

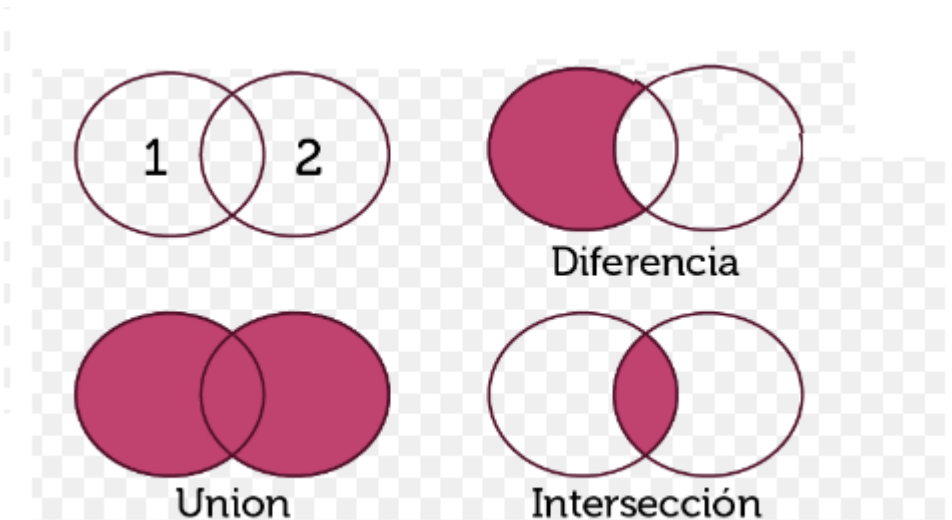
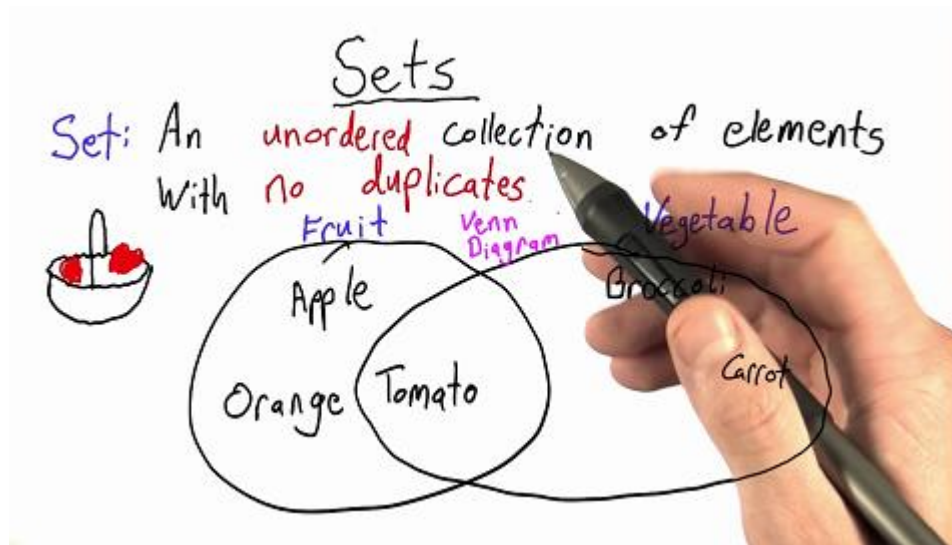
☐ Colecciones de Python

Conjuntos

Análisis de Textos

Expresiones Regulares

6.3. Conjuntos



Como funcionan los conjuntos ?

