**Хеш-функции** используются в [криптографических алгоритмах](https://python-scripts.com/encryption-cryptography), электронных подписях, кодах аутентификации сообщений, обнаружении манипуляций, сканировании отпечатков пальцев, контрольных суммах (проверка целостности сообщений), хеш-таблицах, **хранении паролей** и многом другом.

Как Python-разработчику, вам могут понадобиться эти функции для проверки дубликатов данных и файлов, проверки целостности данных при передаче информации по сети, безопасного **хранения паролей в базах данных** или, возможно, для какой-либо работы, связанной с криптографией.

Некоторые часто используемые хеш-функции:

* **MD5**: Алгоритм производит хеш со значением в 128 битов. Широко используется для проверки целостности данных. Не подходит для использования в иных областях по причине уязвимости в безопасности MD5.
* **SHA**: Группа алгоритмов, что были разработаны NSA Соединенных Штатов. Они являются частью Федерального стандарта обработки информации США. Эти алгоритмы широко используются в нескольких криптографических приложениях. Длина сообщения варьируется от 160 до 512 бит.

Модуль hashlib, включенный в стандартную библиотеку Python, представляет собой модуль, содержащий интерфейс для самых популярных алгоритмов хеширования. hashlib реализует некоторые алгоритмы, однако, если у вас установлен OpenSSL, hashlib также может использовать эти алгоритмы.

Данный код предназначен для работы в **Python 3.5** и выше. При желании запустить эти примеры в Python 2.x, просто удалите вызовы attributems\_available и algorithms\_guaranteed.

Сначала [импортируется модуль](https://python-scripts.com/import-modules-python) hashlib:

import hashlib

Теперь для списка доступных алгоритмов используются algorithms\_available и algorithms\_guaranteed.

print(hashlib.algorithms\_available)

print(hashlib.algorithms\_guaranteed)

Метод algorithms\_available создает **список всех алгоритмов**, доступных в системе, включая те, что доступны через OpenSSl. В данном случае в списке можно заметить дубликаты названий. algorithms\_guaranteed перечисляет только алгоритмы модуля. Всегда присутствуют md5, sha1, sha224, sha256, sha384, sha512.

**Примеры кода с хеш-функциями в Python**

Код ниже принимает строку "Hello World" и выводит дайджест HEX данной строки. hexdigest возвращает строку HEX, что представляет хеш, и в случае, если вам нужна последовательность байтов, нужно использовать дайджест.

## MD5 — пример хеширования

import hashlib

hash\_object = hashlib.md5(b'Hello World')

print(hash\_object.hexdigest())

Обратите внимание, что "b" предшествует литералу строки, происходит конвертация строки в байты, оттого, что функция хеширования принимает только последовательность байтов в качестве параметра. В предыдущей версии библиотеки принимался литерал строки.

Итак, если вам нужно принять какой-то ввод с консоли и хешировать его, не забудьте закодировать строку в последовательности байтов:

import hashlib

mystring = input('Enter String to hash: ')

# Предположительно по умолчанию UTF-8

hash\_object = hashlib.md5(mystring.encode())

print(hash\_object.hexdigest())

Предположим, нам нужно хешировать строку "Hello Word" с помощью функции MD5. Тогда результатом будет 0a4d55a8d778e5022fab701977c5d840bbc486d0.

## SHA1 — пример хеширования

import hashlib

hash\_object = hashlib.sha1(b'Hello World')

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

## Хеширование на SHA224

import hashlib

hash\_object = hashlib.sha224(b'Hello World')

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

## Хеширование на SHA256

import hashlib

hash\_object = hashlib.sha256(b'Hello World')

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

## Пример хеширования на SHA384

import hashlib

hash\_object = hashlib.sha384(b'Hello World')

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

## Пример хеширования на SHA512

import hashlib

hash\_object = hashlib.sha512(b'Hello World')

hex\_dig = hash\_object.hexdigest()

print(hex\_dig)

Предположим, вам нужен алгоритм, предоставленный OpenSSL. Используя algorithms\_available, можно найти название необходимого алгоритма.

В данном случае,  на моем компьютере доступен «DSA». Вы можете использовать методы new и update

import hashlib

hash\_object = hashlib.new('DSA')

hash\_object.update(b'Hello World')

print(hash\_object.hexdigest())

**Реальный пример хеширования паролей Python**

В следующем примере пароли будут хешироваться для последующего сохранения в базе данных. Здесь мы будем использовать salt. salt является случайной последовательностью, добавленной к строке пароля перед использованием хеш-функции. salt используется для предотвращения перебора по словарю (**dictionary attack**) и атак радужной таблицы (**rainbow tables attacks**).

Тем не менее, если вы занимаетесь реально функционирующим приложением и работаете над паролями пользователей, следите за последними зафиксированными уязвимостями в данной области. Для более подробного ознакомления с темой защиты паролей можете просмотреть [следующую статью](https://crackstation.net/hashing-security.htm).

Код для Python 3.x

import uuid

import hashlib

def hash\_password(password):

    # uuid используется для генерации случайного числа

    salt = uuid.uuid4().hex

    return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ':' + salt

def check\_password(hashed\_password, user\_password):

    password, salt = hashed\_password.split(':')

    return password == hashlib.sha256(salt.encode() + user\_password.encode()).hexdigest()

new\_pass = input('Введите пароль: ')

hashed\_password = hash\_password(new\_pass)

print('Строка для хранения в базе данных: ' + hashed\_password)

old\_pass = input('Введите пароль еще раз для проверки: ')

if check\_password(hashed\_password, old\_pass):

    print('Вы ввели правильный пароль')

else:

    print('Извините, но пароли не совпадают')

Код для Python 2.x

import uuid

import hashlib

def hash\_password(password):

    # uuid используется для генерации случайного числа

    salt = uuid.uuid4().hex

    return hashlib.sha256(salt.encode() + password.encode()).hexdigest() + ':' + salt

def check\_password(hashed\_password, user\_password):

    password, salt = hashed\_password.split(':')

    return password == hashlib.sha256(salt.encode() + user\_password.encode()).hexdigest()

new\_pass = raw\_input('Введите пароль: ')

hashed\_password = hash\_password(new\_pass)

print('Строка для сохранения в базе данных: ' + hashed\_password)

old\_pass = raw\_input('Введите пароль еще раз для проверки: ')

if check\_password(hashed\_password, old\_pass):

    print('Вы ввели правильный пароль')

else:

    print('Извините, но пароли не совпадают')

**Задание.**

Выполнить собственную программную реализацию любой хеш функции.