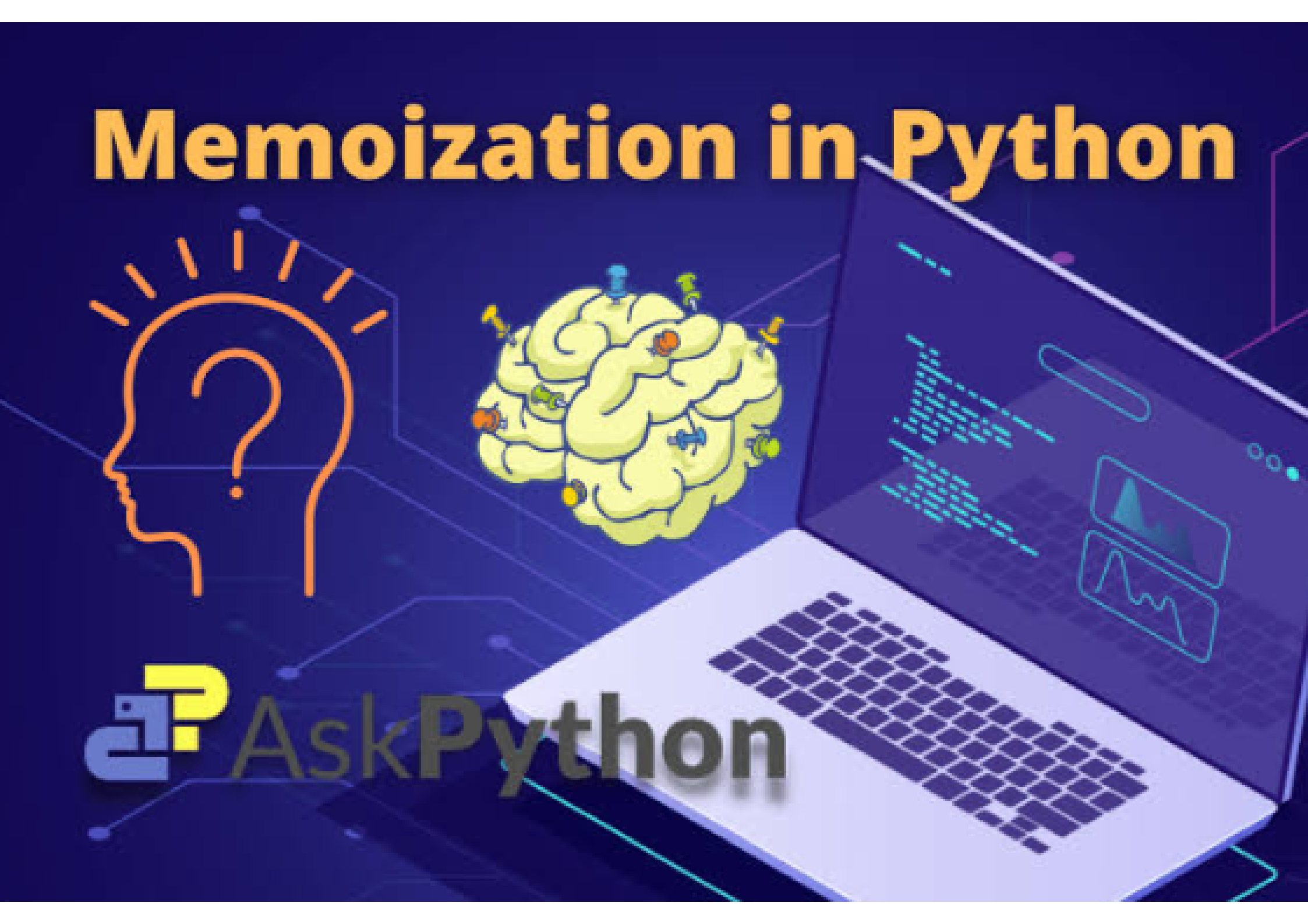
Python

Memoization in Python




from functools import lru_cache
import time
def fib_without_cache(n):
if n < 4:
return n
return fib_without_cache(n-1) + fib_wit
begin = time.time()
fib_without_cache(60)
end = time.time()
print("Time execution without lru_cache
@lru_cache(maxsize = 128)
def fib_with_cache(n):
if n < 4:
return n
PYTHON FUNCTOOLS
LRU_CACHE

Functools module

The **functools module** in python is used to create higher-order functions that interact with other functions. The higher-order functions either return other functions or operate on them to broaden the scope of the function without modifying or explicitly defining them.

The cache decorator

`cache` is a decorator that helps in reducing function execution for the same inputs using the memoization technique. The function returns the same value as `lru_cache(maxsize=None)`, where the cache grows indefinitely without evicting old values. The decorator creates a thin wrapper around a dictionary lookup for the function arguments.

Syntax

```
@cache
```

The decorator has no parameters.

What does the term caching mean?

Caching means storing the data in a place from where it can be served faster. In the case of data that has been frequently used, the computer assigns a cache memory, so it does not have to load it again and again from the main memory. The purpose of the cache is to make the tasks more efficient and quicker. The same is true for web browsers; the pages we load again and again are stored in the cache for faster retrieval. In Python, however, we have to do it all manually, as the program will not store anything in the cache itself.