`set ()`

`.add`

Colección de elementos únicos.

```
In [1]: conjunto = set()
```

```
In [2]: conjunto
```

```
Out[2]: set()
```

```
In [3]: conjunto = {1,2,3}
```

```
In [4]: conjunto
```

```
Out[4]: {1, 2, 3}
```

```
In [5]: conjunto.add(4)
```

```
In [6]: conjunto
```

```
Out[6]: {1, 2, 3, 4}
```

```
In [7]: conjunto.add(0)
```

```
In [8]: conjunto
```

```
Out[8]: {0, 1, 2, 3, 4}
```

```
In [9]: conjunto.add('H')
```

```
In [10]: conjunto
```

```
Out[10]: {0, 1, 2, 3, 4, 'H'}
```

```
In [11]: conjunto.add('A')
```

```
In [12]: conjunto.add('Z')
```

```
In [13]: conjunto
```

```
Out[13]: {0, 1, 2, 3, 4, 'A', 'Z', 'H'}
```

```
In [14]: grupo = {'Hector', 'Juan', 'Mario'}
```

```
In [15]: 'Hector' in grupo
```

```
Out[15]: True
```

```
In [16]: 'Maria' in grupo
```

```
Out[16]: False
```

```
In [17]: 'Hector' not in grupo
```

```
Out[17]: False
```

```
In [18]: test = {'Hector', 'Hector', 'Hector'}
```

```
In [19]: test
```

```
Out[19]: {'Hector'}
```

In = busca en el conjunto

list = pasa conjunto a lista

set (lista) = pasa lista a conjunto

```
In [20]: l = [1,2,3,3,2,1]  
l
```

```
Out[20]: [1, 2, 3, 3, 2, 1]
```

```
In [21]: c = set(l)
```

```
In [22]: c
```

```
Out[22]: {1, 2, 3}
```

```
In [23]: l = list(c)
```

```
In [24]: l
```

```
Out[24]: [1, 2, 3]
```

```
In [25]: l = [1,2,3,3,2,1]
```

```
In [26]: l = list( set( l ) )
```

```
In [27]: l
```

```
Out[27]: [1, 2, 3]
```

Truco para eliminar duplicados de una lista :

1. Convierte la lista a conjunto
2. Convierte el conjunto a lista

Truco:

Convertir una cadena en un conjunto para eliminar letras duplicadas

```
In [28]: s = "Al pan pan y al vino vino"  
         set(s)
```

```
Out[28]: {' ', 'A', 'a', 'i', 'l', 'n', 'o', 'p', 'v', 'y'}
```

Ejemplo – Conjunto de usuarios

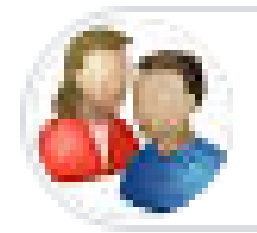
Realiza un programa que siga las siguientes instrucciones:

1. Crea un conjunto llamado usuarios con los usuarios **Marta, David, Elvira, Juan y Marcos**
2. Crea un conjunto llamado administradores con los administradores **Juan y Marta**.
3. Borra al administrador **Juan** del conjunto de administradores.
4. Añade a **Marcos** como un nuevo administrador, pero no lo borres del conjunto de usuarios.
5. Muestra todos los usuarios por pantalla de forma dinámica, además debes indicar cada usuario es administrador o no.

```
#1.  
usuarios = { 'Marta', 'David', 'Elvira', 'Juan' , 'Marcos' }  
type (usuarios)
```

```
#2.  
administradores = { 'Marta', 'Juan' }
```

```
#3.  
administradores -= {'Juan'}  
print (administradores)
```



2. **Avanzado.** Crea un Menú con 4 opciones

1. Crear Usuario,
2. Borrar usuario,
3. Añadir al grupo Administradores
4. Quitar del grupo de Administradores



- Si la opción es 1 pide el nombre del usuario y añádelo a la lista
- Si la opción es 2 elimina el usuario
- Si la opción es 3 o 4 muestra la lista de usuarios y la de administradores, pide un usuario existente para añadir o quitar del grupo de administradores.

Métodos de los conjuntos

Métodos básicos

`add()` Añade un elemento a un conjunto
`discard()` elimina un elemento si lo encuentra
`copy()` copia un conjunto en otro
`clear()` vacía un conjunto

Comparación de conjuntos

`isdisjoint()` Son disjuntos ?
`issubset()` Es un subconjunto ?
`issuperset()` Es un supra-conjunto ?

