



Recordar usar las anotaciones de tipado en todas las variables. Por ejemplo: `def funcion(numero: int) -> bool:`

En los ejercicios se pueden usar funciones matemáticas como por ejemplo: `sqrt`, `round`, `floor`, `ceil`, `%`. Ver especificaciones de dichas funciones en la documentación de Python: <https://docs.python.org/es/3.10/library/math.html> y <https://docs.python.org/es/3/library/functions.html>

Revisar la especificación de las operaciones comunes sobre secuencias: <https://docs.python.org/es/3/library/stdtypes.html?highlight=list#typeseq>

1. Primera Parte

Ejercicio 1. Codificar en Python las siguientes funciones sobre secuencias:

Nota: Cada problema puede tener más de una implementación. Probar utilizando distintas formas de recorrido sobre secuencias, y distintas funciones de Python. No te conformes con una solución, recordar que siempre conviene consultar con tus docentes.

1. problema pertenece (in s:seq< \mathbb{Z} >, in e: \mathbb{Z}) : Bool {
 requiere: { True }
 asegura: { $res = true \leftrightarrow (\exists i : \mathbb{Z})(0 \leq i < |s| \rightarrow s[i] = e)$ }
}

Implementar al menos de 3 formas distintas éste problema.

¿Si la especificamos e implementamos con tipos genéricos, se podría usar esta misma función para buscar un caracter dentro de un string?

2. problema divideATodos (in s:seq< \mathbb{Z} >, in e: \mathbb{Z}) : Bool {
 requiere: { $e \neq 0$ }
 asegura: { $res = true \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow s[i] \bmod e = 0)$ }
}

3. problema sumaTotal (in s:seq< \mathbb{Z} >) : \mathbb{Z} {
 requiere: { True }
 asegura: { res es la suma de todos los elementos de s }
}

Nota: no utilizar la función `sum()` nativa

4. problema ordenados (in s:seq< \mathbb{Z} >) : Bool {
 requiere: { True }
 asegura: { $res = true \leftrightarrow (\forall i : \mathbb{Z})(0 \leq i < (|s| - 1) \rightarrow s[i] < s[i + 1])$ }
}

5. Dada una lista de palabras, devolver verdadero si alguna palabra tiene longitud mayor a 7.

6. Dada una cadena de texto (string), devolver verdadero si ésta es palíndroma (se lee igual en ambos sentidos), falso en caso contrario.

7. Analizar la fortaleza de una contraseña. El parámetro de entrada será un string con la contraseña a analizar, y la salida otro string con tres posibles valores: VERDE, AMARILLA y ROJA. **Nota:** en python la “ñ/Ñ” es considerado un caracter especial y no se comporta como cualquier otra letra.
 - La contraseña será VERDE si:
 - a) la longitud es mayor a 8 caracteres
 - b) Tiene al menos 1 letra minúscula (probar qué hace `'a' <= 'A' <= 'z'`)
 - c) Tiene al menos 1 letra mayúscula Tiene al menos 1 letra minúscula (probar qué hace `'A' <= 'A' <= 'Z'`)
 - d) Tiene al menos 1 dígito numérico (0..9)
 - La contraseña será ROJA si:
 - a) la longitud es menor a 5 caracteres.
 - En caso contrario será AMARILLA.
8. Dada una lista de tuplas, que representa un historial de movimientos en una cuenta bancaria, devolver el saldo actual. Asumir que el saldo inicial es 0. Las tuplas tienen una letra que nos indica el tipo de movimiento “I” para ingreso de dinero y “R” para retiro de dinero, y además el monto de cada operación. Por ejemplo `[('I', 2000), ('R', 20), ('R', 1000), ('I', 300)]` → `saldo = 1280`.
9. Recorrer una palabra y devolver True si ésta tiene al menos 3 vocales distintas. En caso contrario devolver False.

2. Segunda Parte

Ejercicio 2. Implementar las siguientes funciones sobre secuencias pasadas por parámetro:

1. Implementar una función que dada una lista de números, en las posiciones pares borra el valor original y coloca un cero. Esta función modifica el parámetro ingresado. **Nota:** La lista será un tipo inout.
2. Implementar la función del punto anterior pero esta vez sin modificar la lista original, devolviendo una nueva lista, igual a la anterior pero con las posiciones pares en cero. **Nota:** La lista será de tipo in.
3. Implementar una función que dada una cadena de texto de entrada (in) devuelva una cadena igual a la anterior, pero sin las vocales. **Nota:** No agrega espacios, sino que borra la vocal y concatena a continuación.
4. problema `reemplazaVocales` (in s:seq<Char>) : seq<Char> {


```

      requiere: { True }
      asegura: { (∀i : ℤ)(0 ≤ i < |res| → (pertenece(<'a','e','i','o','u', s[i]) ∧ res[i] = '_') ∨
        (¬ pertenece(<'a','e','i','o','u', s[i]) ∧ res[i] = s[i] ) ) }
      }
      
```
5. problema `daVueltaStr` (in s:seq<Char>) : seq<Char> {


```

      requiere: { True }
      asegura: { (∀i : ℤ)(0 ≤ i < |res| → (res[i]=s[|s|-i-1]) ) }
      }
      
```

Ejercicio 3. Vamos a elaborar programas interactivos (usando la función `input()`¹) que nos permita solicitar al usuario información cuando usamos las funciones.