How to use function caching in Python?

Function caching is a way to improve code's performance by storing the function's return values. Before the 3.2 updates of Python, we had to create a cache ourselves by storing the value in a variable or by other such means. But in **Python 3.2**, there is a new update in the **functools** module of Python. To use this module, we have to import it first.

```
import functools
```

We have been facilitated with the help of a decorator known as **lru_cache. Decorators** are an essential part of Python. Decorators in Python can be used for a variety of different purposes.

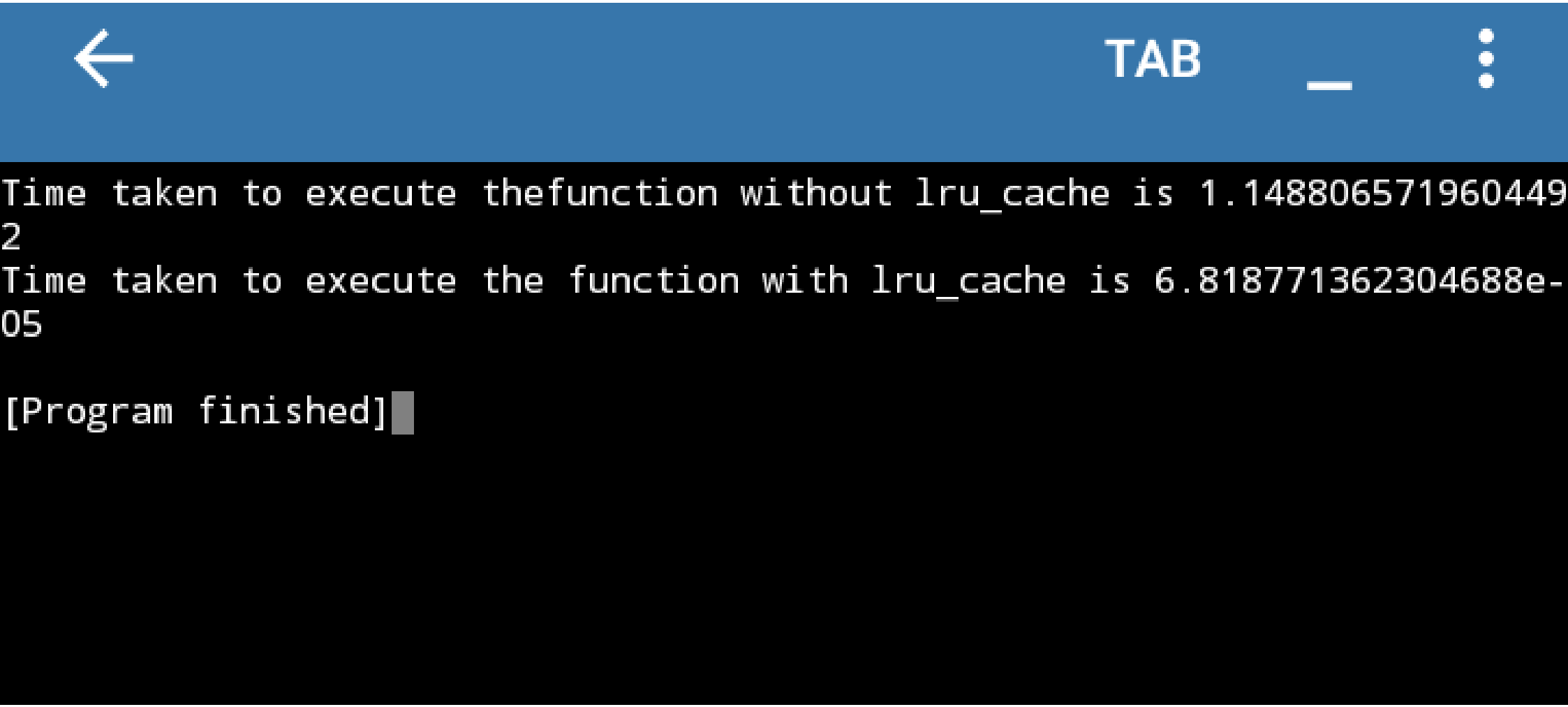
For example

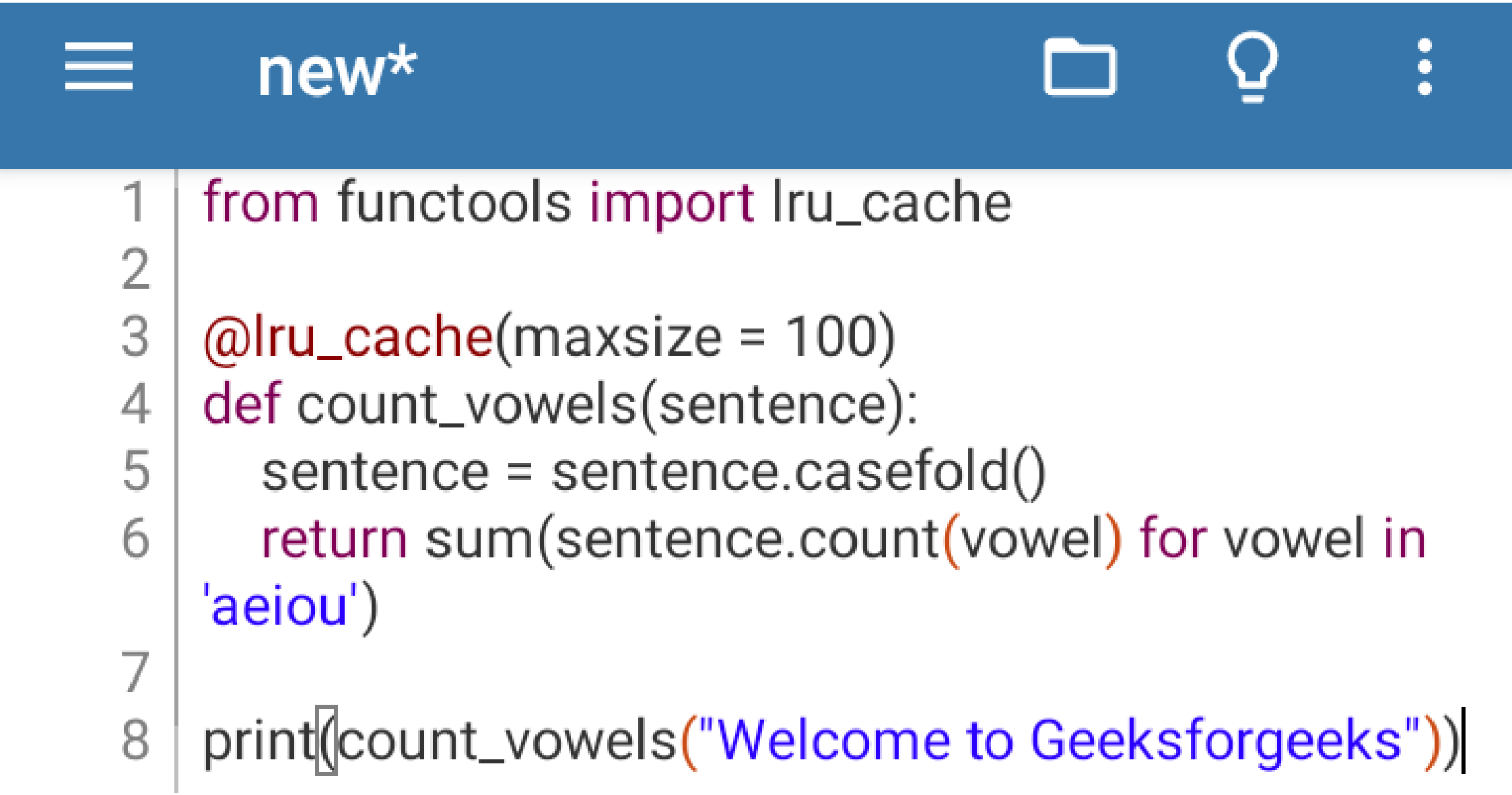
```
@functools.lru_cache(maxsize=4)
def myfunc(x):
    time.sleep(2)
    return x

myfunc(1)
# myfunc(1) takes 2 seconds and results for myfunc(1) are now cached
myfunc(1)
myfunc(2)
myfunc(3)
myfunc(4)
myfunc(5)
```

We set the maxsize equal to 4, and the program uses the same call five times. Then, the program will only be able to retrieve the data faster for the last five calls because caching is only storing data for them. It is important to define the maxsize as per our requirements because it takes up memory accordingly, so for a better program, it should be precisely according to our needs.

