MEMOIZATION

לפי ויקיפדיה: Memoization או memoisation היא טכניקה להאצת תכניות מחשב ע"י שמירה של ערכי חזרה של פונקציות עם סיבוכיות גבוהה בסיס נתונים, ושימוש בנתונים השמורים במקום חישוב מחדש של הפונקציה.
כדי להמחיש את הרעיון נסתכל על הדוגמא הבאה:

```
def fibonacci_recursive(n):
    if n == 1 :
        return 1
    elif n == 2 :
        return 1
    else:
        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)
```

הפונקציה לעיל היא פונקציית פיבונצ'י רקורסיבית- היא מקבלת מספר n כלשהו ומחשבת את המספר ה-n בסדרת פיבונצ'י.
כדי לדעת מהו המספר ה-n היא צריכה לדעת מה המספר שהיה לפני כן וזה שלפניו וכו', לכן כל פעם שנרצה למצוא את המספר ה-n נצטרך לבצע את אותו החישוב על n מספרים לפני. בגלל שהפונקציה רקורסיבית היא מכבידה אפילו יותר על סיבוכיות הקוד. עבור מספרים קטנים קשה להבחין בזה , אבל אם נרצה לחשב את המספר ה-100 בסדרת פיבונצ'י זה יכול לקחת לנו נצח.
אבל אל דאגה, יש פתרון פשוט לבעיה- memoization: כל פעם נשמור את הערכים שקיבלנו עד עכשיו (ערכי הבניים או התוצאות הסופיות) במילון אם הם לא שמורים בו עדיין, כך שלא נצטרך לחשב אותם כל פעם מחדש. ואז אם יבוא ערך שלא פגשנו עד עכשיו נוכל להשתמש בערכים שאנחנו מכירים בינתיים במקום לחשב גם אותם מחדש.
בקוד זה יראה ככה:

```
fibonacci_cache = {}
def fibonacci_recursive_with_cache(n):
    if n in fibonacci_cache:
        return fibonacci_cache[n]
    if n == 1 :
        return 1
    elif n == 2 :
        return 1
    else:
        value = fibonacci_recursive_with_cache(n-1) + fibonacci_recursive_with_cache(n-2)
    fibonacci_cache[n] = value
    return value
```

דרך נוחה יותר להשתמש ב-memoization היא ע"י שימוש ב-decorator מיוחד של הספרייה functool שנקראת lru_cache . lru אלה ראשי תיבות של Least Recently Used cache . הפונקציה מקבלת כפרמטר כמה מספרים היא יכולה לזכור (maxsize), כברירת מחדל היא שומרת עד 100 ערכים.
ועכשיו מבלי לשנות את הפונקציה הראשונה נוכל להאיץ אותה עם הקשטן:

```python
from functools import lru_cache

@lru_cache(maxsize = 100)
def fibonacci_recursive(n):
    if n == 1 :
        return 1
    elif n == 2 :
        return 1
    else:
        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)
```