实验六 Python函数

班级： 21计科3班

学号： B20210302310

姓名： 姚义香

Github地址：https://github.com/blmeue/Python_resources.git

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

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
  { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent': 'Oceania', 'age': 2
  { 'firstName': 'Shufen', 'lastName': 'L.', 'country': 'Taiwan', 'continent': 'Asia', 'age': 35
  { 'firstName': 'Sumayah', 'lastName': 'M.', 'country': 'Tajikistan', 'continent': 'Asia', 'age
]
```

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':
  { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asia', 'age': 3
  { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'Europe', 'ag
  { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'l
]
```

您的程序应该返回如下结果：

```
[
  { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age':
  { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'l
]
```

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

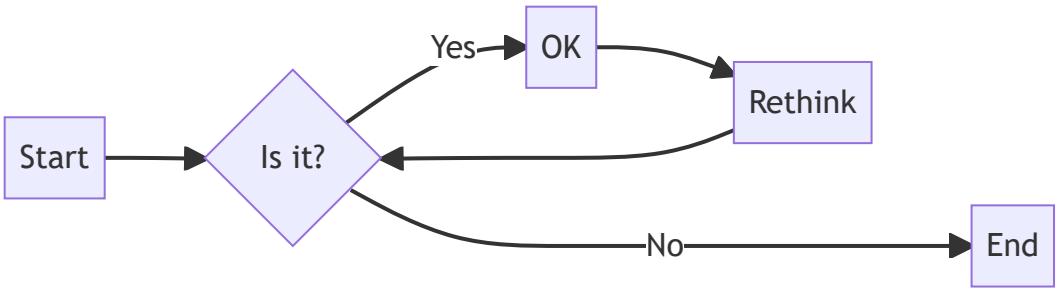
安装VSCode插件：

- 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

1. 第一题：编码聚会1

（1）实验代码：

```
def count_developers(lst):
 # Your code here
 count=0
 for ls in lst:
     if ls['continent']=='Europe' and ls['language']=='JavaScript':
         count+=1
 return count
 #测试用例
 list1 = [
     { 'firstName': 'Noah', 'lastName': 'M.', 'country': 'Switzerland', 'continent': 'Europe
     { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent': 'Oceania',
     { 'firstName': 'Shufen', 'lastName': 'L.', 'country': 'Taiwan', 'continent': 'Asia', 'a
     { 'firstName': 'Sumayah', 'lastName': 'M.', 'country': 'Tajikistan', 'continent': 'Asia
     ]

 print(count_developers(list1))
```

(2) 实验结果：

1

2. 第二题： 使用函数进行计算

（1）实验代码：

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

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

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

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

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

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

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

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

def eight(func=None):
   if func is None:
       return 8
```

```
    else:
        return func(8)

def nine(func=None):
    if func is None:
        return 9
    else:
        return func(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
print(seven(times(five())))
print(four(plus(nine())))
```

（2）实验结果：

35

13

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

（1）实验代码：

```
def shorten_number(suffixes, base):
 def my_filter(number):
     if type(number) != str:
         return str(number)
     else:
         if number.isdigit():
             numb = int(number)
             i=0
             while numb//base>0 and i<len(suffixes)-1:
                 numb=numb//base
                 i=i+1
             return str(numb)+suffixes[i]
         else:
             return number
 return my_filter
 #测试用例
filter1 = shorten_number(['','k','m'],1000)
print(filter1('234234'))
print(filter1('98234324'))
print(filter1([1,2,3]))
print(filter1(1))
```

（2）实验结果

234k

98m

[1, 2, 3]

1

1

4. 第四题： 编码聚会7

（1）实验代码：

