分解，抽象，功能

（下载幻灯片和 .py 文件 Ă Ě 跟随！）

6.0001 第 4 讲

上次

- while循环与for循环

应该知道如何写这两种

应该知道何时使用它们

猜测和检查和近似方法

二分法加速程序

今天

结构化程序和隐藏细节

我们如何编写代码？

到目前为止……

- 涵盖语言机制
- 知道如何为每个计算编写不同的文件
- 每个文件都是一段代码
- 每个代码都是一个指令序列

这种方法的问题

- 容易解决小规模问题
- 大问题的混乱
- 难以追踪细节
- 你怎么知道正确的信息被提供给代码的正确部分

良好的编程

更多的代码不一定是好事

通过功能的数量来衡量优秀的程序员

介绍功能

实现分解和抽象的机制

示例 – 投影仪

投影仪是一个黑匣子

不知道它是如何工作的

了解接口:输入/输出

将任何可以与该输入通信的电子设备连接到它

黑匣子以某种方式将图像从输入源转换为墙壁,放大它

抽象概念:不需要知道投影仪是如何工作的就可以使用它

示例 – 投影仪

将奥运会的大图像投影分解为单独的投影仪的单独任务

每个投影仪都接受输入并产生单独的输出

所有投影仪协同工作以产生更大的图像

DECOMPOSITION IDEA:不同的设备协同工作以实现最终目标

应用这些概念

编程！

创建结构

分解

在投影仪示例中,单独的设备

在编程中,将代码分成模块 ·是自包含的

- 用于分解代码
- 旨在可重复使用
- 保持代码井井有条
- 保持代码连贯性

本讲,用函数实现分解

在几周内,用类实现分解

抑制细节

抽象

在投影仪示例中,如何使用它的说明就足够了,不需要知道如何构建一个

在编程中,把一段代码想象成一个黑盒子

- 看不到详细信息
- 不需要看细节
- 不想看细节
- 隐藏繁琐的编码细节

使用函数规范或文档字符串实现抽象

功能

编写可重用的代码片段/块,称为函数

在程序中“调用”或“调用”函数之前,它们不会在程序中运行

功能特点:

- 有名字
- 有参数（0 个或更多）
- 有一个文档字符串（可选但推荐）
- 有一个正文
- 返回一些东西

如何写作和调用/调用函数

keyword

name

parameters or arguments

specification, docstring

```
def is_even(i):
    """
    输入:i,一个正整数

    如果 i 是偶数则返回 True,否则返回 False
    """
    print(    里面 is_even    )
    返回 i%2 == 0
```

body

later in the code, you call the function using its name and values for parameters

```
is_even(3)
```

在函数体中

定义是_偶数（我）：

输入:i,一个正整数

如果 i 是偶数则返回 True,否则返回 False

```
print(    里面 is_even    )
返回 i%2 == 0
```

run some commands

keyword

expression to evaluate and return

可变范围

调用函数时,形参与实参的值绑定

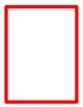
输入函数时创建的新范围/框架/环境

范围是名称到对象的映射

```
定义 f( x ):
    x = x + 1
    print(    in f(x): x =    , x)
    返回 x

x = 3
z = f( x )
```

formal parameter

Function definition

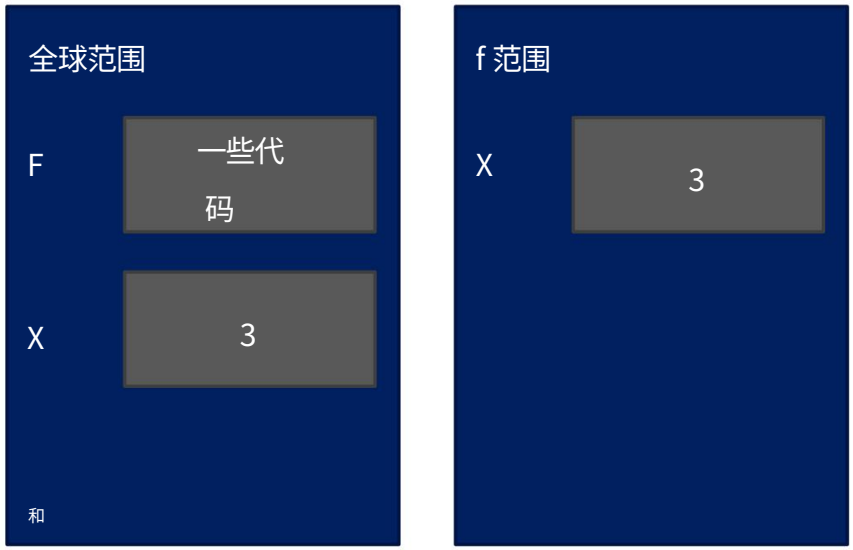


actual parameter

Main program code
* initializes a variable x
* makes a function call f(x)
* assigns return of function to variable z

