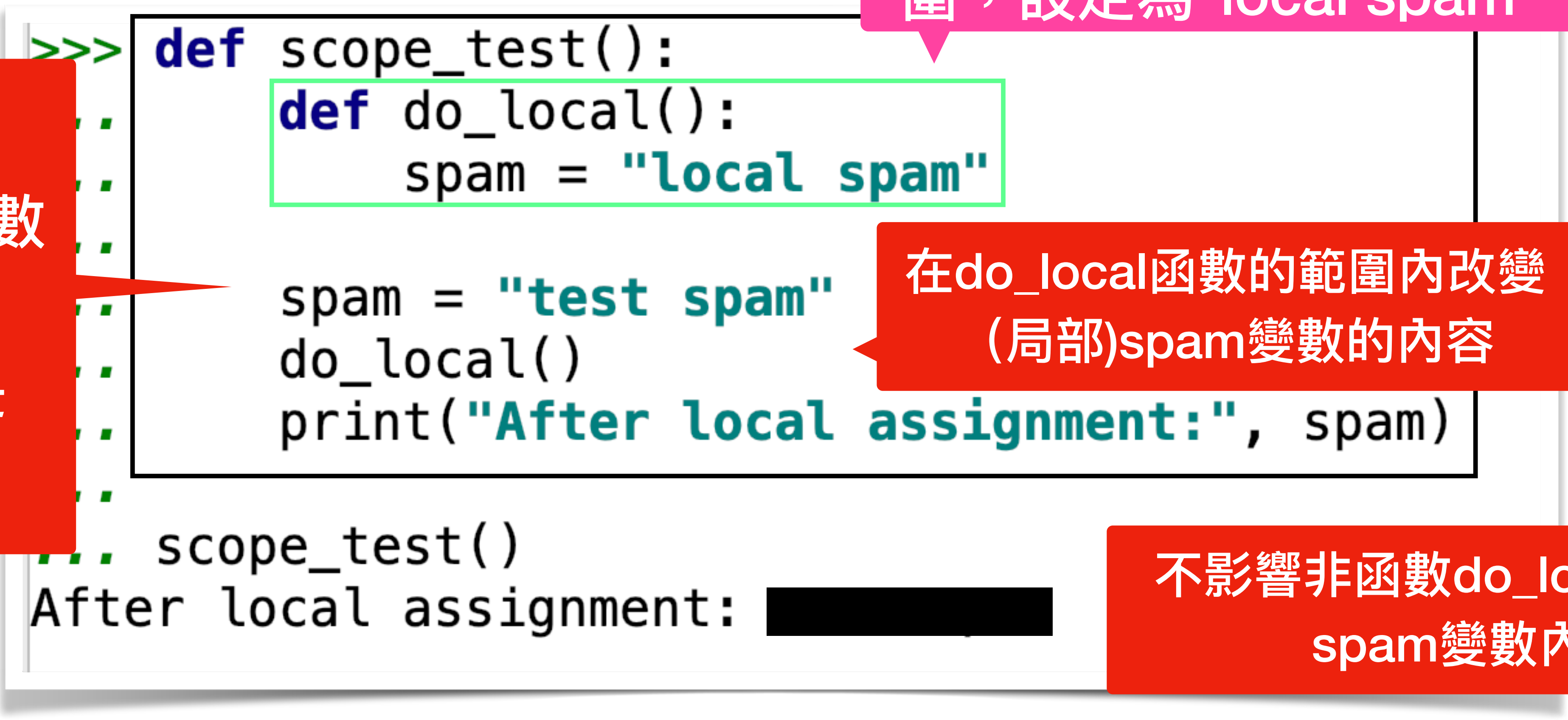
變數d的屬性tricks不會跟著改變

變數範圍：
local（局部）變數
global (全域）變數
non-local 變數

在函數範圍內改變(局部)變數的內容，不影響不屬於函數範圍內的變數內容

spam在函數do_local的範圍，設定為”local spam”

