实验六 Python函数

班级： 21计科1

学号： B20210302110

姓名： 刘湘怡

Github地址：https://github.com/righting1/python_Experiments

CodeWars地址：https://www.codewars.com/users/righting1

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：

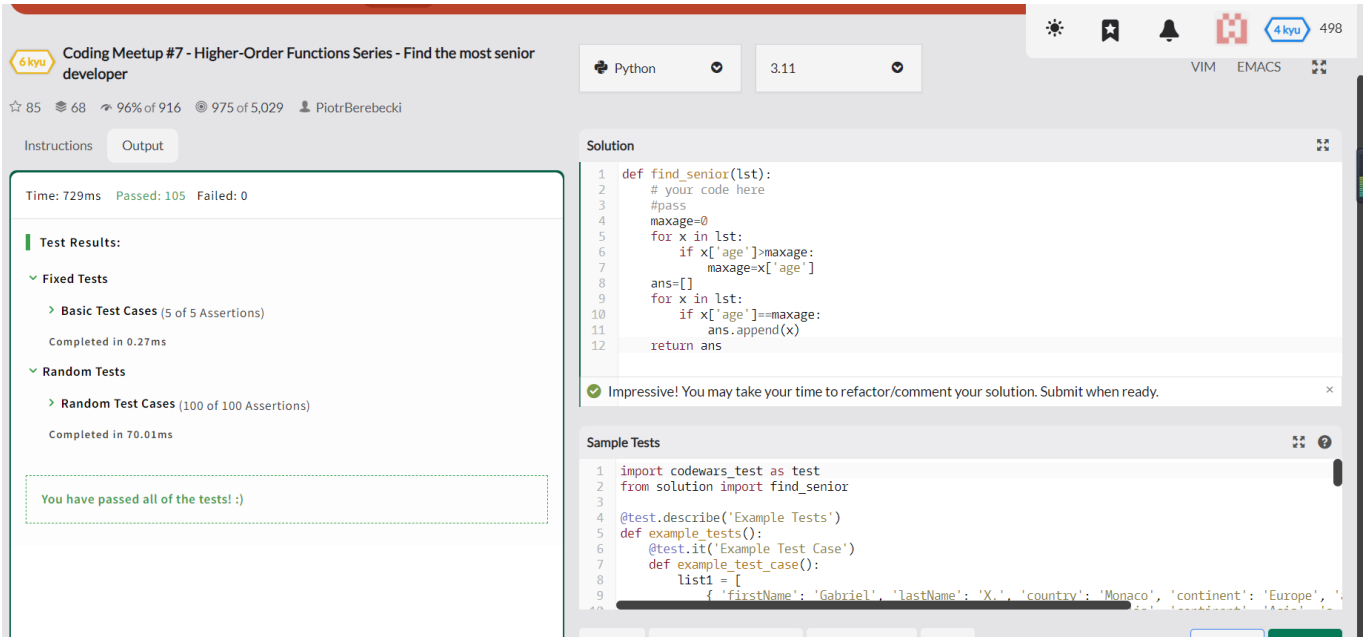
```
lst1 = [
  { 'firstName': 'Noah', 'lastName': 'M.', 'country': 'Switzerland', 'continent':
'Europe', 'age': 19, 'language': 'JavaScript' },
  { 'firstName': 'Maia', 'lastName': 'S.', 'country': 'Tahiti', 'continent':
'Oceania', 'age': 28, 'language': 'JavaScript' },
  { 'firstName': 'Shufen', 'lastName': '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




```
def count_developers(lst):
    # Your code here
    ans=0
    for x in lst:
        if x['continent']=='Europe' and x['language']=='JavaScript':
            ans+=1
    return ans
```

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
```

