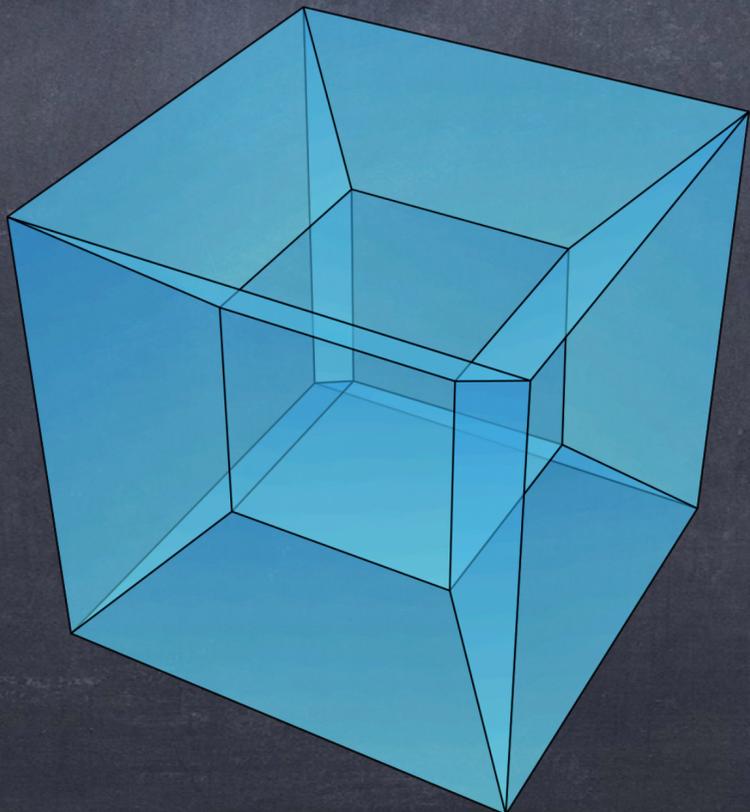


高阶函数

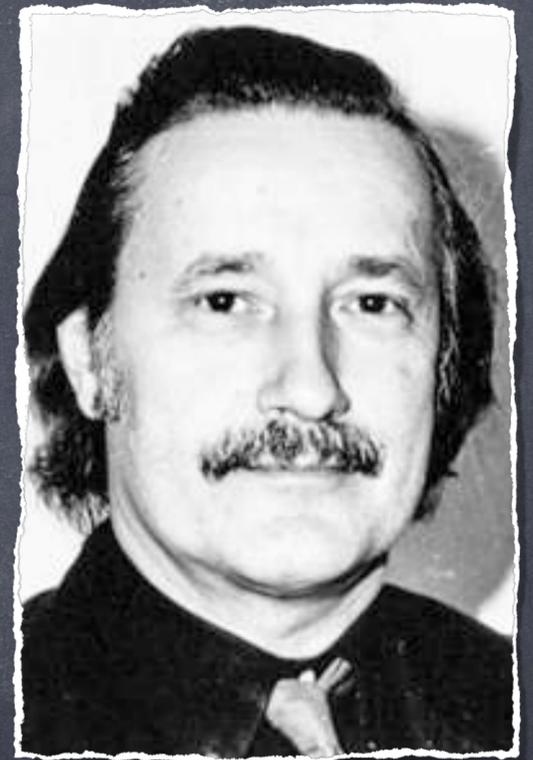


回顾

- ① 函数抽象 (functional abstraction)
- ② 好的函数设计
- ③ 控制语句

高阶 (Higher order) 函数

- ① 函数是“一等” (first-class) 的，他们可以像值一样被操作
- ② 用好高阶函数可以让我们设计出更加好的函数 (即良好的抽象层次、重复更少、泛化性更高)。



Christopher Strachey

"... functions as first-class citizens ..."

高阶 (Higher order) 函数

● 一个高阶函数是：

● 将函数作为实参的函数

and / or

● 其返回值是一个函数

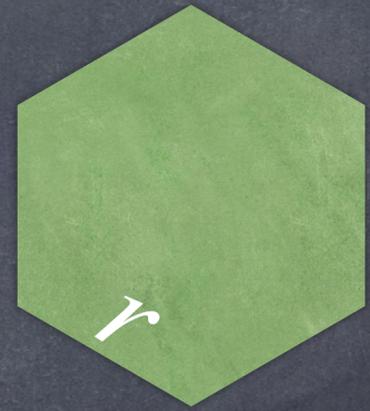
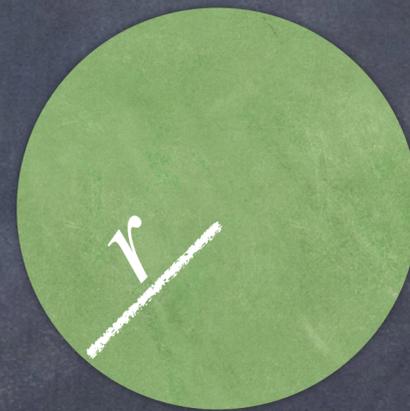
函数作为参数

- ① 让我们先回忆一下“值”作为参数的情况
- ② 抽出共同的部分，剥离不同的部分作为参数

函数作为参数

规则的几何图形将长度和面积关联

图形



面积

$$1 \times r^2$$

$$\pi \times r^2$$

$$\frac{3\sqrt{3}}{2} \times r^2$$

找到共同的部分共享一个实现，不同部分作为参数

函数作为参数

共同部分、不同部分如果不是值，而是计算过程呢？

$$\sum_{k=1}^5 k = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{k=1}^5 k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225$$

$$\sum_{k=1}^5 \frac{8}{(4k-3) * (4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04$$

