实验六 Python函数

班级：21计科04

学号：B20210201229

姓名：王雨晨

Github地址: Python

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：

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]
```

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
-

```
def count_developers(lst):   #第一题
    count = 0
    for dev in lst:
        if dev["continent"] == "Europe" and dev["language"] == "JavaScript":
            count += 1
    return count
```

```
def zero(operation=None): return 0 if operation is   None else operation(0)   #第二题
def one(operation=None): return 1 if operation is None else operation(1)
def two(operation=None): return 2 if operation is None else operation(2)
def three(operation=None): return 3 if operation is None else operation(3)
def four(operation=None): return 4 if operation is None else operation(4)
def five(operation=None): return 5 if operation is None else operation(5)
def six(operation=None): return 6 if operation is None else operation(6)
def seven(operation=None): return 7 if operation is None else operation(7)
def eight(operation=None): return 8 if operation is None else operation(8)
def nine(operation=None): return 9 if operation is None else operation(9)

def plus(y): return lambda x: x + y
def minus(y): return lambda x: x - y
def times(y): return lambda x: x * y
def divided_by(y): return lambda x: x // y  # 整数除法
```

```
def shorten_number(suffixes, base):  #第三题

    # 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
```

```
def find_senior(lst):

    # 利用生成器作为max函数的参数，找到最大的年龄    #第四题
    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]
```

```
def curry_partial(f, *args):      #第五题

    # 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 使用Mermaid绘制程序流程图

```
flowchart LR
    A[Start] --> B[for dev in lst]
    B --> C{来自欧洲的JavaScript开发人员？}
    C -->|YES|X[count += 1]
    X-->D{遍历结束？}
    D-->|NO|C
    C-->|NO|D
    D-->|YES|M[count]
    M-->F[End]
```

```
flowchart LR
    A[Start] --> B{如果输入的数据能转换为整数}
    B -->|NO|v
    B-->|YES|s[while循环]
    s-->t{base=0或者i<长度-1}
    t-->|YES|u[number//=base,i++]
    t-->|NO|B
    u-->v[字符串]
    v-->z[END]
```

```
flowchart LR
    A[Start] --> B[利用生成器作为max函数的参数，找到最大的年龄]
    B --> C[利用列表推导返回结果]
    C-->D[END]
```

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：

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：

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。

1. 什么是函数式编程范式？ 函数式编程范式是一种编程方法论，它强调将计算视为函数应用的过程，而不是通过改变状态和数据来实现计算。在函数式编程中，函数是不可变的，它们不会修改任何传递给它们的参数，也不会改变任何全局状态。函数式编程通过使用高阶函数、纯函数、不可变数据结构和递归来实现代码的组合和复用。函数式编程范式的优点包括代码的可读性、可维护性、可测试性和并行性。

2. 什么是lambda函数？请举例说明。 Lambda函数是一种匿名函数，它可以在一行代码中定义简单的函数。Lambda函数通常用于函数式编程范式中，用于在需要函数作为参数的情况下，快速定义一个简单的函数。

   Lambda函数的基本语法是：lambda 参数列表: 表达式

   例如，下面是一个使用lambda函数的简单例子：

   ```
   # 定义一个lambda函数，用于计算两个数的和
   add = lambda x, y: x + y
   # 调用lambda函数
   result = add(3, 5)
   print(result)  # 输出：8
   ```

   在这个例子中，我们使用lambda函数定义了一个简单的函数，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，并将结果打印出来。

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受一个或多个函数作为参数，并且/或者返回一个函数作为结果的函数。在函数式编程中，高阶函数是非常常见的，它们可以用于实现函数的组合、抽象和复用。

   常用的高阶函数包括map、filter、reduce等。

map函数：map函数接受一个函数和一个可迭代对象作为参数，将函数应用于可迭代对象中的每个元素，并返回一个包含结果的新可迭代对象。

```
# 使用map函数将列表中的每个元素加1
numbers = [1, 2, 3, 4, 5]
result = list(map(lambda x: x + 1, numbers))
print(result)  # 输出：[2, 3, 4, 5, 6]

filter函数：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，并返回一个包含使函数返回True的元素的新可迭代对象。
 # 使用filter函数过滤出列表中的偶数
numbers = [1, 2, 3, 4, 5, 6]
result = list(filter(lambda x: x % 2 == 0, numbers))
print(result)  # 输出：[2, 4, 6]

reduce函数：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依次应用函数，并返回一个单个结果。
from functools import reduce
 # 使用reduce函数计算列表中所有元素的和
numbers = [1, 2, 3, 4, 5]
result = reduce(lambda x, y: x + y, numbers)
print(result)  # 输出：15
```

这些高阶函数的工作原理是，它们接受一个函数作为参数，并对可迭代对象中的每个元素应用该函数，然后根据具体的功能返回一个新的结果。这种方式可以使代码更加简洁、可读，并且可以实现函数的复用和抽象。

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，我学习和使用了以下知识： 编程工具的使用：使用了Codewars平台进行编程练习和提交代码。 数据结构：学习了如何处理和操作Python中的字典和列表数据结构。 程序语言的语法：复习了Python语言的基本语法，包括函数定义、条件语句、循环等。 算法：解决了一些基本的算法问题，包括对列表进行筛选和处理。 编程技巧：学习了如何编写简洁、高效的代码，以及如何利用Python内置函数和方法来简化代码逻辑。 编程思想：学习了函数式编程的概念和实践，包括使用lambda函数和高阶函 数对数据进行处理和过滤。 总的来说，这次实验让我加深了对Python函数的基本用法的理解，同时也学习了一些函数式编程的技巧和思想。