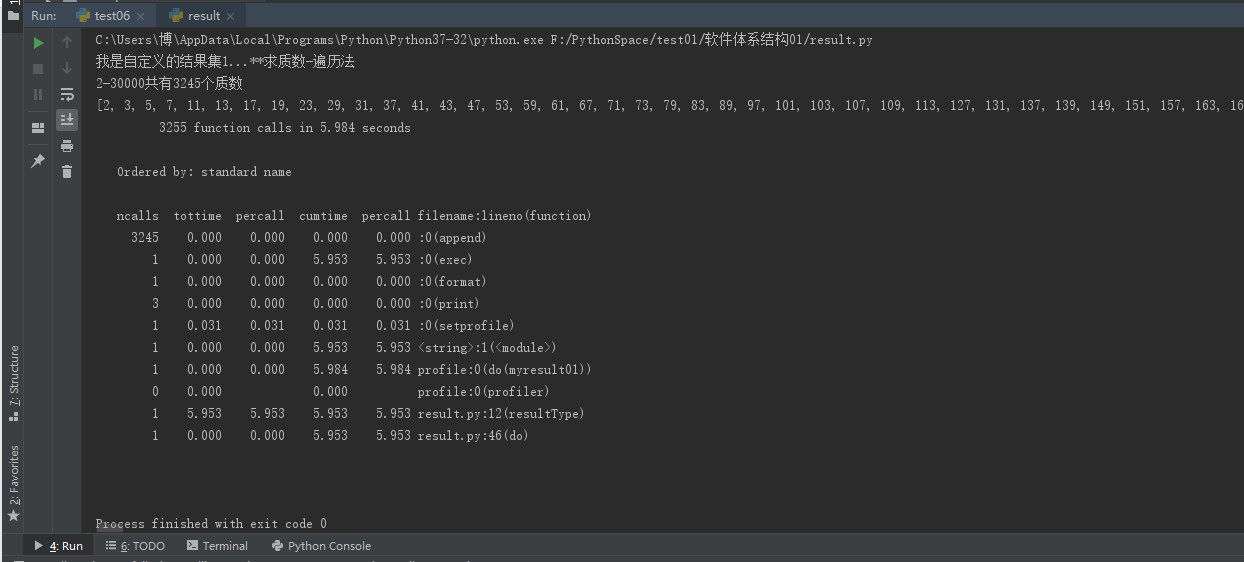
对于上次的自定义实现不同类型的返回结果，完善了上次的代码

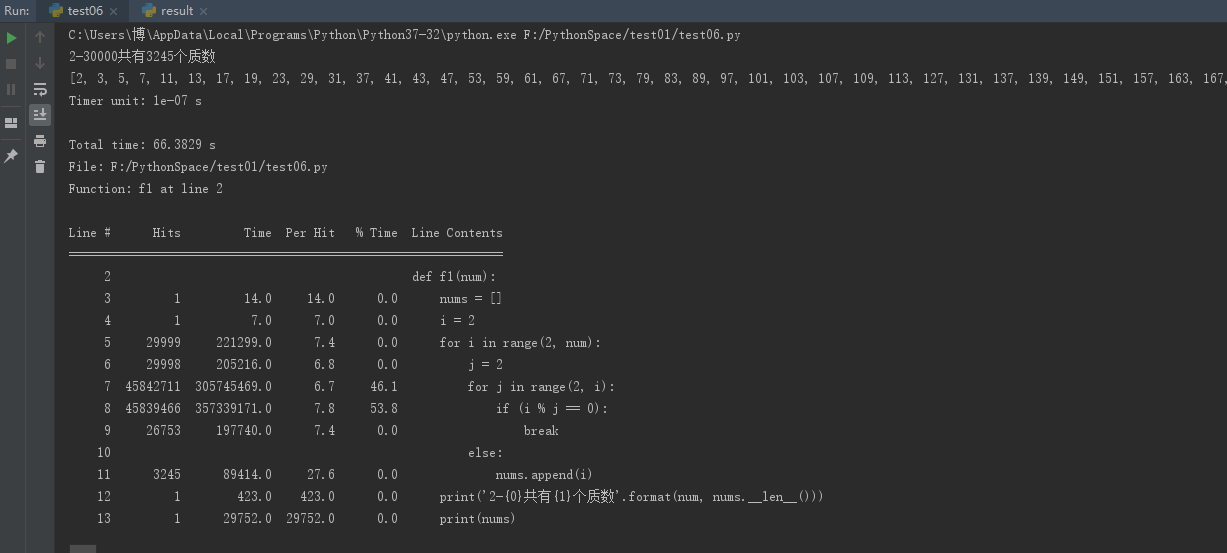
|  |
| --- |
| import profile import math  from line\_profiler import LineProfiler  class result:  def resultType(self,num):   print("我是结果类型")   class myResult01(result):  def resultType(self,num):  nums=[]  print("我是自定义的结果集1...\*\*求质数-遍历法")  i = 2  for i in range(2,num):  j = 2  for j in range(2,i):  if(i%j == 0):  break  else:  nums.append(i)  print('2-{0}共有{1}个质数'.format(num, nums.\_\_len\_\_()))  print(nums)     class myResult02(result):  def resultType(self,num):  print("我是自定义的结果集2...\*\*求质数-删除倍数法")  nums = [n for n in range(2, num)]  i = 2  for i in range(2, num):  j = i \* 2  for j in range(i \* 2, num, i):  # print(j)  if j in nums:  nums.remove(j)  print('2-{0}共有{1}个质数'.format(num, nums.\_\_len\_\_()))  print(nums)     def do(a):  a.resultType(10000)     ''' do(result) do(myresult01) do(myresult02)  ''' #do(myresult01) lp = LineProfiler() lp.add\_function(resultType)  result = result() myresult01 = myResult01() myresult02 = myResult02()   lp\_wrapper = lp(do) lp\_wrapper(myresult02)  lp.print\_stats()  #profile.run('do(myresult01)') #profile.run('do(myresult02)') |