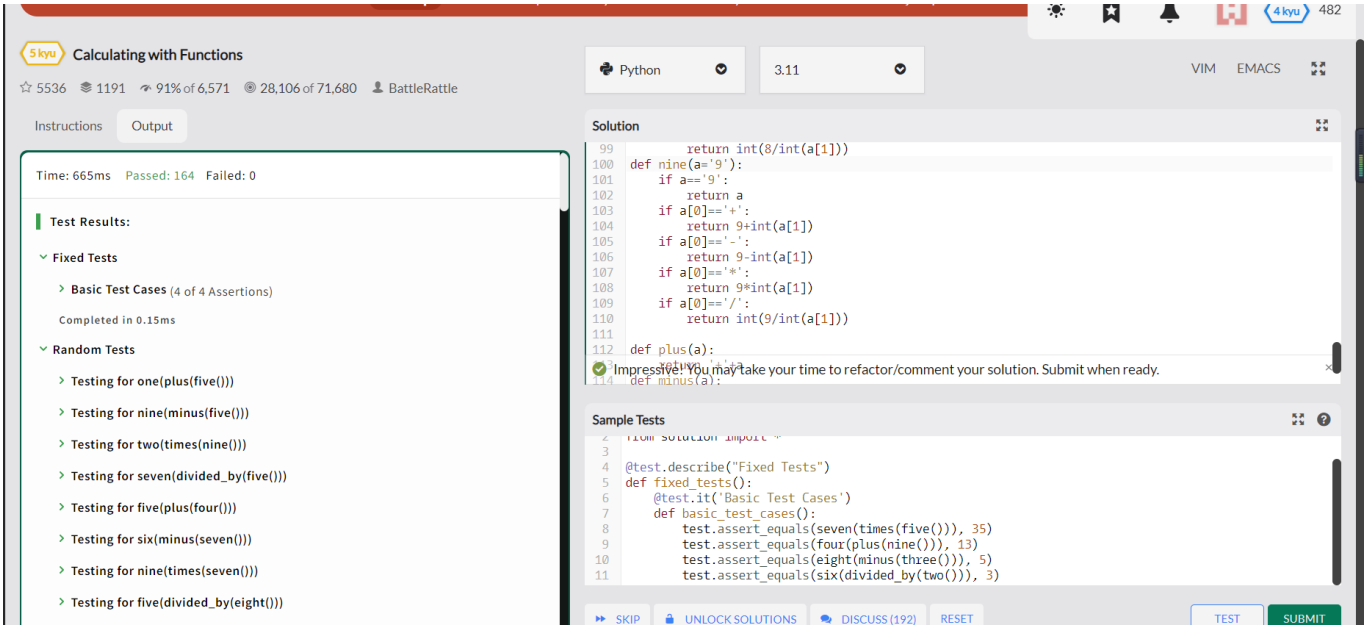
要求：

- 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




```
def zero(a='0'):
    if a=='0':
        return a
    if a[0]=='+':
        return 0+int(a[1])
    if a[0]=='-':
        return 0-int(a[1])
    if a[0]=='*':
```

```
            return 0
        if a[0]=='/':
            return 0
    def one(a='1'):
        if a=='1':
            return a
        if a[0]=='+':
            return 1+int(a[1])
        if a[0]=='-':
            return 1-int(a[1])
        if a[0]=='*':
            return 1*int(a[1])
        if a[0]=='/':
            return int(1/int(a[1]))
    def two(a='2'):
        if a=='2':
            return a
        if a[0]=='+':
            return 2+int(a[1])
        if a[0]=='-':
            return 2-int(a[1])
        if a[0]=='*':
            return 2*int(a[1])
        if a[0]=='/':
            return int(2/int(a[1]))
    def three(a='3'):
        if a=='3':
            return a
        if a[0]=='+':
            return 3+int(a[1])
        if a[0]=='-':
            return 3-int(a[1])
        if a[0]=='*':
            return 3*int(a[1])
        if a[0]=='/':
            return int(3/int(a[1]))
    def four(a='4'):
        if a=='4':
            return a
        if a[0]=='+':
            return 4+int(a[1])
        if a[0]=='-':
            return 4-int(a[1])
        if a[0]=='*':
            return 4*int(a[1])
        if a[0]=='/':
            return int(4/int(a[1]))
    def five(a='5'):
        if a=='5':
            return a
        if a[0]=='+':
            return 5+int(a[1])
        if a[0]=='-':
            return 5-int(a[1])
```

```
    if a[0]=='*':
        return 5*int(a[1])
    if a[0]=='/':
        return int(5/int(a[1]))
def six(a='6'):
    if a=='6':
        return a
    if a[0]=='+':
        return 6+int(a[1])
    if a[0]=='-':
        return 6-int(a[1])
    if a[0]=='*':
        return 6*int(a[1])
    if a[0]=='/':
        return int(6/int(a[1]))
def seven(a='7'):
    if a=='7':
        return a
    if a[0]=='+':
        return 7+int(a[1])
    if a[0]=='-':
        return 7-int(a[1])
    if a[0]=='*':
        return 7*int(a[1])
    if a[0]=='/':
        return int(7/int(a[1]))
def eight(a='8'):
    if a=='8':
        return a
    if a[0]=='+':
        return 8+int(a[1])
    if a[0]=='-':
        return 8-int(a[1])
    if a[0]=='*':
        return 8*int(a[1])
    if a[0]=='/':
        return int(8/int(a[1]))
def nine(a='9'):
    if a=='9':
        return a
    if a[0]=='+':
        return 9+int(a[1])
    if a[0]=='-':
        return 9-int(a[1])
    if a[0]=='*':
        return 9*int(a[1])
    if a[0]=='/':
        return int(9/int(a[1]))

def plus(a):
    return '+'+a
def minus(a):
    return '-'+a
def times(a):
```

```
    return '*'+a
def divided_by(a):
    return '/'+a
```