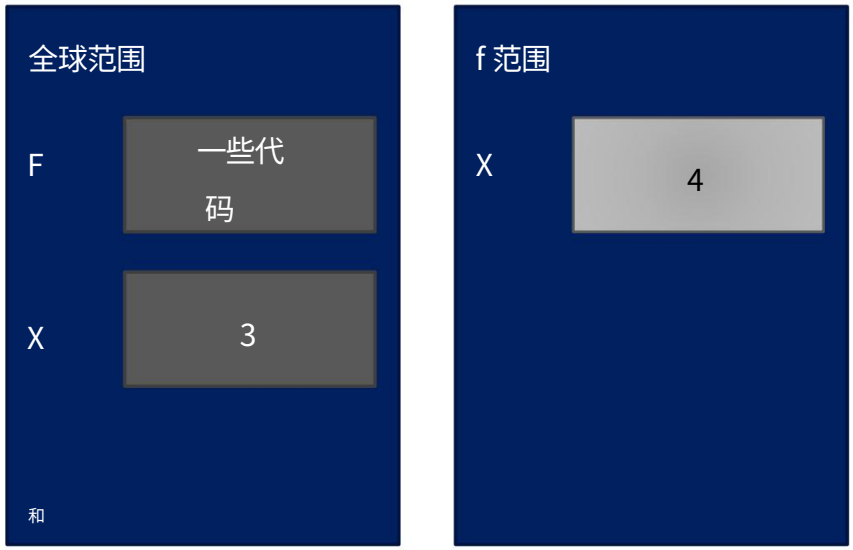
可变范围

```
定义 f( x ):
        x = x + 1
        print(    in f(x): x =    , x)
        返回 x

x = 3
z = f( x )
```



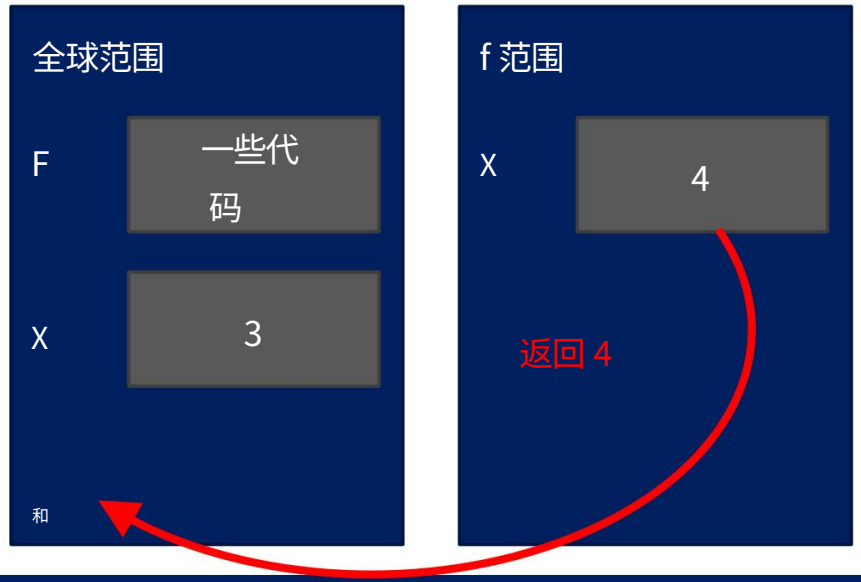
可变范围

```
定义 f( x ):
        x = x + 1
        print(    in f(x): x =    , x)
        返回 x

x = 3
z = f( x )
```



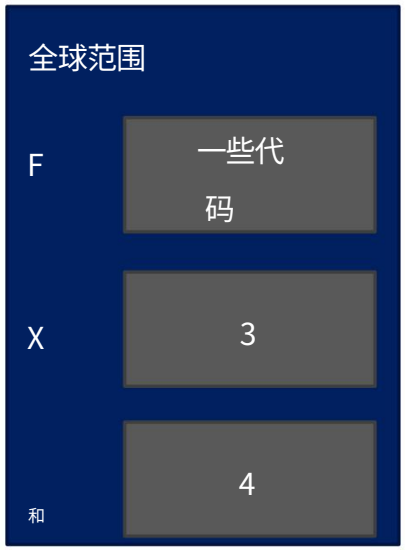
可变范围

```
定义 f( x ):
        x = x + 1
        print(    in f(x): x =    , x)
        返回 x

x = 3
z = f( x )
```



可变范围

```
定义 f( x ):
        x = x + 1
        print(    in f(x): x =    , x)
        返回 x

x = 3
z = f( x )
```



