函数式编程

什么是函数式编程

函数式编程的一个特点就是，允许把函数本身作为参数传入另一个函数，还允许返回一个函数！

题目

写一个程序完成累加和累积的操作

```
def addSum(array):
  sum = 0
  for i in array:
    sum+=i
  return sum

def mulSum(array):
  sum = 1
  for i in array:
    sum *= i
  return sum

array=list(range(1,101))
print('addSum is',addSum(array))
print('mulSum is',mulSum(array))
```

```
addSum is 5050
mulSum is 933262154439441526816992388562667004907159682643816214685929638952175999
```

```python
# 函数式写法
def add(a,b):
  return a+b
def mul(a,b):
  return a*b

def sum(fun,array):
  s = array[0]
  for i in array[1:]:
    s = fun(s,i)
  return s

array=list(range(1,101))
print('addSum is',sum(add,array))
print('mulSum is',sum(mul,array))
```

```
addSum is 5050
mulSum is 933262154439441526816992388562667004907159682643816214685929638952175999
```

高阶函数

将函数作为变量，通过函数变量调用函数的函数。

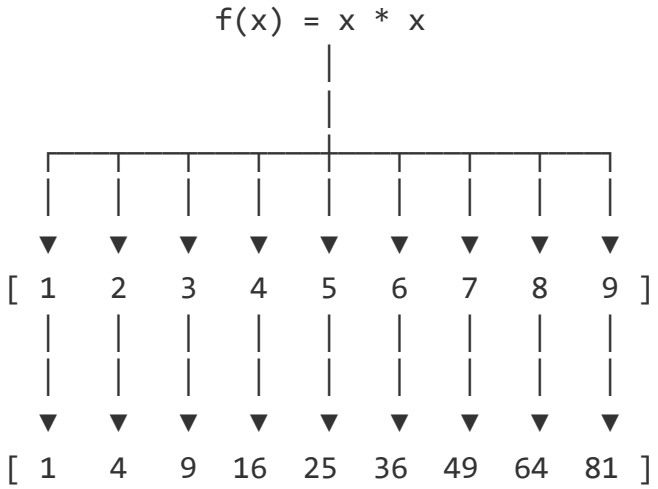
python内置高阶函数

map/redduce

filter

sorted

map()函数接收两个参数，一个是函数，一个是Iterable，map将传入的函数依次作用到序列的每个元素，并把结果作为新的Iterator返回。




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

r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
list(r)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

reduce把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算，其效果就是：

```
reduce(f, [x1, x2, x3, x4]) = f(f(f(x1, x2), x3), x4)
```

```
#采用reduce函数
from functools import reduce
def add(a,b):
  return a+b
def mul(a,b):
  return a*b

array=list(range(1,101))
print('addSum is',reduce(add,array))
print('mulSum is',reduce(mul,array))
```

```
addSum is 5050
mulSum is 933262154439441526816992388562667004907159682643816214685929638952175999
```

匿名函数

当我们在传入函数时，有些时候，不需要显式地定义函数，直接传入匿名函数更方便

```
#采用reduce+lambda函数

from functools import reduce

array=list(range(1,101))
print('addSum is',reduce(lambda a,b:a+b,array))
print('mulSum is',reduce(lambda a,b:a*b,array))
```

```
addSum is 5050
mulSum is 933262154439441526816992388562667004907159682643816214685929638952175999
```

函数作为返回值

```
def calc_sum(*args):
  ax = 0
  for n in args:
    ax = ax + n
  return ax

def lazy_sum(*args):
  def sum():
      ax = 0
      for n in args:
          ax = ax + n
      return ax
  return sum


f = lazy_sum(1, 3, 5, 7, 9)
print(f)
```

```
<function lazy_sum.<locals>.sum at 0x000001DE4FA578B8>
```

```
f()
```

```
25
```

装饰器

增强函数的功能，比如，在函数调用前后自动打印日志，但又不修改now()函数的定义，这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。

```
import time
def hello():
  print('hello')

def log(func):
    def wrapper(*args, **kw):
        print(time.strftime('[%Y-%m-%d %H:%M:%S] ',time.localtime(time.time())) ,end='')
        return func(*args, **kw)
    return wrapper

hello()
```

```
hello
```

```
[2025-10-24 15:11:28] hello
```

```
@log
def hello():
  print('hello')

hello()
```

[2025-10-24 15:12:49] hello