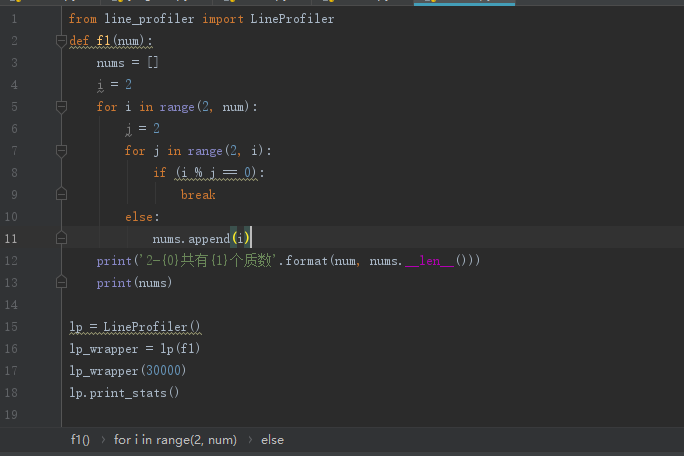
**下面对返回结果1进行测试：**

|  |
| --- |
| from line\_profiler import LineProfiler def f1(num):  nums = []  i = 2  for i in range(2, num):  j = 2  for j in range(2, i):  if (i % j == 0):  break  else:  nums.append(i)  print('2-{0}共有{1}个质数'.format(num, nums.\_\_len\_\_()))  print(nums)  lp = LineProfiler() lp\_wrapper = lp(f1) lp\_wrapper(30000) lp.print\_stats() |

