实验六 Python函数

班级： 21计科3班

学号： B20210302303

姓名： 文凯

Github地址：https://github.com/kaihuang614

CodeWars地址：https://www.codewars.com/users/kaihuang614

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：

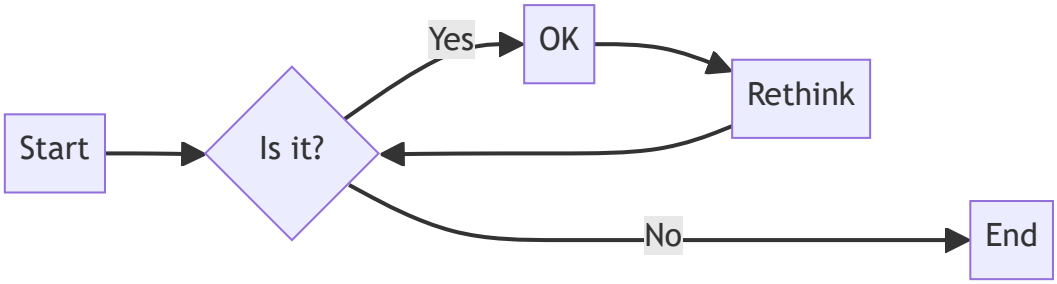
- Markdown Preview Mermaid Support

- 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：

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

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⚓

第一题：编码聚会1

```
def count_developers(lst):
    res_list = list(filter(lambda x : x['continent'] == 'Europe' and x['language'] == 'JavaScript', lst)) # 答案列表存储符合条件的数据
```

```
    return len(res_list) # 返回答案列表的元素个数
```

第二题： 使用函数进行计算

```
num_dict = {'1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '0': 0}

def zero(x=None):
    return evaluate_expression(0, x)

def one(x=None):
    return evaluate_expression(1, x)

def two(x=None):
    return evaluate_expression(2, x)

def three(x=None):
    return evaluate_expression(3, x)

def four(x=None):
    return evaluate_expression(4, x)

def five(x=None):
    return evaluate_expression(5, x)

def six(x=None):
    return evaluate_expression(6, x)

def seven(x=None):
    return evaluate_expression(7, x)

def eight(x=None):
    return evaluate_expression(8, x)

def nine(x=None):
    return evaluate_expression(9, x)

def plus(x):
    return '+,' + str(x)

def minus(x):
    return '-,' + str(x)

def times(x):
    return '*,' + str(x)

def divided_by(x):
    return '/,' + str(x)

def evaluate_expression(x, y):
    if y is None:
```

```
        return x

    operator, num = y.split(',')
    num = num_dict[num]

    if operator == '+':
        return x + num
    elif operator == '-':
        return x - num
    elif operator == '*':
        return x * num
    elif operator == '/':
        return int(x / num)
```

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

```
def shorten_number(suffixes, base):
    def filter(text):
        try:
            num = int(text)
        except (ValueError, TypeError):
            return str(text)

        i = 0
        while num > base:
            if i == len(suffixes) - 1:
                break
            num = num / base
            i += 1

        return str(int(num)) + suffixes[i]

    return filter
```

第四题： 编码聚会7

```
def find_senior(lst):
    max_age = max(item['age'] for item in lst) # 最大年龄存储所给列表lst中元素字典字段'age'的最大值
    res_list = list(filter(lambda x : x['age'] == max_age, lst)) # 答案列表存储满足最大年龄的数据 用filter函数过滤lst
    return res_list # 返回答案列表
```

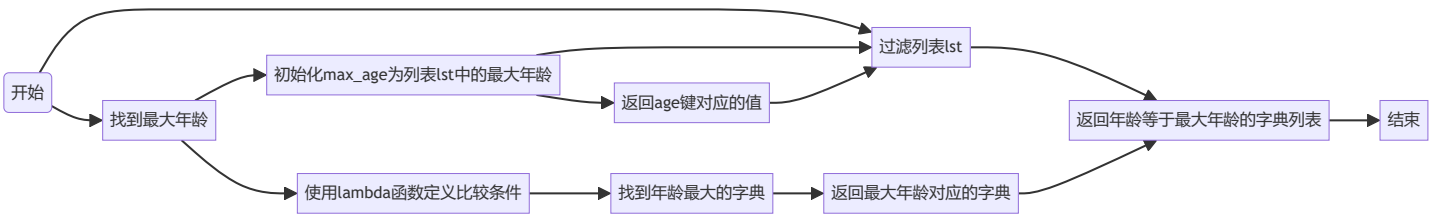
第五题： Currying versus partial application

```
def curry_partial(f,*initial_args):
    if not callable(f):
        return f
```

```
    num_args = f.__code__.co_argcount


    if num_args == 0:
        return f(*initial_args)

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

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


    if not initial_args:
        return curry_partial(f, *all_args)

    if not callable(initial_args[0]):
        return curry_partial(f, *all_args)


    fn = initial_args[0]
    num_args2 = fn.__code__.co_argcount

    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 使用Mermaid绘制程序流程图⚓

第四题： 编码聚会7



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，它强调使用纯函数来构建程序。在函数式编程中，函数被视为一等公民，可以像变量一样被传递、存储和操作。

函数式编程的特点包括：

1、不可变性：函数式编程鼓励使用不可变数据结构和不可变变量，避免副作用，从而减少错误和提高代码可读性。
2、纯函数：函数式编程鼓励编写没有副作用的纯函数，即对于相同的输入，始终返回相同的输出，不依赖于外部状态。
3、高阶函数：函数可以接受函数作为参数或返回函数，这样可以使用函数组合和函数传递来实现更灵活的编程。
4、递归：函数式编程常常使用递归来实现循环和迭代，通过函数的自我调用来解决问题。

以下是一个简短的Python代码示例，展示了函数式编程的一些特性：

