实验六 Python函数

班级： 21计科03

学号： B20210302325

姓名： 欧阳浩

Github地址：https://github.com/0hh4o/python_course

CodeWars地址：https://www.codewars.com/users/0hh4o

实验目的

1. 学习Python函数的基本用法
2. 学习lambda函数和高阶函数的使用
3. 掌握函数式编程的概念和实践

实验环境

1. Git
2. Python 3.10
3. VSCode
4. 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
  { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent': 'Oceania', 'age': 2
  { 'firstName': 'Shufen', 'lastName': 'L.', 'country': 'Taiwan', 'continent': 'Asia', 'age': 35
  { 'firstName': 'Sumayah', 'lastName': 'M.', 'country': 'Tajikistan', 'continent': 'Asia', 'age
]
```

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。

代码提交地址

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()))
```

代码提交地址：
https://www.codewars.com/kata/525f3eda17c7cd9f9e000b39

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'
```

代码提交地址：
https://www.codewars.com/kata/56b4af8ac6167012ec00006f

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'
  { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asia', 'age': 
  { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'Europe', 'a
  { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, '
]
```

您的程序应该返回如下结果：

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

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。

代码提交地址：
https://www.codewars.com/kata/582887f7d04efdaae3000090

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

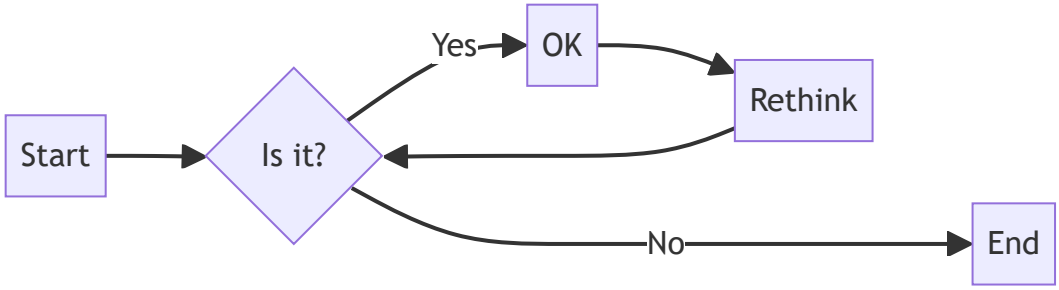
使用Mermaid绘制程序流程图

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：

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

第二部分

第一题：编码聚会1

代码如下：

```
def count_developers(lst):
    num = 0
    for dic in lst:
        for c in dic:
            if dic['continent'] == 'Europe' and dic['language'] == 'JavaScript':
                num += 1
    if num != 0:
        return num/6
    else:
        return 0
```

7kyu Coding Meetup #1 - Higher-Order Functions Series - Count the number of JavaScript developers coming from Europe

Python:

```python
def count_developers(lst):
    num = 0
    for dic in lst:
        for c in dic:
            if dic['continent'] == 'Europe' and dic['language'] == 'JavaScript':
                num += 1
    if num != 0:
        return num/6
    else:
        return 0
```

5 days ago • Refactor • Discuss

第二题：使用函数进行计算

代码如下：

```
def zero(func=None):
    if func:
        return func(0)
    return 0

def one(func=None):
    if func:
        return func(1)
    return 1

def two(func=None):
    if func:
        return func(2)
    return 2

def three(func=None):
    if func:
        return func(3)
    return 3

def four(func=None):
    if func:
        return func(4)
    return 4

def five(func=None):
    if func:
        return func(5)
    return 5

def six(func=None):
    if func:
        return func(6)
    return 6

def seven(func=None):
    if func:
        return func(7)
    return 7

def eight(func=None):
    if func:
        return func(8)
```

```
    return 8

def nine(func=None):
    if func:
        return func(9)
    return 9

def plus(num):
    return lambda x: x + num

def minus(num):
    return lambda x: x - num

def times(num):
    return lambda x: x * num

def divided_by(num):
    return lambda x: x // num
```