相對於 do_local函數的範圍 spam不是 local變數

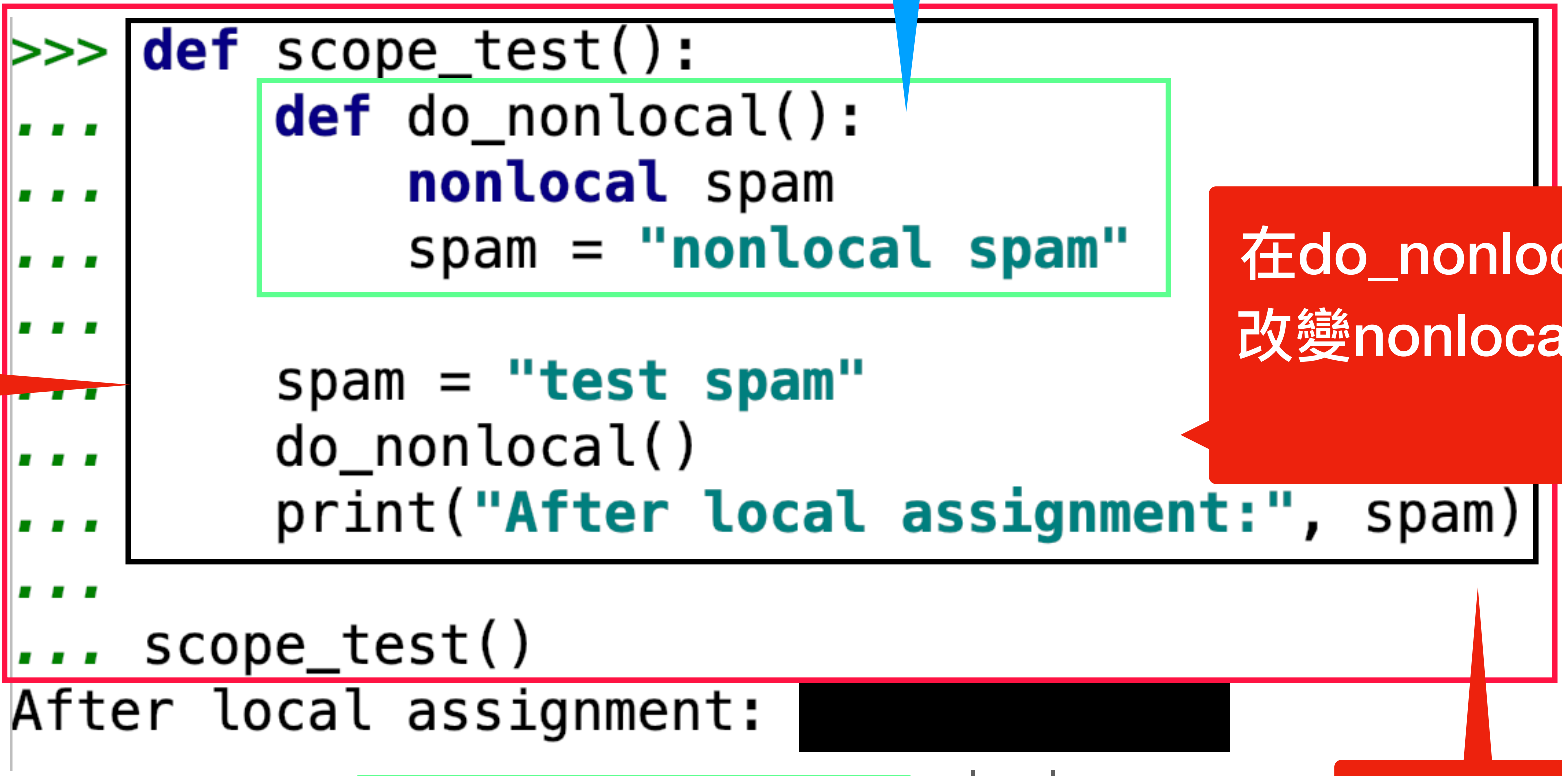


```
>>> def scope_test():
...     def do_local():
...         spam = "local spam"
...
...     spam = "test spam"
...     do_local()
...     print("After local assignment:", spam)
...
... scope_test()
After local assignment: [redacted]
```

在do_local函數的範圍內改變(局部)spam變數的內容

不影響非函數do_local範圍的spam變數內容

在do_nonlocal函數範圍中，宣告spam為nonlocal變數，指定為字串” nonlocal spam”



```
>>> def scope_test():
...     def do_nonlocal():
...         nonlocal spam
...         spam = "nonlocal spam"
...
...     spam = "test spam"
...     do_nonlocal()
...     print("After local assignment:", spam)
...
... scope_test()
After local assignment: [redacted]
```

