# PYTHON BASICS

# 高阶函数 - 参数和返回

2020-04-23


STEVEN WANG

# 上节总结

| 类型 | 定义 |
| --- | --- |
| 普通函数 | `def fun_name(args):`<br>`  statement` |
| 匿名函数 | `lambda args : expr` |

args
- 位置参数
- 默认参数
- 可变参数
- 命名关键字参数
- 关键字参数

# 函数分类



- 函数
  - 普通函数
  - 匿名函数
  - 高阶函数
    - 返回 ⇨ 函数
      - map(f, iter)
      - filter(f, iter)
      - reduce(f, iter)
      - partial(f, arg)
    - 函数 ⇨ 参数
      - 闭包 (closure)

# 函数（下）

- 函数式编程
- 函数当返回
- 函数当参数

# 一等公民的函数



函数赋值给变量

```
def f(args):
```

↓

```
f_var = f
```

函数里返回函数

```
def f_outer(args):
    def f_inner():
        do sth on args
    return f_inner
```

↓

```
f_var = f_outer(args)
```

函数储存到容器

```
fn(args)
⋮
f2 =lambda args:expr
def f1(args):
```

↓

```
f_list = [f1, f2, …, fn]
```

函数传递给函数

```
def f(args):
def F(func):
```

↓

```
f_var = f
F(f_var)
```

# 函数（下）

- 函数式编程
- 函数当返回
- 函数当参数

# 闭包 (closure)



1. 闭包通常是嵌套函数 (nested function) 的结构。
2. 该结构由外函数 (outer function) 嵌套内函数 (inner function)。
3. 内函数必须引用非本地 (non-local) 变量。
4. 外函数必须返回内函数。

```
def make_counter(init):
    counter = [init]

    def inc(): counter[0] += 1
    def dec(): counter[0] -= 1
    def get(): return counter[0]
    def reset(): counter[0] = init

    return inc, dec, get, reset
```

```
inc, dec, get, reset = make_counter(0)
```

- make_counter 是外函数
- inc, dec, get, reset 是内函数
- counter 是非本地变量

```
inc()
inc()
inc()
get()
```

3

```
dec()
get()
```

2

```
reset()
get()
```

0

# LEGB 作用域

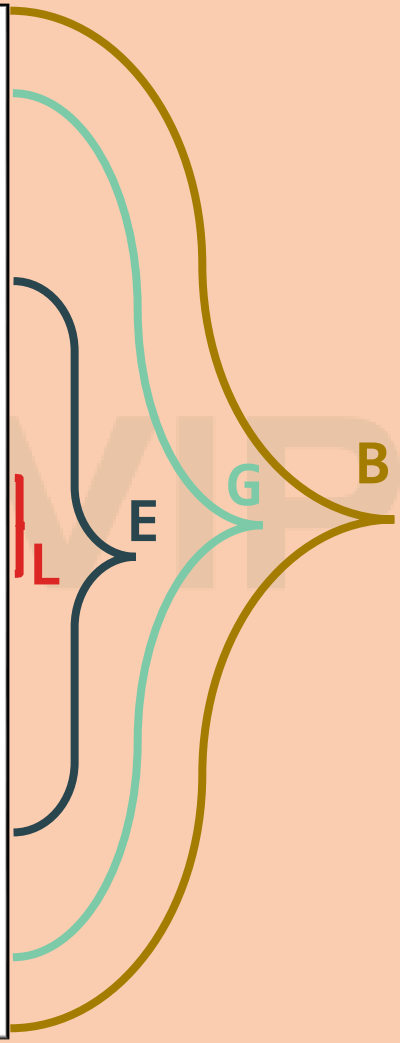
```
x = 'global x'

def outer():
    x = 'enclosed x'

    def inner():
        x = 'local x'
        print(x)

    inner()
    print(x)

outer()
print(x)
```



