conjuntos

Son estructuras de información que contienen un grupo de datos, no necesariamente homogéneos. Se diferencian de las listas en que no pueden tener elementos duplicados ni se puede acceder a sus elementos en un orden determinado.

No admiten subíndices pero hay mecanismos para ver qué elementos forman parte de un conjunto.

Las constantes de tipo conjunto se escriben entre llaves, por ejemplo un conjunto de enteros sería $\{5, 23, -4, 67, 12, 9\}$. El conjunto vacío es una excepción y debe escribirse como `set()`.

conjuntos

```
mi_conjunto={5,8,1,2,1,5} # un conjunto de 4 enteros  
print(mi_conjunto)
```

mostraría (quizá en otro orden):
{8, 1, 2, 5}

```
otro_conjunto={"gato", 2+3, True, 27.8, 6-1}  
print(otro_conjunto)
```

mostraría:
{'gato', 27.8, 5, True}

funciones con conjuntos

- `len(p)` devuelve el número de elementos del conjunto
- `set(p)` convierte el parámetro a conjunto
- `sorted(p)` devuelve una lista ordenada con los elementos del conjunto

La función `sorted()` requiere que el conjunto sea homogéneo.

La función que más se usa es la `set()` que permite obtener un conjunto a partir de una cadena o de una lista. Por ejemplo si **palabras** es la lista de palabras que hay en un texto entonces **`set(palabras)`** es el conjunto de todas las palabras distintas, **`list(set(palabras))`** sería una lista de las palabras distintas y **`sorted(set(palabras))`** sería una lista ordenada de esas palabras distintas.

operadores para conjuntos

- resta de conjuntos, produce otro conjunto con los elementos del primero que no están en el segundo
- | unión de conjuntos, produce otro conjunto que los elementos que están bien en uno o bien en otro
- & intersección de conjuntos, produce otro conjunto con los elementos comunes a ambos
- ^ elementos que están en uno u otro pero no en ambos
- in cierto si un elemento pertenece a un conjunto
- <= => cierto si un conjunto está contenido en el otro
- < > cierto si un conjunto está contenido en el otro pero son distintos
- == != cierto si son iguales y cierto si son diferentes

instrucciones para conjuntos

bucle for iterable, permite recorrer un conjunto elemento a elemento para procesar cada uno de ellos

```
for elemento in conjunto:  
    # cuerpo del bucle  
    ...  
    ...  
    ...
```

métodos para conjuntos

<code>.add(p)</code>	añade el elemento al conjunto
<code>.discard(p)</code>	elimina (si lo contiene) el elemento del conjunto
<code>.pop()</code>	devuelve y elimina un elemento al azar del conjunto
<code>.clear()</code>	borra el conjunto dejándolo vacío

Ejemplo:

```
vocales="aeiou"  
palabra="ayuntamiento"  
print(set(vocales)<=set(palabra)) # es cierto
```

funciones (también llamadas subprogramas)

Además de las funciones estándar (len, input, int, str, list, etc ...) es posible definir nuevas funciones. Estas funciones se comportan como programas completos que reciben los datos a procesar entre paréntesis y devuelven o retornan resultados. Los datos a procesar reciben el nombre de parámetros.

```
def doblesuma(a,b):  
    return 2*a+b
```

```
x=5  
y=3+doblesuma(x,8-2) # y valdrá 3+(2*5+6) o sea 19
```

funciones (también llamadas subprogramas)

Importante, las variables que se emplean dentro de una función solo existen mientras se ejecuta la función y únicamente dentro de esa función. No existen en el programa principal. Dado que pueden emplear entonces todos los nombre de variables (y otras funciones) es por eso por lo que se denominan subprogramas.

```
def f(): # una función simple que no devuelve nada
    i=5  # esta variable i es local y solo existe en f

i=7     # esta variable i se dice que es global
f()     # llamada a la función
print(i) # muestra 7
```

funciones (también llamadas subprogramas)

Importante, los parámetros que recibe una función son copias de los valores que hay en la llamada, excepto en el caso de listas y en general en todos los tipos de datos que pertenecen al grupo de los llamados "mutables".

```
def f(x):    # una función simple que no devuelve nada
    x=x*2   # x es una copia de lo que recibe

y=3        # variable y global, envía 3 a la función
f(y)       # llamada a la función, se le pasa el número 3
print(y)   # muestra 3
```

funciones con listas

En cambio con listas se recibe un puntero a donde apunta la lista, es decir la lista del programa que hace la llamada y la de la función son la misma.

```
def f(x): # una función simple que no devuelve nada
    for i in range(len(x)):
        x[i]=x[i]+1
```

```
z=[6, -2, 47]
```

```
f(z) # llamada a la función, la lista z del
      # programa es la misma que la x de la función
```

```
print(z) # muestra [7, -1, 48]
```

ejemplos de funciones

Supongamos que no existe la función len, vamos a definir una nueva.

```
def mi_len(parametro):  
    contador=0  
    for elemento in parametro:  
        contador=contador+1  
    return contador
```

Posibles llamadas serían mi_len("prueba"), mi_len(unalista)

funciones que devuelven varios valores

Simplemente se ponen los resultados a devolver separados por comas.
En la llamada hay que poner comas también.

```
def ordena(a,b): # devuelve el menor y el mayor
    if a<b:
        return a,b
    else:
        return b,a

# programa principal con ejemplos de llamadas
x,y=ordena(5,2)          # x será 2 e y será 5
p,q=ordena("hola","tú") # p será "hola" y q será "tú"
```

para hacer:

Reescribir empleando funciones alguno de los ejercicios ya vistos. El criterio a emplear es uno de estos dos o ambos:

- que tanto las funciones como el programa principal no ocupen más de una página de texto, para mantener la legibilidad
- que las misma operaciones haya que realizarlas con varios valores

Ejemplo, si en varias partes de un programa debe verificarse si una palabra contiene las cinco vocales minúsculas sin acentuar:

```
def tiene_5_vocales(cadena):  
    return set(cadena) >= set("aeiou")
```

```
def tiene_las_5_vocales(cadena): # ejemplo más realista
    # paso todo a minúsculas
    cadena=cadena.lower()
    # cambio de letras acentuadas y diéresis
    cadena=cadena.replace("á","a").replace("é","e")
    cadena=cadena.replace("í","i").replace("ó","o")
    cadena=cadena.replace("ú","u").replace("ü","u")
    return set("aeiou")<=set(cadena)

# programa principal
print(tiene_las_5_vocales("Murciélago")) # True
print(tiene_las_5_vocales("Esperteyu")) # False
```