相對於do_nonlocal函數的範圍spam不是local變數，但屬於nonlocal變數

在do_nonlocal函數的範圍內改變nonlocal 變數spam的內容

影響nonlocal變數spam的內容

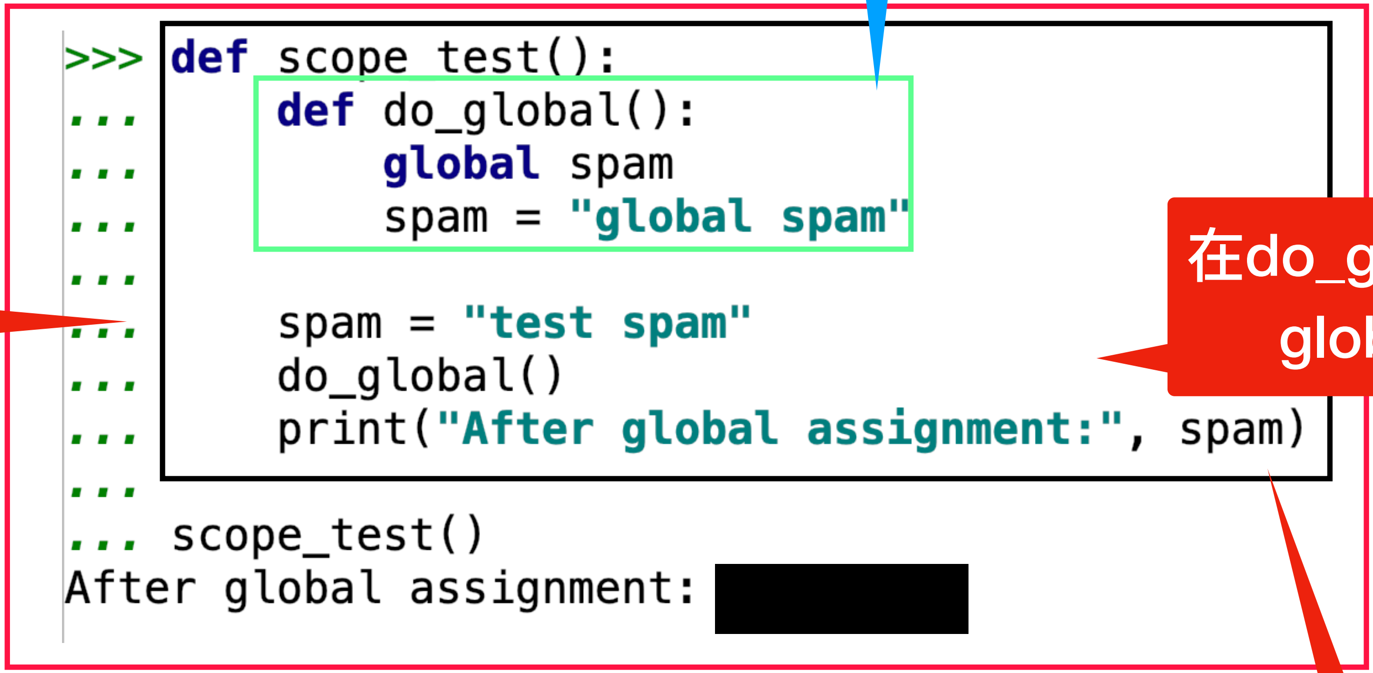
local 🟩

nonlocal ⬜

global 🟥

在do_global函數範圍中宣告，宣告spam為global變數，指定為字串”global spam”

相對於do_global函數的範圍spam不是local變數，屬於nonlocal變數，也不是global變數



```
>>> def scope_test():
...     def do_global():
...         global spam
...         spam = "global spam"
...
...     spam = "test spam"
...     do_global()
...     print("After global assignment:", spam)
...
... scope_test()
After global assignment: [redacted]
```