Métodos avanzados

`union()` Unión de 2 conjuntos
`update()` Añade elementos
`difference()` Diferencia de 2 conjuntos
`difference_update()`
`intersection()` Intersección de 2 conjuntos
`intersection_update()`
`symmetric_difference()` Estan en uno o en otro pero no en los 2

Abre el archivo `test_conjuntos.py` que te ofrece el tutor y comprueba que funciona.

Aprovecha para añadir algún ejemplo de los métodos que no aparecen.


```
c1 = set ()                # Un nuevo conjunto vacío
c1.add ("gato")             # Agregar un solo elemento
c1.update (["perro", "ratón"]) # Agregar varios elementos,
                              # por extensión de la lista
c1 |= set (["asno", "caballo"]) # Agregar 2 elementos
if "gato" in c1 :           # Esta en la colección ?
    c1.remove ("gato")
# c1.remove ("elefante") arroja un error
c1.discard ("elefante")     # No lanza ningún error
print (c1)

for item in c1:             # Iteracion for each element
    print (item)
print ("Item count:", len(c1)) # Longitud /tamaño/núm.elementos
primero = c1[0]
    # Error: no hay índices para conjunt
es_vacio = len(c1) == 0     # Prueba de vacío

c1 = {"gato", "perro"}      # Inicializar el set usando llaves;
#c1 = {}                   # Esto no es un diccionario
c1 = set(["gato", "perro"]) # Inicializar usando una lista
c2 = set(["perro", "raton"])
c3 = c1 & c2                # Intersección
```

```
c4 = c1 | c2          # Union
c5 = c1 - c3          # Diferencia de conjuntos
c6 = c1 ^ c2          # Diferencia simétrica
issubc = c1 <= c2     # Prueba de subconjunto
issuperc = c1 >= c2   # Prueba de superconjunto

c7 = c1.copy()        # Crea una copia
c7.remove("gato")
print (c7.pop())       # Elimina un elemento arbitrario

c8 = c1.copy()
c8.clear()             # Limpia, vacia el conjunto
c9 = {x for x in range(10) if x % 2}

print (c1, c2, c3, c4, c5, c6, c7, c8, c9, issubc, issuperc)
```

Practica P01 – Métodos de los conjuntos

Ejercicio 1

Sea el conjunto $c1 = \{1, 2, 3, 4\}$

- Añade el 5, descarta el 2 usando métodos.
- Agrega 3 números aleatorios (random) del 1 al 20 (*)
- Crea otro conjunto llamado $c2$ con los cuadrados del conjunto $c1$.
- Calcula e imprime la unión, intersección y diferencia de los conjuntos $c1$ y $c2$.

(*) `import random .random.randrange(1, 20)`

Ejercicio 2

Crea una función que se llame `modificar()` que partiendo de una lista de números realice las siguientes tareas sin modificar el original

- Borrar los elementos duplicados
- Ordenar la lista de mayor a menor
- Eliminar todos los números impares
- Realizar una suma de todos los números que quedan
- Añadir como primer elemento de la lista la suma realizada
- Devolver la lista modificada
- Finalmente, después de ejecutar la función, comprueba que la suma de todos los números a partir del segundo concuerda con el primer número de la lista.

Ejercicio 3 (Muy Avanzado)

- Pide por pantalla dos listas de elementos (lista1, lista2) cada una con elementos separados por comas.
- Pásalo a conjuntos y calcula su unión e intersección. Imprímelo.
- Usando la librería PIL dibuja los 2 conjuntos y escribe su unión e intersección
- Reflexiona sobre el tamaño y la posición de los elementos, que criterio de cálculo usarías.

<https://note.nkmk.me/en/python-pillow-imagedraw/>

Sintaxis de PIL

```
from PIL import Image, ImageDraw

img = Image.new('RGB', (500, 300), (128, 128, 128))
dib = ImageDraw.Draw(img)

dib.ellipse((100, 100, 150, 200), fill=(255, 0, 0),
            outline=(0, 0, 0))

dib.text(xtuple, text, fill=None, font=None, anchor=None,
        spacing=0, align="left")
```

