07-Python函数式编程与高阶函数

提纲

- 函数式编程的介绍
- Lambda函数的介绍
- 高阶函数的介绍
- 高阶函数的应用
- 生成器

编程范式：

- 🛠面向过程编程： C语言
- 🥓面向对象编程： Java语言, Python语言, C#
- 🔢函数式编程： Lisp语言, Haskell语言, Go语言, Python语言, Julia语言

什么是函数式编程?

函数式编程是一种编程范式，它将计算机运算视为数学函数的计算，并且避免使用程序状态以及易变对象。

- 函数是头等对象，函数可以是：
  - 变量
  - 另一个函数的参数
  - 另一个函数的返回值
- 不可变性：不改变变量的值，而是创建新的变量
- 函数是无副作用的（不要改变程序状态也就是变量的值）

函数式编程是现代编程语言的趋势：

Go语言和Julia语言没有 `class` 关键字。

1. 🌐并行和分布式计算：函数式编程的纯函数可以避免共享状态和可变数据，使得多线程和分布式环境下的编程更加安全和可靠。
2. 🧩更好的代码组织和可维护性：函数式编程的组织方式使得代码更易于理解、测试和维护，减少了副作用和全局状态的引入，使得程序的行为更加可预测和可控。
3. 🔄强调数据转换和处理：函数式编程可以更轻松地实现数据流的转换和操作。这使得函数式编程在数据处理、数据分析和机器学习等领域具有很高的适用性。
4. 🚀容易进行代码优化和推理：函数式编程的函数相同输入总是产生相同的输出。这种特性使得编译器和运行时环境能够进行更多的优化，使得程序验证和推理更加容易。

Lambda函数（匿名函数）

In [ ]:
```
def double(value):
    return value * 2

my_fun = double
my_fun(10)
```

In [ ]:
```
# 定义一个等价的匿名函数，然后赋值给一个变量
my_fun2 = lambda x: x * 2
my_fun2(10)
```

高阶函数

函数参数是一个函数，或者函数的返回值是一个函数

In [ ]:
```
cars = ['Ford', 'Volvo', 'BMW', 'Honda', 'Tesla']

# 默认按照字母顺序排序
cars.sort()
cars
```

list的sort方法的key参数，可以 接收一个函数作为参数，这个函数的返回值将作为排序的依据。

In [ ]:
```
# 根据元素的长度来排序
cars.sort(key=lambda x: len(x))
cars
```

编写一个lambda函数作为sort方法的参数，将下面的数据按照成绩排序：

```
[('English', 88), ('Science', 90), ('Maths', 97), ('Social sciences', 82)]
```

In [ ]:
```
scores = [('English', 88), ('Science', 97), ('Maths', 97), ('Social sciences', 82)]
scores.sort(      )
scores
```

首先按照成绩排序，然后按照科目排序

In [ ]:
```
scores = [('English', 88), ('Science', 97), ('Maths', 97), ('Social sciences', 82)]
scores.sort(     )
scores
```

Map函数

这是最常见的高阶函数。

- 函数原型： `map(function, iterable, ...)`
- 第1个参数：一个函数（通常是lambda函数）
- 第2个参数：一个或多个可迭代对象（例如list或者tuple）
- 返回值：然后将这个函数依次作用在可迭代对象的每个元素上，最后返回一个新的可迭代对象。

In [ ]:
```
nums = (1, 2, 3, 4)
mapped = map(lambda x: x+x, nums)
```

```
print(list(mapped))
```

In [ ]:
```
msg = 'Hello'
mapped = map(lambda x: x+x, msg)
print(list(mapped))
```

In [ ]:
```
odds = [1, 3, 5]
evens = [2, 4, 6]
mapped = map(lambda a,b:a*b, odds, evens)
print(list(mapped))
```

练习：

- 判断列表中函数是否包含 anonymous 字符串,
- 如果包含，返回 (True,s)，
- 否则返回 (False,s).
- 其中 s 是列表中的字符串。

In [ ]:
```
txt = ['lambda functions are anonymous functions.',
    'anonymous functions dont have a name.',
    'functions are objects in Python.']
```

In [ ]:
```

```

In [ ]:
```

```

max, min函数

max, min函数的key参数，可以 接收一个函数作为参数，这个函数的返回值将作为排序的依据。

In [51]:
```
# 找出总分最高的和总分最低的
scores = [(201, 85), (302, 92), (130, 398), (422, 88)]

print(max(scores))

highest_score = max(scores)
lowest_score = min(scores)

print(highest_score)
print(lowest_score)
```

```
(422, 88)
(130, 398)
(201, 85)
```

filter函数

filter高阶函数：

- 函数原型： filter(function, iterable)
- 第1个参数：一个函数（通常是lambda函数）
- 第2个参数：一个可迭代对象（例如list或者tuple）

- 返回值：将这个函数依次作用在可迭代对象的每个元素上，最后返回一个新的可迭代对象，其中包含了所有返回值为 True 的元素。

In [ ]:
```
# 列表中长度大于3的字符串
def filter_long_strings(string):
    return len(string) >= 3

words = ["apple", "banana", "be", "a", "cat", "to", "elephant"]

long_words = list(filter(filter_long_strings, words))
print(long_words)
```