一个警告如果没有退货声明

定义是_偶数（我）：

输入:i,一个正整数

不返回任何东西

```
i%2 == 0
```

without a return statement

Python 返回值None,如果没有给出返回值

表示没有值

返回与打印

return 只在函数内部有意义

在一个函数内只执行一个返回

函数内部的代码,但返回语句后未执行

有一个与之关联的值,给函数调用者

打印可以在外面使用功能

可以在一个函数内执行许多打印语句

函数内的代码可以在打印后执行

陈述

有一个与之关联的值,输出到控制台

作为参数的函数

参数可以采用任何类型,甚至函数def func_a():

```
        打印  内部 func_a
def func_b(y):
        打印  内部func_b
        返回 y
def func_c(z):
        打印  内部 func_c
        返回 z()
打印 func_a()
打印 5 + func_b(2)
打印 func_c(func_a)
```

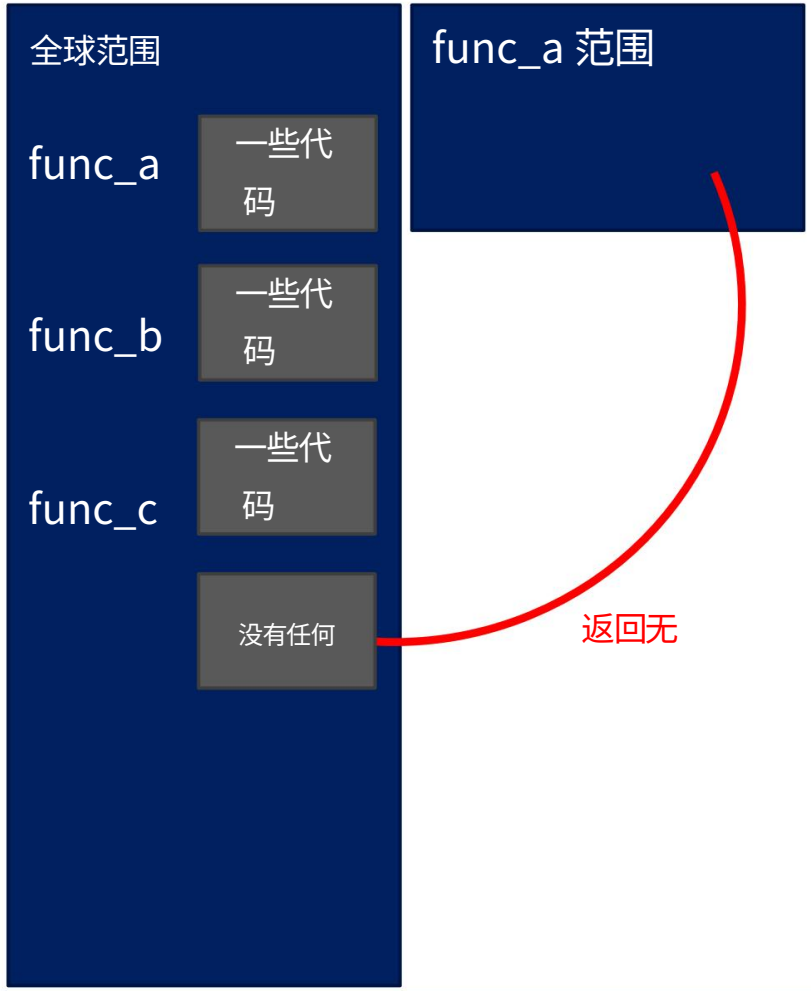
call `func_a`, takes no parameters

