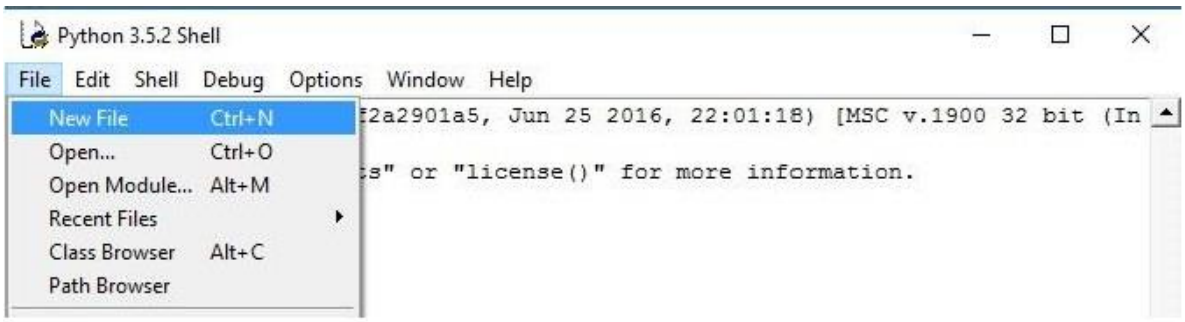
Curso aprendiendo a programar con Python
2016

Trabajo Práctico N°3

Listas

Ejercicio 1 – Lista

1. El objetivo del programa es imprimir una lista de números del 0 al 10.
2. Abrir el programa IDLE (Python 3.5)
3. Observar que aparece el prompt esperando ingreso de datos.
4. Ingresar al menú **File → New File** o **Ctrl+N**



5. Se abrirá una nueva pantalla en blanco

6. Cree un archivo, llamado Lista.
7. Cree una variable entera llamada cantidad e inicialícela con 11
8. Cree una variable de tipo arreglo llamada lista con contenido vacío mediante la siguiente línea:

```
lista=[]
```

9. Utilice un bucle para recorrer el arreglo con la siguiente línea:

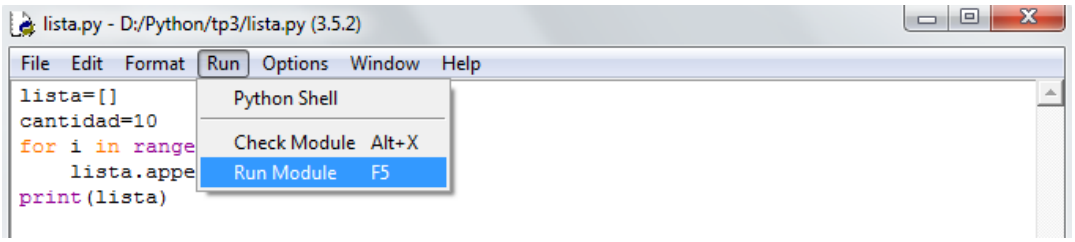
```
for i in range(cantidad):
```

10. Indique al programa que vaya colocando los elementos en el arreglo con la siguiente línea:

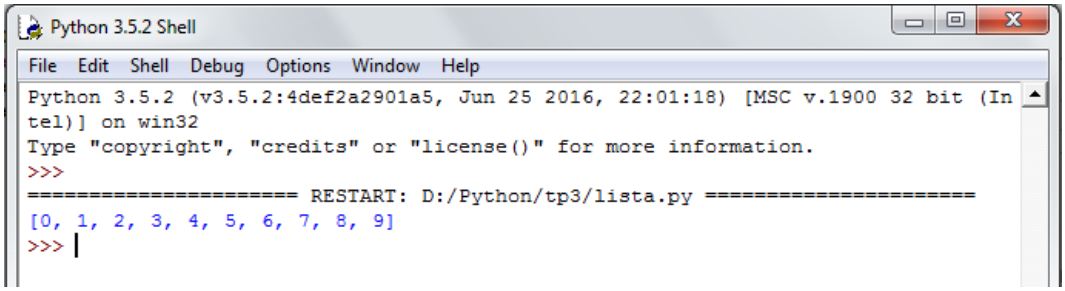
```
lista.append(i)
```

Append es una función predefinida en Python que permite agregar al final de un arreglo elemento que se le pase por parámetro, en nuestro caso el numero i.