In [ ]:
```
# 从字符串中筛选出元音字母
word = "apple"

def vowel(c):
    return c.lower() in 'aeiou'

vowel = lambda c: c.lower() in 'aeiou'

filtered = filter(vowel, word)
print(filtered)
print(list(filtered))  # filtered 只能生成一次序列数据
print(list(filtered))  # 第二次产生的数据为空
```

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

In [ ]:
```
lst = [
  { 'firstName': 'Gabriel', 'lastName': 'X.', 'country': 'Monaco', 'continent': 'Euro
  { 'firstName': 'Odval', 'lastName': 'F.', 'country': 'Mongolia', 'continent': 'Asia
  { 'firstName': 'Emilija', 'lastName': 'S.', 'country': 'Lithuania', 'continent': 'E
  { 'firstName': 'Sou', 'lastName': 'B.', 'country': 'Japan', 'continent': 'Asia', 'a
]

# 使用max函数的key参数可以找到年龄最大的程序员
max(lst, key=lambda d:d['age'])
```

In [ ]:
```
# 找到年龄最大的程序员
mage = max(lst, key=lambda d:d['age'])

# 打印他的年龄
mage_age = mage['age']
print(mage_age)

# 根据年龄过滤列表
list(filter(lambda d:d['age'] == mage_age, lst))
```

In [ ]:
```
# best solution:
def find_senior(lst):

    # 利用生成器作为max函数的参数,找到最大的年龄
    mage = max(a['age'] for a in lst)

    # 利用列表推导返回结果
    return [a for a in lst if a['age']==mage]
```

functools模块

partial(func, *args, **kwargs)：创建一个带有预设参数的新函数。

In [ ]:
```
from functools import partial

def divide(x, y):
    return x / y

# Create a new function, first argument is 2
two_divide = partial(divide, 2)
print(two_divide(5))  # Output: 0.4
```

lru_cache(maxsize=None)：用于函数结果的记忆/缓存的装饰器。它会缓存最近的函数调用及其结果。

In [ ]:
```
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# 统计运行时间
%timeit fibonacci(25)
```

In [ ]:

```
from functools import lru_cache

@lru_cache(maxsize=3)
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(5))  # Output: 5
print(fibonacci.cache_info())  # Output: CacheInfo(hits=4, misses=6, maxsize=3, currs

%timeit fibonacci(25)
```

In [ ]:
```
%timeit fibonacci(25)
```

生成器（Generator）

生成器使用简明地方式，懒惰地（lazily）返回序列数据，每次请求数据后会暂停，当有新请求时再次启动。

In [ ]:
```
def gen_nums():
    yield 1
    yield 2
    yield 3

for x in gen_nums():
    print(x)
```

这段代码会输出什么?

In [ ]:
```
def squares(n=10):
    print(f'Generating squares from 1 to {n}')
    for i in range(1, n+1):
        yield i**2

gen = squares()
```

In [ ]:
```
# next()函数可以获取生成器的下一个值
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
```

In [ ]:
```
# 将生成器中的数据转换成列表
print(list(gen))
```

In [ ]:
```
# 遍历生成器生成的数据，但是生成器已经没有数据了
for square in gen:
    print(square)
```

生成器表达式

In [ ]:
```
# 和用def和yield关键字定义生成器函数的效果是一样的
squares = (x**2 for x in range(1, 10))
```

In [ ]:
```
def squares():
    for i in range(1, 10):
        yield i**2
```

生成器表达式用作函数参数，例如：

- sum
- all
- any
- 等可以接收可迭代的数据作为参数的函数

In [ ]:
```
my_sum = sum(x**2 for x in range(1, 10))
print(my_sum)

print(all( x%2==0  for x in [2, 4, 6, 8, 10]))
```

内置的序列生成器

- enumerate：枚举出序列的索引和值
- zip：拉链，把数据按照它们的索引组合到一起

In [ ]:
```
# zip对象只能生成一次数据
zipped = zip(['a', 'b', 'c'], [1, 2, 3])
print(list(zipped))
print(list(zipped))
```

练习

搜索违法的公司，找到所有违反最低收入法律（最低收入为9）的所有公司放入列表。

提示：

- 使用列表推导
- 使用any函数

In [ ]:
```
companies = {'CoolCompany' : {'Alice' : 33, 'Bob' : 28, 'Frank' : 29},
             'CheapCompany' : {'Ann' : 4, 'Lee' : 9, 'Chris' : 7},
             'SosoCompany' : {'Esther' : 38, 'Cole' : 8, 'Paris' : 18}}
```

代码的逻辑结构：

```
违法公司 = [   company  for company in companies   if 公司是否违法  ]
公司是否违法 = any(  employee的salary < 9  for employee in company)
```

In [ ]:
```
# companies[c].values() 是公司c的所有员工的薪水
illegals = [c  for c in companies
            if any(  s<9  for s in companies[c].values())]
print(illegals)
```

使用函数来表示数字0至9和进行四则运算

https://www.codewars.com/kata/525f3eda17c7cd9f9e000b39/train/python

```
five() # must return 5
six() # must return 6
```

```
five(times(five())) # 5 * 5 return 25
four(plus(nine())) # 4 + 9 return 13
eight(minus(three())) # 8 - 3 return 5
six(divided_by(two())) # 6 // 2 return 3
```

In [ ]:

```
# 数字函数的参数为空，或者是一个函数
def one(op=None):
    return 1 if op==None else op(1)

def zero(op=None):
    return 0 if op==None else op(0)

# 操作函数的返回值必须是一个函数
def plus(x):
    return lambda y: y + x
```