Summation 的例子

```
def cube(k):  
    return pow(k, 3)
```

带有一个参数的函数

```
def summation(n, term):  
    """Sum the first n terms of a sequence.
```

形参绑定到函数上

```
>>> summation(5, cube)
```

```
225
```

```
"""
```

```
total, k = 0, 1
```

```
while k <= n:
```

```
    total, k = total + term(k), k + 1
```

```
return total
```

cube函数作为实参

被term所绑定的函数在这里调用

$$0 + 1 + 8 + 27 + 64 = 125$$

函数作为返回值

本地 (Locally) 定义的函数

定义在其它函数内的函数被绑定在一个本地的帧中

返回值为一个函数的函数

```
def make_adder(n):  
    """Return a function that takes one argument k and returns k + n.
```

```
>>> add_three = make_adder(3)
```

add_three 名被绑定在一个函数上

```
>>> add_three(4)
```

```
7
```

```
"""
```

```
def adder(k):
```

```
    return k + n
```

在一个def复合语句中的def复合语句

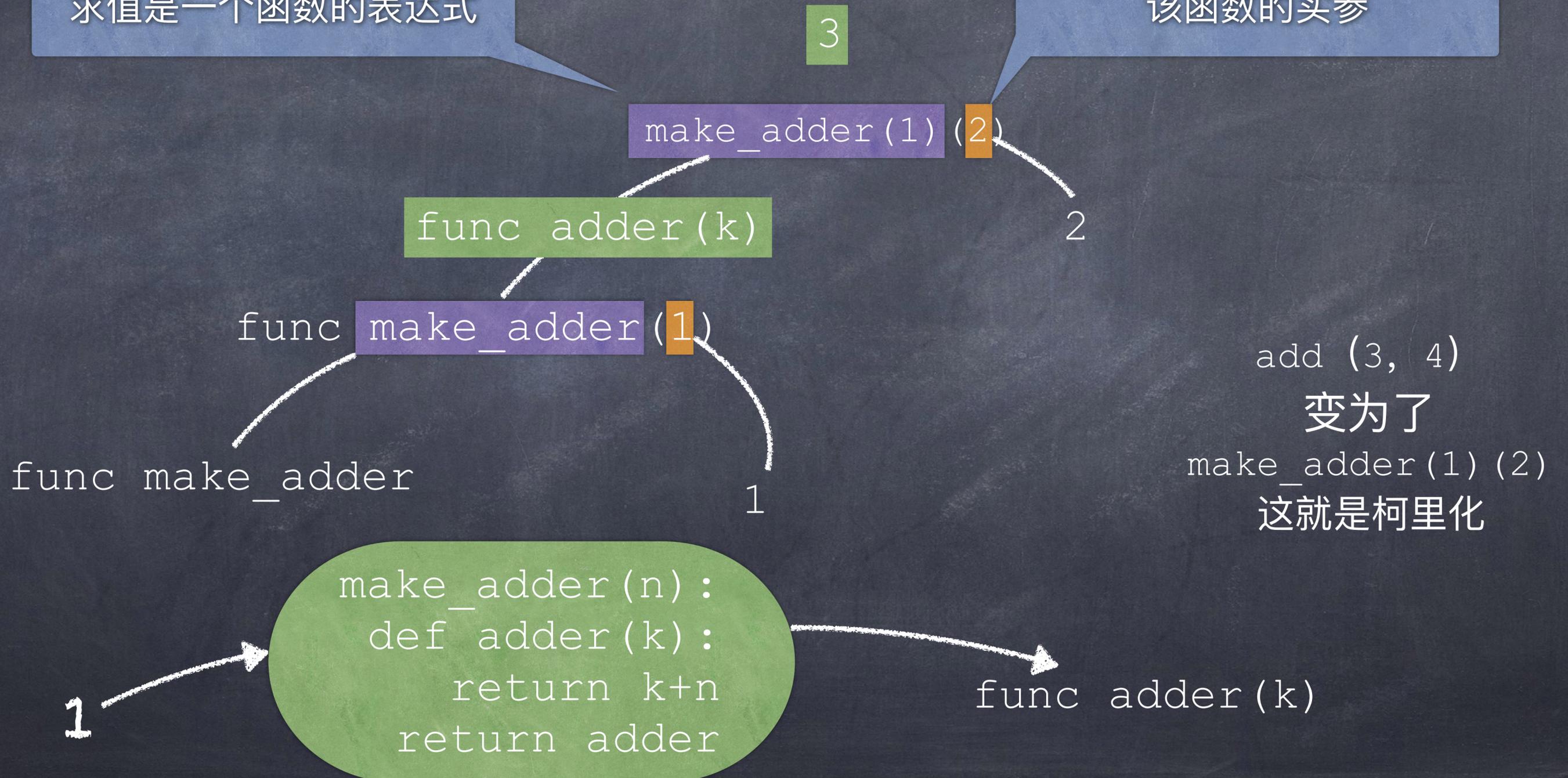
```
return adder
```

可以索引到enclosing函数里的名字 - 闭包

调用表达式作为运算表达式

求值是一个函数的表达式

该函数的实参



例子

```
def make_adder(n):  
    """Return a function that takes one argument k and returns k + n.
```

```
>>> add_three = make_adder(3)
```

```
>>> add_three(4)
```

```
"""
```

```
def adder(k):  
    return k + n  
return adder
```

```
def square(x):  
    return x * x
```

```
def compose(f, g):  
    def h(x):  
        return f(g(x))  
    return h
```

```
compose(square, make_adder(2))(3)
```

例子

```
def print_sums(n):  
    print(n)  
    def next_sum(k):  
        return print_sums(n + k)  
    return next_sum
```

```
print_sums(1)(3)(5)
```

Any questions ?