1. Implementar una función para construir una lista con los nombres de mis estudiantes. La función solicitará al usuario los nombres hasta que ingrese la palabra “listo”. Devuelve la lista con todos los nombres ingresados.
2. Implementar una función que devuelve una lista con el historial de un monedero electrónico (por ejemplo la SUBE). El usuario debe seleccionar en cada paso si quiere:
 - “C” = Cargar créditos,
 - “D” = Descontar créditos,
 - “X” = Finalizar la simulación (terminar el programa).

¹<https://docs.python.org/es/3/library/functions.html?highlight=input#input>

En los casos de cargar y descontar créditos, el programa debe además solicitar el monto para la operación. Vamos a asumir que el monedero comienza en cero. Para guardar la información grabaremos en el historial tuplas que representen los casos de cargar (“C”, monto a cargar) y descontar crédito (“D”, monto a descontar).

3. Vamos a escribir un programa para simular el juego conocido como 7 y medio. El mismo deberá generar un número aleatorio entre 0 y 12 (excluyendo el 8 y 9) y deberá luego preguntarle al usuario si desea seguir sacando otra “carta” o plantarse. En este último caso el programa debe terminar. Los números aleatorios obtenidos deberán sumarse según el número obtenido salvo por las “figuras” (10, 11 y 12) que sumarán medio punto cada una. El programa deberá ir acumulando los valores y si se pasa de 7.5 deberá informar que el usuario ha perdido. Al finalizar la función devuelve el historial de “cartas” que hizo que el usuario gane o pierda. Nota: Para esta función utilizaremos la función `random.randint(1, 12)` para generar números aleatorios entre 1 y 12. **Nota:** La función `random.choice()` puede ser de gran ayuda a la hora de repartir cartas.

Ejercicio 4. Implementar las siguientes funciones sobre listas de listas:

1. **problema perteneceACadaUno** (in s:seq<seq< \mathbb{Z} >>, in e: \mathbb{Z} , out res: seq<Bool>) {
`requiere:` { True }
`asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow (res[i] = \text{true} \leftrightarrow \text{pertenece}(s[i], e)))$ }
}

Nota: Reutilizar la función `pertenece()` implementada previamente para listas

2. **problema esMatriz** (in s:seq<seq< \mathbb{Z} >>) : Bool {
`requiere:` { True }
`asegura:` { $res = \text{true} \leftrightarrow (|s| > 0) \wedge (|s[0]| > 0) \wedge (\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow |s[i]| = |s[0]|)$ }
}

3. **problema filasOrdenadas** (in m:seq<seq< \mathbb{Z} >>, out res: seq<Bool>) {
`requiere:` { esMatriz(m) }
`asegura:` { $(\forall i : \mathbb{Z})(0 \leq i < |res| \rightarrow (res[i] = \text{true} \leftrightarrow \text{ordenados}(s[i])))$ }
}

Nota: Reutilizar la función `ordenados()` implementada previamente para listas

4. Implementar una función que tome un entero d y un float p y eleve una matriz cuadrada de tamaño d (con valores generados al azar) a la potencia p . Es decir, multiplique a la matriz por sí misma p veces. Luedo de implementarla probar con diferentes valores de d . ¿Qué pasa con valores muy grandes?

Nota 1: recordá que en la multiplicación de una matriz cuadrada de dimensión d por si misma cada posición se calcula como $res[i][j] = \sum_{n=0}^{d-1} m[i][n] * m[n][j]$

Nota 2: para generar una matriz cuadrada de dimensión d con valores aleatorios hay muchas opciones de implementación, analizar las siguientes usando el módulo `numpy` (ver recuadro):

Opción 1:

```
import numpy as np
m = np.random.random((d, d))2
```

Opción 2:

```
import numpy as np
m = np.random.randint(i,f, (d, d))3
```

Para poder importar la librería `numpy` es necesario instalarla. Para ello es necesario tener instalado un gestor de paquetes, por ejemplo `pip` (**Ubuntu:** `sudo apt install pip3`. **Windows:** se instala junto con Python). Una vez instalado `pip` se ejecuta `pip install numpy`.

²<https://numpy.org/doc/stable/reference/random/generated/numpy.random.Generator.random.html#numpy.random.Generator.random>

³<https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html>