

CS61A学习笔记

水一寿

2024 年 1 月 23 日

1 Function

1.1 ch.1 expressions

在powershell环境中给出了：

- pi 从 math中import: `from math import pi, sin`
- 第一个Python自带的函数: `max()`
- 自定义函数: `def f(x):`
eg. `return mul(x,x)`

实现参数与参数的相互链接：通过函数实现。

在第一节的最最后给出了一系列函数之间的赋值和计算的一个题目。

1.2 ch.2 environment diagrams

程序员常常在各种盒子之间画箭头：想想肖老师的灵魂画手。

code on the left and frames on the right

the course of execution.

在赋值语言中，python的赋值规则是：先计算右边的表达式，表达式从左算到右，再按照顺序赋值到左边。

1.3 ch.3 defining function

前面提到的def

signature:

```
1 def <name>(<formal parameters>)  
2     return <return expression>
```

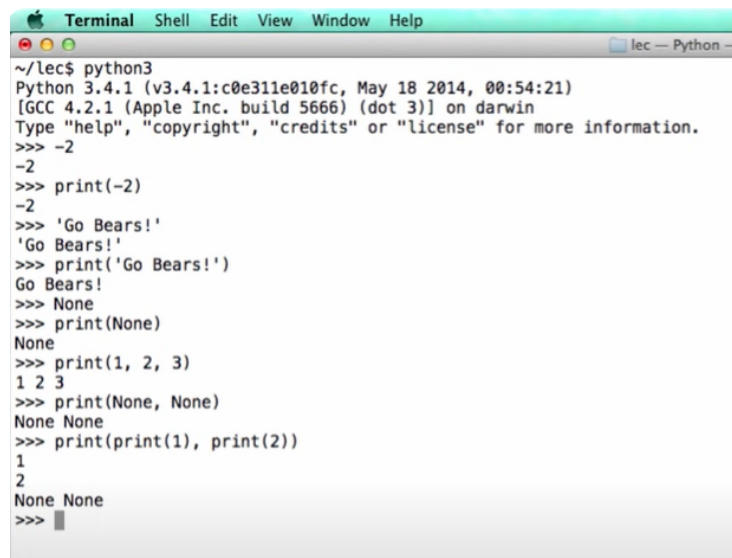
Listing 1: 函数定义的一般形式

函数内部的实际运算发生在函数被调用的时候。当你调用一个函数时，*Python*会执行函数体中的代码，进行相应的运算。在函数被调用之前，函数体内的代码不会执行。

调用的时候直接用:f() **built-in function VS User-defined function**

1.4 ch.4 print and none

解释：先进行括号里面的print(1)和print(2),得到1和2，然后最外面的大括号内进行运算。 print函数的返回值是none，所以返回两个None。



```
Terminal Shell Edit View Window Help  
~/lec$ python3  
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 00:54:21)  
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> -2  
-2  
>>> print(-2)  
-2  
>>> 'Go Bears!'  
'Go Bears!'  
>>> print('Go Bears!')  
Go Bears!  
>>> None  
>>> print(None)  
None  
>>> print(1, 2, 3)  
1 2 3  
>>> print(None, None)  
None None  
>>> print(print(1), print(2))  
1  
2  
None None  
>>> █
```

图 1: 教学示例

None代表了nothing 如果一个函数并不明显地返回什么值，那么它返回None。

函数的分类：

pure functions(有返回值的函数): eg:abs(-2) pow(2,100)

Non-pure functions(无返回值的函数): print()函数

疑难：

python中如何理解”environment is a sequence of frames”

”Environment is a sequence of frames” 这个说法是指在Python中，执行代码时，变量和函数等符号的查找是通过一个被称为环境（environment）的数据结构来实现的。这个环境可以被视为一系列帧（frames）的序列。

在这个上下文中，一个“frame”通常对应于一个作用域，比如函数、模块或者类。每个frame都包含了在该作用域内定义的变量、函数等信息。而整个环境则是这些帧的序列。

当你在代码中引用一个变量或者调用一个函数时，Python解释器会在当前环境的帧序列中查找相应的符号。它会从当前帧开始查找，如果找不到，就会顺着帧序列往外层查找，直到找到为止。

举个简单的例子，考虑下面的代码：

```
1 python
2 x = 10
3
4 def foo():
5     y = 5
6     print(x + y)
7
8 foo()
```

Listing 2: 示例代码

在这个例子中，有两个帧：全局作用域的帧包含变量 ‘x’，而 ‘foo’ 函数的帧包含变量 ‘y’。当 ‘foo’ 函数中执行 ‘print(x + y)’ 时，解释器首先在当前帧（‘foo’ 函数的帧）中查找变量 ‘x’ 和 ‘y’，如果找不到，它会继续在外层的帧（全局作用域的帧）中查找。在这种方式下，环境的帧序列形成了一个符号查找的链。

总的来说，“environment is a sequence of frames” 可以理解为**Python中符号查找的机制**，其中变量、函数等的作用域由帧构成，而环境则是这些帧的有序序列。

2 Control

2.1 Multiple Environments

the procedure of user-defined function def statement call expression calling

引出小节：square(square(3)) 先计算square（3）等于9，再代入。-i得到多个frame.

