Secuencias

Temas incluidos en la guía

- Secuencias
  - Cadenas (str)
  - Listas (list)
  - Tuplas (tuple)

Ejercicios

> Nota: (★☆☆, ★★☆, ★★★) Esta notación indica la dificultad (ascendente).

Cadenas

1. ★☆☆ Corrija este código para que imprima la longitud de un string:

```
print(length("Esto es un texto"))
```

2. ★☆☆ Escribir una función que cuente la cantidad de letras A que se encuentran en una palabra. Complete el siguiente código:

```
def count_A(the_string: str) -> int:
    # complete me
```

3. ★☆☆ Escriba una función `count(s, character)` que cuente cuántas veces aparece el carácter `character` en la cadena `s`. Complete el siguiente código:

```
def count(s: str, character: str) -> int:
   # complete me
```

   1. ★☆☆ Escriba un programa que le pida al usuario un carácter y un texto y luego muestre la cantidad de ocurrencias del carácter en el texto ingresado. Por ejemplo, puede seguir la siguiente interfaz:

```
texto: podemos ingresar una cadena relativamente larga y contar la
cantidad de ocurrencias de un carácter
carácter: a
> la cantidad de "a" en el texto es: 15
```

   2. ★☆☆ Escriba un programa que le pida al usuario un texto y dos caracters e indique qué carácter tiene más ocurrencias. Por ejemplo, puede seguir la siguiente interfaz:

```
texto: podemos ingresar una cadena relativamente larga y contar la
cantidad de ocurrencias de un carácter
carácter 1: a
carácter 2: e
El texto tiene más "a" que "e".
```

4. ★☆☆ Escriba una función que tome una cadena de texto y retorne una nueva cadena formada por la letra inicial, la letra del medio, y la letra final, de la cadena inicial.

   ```
   def shorten(s: str) -> str:
       # complete me
   ```

   En la tabla siguiente se muestran pares entrada/salida para la función:

msg	***shorten(msg)***	**¿Por qué nos interesa este caso?**
`""`	`""`	Cadena vacía
`"P"`	`"P"`	Cadena de largo 1
`"PM"`	`"PM"`	Cadena de largo 2
`"PMZ"`	`"PMZ"`	Cadena de largo 3
`"code"`	`"cde"`	Cadena de largo mayor a 3, par
`"hax0r"`	`"hxr"`	Cadena de largo mayor a 3, impar

5. ★☆☆ Una dirección IP (IPv4) es una cuarteto de números entre 0 y 255 (inclusive) separados por ".", y es la dirección única de cada dispositivo conectado a una red, como la dirección postal de una casa en una ciudad. Escribir una función que dada una dirección IPv4 válida, devuelva una versión modificada de esa dirección IP, a esta versión modificada la llamaremos "IP con colmillos". Una dirección IP con colmillos reemplaza cada punto "." de la dirección IP original con "[.]".

   En la tabla siguiente se muestran pares entrada/salida para la función:

IPv4	***IP con colmillos***
`"1.1.1.1"`	`"1[.]1[.]1[.]1"`
`"255.100.50.0"`	`"255[.]100[.]50[.]0"`

6. ★★☆ Algunos tipos de piedras son joyas, y todas las joyas son piedras. Vamos a representar cada piedra con un carácter, por ejemplo la "a", por lo que una cadena de caracteres representa un conjunto de piedras, por ejemplo "aaAb" es un conjunto de piedras. Dada una cadena de caracteres que representa los tipos de piedras que son joyas, y otra cadena de caracteres que representa las piedras que tienes, quieres saber cuántas joyas tienes, es decir, cuántas de las piedras que tienes son también joyas.

   Las letras distinguen entre mayúsculas y minúsculas, por lo que "a" se considera un tipo de piedra diferente de "A".

   En la tabla siguiente se muestran resultados esperados para distintos valores de la entrada:

Piedras que tienes	*Piedras que son joyas*	*Cantidad de joyas*
`"aAAbbbb"`	`"aA"`	3
`"ZZ"`	`"z"`	0

7. ★★☆ Queremos acortar el largo de un mensaje para que entre en una cantidad fija de caracteres, *n*. Si el mensaje tiene más de *n* caracteres, es decir, no entra en *n* caracteres, entonces eliminaremos el excedente y lo completaremos con "." hasta completar los *n* caracteres, y a lo sumo 3 puntos. En la tabla siguiente se muestran, dados un mensaje y un n, se muestran ejemplos del resultado esperado.

mensaje	*n*	*Resultado*	**¿Por qué nos interesa este caso?**
`""`	10	`""`	Cadena vacía
`"Mensaje"`	0	`""`	n = 0
`"Mensaje"`	1	`"."`	n = 1
`"Mensaje"`	2	`".."`	n = 2
`"Mensaje"`	3	`"..."`	n = 3
`"Mensaje"`	4	`"M..."`	n > 3, supera la cantidad de puntos
`""`	1	`""`	n es positivo y la cadena vacía
`""`	-1	`""`	n es negativo
`"Este es el mensaje largo"`	20	`"Este es el mensaj..."`	Ejemplo
`"Mensaje corto"`	20	`"Mensaje corto"`	Ejemplo
`"Mensaje corto"`	10	`"Mensaje..."`	Ejemplo

8. ★★☆ Programar una función que retorne True si un password (string) cumple con los siguientes requisitos:

   - Tiene que tener por lo menos una letra minúscula.
   - Tiene que tener por lo menos una letra mayúscula.
   - Tiene que tener por lo menos un número.
   - Tiene que tener por lo menos un signo no alfanumérico (p.ej. ! ó $).
   - Tiene que tener entre 8 y 31 caracteres.

