Listas en Python

Grado 10° Guía 2

TIC

Estudiantes

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

Listas en Python

Grado 10° Guía 2

TIC

Estudiantes

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Julián Molina Gómez
Ministro TIC

Luis Eduardo Aguiar Delgadillo
Viceministro (e) de Conectividad

Yeimi Carina Murcia Yela
Viceministra de Transformación Digital

Óscar Alexander Ballen Cifuentes
Director (e) de Apropiación de TIC

Alejandro Guzmán
Jefe de la Oficina Asesora de Prensa

Equipo Técnico
Lady Diana Mojica Bautista
Cristhiam Fernando Jácome Jiménez
Ricardo Cañón Moreno

Consultora experta
Heidy Esperanza Gordillo Bogota

BRITISH COUNCIL

Felipe Villar Stein
Director de país

Laura Barragán Montaña
Directora de programas de Educación, Inglés y Artes

Marianella Ortiz Montes
Jefe de Colegios

David Vallejo Acuña
Jefe de Implementación Colombia Programa

Equipo operativo
Juanita Camila Ruiz Díaz
Bárbara De Castro Nieto
Alexandra Ruiz Correa
Dayra Maritza Paz Calderón
Saúl F. Torres
Óscar Daniel Barrios Díaz
César Augusto Herrera Lozano
Paula Álvarez Peña

Equipo técnico
Alejandro Espinal Duque
Ana Lorena Molina Castro
Vanesa Abad Rendón
Raisa Marcela Ortiz Cardona
Juan Camilo Londoño Estrada

Edición y coautoría versiones finales
Alejandro Espinal Duque
Ana Lorena Molina Castro
Vanesa Abad Rendón
Raisa Marcela Ortiz Cardona

Edición
Juanita Camila Ruiz Díaz
Alexandra Ruiz Correa

British Computer Society – Consultoría internacional

Niel McLean
Jefe de Educación

Julia Adamson
Directora Ejecutiva de Educación

Claire Williams
Coordinadora de Alianzas

Asociación de facultades de ingeniería - ACOFI

Edición general
Mauricio Duque Escobar

Coordinación pedagógica
Margarita Gómez Sarmiento
Mariana Arboleda Flórez
Rafael Amador Rodríguez

Coordinación de producción
Harry Luque Camargo

Asesoría estrategia equidad
Paola González Valcárcel

Asesoría primera infancia
Juana Carrizosa Umaña

Autoría
Arlet Orozco Marbello
Harry Luque Camargo
Isabella Estrada Reyes
Lucio Chávez Mariño
Margarita Gómez Sarmiento
Mariana Arboleda Flórez
Mauricio Duque Escobar
Paola González Valcárcel
Rafael Amador Rodríguez
Rocío Cardona Gómez
Saray Piñerez Zambrano
Yimzay Molina Ramos

PUNTOAPARTE EDITORES

Diseño, diagramación, ilustración, y revisión de estilo

Impreso por Panamericana Formas e Impresos S.A., Colombia

Material producido para Colombia Programa, en el marco del convenio 1247 de 2023 entre el Ministerio de Tecnologías de la Información y las Comunicaciones y el British Council

"Esta guía corresponde a una versión preliminar en proceso de revisión y ajuste. La versión final actualizada estará disponible en formato digital y puede incluir modificaciones respecto a esta edición"

Prólogo

Estimados educadores, estudiantes y comunidad educativa:

En el Ministerio de Tecnologías de la Información y las Comunicaciones, creemos que la tecnología es una herramienta poderosa para incluir y transformar, mejorando la vida de todos los colombianos. Nos guia una visión de tecnología al servicio de la humanidad, ubicando siempre a las personas en el centro de la educación técnica.

Sabemos que no habrá progreso real si no garantizamos que los avances tecnológicos beneficien a todos, sin dejar a nadie atrás. Por eso, nos hemos propuesto una meta ambiciosa: formar a un millón de personas en habilidades que les permitan no solo adaptarse al futuro, sino construirlo con sus propias manos. Hoy damos un paso fundamental hacia este objetivo con la presentación de las guías de pensamiento computacional, un recurso diseñado para llevar a las aulas herramientas que fomenten la creatividad, el pensamiento crítico y la resolución de problemas.

