



# Tipos de datos estructurados

Informática I - 2547100

Departamento de Ingeniería Electrónica  
y de Telecomunicaciones

Facultad de Ingeniería

2016-2

# Data types until now

Tipo de dato	Ejemplo	Descripción
int	5	números enteros
float	67.8	números reales
str	'hola'	cadenas de caracteres
bool	True	Verdadero o falso
None		

no escalar

escalar

Si quisiera almacenar las notas de un curso o las medidas de temperatura que hace un sensor cada hora, necesitaría muchas variables, cada una con nombre diferente.

# Tuples

Son secuencias de elementos como los strings, pero con la diferencia de que cada elemento puede ser de un tipo de dato diferente.

```
t1 = (5.24,)
t2 = (3.67, 'hola', 4, True)
t3 = (t2, 4, False)
t4 = t3 + t2
print(t4)
print(t4[2])
print(t4[0])
print(t4[2:4])
```

Diagram illustrating tuple operations:

- `t1 = (5.24,)`: una tupla de un solo elemento
- `t2 = (3.67, 'hola', 4, True)`: una tupla como elemento de otra tupla
- `t3 = (t2, 4, False)`: concatenación
- `t4 = t3 + t2`: concatenación
- `print(t4)`: indexación
- `print(t4[2])`: indexación
- `print(t4[0])`: segmentación
- `print(t4[2:4])`: segmentación

Output of `print(t4)`: `(3.67, 'hola', 4, True, 4, False, 3.67, 'hola', 4, True)`

# Example with tuples

```
def findDivisors(n1, n2):  
    '''Asume que n1 y n2 son enteros positivos  
    Retorna una tupla con todos los  
    divisores comunes de n1 y n2'''  
    divisors = () #tupla vacía  
    for i in range(1, min(n1,n2)+1):  
        if n1%i == 0 and n2%i == 0:  
            divisors = divisors + (i,)   
    return divisors  
  
divs = findDivisors(20, 100)  
print(divs)  
total = 0  
for d in divs:  
    total += d  
print('La suma de los divisores es: ', total)
```

# Lists

Son secuencias de elementos que pueden ser de diferente tipo, como las tuplas, pero con la diferencia de que las listas son **mutables**: pueden ser modificadas.

```
list1 = [9]
list2 = [4, 'free', 8.4]
list3 = [list2, True]
list4 = list3 + [4, -7.8]
print(list4)
list2[1] = 6.34
print(list4[2])
print(list4[0])
print(list4[2:4])
```

mutación:  
modificación  
de la lista

# Lists and mutability

```
electr = ['Control', 'Dig. Sys.']  
telecom = ['Antennas', 'Networks']  
  
areas1 = [electr, telecom]  
areas2 = [['Control', 'Dig. Sys.'], ['Antennas', 'Networks']]  
  
print(areas1)  
print(areas2)  
print(areas1==areas2)  
  
electr.append('Power Electronics')  
  
print(areas1)  
print(areas2)  
print(areas1==areas2)
```

# List methods

<b><code>list.append(e)</code></b>	agrega el objeto e al final de la lista
<b><code>list.count(e)</code></b>	retorna el número de veces que e se encuentra en la lista
<b><code>list.insert(i, e)</code></b>	inserta el objeto e en la posición i de la lista
<b><code>list.extend(list2)</code></b>	agrega todos los elementos de list2 al final de la lista
<b><code>list.remove(e)</code></b>	borra el primer elemento e que encuentre
<b><code>list.index(e)</code></b>	retorna la posición del primer elemento e que encuentre
<b><code>list.pop(i)</code></b>	borra el elemento de la posición i
<b><code>list.sort()</code></b>	ordena la lista
<b><code>list.reverse()</code></b>	invierte el orden de la lista

