

Taller de Programación

Clase 14: Diccionarios, APIs y Módulos

Daniela Opitz, Diego Caro
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

- Casos de uso para dict()
- Módulos

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']  
cursos = ['progra', 'algebra', 'fisica', 'calculo']  
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**.

Ejemplo: ¿Cómo almacenar las notas de un curso?

Nota y curso de Diego?

```
def obtener_notas(estudiante, lista_nombres, lista_notas, lista_cursos):  
    i = lista_nombres.index(estudiante)  
    nota = lista_notas[i]  
    curso = lista_cursos[i]  
    return (curso, nota)  
  
curso, nota = obtener_notas('Diego', nombres, notas, cursos)  
print('Diego tiene un', nota, 'en', curso)
```

- Complicado si tienes varios tipos de información que almacenar
- Debes mantener **varias listas**, y pasarlas como argumento
- **Índice** siempre es un **entero** con la posición
- **DIFICIL DE MANTENER!**



Ejemplo: ¿Cómo almacenar las notas de un curso?

Clave **Valor**

↓ ↓

Diccionario →

```
1 notas = {'Diego'      : ('prograudd', 4.1)
2          , 'Francisca': ('algebra'   , 5.5)
3          , 'Daniela'   : ('fisica'    , 6.8)
4          , 'Leo'       : ('prograudd' , 3.9)}
5
6 def obtener_notas(estudiante, dict_notas):
7     curso, nota = dict_notas[estudiante]
8     return (curso, nota)
9
10 curso, nota = obtener_notas('Diego', notas)
11 print('Diego tiene un', nota, 'en', curso)
```

← **Tuple**

← **Tuple**

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)
```

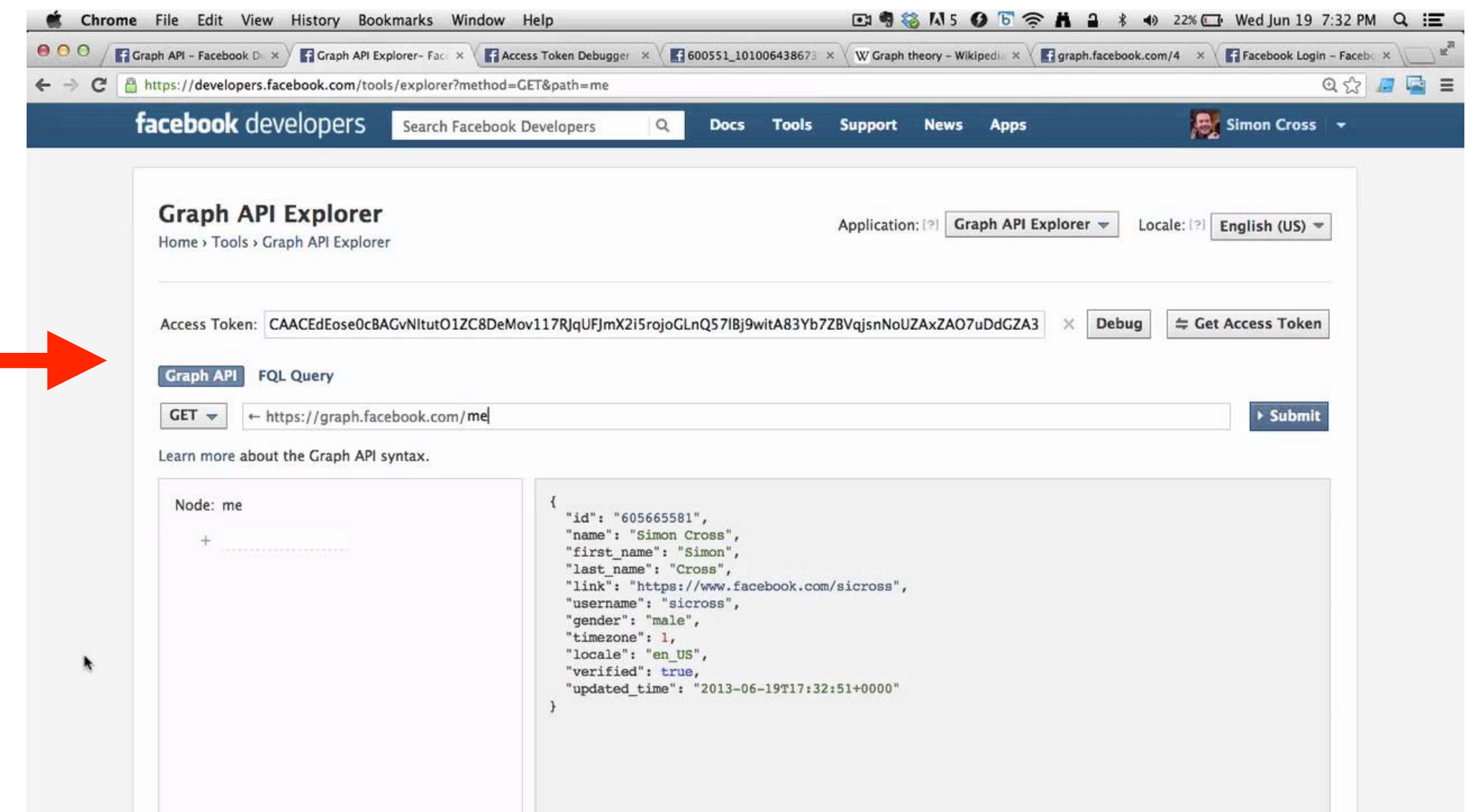
```
notas = {'Diego' : 4.1  
        , 'Francisca' : 5.5  
        , 'Daniela' : 6.8  
        , 'Leo' : 3.9}
```

API: Interfaz de programación de aplicaciones

- API: conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.
- Características:
 - Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios .
 - Uno de los principales es proporcionar un conjunto de **funciones** de uso general para evitar programar todo desde el principio.

Facebook Api Graph

Expone ciertas funcionalidades de Fb
para que otros puedan usarla



Facebook

<https://developers.facebook.com/docs/graph-api/>

Api Grap

Módulos

- Hasta ahora todos nuestros programas han consistido de un simple archivo.py
- Si nuestro código es muy extenso, es complicado mantener actualizado el archivo.py
- Para solucionar lo anterior podemos definir y ejecutar funciones en otros archivos
 - Permite reutilizar código: un programa usar código que ya ha sido escrito, sin necesariamente copiar-pegar
 - Permite hacer programación **modular**: construir programas componiendo código de diversas fuentes (otra vez, sin copiar y pegar)

Modulo Random