第三题：缩短数值的过滤器

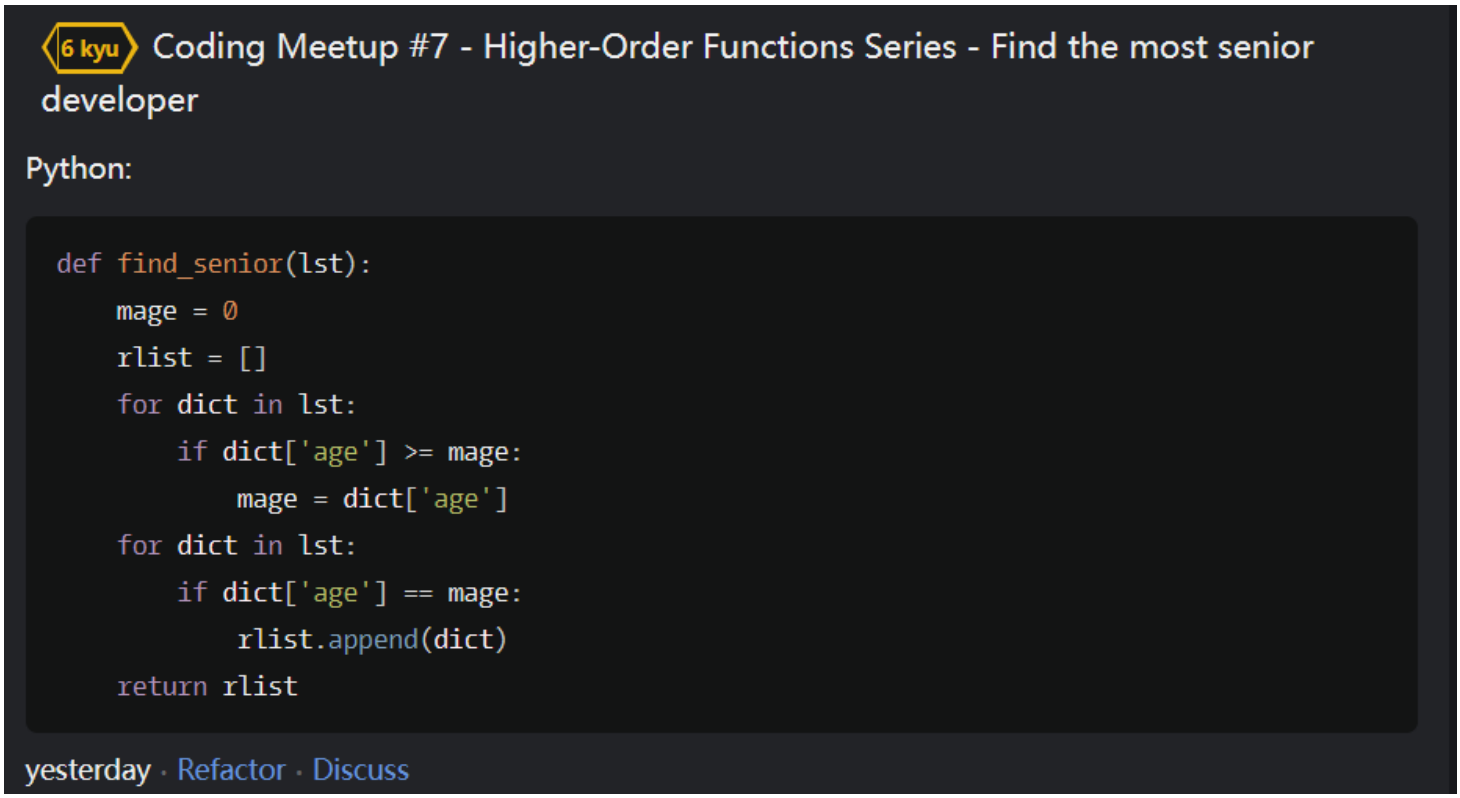
代码如下：

```python
def shorten_number(units, base):
    def shorten(num):
        if isinstance(num, str) and num.isdigit():
            num = int(num)
            for index,unit in enumerate(units):
                if (num < base) or (index == len(units)-1):
                    return f"{num}{unit}"
                num /= base
                num = int(num)
            return f"{num}{units[-1]}"
        else:
            return str(num)
    return shorten
```

第四题：编码聚会7

代码如下：

```python
def find_senior(lst):
    mage = 0
    rlist = []
    for dict in lst:
        if dict['age'] >= mage:
            mage = dict['age']
    for dict in lst:
        if dict['age'] == mage:
            rlist.append(dict)
    return rlist
```