11. Muestre el resultado en la salida estándar mediante un mensaje.
12. Guarde el código del programa con **File → Save** o **Ctrl+S** con el nombre Lista.py (recuerde que el programa debe tener extensión .py o Tipo “Python Files”, de lo contrario no funcionará).
    No se olvide de seleccionar la carpeta donde desee que se guarde el archivo (**Recomendación:** Modificar la que aparece por defecto).
13. Ejecute el programa con **F5** o con **Run → Run Module**



14. Se abrirá nuevamente la pantalla principal del IDLE (donde figura el prompt >>>).
15. A continuación comenzará a ejecutarse su programa y se visualizará el resultado del mismo de color azul.



Ejercicio 2 – Pacientes

1. El objetivo del programa es imprimir una lista de 5 pacientes ingresados por el usuario.
2. Cree un nuevo archivo llamado Pacientes.
3. Cree una variable entera llamada cantidad e inicialícela con 6
4. Cree una variable de tipo arreglo llamada pacientes con contenido vacío mediante la siguiente línea:
   ```
   pacientes=[]
   ```
5. Utilice un bucle para recorrer el arreglo con la siguiente línea:
   ```
   for i in range(cantidad):
   ```
6. Solicite al usuario ingresar el nombre del paciente y guarde el nombre en una variable string llamada nombre.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1, pero pasando por parámetro a nombre.
8. Muestre el resultado.
9. Guarde el archivo.
10. Ejecute

Ejercicio 3 – Lista invertida

1. El objetivo del programa es imprimir una lista de 10 elementos cualquiera, pero en el orden inverso a como fueron ingresados.
2. Cree un nuevo archivo llamado ListaInvertida.
3. Cree una variable entera llamada cantidad e inicialícela con 10
4. Cree una variable de tipo arreglo llamada lista con contenido vacío mediante la siguiente línea:
   ```
   lista=[]
   ```
5. Utilice un bucle para recorrer el arreglo con la siguiente línea:
   ```
   for i in range(cantidad):
   ```
6. Solicite al usuario ingresar un elemento y guárdelo en una variable llamada elemento.

7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append`.
8. Recorra la lista en sentido inverso mediante la siguiente línea:
```
for i in range(len(lista)-1, -1, -1):
```
9. Muestre el resultado.
10. Guarde el archivo.
11. Ejecute.

Ejercicio 4– InversoVariable

1. Cree un nuevo archivo llamado InveroVariable.
2. Realice una copia del programa del ejercicio 3, pero permita al usuario elegir la cantidad de números que desea ingresar, en vez de hacerlo siempre con 10 elementos.
3. Guarde el archivo.
4. Compile y ejecute el programa.

Ejercicio 5 – Corresponde

1. El objetivo del programa es saber si un paciente ingresado por teclado figura en la lista de pacientes del ejercicio 2.
2. Cree un nuevo archivo llamado corresponde.
3. Utilice el código escrito en el ejercicio 2.
4. Solicite al usuario ingresar el nombre del paciente que desea buscar y guarde el nombre en una variable string llamada nombre.
5. Indique al programa que recorra el arreglo.
6. Solicite al programa que verifique si el paciente figura en la lista con la siguiente línea:
```
if nombre in pacientes:
```

Donde la palabra reservada `in` es utilizada para indicarle al programa dónde tiene que corroborar lo solicitado.
Anteriormente debimos colocar un condicional.

7. Muestre el resultado.
8. Guarde el archivo.
9. Ejecute

Ejercicio 6 – Borrar Paciente

1. El objetivo del programa es borrar del registro un paciente ingresado por teclado de la lista de pacientes del ejercicio 2.
2. Cree un nuevo archivo llamado eliminarPaciente.
3. Utilice el código escrito en el ejercicio 2.
4. Solicite al usuario ingresar el nombre del paciente que desea eliminar y guarde el nombre en una variable string llamada nombre.
5. Indique al programa que recorra el arreglo.
6. Solicite al programa que elimine el paciente de la posición actual si es igual al paciente ingresado mediante el siguiente código:
```
if nombre == lista[i]:
    del lista[i]
```

Donde la palabra reservada `del` es utilizada para indicarle al programa que tiene que eliminar un elemento del arreglo.