```
def find_senior(lst):
 # your code here
 max_age = max(dev['age'] for dev in lst)
 ret=[]
 for ls in lst:
     if ls['age']==max_age:
         ret.append(ls)
 return ret
 list1 = [
         { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Eur
         { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asi
         { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'E
         { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'a
         ]
print(find_senior(list1))
```

（2）实验结果：

[{'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Europe', 'age': 49, 'language': 'PHP'}, {'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'age': 49, 'language': 'PHP'}]

5. 第五题： Currying versus partial application

(1) 实验代码：

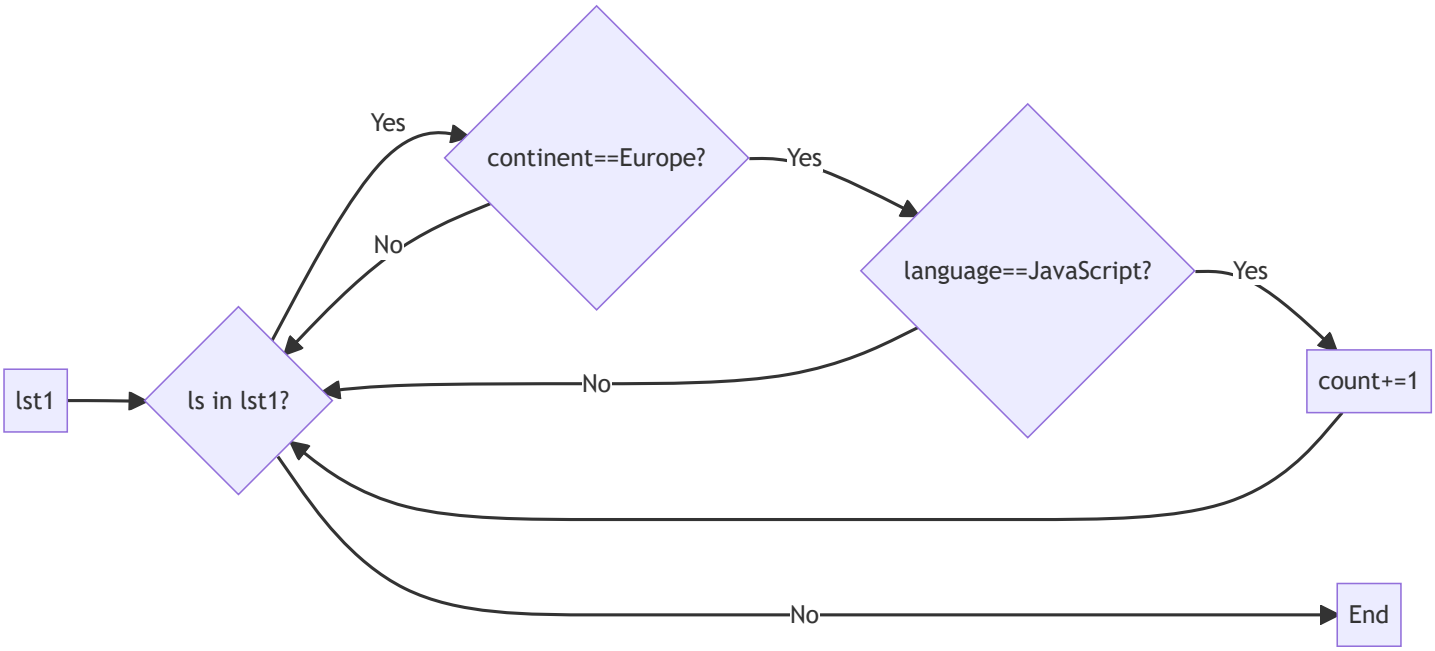
```
import types
def curry_partial(f,*initial_args):
   """ Curries and partially applies the initial arguments to the function """
   if type(f) != types.FunctionType:#判断第一个参数是否为函数
       return f
       # 查看函数f需要的参数个数
   num_args = f.__code__.co_argcount
   # 如果f函数不需要参数，说明f是curry_partial函数
   if num_args == 0:
       return f(*initial_args)

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

   def my_ini(*new_args):
       all_args = initial_args + new_args
       return curry_partial(f, *all_args)

   return my_ini

# 定义一个三个数的加法
add = lambda a,b,c: a+b+c
#测试用例
curriedAdd = curry_partial(add)
print(curriedAdd(1)(2)(3))  # => 6

partialAdd = curry_partial(add, 1)
print(partialAdd(2, 3))  # => 6
```

（2）实验结果：

6

6

- 第三部分 使用Mermaid绘制程序流程图



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?
   答：函数式编程范式是一种编程思想，它将计算过程视为一系列数学函数的求值。在函数式编程中，函数是一等公民，即函数可以作为参数传递给其他函数，也可以作为其他函数的返回值。函数式编程的核心思想是通过组合纯函数（没有副作用的函数）来构建程序。它强调将计算视为函数求值的过程，避免了状态和可变数据的使用，使程序更易于理解、维护和测试。
2. 什么是lambda函数？请举例说明。
   答：Lambda函数是一种匿名函数，也就是没有名字的函数。它们主要用于短小的函数定义，而无需使用def关键字显式地创建一个函数。Lambda函数以lambda关键字开头，后面跟着参数列表和冒号，然后是表达式。
   例如：

```
f = lambda x: x ** 2
print(f(5))  # 输出: 25
```

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。
   答：高阶函数是接受一个或多个函数作为输入（参数）或返回一个函数作为结果的函数。换句话

说，函数可以接受函数作为输入，或者返回一个函数作为结果，或者两者都做。在编程中，有很多常见的高阶函数，如map(), filter(), reduce(), sorted()。
例如：

```
numbers = [1, 2, 3, 4, 5]
squared = map(lambda x: x ** 2, numbers)
print(list(squared))  # 输出：[1, 4, 9, 16, 25]
numbers = [1, 2, 3, 4, 5]
even = filter(lambda x: x % 2 == 0, numbers)
print(list(even))  # 输出：[2, 4]
```

实验总结

在本次实验过程中，逐渐掌握了函数的基本原理和使用方法，同时对函数式编程有了初步的了解。在本次实验中，我使用Python语言实现了函数式编程的基本概念，包括高阶函数、函数组合等。通过本次实验，我对函数式编程有了更深的理解，同时对Python语言的函数式编程有了更深刻的认识。