Algorítmica y Programación

Enero - Mayo 2020

Dr. Iván S. Razo Zapata
(ivan.razo@itam.mx)

ITAM

Tuplas y diccionarios

Tuplas

- Ejemplo

```
In [1]: tupla = (1,2,3)

In [2]: tupla[0]
Out[2]: 1
```

```
In [3]: tupla[0] = 5
Traceback (most recent call last):

  File "<ipython-input-3-a2f896436084>", line 1, in <module>
    tupla[0] = 5

TypeError: 'tuple' object does not support item assignment
```

ITAM

Diccionarios

- Hash map o arreglos (matrices) asociativas

Hash

Clave1	Valor1
Clave2	Valor2
Clave3	Valor3
Clave4	Valor4

Diccionarios

- Hash map o arreglos (matrices) asociativas

Hash

Clave1	Entero
Clave2	Real
Clave3	Cadena de caracteres
Clave4	Lista, tupla

Diccionarios

- Valores a usar como hash (claves)
- Hashability
- Identificador único

```
In [1]: hash(45)
Out[1]: 45

In [2]: hash(34.5)
Out[2]: 115292150460684701O

In [3]: hash("Clave")
Out[3]: 7651671161965820483

In [4]: hash((2,3,4))
Out[4]: 37897050175964770S0
```

```
In [14]: hash([1,2,3])
Traceback (most recent call last):

  File "<ipython-input-14-35e31e935e9e>", line 1, in <module>
    hash([1,2,3])

TypeError: unhashable type: 'list'
```

Diccionarios

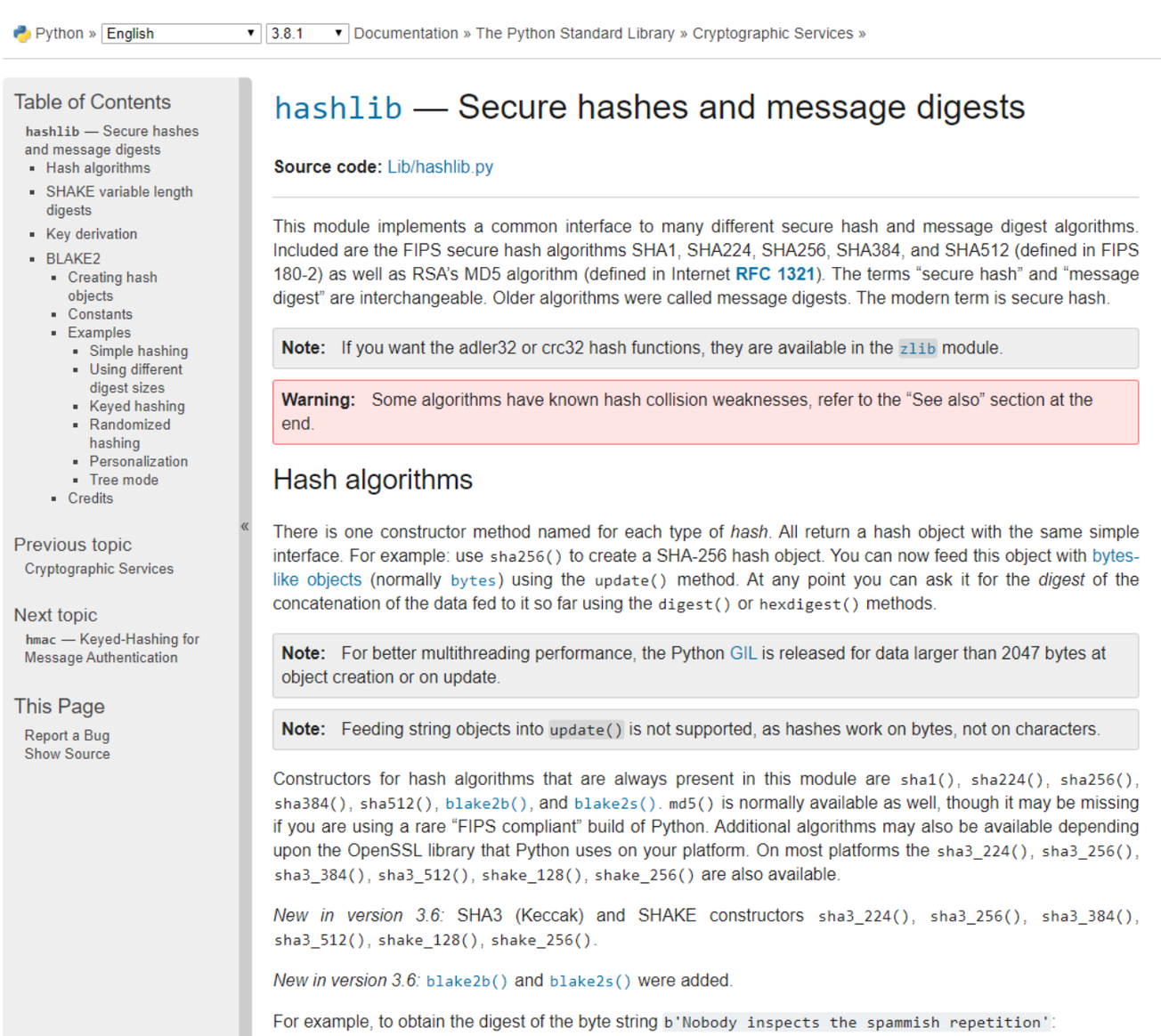
- Python implementa un hash básico por default

```
In [12]: hash(34.5001)
Out[12]: 1153152088907776034

In [13]: hash(34.5002)
Out[13]: 1153382673208688674

In [14]: hash(34.5003)
Out[14]: 1153613257509617698
```