```
import random

regalos = ['sartén', 'jamón', 'mp4', 'muñeca', 'tv',
           'patín', 'balón', 'reloj', 'bicicleta', 'anillo']

for sorteo in range(5):
    regalo = regalos[random.randint(0, 9)]
    print('Sorteo', sorteo + 1, ':', regalo)
```

Fuente: <https://python-para-impacientes.blogspot.com/2015/09/el-modulo-random.html>

Modulos

```
# Módulo de números Fibonacci
```

```
def fib(n):    # Escribe la serie Fibonacci hasta n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
```

```
def fib2(n):   # Devuelve la serie Fibonacci hasta n
    resultado = []
    a, b = 0, 1
    while b < n:
        resultado.append(b)
        a, b = b, a+b
    return resultado
```



```
1 import fibo
```

```
1 fibo.fib(100)
```

```
1 1 2 3 5 8 13 21 34 55 89
```

```
1 fibo.fib2(100)
```

```
2
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

```
1 from fibo import fib, fib2
```

```
1 fib(100)
```

```
1 1 2 3 5 8 13 21 34 55 89
```

```
1 fib2(100)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

```
1 from fibo import *
```

```
1 fib(100)
```

```
2
```

```
1 1 2 3 5 8 13 21 34 55 89
```

```
1 fib2(100)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

Ejecutando modelos como scripts

```
# Módulo de números Fibonacci

def fib(n):    # Escribe la serie Fibonacci hasta n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b

def fib2(n):   # Devuelve la serie Fibonacci hasta n
    resultado = []
    a, b = 0, 1
    while b < n:
        resultado.append(b)
        a, b = b, a+b
    return resultado

if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```



```
[Daniela:~] daniela% cd
[Daniela:~] daniela% cd Desktop/Clase14/
[Daniela:~/Desktop/Clase14] daniela% ls
fib.py          fibo_2.py
[Daniela:~/Desktop/Clase14] daniela% python3 fibo_2.py 100
1 1 2 3 5 8 13 21 34 55 89 [Daniela:~/Desktop/Clase14] daniela%
```



Hace que el código pueda ser ejecutado como script o como módulo importable

Modulos

matriz.py