7. Muestre el resultado.

8. Guarde el archivo.
9. Ejecute

Ejercicio 7 – Número de Atención

1. El objetivo del programa es consultar el numero de atención del registro un paciente ingresado por teclado de la lista de pacientes del ejercicio 6.
2. Cree un nuevo archivo llamado numeroPaciente.
3. Utilice el código escrito en el ejercicio 6.
4. Solicite al programa el número de orden del paciente de la posición de un paciente ingresado mediante el siguiente código:

```
posicion=lista.index(nombre)
```

Donde el parámetro que figura entre comillas debe ser el nombre del paciente ingresado que se desea saber su posición.
Observe que el índice obtenido se guarda en una variable llamada posicion.

5. Muestre el resultado.
6. Guarde el archivo.
7. Ejecute

Ejercicio 8 – Prioridad

1. Suponga que hubo una emergencia y se debe atender a un paciente con prioridad.
2. El objetivo del programa es modificar el programa realizado en el ejercicio anterior, añadiendo en el primer lugar de la lista un nombre paciente.
3. Cree un archivo llamado Prioridad
4. Utilice la lista de pacientes creada en el ejercicio 2.
5. Muestre por pantalla el resultado de la lista sin el nuevo paciente.
6. Solicite al usuario que ingrese un nombre del paciente y guárdelo en una variable llamada prioridad
7. Agregue el nuevo paciente a la lista creada anteriormente mediante la siguiente línea:

```
lista.insert(0,prioridad)
```

8. Muestre el resultado
9. Guarde el archivo
10. Ejecute

Ejercicio 9 – Dividir lista

1. El objetivo del programa es dividir una lista cualquiera en 2 mitades.
2. Cree un nuevo archivo llamado DividirLista
3. Cree una lista de 6 elementos cualquiera y llámela lista.
   Ejemplo: lista=[‘a’,b’,4,5]
4. Calcule la longitud de dicha lista mediante la siguiente línea y guárdelo en una variable:

```
longitud=len(lista)
```

5. Calcule la mitad de la longitud y guárdelo en una variable llamada mitad.
6. Divida la lista en 2 mitades mediante el siguiente código:

```
lista[0:mitad]
lista[mitad:longitud]
```

Observe que el primer parámetro de los corchetes es la posición a partir de la cual queremos que comience el segmento y el segundo parámetro (luego de los 2 puntos) es el límite deseado.

7. Muestre el resultado.
8. Guarde el archivo.
9. Ejecute

Ejercicio 10 – Mínimo

1. El objetivo del programa es encontrar el menor elemento de una lista de números.
2. Cree un nuevo archivo llamado Minimo.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada numeros con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar un número y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. Busque el menor elemento mediante la siguiente instrucción:
   ```
   min(numeros)
   ```
9. Muestre el resultado.
10. Guarde el archivo.
11. Ejecute

Ejercicio 11 – Máximo

1. El objetivo del programa es encontrar el mayor elemento de una lista de números.
2. Cree un nuevo archivo llamado Maximo.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada numeros con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar un número y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. Busque el mayor elemento mediante la siguiente instrucción:
   ```
   max(numeros)
   ```
9. Muestre el resultado.
10. Guarde el archivo.
11. Ejecute

Ejercicio 12 – Lista Contactos

1. El objetivo del programa es Ordenar una lista de nombres por orden alfabético.
2. Cree un nuevo archivo llamado Contactos.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada contactos con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar un nombre y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. Ordene la lista alfabéticamente mediante la siguiente instrucción:
   ```
   ordenada=sorted(lista)
   ```

Observe que se creo una nueva variable llamada ordenada, que es la lista original con el orden solicitado.

9. Muestre el resultado.
10. Guarde el archivo.
11. Ejecute

Ejercicio 13 – Buscar palabra

1. El objetivo del programa es verificar cuántas veces figura una palabra en la lista.
2. Cree un nuevo archivo llamado BuscarPalabra.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada lista con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar una palabra y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. En cada iteración del bucle, compare el valor de la lista con el valor buscado y si coinciden, aumente el valor de un contador
9. Muestre el resultado.
10. Guarde el archivo.
11. Ejecute

Ejercicio 14 – Sustituir palabra

1. El objetivo del programa es sustituir una palabra ingresada por el usuario por una existente en una lista.
2. Cree un nuevo archivo llamado Sustituir.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada lista con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar una palabra y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. Solicite al usuario que ingrese una palabra a buscar y guárdela en una variable llamada buscar.
9. Solicite al usuario que ingrese una palabra para sustituir y guárdela en una variable llamada sustituir.
10. En cada iteración del bucle, compare el valor de la lista con el valor buscado y si coinciden, sustituya el valor de la lista mediante el siguiente código:

```
if lista[i] == buscar:
   lista[i] = sustituir
```

11. Muestre el resultado.
12. Guarde el archivo.
13. Ejecute

Ejercicio 15 – Insertar palabra

1. El objetivo del programa es insertar una palabra ingresada por el usuario en un arreglo en la posición indicada por el usuario.

