实验六 Python函数

班级： 21计科4

学号： B20210404205

姓名： 康佳程

Github地址：https://github.com/ktxiaok/python_experiments_2023.git

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

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

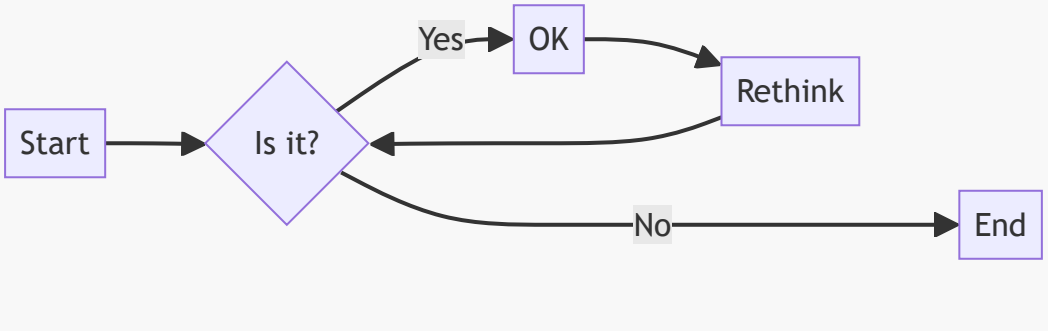
使用Mermaid绘制程序流程图

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）

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

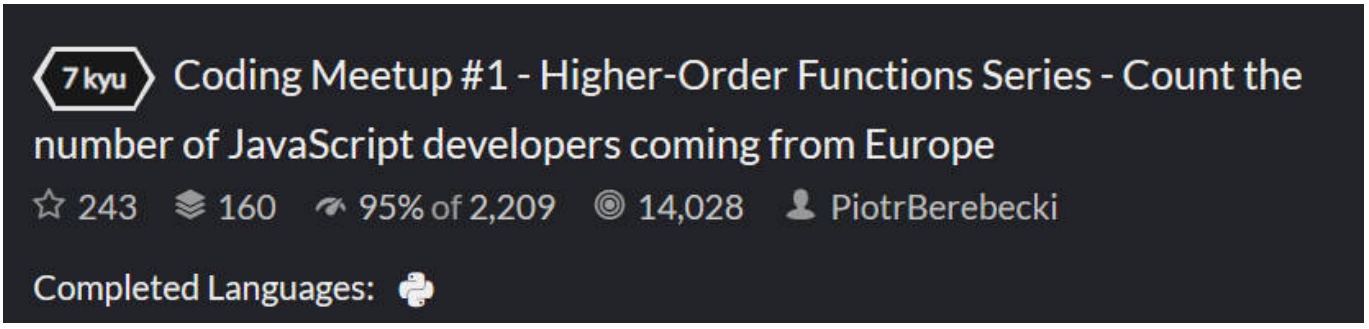
请将实验过程与结果放在这里，包括：

- 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

Codewars Kata挑战

第一题：编码聚会1

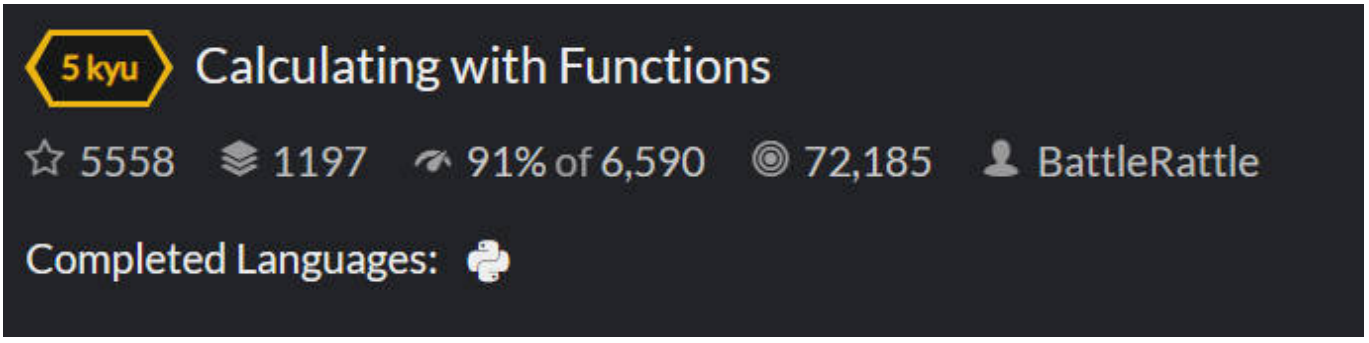
```
def count_developers(lst):
    from functools import reduce
    return reduce(lambda count, item: count + 1 if item['language'] ==
'JavaScript' and item['continent'] == 'Europe' else count, lst, 0)
```