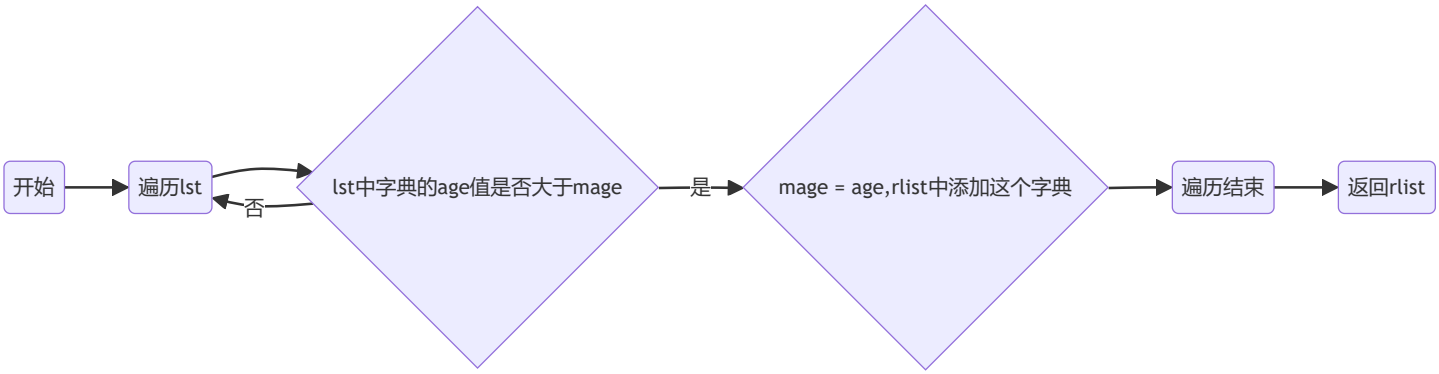


第五题：Currying versus partial application

这题不会写，我自己的代码不能通过最后一个测试样例，
这里就贴上了老师的答案
代码如下：

```python
def curry_partial(f, *args):

    # 如果f不是函数，直接返回
    if not callable(f):
        return f

    # 查看函数f需要的参数个数
    num_args = f.__code__.co_argcount

    # 如果f函数不需要参数，说明f是curry_partial函数
    if num_args == 0:
        return f(*args)

    if len(args) >= num_args:
        return f(*args[:num_args])

    def inner(*params):
        all_args = [*args, *params]

        # 如果没有参数，这是curry函数，使用链式调用
        if not args:
            return curry_partial(f, *all_args)

        # 如果第一个参数不是函数，这是curry函数，使用链式调用
        if not callable(args[0]):
            return curry_partial(f, *all_args)

        # 如果第一个参数是函数，这是partial函数，使用部分函数调用
        fn = args[0]
        num_args2 = fn.__code__.co_argcount

        # 如果fn函数不需要参数，说明fn是curry_partial函数
        if num_args2 == 0:
            return fn(*all_args)

        if len(all_args) >= num_args2:
            return fn(*all_args[:num_args2])
        else:
            return curry_partial(fn, *all_args)

    return inner
```

第三部分



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。

1. 什么是函数式编程范式?
   答：函数式编程范式是一种编程范式，它将计算视为数学函数的计算，并避免使用状态和可变数据。函数式编程强调函数的纯粹性、不可变性和高阶函数的使用。它通常使用lambda表达式和闭包来创建函数，并使用高阶函数和函数组合来进行复杂的计算。函数式编程范式的目标是编写简洁、可靠和可维护的代码，以及更好地利用多核处理器和并行计算。
2. 什么是lambda函数？请举例说明。
   答：Lambda函数是一种匿名函数，也称为内联函数或者函数字面量。它是一种在需要时即时定义的简单函数，通常用于函数式编程范式中。
   Lambda函数的语法通常是：lambda 参数列表: 表达式
   例如，下面是一个简单的lambda函数，它接受两个参数并返回它们的和：
   add = lambda x, y: x + y
   print(add(3, 5)) # 输出结果为8
   在这个例子中，lambda函数定义为lambda x, y: x + y，它接受两个参数x和y，并返回它们的和。然后我们调用这个lambda函数，传入参数3和5，得到的结果是8。Lambda函数通常用于简单的、一次性的函数，可以在需要时快速定义和使用。
3. 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。
   答：常用的高阶函数包括map()、filter()、reduce()和sorted()。
   map()函数接受一个函数和一个可迭代对象作为参数，对可迭代对象中的每个元素应用函数，并返回一个包含结果的迭代器。
   使用map函数将列表中的每个元素平方
   numbers = [1, 2, 3, 4, 5]
   squared = map(lambda x: x**2, numbers)

print(list(squared)) # 输出结果为[1, 4, 9, 16, 25]

filter()函数接受一个函数和一个可迭代对象作为参数，对可迭代对象中的每个元素应用函数，并返回一个由使得函数返回值为True的元素所组成的迭代器。

使用filter函数筛选出列表中的偶数

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

even_numbers = filter(lambda x: x % 2 == 0, numbers)

print(list(even_numbers)) # 输出结果为[2, 4, 6, 8]

reduce()函数接受一个函数和一个可迭代对象作为参数，对可迭代对象中的元素进行累积计算，并返回一个单一的结果。

使用reduce函数计算列表中所有元素的和

from functools import reduce

numbers = [1, 2, 3, 4, 5]

sum = reduce(lambda x, y: x + y, numbers)

print(sum) # 输出结果为15

sorted()函数接受一个可迭代对象作为参数，并返回一个新的已排序的列表。

使用sorted函数对列表进行排序

numbers = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

sorted_numbers = sorted(numbers)

print(sorted_numbers) # 输出结果为[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]

实验总结

这次实验中我学习了python中函数的编程理念，学习了lambda表达式和一些高阶函数，但我还不能解决所有的kata，还需要努力