Estas guías no son solo materiales educativos; son una invitación a imaginar, cuestionar y crear. En un mundo cada vez más impulsado por la inteligencia artificial, desarrollar habilidades como el pensamiento computacional se convierte en la base, en el primer acercamiento para que las y los ciudadanos aprendan a programar y solucionar problemas de forma lógica y estructurada.

Estas guías han sido diseñadas pensando en cada región del país, con actividades accesibles que se adaptan a diferentes contextos, incluyendo aquellos con limitaciones tecnológicas. Esta es una apuesta por la equidad, por cerrar las brechas y asegurar que nadie se quede atrás en la revolución digital. Quiero destacar, además, que son el resultado de un esfuerzo colectivo: más de 2.000 docentes colaboraron en su elaboración, compartiendo sus ideas y experiencias para que este material realmente se ajuste a las necesidades de nuestras aulas. Además, con el apoyo del British Council y su red de expertos internacionales, hemos integrado prácticas globales de excelencia adaptadas a nuestra realidad nacional.

Hoy presentamos un recurso innovador y de alta calidad, diseñado en línea con las orientaciones curriculares del Ministerio de Educación Nacional. Cada página de estas guias invita a transformar las aulas en espacios participativos, creativos y, sobre todo, en ambientes donde las y los estudiantes puedan desafiar estereotipos y explorar nuevas formas de pensar.

Trabajemos juntos para garantizar que cada estudiante, sin ·importar dónde se encuentre, tenga acceso a las herramientas necesarias para imaginar y construir un futuro en el que todos seamos protagonistas del cambio. Porque la tecnología debe ser un instrumento de justicia social, y estamos comprometidos a que las herramientas digitales ayuden a cerrar brechas sociales y económicas, garantizando oportunidades para todos.

Con estas guias, reafirmamos nuestro compromiso con la democratización de las tecnologías y el desarrollo rural, porque creemos en el potencial de cada región y en la capacidad de nuestras comunidades para liderar el cambio.

Julián Molina Gómez
Ministro de Tecnologías de la Información y las Comunicaciones
Gobierno de Colombia

Grado 10° Guía 2

TIC

Guía de íconos

 Lógica, programación y depuración

Aprendizajes de la guía

Con las actividades de esta guía se espera que puedas avanzar en:

- Aplicar la descomposición, abstracción y pensamiento algorítmico para resolver retos y tareas.
- Declarar y asignar variables de diferentes tipos y usarlas de manera apropiada.
- Crear casos de prueba para validar la ejecución de sus programas.

Resumen de la guía

Esta guía propone 5 sesiones de trabajo orientadas a continuar desarrollando habilidades de programación en *Python*, asociadas al uso de listas de una y dos dimensiones. La clase aprenderá métodos comunes para insertar, hallar y eliminar elementos de una lista.

Resumen de las sesiones

Sesión 1	Se busca realizar una introducción corta a *Python* y se refrescan conocimientos sobre el manejo de variables y sus operaciones básicas.
Sesión 2	Se trabaja el concepto de listas con actividades desconectadas a través de ejemplos de la vida cotidiana.
Sesión 3	Se aprende a utilizar listas unidimensionales en *Python* y sus funciones básicas para añadir, eliminar y acceder a los elementos.
Sesión 4	Se aprende a iterar sobre una lista, realizar operaciones sobre los elementos y hacer comparaciones con otros elementos o condiciones.

Aprendizajes de la guía

- Interpretar los mensajes de error y usarlos para identificar y corregir los errores de lógica y sintaxis en sus programas.
- Automatizar tareas simples relacionadas a otras áreas de conocimiento.

Sesión 5

Se aborda el diseño y solución de un reto que requiere el uso de listas en *Python*.

Si se requiere

- Guías 1 y 2 de Grado 9 para aprendizajes iniciales sobre *Python*.