第二题： 使用函数进行计算

```
OP_PLUS = 0
OP_MINUS = 1
OP_TIMES = 2
OP_DIV = 3

class Operation:
    __op_type = None
    __right = None

    def __init__(self, op_type, right):
        self.__op_type = op_type
        self.__right = right

    def calc(self, left):
        if self.__op_type == OP_PLUS:
            return left + self.__right
        if self.__op_type == OP_MINUS:
            return left - self.__right
```

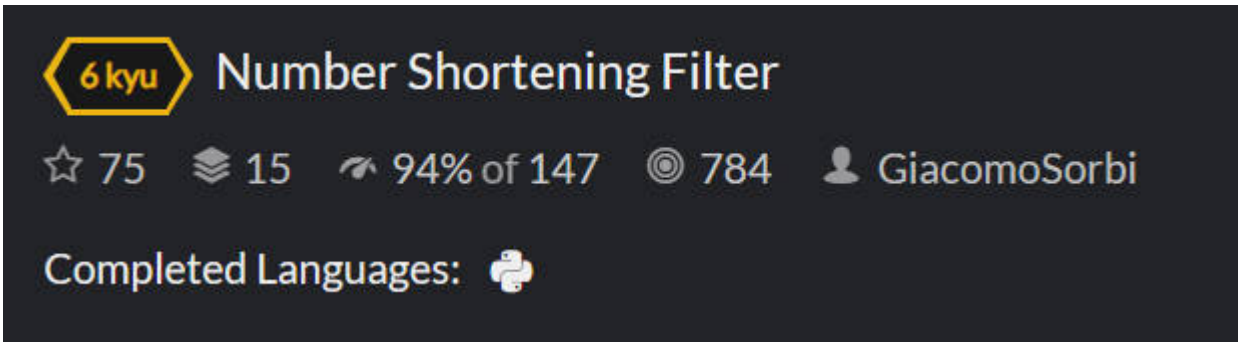
```
        if self.__op_type == OP_TIMES:
            return left *  self.__right
        if self.__op_type == OP_DIV:
            return left // self.__right
        raise "invalid operation type!"

def number(num, operation = None): return num if operation == None else
operation.calc(num)

def zero(operation = None): return number(0, operation)
def one(operation = None): return number(1, operation)
def two(operation = None): return number(2, operation)
def three(operation = None): return number(3, operation)
def four(operation = None): return number(4, operation)
def five(operation = None): return number(5, operation)
def six(operation = None): return number(6, operation)
def seven(operation = None): return number(7, operation)
def eight(operation = None): return number(8, operation)
def nine(operation = None): return number(9, operation)

def plus(right): return Operation(OP_PLUS, right)
def minus(right): return Operation(OP_MINUS, right)
def times(right): return Operation(OP_TIMES, right)
def divided_by(right): return Operation(OP_DIV, right)
```



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

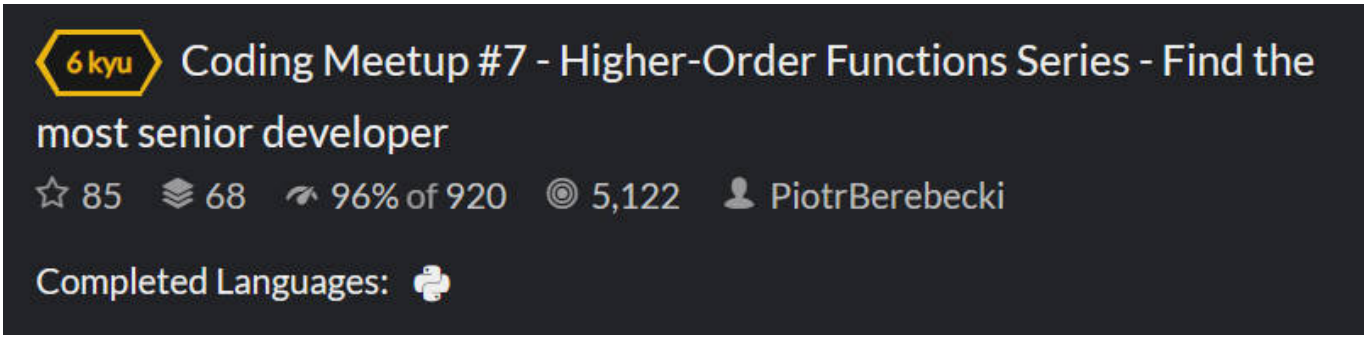
```
def shorten_number(suffixes, base):
    def shortening_filter(num_str):
        if not isinstance(num_str, str):
            return str(num_str)
        try:
            num = int(num_str)
        except ValueError:
            return num_str

        suffix_idx = 0
        max_suffix_idx = len(suffixes) - 1
        while True:
            next_num = num // base
            if next_num == 0:
```

```
                break
            suffix_idx += 1
            num = next_num
            if suffix_idx == max_suffix_idx:
                break

        return str(num) + suffixes[suffix_idx]
    return shortening_filter
```



