1. Los caracteres individuales que forman una cadena alfanumérica (string) pueden ser accesados usando el número del *índice* correspondiente. En Python **el valor de los índices siempre comienza en 0**, luego 1, 2, etc. Para ilustrar esto, abra un intérprete Python (o IPython) y ejecute los siguientes comandos:

```
x="Hola estudiantes de primer semestre"
print(x[0])
print(x[1])
print(x[2])
print(x[3])
```

2. Como comprobó en la práctica anterior la función `len()` entrega el largo del string, es decir, el número de caracteres que contiene. Por lo tanto

```
print(x[len(x)-1])
```

   Imprime el último caracter del string, cuyo índice es `len(x)-1`, debido que el índice del primer caracter es 0. El mismo resultado, puede ser conseguido usando

```
print(x[-1])
```

   Similarmente,

```
print(x[-2])
```

   imprime el penúltimo caracter, y así sucesivamente. Por ejemplo, ejecute

```
x[-3]==x[len(x)-3]
```

   para comprobar que se refieren al mismo caracter.

3. También es posible acceder a algunos caracteres del string usando, en nuestro caso, `x[inicio:fin:paso]`, donde `inicio` y `fin` los índices de los caracteres iniciales y finales y `paso` es un entero que define el paso. Si `paso` no es ingresado, se asume `paso=1` . Por ejemplo, ejecute y verifique qué hacen los siguientes comandos

```
print(x[0:4:1])
print(x[0:4])
print(x[5:16])
print(x[1:20:2])
```

   Note que el caracter correspondiente al índice `fin` NO es desplegado. En lenguaje matemático podríamos decir que `x[inicio:fin]` suministra los caracteres de `x` con índices en el intervalo desde `inicio` *cerrado* hasta `fin` *abierto*.

4. Además, si no se especifica `inicio` se asume el valor 0 (inicio del string) y si no se especifica `fin` se asume el valor `len(x)` (fin del string). Verifique esto ejecutando:

```
print(x[:4])
print(x[5:])
print(x[5:-3])
print(x[::-1])
```

5. **Bonus track**:

   ¿Qué hacen los siguientes comandos?, ¿Modifican el valor de `x`?

   ```
   x.upper()
   x.replace("a","e")
   x.find("p")
   x.find("semestre")
   print(x)
   ```

6. Otro concepto muy importante en Python es el de *listas*. Las listas son similares a las cadenas, excepto que cada elemento puede ser de un tipo diferente. La sintaxis para crear listas en Python es [..., ..., ...]. Por ejemplo, ejecute:

   ```
   lista = [1, "hola", 1.0, 1-1j, True]
   type(lista)
   print(lista)
   ```

   Como puede ver, la variable `l` es un nuevo tipo de objeto: 'list'. En este caso, es una lista cuyos elementos son un entero, un string, un float, un complejo, y un booleano. Para verificar esto, imprima el valor y el tipo de cada elemento de la lista. Por ejemplo,

   ```
   print(lista[0],type(lista[0]))
   print(lista[1],type(lista[1]))
   ```

   Este ejemplo también muestra que los índices de cada elemento de la lista son numerados de la misma manera que en un string:

   ```
   print(lista[0:3])
   print(lista[::2])
   ```

7. Los elementos de una lista pueden tener cualquier tipo reconocido por Python, por ejemplo, pueden ser otra lista!:

   ```
   superlista=["cool",lista]
   print(superlista)
   ```

   Imprima el valor y el tipo de cada elementos de esta lista. ¿Cuántos elementos tiene la lista `superlista`? (respuesta, use la función `len()`).

8. Existen diversas funciones en Python que crean listas. La función `list()` crea una lista, por ejemplo, a partir de un string. Usando el string `x` definido anteriormente, ejecute

   ```
   y=list(x)
   print(y)
   print(type(y))
   ```

9. Otra función que crea listas útiles, esta vez de números *enteros*, es `range(inicio,fin,paso)`, que crea una lista de valores desde `inicio` (cerrado) hasta `fin` (abierto!!), con paso `paso`. Ejecute,

   ```
   z=range(2,26,3)
   print(z)
   ```

   Nuevamente, `(inicio,fin,paso)` funcionan de forma similar a los índices de un string o una lista:

   ```
   print(range(2,26))
   print(range(26,2,-1))
   ```

Bonus track: ¿Qué hacen los siguientes comandos?, ¿Modifican el valor de `x` y/o `lista`?

```
x.split(" ")
x.split("e")
lista.append("chao")
lista.insert(2,"cool")
```