call `func_b`, takes one parameter

call `func_c`, takes one parameter, another function

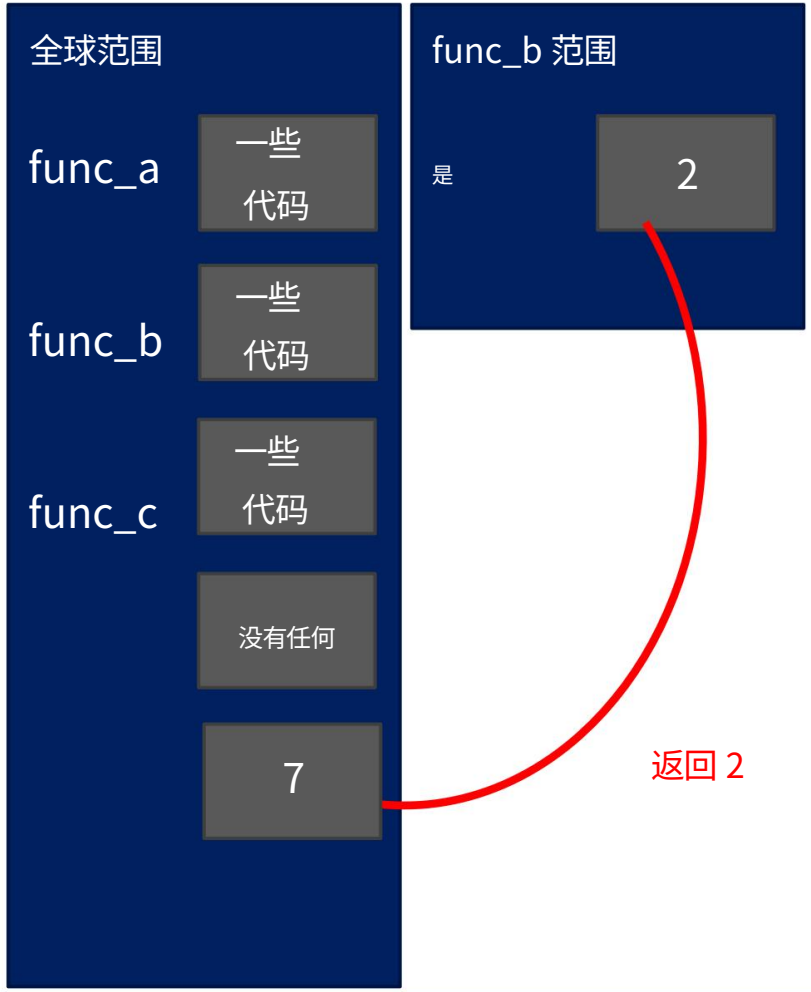
作为参数的函数

```
def func_a():
    打印 内部 func_a
def func_b(y):
    打印 内部func_b
    返回 y
def func_c(z):
    打印 内部 func_c
    返回 z()
打印 func_a()
打印 5 + func_b(2)
打印 func_c(func_a)
```



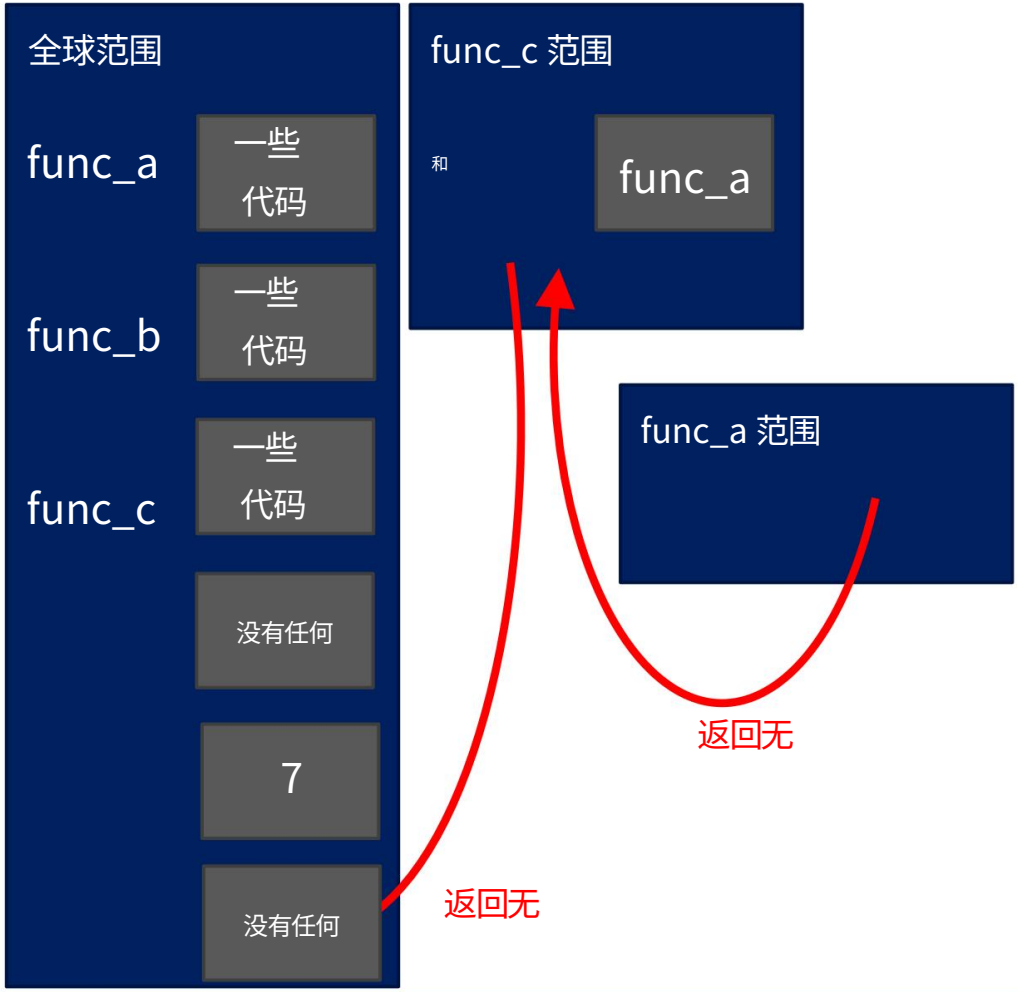
作为参数的函数

```
def func_a():
    打印 内部 func_a
def func_b(y):
    打印 内部func_b
    返回 y
def func_c(z):
    打印 内部 func_c
    返回 z()
打印 func_a()
打印 5 + func_b(2)
打印 func_c(func_a)
```