第四题： 编码聚会7

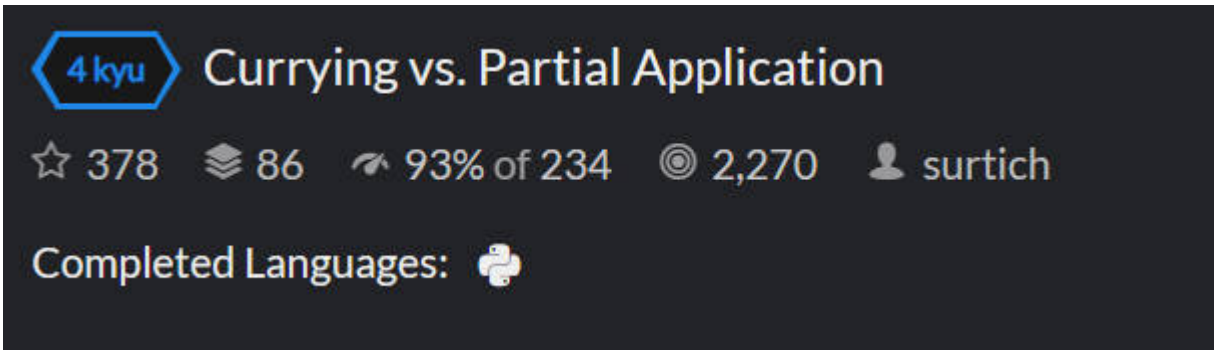
```
def find_senior(lst):
    max_age = max(map(lambda item: item['age'], lst))
    return list(filter(lambda item: item['age'] == max_age, lst))
```