Hashing



Python » English 3.8.1 Documentation » The Python Standard Library » Cryptographic Services »

Table of Contents

- **hashlib** — Secure hashes and message digests
  - Hash algorithms
  - SHAKE variable length digests
  - Key derivation
  - BLAKE2
    - Creating hash objects
    - Constants
    - Examples
      - Simple hashing
      - Using different digest sizes
      - Keyed hashing
      - Randomized hashing
      - Personalization
      - Tree mode
    - Credits

Previous topic: Cryptographic Services

Next topic: hmac — Keyed-Hashing for Message Authentication

This Page: Report a Bug, Show Source

hashlib — Secure hashes and message digests

Source code: Lib/hashlib.py

This module implements a common interface to many different secure hash and message digest algorithms. Included are the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512 (defined in FIPS 180-2) as well as RSA's MD5 algorithm (defined in Internet **RFC 1321**). The terms "secure hash" and "message digest" are interchangeable. Older algorithms were called message digests. The modern term is secure hash.

Note: If you want the adler32 or crc32 hash functions, they are available in the `zlib` module.

Warning: Some algorithms have known hash collision weaknesses, refer to the "See also" section at the end.

Hash algorithms

There is one constructor method named for each type of *hash*. All return a hash object with the same simple interface. For example: use `sha256()` to create a SHA-256 hash object. You can now feed this object with bytes-like objects (normally `bytes`) using the `update()` method. At any point you can ask it for the *digest* of the concatenation of the data fed to it so far using the `digest()` or `hexdigest()` methods.

Note: For better multithreading performance, the Python GIL is released for data larger than 2047 bytes at object creation or on update.

Note: Feeding string objects into `update()` is not supported, as hashes work on bytes, not on characters.

Constructors for hash algorithms that are always present in this module are `sha1()`, `sha224()`, `sha256()`, `sha384()`, `sha512()`, `blake2b()`, and `blake2s()`. `md5()` is normally available as well, though it may be missing if you are using a rare "FIPS compliant" build of Python. Additional algorithms may also be available depending upon the OpenSSL library that Python uses on your platform. On most platforms the `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, `shake_128()`, `shake_256()` are also available.

New in version 3.6: SHA3 (Keccak) and SHAKE constructors `sha3_224()`, `sha3_256()`, `sha3_384()`, `sha3_512()`, `shake_128()`, `shake_256()`.