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

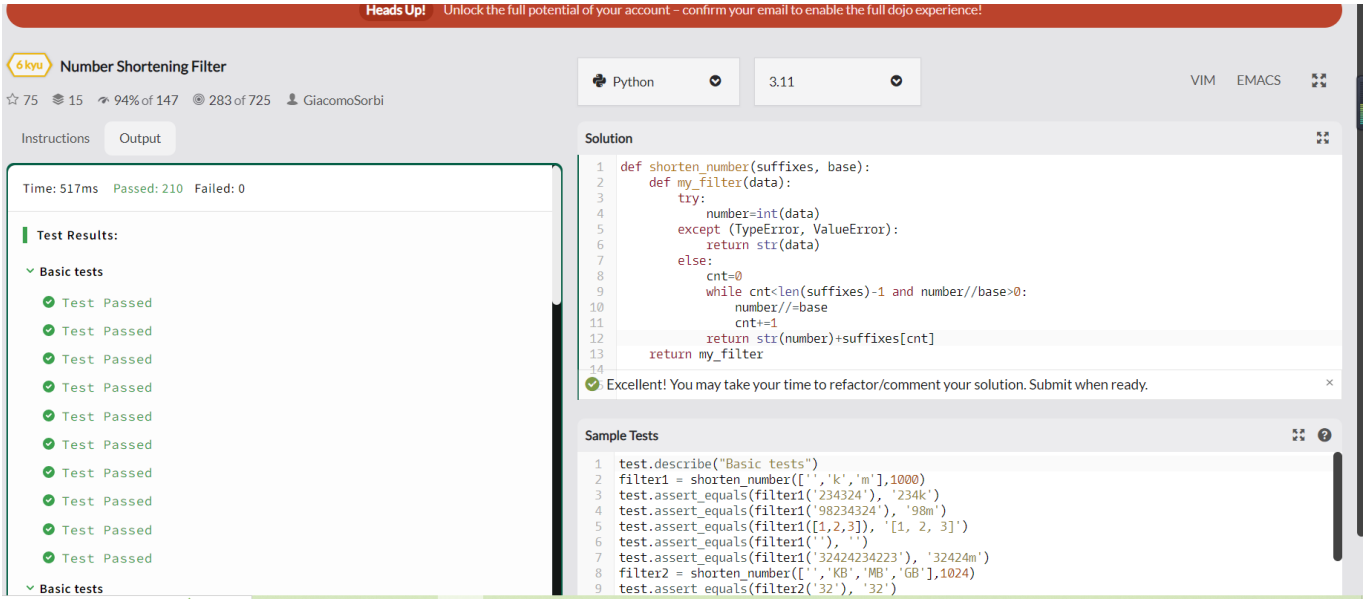
难度：6kyu

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




```
def shorten_number(suffixes, base):
    def my_filter(data):
        try:
            number=int(data)
        except (TypeError, ValueError):
            return str(data)
