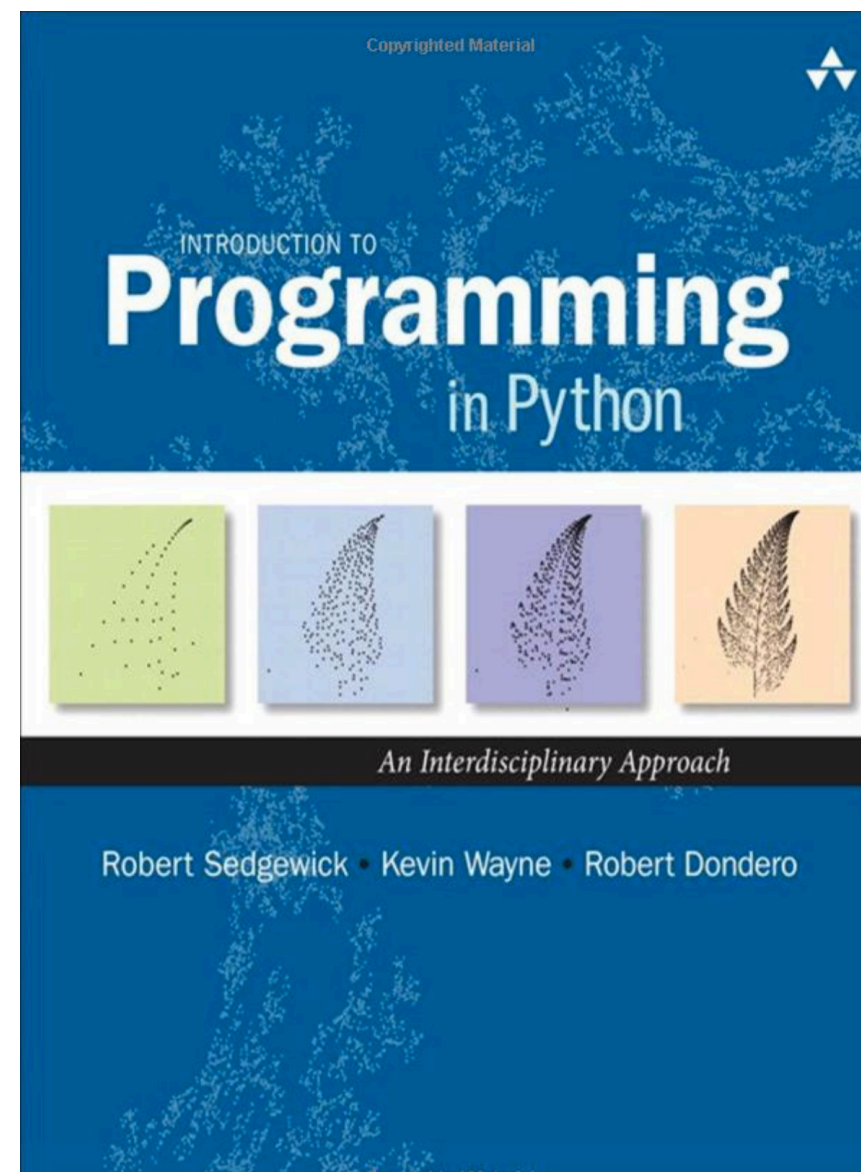


Parte I: Intro pensamiento computacional

Clase 08: Estructuras de datos

dict, set y tuple

Diego Caro
dcaro@udd.cl



Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

Disponible en <https://introcs.cs.princeton.edu/python>

Outline

- API: Application Programming Interface
- Tabla de símbolos: `dict()`
- Conjunto de elementos: `set()`
 - Conjunto de elementos inmutable: `frozenset()`
- Lista inmutable: `tuple()`
- Rendimiento

¿Qué hace este código?

