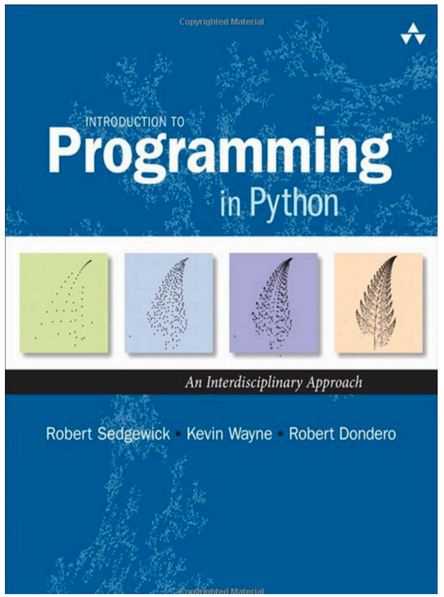
Taller de Programación

Clase 11: Tuplas, sets y diccionarios

Daniela Opitz
dopitz@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

- Listas anidadas
- Tuplas
- Sets
- Diccionarios

Listas Anidadas

- Una lista anidada es una lista que aparece como un elemento en otra lista
- Son útiles para representar matrices
- Una matriz A = $\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ por ejemplo en python quedaría así:

```
[[a11,a12,a13], [a21,a22,a23], [a31,a32,a33]]
```

Listas Anidadas

```python
from random import random

rows = 3
cols = 4
matrix =[]

for i in range(rows):
    matrix.append([0]*cols)

print(matrix)
```

Número de filas

Número de columnas

```python
def fillrandom(matrix, rows, cols):
    for i in range(rows):
        for j in range(cols):
            matrix[i][j] = random()

fillrandom(matrix, rows, cols)
print(matrix)
```

Matriz de ceros

Número de filas

```
matrix[0] ⟶ [0, 0, 0, 0]
matrix[1] ⟶ [0, 0, 0, 0]
matrix[2] ⟶ [0, 0, 0, 0]
```

Número de columnas

Listas Anidadas

- Cómo llamo a cada elemento de la lista anidada en python?

`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]`

`tuple()`

- Una tupla es una secuencia de valores agrupados. Sirve para agrupar en un único valor, valores que deben ir juntos.

- El tipo de datos que representa una dupla se llama **`tuple().`**

- Las tuplas se crean con valores separados por comas y entre paréntesis (elem1, elem2)

- La estructura de datos **`tuple()`** es parecida a estructura de lista pero a diferencia de esta, **`tuple()`** es **INMUTABLE**.

Ejemplo

Supongamos que estás desarrollando un programa en Python para administrar una pequeña tienda de ropa. En el programa, necesitas almacenar información sobre los productos que se venden en la tienda, como su nombre, precio y stock disponible. Una forma de hacerlo es utilizar tuplas.

```
producto1 = ("Camiseta blanca", 15.99, 50)
```

Luego, puedes almacenar varias tuplas como esta en una lista para representar todos los productos disponibles en la tienda:

```
productos = [("Camiseta blanca", 15.99, 50), ("Camiseta negra", 17.99,
30),    ("Pantalón vaquero", 29.99, 20),    ("Vestido floreado", 39.99,
10)]
```

De esta manera, puedes acceder a la información de cada producto utilizando los índices de la tupla. Por ejemplo, para obtener el precio del segundo producto en la lista, puedes hacer

```
precio = productos[1][1]
```

`set()`

- Un set es una colección **no ordenada ni indexada** de elementos.
- Los elementos son **únicos** y no se pueden repetir.
- El tipo de datos que representa un set es **`set()`**
- Los sets pueden crear con el operador corchete **`{elem1, elem2}`**
- Los sets son **mutables** pero **no pueden contener elementos mutables**
- Sirven para:
    - chequear si existe un elemento
    - realizar operaciones de unión, intersección, diferencia entre conjuntos
    - eliminar duplicados de una secuencia

Tuplas y sets


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

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

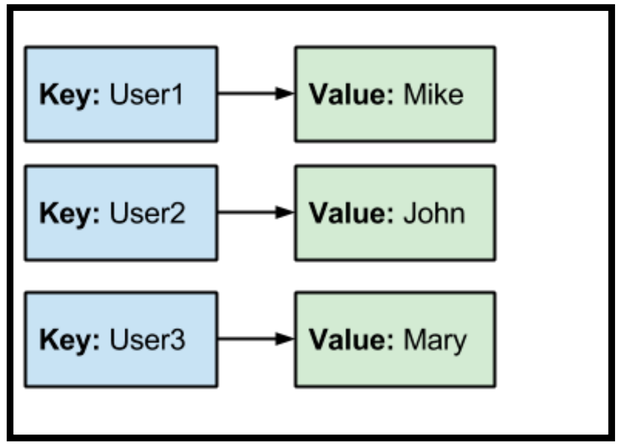
```
$ python3 tupleset.py
A: {(1, 2, 3, 4), ('a', 'b', 'c')}
(1, 2, 3, 4) in A?: True
```