SOLUCION P01. Ejercicio 1

```
from random import sample, randrange
c1 = {1, 2, 3, 4}
c1.add(5)
c1.discard (2)
print(c1)
# https://pynative.com/python-random-randrange/
c1.update ( {x for x in sample(range(1, 21), 3)} )
print("c1: ", c1)

c2 = { x**2 for x in c1 }
print("c2: ", c2)

print("c1 | c2: ", c1.union(c2))
print("c1 & c2: ", c1.intersection(c2))
print("c1 - c2: ", c1.difference(c2))
```

SOLUCION P01. Ejercicio 2

```
#-----  
# Funcion modificar que apartir de una lista de números  
  
def modificar (lista) :  
  
    lis1 = list(set(lista))  
    print(">>> sin duplicados :", lis1)  
    lis1.sort()  
    print (">>> ordenada :", lis1)  
  
    lis2 = [ x for x in lis1 if x % 2 == 0]  
    print(">>> solo pares :", lis2)  
  
    lis2.insert(0, sum(lis2))  
    print(">>> añadir suma :", lis2)  
    return (lis2)  
  
#----- Inicio del programa  
lini = [1,2,3,4,5,6,7,8,3,4,8]  
lfin = modificar (lini)  
  
print (lfin[0] == sum(lfin [1::]))
```

SOLUCION P01. Ejercicio 3

```
# lista1 = input ("Entri conjunt 1 (separat per comes) : ")
# lista1 = lista1.replace(" ", "").split(",")

# lista2 = input ("Entri conjunt 2 (separat per comes) : ")
# lista2 = lista2.replace(" ", "").split(",")

lista1 = ['banana', 'pera', 'manzana']
lista2 = ['baloo', 'banana', 'mowgli']

c1 = set(lista1)
c2 = set(lista2)

print ("c1 : ", c1)
print ("c2 : ", c2)
print ("Union  c1 | c2: ", c1 | c2)
print ("Inter. c1 & c2  : ", c1 & c2)
```

Continua en otra celda, ver diapositiva siguiente ...


```
from PIL import Image, ImageDraw
```

segueix
SOLUCION P01. Ejercicio 3

```
img = Image.new('RGB', (600, 400), 'White')  
dib = ImageDraw.Draw(img)
```

```
tam1 = len(c1)  
tam2 = len(c2)
```

```
#----- dibuja una elipse por cada conjunto  
font = ImageFont.truetype('arial',14)
```

```
dib.ellipse((10, 10, tam1 * 60, tam1 * 60), 'Crimson')  
# (x0, y0, x1, y1)  
for i, item in enumerate(c1) :  
    dib.text((40 + (i * 20) ,40 + (i * 20)),  
            str(item), (255,255,255), font=font)
```

```
xi = max(300, tam2 * 30)  
yi = max(200, tam2 * 30)  
dib.ellipse((xi, yi, 550, 350), 'Green') # (x0, y0, x1, y1)  
for i, item in enumerate(c2) :  
    dib.text((xi +40 + (i * 20) , yi + 40 +(i * 20)),  
            str(item), 'White', font=font)
```

```
dib.text((400, 10), "Inters: " +str(c1&c2), 'Red', font=font)  
dib.text((400, 70), "Unio: " +str(c1|c2), 'Black', font=font)
```

```
img.show()
```

El Análisis de texto automático

- Descartar palabras (limpieza previa)
- Buscar las palabras más usadas (relevancia)
- Detectar conjuntos de palabras (patrones)



Práctica P02-Conjuntos

Realiza los siguientes pasos :

1- Dada la cadena = "T7 Prom 5'-d CGC CAG GGT TTT TCA AGT CAC GAC
GAC AGT TAT TGC TCA GCG G"

2- Pásalo a formato lista usando el espacio como separador

`palabras = cadena.split(" ")`

3- Muestra las distintas palabras del texto (pasa a conjuntos) (`set`)

`distintas =`

4- Ahora recorre `distintas` con un `for`, y responde (imprime) :

4.1- Cuántas veces aparece cada palabra

4.2- Cuántas palabras hay de 3 letras

Práctica P03-Conjuntos (avanzado)

Realiza los siguientes pasos :