Conexión con otras áreas

A continuación, se brindan puntos de conexión de los temas abordados en las sesiones con otras áreas:

Matemáticas

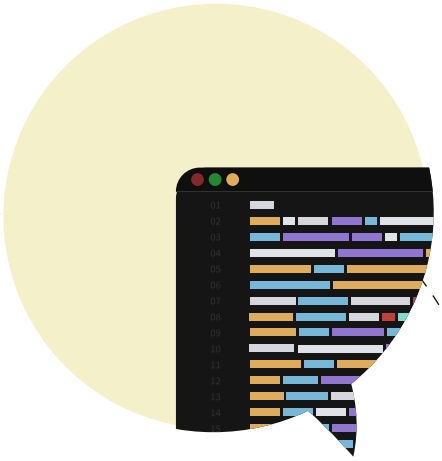
- En el caso de matemáticas, las actividades incluyen operaciones básicas en el manejo de listas, como el uso de índices para acceder a elementos guardados en listas y hacer cálculos. Esto implica establecer relaciones entre conjuntos de datos, como tablas o listas.

Ciencias Naturales

- El diseño de sistemas para clasificar datos, iterar sobre listas o validar elementos de una lista, fortalece la habilidad de organizar información. Esto está relacionado con el análisis y manejo de datos científicos, necesarios en las ciencias naturales o sociales.

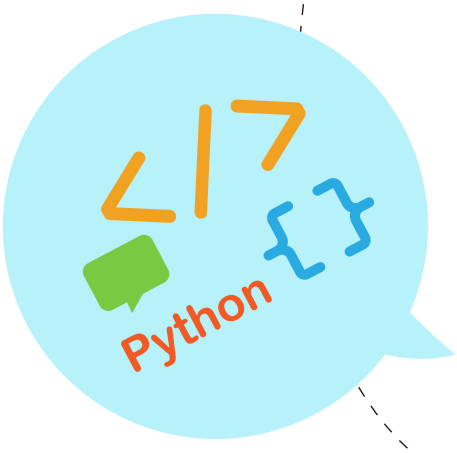
Lenguaje

- Esta guía promueve la habilidad de identificar errores de lógica y sintaxis en el texto, lo cual tiene relación con la lectura crítica y la precisión en la escritura.



En esta guía se encuentran varias oportunidades relacionadas con el cierre de brechas al motivar a las personas hacia el pensamiento computacional:

- En la asignación de los grupos para cada sesión de trabajo.
- En el seguimiento del trabajo de los grupos para evitar que algunos de sus integrantes monopolicen las actividades.
- En la sesión 5, al final, podrán escoger el reto que deseen o les sea de mayor interés.



Grado 10° Guía 2

TIC

Sesión 1

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

- Interpretar los mensajes de error y usarlos para identificar y corregir los errores de lógica y sintaxis en tus programas.
- Automatizar tareas simples relacionadas a otras áreas de conocimiento.
- Utilizar casos de prueba sencillos para evaluar un código.

Material para la clase

- ○ Acceso a un computador con internet.
- ○ Anexos 1.1, 1.2 y 1.3 A y B.

Duración sugerida

15% 70% 15%

Nota

Python se puede utilizar con herramientas de accesibilidad, como lectores de pantalla y software de reconocimiento de voz, en caso de ser necesario.

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

¿Cuántos idiomas conoces? ¿Sabes decir alguna palabra en un idioma diferente al español? Antes de empezar la clase, comparte con la clase algunas palabras que conozcas en otro idioma. ¿Cómo las aprendiste?

Tal vez te sorprenda descubrir que alguien de tu clase se aprendió una canción en coreano, o que una de tus compañeras habla un idioma diferente al español en su casa. De hecho, en Colombia se hablan 70 lenguas: el castellano, 65 lenguas indígenas, 2 lenguas criollas (palenquero de San Basilio y la de las islas de San Andrés y Providencia - creole), la Romaní o Romaníes del pueblo Rrom – Gitano y la lengua de señas colombiana.

Ahora mira esta imagen, en español le llamamos mariposa, pero ¿sabes cómo se llama en otros idiomas?

A continuación, te presentamos una lista con la escritura de esta palabra en algunos idiomas, ¿conocías alguno de estos términos? ¿Puedes completar la lista agregando cómo se dice mariposa en otros idiomas diferentes?