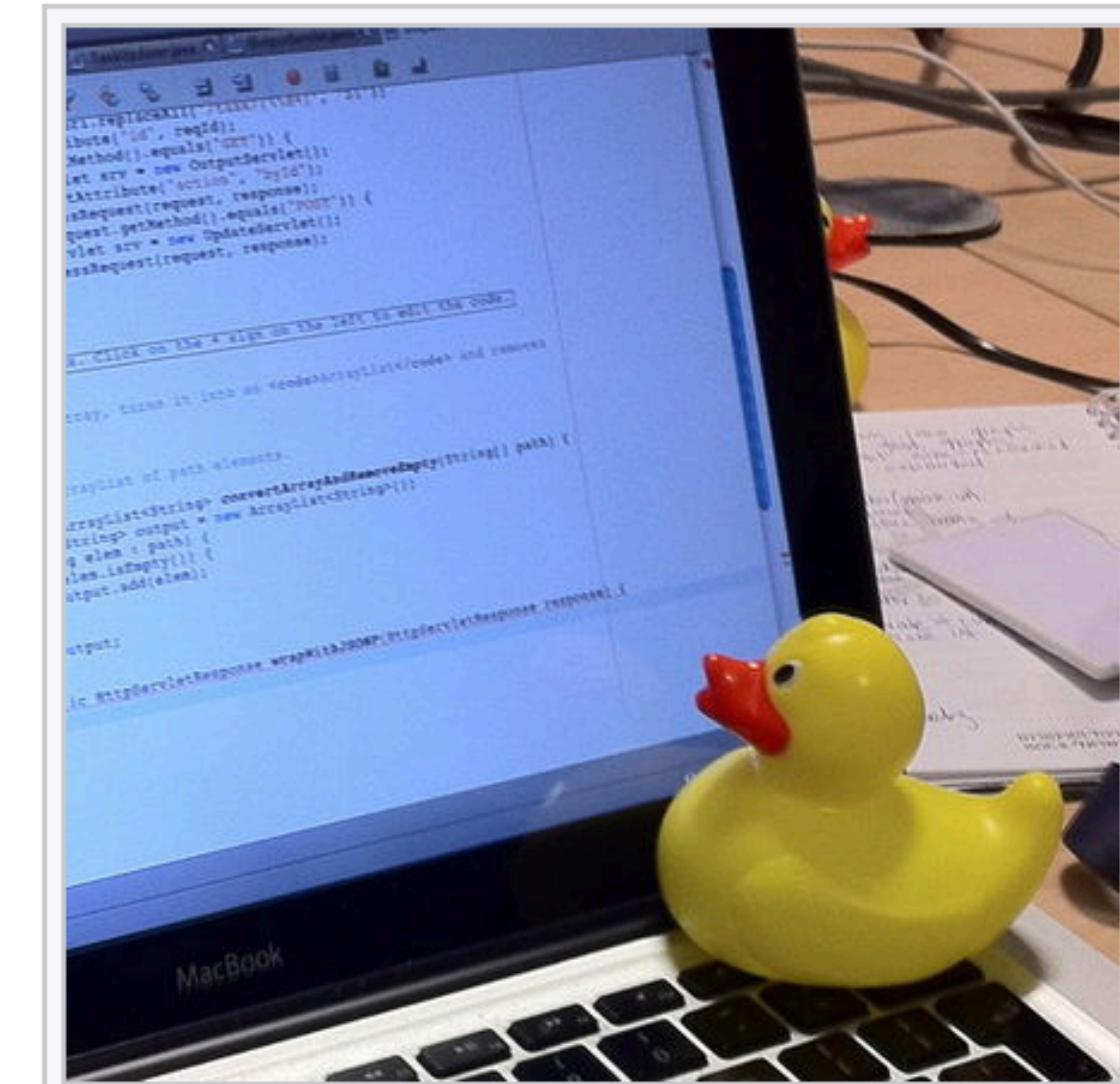
```
1 def f(a):  
2     a[2] = 'a'  
3  
4 x = "hola"  
5 f(x)  
6 print(x)  
7  
8 x = [5, 7, 11, 42]  
9 f(x)  
10 print(x)
```



