07-Python函数式编程与高阶函数

Python函数的类型提示

如果实际的参数类型与类型提示不符合，程序会报错吗?

函数定义的类型提示需要接收:

- 一个列表参数
- 一个整型参数
- 返回一个整数值

In [82]:
```
def pick(l:list, index:int) -> int:
    return l[index]

pick("hello", 2)
```

Out[82]: 'l'

Python中的类型提示:

- 类型提示是可选的
- 不会在运行时捕捉任何类型错误
- 不会用于优化程序的性能

定义main函数

main函数是程序的入口函数，Python中的main函数是可选的，通常的写法如下:

In [14]:
```
def main():
    # main函数应该作为程序的入口函数
    print("This is main function.")

# 只有当前文件是程序启动的文件时，if条件判断为True
if __name__ == "__main__":
    main()
```

```
This is main function.
```

main函数的最佳实践:

- 把运行时间长的或者有其他效果的代码放入函数或类中。
- 使用name和条件语句来控制代码的执行。
- 将入口函数命名为main()，把程序的入口逻辑放入main()函数中。
- 在main()函数中调用其他函数或者类。

变量作用域范围

- 全局变量（global）：在函数外部定义的变量
- 局部变量（local）：在函数内部定义的变量

- global： 在函数内部使用global关键字声明全局变量

In [11]:
```
# 这是一个全局变量
msg = 'hello'

def greet():

    # 这是一个局部变量
    local_var = 100
    print(local_var)

    # 局部变量与全局变量同名，会覆盖全局变量
    global msg # 声明为全局变量
    msg = 'goodbye'
    print(msg)

greet()
# 打印全局变量
print(msg)
```

```
100
goodbye
goodbye
```

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
- 函数式编程： Lisp语言，Haskell语言，Scala语言，Python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
  - 函数的参数
  - 函数的返回值
- 变量是不可变的
- 递归取代循环
- 函数是无副作用的（不要改变程序状态也就是变量的值）
- 使用Lambda函数：匿名函数
- 使用高阶函数：函数参数是一个函数，或者函数的返回值是一个函数

Lambda函数（匿名函数）

In [17]:
```
lambda x: x * 2
```

Out[17]:
```
<function __main__.<lambda>(x)>
```

In [16]:
```
double = lambda x: x * 2
double(10)
```

Out[16]: 20

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

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 [19]:
```
cars = ['Ford', 'Volvo', 'BMW', 'Honda', 'Tesla']
# 根据元素的长度来排序
cars.sort(key=lambda x: len(x))
cars
```

Out[19]: ['BMW', 'Ford', 'Volvo', 'Honda', 'Tesla']

将下面的数据按照成绩排序：

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

In [25]:
```
scores = [('English', 88), ('Science', 97), ('Maths', 97), ('Social sciences', 82)]
scores.sort(key=lambda x:x[-1])
scores
```

Out[25]: [('Social sciences', 82), ('English', 88), ('Science', 97), ('Maths', 97)]

首先按照成绩排序，然后按照科目排序

In [28]:
```
scores = [('English', 88), ('Science', 97), ('Maths', 97), ('Social sciences', 82)]
scores.sort(key=lambda x:(x[-1], x[0]))
scores
```

Out[28]: [('Maths', 97), ('Science', 97), ('English', 88), ('Social sciences', 82)]

Map函数

这是最常见的高阶函数。它的两个参数和返回值：

- 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 [29]:
```
nums = (1, 2, 3, 4)
mapped = map(lambda x: x+x, nums)
print(list(mapped))
```

[2, 4, 6, 8]

In [30]:
```
odds = [1, 3, 5]
evens = [2, 4, 6]
mapped = map(lambda a,b:a+b, odds, evens)
print(list(mapped))
```

```
[3, 7, 11]
```

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 [32]:
```
txt = ['lambda functions are anonymous functions.',
    'anonymous functions dont have a name.',
    'functions are objects in Python.']
```