- 1- Crea una variable multi-línea, y copia-pegas el texto de la diapositiva siguiente (termina cada línea con `\`)
- 2- Pásalo a formato lista usando el espacio como separador (`split`)
- 3- Muestra las `distintas` palabras del texto (pasa a conjuntos)
- 4- Elimina las palabras = “el, la, los, las, un, de”
- 5- Ahora recorre la lista de palabras `distintas` con un `for`, y responde
 - 5.1- Cuántas veces aparece cada palabra
 - 5.2- Cuántas palabras hay de más de 5 letras

Solución P02 Conjuntos

#P02– Conjuntos-----

```
cadena = 'T7 Prom 5-d CGC CAG GGT TTT TCA AGT CAC GAC GAC AGT TAT  
TGC TCA GCG G '  
  
palabras = cadena.split(" ")          # divide cadena en palabras  
c = set(palabras)                     # crea conjunto de palabras distintas  
print (c)  
distintas = list(c)                   # crea lista a partir del conjunto  
  
pal3 = 0                              # crea contador para palabra de 3 letras  
  
for palabra in distintas :             # cuantas veces se encuentra  
    print ( f" {palabra} : {palabras.count(palabra)} veces " )  
    if len(palabra) == 3 :  
        pal3 +=1  
print (f"Hay {pal3} palabras de 3 letras ")
```

Solución P03

#P03

cadena = "La experiencia de aprendizaje de Python difiere bastante en función de la experiencia de cada uno. \n Sea cual sea el lenguaje informático practicado, es necesario tener cierta lógica y ser capaz de dominar ciertos \n conceptos algorítmicos. \n

Escoger Python como primer lenguaje es la mejor elección que puede realizar: muy próximo al lenguaje natural y a \n los conceptos algorítmicos clásicos, le permitirá hacer gran cantidad de cosas de manera muy natural y aprovechar \n una curva de aprendizaje muy pronunciada. \n

Esta experiencia de aprendizaje difiere bastante según los lenguajes practicados en el pasado. En efecto, cada \n lenguaje aporta su propia manera de pensar y su implementación de las técnicas algorítmicas, lo que moldea el \n pensamiento del que lo practica."

```
palabras = cadena.split(" ")          # Convierte el texto en una lista
print ("hay : ",len(palabras))        # Cuantas palabras hay
c = set(palabras)                     # convierte lista en conjunto
```

```
#conjunto de palabras a eliminar
cexcluye = {"el", "la", "los", "las", "un", "de"}
c = c - cexcluye
difers = list(c)
```

```
# y pasa el conjunto a otra lista palabras pero ya diferentes
print ("hay diferentes:",len(difers))
# Cuantas palabras diferentes hay
difers.sort()                        # Ordena la lista
```

```
print (":15} {:5} {}".format("palabra", "veces", "m5"))
print (":15} {:5} {}".format("-----", "----", "---"))

masde5letras = 0
for p in difers :
    # para cada palabra diferente ....
    extra = ""
    if len(p) > 5 :
        masde5letras +=1
        extra = str(masde5letras)

    # cuenta cuantas veces aparecia en la lista original

    print (":15} {:5} {}".format(p, palabras.count(p), extra) )

print ("palabras de más de 5 letras : ", masde5letras)
```

Practica conjunta

- Abre un fichero de texto en modo lectura (por ejemplo texto2.txt).
- Pasa todo el texto a minúsculas.
- Busca y muestra todas las palabras de más de 6 letras, y muestra su frecuencia, usando un diccionario para guardar las veces que aparece cada palabra.
- Observa el resultado y valora si las palabras más frecuentes te servirían para etiquetar conceptualmente el texto.


```

import re

#-----
# Localiza en un texto la frecuencia de las palabras de más de 6
letras
#-----

veces = {}      # crea un diccionario para veces de cada palabra
fiche = open('dat\\texto2.txt', 'r', encoding='utf-8')
texto = fiche.read().lower()
print("texto base", texto)

# obtiene una lista de las palabras de 6 a 15 letras
lista_palabras = re.findall(r'[a-z]{6,15}', texto, re.MULTILINE)

print ("Total palabras de más de 6 letras: ", len(lista_palabras))

for p in lista_palabras:
    count = veces.get(p,0)
    veces[p] = count + 1

claves = veces.keys()

for p in claves:
    print ( p, ": ", veces[p])

```

Usando re.findall para el buscar en todo el texto multilinea

El módulo re.findall () se usa cuando se desea recorrer en iteración las líneas del archivo, devolverá una lista de todas las coincidencias en un solo paso

Expresiones Regulares

Una *expresión regular* es un cadena de caracteres especial que define una búsqueda de patrones.