Método de depuración del patito de goma

El **método de depuración del patito de goma** es un término informal utilizado en [ingeniería de software](#) para describir un método de [revisión de código](#). El nombre es una referencia a una historia del libro *The Pragmatic Programmer* en donde un programador toma un [patito de goma](#) y revisa su código forzándose a sí mismo a explicarlo, línea por línea, al pato.¹ Existen otros muchos términos para esta técnica, que a menudo tienen que ver con objetos inanimados.

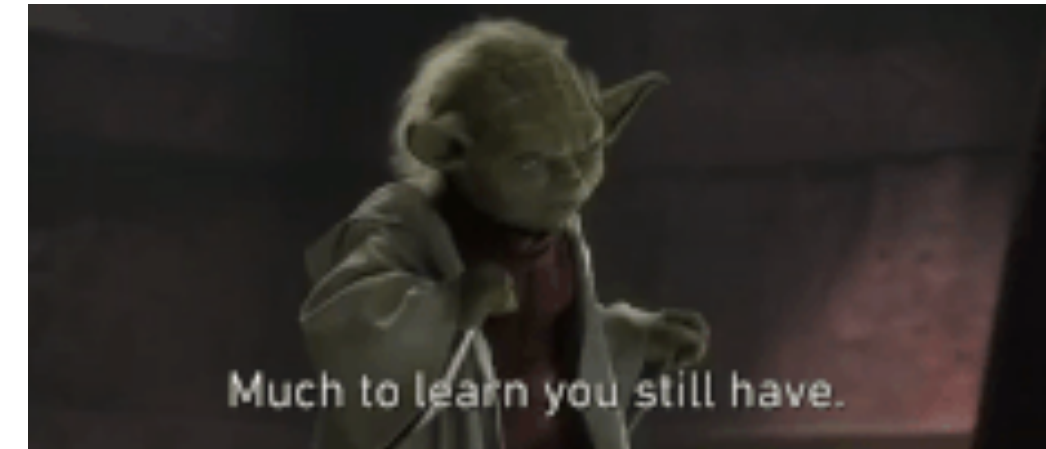
Muchos programadores han tenido la experiencia de explicar un problema de programación a alguien más, posiblemente a alguien que no sabe nada sobre programación, y encontrar la solución en el proceso de explicar el problema. Al comparar lo que supuestamente hace el código con lo que hace en realidad, cualquier incongruencia resulta evidente.² Usando un objeto inanimado, el programador puede tratar de lograr el mismo efecto sin tener que hablar con otra persona.



El patito de goma es usado por un desarrollador para ayudar en la [revisión de código](#)

https://es.wikipedia.org/wiki/M%C3%A9todo_de_depuraci%C3%B3n_del_patito_de_goma

Sobre la ayudantía: Listas anidadas



Entregar la ayudantía deben!

```
1 from random import random
2
3 rows = 3
4 cols = 4
5
6 matrix = [ [0]*cols ]*rows
7
8 print(matrix)
9
10 def fillrandom(matrix, rows, cols):
11     for i in range(rows):
12         for j in range(cols):
13             matrix[i][j] = random()
14
15 fillrandom(matrix, rows, cols)
16 print(matrix)
```

número de
columnas

número de
filas



número de
filas

número de
columnas

matrix[0] → [0, 0, 0, 0]

matrix[1] → [0, 0, 0, 0]

matrix[2] → [0, 0, 0, 0]

| | | | |
|--------------|--------------|--------------|--------------|
| matrix[0][0] | matrix[0][1] | matrix[0][2] | matrix[0][3] |
| matrix[1][0] | matrix[1][1] | matrix[1][2] | matrix[1][3] |
| matrix[2][0] | matrix[2][1] | matrix[2][2] | matrix[2][3] |

Ejercicio: Crea una función que devuelve la transpuesta de una matriz de tamaño $m \times n$. La función debe devolver una nueva matriz $n \times m$.

