实验六 Python函数 班级： 21计科1

学号： B20210302112

姓名： 张栋棋

Github地址：https://github.com/yourusername/pykc6

实验目的 学习Python函数的基本用法 学习lambda函数和高阶函数的使用 掌握函数式编程的概念和实践 实验环境 Git Python 3.10 VSCode VSCode插件 实验内容和步骤 第一部分 Python函数

完成教材《Python编程从入门到实践》下列章节的练习：

第8章 函数 第二部分 在Codewars网站注册账号，完成下列Kata挑战：

第一题：编码聚会1 难度： 7kyu

你将得到一个字典数组，代表关于首次报名参加你所组织的编码聚会的开发者的数据。 你的任务是返回来自欧洲的JavaScript开发者的数量。 例如，给定以下列表：

lst1 = [ { 'firstName': 'Noah', 'lastName': 'M.', 'country': 'Switzerland', 'continent': 'Europe', 'age': 19, 'language': 'JavaScript' }, { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent': 'Oceania', 'age': 28, 'language': 'JavaScript' }, { 'firstName': 'Shufen', 'lastName': 'L.', 'country': 'Taiwan', 'continent': 'Asia', 'age': 35, 'language': 'HTML' }, { 'firstName': 'Sumayah', 'lastName': 'M.', 'country': 'Tajikistan', 'continent': 'Asia', 'age': 30, 'language': 'CSS' } ] 你的函数应该返回数字1。 如果，没有来自欧洲的JavaScript开发人员，那么你的函数应该返回0。

注意： 字符串的格式将总是"Europe"和"JavaScript"。 所有的数据将始终是有效的和统一的，如上面的例子。

这个卡塔是Coding Meetup系列的一部分，其中包括一些简短易行的卡塔，这些卡塔是为了让人们掌握高阶函数的使用。在Python中，这些方法包括：filter, map, reduce。当然也可以采用其他方法来解决这些卡塔。

代码： lst1 = [ { 'firstName': 'Noah', 'lastName': 'M.', 'country': 'Switzerland', 'continent': 'Europe', 'age': 19, 'language': 'JavaScript' }, { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent': 'Oceania', 'age': 28, 'language': 'JavaScript' }, { 'firstName': 'Shufen', 'lastName': 'L.', 'country': 'Taiwan', 'continent': 'Asia', 'age': 35, 'language': 'HTML' }, { 'firstName': 'Sumayah', 'lastName': 'M.', 'country': 'Tajikistan', 'continent': 'Asia', 'age': 30, 'language': 'CSS' } ]

# solution 1:

developers = [ d for d in lst1 if d['continent'] == 'Europe' and d['language'] == 'JavaScript' ] print(developers) print(len(developers))

# solution 2:

developers2 = list(filter( lambda d: d['continent']=='Europe' and d['language'] == 'JavaScript' , lst1))

print(developers2) print(len(developers2))

第二题： 使用函数进行计算 难度：5kyu

这次我们想用函数来写计算，并得到结果。让我们看一下一些例子：

seven(times(five())) # must return 35 four(plus(nine())) # must return 13 eight(minus(three())) # must return 5 six(divided_by(two())) # must return 3 要求：

从0（"零"）到9（"九"）的每个数字都必须有一个函数。 必须有一个函数用于以下数学运算：加、减、乘、除。 每个计算都由一个操作和两个数字组成。 最外面的函数代表左边的操作数，最里面的函数代表右边的操作数。 除法应该是整数除法。 例如，下面的计算应该返回2，而不是2.666666...。

eight(divided_by(three())) 代码： def zero(fun=None): # 如果fun是None，就返回0，否则返回fun(0) return fun(0) if fun else 0

def one(fun=None): return fun(1) if fun else 1

def plus(y): # 返回一个函数，这个函数的作用是加y return lambda x: x + y

# 这是一个加5的函数，可以用来加任何数

plus5 = plus(5) print(plus5(3)) 8

# 测试没有参数的函数

print(zero()) print(one()) 0 1 zero(print) # print(0) one(print) # print(1) 0 1

# one(plus(zero())) ->

# one(plus(0)) ->

# one(lambda x:x + 0) ->

# fun(1) = lambda:1 + 0 -> 1

one(plus(zero())) 1

第三题： 缩短数值的过滤器(Number Shortening Filter) 难度：6kyu

在这个kata中，我们将创建一个函数，它返回另一个缩短长数字的函数。给定一个初始值数组替换给定基数的 X 次方。如果返回函数的输入不是数字字符串，则应将输入本身作为字符串返回。