作为参数的函数

```
def func_a():
    打印 内部 func_a
def func_b(y):
    打印 内部func_b
    返回 y
def func_c(z):
    打印 内部 func_c
    返回 z()
打印 func_a()
打印 5 + func_b(2)
打印 func_c(func_a)
```



范围示例

在函数内部，可以访问外部定义的变量

在函数内部，不能修改外部定义的变量　可以使用全局变量，但不赞成

```
定义 f(y): x = 1
x += 1
打印（x）
x = 5
f(x) 打
印(x)
```

x is re-defined in scope of f

different x objects

```
def g(y): 打印(x)
打印(x + 1)
x = 5
g(x) 打
印(x)
```

x from outside g

x inside g is picked up from scope that called function g

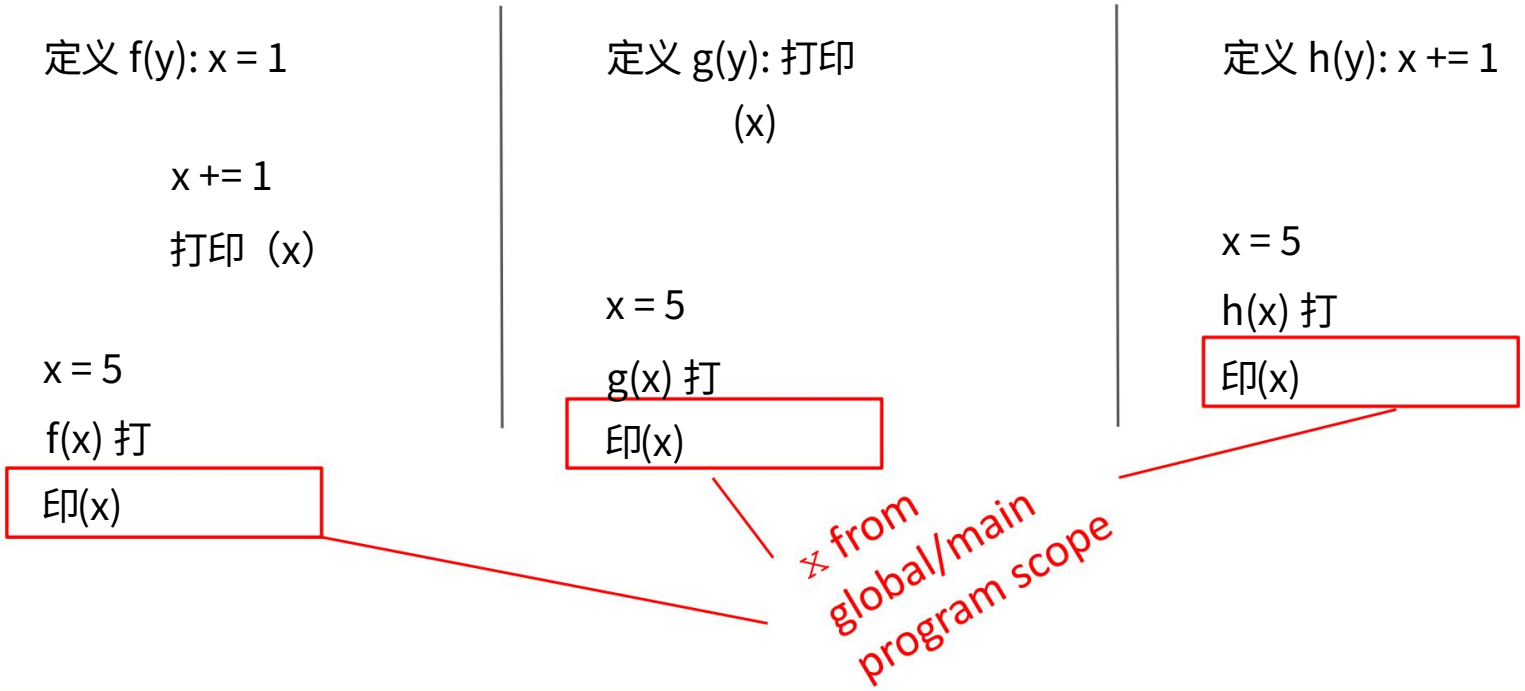
```
定义 h(y): x += 1
x = 5
h(x) 打
印(x)
```