# Cloning

La clonación se refiere a crear una copia de un objeto de datos, a diferencia de enlazar un nuevo nombre de variable:

```
a = [2, 7, 4, 6]
b = a
```

vs.

```
a = [2, 7, 4, 6]
b = a[:]
```

```
def removeDups(L1, L2):
    '''Asume que L1 y L2 son listas sin elementos
    repetidos internamente. Elimina todos los
    elementos de L1 que también están en L2'''
    newL1 = L1[:]
    for e in newL1:
        if e in L2:
            L1.remove(e)
L1 = [1, 2, 3, 4]
L2 = [1, 2, 5, 6]
removeDups(L1, L2)
print(L1)
```

clonación:  
crear una  
copia de la lista



# Strings vs Tuples vs Lists

<code>seq[i]</code>	retorna el elemento i-ésimo de seq
<code>len(seq)</code>	retorna la longitud de seq
<code>seq1 + seq2</code>	retorna la concatenación de seq1 y seq2
<code>n * seq</code>	repite seq n veces
<code>seq[start:end]</code>	retorna un segmento de seq
<code>e in seq</code>	retorna True si e está en seq
<code>e not in seq</code>	retorna True si e no está en seq
<code>for e in seq</code>	itera sobre los elementos de seq

Tipo	Elementos	Ejemplos	Mutables
str	caracteres	<code>''</code> , <code>'a'</code> , <code>'abc4'</code>	No
tuple	cualquiera	<code>()</code> , <code>(3,)</code> , <code>('ab', 4)</code>	No
list	cualquiera	<code>[]</code> , <code>[3]</code> , <code>['ab', 4]</code>	<b>Sí</b>

# Advantages of Strings

Las listas son el tipo de dato más usado, sin embargo, los strings tienen funciones muy útiles:

<code>s.count(s1)</code>	cuenta cuantas veces está la cadena s1 en s
<code>s.find(s1)</code>	retorna el índice de la primera vez que encuentre s1
<code>s.rfind(s1)</code>	igual que find pero empieza desde el final
<code>s.lower()</code>	retorna una cadena convertida a minúsculas
<code>s.replace(old,new)</code>	retorna una cadena en la que se reemplaza la subcadena old por new
<code>s.rstrip()</code>	retorna una cadena borrando espacios al final
<code>s.split(d)</code>	retorna una lista de cadenas separadas por d

# Dictionaries

Son secuencias de elementos cuyos índices no son enteros sino valores inmutables (ej; strings). A los índices se les llama **claves** (keys).

```
monthsdays = {'enero':31, 'febrero':28, 'marzo':31, 'abril':30}
print(monthsdays['febrero'])
print(monthsdays['abril'])
m = input('Ingrese un mes: ')
print('El mes', m.title())
print('tiene', monthsdays[m.lower()], 'días')
```



*el diccionario es indexado por un string*

# Working with dictionaries

```
for e in monthsdays:  
    print(e, monthsdays[e])
```

→ febrero 28  
→ abril 30  
→ marzo 31  
→ enero 31

→ ¡desordenados!

<code>len(d)</code>	retorna la cantidad de elementos
<code>d.keys()</code>	retorna una lista con las claves de d
<code>d.values()</code>	retorna una lista con los valores de d
<code>k in d</code>	retorna True si la clave k está en d
<code>d[k]</code>	retorna el valor asociado a la clave k
<code>d.get(k, e)</code>	retorna d[k] si k está en d, de lo contrario retorna e
<code>d[k] = v</code>	asocia la clave con k con el valor v; si k no existía, crea un nuevo elemento en el diccionario.
<code>d.pop(k)</code>	elimina el elemento con clave k y retorna su valor asociado
<code>for k in d</code>	itera sobre las claves de d

# A language dictionary

```
ESToEN = {'pan': 'bread', 'vino': 'wine', \
          'come': 'eats', 'bebe': 'drinks', \
          'quiere': 'wants', 'un': 'a', \
          'nadie': 'nobody'}
ESToIT = {'pan': 'pane', 'vino': 'vino', \
          'come': 'mangia', 'toma': 'bebe', \
          'quiere': 'vuole', 'un': 'un', \
          'nadie': 'nessuno'}

def translateWord(word, dictionary):
    if word in dictionary:
        return dictionary[word]
    else:
        return word
```