```
# 使用函数式编程实现列表中所有元素的平方
numbers = [1, 2, 3, 4, 5]

# 使用map函数对列表中的每个元素进行平方操作
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)  # 输出: [1, 4, 9, 16, 25]

# 使用filter函数筛选出列表中的偶数
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # 输出: [2, 4]

# 使用reduce函数计算列表中所有元素的和
from functools import reduce
sum_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)  # 输出: 15
```

在上述示例中，我们使用了map函数对列表中的每个元素进行平方操作，使用filter函数筛选出偶数，以及使用reduce函数计算列表中所有元素的和。这些函数都接受函数作为参数，并且没有副作用，符合函数式编程的思想。

2. 什么是lambda函数？请举例说明。

Lambda函数是一种匿名函数，它可以在需要函数对象的地方定义并使用，而无需使用def语句来定义一个常规函数。Lambda函数通常用于函数式编程中，特别是在需要传递简单函数作为参数的情况下。

Lambda函数的语法如下：

```
lambda arguments: expression
```

其中，arguments是函数的参数列表，expression是一个表达式，用于定义函数的返回值。

下面是一个简单的例子，展示了lambda函数的使用：

```
# 使用lambda函数对列表中的每个元素进行平方操作
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)  # 输出: [1, 4, 9, 16, 25]
```

```
# 使用lambda函数对列表中的字符串进行长度排序
strings = ["apple", "banana", "cherry", "date"]
sorted_strings = sorted(strings, key=lambda s: len(s))
print(sorted_strings)  # 输出: ['date', 'apple', 'cherry', 'banana']
```

在上面的例子中，我们首先使用lambda函数对列表中的每个元素进行平方操作。通过map函数结合lambda函数，我们可以方便地对列表中的所有元素进行操作。然后，我们使用lambda函数作为key参数传递给sorted函数，对字符串列表按照字符串长度进行排序。

Lambda函数的优点在于它们具有简洁的语法，可以在需要时快速定义和使用函数，而无需显式命名函数。然而，由于其匿名性，lambda函数通常用于表示简单的、一次性的操作，对于复杂的逻辑，通常建议使用常规的命名函数来提高代码的可读性和可维护性。

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。

高阶函数是指可以接收函数作为参数、或者返回函数作为结果的函数。在函数式编程中，高阶函数是一种重要的概念，它提供了灵活而强大的抽象能力。

以下是一些常用的高阶函数：

1、map(function, iterable)：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，其中包含函数应用后的结果。

代码示例：
```
# 使用map函数对列表中的每个元素进行平方操作
numbers = [1, 2, 3, 4, 5]
squared_numbers = list(map(lambda x: x**2, numbers))
print(squared_numbers)  # 输出: [1, 4, 9, 16, 25]
```

2、filter(function, iterable)：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，其中包含使函数返回True的元素。

代码示例：
```
# 使用filter函数筛选出列表中的偶数
numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # 输出: [2, 4]
```

3、reduce(function, sequence)：接受一个函数和一个序列作为参数，通过对序列中的元素依次应用函数来将序列归约为单个值。

代码示例：
```
# 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_of_numbers = reduce(lambda x, y: x + y, numbers)
print(sum_of_numbers)  # 输出: 15
```

4、sorted(iterable, key=function)：接受一个可迭代对象和一个可选的关键字函数作为参数，返回一个新的列表，其中包含可迭代对象中的元素按照关键字函数的规则排序后的结果。

代码示例：

```
# 使用sorted函数对字符串列表按照字符串长度进行排序
strings = ["apple", "banana", "cherry", "date"]
sorted_strings = sorted(strings, key=lambda s: len(s))
print(sorted_strings)  # 输出: ['date', 'apple', 'cherry', 'banana']
```

这些高阶函数的工作原理是通过接受函数作为参数，然后在函数内部对传入的函数进行处理或应用。通过这种方式，高阶函数可以灵活地适应不同的业务需求，提供了一种对函数进行组合和抽象的方式。

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。

完成实验6，我的收获如下：

- 这次实验学习了我了解了函数式编程范式以及一些高级函数的用法。
- 学会了如何使用lamda表达式，能借助lamda表达式简化代码。
- 做了codewars上面不同难度的题目，这对我熟悉python的基本语法很有帮助。