Profiling

Sometimes the programmer wants to measure some attributes like the use of memory, time complexity or usage of particular instructions about the programs to measure the real capability of that program. Such kind of measuring about program is called profiling. Profiling uses dynamic program analysis to do such measuring.

In the subsequent sections, we will learn about the different Python Modules for Profiling.

cProfile – the inbuilt module

cProfile is a Python built-in module for profiling. The module is a C-extension with reasonable overhead that makes it suitable for profiling long-running programs. After running it, it logs all the functions and execution times. It is very powerful but sometimes a bit difficult to interpret and act on. In the following example, we are using cProfile on the code below:

Profiling01.py

Line Code

```python
import threading

def increment_global():
    global x
    x += 1

def taskofThread(lock):
    for _ in range(50000):
        lock.acquire()
        increment_global()
        lock.release()

def main():
    global x
    x = 0

    lock = threading.Lock()

    t1 = threading.Thread(target=taskofThread, args=(lock,))
    t2 = threading.Thread(target= taskofThread, args=(lock,))

    t1.start()
    t2.start()

    t1.join()
    t2.join()

if __name__ == "__main__":
    for i in range(5):
        main()
        print("x = {1} after Iteration {0}".format(i,x))
```

The above code is saved in the **Profiling01.py** file. Now, execute the code with cProfile on the command line as follows −

```
>python -m cProfile thread_increment.py
x = 100000 after Iteration 0
x = 100000 after Iteration 1
x = 100000 after Iteration 2
x = 100000 after Iteration 3
x = 100000 after Iteration 4
      3577 function calls (3522 primitive calls) in 1.688 seconds

   Ordered by: standard name

   ncalls tottime percall cumtime percall filename:lineno(function)

   5        0.000        0.000        0.000        0.000        <frozen
importlib._bootstrap>:103(release)
   5        0.000        0.000        0.000        0.000        <frozen
importlib._bootstrap>:143(__init__)
   5        0.000        0.000        0.000        0.000        <frozen
importlib._bootstrap>:147(__enter__)
   … … … …
```

From the above output, it is clear that cProfile prints out all the 3577 functions called, with the time spent in each and the number of times they have been called. Followings are the columns we got in output −

- **ncalls** − It is the number of calls made.
- **tottime** − It is the total time spent in the given function.
- **percall** − It refers to the quotient of tottime divided by ncalls.
- **cumtime** − It is the cumulative time spent in this and all subfunctions. It is even accurate for recursive functions.
- **percall** − It is the quotient of cumtime divided by primitive calls.
- **filename:lineno(function)** − It basically provides the respective data of each function.