```
1  from functools import lru_cache
2  import time
3
4
5  # Function that computes Fibonacci
6  # numbers without lru_cache
7  def fib_without_cache(n):
8      if n < 2:
9          return n
10     return fib_without_cache(n-1) +
   fib_without_cache(n-2)
11
12 # Execution start time
13 begin = time.time()
14 fib_without_cache(30)
15
16 # Execution end time
17 end = time.time()
18
19 print("Time taken to execute the\
20 function without lru_cache is", end-begin)
21
22 # Function that computes Fibonacci
23 # numbers with lru_cache
24 @lru_cache(maxsize = 128)
25 def fib_with_cache(n):
26     if n < 2:
27         return n
28     return fib_with_cache(n-1) + fib_with_cache(n-2)
29
30 begin = time.time()
31 fib_with_cache(30)
32 end = time.time()
33
34 print("Time taken to execute the \
35 function with lru_cache is", end-begin)
36
```


TAB
Time taken to execute thefunction without lru_cache is 1.1488065719604492
Time taken to execute the function with lru_cache is 6.818771362304688e-05
[Program finished]


new*
from functools import lru_cache
@lru_cache(maxsize = 100)
def count_vowels(sentence):
sentence = sentence.casefold()
return sum(sentence.count(vowel) for vowel in 'aeiou')
print(count_vowels("Welcome to Geeksforgeeks"))


TAB
9
[Program finished]