New in version 3.6: `blake2b()` and `blake2s()` were added.

For example, to obtain the digest of the byte string `b'Nobody inspects the spammish repetition'`:

```
 7
 8 from hashlib import blake2b
 9
10 h = blake2b()
11 h.update(b"Frase de ejemplo")
12 print(h.hexdigest())
13
14 i = blake2b()
15 i.update(b"Frase de ejempl")
16 print(i.hexdigest())
17
```

```
e901a45bd19c0aeca59709a2f6296e220eb0c5d43d40b1f41aa8bcdbfe68b69afc9b6402d859e65fcbf82073d3cdb243223d63af3ee2146b0b5c06cc018c2a0d
2288a738f027824e1ca1d4bc25a727d5ef15d45520374a1ea5212ddef672344cce8c001b1d215aeaf14e18de0b4d80f53be7772bba8b1a4f79d54b8611f445ef
```

Hashing

```python
from hashlib import blake2b

h = blake2b()
h.update(b"Frase de ejemplo")
print(h.hexdigest())

i = blake2b()
i.update(b"Frase de ejempl")
print(i.hexdigest())

j = blake2b()
j.update(b"45.5001")
print(j.hexdigest())


k = blake2b()
k.update(b"45.5002")
print(k.hexdigest())

l = blake2b()
l.update(b"45.5003")
print(l.hexdigest())

```

```
e901a45bd19c0aeca59709a2f6296e220eb0c5d43d40b1f41aa8bcdbfe68b69afc9b6402d859e65fcbf82073d3cdb243223d63af3ee2146b0b5c06cc018c2a0d
2288a738f027824e1ca1d4bc25a727d5ef15d45520374a1ea5212ddef672344cce8c001b1d215aeaf14e18de0b4d80f53be7772bba8b1a4f79d54b8611f445ef
d48877836dc9a496375dfce5b718e9429fac6d522e36cb59686aa5722801567de54bb6a9285e7377bdd0803f914d36ef0ee4fb0b8bbcdd1eb3997d7d4aebf1ee
39313a12c2fbda4a4c2093cc24825400b191eff7250e2a6995ced8e71116da5544b36920dfacdbafe125d89ea026a64f4a8dbf4b28d3f88dfc2a6ebcaf408d3f
7aaf7ad5c76332cb1d5880d95cbed02cbd9e13d030b2620710d3a4aa9c88c68a1c07d0ca11ab37c0a126e034db042d2290615f819a043ce2747b0097fd4df0ef
```

Diccionarios - creación

```
In [2]: dict1 = {"X23": ['Nombre1',45,67.9,[2, 3]], "Y45": ['Nombre2',31,237.5,[5, 3.7]], "W12": ['Nombre3',10.50,49]}

In [3]: dict1
Out[3]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'W12': ['Nombre3', 10.5, 49]}

In [4]:
```

Diccionarios – accediendo a la información

```
In [6]: dict1["W12"]
Out[6]: ['Nombre3', 10.5, 49]

In [7]: dict1["X23"]
   ...:
Out[7]: ['Nombre1', 45, 67.9, [2, 3]]

In [8]: dict1["Y45"]
Out[8]: ['Nombre2', 31, 237.5, [5, 3.7]]

In [8]:

In [9]: dict1[0]
Traceback (most recent call last):

  File "<ipython-input-9-90e2046cfae3>", line 1, in <module>
    dict1[0]

KeyError: 0
```

Diccionarios – agregando información

```
In [11]: dict1["ABC"] = (4,5,6)

In [12]: dict1
Out[12]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'W12': ['Nombre3', 10.5, 49],
 'ABC': (4, 5, 6)}
```

```
In [13]: dict1["DEF"] = 2
    ...:

In [14]: dict1
Out[14]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'W12': ['Nombre3', 10.5, 49],
 'ABC': (4, 5, 6),
 'DEF': 2}
```