```
def crear(m,n):  
    #Retorna una matriz de ceros con m filas por n columnas  
    matriz = []  
    for i in range(m):  
        matriz.append([0]*n)  
    return matriz  
  
def imprimir(matriz):  
    #Imprime cada fila de una matriz  
    for fila in matriz:  
        for elem in fila:  
            print(elem, end=' ')  
        print()  
  
def asignar(matriz, i,j,v):  
    matriz[i][j] = v  
  
def main():  
    print(crear(10,20))  
  
if __name__ == '__main__':  
    main()
```

importa el código
en matriz.py

cliente.py

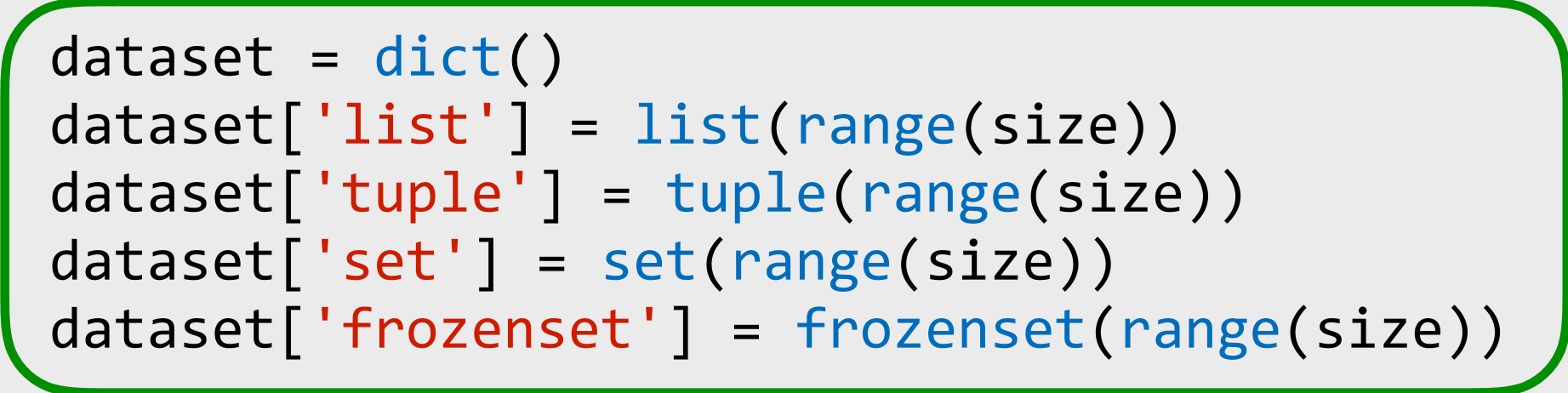
```
import matriz  
  
m = matriz.crear(3,3)  
matriz.asignar(m, 0, 1, 9)  
matriz.asignar(m, 2, 2, 3)  
matriz.asignar(m, 1, 2, 1)  
matriz.imprimir(m)
```


Desempeño en estructuras de datos

```
1 import timeit
2
3 def find(data, elem):
4     return elem in data
5
6 number_trials = 10
7
8 for size in [100000, 10**6, 10**7, 10**8]:
9     x = size - 1
10
11     dataset = dict()
12     dataset['list'] = list(range(size))
13     dataset['tuple'] = tuple(range(size))
14     dataset['set'] = set(range(size))
15     dataset['frozenset'] = frozenset(range(size))
16
17     for (datatype, datavalue) in dataset.items():
18         code = "find(dataset['{}'], x)".format(datatype)
19         avg_time = timeit.timeit(code, number=number_trials, globals=globals()) / number_trials
20         print('Average time for {} elements using {}: {:.3f}s'.format(size, datatype, avg_time))
```

datos están en:

- list
- tuple
- set
- frozenset



```
Average time for 100000 elements using list: 0.001s
Average time for 100000 elements using tuple: 0.001s
Average time for 100000 elements using set: 0.000s
Average time for 100000 elements using frozenset: 0.000s
Average time for 1000000 elements using list: 0.009s
Average time for 1000000 elements using tuple: 0.010s
Average time for 1000000 elements using set: 0.000s
Average time for 1000000 elements using frozenset: 0.000s
Average time for 10000000 elements using list: 0.094s
Average time for 10000000 elements using tuple: 0.094s
Average time for 10000000 elements using set: 0.000s
Average time for 10000000 elements using frozenset: 0.000s
Average time for 100000000 elements using list: 2.328s
Average time for 100000000 elements using tuple: 3.915s
Average time for 100000000 elements using set: 0.000s
Average time for 100000000 elements using frozenset: 0.000s
```