9. ★★☆ Escribir una función que reciba una cadena, representando una fecha en formato `AAAA-MM-DD`, y retorne un True si esa fecha corresponde a Sagitario, el signo del zodiaco. Sagitario es entre el 22 de noviembre y el 21 de diciembre (ambas fechas inclusive).

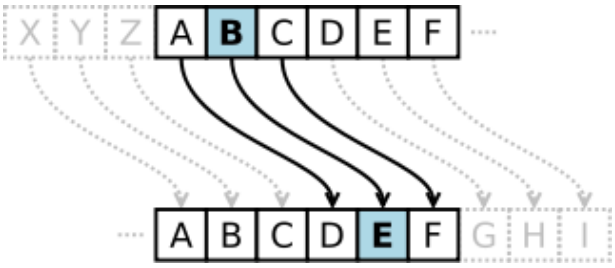
   En el formato `AAAA-MM-DD`, `AAAA` representa el año con 4 dígitos, `MM` el mes con 2 dígitos, y `DD` el día con 2 dígitos.

   La siguiente tabla le puede ayudar a validar su función:

Fecha	*¿Es de sagitario?*

Fecha	*¿Es de sagitario?*
1934-11-21	False
0836-11-22	True
1815-12-10	True
2022-12-20	True
3910-12-21	True
4096-12-22	False

10. ★★☆ El cifrado César es un tipo de cifrado de mensajes que se basa en intercambiar unas letras por otras, según un desplazamiento, como se muestra en la figura a continuación:



En este cifrado, se toman todas las letras que se van a utilizar, por ejemplo, de la A a la Z y de la a a la z. Luego, se define una rotación, un número que utilizaremos para desplazar cada letra, por ejemplo, en la figura es 3. De ese modo, la A se convertirá en una D, la B en E, la C en F, la X en A, la Y en B y la Z en C.

Implemente dos funciones, una para cifrar y una para descifrar un mensaje. Para probar el funcionamiento, considere que `decipher(cipher(msg))` devuelve el mensaje original. La siguiente tabla contiene ejemplos de cifrados.

Mensaje original	*n*	*Mensaje cifrado*
`"PENSAMIENTOCOMPUTACIONAL"`	0	`"PENSAMIENTOCOMPUTACIONAL"`
`"PENSAMIENTOCOMPUTACIONAL"`	3	`"MBKPXJFBKQLZLJMRQXZFLKXI"`
`"PENSAMIENTOCOMPUTACIONAL"`	13	`"CRAFNZVRAGBPBZCHGNPVBANY"`
`"PENSAMIENTOCOMPUTACIONAL"`	26	`"PENSAMIENTOCOMPUTACIONAL"`
`"MENSAJE"`	3	`"JBKPXGB"`
`"JBKPXGB"`	3	`"GYHMUDY"`
`"THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG"`	3	`"QEBNRFZHYOLTKCLUGRJMPLSBOQEBIXWVALD"`
`"QEBNRFZHYOLTKCLUGRJMPLSBOQEBIXWVALD"`	3	`"NBYKOCWEVLIQHZIRDOGJMIPYLNBYFUTSXIA"`

> Nota: solamente para *n = 13* las funciones son iguales, es decir, `cipher(cipher(msg, 13), 13)` da `msg`.

11. ★★☆ Escribir una función que permita determinar si una cadena podría formar una palabra, del siguiente modo: el carácter "?" sirve de comodín para cualquier letra, mientras que sea 1 (una) letra, por lo que "?oca" puede formar la palabra "poca", también la palabra "boca", o "roca". La palabra "?" puede formar cualquier cadena de largo 1, mientras que "?o?a" puede formar "poca", "bota", "rosa", etc.

