面向接口编程,写接口,让别人把接口完成,我只管用.

过程也是对象,也可以像数据一样给他们命名.

闭包性质(closure):可以帮助我们构成更复杂的东西,先创造两个或多个亚复杂的数对,再用数对把他们连在一起,构成复杂的数对,再把两个或多个复杂的数对连在一起构成超复杂的数对.

|  |
| --- |
| # 用DONS实现List  (DEFINE CONS DATA NEXT\_CONS\_POINTER)  (DEFINE MYLIST (LIST DATA1 DATA2 ...))  (CAR (CDR (CDR MYLIST)))  # nil = end of xxx  struct list\_node:  DATA  NEXT\_NODE  def List = list\_node  let MYLIST = A List  # MYLIST.DATA = DATA1  # MYLIST.NEXT\_CONS.DATA = DATA2  # 接下来定义一个放大过程SCALE(MYLIST, SCALE):  LIST-SCALE(LIST\_NODE, SCALE):  if(LIST\_NODE.DATA = null):  ret null  else:  LIST\_NODE.DATA \*= SCALE  ret LIST-SCALE(LIST\_NODE.NEXT\_NODE, SCALE)  # 也可以定义一些别的对List操作的过程  ...... |

|  |
| --- |
| # 现在不再需要对List干啥了,现在要写一个新过程来处理List中的数据再获取这些新数据  # PROCEDURE是一个过程,类似于C的函数指针  MAP(LIST, PROCEDURE):  if(LIST\_NODE.DATA = null):  ret null  else:  PROCEDURE(LIST\_NODE.DATA)  ret MAP (LIST\_NODE.NEXT\_NODE, PROCEDURE)  # 返回的表是List的各节点经过PROCEDURE过程返回的结果组成的新List,即映射(MAP).  # 用Lisp表示为:  (MAP 一个列表 处理列表中各结点元素的过程)  # 返回List每一个数据经过过程Procedure处理后的List  # 现在我不要返回,只需要对List中的DATA做某个过程的操作  # 给定列表,过程,不返回新的List而是只对List完成操作,称为for-each  FOR-EACH(LIST, PROCEDURE)  if(LIST\_NODE.DATA = null)  ret DONE  else:  PROCEDURE(LIST\_NODE.DATA)  ret FOR-EACH (LIST\_NODE.NEXT\_NODE, PROCEDURE)  # FOR-EACH不是递归,因为之前的函数不需要等待返回的值 |

更多的关注于策略操作.

流处理.

元语言抽象:新建一个合适的,强大的语言用于工程中.

考察语言时,关心它的基本元素,它的组合方法,关心能让我们构造更强大东西的东西,和抽象的方式,即把构造出来的东西放入黑箱子,再用它们构造更更强大的东西.

接下来学一个语言,对图像操作:

Rotate(Degree):旋转,返回被旋转了Degree度图

Flip():垂直翻转

Beside(Picture\_A, Picture\_B, S):S-(0, 1),把A和B水平缩放S倍放入新的图O中

Above:跟Beside反着来

经过过程后返回的也是可操作的图像,这就是闭包(封闭性),可以通过这个返回的图像再进行操作,得到新图像在操作...这就是闭包,可以帮助我们快速增加复杂度,例子就快速增加了图像的复杂度.

这个语言的实现:原点,水平分量,竖直分量,这仨构成了一个矩形.

MAKE过程:MAKE-IMAGE

SELECT过程:水平分量,垂直分量,原点

现在给定一个边长为1的正方形,把它的任一定点作为原点,要把它里边的任一线段映射到另一个XY坐标中.

根据要求,可得一Map过程:略去不写.

主要是创建新矩形,然后枚举线段,然后对组成线段的两点进行处理,处理完了再把这两个点连在一起,打印到屏幕上.

这里的每个图像实质上是一个过程,是由仨东西构造成的.

语言嵌入,在一个语言里实现了另一个语言,新实现的语言继承了实现语言的所有东西,不丧失源语言强大的力量.

......此处讲了很多关于图像处理的方法和思想......

什么是数据,什么是过程,他们有区别吗?他们都是对象,有时候过程可以是数据.

软件工程:一种方法学,先计算出任务精准正确,再把它划分为三个子任务,在把三个子子任务之一的子子子任务们处理完,再回退到三个子子任务,处理第二个子子任务,把它详细实现,直到所有任务实现完毕.就得到了一个完整的软件,总览而言,它是一颗任务树.

至此,把那个新实现的语言封装起来,就成了一个Object,这个新语言又是由另一个语言实现的,它被大量使用与新语言的构建,这个语言就是新语言的父语言,新语言就是语言的子语言,子语言继承了父语言的诸多财产.

真正的设计过程,与其说是设计程序,不如说是设计语言.