UnboundLocalError: local variable 'x' referenced before assignment

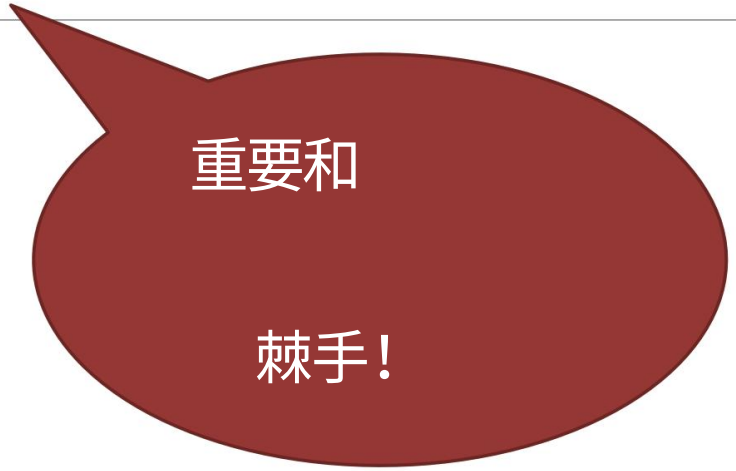
范围示例

在函数内部，可以访问外部定义的变量

在函数内部，不能修改外部定义的变量　可以使用全局变量，但不赞成



更难的范围示例



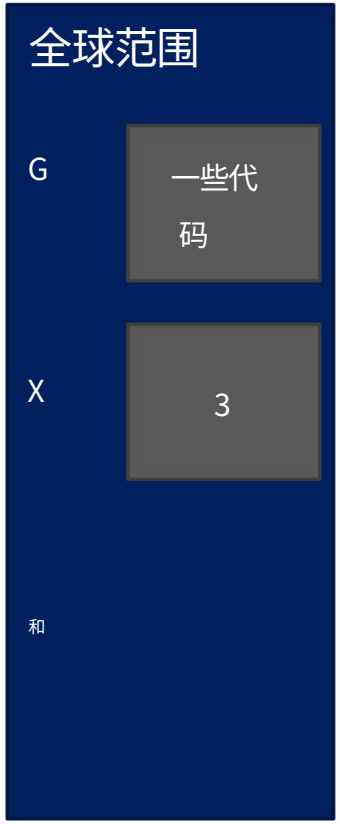
Python Tutor 是您最好的朋友，可以帮助您解决这个问题！

http://www.pythontutor.com/

范围详情

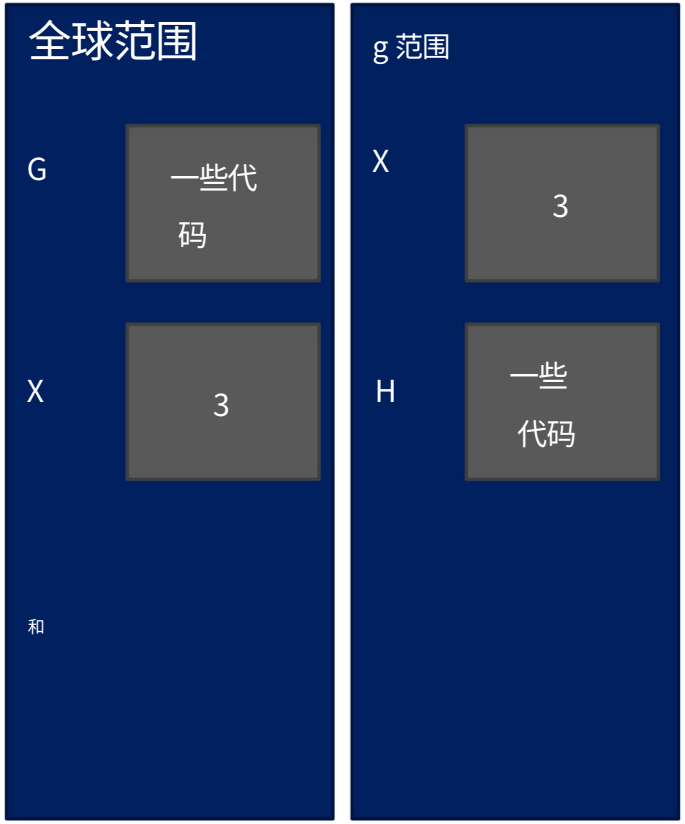
```
定义 g(x):
    定义 h():
        x = abc
    x = x + 1
    打印（ g:x = ,x）
    H（）
    返回 x

x = 3
z = g(x)
```