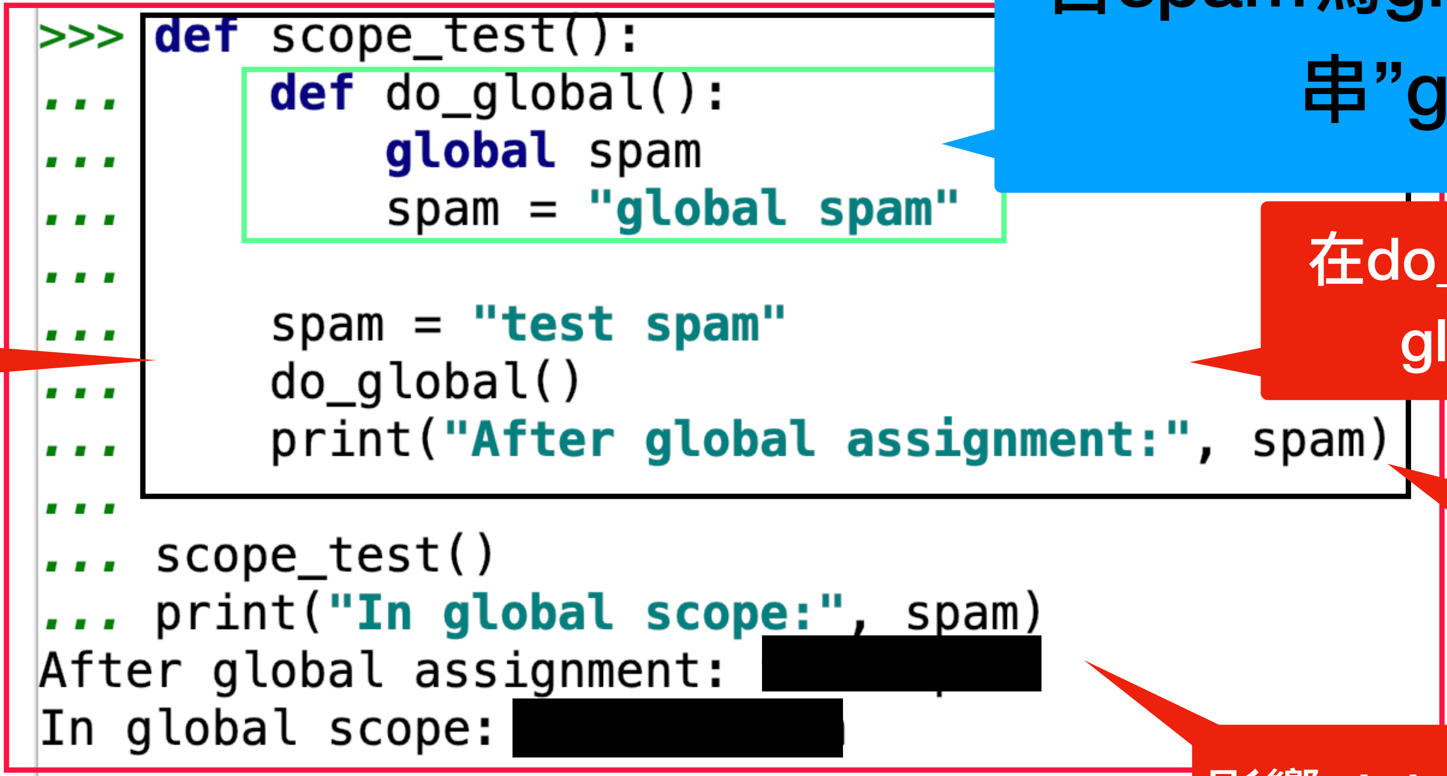
在do_global函數的範圍內改變global變數spam的內容

『不』影響nonlocal變數spam的內容

🟩	local
⬛	nonlocal
🟥	global

相對於do_global函數的範圍spam不是local變數，屬於nonlocal變數，也不是global變數

在do_global函數範圍中宣告，宣告spam為global變數，指定為字串"global spam"



```
>>> def scope_test():
...     def do_global():
...         global spam
...         spam = "global spam"
...
...     spam = "test spam"
...     do_global()
...     print("After global assignment:", spam)
...
... scope_test()
... print("In global scope:", spam)
After global assignment: [redacted]
In global scope: [redacted]
```

在do_global函數的範圍內改變global變數spam的內容

『不』影響nonlocal變數spam的內容

影響global變數spam的內容

🟩 local

⬛ nonlocal

🟥 global