CS 61A

控制和环境 2020年秋季

讨论1：2020年9月2日

1 控制

控制结构使用逻辑语句指导程序的流程。例如，条件语句（if-elif-else）允许程序跳过代码部分，而迭代（while）则允许程序重复一个部分。

如果声明

条件性语句让程序根据某些条件执行不同的代码行。让我们回顾一下if-elif-else的语法。

回顾以下几点。

- else和elif子句是可选的，你可以有任何数量的elif条款。
- **条件表达式**是一个表达式，它评估为一个真值（True，一个非零的整数，等等）或一个假值（False，0，None，1111，[]，等等）。
- 只有缩进在第一个if/elif下的**套件**，其**条件表达式**评估为真值，才会被执行。
- 如果没有一个**条件表达式**评估为真值，那么就执行else套件。一个条件语句中只能有一个else子句!

```
如果<条件表达式>。
    <一系列的声明>。
elif <conditional expression>:
    <一系列的声明>。
否则。
    <一系列的声明>。
```

布尔运算符

Python 还包括**布尔运算符** and, or, and not。这些运算符被用来组合和操作布尔值。

- not 返回以下表达式的相反真值（所以 not 总是返回 True 或 False）。
- 并按顺序评估表达式，一旦达到第一个假值就停止评估（短路），然后返回。如果所有值都评估为真值，则返回最后一个值。
- 或在第一个真值处短路，并返回它。如果所有的值都评估为假值，则返回最后一个值。

```
>>不是无真
>> 不是真的，是
假的
>> -1和0和1
0
>> 虚假或9999或1/0 9999
```

问题

1.1 阿方索只有在气温低于60度或下雨的情况下才会穿外套出门。

编写一个函数，输入当前温度和一个布尔值，告诉它是否在下雨，如果阿方索将穿上夹克，则返回 "真"，否则返回 "假"。

首先，尝试用if语句来解决这个问题。

```
def wears_jacket_with_if(temp,
    raining):llllll
    >>> wears_jacket_with_if(90, False)
    False
    >>> wears_jacket_with_if(40, False)
    True
    >>> wears_jacket_with_if(100, True)
    True
    llllll
```

请注意，我们将根据一个条件返回True或False，这个条件的真值也将是True或False。了解了这一点，试着用一行字来写这个函数。

```
def wears_jacket(temp, raining):
```

循环的时候

为了在一个程序中多次重复相同的语句，我们可以使用迭代。在Python中，我们可以用一个**while循环来**做这件事。

只要<conditional clause>评估为真值，<body of statements>将继续被执行。每当语句体执行完毕，条件子句就会被评估。

而<条件性条款>。
<声明的主体>。

问题

1.2 评估以下代码的结果是什么？

```
def square(x):
    print(11here!11)
    return x * x

def so_slow(num):
    x = num
    而x>0。
        x = x + 1
    返回 x / 0

square(so_slow(5))
```

1.3 **教程。**写一个函数，如果一个正整数n是质数，则返回True，否则返回False。

质数*n*是指不能被1和*n*本身以外的任何数字所整除。例如，13是质数，因为它只能被1和13整除，但14不是，因为它能被1、2、7和14整除。

提示：使用%运算符：x % y返回x除以y后的余数。

```
def is_prime(n):
    llllll
    >> is_prime(10)
    False
    >> is_prime(7)
    True
    llllll
```

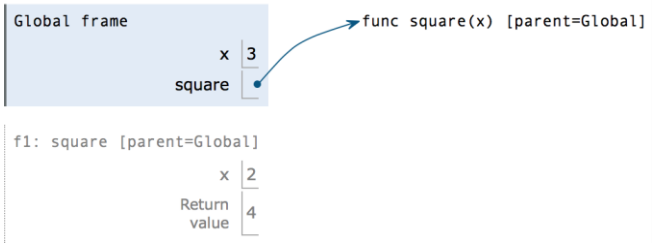
2 环境图

环境图是我们用来跟踪所有已定义的变量和它们所绑定的值的一个模型。我们将在整个课程中使用这一工具来理解涉及几个不同的赋值和函数调用的复杂程序。

```
x = 3

def square(x):
    返回 x ** 2

square(2)
```



请记住，程序只是一组语句，或者说是指令，所以要画.....。
绘制代表这些程序的图表也涉及到遵循一系列的指令！让我们深入了解一下。让我们深入了解一下。

作业声明

赋值语句，如x=3，定义了程序中的变量。要在环境图中执行一个，记录变量名称和数值。

1. 评估=符号右侧的表达式
2. 在当前帧中写入变量名称和表达式的值。

2.1 利用这些规则，为下面的赋值语句画一个简单的图。

```
x = 10 % 4
y = x
x **= 2
```

def 声明

def语句创建函数对象并将其与名称绑定。为了图解def语句，记录函数名称，并将函数对象与名称绑定。写出函数的**父框架**也很重要，这是定义函数的地方。**非常重要的一点是：**def语句的赋值使用指向函数的指针，这可能与原始赋值有不同的行为。

1. 将函数对象画在框架的右侧，表示函数的内在名称、参数和父框架（例如：func square(x) [parent = Global]。[1]
2. 在当前帧中写入函数名称，并从名称到函数对象画一个箭头。

2.2 利用这些规则和赋值语句的规则，为下面的代码画一个图。

```
def double(x):
    返回x * 2

def triple(x):
    返回x * 3
```

帽子=双份 双份=三份

[1]当导入函数时，我们仍然在环境图中创建一个函数对象，与导入函数的名称绑定。然而，导入的函数的父级和参数是未知的，所以只包括函数的名称。例如，如果我们导入了函数add，函数对象将只是add(...)

呼叫表达

调用表达式，例如 square(2)，将函数应用于参数。在执行调用表达式时，我们在图中创建了一个新的框架来跟踪局部变量。

1. 评估运算符，它应该评估为一个函数。
2. 从左到右对操作数进行评估。
3. 绘制一个新的框架，给它贴上以下标签。 [2]
   - 一个唯一的索引（f1，f2，f3，...）。
   - 函数的**内在名称**，也就是函数对象本身的名称。例如，如果函数对象是func square(x) [parent=Global]，其内在名称就是square。
   - 父框架（[父=全球]）。
4. 将形式参数与步骤2中获得的参数值绑定（例如，将x与3绑定）。
5. 在这个新框架中评估函数的主体，直到得到一个返回值。写下框架中的返回值。

如果一个函数没有返回值，它将隐含地返回None。在这种情况下，"返回值 "框应该包含无。

2.3 让我们把它放在一起!为下面的代码画一个环境图。

```
def double(x):
    返回x * 2

hmmm = double
wow = double(3)
hmmm(wow)
```

[2]由于我们不知道min(...)等内置函数或add(...)等导入函数是怎样的的实现，为了我们自己的利益，我们在调用它们时*不会*画一个新的框架。

2.4 **教程。**画出执行以下代码后的环境图。

```
def f(x):
    返回x

def g(x, y):
    如果x(y)。
        返回不是y
    返回y

x = 3
x = g(f, x)
f = g(f, 0)
```