实验六 Python函数

班级： 21计科02

学号： 20210302221

姓名： 王日晖

Github地址：https://github.com/wrrh

CodeWars地址：https://www.codewars.com/users/wrhh

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': 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': 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：

![程序流程图] 显示效果如下：

```
flowchart LR
    A[Start] --> B{Is it?}
    B -->|Yes| C[OK]
    C --> D[Rethink]
    D --> B
    B ---->|No| E[End]
```

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：

第一题：编码聚会1

显示效果如下：

```
def count_developers(lst):
    developers = 0
    for row in lst:
        continent = row.get('continent')
        language = row.get('language')
        if continent == 'Europe' and language == 'JavaScript':
            developers += 1
    return developers
```

第二题： 使用函数进行计算

显示效果如下：

```
def identity(a): return a

def zero(f=identity): return f(0)
def one(f=identity): return f(1)
def two(f=identity): return f(2)
def three(f=identity): return f(3)
def four(f=identity): return f(4)
def five(f=identity): return f(5)
def six(f=identity): return f(6)
def seven(f=identity): return f(7)
def eight(f=identity): return f(8)
def nine(f=identity): return f(9)

def plus(b): return lambda a: a + b
def minus(b): return lambda a: a - b
def times(b): return lambda a: a * b
def divided_by(b): return lambda a: a // b
```

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

显示效果如下：

```
def shorten_number(suffixes,base):
    def func(n):
        try: n = int(float(n))
```

```
            except: return str(n)
            i = 0; m = len(suffixes)-1
            while base<n and i<m: n = n//base; i += 1
            return str(n)+suffixes[i]
        return func
```

第三题： 缩短数值的过滤器(Number Shortening Filter)[程序流程图]

显示效果如下：

```
flowchart LR
A[输入参数: suffixes, base, n]
B[将n转换为整数]
C[设置循环计数器i为0]
D[设置循环终止条件i B
B --> C
C --> D
D --> |是| E
D --> |否| G
E --> F
F --> D
G --> H
H --> I
```

第四题： 编码聚会7

显示效果如下：

```
def find_senior(lst):
    new_lst = [lst[0]]
    for person in lst[1:]:
        if person['age'] > new_lst[0]['age']:
            new_lst = [person]
        elif person['age'] == new_lst[0]['age']:
            new_lst += [person]
    return new_lst
```

第五题： Currying versus partial application

显示效果如下：

```

```

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。

1. 什么是函数式编程范式?

> 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求值过程。函数式编程强调函数的纯粹性和无状态性，避免使用可变状态和副作用。它通过将问题分解为独立的函数，以及利用函数的组合、高阶函数和递归等技术来实现程序的设计和开发。

2. 什么是lambda函数？请举例说明。

> lambda函数是一种匿名函数，它可以在不定义函数名称的情况下直接使用。lambda函数通常用于简单的函数操作，特别是作为参数传递给其他函数。它的语法形式是`lambda 参数: 表达式`。

示例：

```
# 计算两个数的和
add = lambda a, b: a + b
result = add(3, 5)
print(result)  # 输出结果为8
```

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。

> 高阶函数是指能够接受其他函数作为参数或返回函数作为结果的函数。常用的高阶函数包括`map()`、`filter()`和`reduce()`。

- `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，并返回一个新的可迭代对象。

  示例：

```
# 对列表中的每个元素进行平方运算
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(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，根据函数的返回值来筛选可迭代对象中的元素，并返回一个新的可迭代对象。

示例：

```
# 筛选出列表中的偶数
numbers = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x % 2 == 0, numbers))
print(evens)  # 输出结果为[2, 4]
```

- `reduce()`函数在Python 3中已经被移至`functools`模块中。它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，并返回最终的结果。

  示例：

```
# 计算列表中所有元素的乘积
from functools import reduce
numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # 输出结果为120
```

实验总结

总结一下这次实验你学习和使用到的知识，例如：编程工具的使用、数据结构、程序语言的语法、算法、编程技巧、编程思想。

> 在这次实验中，我学习了函数式编程范式的基本概念和特点，以及lambda函数和高阶函数的用法。通过编写简短的代码示例，我理解了这些概念在实际编程中的应用场景和工作原理。此外，我还学习了使用Python编程语言来实现这些概念，并运用编程工具进行代码开发和调试。通过这次实验，我对函数式编程范式有了更深入的理解，并且掌握了一些常用的函数式编程技巧和思想。