例子：

filter1 = shorten_number(['','k','m'],1000) filter1('234324') == '234k' filter1('98234324') == '98m' filter1([1,2,3]) == '[1,2,3]' filter2 = shorten_number(['B','KB','MB','GB'],1024) filter2('32') == '32B' filter2('2100') == '2KB'; filter2('pippi') == 'pippi' 代码： def shorten_number(suffixes, base):

```
# 定义一个函数
def my_filter(number):
    # 在函数内部可以使用外部的变量suffixes, base
    return number

# 返回值是一个函数
return my_filter
```

my_fun = shorten_number(['','k','m'],1000) my_fun('234234') '234234' def shorten_number(suffixes, base):

```
# 定义一个函数
def my_filter(data):
    try:
        # 将函数输入转换为整数
        number = int(data)

    # 如果输入的数据不能转换为整数，直接转换为str返回
    except (TypeError, ValueError):
        return str(data)

    # 输入的number可以转换为整数
    else:
        # i用来跟踪suffixes列表的索引
        i = 0

        # 每次循环将输入的数字除以base，索引i+1
        # 如果除以base等于0或者索引等于len(suffixes)-1，结束循环
        while number//base > 0 and i < len(suffixes)-1:
            number //= base
            i += 1
        return str(number) + suffixes[i]

# 返回值是一个函数
return my_filter
```

filter1 = shorten_number(['','k','m'],1000) print(filter1('234324')) # == '234k' print(filter1('98234324')) # == '98m' print(filter1([1,2,3])) # == '[1,2,3]'

filter2 = shorten_number(['B','KB','MB','GB', 'TB'],1024) print(filter2('32')) # == '32B' print(filter2('2100')) # == '2KB'; print(filter2('2100000000000000000000')) # == '2KB'; print(filter2('pippi')) # == 'pippi' 234k 98m [1, 2, 3] 32B 2KB 1909938873TB pippi

第四题： 编码聚会7 难度： 6kyu

您将获得一个对象序列，表示已注册参加您组织的下一个编程聚会的开发人员的数据。

您的任务是返回一个序列，其中包括最年长的开发人员。如果有多个开发人员年龄相同，则将他们按照在原始输入数组中出现的顺序列出。

例如，给定以下输入数组：

list1 = [ { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP' }, { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asia', 'age': 38, 'language': 'Python' }, { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'Europe', 'age': 19, 'language': 'Python' }, { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'language': 'PHP' }, ] 您的程序应该返回如下结果：

[ { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP' }, { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'language': 'PHP' }, ] 注意：

输入的列表永远都包含像示例中一样有效的正确格式的数据，而且永远不会为空。 代码： lst = [ { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP' }, { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asia', 'age': 38, 'language': 'Python' }, { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'Europe', 'age': 19, 'language': 'Python' }, { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'language': 'PHP' }, ]

# 使用max函数的key参数可以找到年龄最大的程序员

max(lst, key=lambda d:d['age']) {'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP'}

# 找到年龄最大的程序员

oldest = max(lst, key=lambda d:d['age'])

# 打印他的年龄

old_age = oldest['age'] print(old_age)

# 根据年龄过滤列表

list(filter(lambda d:d['age'] == old_age, lst)) 49 [{'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP'}, {'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'language': 'PHP'}]

# best solution：

def find_senior(lst):

```
# 利用生成器作为max函数的参数
# 找到最大的年龄
mage = max(a['age'] for a in lst)

# 利用列表推导返回结果
return [a for a in lst if a['age']==mage]
```

第五题： Currying versus partial application 难度： 4kyu

Currying versus partial application是将一个函数转换为具有更小arity(参数更少)的另一个函数的两种方法。虽然它们经常被混淆，但它们的工作方式是不同的。目标是学会区分它们。

Currying

是一种将接受多个参数的函数转换为以每个参数都只接受一个参数的一系列函数链的技术。

Currying接受一个函数：

f：X × Y → R 并将其转换为一个函数：

f'：X → (Y → R) 我们不再使用两个参数调用f，而是使用第一个参数调用f'。结果是一个函数，然后我们使用第二个参数调用该函数来产生结果。因此，如果非curried f被调用为：

f(3, 5) 那么curried f'被调用为：

f'(3)(5) 示例 给定以下函数：

def add(x, y, z): return x + y + z 我们可以以普通方式调用：

add(1, 2, 3) # => 6 但我们可以创建一个curried版本的add(a, b, c)函数：

curriedAdd = lambda a: (lambda b: (lambda c: add(a,b,c))) curriedAdd(1)(2)(3) # => 6 Partial application 是将一定数量的参数固定到函数中，从而产生另一个更小arity(参数更少)的函数的过程。

部分应用接受一个函数：

f：X × Y → R 和一个固定值x作为第一个参数，以产生一个新的函数

f'：Y → R f'与f执行的操作相同，但只需要填写第二个参数，这就是其arity比f的arity少一个的原因。可以说第一个参数绑定到x。

示例:

partialAdd = lambda a: (lambda *args: add(a,*args)) partialAdd(1)(2, 3) # => 6 你的任务是实现一个名为curryPartial()的通用函数，可以进行currying或部分应用。