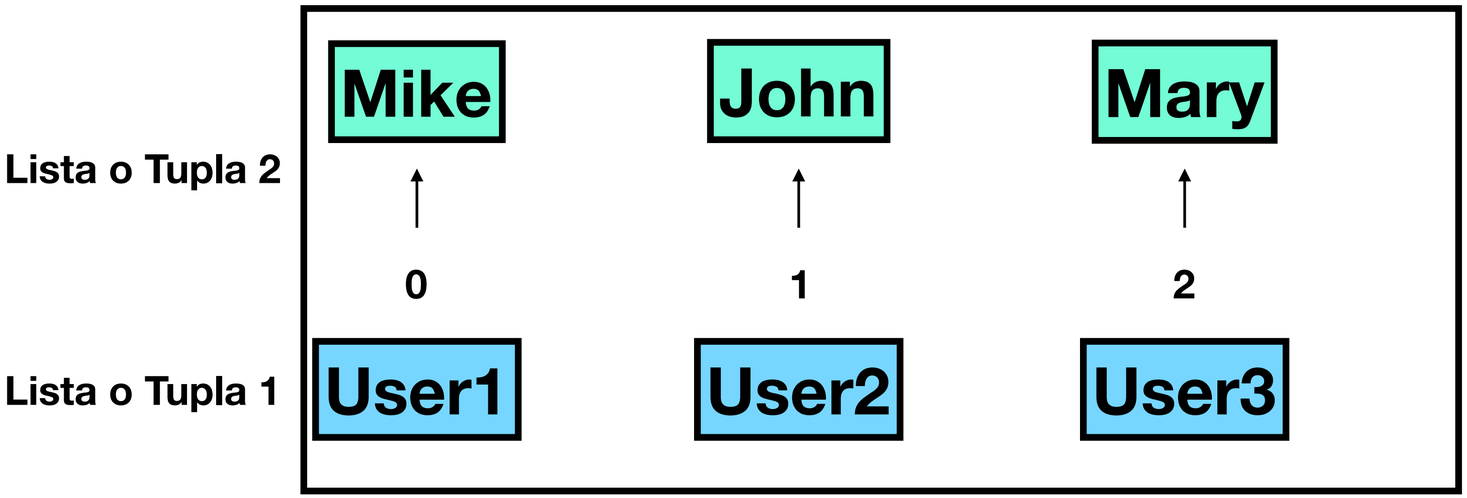
API disponible en https://docs.python.org/3/library/stdtypes.html#tuples

`dict()`

- Un diccionario es una colección de datos que no esta ordenada pero está indexada
- El tipo de datos de los diccionarios es **`dict()`**
- Un diccionario asocia un valor a una clave (key)
- La clave debe ser inmutable (de tipo int(), str(), tuple())

Diccionario





Ejemplo: ¿Cómo almacenar las notas de un curso?

- Podríamos usar una lista para nombre de alumno, notas y el curso:

```
nombres = ['Diego', 'Francisca', 'Loreto', 'Leo']
notas = [4.1, 5.5, 6.8, 3.9]
```

- Cada lista contiene información distinta.

- Las listas deben ser del **mismo tamaño**.

- Información entre las listas deben estar en la misma **posición**.

Extraído desde slides prof. Eric Grimson https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/lecture-slides-code/MIT6_0001F16_Lec6.pdf