ITAM

Diccionarios – verificando contenido

```
In [26]: dict1
Out[26]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'W12': ['Nombre3', 10.5, 49],
 'ABC': (4, 5, 6),
 'DEF': 2}

In [27]: "W12" in dict1
Out[27]: True

In [28]: "W13" in dict1
Out[28]: False

In [28]:

In [29]: del dict1["W12"]

In [30]: dict1
Out[30]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'ABC': (4, 5, 6),
 'DEF': 2}
```

Diccionarios – eliminando contenido

```
In [29]: del dict1["W12"]

In [30]: dict1
Out[30]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'ABC': (4, 5, 6),
 'DEF': 2}

In [31]: del dict1["W12"]
Traceback (most recent call last):

  File "<ipython-input-31-431ef3229d0b>", line 1, in <module>
    del dict1["W12"]

KeyError: 'W12'
```

Diccionarios – claves, valores, y pares (ítems)

```
In [50]: dict1
Out[50]:
{'X23': ['Nombre1', 45, 67.9, [2, 3]],
 'Y45': ['Nombre2', 31, 237.5, [5, 3.7]],
 'ABC': (4, 5, 6),
 'DEF': 2}

In [51]: dict1.keys()
Out[51]: dict_keys(['X23', 'Y45', 'ABC', 'DEF'])

In [52]: dict1.values()
Out[52]: dict_values([['Nombre1', 45, 67.9, [2, 3]], ['Nombre2', 31, 237.5, [5, 3.7]], (4, 5, 6), 2])

In [53]: dict1.items()
Out[53]: dict_items([('X23', ['Nombre1', 45, 67.9, [2, 3]]), ('Y45', ['Nombre2', 31, 237.5, [5, 3.7]]), ('ABC', (4, 5, 6)), ('DEF', 2)])
```

ITAM

Diccionarios – comprensión

Crear diccionarios a partir de otros diccionarios

```
In [54]: dict2 = {k:type(v) for (k,v) in dict1.items()}

In [55]: dict2
Out[55]: {'X23': list, 'Y45': list, 'ABC': tuple, 'DEF': int}
```

Diccionarios – comprensión

Crear diccionarios a partir de otros diccionarios

```
In [57]: dict3 = {'a':2, 'b':3, 'c':4}

In [58]: dict4 = {k:v**3 for (k,v) in dict3.items()}

In [59]: dict4
Out[59]: {'a': 8, 'b': 27, 'c': 64}

In [60]:
```

También funciona con listas

```
In [60]: lista1 = [2,3,4]

In [61]: lista2 = [v**3 for v in lista1]

In [62]: lista2
Out[62]: [8, 27, 64]
```

Diccionarios – comprensión

Crear diccionarios a partir de listas

```
import numpy as np

list1 = ["A","B","C","D","E"]
list2 = [10,18,12,24,20]

dict1 = dict(zip(list1,list2))

media = np.array(list(dict1.values())).mean()

dict2 = {k:v for (k,v) in dict1.items() if v > media}

print(dict2)
```

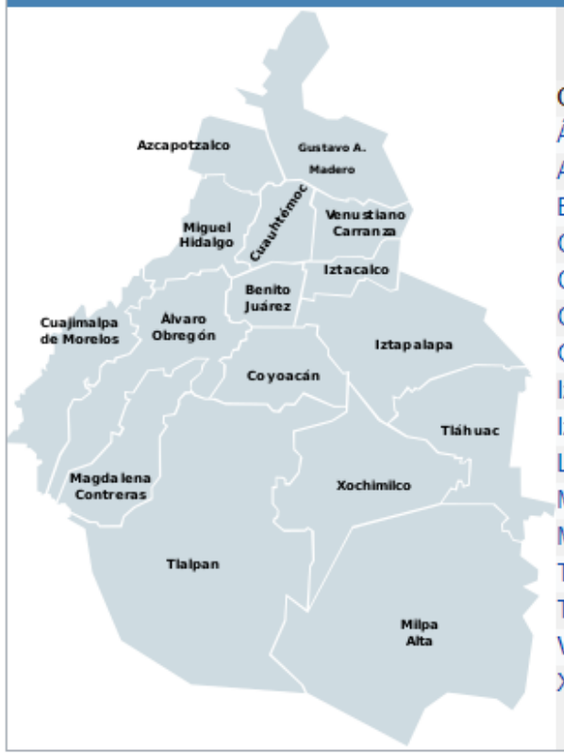
Diccionarios - Ejercicios

- Crear un diccionario en base a dos listas
  lista1 = nombres
  lista2 = población y superficie