- Emberá eyabida: Bôbô
- Wayuunaiki: julirü
- Portugués: borboleta

- Euskera: tximeleta
- Criollo haitiano: papiyon
- Alemán: Schmetterling

Vuelve a leer las palabras y analízalas. ¿Encuentras algún parecido entre ellas? ¿Podrías haber adivinado su significado con solo escucharlas?

Como puedes notar, no existen muchas similitudes entre las palabras, algunas son cortas y otras, como en el alemán, usan muchas letras. Sin embargo, todas se refieren a lo mismo: una mariposa.

En programación pasa algo similar. Así como existen diferentes idiomas, existen diferentes lenguajes de programación. Cada lenguaje suele asociarse a un uso en particular, sin embargo, existen conceptos que se comparten y habilidades que son útiles, sin importar en qué lenguaje programes.

Mira las imágenes a continuación, ¿reconoces algún lenguaje? ¿Cuáles has usado? ¿Puedes explicar qué hacen los códigos? ¿Notas algunas diferencias entre cada uno?

Figura 1. Programa en lenguaje de programación basado en bloques - Scratch

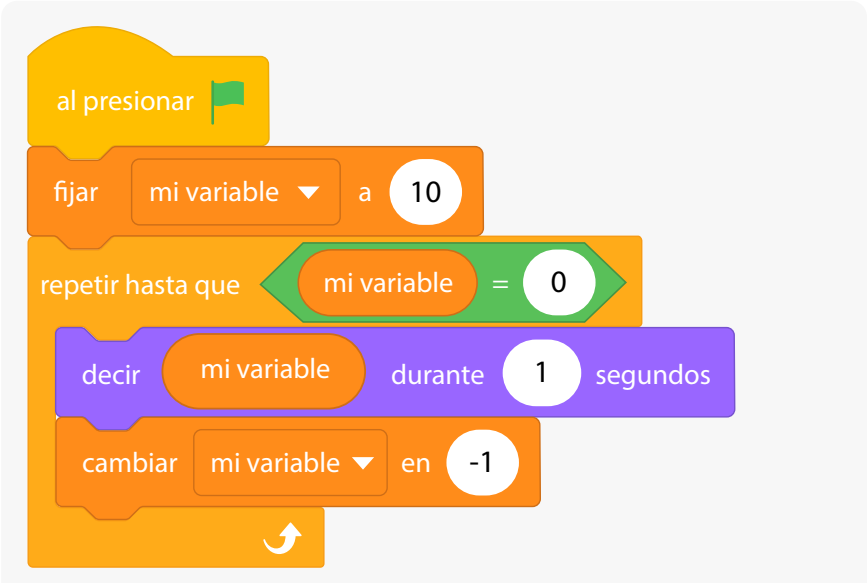


Figura 2. Programa en lenguaje de programación basado en bloques - MakeCode

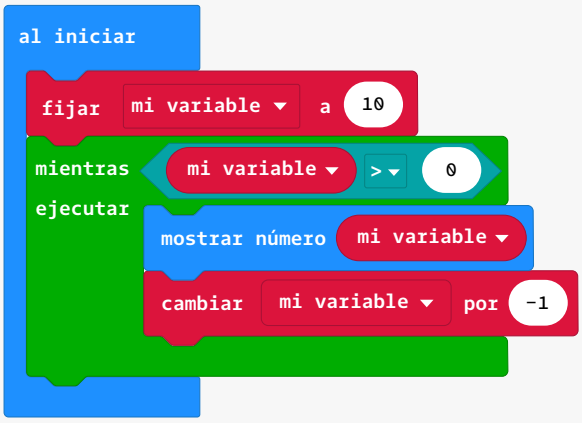


Figura 3. Programa en lenguaje de programación basado en texto - JavaScript

```
let mi_variable = 10
while (mi_variable › 0 ) {
        basic.showNumber(mi_variable)
        mi_variable += -1
}
```

Figura 4. Programa en lenguaje de programación basado en texto - Arduino

```
void loop () {
mi_variable = 10;
while (mi_variable › 0) {
        Serial.println(mi_variable);
        mi_variable--;
        delay(1000);
}
}
```