例如：

curriedAdd = curryPartial(add) curriedAdd(1)(2)(3) # => 6

partialAdd = curryPartial(add, 1) partialAdd(2, 3) # => 6 我们希望函数保持灵活性。

所有下面这些例子都应该产生相同的结果：

curryPartial(add)(1)(2)(3) # =>6 curryPartial(add, 1)(2)(3) # =>6 curryPartial(add, 1)(2, 3) # =>6 curryPartial(add, 1, 2)(3) # =>6 curryPartial(add, 1, 2, 3) # =>6 curryPartial(add)(1, 2, 3) # =>6 curryPartial(add)(1, 2)(3) # =>6 curryPartial(add)()(1, 2, 3) # =>6 curryPartial(add)()(1)()()(2)(3) # =>6

curryPartial(add)()(1)()()(2)(3, 4, 5, 6) # =>6 curryPartial(add, 1)(2, 3, 4, 5) # =>6

curryPartial(curryPartial(curryPartial(add, 1), 2), 3) # =>6 curryPartial(curryPartial(add, 1, 2), 3) # =>6 curryPartial(curryPartial(add, 1), 2, 3) # =>6 curryPartial(curryPartial(add, 1), 2)(3) # =>6 curryPartial(curryPartial(add, 1)(2), 3) # =>6 curryPartial(curryPartial(curryPartial(add, 1)), 2, 3) # =>6 代码提交地址： https://www.codewars.com/kata/53cf7e37e9876c35a60002c9

第三部分 使用Mermaid绘制程序流程图

安装VSCode插件：

Markdown Preview Mermaid Support Mermaid Markdown Syntax Highlighting 使用Markdown语法绘制你的程序绘制程序流程图（至少一个），Markdown代码如下：

程序流程图

显示效果如下：

查看Mermaid流程图语法-->点击这里

使用Markdown编辑器（例如VScode）编写本次实验的实验报告，包括实验过程与结果、实验考查和实验总结，并将其导出为 PDF格式 来提交。

实验过程与结果 请将实验过程与结果放在这里，包括：

第一部分 Python函数 第二部分 Codewars Kata挑战 第三部分 使用Mermaid绘制程序流程图 注意代码需要使用markdown的代码块格式化，例如Git命令行语句应该使用下面的格式：

Git命令

显示效果如下：

git init git add . git status git commit -m "first commit" 如果是Python代码，应该使用下面代码块格式，例如：

Python代码

显示效果如下：

def add_binary(a,b): return bin(a+b)[2:] 代码运行结果的文本可以直接粘贴在这里。

注意：不要使用截图，Markdown文档转换为Pdf格式后，截图可能会无法显示。

实验考查 请使用自己的语言并使用尽量简短代码示例回答下面的问题，这些问题将在实验检查时用于提问和答辩以及实际的操作。

什么是函数式编程范式？ 函数式编程（Functional Programming，FP），是一种编程范式。我们常听说的编程范式还有面向过程编程、面向对象编程。（编程范式：思想 + 实现的方式） 面向对象编程：把现实世界中的事物抽象成程序世界中的类和对象，通过封装、继承和多态来演示事物事件的联系。 函数式编程**：把现实世界

的事物和事物之间的联系**抽象到程序世界（对运算过程进行抽象） 程序的本质：根据输入通过某种运算获得相应的输出，程序开发过程中会涉及很多有输入和输出的函数。 x -> f(联系、映射) -> y, y=f(x) 函数式编程中的函数指的不是程序中的函数（方法），而是数学中的函数即映射关系，例如：y = sin(x)，x和y的关系。 纯函数：相同的输入始终得到相同的输出 函数式编程用来描述数据（函数）之间的映射 什么是lambda函数？请举例说明。 lambda函数，也称匿名函数，是Python中一种快速定义小型函数的技巧。它可以在不用完整定义一个函数的情况下完成函数调用，并且通常只在使用一次时使用。 语法格式： lambda arguments: expression 其中，arguments是lambda函数的形式参数，而expression则是函数的主体内容（通常是一个表达式），它会根据形参返回函数的执行结果。

什么是高阶函数？常用的高阶函数有哪些？这些高阶函数如何工作？使用简单的代码示例说明。 接受一个或多个函数作为输入 输出一个函数 也就是说高阶函数是对其他函数进行操作的函数，可以将它们作为参数传递，或者是返回它们。 简单来说，高阶函数是一个接收函数作为参数传递或者将函数作为返回值输出的函数。

已知如下数组，编写一个程序将数组扁平化去并除其中重复部分数据，最终得到一个升序且不重复的数组

var arr = [ [1, 2, 2], [3, 4, 5, 5], [6, 7, 8, 9, [11, 12, [12, 13, [14] ] ] ], 10];

arr.toString().split(',').sort((a,b) => a-b).filter((item,index,self) => self.indexOf(item) === index) 实验总结 我学会了各种函数的用法以及语法，可以解决跟多的实际问题，特别的我理解了高阶函数是一个接收函数作为参数传递或者将函数作为返回值输出的函数。