Some code



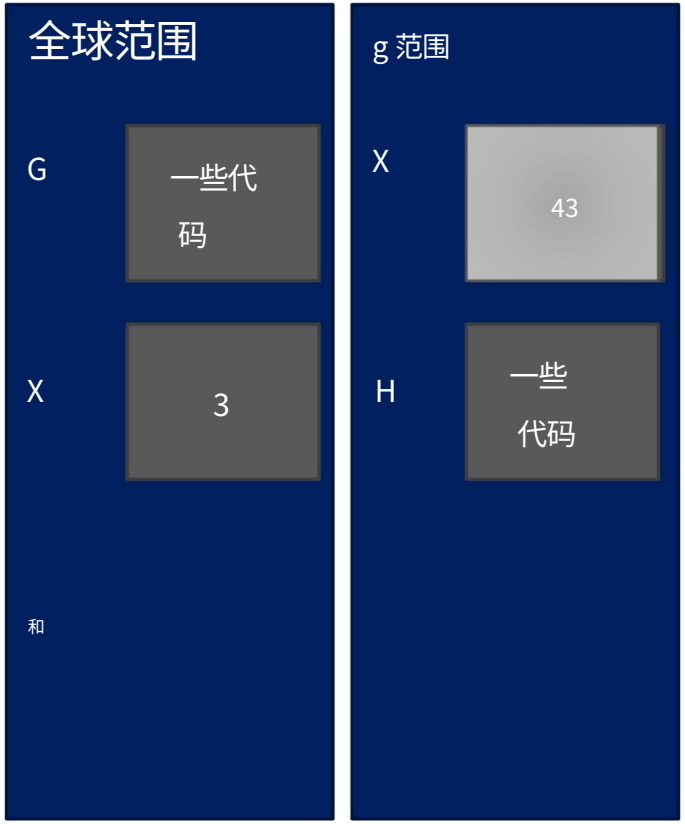
范围详情

```
定义 g(x):
    定义 h():
        x = abc
    x = x + 1
    打印（ g:x = ,x）
    H（）
    返回 x

x = 3
z = g(x)
```



范围详情

```
定义 g(x):
    定义 h():
        x = abc
    x = x + 1
    打印（ g:x = ,x）
    H（）
    返回 x

x = 3
z = g(x)
```



范围详情

定义 g(x):

定义 h():