API: Interfaz de programación de aplicaciones

- API: conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.
- Objetivo: utilizar software que ya ha sido desarrollado, reutilizar código. Debe estar bien documentada.
- Ya hemos visto varias!
 - módulo random
 - <https://docs.python.org/3/library/random.html>
 - <https://github.com/python/cpython/blob/3.7/Lib/random.py>
 - tipos de datos: str, list
 - <https://docs.python.org/3/library/stdtypes.html>

| <i>operation</i> | <i>description</i> |
|----------------------------------|------------------------------------------------------------------------------------------------------------------|
| <code>len(s)</code> | <i>length of s</i> |
| <code>s + t</code> | <i>a new string that is the concatenation of s and t</i> |
| <code>s += t</code> | <i>assign to s a new string that is the concatenation of s and t</i> |
| <code>s[i]</code> | <i>the ith character of s (a string)</i> |
| <code>s[i:j]</code> | <i>the ith through (j-1)st characters of s (i defaults to 0; j defaults to len(s))</i> |
| <code>s[i:j:k]</code> | <i>slice from i to j with step size k</i> |
| <code>s < t</code> | <i>is s less than t?</i> |
| <code>s <= t</code> | <i>is s less than or equal to t?</i> |
| <code>s == t</code> | <i>is s equal to t?</i> |
| <code>s != t</code> | <i>is s not equal to t?</i> |
| <code>s >= t</code> | <i>is s greater than or equal to t?</i> |
| <code>s > t</code> | <i>is s greater than t?</i> |
| <code>s in t</code> | <i>does s appear as a substring in t?</i> |
| <code>s not in t</code> | <i>does s not appear as a substring in t?</i> |
| <code>s.count(t)</code> | <i>number of occurrences of substring t in s</i> |
| <code>s.find(t, start)</code> | <i>the first index in s where t appears (-1 if not found) starting at start (default 0)</i> |
| <code>s.upper()</code> | <i>a copy of s with lowercase letters replaced with uppercase ones</i> |
| <code>s.lower()</code> | <i>a copy of s with uppercase letters replaced with lowercase ones</i> |
| <code>s.startswith(t)</code> | <i>does s start with t?</i> |
| <code>s.endswith(t)</code> | <i>does s end with t?</i> |
| <code>s.strip()</code> | <i>a copy of s with leading and trailing whitespace removed</i> |
| <code>s.replace(old, new)</code> | <i>a copy of s with all occurrences of old replaced by new</i> |
| <code>s.split(delimiter)</code> | <i>an array of substrings of s, separated by delimiter (default whitespace)</i> |
| <code>delimiter.join(a)</code> | <i>concatenation of strings in a[], separated by delimiter</i> |


Partial API for Python's built-in str data type

dict(): tabla de símbolos


- dict() es un diccionario que asocia un valor a una clave.
- También se puede usar llaves {key:value} para crear un diccionario.

| Dominio | Dirección IP |
|-------------------|-----------------|
| udd.cl | 201.221.123.142 |
| ingenieria.udd.cl | 201.221.123.142 |
| google.com | 64.233.190.101 |
| www.pokemongo.com | 13.33.131.6 |