In [35]:
```
print('anonymous' in txt[2])
```

```
False
```

In [37]:
```
mapped = map(lambda s:('anonymous' in s, s), txt)
list(mapped)
```

Out[37]:
```
[(True, 'lambda functions are anonymous functions.'),
 (True, 'anonymous functions dont have a name.'),
 (False, 'functions are objects in Python.')]
```

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 [1]:
```
# 找出总分最高的和总分最低的
scores = [(201, 85), (302, 92), (130, 78), (422, 88)]

# Sorting the list of tuples by the second element
highest_score = max(scores, key=lambda s: sum(s))
lowest_score = min(scores, key=lambda s: sum(s))

print(highest_score)
print(lowest_score)
```

```
(422, 88)
(130, 78)
```

filter函数的范例

In [85]:
```
# 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))
long_words = list(filter(lambda s: len(s)>2, words))
print(long_words)
```

```
['apple', 'banana', 'cat', 'elephant']
```

In [7]:
```
# 从字符串中筛选出元音字母
word = "apple"

def vowel(c):
    return 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))
```

```
<filter object at 0x00000296252E0340>
['a', 'e']
[]
```

functools模块

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

In [79]:
```
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
```

```
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 [101]:
```
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

# 统计运行时间
%timeit fibonacci(25)
```

```
20.4 ms ± 877 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

In [102]:
```
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, currsize=3)

%timeit fibonacci(25)
```

```
5
CacheInfo(hits=3, misses=6, maxsize=3, currsize=3)
53.3 ns ± 0.275 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

In [96]:
```
%timeit fibonacci(25)
```

```
55.4 ns ± 2.22 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

any 和 all 函数

都是Python内置函数，用于对于序列数据（布尔变量）连续进行Or或者And操作，返回True或False

In [11]:
```
print(all([True, True, True, True]))
print(all([True, True, False, True]))
```

```
True
False
```

In [5]:
```
print(any([False, True, False, False]))
print(any([False, False, False, False]))
```

```
True
False
```

In [6]:
```
# 判断一个列表中全是偶数（all）和存在任意偶数（any）

print(all( [x%2==0  for x in [2, 4, 6, 8, 10]]))
print(all( [x%2==0  for x in [2, 4, 6, 8, 11]]))


print(any([ x%2==0  for x in [2, 4, 6, 8, 11]]))
print(any([ x%2==0  for x in [1, 3, 5, 7, 9]]))
```

```
True
False
True
False
```

生成器（Generator）

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

In [12]:
```
def gen_nums():
    yield 1
    yield 2
    yield 3

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

```
1
2
3
```

这段代码会输出什么?

In [50]:
```
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 [51]:
```
# 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))
```

```
Generating squares from 1 to 10
1
4
9
16
25
```

In [52]:
```
# 将生成器中的数据转换成列表
print(list(gen))
```

```
[36, 49, 64, 81, 100]
```

In [57]:
```
# 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))
```

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 [58]:
```
my_sum = sum(x**2 for x in range(1, 10))
print(my_sum)
```

```
285
```

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 [59]:
```
# 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))
```

```
[('a', 1), ('b', 2), ('c', 3)]
[]
```

练习一

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 [89]:
```
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(  salary < 9  for salary in 公司所有员工的薪水)
```

In [90]:
```
illegals = [ c for c,salaries in companies.items()
             if any(salary < 9  for salary in salaries.values())]
print(illegals)
```

```
['CheapCompany', 'SosoCompany']
```

In [65]:
```
illegals = [ c for c in companies
             if any(companies[c][e] < 9
                 for e in companies[c] )]
print(illegals)
```

```
['CheapCompany', 'SosoCompany']
```

练习二

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

```
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
```

In [79]:
```
def zero(fun=None): return fun(0) if fun else 0
def one(fun=None): return fun(1) if fun else 1

def plus(y): return lambda x: x + y
```

In [76]:
```
# 测试没有参数的函数
print(zero())
print(one())
```

Out[76]:
```
0
```

In [81]:
```
zero(print)  # print(0)
one(print)  # print(1)

# one(plus(zero())) -> one(plus(0)) -> one(lambda:x + 0) -> fun(1) = lambda:1 + 0 -> 1
one(plus(zero()))
```

```
0
```

Out[81]:
```
1
```