La función debe cumplir el siguiente prototipo:

```
def matches(base: str, word: str) -> bool:
   # complete me
```

El siguiente código debe funcionar sin errores con la función desarrollada:

```
def test_matches():
    if matches("?", ""):
        return False
    if not matches("?", "a"):
        return False
    if matches("?", "aa"):
        return False
    if matches("x", "a"):
        return False
    if not matches("?o?a", "rota"):
        return False
    if not matches("?o?a", "poca"):
        return False
    if not matches("?o?a", "bota"):
        return False
    if not matches("?o?a", "sola"):
        return False
    if not matches("?o?a", "cosa"):
        return False
    if matches("?osa", "rota"):
        return False
    if matches("?ola", "poca"):
        return False
    if matches("co?a", "bota"):
        return False
    if matches("mo?a", "sola"):
        return False
    if matches("so?a", "cosa"):
        return False

if test_matches() is False:
    print("Esto no funciona")
```

1. ★★★ Siguiendo la misma idea, implemente una función que, además, considere al "*" como un comodín para múltiples letras, de forma tal que "*ta" coincide contra cualquiera palabra que termina en "ta", "r*ta" con cualquier palabra que comienza con "r" y termina con "ta", como "respuesta". Considere el caso en que haya más de un comodín, por ejemplo, "est?rno*le*domas*d?o" que da True para "esternocleidomastoideo".

Listas

1. ★☆☆ Programar una función que tome una lista de números y compute la suma de todos los números en la lista.

2. ★☆☆ Programar una función que elimine todas las ocurrencias de un cierto elemento en una lista.

3. ★☆☆ Programar una función que tome una lista y elimine todos los elementos duplicados.

4. ★☆☆ Programar una función que tome dos listas y retorne True si las listas tienen por lo menos un elemento en común.

5. ★☆☆ Programar una función `my_max` que tome por argumento una lista con números (int o float) y retorne el número máximo.

6. ★☆☆ Hay un lenguaje de programación con sólo cuatro operaciones y una variable, x, definido por los siguientes items:

   - Inicialmente, el valor de x siempre es 0.
   - ++x y x++ incrementa el valor de la variable x en 1.
   - --x y x-- decrementa el valor de la variable x en 1.

   Dada una lista de cadenas que contienen una lista de operaciones, devuelva el valor final de x después de realizar todas las operaciones.

   En la tabla siguiente se muestran resultados esperados para distintos valores de la entrada:

Operaciones	***Valor final de x***
`[]`	`0`
`["++x"]`	`1`
`["--x"]`	`-1`
`["--x", "x++"]`	`0`
`["++x", "x++"]`	`2`
`["++x", "x++", "x--"]`	`1`

7. ★★☆ Programar una función llamada `terminan_en_vocal` que tome por argumento una lista de strings, y devuelva otra lista con los strings que terminan en vocal.

   Pueden probar la función con las palabras que surgen de aplicar la función *split* al preámbulo de la Constitución Nacional.

8. ★★☆ Programar una función que tome como argumento un párrafo (un string con muchas palabras), y retorne una lista indicando la frecuencia con la que ocurren palabras de distinta longitud (la estructura de la lista a retornar queda a criterio del diseñador, pero debe ser explicada claramente en el docstring). Es decir, la función debe computar cuantas palabras de largo 1 hay en el párrafo, cuantas de largo 2, y así sucesivamente hasta la longitud más larga que tenga el párrafo.

   > **Nota**: los signos de puntuación no son considerados parte de la palabra, así como cualquier otro carácter que no sea una letra del abecedario.

9. ★★☆ Escribir una función que reciba una lista de listas y elimine todas las listas cuyo 3er elemento sea mayor a un número dado. Por ejemplo, si se tiene la lista: `[["W", 5, 31], ["U", 7, 18], ["F", 11, 15], ["B", 16, 28]]`, y el número 20, debe devolver la lista `[["U", 7, 18], ["F", 11, 15]]`.

10. ★★★ En este ejercicio practicaremos el concepto de filtrado: queremos quedarnos con algunos elementos de entre otros tantos, para ello:
    - Escribir una función que reciba una lista y retorne True o False dada alguna condición sobre sus elementos. Por ejemplo, una función que recibe una lista de al menos 3 elementos y retorna True si el segundo es mayor al tercero **y** la suma de estos dos es mayor 20.
    - Escribir una función llamada `filtrar` que recibe una lista `L` **y una función** `f` y retorne una lista con aquellos elementos de `L` que al aplicarles `f` (al ejecutar `f` con cada uno de ellos como argumento, individualmente) devuelven True.
    - Resolver el ejercicio anterior utilizando la función `filtrar` desarrollada en el inciso anterior.

Tuplas

1. ★☆☆ ¿Cuál es la principal diferencia entre una lista y una tupla?
2. ★★☆ Escribir una función que reciba una fecha en formato `AAAA-MM-DD` y retorne un tupla de 3 enteros con la fecha. En el formato `AAAA-MM-DD`, `AAAA` representa el año con 4 dígitos, `MM` el mes con 2 dígitos, y `DD` el día con 2 dígitos.
3. ★★★ Escribir una función que reciba una lista de tuplas, donde cada tupla contiene un nombre y una fecha (la fecha puede ser una tupla o una cadena en formato `AAAA-MM-DD`, a elección de quien programa la función). La función debe retornar True si todas las personas son de sagitario. Considere el caso particular en que la lista contenga 8 tuplas, los primeros 6 nombres sean "Gachi", "Pachi", "Lorena", "novio", "exnovio" y "Andrea", y sean todos de sagitario, en cuyo caso debe imprimirse un mensaje acorde, además de retornar el correspondiente True.