2. Cree un nuevo archivo llamado InsertarPalabra.
3. Solicite al usuario que ingrese una longitud para crear un arreglo.
4. Cree una variable de tipo arreglo llamada lista con la longitud ingresada por el usuario.
5. Utilice un bucle para recorrer el arreglo
6. Solicite al usuario ingresar una palabra y guárdelo en la lista.
7. Indique al programa que vaya colocando los elementos en el arreglo con la instrucción `append` vista en el ejercicio 1.
8. Solicite al usuario que ingrese una palabra y guárdelo en una variable llamada palabra.
9. Solicite al usuario que ingrese un número para la posición donde desee ingresar la palabra y guárdelo en una variable llamada posición. Recuerde realizar la conversión del input a tipo entero.
10. Inserte la palabra en la posición deseada con la siguiente línea:
    ```
    lista.insert(posicion,palabra)
    ```
11. Muestre el resultado.
12. Guarde el archivo.
13. Ejecute

Tuplas

Ejercicio 16 – Nombre y Apellido

1. El objetivo del programa es mostrar una tupla que contenga el nombre y apellido del usuario.
2. Cree un archivo llamado NombreYApellido
3. Solicite al usuario que ingrese su nombre y guárdelo en una variable llamada nombre
4. Solicite al usuario que ingrese su apellido y guárdelo en una variable llamada apellido
5. Cree una tupla llamada tupla con la siguiente línea:
    ```
    tupla=tuple()
    ```
6. Inserte los valores ingresados por el usuario dentro de la nueva tupla mediante la siguiente línea:
    ```
    tupla=(nombre,apellido)
    ```
7. Muestre el resultado.
8. Guarde el archivo.
9. Ejecute

Ejercicio 17 – Becarios

1. El objetivo del programa es crear una pequeña base de datos donde se registren los alumnos becados, discriminando en cada uno la beca que posea
2. Cree un archivo llamado Becas
3. Solicite a usuario que ingrese la cantidad de becarios y guárdela en una variable llamada cantidad
4. Cree una variable de tipo arreglo llamada becarios con contenido vacío
5. Utilice un bucle para recorrer el arreglo con la longitud ingresada por el usuario con la siguiente línea:
6. Solicite al usuario ingresar el nombre del becario y guarde el nombre en una variable string llamada nombre.

7. Solicite al usuario ingresar el nombre de la beca correspondiente al becario y guarde el nombre en una variable string llamada nombrebeca.
   Recuerde que las becas ofrecidas actualmente por el Centro de Estudiantes son: fotocopias, trabajo, comedor, plotter y transporte.
8. Indique al programa que cree una tupla por cada becario y beca ingresados mediante la siguiente línea:
   `becas=(nombre,beca)`
9. Agregue la tupla instanciada al arreglo becarios mediante la siguiente línea:
   `becarios.append(becas)`
10. Muestre el resultado.
11. Guarde el archivo.
12. Ejecute

Ejercicio 18 - Contar becas

1. El objetivo del programa es ordenar los becarios ingresados según su promedio (primero el de mayor promedio y último el de promedio más bajo)
2. Cree un nuevo archivo llamado ContarBecas.
3. Utilice la “base de datos” generada en el ejercicio 17.
4. Antes de solicitar que ingrese el nombre, solicite al usuario que ingrese el promedio del alumno y guárdelo en una variable llamada promedio.
5. Indique al programa que ordene la lista creada de mayor promedio a menor promedio mediante la siguiente línea:
   `becarios.sort(reverse=True)`

Donde la ordena automáticamente los valores ingresados.
Observe que en este caso le enviamos por parámetros `reverse=True`, esto lo que hace es indicar al programa que ordene de mayor a menor (Python por defecto lo hace al revés).

