What is optimization

In Python, *optimization* refers to techniques used to improve the efficiency of code in terms of speed (time complexity) or resource consumption (space complexity). There are different forms of optimization, which can be applied to algorithms, code structure, or memory usage.

Type of optimization

1. Algorithmic Optimization

- Choose the right algorithm and data structure for the problem.
- Example: Using a hash map (dict) for fast lookups instead of a list if the problem requires many searches.

2. Code-Level Optimization

- Use built-in functions and libraries that are implemented in C for better performance (e.g., sum(), map(), filter(), etc.).
- Example: Instead of using a for loop to sum elements in a list, use sum().

```
python
# Less efficient
total = 0
for num in numbers:
    total += num

# More efficient
total = sum(numbers)
```

3. Memory Optimization

- Use data structures that are memory efficient like generators instead of lists.
- Example: Using a generator expression instead of a list comprehension if you don’t need to store all values at once.

```
python
# List comprehension (memory inefficient for large datasets)
squares = [x**2 for x in range(1000000)]

# Generator (memory efficient)
squares_gen = (x**2 for x in range(1000000))
```

4. Parallelism and Concurrency

- Utilize Python's multiprocessing or concurrent.futures libraries to run tasks in parallel or asynchronously.
- Example: Running CPU-bound tasks in parallel using multiprocessing.

5. External Libraries

- Use optimized libraries like NumPy for numerical computations, as they are implemented in highly efficient C code.

```
python
import numpy as np

# Vectorized operation using NumPy (efficient)
arr = np.array([1, 2, 3, 4])
```

```
result = np.sum(arr)
```

6. Profiling and Benchmarking

- Tools like cProfile and timeit can help identify bottlenecks in the code. Once bottlenecks are identified, they can be optimized.

```
python
import cProfile
cProfile.run('my_function()')
```

7. Caching and Memoization:-

- Caching frequently used results can save computation time. Python’s functools.lru_cache provides a simple way to cache results of function calls.

```
python
from functools import lru_cache

@lru_cache(maxsize=None)
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

These are just a few techniques for optimization in Python.