**测试截图：**





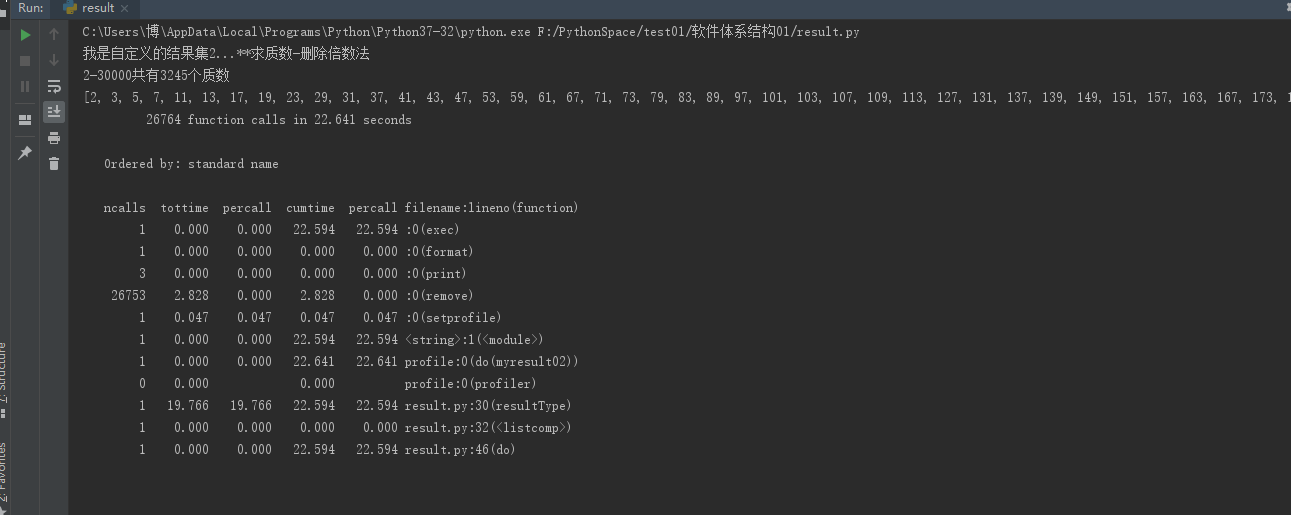


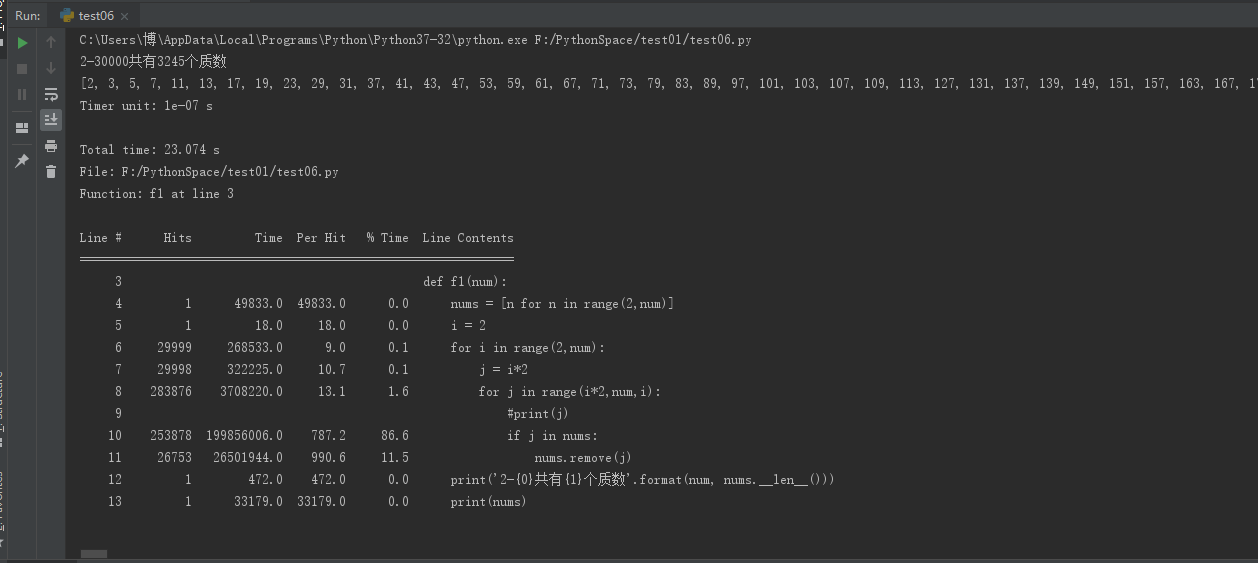
测试结果出入很大，可以看出来采用循环嵌套的方法，使得内层循环的次数达到了3亿次，这是一个很大的数字了，时间都用在不停地循环与对2去模的运算上面，使得函数的效率变得很低。

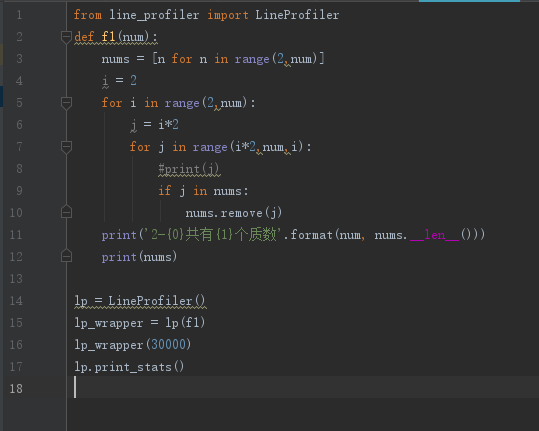
**下面对返回结果2进行测试：**

|  |
| --- |
| from line\_profiler import LineProfiler  def f1(num):  nums = [n for n in range(2,num)]  i = 2  for i in range(2,num):  j = i\*2  for j in range(i\*2,num,i):  #print(j)  if j in nums:  nums.remove(j)  print('2-{0}共有{1}个质数'.format(num, nums.\_\_len\_\_()))  print(nums)  lp = LineProfiler()   lp\_wrapper = lp(f1) lp\_wrapper(30000)  lp.print\_stats()  #f1(30000) |

**测试截图：**







测试结果出入不大，可以看出来主要是列表的remove方法在占用时间，循环判断等占用时间很少,因为是对于列表的操作，前面的元素被删除，后面的元素都需要向前不断移动，造成时间上的极大浪费，从而使得该方法的计算时间变长