每个名称在不同的环境中又不同的意义：

eg:

```
1 def square (square)
2     return mul(square,square)
3 square(4)
4 >>>16
```

Listing 3: 示例代码

2.2 Miscellaneous Features

一些难以分类的特点： 2024/10 202.4 2024//10 202 2024 % 10 4 分别与函数truediv floordiv mod对应：

定义函数的时候的时候返回值可以是两个甚至多个
doctest的思想 """ """

2.3 conditional statement

statement: action

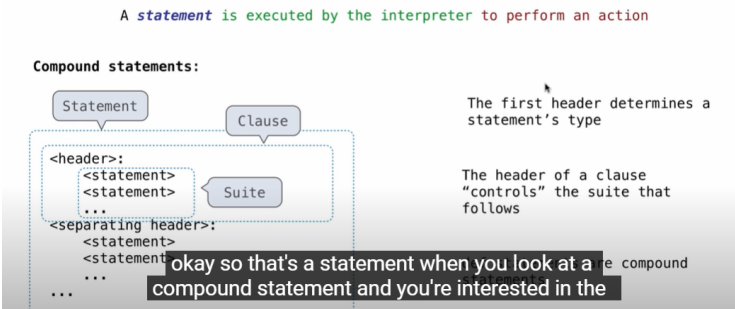


图 2: 教学示例

条件语句用绝对值函数引出基础的条件语句(有abs 的built-in函数):

```
1 def absolute_value(x)
2     if x<0:
3         return -x
4     elif x==0:
5         return 0
6     else:
7         return x
```

Listing 4: 示例代码

仅需注意else if 缩写成了elif

George Boole:逻辑学家, 规定了: **False 0 ' None**都是false的值

anything else 是 true 值。(和C语言一模一样)

2.4 iteration 循环

仅提到while, 和C仅仅在形式上不同。

3 higher-order functions

3.1 fibonacci

斐波那契数列, fibonacci sequence:

```
1 python
2 def fib(n):
3     pred,curr=0,1
4     k=1
5     while k<n:
6         pred,curr=curr,pred+curr
7         k=k+1
8     return curr
```

Listing 5: 斐波那契实现

3.2 control

3.2.1 if statement

条件句子: if statement检测表达式的值, 决定是否执行

写一个函数，使它和if statement的作用大致相同。

```
1 python
2 def if(x,y,t):
3     if x:
4         return y
5     else:
6         return t
```

Listing 6: if statement

但是control结构（if statement）和函数的调用有很大的区别。函数调用的时候会先计算函数内表达式的值，可能不会和控制结构有相同作用。

3.2.2 control expression 控制表达式

逻辑表达式：&& || & | 同C语言

3.3 higher order function 使用参数泛化模式

用一个例子引出：面积的计算。正方形，圆，正六边形等等的计算公式中都存在边长的平方项，故可都统一为边长的平方项乘以一个“形状常数”。

命令行中与Python中利用assert，来判断变量是否满足条件。

```
1 def area(r, shape_constant):
2     assert r > 0, 'A length must be positive'
```

Listing 7: assert

一些计算过程的一般化（如k的累加， k^3 的累加）：找到不同的运算的共同性质，故得以generalize.

3.4 functions as returned values 函数作为返回值

在函数内部再次定义一个函数。

Functions defined **within other function bodies** are bound to names in a *local frame*

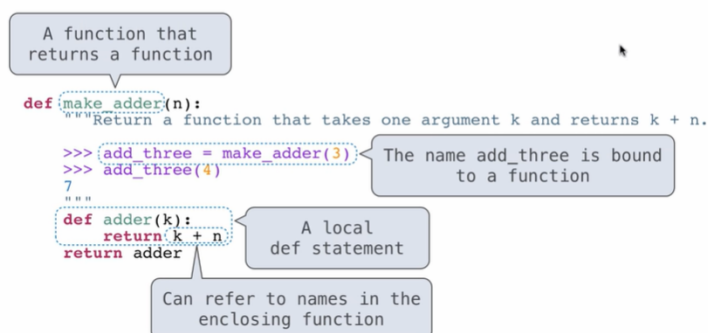


图 3: 返回函数的函数

多级函数的目的在于:多级函数将函数作为变量的值,表明了计算的方法,避免了代码的啰嗦,并且将“关注点”分离,简单来说就是:如果一个问题能分解为独立且较小的问题,就是相对较易解决的。