        else:
```

```
            cnt=0
            while cnt<len(suffixes)-1 and number//base>0:
                number//=base
                cnt+=1
            return str(number)+suffixes[cnt]
    return my_filter
```

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' },
]
```

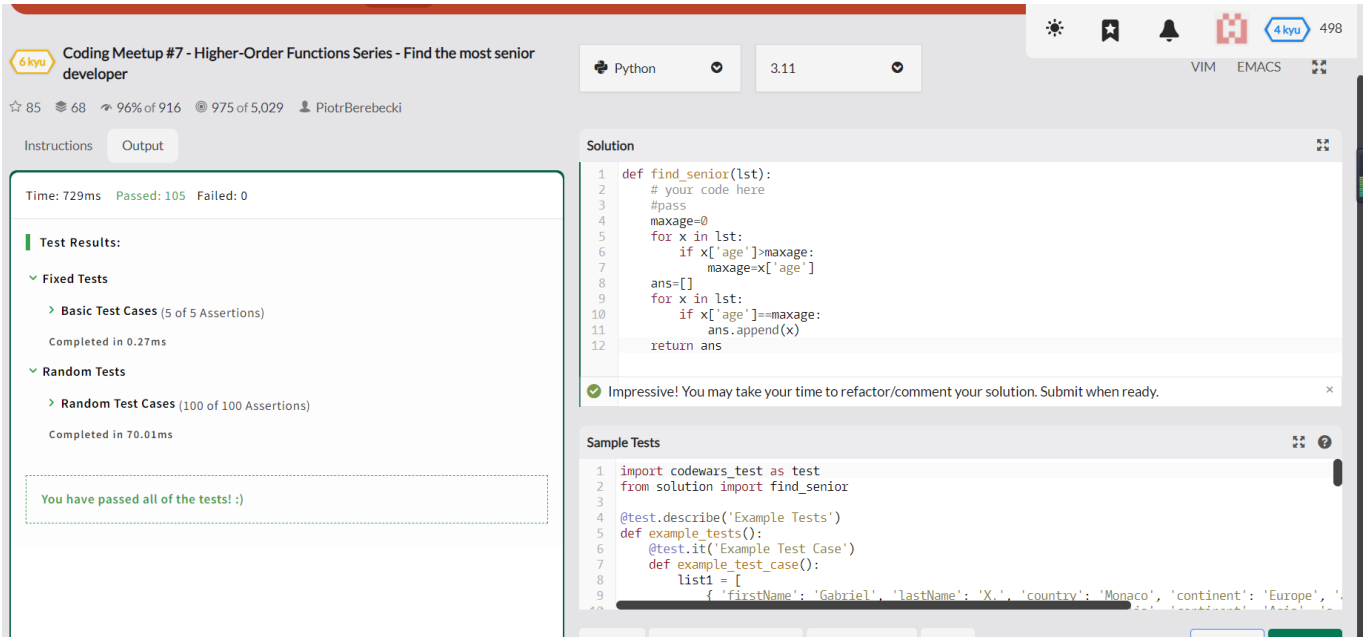
您的程序应该返回如下结果：