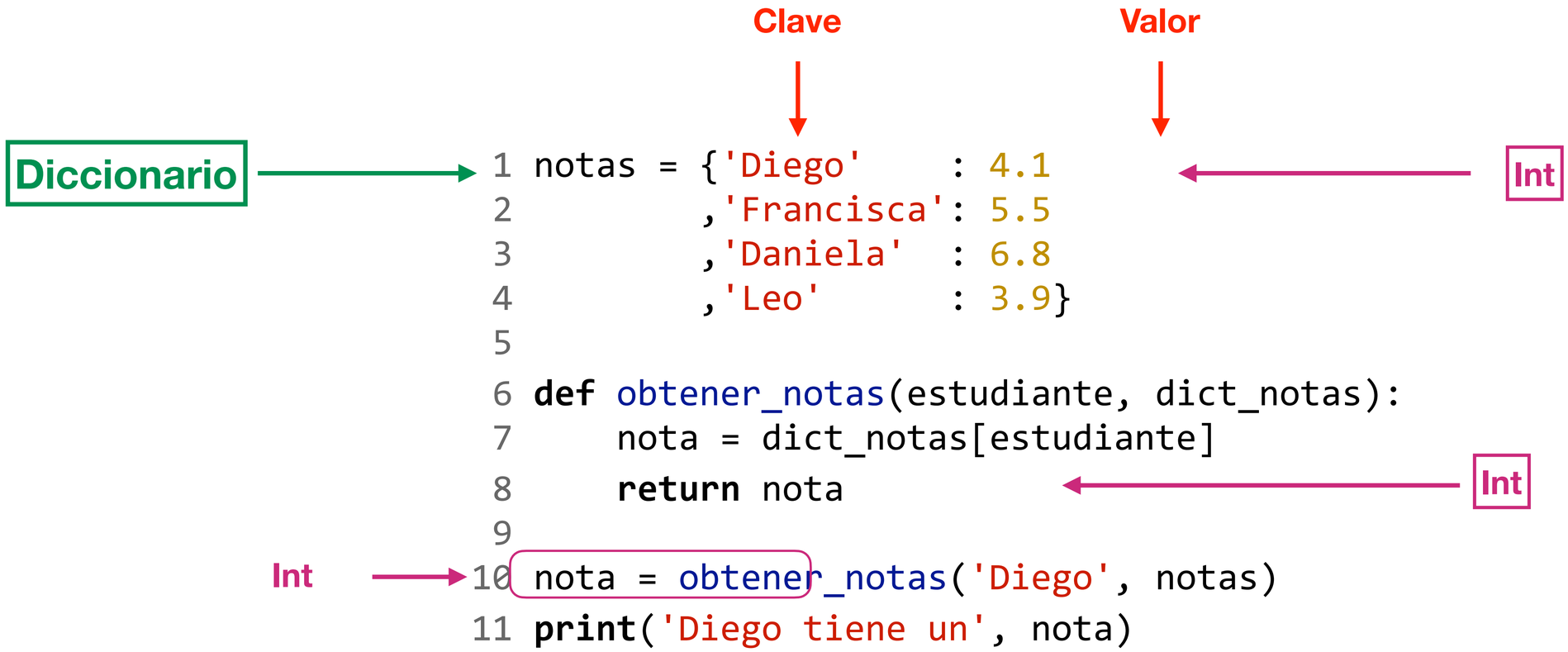
Ejemplo: ¿Cómo almacenar las notas de un curso?

Nota de Diego?

```
def obtener_notas(estudiante, lista_nombres, lista_notas):
    i = lista_nombres.index(estudiante)
    nota = lista_notas[i]

    return nota

nota = obtener_notas('Diego', nombres, notas)

print('Diego tiene un', nota)
```

- Complicado si tienes varios tipos de información que almacenar
- Debes mantener **varias listas**, y pasarlas como argumento
- Necesita de un í**ndice** (un **entero** con la posición)
- **DIFICIL DE MANTENER!**

Ejemplo: ¿Cómo almacenar las notas de un curso?



Patrón típico de uso diccionario

- Recorrer todas las llaves del diccionario:

```
for llave in notas.keys():
    print(llave)
```

- Recorrer los valores del diccionario:

```
for valor in notas.values():
    print(valor)
```

- Recorrer todas las llaves y valores:

```
for llave, valor in notas.items():
    print(llave, valor)
```

```
notas = {'Diego'    : 4.1
        ,'Francisca': 5.5
        ,'Daniela'  : 6.8
        ,'Leo'      : 3.9}
```

Patrón típico de uso diccionario

Sea el diccionario d cuyas claves corresponden a lenguajes de programación y sus valores al año en que fueron creados.

```
d = {"Python": 1991, "C": 1972, "Java": 1996}
```

Imprimir las llaves

```
for key in d:
    print(key)
```

```
Python
C
Java
```

Imprimir los valores

```
for value in d.values():
    print(value)
```

```
1991
1972
1996
```

Patrón típico de uso diccionario

```
d = {"Python": 1991, "C": 1972, "Java": 1996}
```

```
for llave, valor in d.items():
        print(llave, valor)
```

Imprimir llaves y valores

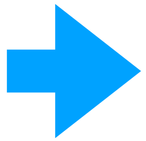
```
Python, 1991
C, 1972
Java, 1996
```

Aplicaciones de los diccionarios

application	*key*	*value*
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
...	...	...

dict()

Dominio	Dirección IP
udd.cl	201.221.123.142
ingenieria.udd.cl	201.221.123.142
google.com	64.233.190.101
pokemongo.com	13.33.131.6
llave (key)	**valor**



```
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['pokemongo.com'] = '13.33.131.6'
6
7 print('Dirección IP de udd.cl:', ipaddress['udd.cl'])
```

```
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['pokemongo.com'] = '13.33.131.6'
5
6 print('Dirección IP de udd.cl:', ipaddress['udd.cl'])
```

Ejemplo: Contando Palabras

```python
def word_count(message):
    counts = dict()
    words = message.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    return counts


mambo = '''Desde lima vengo a mi machaguay
Desde Lima vengo a mi machaguay
A bailar el mambo de mi machaguay
A bailar el mambo de mi machaguay'''

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}
```

Resumen

Estructuras de datos

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

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

Conceptos

- **Inmutable:** que no se puede modificar una vez creada.

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

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