实验六 Python函数

班级： 21计科2班

学号： B20210302211

姓名： 刘鑫

Github地址：https://github.com/leonidluo/python_course

CodeWars地址：https://www.codewars.com/users/Leonid712

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Mermaid的VSCode插件：

- Markdown Preview Mermaid Support
- Mermaid Markdown Syntax Highlighting

实验过程与结果

一.codewars做题

第一题：编码聚会1

源代码：

```
def count_developers(lst):

    european_js_devs = list(filter(lambda dev: dev['continent'] == 'Europe' and dev['language'] == 'JavaScript', lst))
    return len(european_js_devs)
```

第二题： 使用函数进行计算

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

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

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

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

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

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

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


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


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


def nine(func=None):
    return func(9) if func else 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(Number Shortening Filter)

```
def shorten_number(suffixes, base):
    def filter_func(num):
            if not isinstance(num, str):
                return str(num)
            if not num.isdigit():
                return str(num)
            num = int(num)
            for suffix in suffixes:
                if num > base:
                    num /= base
```

```
                else:
                    return f"{int(num)}{suffix}"
            return f"{int(num*base)}{suffix}"
    return filter_func
```

第四题： 编码聚会7

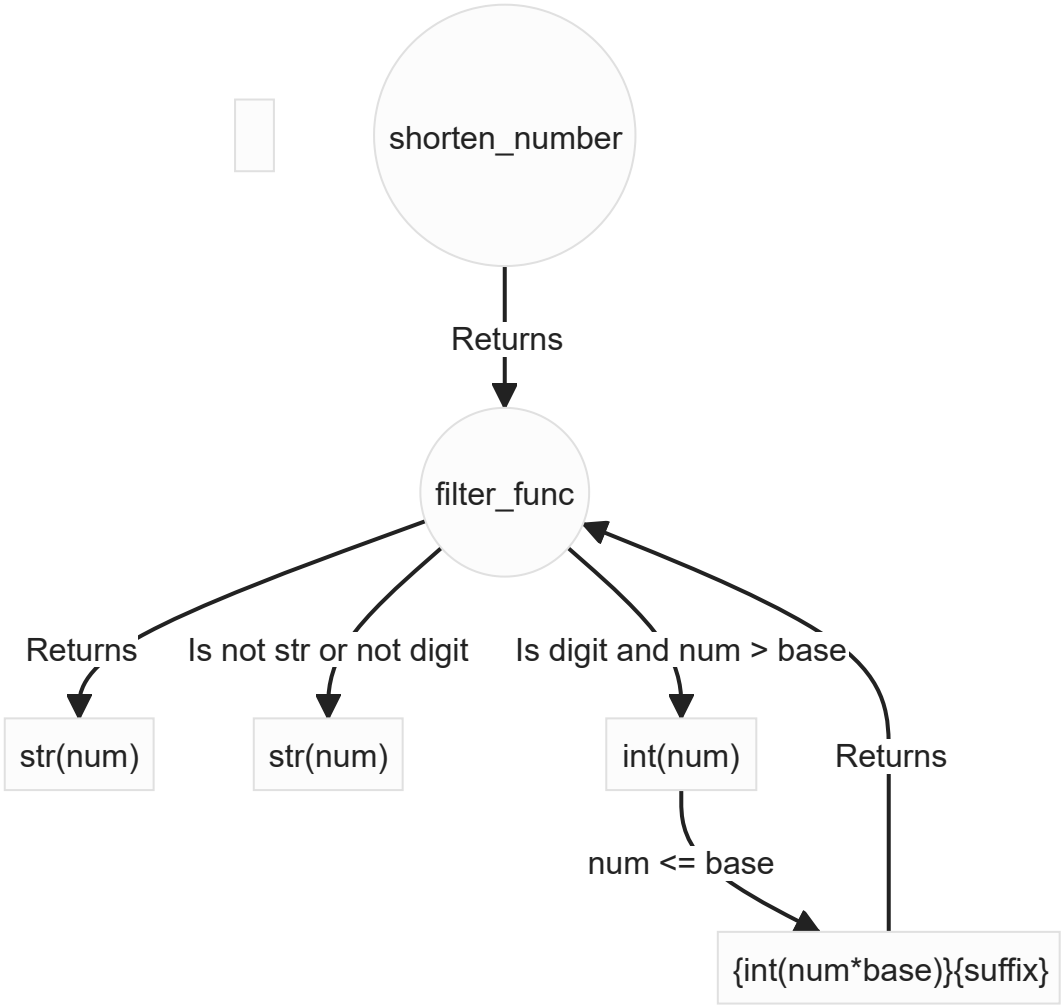
```
def oldest_developers(lst):
    max_age = max(lst, key=lambda dev: dev['age'])['age']
    oldest_devs = [dev for dev in lst if dev['age'] == max_age]
    return oldest_devs
```

第五题： Currying versus partial application

> 暂时未写

二.mermaid流程图

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



实验考查

1. **函数式编程范式：** 函数式编程是一种编程范式，它强调函数的纯粹性和避免副作用，将计算看作数学函数的求值。
2. **Lambda函数：** Lambda函数是匿名函数，使用 `lambda` 关键字定义，通常用于需要简短函数的场合。

```python
add = lambda x, y: x + y
print(add(2, 3))  # 输出: 5
```

3. **高阶函数：** 高阶函数接受和/或返回一个或多个函数。常见的高阶函数有 `map`、`filter`、`reduce`。

```python
# map函数
squared = list(map(lambda x: x**2, [1, 2, 3, 4]))
# filter函数
even_numbers = list(filter(lambda x: x % 2 == 0, [1, 2, 3, 4]))
# reduce函数
from functools import reduce
```

```
product = reduce(lambda x, y: x * y, [1, 2, 3, 4])
```

`map` 对集合中的每个元素应用函数，`filter` 根据条件过滤元素，`reduce` 对集合中的元素进行累积操作。

实验总结

1. **编程工具的使用**:

- 我使用了Python编程语言来解决各种编程问题。

- 了解了Python中的各种内置函数和模块，如字符串操作、集合操作等。

2. **数据结构**:

- 掌握了不同数据结构的特点和用途，包括列表（List）、集合（Set）、字典（Dictionary）等。

- 学习了如何使用这些数据结构来组织和操作数据。

3. **程序语言的语法**:

- 深入了解了Python语言的语法规则，包括变量、数据类型、条件语句、循环语句等。

- 学会了如何编写清晰和可读的代码。

4. **算法**:

- 解决了各种算法问题，包括字符串处理、数据验证、数独验证等。

- 学到了如何设计和实现算法来解决复杂问题。

5. **编程思想**:

- 强调了编程思想的重要性，如问题分解、模块化、重用性等。

- 学到了如何以更有效和高效的方式解决问题。