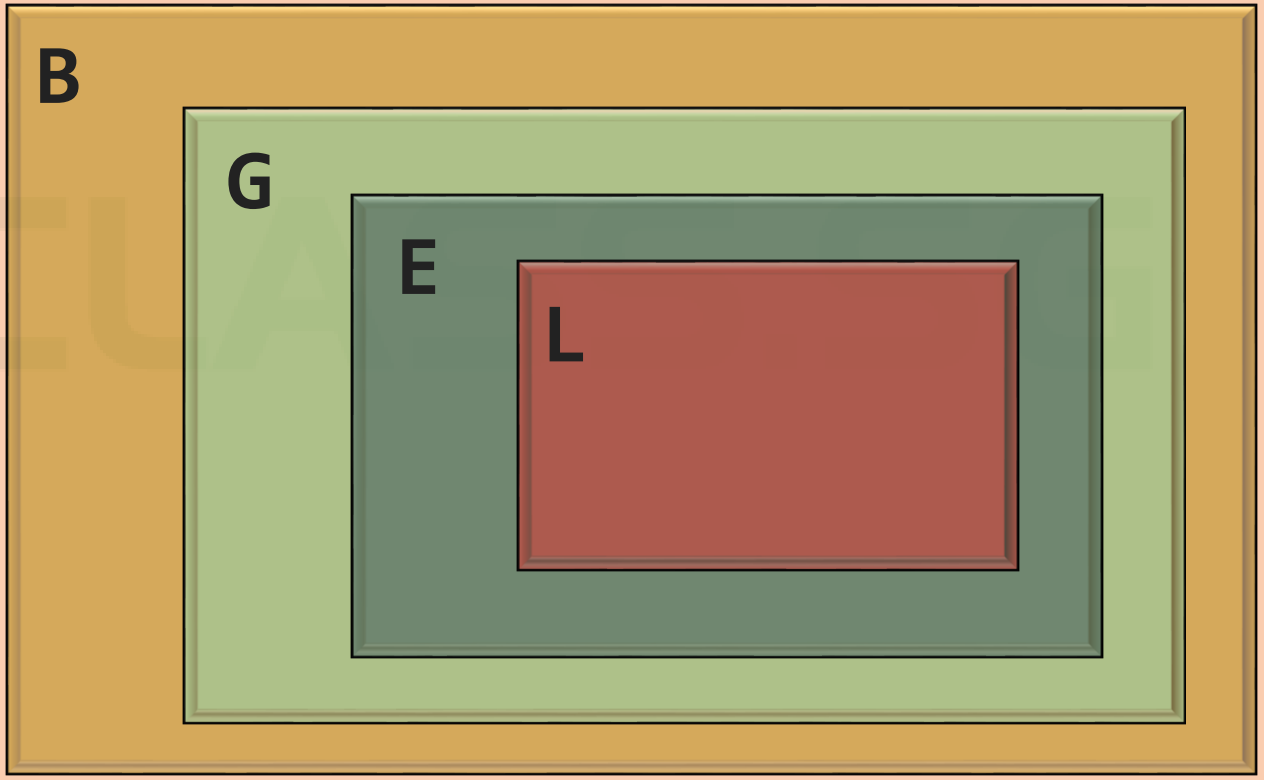

Local: 本地

Global: 全局

LEGB

Enclosed: 内嵌

Built-in: 自带




BGEL = 倍感饿了

# 函数（下）

- 函数式编程
- 函数当返回
- 函数当参数

# 映射函数：map



map( cook, [🐮, 🍠, 🐤, 🌽] ) = [🍔, 🍟, 🍗, 🍿]

# 筛选函数：filter



filter( ismeat, [🍔, 🍟, 🍗, 🍿] ) = [🍔, 🍗]

# 累计函数：reduce



reduce( eat, [🍔, 🍟, 🍗, 🍿] ) = 💩

# Emoji vs 代码



**Emoji 版本😂**

map( cook, [🐮, 🍠, 🐤, 🌽] )
= [🍔, 🍟, 🍗, 🍿]

filter( ismeat, [🍔, 🍟, 🍗, 🍿] )
= [🍔, 🍗]

reduce( eat, [🍔, 🍟, 🍗, 🍿] )
= 💩

```
l = [1, 2, 3, 4, 5]
```

```
m_iter = map( lambda x: x**2, l )
list(m_iter)
```

```
[1, 4, 9, 16, 25]
```

```
f_iter = filter( lambda x: x%2==1, l )
list(f_iter)
```

```
[1, 3, 5]
```

```
from functools import reduce
reduce( lambda x,y: x+y, l )
```

```
15
```

# 偏函数：partial



1. 固定函数中一个或多个参数创建新函数。
2. 新函数用于专门的应用上。

```
from functools import partial
desired_function = partial( origial_function, arguments_to_fix )
```

固定 reverse 参数为 True
创建一个专门逆序排列的函数

```
l = [3, 1, 2, 5, 4]
print( sorted( l ) )
print( sorted( l, reverse = True ) )
```

```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

```
dsort = partial( sorted, reverse = True )
dsort( l )
```

```
[5, 4, 3, 2, 1]
```

# 柯里化 (currying)



柯里化 = 接受一个多参数的函数
⇩
接受多个单参数的函数

$$f(x_1, x_2, x_3, \cdots, x_{n-1}, x_n)$$
$$\Downarrow$$
$$\underbrace{\underbrace{\underbrace{f_1(x_1)(x_2)}_{f_2}(x_3)}_{f_3}\cdots(x_{n-1})}_{f_{n-1}}(x_n)$$

**用闭包实现**

```
def f(x, y):
    return x + y

def g_y(x):
    def f(y):
        return x + y
    return f

g = g_y(2)
```

**用偏函数实现**

```
def f(x, y):
    return x + y

g = partial(f, 2)
```

```
f
g
```

```
<function __main__.f(x, y)>
<function __main__.g_y.<locals>.f(y)>
```

```
print( f(2, 3) )
print( g(3) )
```

```
5
5
```

# 总结

| 函数式编程 | 知识点 | 用处 |
| --- | --- | --- |
| 函数赋值变量 | f_var = f | 定义调用分离 |
| 函数储存容器 | f_list = [f1, f2, …, fn] | 循环调用 |
| 函数当返回 | 闭包 | 装饰器 |
| 函数当参数 | 偏函数 | 柯里化 |

高阶函数

# 下节预告：面向对象编程


VIPCLASS.SG
small class, world class
终身学习
快乐学习
王圣元 Steven Wang
微信公众号：王的机器