Llave **Valor**

diccionario vacío

```
1 ipaddress = dict()
2 ipaddress['udd.cl'] = '201.221.123.142'
3 ipaddress['ingenieria.udd.cl'] = '201.221.123.142'
4 ipaddress['google.com'] = '64.233.190.101'
5 ipaddress['www.pokemongo.com'] = '13.33.131.6'
6
7 print('Dirección IP de udd.cl:', ipaddress['udd.cl'])
```

diccionario inicializado con dos llaves

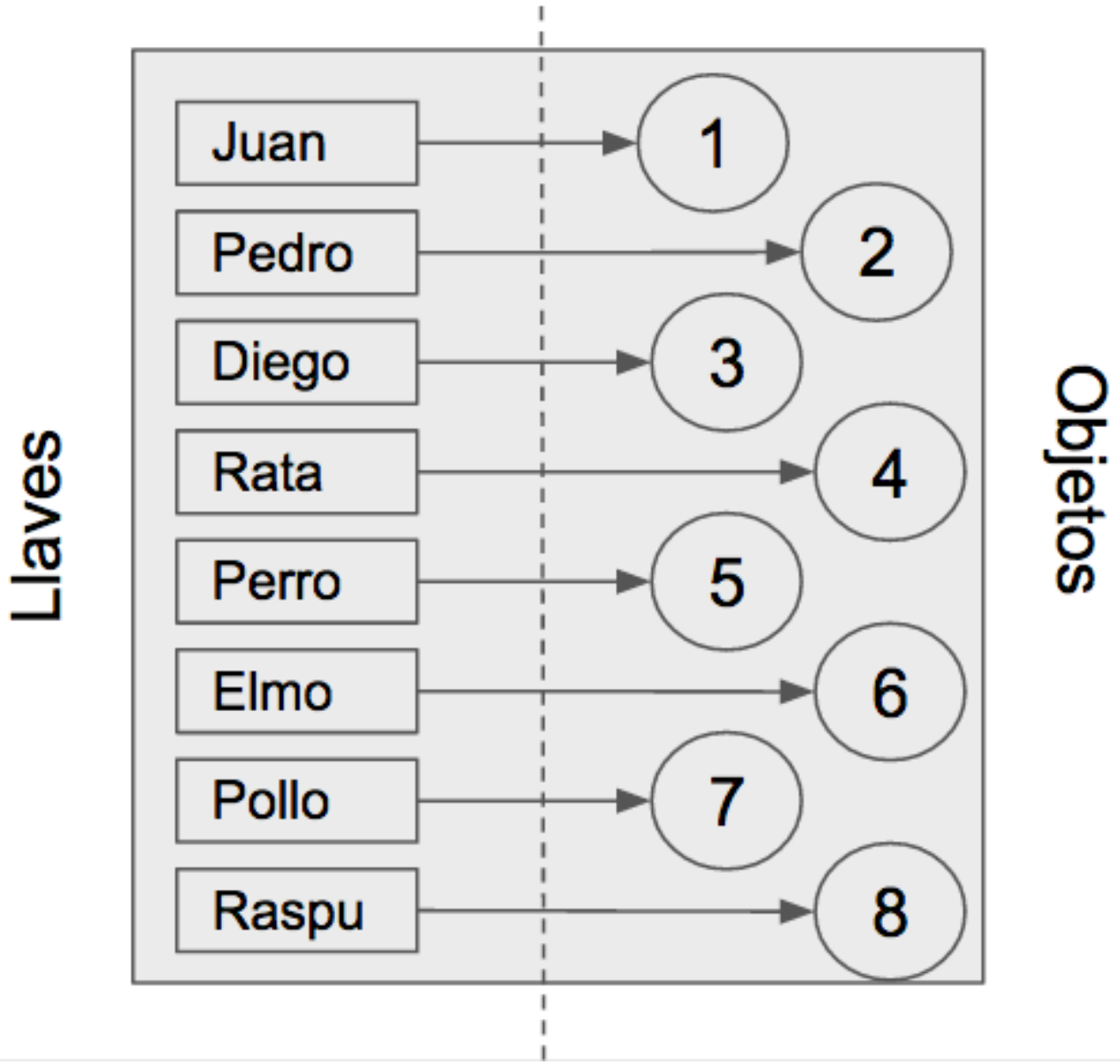
```
1 ipaddress = {'udd.cl': '201.221.123.142',
2             'google.com': '64.233.190.101'}
3 ipaddress['ingenieria.udd.cl'] = '201.221.123.142'
4 ipaddress['www.pokemongo.com'] = '13.33.131.6'
5
6 print('Dirección IP de udd.cl:', ipaddress['udd.cl'])
```