第五题： Currying versus partial application

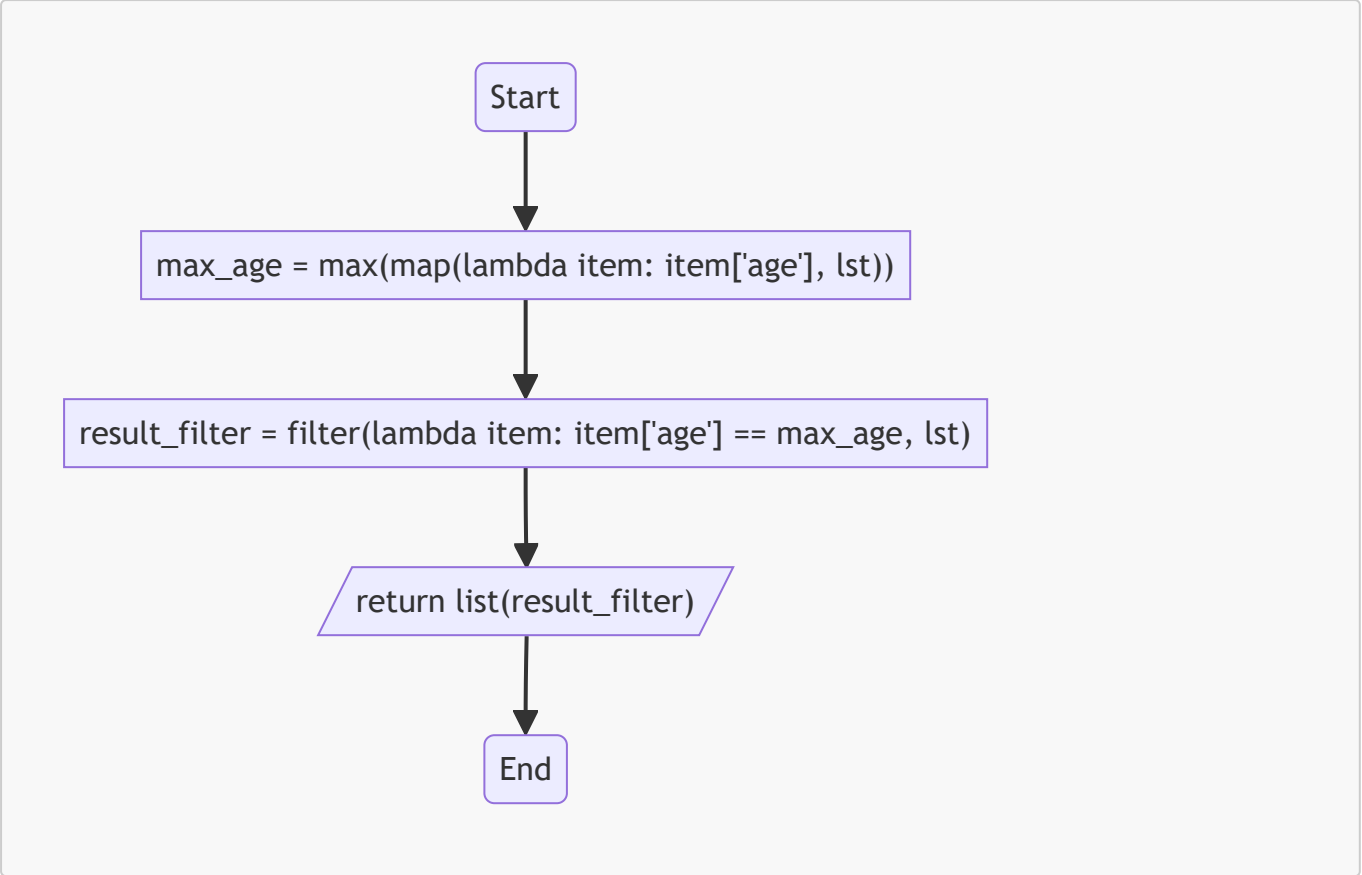
```
class ArgRecorder:
    __func = None
    __argcount = None
    __args = None

    def __init__(self, func):
        if func == None: return
        self.__func = func
        self.__argcount = func.real_argcount if hasattr(func, 'real_argcount')
else func.__code__.co_argcount
        self.__args = []
```

```
    def clone(self):
        instance = ArgRecorder(None)
        instance.__func = self.__func
        instance.__argcount = self.__argcount
        instance.__args = self.__args.copy()
        return instance

    def record(self, arg):
        args = self.__args
        if len(args) >= self.__argcount:
            return False
        args.append(arg)
        return True

    def is_ready(self):
        return len(self.__args) == self.__argcount

    def call_func(self):
        return self.__func(*self.__args)

    def get_remaining_argcount(self):
        return self.__argcount - len(self.__args)

    def get_expected_argcount(self):
        return self.__argcount

def curry_partial(f, *initial_args):
    if not callable(f):
        return f
    recorder = ArgRecorder(f)
    for arg in initial_args:
        if not recorder.record(arg):
            break
    if recorder.is_ready():
        return recorder.call_func()
    def result_func_wrapper(recorder):
        def result_func(*args):
            next_recorder = recorder.clone()
            for arg in args:
                if not next_recorder.record(arg):
                    break
            if next_recorder.is_ready():
                return next_recorder.call_func()
            else:
                return result_func_wrapper(next_recorder)
        result_func.real_argcount = recorder.get_remaining_argcount()
        return result_func
    return result_func_wrapper(recorder)
```



使用Mermaid绘制程序流程图

第四题： 编码聚会7



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?

函数式编程是一种将电脑运算视为函数的计算的编程范式，它的主要思想是把运算过程写成嵌套的函数调用，其基础为λ演算。

2. 什么是lambda函数？请举例说明。

Python中的lambda函数是一种写法简洁的匿名函数，通常只有一行，可以直接赋值给变量或者作为函数参数传递。

```
a = lambda: 1 + 1
print(a()) # 2

b = lambda x: x + 1
print(b(5)) # 6

c = lambda a, b: a + b
print(c(2, 3)) # 5
```

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。

高阶函数是一种参数类型为函数或者返回值类型为函数的函数。Python中常见的高阶函数有filter、map、reduce、sorted等等。

```
# filter函数接受一个函数，把一个可迭代的对象的每个元素放入该函数进行测试，返回True的元素将被保留，最终返回一个容器。
print(list(filter(lambda x: x % 2 == 0, (1, 2, 3, 4, 5, 6)))) # [2, 4, 6]

# map函数接受一个函数，把一个可迭代的对象的每个元素放入该函数进行计算，将所有返回的值组成一个新的序列。
print(list(map(lambda x: x + 1, (1, 2, 3)))) # [2, 3, 4]
```

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。

本次实验学习了函数式编程范式、Python中的lambda函数、Python中常见的高阶函数(filter, map, reduce, sorted等等)、Python函数的一些高级知识（如函数的__code__属性可以访问函数的一些基本信息）。