- Crear un diccionario con alcaldías cuya población sea mayor a la media

- Crear un diccionario con alcaldías cuya superficie sea al menos el doble de la superficie menor

Demarcaciones territoriales de la Ciudad de México



Demarcaciones territoriales	Población (2010)	Superficie (km²)
Ciudad de México	**8 851 080**	**1 479,00**
Álvaro Obregón	727 034	96,17
Azcapotzalco	414 711	33,66
Benito Juárez	385 439	26,63
Coyoacán	620 416	54,40
Cuajimalpa	186 391	74,58
Cuauhtémoc	531 831	32,40
Gustavo A. Madero	1 185 772	94,07
Iztacalco	384 326	23,30
Iztapalapa	1 815 786	117,00
La Magdalena Contreras	239 086	74,58
Miguel Hidalgo	372 889	46,99
Milpa Alta	130 582	228,41
Tláhuac	360 265	85,34
Tlalpan	650 567	340,07
Venustiano Carranza	430 978	33,40
Xochimilco	415 007	118,00

Fuente: INEGI[8]

Diccionarios - Ejercicios

```
 8 import numpy as np
 9
10 list1 = ["Álvaro Obregón", "Azcapotzalco", "Benito Juárez",
11         "Coyoacán","Cuajimalpa", "Cuauhtémoc", "Gustavo A. Madero",
12         "Iztacalco", "Iztapalapa", "La Magdalena Contreras",
13         "Miguel Hidalgo", "Milpa Alta", "Tláhuac", "Tlalpan",
14         "Venustiano Carranza", "Xochimilco"]
15
16 list2 = [[727034,   96.17], [414711,    33.66],
17          [385439,   26.63], [620416,    54.40],
18          [186391,   74.58], [531831,    32.40],
19          [1185772,  94.07], [384326,    23.30],
20          [1815786,  117.00], [239086,   74.58],
21          [372889,   46.99], [130582,    228.41],
22          [360265,   85.34], [650567,    340.07],
23          [430978,   33.40], [415007,    118.00],
24         ]
25
26 dict1 = dict(zip(list1,list2))
27
28 list3 = list(dict1.values())
29 list4 = []
30 list5 = []
31
32 for item in list3:
33     list4.append(item[0])
34     list5.append(item[1])
35
36
37 media  = np.array(list4).mean()
38 minimo = np.array(list5).min()
39
40 dict2 = {k:v for (k,v) in dict1.items() if v[0] > media}
41 dict3 = {k:v for (k,v) in dict1.items() if v[1] >= (2*minimo)}
42
43 print(media)
44 print(dict2)
45 print(minimo)
46 print(dict3)
47
```

Salida

```
553192.5
{'Álvaro Obregón': [727034, 96.17], 'Coyoacán': [620416, 54.4], 'Gustavo A.
Madero': [1185772, 94.07], 'Iztapalapa': [1815786, 117.0], 'Tlalpan':
[650567, 340.07]}
23.3
{'Álvaro Obregón': [727034, 96.17], 'Coyoacán': [620416, 54.4],
'Cuajimalpa': [186391, 74.58], 'Gustavo A. Madero': [1185772, 94.07],
'Iztapalapa': [1815786, 117.0], 'La Magdalena Contreras': [239086, 74.58],
'Miguel Hidalgo': [372889, 46.99], 'Milpa Alta': [130582, 228.41],
'Tláhuac': [360265, 85.34], 'Tlalpan': [650567, 340.07], 'Xochimilco':
[415007, 118.0]}

In [96]:
```