实验六 Python函数

班级： 21计科1

学号： B20210302104

姓名： 金皓翔

Github地址：https://github.com/jhx666oo/python

CodeWars地址：https://www.codewars.com/users/jhx666oo

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

代码：

```
def count_developers(lst):
    count=0
    for list1 in lst:
        if list1["continent"]=="Europe" and list1["language"]=="JavaScript":
            count+=1
    return count
```

思路： 初始化计数器，遍历列表，如果列表有满足条件"continent" 键的值为 "Europe"，"language" 键的值为 "JavaScript"，将计数器 count 的值增加 1。遍历完成后，返回计数器 count 的值，即满足条件的开发者数量。

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

代码：

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

思路： 利用高阶函数和闭包在 Python 中模拟基本的算术运算，将数字和运算操作都表示为函数，实现以函数调用的形式表达和计算算术表达式。

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

代码：

```
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
```

思路：这道题目使用高阶函数创建另一函数，用于将数字字符串转换为基于给定基数和后缀列表的简短格式。

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

代码：

```
def find_senior(lst):
   mage = max(a['age'] for a in lst)
   return [a for a in lst if a['age']==mage]
```

思路： 利用生成器作为max函数的参数，找到最大的年龄，利用列表推导返回结果。

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

代码：

```
from inspect import signature
from functools import partial

def curry_partial(main_func, *args):

    if not(callable(main_func)):
        return main_func

    p = len(signature(main_func).parameters)
    func = partial(main_func)

    for a in args:
        if len(func.args) == p: break
        func = partial(func, a)

    if len(func.args) < p:
        return partial(curry_partial, main_func, *func.args)

    return func()
```

思路： 这个函数的目的是将一个可调用函数进行柯里化和偏函数化，允许逐步传递参数并最终执行该函数。如果传递的参数数量不足，它会返回一个新的偏函数，等待更多参数的传递。如果传递的参数超过函数所需的数量，多余的参数会被忽略。这个函数的核心思想是参数逐步传递和延迟执行。

第三部分

使用Mermaid绘制程序流程图

```
graph TD
    A[开始] --> B[初始化计数器 count 为 0]
    B --> C[遍历列表 lst 中的每个元素]
    C --> D{检查元素的大陆是否为 'Europe'}
    D -->|是| E{检查元素的语言是否为 'JavaScript'}
    D -->|否| C
    E -->|是| F[计数器 count 加 1]
    E -->|否| C
    F --> C
    C --> G[完成遍历]
    G --> H[返回计数器 count]
    H --> I[结束]
```

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：

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?
2. 什么是lambda函数？请举例说明。
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。

1. 什么是函数式编程范式?

函数式编程（Functional Programming, FP）是一种编程范式，它将计算视为数学函数的评估，并避免使用程序状态和可变数据。在函数式编程中，函数是一等公民，这意味着它们可以作为参数传递给其他函数，也可以作为值返回。函数式编程强调纯函数（无副作用的函数）和不可变性（数据不会被修改）。

2. 什么是lambda函数？请举例说明。

Lambda 函数，也称为匿名函数，是一种在 Python 中定义小型、一次性、没有名称的函数的方法。Lambda 函数可以接受任意数量的参数，但只能有一个表达式。它们通常用于需要函数对象的地方，但仅用一次或非常简短。

示例：

```
# 使用 lambda 函数计算两数之和
add = lambda x, y: x + y
print(add(5, 3))  # 输出: 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。

高阶函数是指至少满足下列一个条件的函数：

- 接受一个或多个函数作为参数。
- 返回一个函数作为结果。

在 Python 中，常见的高阶函数包括 `map()`, `filter()`, 和 `reduce()`。

- **map()**：对序列的每个元素应用一个给定的函数，并返回一个 map 对象（可迭代）。

```
# 使用 map() 函数将每个数字平方
numbers = [1, 2, 3, 4, 5]
squares = map(lambda x: x**2, numbers)
print(list(squares))  # 输出: [1, 4, 9, 16, 25]
```

- **filter()**：使用一个函数来测试序列中的每个元素，返回一个包含所有通过测试的元素的迭代器。

```
# 使用 filter() 函数过滤出大于 2 的数字
numbers = [1, 2, 3, 4, 5]
filtered = filter(lambda x: x > 2, numbers)
print(list(filtered))  # 输出: [3, 4, 5]
```

- **reduce()**（需从 `functools` 导入）：对序列中的元素进行累积操作（如求和、求积）。

```
from functools import reduce
# 使用 reduce() 函数计算所有数字的乘积
numbers = [1, 2, 3, 4, 5]
product = reduce(lambda x, y: x * y, numbers)
print(product)  # 输出: 120
```

实验总结

- 通过这次实验，我更加了解了python函数方面的知识，也接触了到了lambda函数和高阶函数的使用，并熟悉函数式编程的概念完成了实践。