6. Muestre el resultado
7. Guarde el archivo.
8. Ejecute

Ejercicio 19 – Orden de mérito

1. El objetivo del programa es mejorar el código desarrollado en el ejercicio 18. Ahora además de ordenar a los becarios por promedio, deseo que se me muestre por pantalla el número de orden de mérito obtenido.
2. Cree un nuevo archivo llamado OrdenDeMerito
3. Utilice nuevamente la “base de datos” generada en el ejercicio 17.
4. Utilice el ordenamiento obtenido en el ejercicio 18.
5. Indique al programa que le asigne un número de orden de mérito a cada tupla obtenida mediante la siguiente línea:
   `numerada= list(enumerate(becarios,1))`

Donde la función enumerate le asigna un subíndice a cada elemento de la lista llamada becarios.
Observe que además de pasarle por parámetro el nombre de la lista que deseo numerar, le envío el número 1. Esto quiere decir que yo deseo que comience a enumerar a partir de dicho número (Python por defecto comienza en 0).
Observe además que a la lista ordenada la guardo en una nueva lista llamada numerada.

6. Muestre el resultado.
7. Guarde el archivo.

8. Ejecute

Ejercicio 20 – Números telefónicos

1. El objetivo del programa es combinar 2 listas en una: una contendrá nombres de contactos y la otra contendrá sus números de teléfono.
2. Cree un nuevo archivo llamado NumerosTelefonicos.
3. Cree una tupla llamada contactos con tantos elementos como desee. Recuerde que para ser string los valores deben estar entre comillas y separados por coma.
4. Cree una tupla de igual cantidad de elementos a contactos llamada telefonos. Recuerde que aunque sean números, los valores deben estar entre comillas y separados por coma.
5. Indique al programa que relacione estas 2 tuplas mediante la siguiente línea:

```
agenda = zip(contacto, telefono)
```

Donde la función zip sirve para unir 2 tuplas

6. Muestre por pantalla mediante el siguiente código:

```
for nombre, valor in agenda:
    print("%s: %s" % (nombre, valor))
```

7. Muestre el resultado.
8. Guarde el archivo.
9. Ejecute.
10. ¿Encuentra algo diferente en el formato de la salida?

Diccionarios

Ejercicio 21 – Selección Argentina

1. El objetivo del programa es programar un diccionario donde se pueda visualizar la formación de la selección Argentina frente a Uruguay.
2. Cree un nuevo archivo llamado SeleccionArgentina
3. Declare e inicialice un diccionario llamado seleccionado con contenido vacío mediante la siguiente línea:

```
seleccionado=dict()
```

4. Indique al programa los jugadores que formaron parte del 11 inicial de la siguiente manera:

```
seleccionado={
    1: “Romero”, 4: “Zabaleta”,
    17: “Otamedi”, 13: “Funes Mori”,
    16: “Mas”, 14: “Mascherano”,
    6: “Biglia”, 10: “Messi”,
    22: “Dybala”, 11: “Di Maria”,
    18: “Pratto”
}
```

Observe que el diccionario se declara entre los caracteres '{}' y cada uno de los elementos los separamos por comas. Cada elemento del diccionario lo definimos con su par clave:valor, pudiendo ser la clave y el valor de cualquier tipo de dato (‘int’, ‘float’, ‘chr’, ‘str’, ‘bool’, ‘object’).

5. Utilice un bucle para recorrer el diccionario mediante el siguiente código:

```
for k,v in seleccionado.items():
```

Donde la función .items sirve para indicar al programa que recorra el contenido del diccionario.

6. Dentro del bucle muestre por pantalla con la siguiente línea:

```
print ("%s -> %s" %(k,v))
```

Los parámetros (k,v) corresponden a “key” y “value”. Estos podrían poseer cualquier nombre.

7. Preste atención al formato de salida.
8. Muestre el resultado.
9. Guarde el archivo.
10. Ejecute.

Ejercicio 22 – Número de Dorsales

1. El objetivo del programa es imprimir el número de los dorsales de los jugadores de ejercicio anterior.
2. Cree un nuevo archivo, llamado Dorsales.
3. Utilice el diccionario declarado en el ejercicio 21.
4. Utilice la siguiente función predefinida de los diccionarios en Python para imprimir las claves de los elementos que lo componen:

```
keys = seleccionado.keys();
```

5. Muestre por pantalla el listado de dorsales mediante la siguiente línea:

```
print ("\nDorsales de jugadores: \n%s" %keys)
```

6. Guarde el archivo
7. Ejecute.

Ejercicio 23 - Datos Productos

1. El objetivo del programa es guardar datos de varios productos de un catálogo en un diccionario.
2. Cree un nuevo archivo, llamada DatosProductos
3. Solicite al usuario que ingrese la cantidad de productos que desea guardar
4. Declare e inicialice un diccionario llamado productos, mediante la siguiente línea:

```
productos = dict()
```