x = abc

x = x + 1

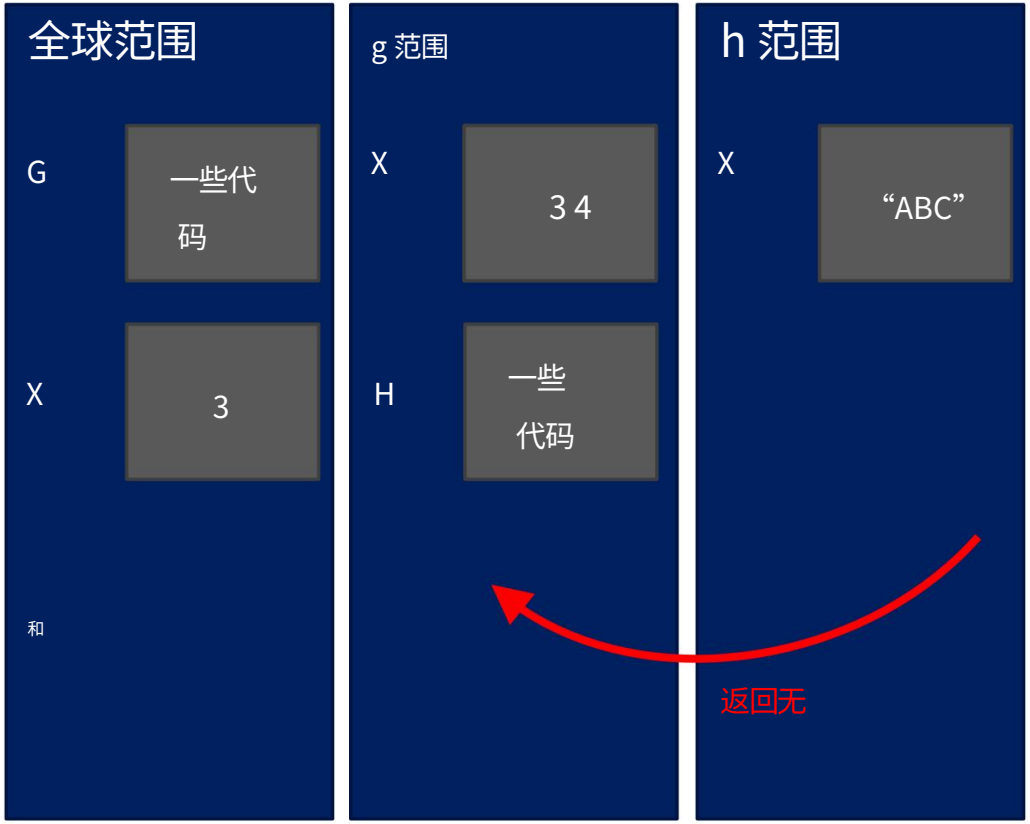
打印（ g:x = ,x）

H（）

返回 x

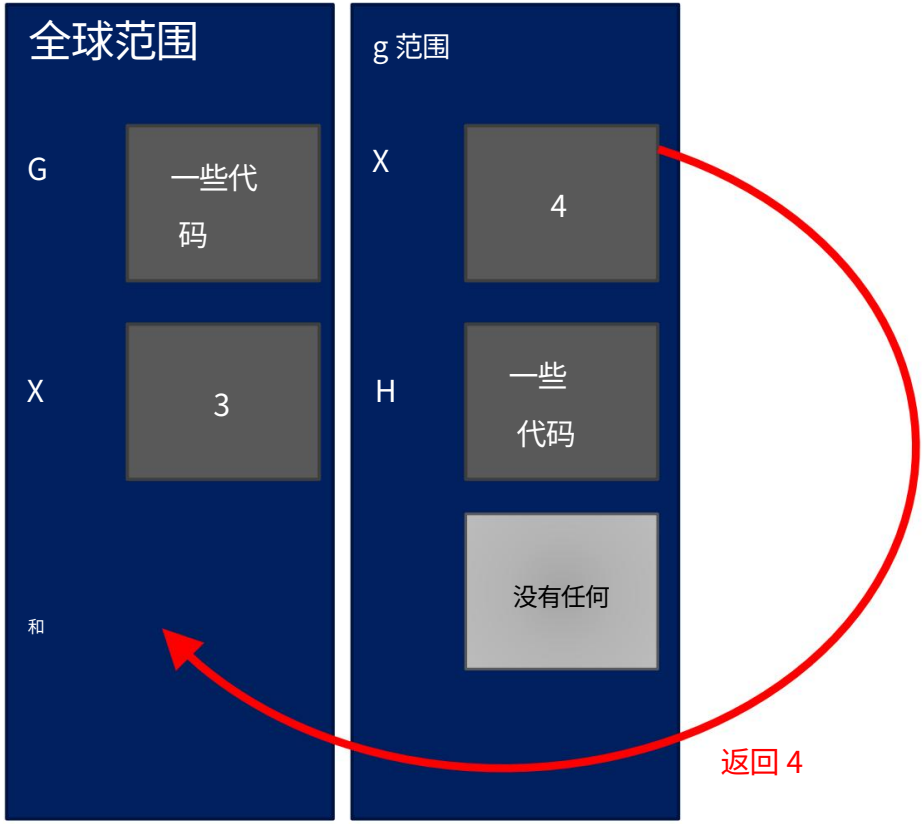
x = 3

z = g(x)



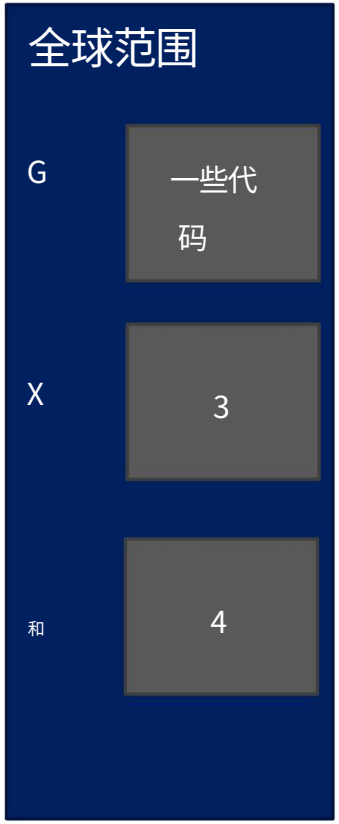
范围详情

```
定义 g(x):
    定义 h():
        x = abc
    x = x + 1
    打印（ g:x = ,x）
    H（）
    返回 x

x = 3
z = g(x)
```



范围详情

```
定义 g(x):
    定义 h():
        x =  abc
    x = x + 1
    打印（ g:x =  ,x）
    H（）
    返回 x

x = 3
z = g(x)
```



分解&
抽象

强强联合

代码可以多次使用,但只需要调试一次!

6.0001 计算机科学和 Python 编程简介

2016 年秋季