```
[
  { '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




```python
def find_senior(lst):
    # your code here
    #pass
    maxage=0
    for x in lst:
        if x['age']>maxage:
            maxage=x['age']
    ans=[]
    for x in lst:
        if x['age']==maxage:
            ans.append(x)
    return ans
```

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。

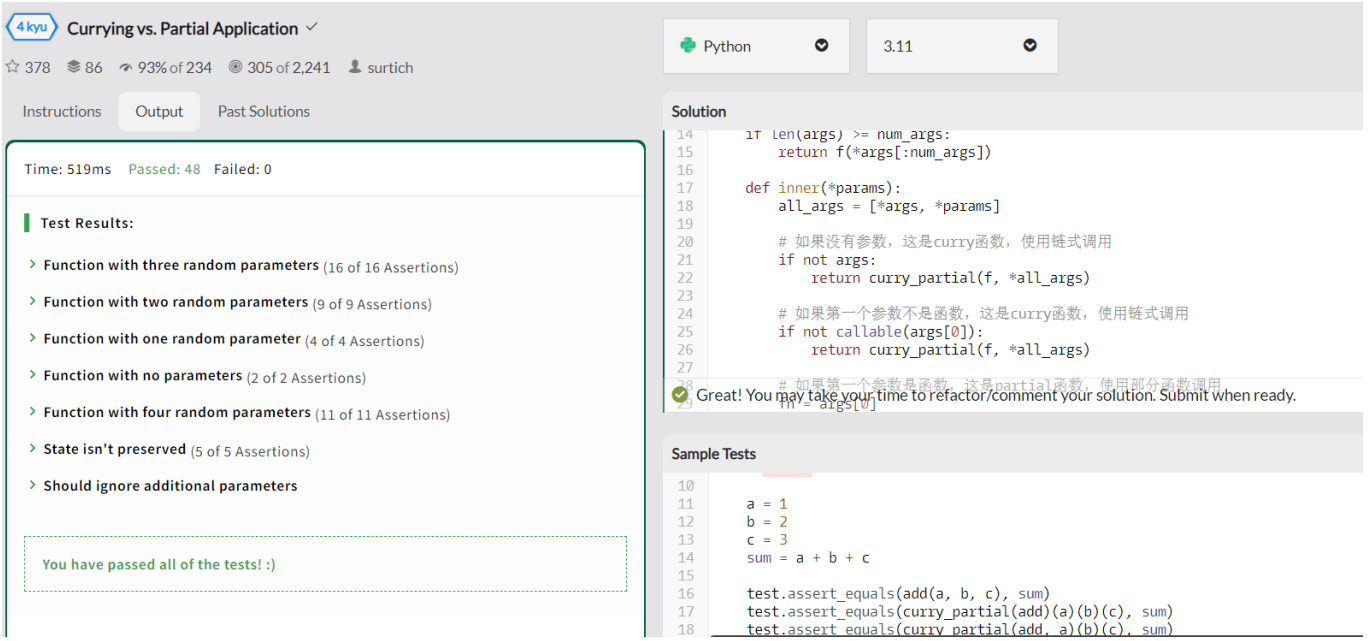
所有下面这些例子都应该产生相同的结果：

```
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