5. Utilizando un bucle, solicite al usuario que ingrese para cada producto, su nombre, el stock existente y el precio.
6. Guarde cada uno de estos datos en el diccionario mediante la siguiente línea:

```
productos[nombre]= [stock, precio]
```

7. Una vez ingresados los valores, muestre el catálogo de productos con el siguiente formato.

Producto	**Stock**	**Precio**
Mouse	[‘5’,	‘120.0’]
Teclado	[‘10’,	‘90.0’]

8. Utilice el siguiente código para mostrar por pantalla:

```
print("Producto","Stock","Precio")
for nombre,stock in productos.items():
   print (nombre, ":", stock)
```

9. Guarde el archivo.
10. Compile y ejecute el programa.

Conjuntos

Ejercicio 22 – Materias de la Universidad

1. El objetivo del programa es determinar que materias tienen en común en primer año Ingeniería en sistemas e Ingeniería Electrónica.
2. Cree un nuevo archivo llamado MateriasUniversidad
3. Declare e inicialice un conjunto llamado sistemas con contenido vacío mediante la siguiente línea:
   ```
   sistemas=set()
   ```
4. Indique al programa las materias que forman parte del primer año de la carrera de la siguiente manera:
   ```
   sistemas={'Analisis Matemático I', 'Algebra y Geometría
   Analítica', 'Ingles I','Matemática Discreta','Sistemas y
   Organizaciones', 'Algoritmos y estructuras de datos',
   'Arquitectura de las computadoras','Fisica I'}
   ```
5. Declare e inicialice un conjunto llamado sistemas con contenido vacío mediante la siguiente línea:
   ```
   electronica=set()
   ```
6. Indique al programa las materias que forman parte del primer año de la carrera de la siguiente manera:
   ```
   electronica={'Informática I (Anual)','Algebra y Geometría
   Analítica','Análisis Matemático I','Ingeniería y
   Sociedad','Representación gráfica','Análisis Matemático
   II','Fisica I','Sistemas de Representacion'}
   ```
7. Solicite al programa que filtre las materias que se encuentran en ambas carreras mediante la siguiente línea:
   ```
   print (sistemas & electronica)
   ```
8. Muestre el resultado.
9. Guarde el archivo.
10. Ejecute.

Ejercicio 23 – Materias de la Universidad

1. El objetivo del programa es determinar que materias no tienen en común en primer año Ingeniería en sistemas e Ingeniería Electrónica.
2. Cree un nuevo archivo llamado MateriasUniversidad2
3. Utilice la pequeña base de datos creada en el ejercicio anterior.
4. Solicite al programa que filtre las materias que se encuentran en ambas carreras mediante la siguiente línea:
   ```
   print (sistemas | electronica)
   ```
5. Muestre el resultado.
6. Guarde el archivo.
7. Ejecute.

Ejercicio 24 – Materias de la Universidad

1. El objetivo del programa es determinar que materias se encuentran en primer año Ingeniería en sistemas y no en el de Ingeniería Electrónica.
2. Cree un nuevo archivo llamado MateriasUniversidad3
3. Utilice la pequeña base de datos creada en el ejercicio 22.

4. Solicite al programa que filtre las materias que se encuentran sólo en la carrera de sistemas mediante la siguiente línea:

```
print (sistemas - electronica)
```

5. Muestre el resultado.
6. Guarde el archivo.
8. Ejecute.

Ejercicio 25 – Existencia de Materia

1. El objetivo del programa es determinar si una materia ingresada por teclado se encuentran en primer año Ingeniería en sistemas o no.
2. Cree un nuevo archivo llamado ExisteMateria
3. Indique al programa las materias que forman parte del primer año de la carrera de la siguiente manera:

```
sistemas={'Analisis Matemático I', 'Algebra y Geometría
Analítica', 'Ingles I','Matemática Discreta','Sistemas y
Organizaciones', 'Algoritmos y estructuras de datos',
'Arquitectura de las computadoras','Fisica I'}
```

4. Solicite al usuario que ingrese el nombre de la materia que desea buscar y guárdela en una variable llamada materia.
5. Verifique que la materia exista o no en su pequeña base de datos mediante la siguiente línea:

```
resultado=materia in sistemas
```

Observe que se guarda en una variable llamada Resultado el resultado de la consulta.

6. Muestre el resultado.
7. Guarde el archivo.
8. Ejecute.