Простота кортежей, с одной стороны, даёт преимущества (кортежи занимают мало памяти), но с другой — у неё есть и недостатки.

Например, если мы захотим проверить, есть ли какой-то объект внутри кортежа, Python пройдёт по всем объектам, сравнивая их с целевым. Соответственно, в худшем случае сложность такой операции будет равна O(n).

При этом у словарей и множеств такая операция будет иметь сложность O(1). Выходит, сколько бы элементов ни добавляли в словарь, время поиска ключа почти не будет увеличиваться. В чём магия? Чтобы в этом разобраться, нужно познакомиться с тремя понятиями:

- хеш;
- хеш-функция;
- хеш-таблица.

Хеш, или хеш-значение, — результат применения хеш-функции к каким-либо данным (в нашем случае к неизменяемым). Это число фиксированной длины, служащее «подписью» для входных данных, к которым применена хеш-функция.

Хеш-функция — функция, которая преобразует входные данные произвольной длины в выходное хеш-значение фиксированной длины. Грубо говоря, хеш-функция кодирует (шифрует) данные.

Например:

В Python встроена реализация хеш-функции — hash. Попробуем её использовать:

```
tuple = (1, 2, 3) # Есть неизменяемый объект (кстати, попробуйте потом повторить этот код с изменяемым объектом)
hash_value = hash(tuple) # Применим к этому объекту функцию hash
print(hash_value) # Проверим, что получилось (бессмысленный набор чисел)
hash_value_2 = hash(tuple) # Попробуем ещё раз
print(hash_value_2) # Опять набор чисел
print(hash_value == hash_value_2) # И он в точности равен первому
```

В идеале при работе с одинаковыми объектами получаются одинаковые хеш-значения, а с разными объектами — разные хеш-значения. Но иногда бывают коллизии: два разных объекта возвращают одно и то же хеш-значение.

Чем лучше реализована хеш-функция, тем меньше с ней возникает коллизий.

При желании вы можете сами реализовать такую хеш-функцию. Один из самых простых примеров:

```
def simple_hash(input_string): # На вход получаем строку
    hash_value = 0
    for char in input_string: # Запускаем цикл по символам строки
        hash_value += ord(char) # Суммируем код каждого символа
    return hash_value # На выходе получаем сумму — некое числовое
```