```python
def curry_partial(f,*args):
     #""" Curries and partially applies the initial arguments to the function """
     # 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
```

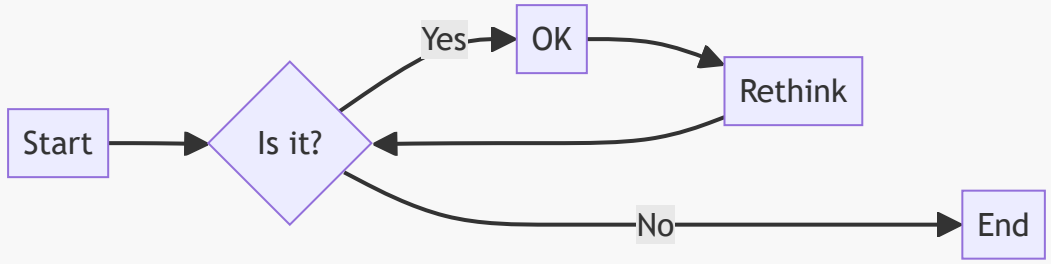
第三部分

使用Mermaid绘制程序流程图

安装VSCode插件：

- Markdown Preview Mermaid Support
- Mermaid Markdown Syntax Highlighting

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：

```
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：

```
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（Functional Programming，简称 FP）是一种编程范式，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，这意味着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。 此外，函数式编程是一种声明式编程范式，编程是用表达式或声明而不是语句来完成的。它更加强调程序执行的结果而非执行的过程，倡导利用若干简单的执行单元让计算结果不断渐进，逐层推导复杂的运算，而不是设计一个复杂的执行过程。
2. 什么是lambda函数？请举例说明。 Lambda函数是一种匿名函数，也就是没有名字的函数。它们常常在需要短小函数的地方使用，特别是在需要使用递归或者需要生成动态函数的地方。Lambda函数是Python中唯一一种没有名称的函数形式。 下面是一个简单的lambda函数的例子：

```
# 这是一个lambda函数，它接受两个参数，并返回它们的和
add = lambda x, y: x + y

# 使用这个函数
result = add(5, 3)
print(result)  # 输出: 8
```

在这个例子中，`add`是一个lambda函数，它接受两个参数`x`和`y`，并返回它们的和。然后我们调用这个函数，传入两个数字（5和3），并打印结果。 需要注意的是，lambda函数只能包含一个表达式，它们不能包含复杂的逻辑或多个语句。如果需要编写更复杂的函数逻辑，应该使用常规的def语句来定义函数。 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。这些函数通常将一些操作封装在一个黑匣子中，可以用于各种不同的任务。高阶函数是函数式编程的主要组成部分，为开发人员提供了编写模块化、可重用代码的强大工具。

以下是一些常见的高阶函数：

map()：它接受一个函数和一个或多个可迭代对象作为输入，然后返回一个将输入函数应用于可迭代对象的每个元素的迭代器。

```
def square(x):
    return x * x

numbers = [1, 2, 3, 4, 5]
```

```
squared = map(square, numbers)
print(list(squared))  # 输出: [1, 4, 9, 16, 25]
```

filter()：它接受一个函数和一个可迭代对象作为输入，然后返回一个迭代器，其中包含将输入函数返回True的元素。

```
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4, 5]
even_numbers = filter(is_even, numbers)
print(list(even_numbers))  # 输出: [2, 4]
```

reduce()：它接受一个函数和一个可迭代对象作为输入，然后使用该函数将可迭代对象中的元素组合成一个单一的结果。例如，可以使用reduce()计算列表中所有元素的乘积。

```
from functools import reduce

def multiply(x, y):
    return x * y

numbers = [1, 2, 3, 4, 5]
product = reduce(multiply, numbers)
print(product)  # 输出: 120
```

sorted()：它接受一个可迭代对象作为输入，并返回一个新的排序列表。可以传递一个可选的key参数，该参数是一个函数，用于指定如何对元素进行排序。

```
numbers = [5, 3, 1, 4, 2]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # 输出: [1, 2, 3, 4, 5]
```

这些高阶函数的工作方式是通过将函数作为参数传递给其他函数，从而扩展了这些函数的用途。这使得开发人员可以使用更少的代码来完成更多的任务，并且代码更加模块化和可重用。

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。 这次实验让我更深入地理解了编程的各个方面，包括工具的使用、数据结构、语法、算法、技巧和思想等。这些知识和技能将对我未来的编程工作产生积极的影响。