Puedes pensar en las expresiones regulares como un tipo de comodín con asteriscos.

En Python tenemos la librería `re` que permite hacer operaciones de búsqueda, y reemplazo sobre expresiones regulares.

Function	Description
<u>findall</u>	Returns a list containing all matches
<u>search</u>	Returns a <u>Match object</u> if there is a match anywhere in the string
<u>split</u>	Returns a list where the string has been split at each match
<u>sub</u>	Replaces one or many matches with a string

El módulo `re` de Python incluye funciones para trabajar con expresiones regulares.

`re.findall()`: busca todas las coincidencias en una cadena

La `findall()` función devuelve una lista que contiene todas las coincidencias.

Imprime una lista de todos los partidos:

```
import re

str = "The rain in Spain"
x = re.findall("ai", str)
print(x)
```

Ejemplo

re.match: busca un patrón al principio de otra cadena

Split at each white-space character:

```
import re
```

```
txt = "The rain in Spain"  
x = re.split("\s", txt)  
print(x)
```

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
```

```
txt = "The rain in Spain"  
x = re.search("^The.*Spain$", txt)
```

re.search: busca un patrón en otra cadena:

Replace every white-space character with the number 9:

```
import re
```

```
txt = "The rain in Spain"  
x = re.sub("\s", "9", txt)  
print(x)
```

Info extra : palabras que ...



Puedes pensar en las expresiones regulares como un tipo de comodín con asteriscos.

`/hola/` : Buscar el texto explícito `hola`.

`/a[nr]t/` : Busca un patrón el cual su primera letra sea la `a` y su última letra sea la `t`, y entre dichas letras contenga la letra `n` o la `r`.

Por lo tanto, casos de éxito para este patrón, serían las palabras inglesas `ant` y `art`.

`/ca[tbr]/` : Busca palabras que empiecen con `ca` y terminen con alguno de estos caracteres `tbr`

Identifiers	Modifiers	White space characters	Escape required
\d= any number (a digit)	\d represents a digit.Ex: \d{1,5} it will declare digit between 1,5 like 424,444,545 etc.	\n = new line	. + * ? [] \$ ^ () {} \
\s = space (tab,space,newline etc.)	? = matches 0 or 1	\t =tab	
\S= anything but a space	* = 0 or more	\e = escape	
\w = letters (Match alphanumeric character, including "_")	\$ match end of a string	\r = carriage return	
\W =anything but letters (Matches a non-alphanumeric character excluding "_")	^ match start of a string	\f= form feed	
. = anything but letters (periods)	matches either or x/y	----- ---	
\b = any character except for new line	[] = range or "variance"		
\.	{x} = this amount of preceding code		

Practica P04 – Practicando expresiones regulares

Realiza algunos de los ejemplos de la web

<https://python-para-impacientes.blogspot.com/2014/02/expresiones-regulares.html>

Practica P05-Nube de Etiquetas

.Extrae de forma automatizada los textos de diferentes páginas web y crea nubes de palabras personalizadas con Python.

En esta ocasión vamos a ver **cómo podemos crear una nube de palabras** gracias a la librería WordCloud de Andreas Muller (ver [web](#)), como por ejemplo esta:

<https://antonio-fernandez-troyano.medium.com/nube-de-palabras-word-cloud-con-python-a-partir-de-varias-webs-111e94220822>

1. Extraer textos de varias páginas web (web scraping)
2. “Limpiar” los textos (NLTK procesamiento del lenguaje) para eliminar signos de puntuación y palabras demasiado comunes (artículos, pronombres, adverbios, etc)
3. Abrir y convertir una imagen
4. Generar las nubes de palabras con WordCloud