API dict()

- El único requisito para los diccionarios es que la llave debe ser **inmutable**.
 - Tipos inmutables: los que no se pueden modificar una vez creados. Ejemplo: int, str, tuple, frozenset.

class es sinónimo
de tipo de dato
(ya veremos más
sobre esto)

| | | |
|----------|------------------------|-----------------------------------------------------------------------------------------------------------------|
| | class dict | |
| | dict() | |
| int | len(d) | Número de entradas en diccionario |
| value | d[key] | Obtiene valor de llave key |
| | d[key] = value | Asigna value a llave key |
| | del d[key] | Elimina llave key del diccionario |
| | clear() | Elimina todas las llaves del diccionario |
| value | get(key[, default]) | Retorna el valor de la llave key, si no retorna default. Si el argumento default no es entregado, retorna None. |
| sequence | items() | Retorna una secuencia pares (key, value) con los elementos del diccionario. |
| sequence | keys() | Retorna una secuencia con las llaves |
| sequence | values() | Retorna una secuencia con los valores |



Más info en <https://docs.python.org/3/library/stdtypes.html?highlight=dict#dict>

Aplicaciones

| <i>application</i> | <i>key</i> | <i>value</i> |
|---------------------|----------------|-----------------------|
| contacts | name | phone number, address |
| credit card | account number | transaction details |
| file share | name of song | computer ID |
| dictionary | word | definition |
| web search | keyword | list of web pages |
| book index | word | list of page numbers |
| cloud storage | file name | file contents |
| domain name service | domain name | IP address |
| reverse DNS | IP address | domain name |
| compiler | variable name | value and type |
| internet routing | destination | best route |
| ... | ... | ... |

Aplicación: contar palabras

```
1 def word_count(message):
2     counts = dict()
3     words = message.split()
4
5     for word in words:
6         if word in counts:
7             counts[word] += 1
8         else:
9             counts[word] = 1
10
11     return counts
12
13
14 mambo = '''Desde lima vengo a mi machaguay
15 Desde Lima vengo a mi machaguay
16 A bailar el mambo de mi machaguay
17 A bailar el mambo de mi machaguay'''
18
19 print(word_count(mambo))
```

```
{'A': 2,
'Desde': 2,
'Lima': 1,
'a': 2,
'bailar': 2,
'de': 2,
'el': 2,
'lima': 1,
'machaguay': 4,
'mambo': 2,
'mi': 4,
'vengo': 2}
```

Q: ¿Existe algo que se pueda mejorar en este código para contar palabras?

set(): conjunto de elementos

- set() es una colección no-ordenada, elementos se pueden repetir sólo una vez.
 - No podemos acceder a la i-ésima posición (como en las listas).
 - Se puede crear con el operador corchete {elem1, elem2}
 - Elementos deben ser inmutables.
- Optimizado para:
 - chequear si existe un elemento
 - unión, intersección, diferencia
 - y eliminar duplicados de una secuencia

```
A: {1, 2, 3}
A.add(99): {99, 1, 2, 3}
A.remove(2): {99, 1, 3}
B: {'hola', 1, 2, 3}
A ∩ B: {1, 3}
A ∩ B: {1, 3}
A ∪ B: {'hola', 1, 2, 3, 99}
A ∪ B: {'hola', 1, 2, 3, 99}
A - B: {99}
A - B: {99}
```

```
1 A = {1, 2, 2, 3}
2 print('A:', A)
3
4 print('2 in A?:', 2 in A)
5
6 A.add(99)
7 print('A.add(99):', A)
8
9 A.remove(2)
10 print('A.remove(2):', A)
11
12 B = set([1, 2, 3, 'hola'])
13 print('B:', B)
14
15 print('A ∩ B:', A.intersection(B))
16 print('A ∩ B:', A & B)
17
18 print('A ∪ B:', A.union(B))
19 print('A ∪ B:', A | B)
20
21 print('A - B:', A.difference(B))
22 print('A - B:', A - B)
```

tuple(): secuencia inmutable

- tuple() es una secuencia inmutable, es decir, que no se puede actualizar una vez creada.
 - Es "*lo mismo*" que una lista que no se puede modificar.
 - Se pueden crear usando paréntesis (elem1, elem2)
- Son usadas en casos donde queremos representar una secuencia en un dict() o en un set().



```
1 A = {[1, 2, 3, 4], list(['a', 'b', 'c'])}
2 print('A:', A)
3 print('(1, 2, 3, 4) in A?:', (1, 2, 3, 4) in A)
```

```
$ python3 badset.py
Traceback (most recent call last):
  File "badset.py", line 1, in <module>
    A = {[1, 2, 3, 4], ['a', 'b', 'c']}
TypeError: unhashable type: 'list'
```



```
1 A = {(1, 2, 3, 4), tuple(['a', 'b', 'c'])}
2 print('A:', A)
3 print('(1, 2, 3, 4) in A?:', (1, 2, 3, 4) in A)
```

```
$ python3 tupleset.py
A: {(1, 2, 3, 4), ('a', 'b', 'c')}
(1, 2, 3, 4) in A?: True
```


Resumen

Conceptos

- Immutable:** colección/variable que no se puede modificar una vez creada.

| | | | | |
|--------|----------|---------|----------|--------|
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

https://docs.python.org/3/reference/lexical_analysis.html

Estructuras de datos

- `dict()`: tabla de símbolos clave:valor
- `set()`: conjunto de elementos, no-ordenada
- `tuple()`: colección inmutable y ordenada
- `frozenset()`: versión inmutable de `set()`

| | | Built-in Functions | | |
|----------------------------|--------------------------|---------------------------|---------------------------|-----------------------------|
| <code>abs()</code> | <code>delattr()</code> | <code>hash()</code> | <code>memoryview()</code> | <code>set()</code> |
| <code>all()</code> | <code>dict()</code> | <code>help()</code> | <code>min()</code> | <code>setattr()</code> |
| <code>any()</code> | <code>dir()</code> | <code>hex()</code> | <code>next()</code> | <code>slice()</code> |
| <code>ascii()</code> | <code>divmod()</code> | <code>id()</code> | <code>object()</code> | <code>sorted()</code> |
| <code>bin()</code> | <code>enumerate()</code> | <code>input()</code> | <code>oct()</code> | <code>staticmethod()</code> |
| <code>bool()</code> | <code>eval()</code> | <code>int()</code> | <code>open()</code> | <code>str()</code> |
| <code>breakpoint()</code> | <code>exec()</code> | <code>isinstance()</code> | <code>ord()</code> | <code>sum()</code> |
| <code>bytearray()</code> | <code>filter()</code> | <code>issubclass()</code> | <code>pow()</code> | <code>super()</code> |
| <code>bytes()</code> | <code>float()</code> | <code>iter()</code> | <code>print()</code> | <code>tuple()</code> |
| <code>callable()</code> | <code>format()</code> | <code>len()</code> | <code>property()</code> | <code>type()</code> |
| <code>chr()</code> | <code>frozenset()</code> | <code>list()</code> | <code>range()</code> | <code>vars()</code> |
| <code>classmethod()</code> | <code>getattr()</code> | <code>locals()</code> | <code>repr()</code> | <code>zip()</code> |
| <code>compile()</code> | <code>globals()</code> | <code>map()</code> | <code>reversed()</code> | <code>__import__()</code> |
| <code>complex()</code> | <code>hasattr()</code> | <code>max()</code> | <code>round()</code> | |

<https://docs.python.org/3/library/functions.html>