Figura 5. Programa en lenguaje de programación basado en texto - Python

```
mi_variable = 10
while mi_variable › 0:
        print(mi_variable)
        mi_variable -= 1
```

Al igual que en el ejemplo de la mariposa, las imágenes muestran cinco formas diferentes de lograr un objetivo similar, utilizando diferentes lenguajes. En este caso, las imágenes presentaban dos lenguajes basados en bloques, y tres en texto así: *1. Scratch, 2. MakeCode, 3. JavaScript, 4. Arduino y 5. Python.*

Como puedes observar, algunos lenguajes utilizan bloques, otros necesitan utilizar símbolos especiales como el punto y coma (;) para separar cada instrucción y otros necesitan cambios en la **indentación** o el espacio entre instrucciones. Durante esta guía estarás trabajando con *Python*.

Es posible que recuerdes que *Python* es un lenguaje de alto nivel muy popular, ya que puede ser utilizado para crear diferentes tipos de aplicaciones. Además, su principal ventaja es que puede ser fácil de leer ya que usa palabras comunes (en inglés) y no requiere de muchos símbolos en su escritura, en comparación a otros lenguajes. Esta característica permite que quienes programan se concentren más en resolver problemas que en la **sintaxis** del lenguaje.

A lo largo de esta guía, aprenderás a crear programas más avanzados en *Python*. En la siguiente actividad, vas a repasar las funciones que ya conoces y practicar cómo encontrar y corregir errores en tu código. Así vas a prepararte para los próximos retos.

Glosario

Indentación: la indentación en *Python* es el uso de espacios o tabulaciones al comienzo de una línea de código para definir la estructura del programa. Es crucial para indicar qué bloques de código pertenecen a una función, bucle o condicional.

Sintaxis: es el conjunto de reglas que define cómo deben escribirse y estructurarse los comandos y expresiones en un lenguaje de programación para que sean entendidos y ejecutados correctamente por la computadora.

Anexos

Anexo 1.1

Anexo 1.2

Anexo 1.3

Manos a la obra

Desconectadas

Esta sección corresponde al 85% de avance de la sesión

En esta actividad recordarás algunos de los comandos básicos de *Python* que has aprendido y reforzarás tus habilidades de depuración. Trabajarás en parejas, utilizando un editor de *Python*, como *Anaconda*, *Google Colab*, o *Python-Online*, dependiendo de lo que tu docente te haya indicado. Si tienes dudas sobre cómo acceder a alguno de estos editores, consulta el *Anexo 1.1*, que contiene tutoriales detallados para cada opción.

Para empezar, forma un grupo de dos personas y desarrollen el *Anexo 1.2*, el cual cuenta con dos ejercicios:

1. Emparejamiento de funciones y su uso: aquí recordarán cómo usar funciones como **input()**, **print()**, bucles **while**, condicionales **if**, y operadores como **=** y **==**. Unan cada función con su propósito.

2. Análisis de un código: lean cuidadosamente el código que se presenta y detecten posibles errores o comportamientos inesperados. Además, respondan las preguntas para identificar las variables, expresiones aritméticas, expresiones booleanas y las líneas que interactúan con el usuario.

Cuando terminen, levanten la mano para que su docente les entregue el *Anexo 1.3* para continuar con la actividad. En este, encontrarán nuevos retos que pondrán a prueba sus habilidades de programación:

1. Ambos trabajarán juntos en el ejercicio común: un programa que calcula la edad de una persona según su año de nacimiento. Sin embargo, este programa tiene un error. El objetivo es identificar y corregir el error y luego pensar cómo mejorarlo, por ejemplo, para que funcione correctamente en otros años o para más de una persona.

2 Cada estudiante recibirá un reto diferente. Si te asignan el reto A, deberás diseñar el código para programar una calculadora de pesos en diferentes planetas. Si te asignan el reto B, deberás diseñar el código de un conversor de unidades. El reto estará en que cada persona debe dictarle su solución a la otra persona y superar varios casos de prueba.

Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1 ¿Puedes interpretar los mensajes de error y usarlos para identificar y corregir los errores en sus programas (lógica, sintaxis)?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2 ¿Puedes automatizar tareas simples relacionadas a otras áreas de conocimiento?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3 ¿Puedes utilizar casos de prueba sencillos para evaluar un código?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Para cerrar la sesión te sugerimos reflexionar sobre las siguientes preguntas:

¿Cómo te sentiste durante el tiempo de planeación del código? ¿Crees que habría sido más fácil empezar a programar directamente? ¿Por qué?

¿Fue fácil dictarle el código a tu compañero(a)? ¿Cómo crees que afectó tu solución tener que pedirle a alguien más que escribiera por ti?

¿Puedes pensar en otros ejemplos de problemas que podrías automatizar?

Aprovecha unos minutos para discutir tus respuestas y las dudas que te queden con el resto de la clase. Además, pueden comenzar la construcción de una memoria colectiva con las palabras y funciones que requieran.

Grado 10° | Guía 2

TIC

Sesión 2

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

- Reconocer las listas como un tipo de variable que contiene varios elementos.
- Extraer los valores e índices de los elementos de una lista.
- Utilizar elementos de una lista en una función.

Duración sugerida

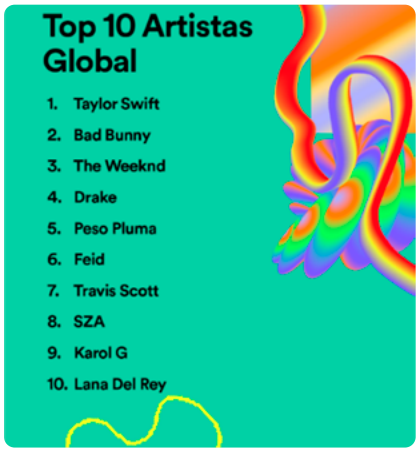
40% 40% 20%

Material para la clase

- ○ Dados o generador de números virtual.

Figura 1. Diferentes Rankings

Pos	Club	20/21
1	Barcelona España	22.000
2	Lyon Francia	15.000
3	Chelsea Inglaterra	20.000
4	Paris Francia	15.000
5	Bayern Alemania	20.000
6	Wolfsburg Alemania	13.000
7	Arsenal Inglaterra	-



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

¿Qué es lo primero en lo que piensas cuando escuchas la palabra **lista**? Tal vez pienses en algo relacionado con el mercado o una de tareas pendientes.

Teniendo esto en cuenta, haz una lluvia de ideas sobre las diferentes listas que se usan o crean en el día a día.

¿Qué identificaste? ¿Qué mencionaron otras compañeras y compañeros?

Además de las que ya se han mencionado, hay muchos otros tipos de listas. Por ejemplo:

El listado de los mejores equipos de fútbol femenino según la UEFA (Unión de Federaciones Europeas de Fútbol), ell top 10 de artistas globales de la plataforma de música Spotify , o la lista de las mejores universidades de Colombia ver *Figura 1*.

¿Qué tienen en común estas listas? ¿Qué condiciones cumplen los elementos de cada una de las listas? ¿Se te ocurren otros ejemplos?

Ahora mira la *Figura 1* y responde:

¿Cuál fue la mejor universidad de Colombia en el 2020?
¿Quién fue el artista masculino más escuchado, según la imagen?
¿En qué puesto de la lista estaba Feid?
¿Cuál es el tercer mejor equipo de fútbol, según la imagen?

Figura 2. Listado de clase

AÑO ACADÉMICO 2016/2017

LISTA DE CLASE DE: 001A (1ºC2EI)

TUTORA : Jiménez Gil, Inmaculada

Nº orden	Nº Expediente	Alumno
1	2146	Amado Diez, Valeria
2	2154	Aparicio Mateos, María del Carmen
3	2106	Bastos Salgado, Hugo
4	2107	Bernardo Serrano, Gonzalo
5	2165	Cambero Cisneros, Nerea
6	2098	Campos Gómez, Marina
7	2135	Cano Pavón, David
8	2084	Carretero Pacheco, Irene
9	2087	Carretero Pacheco, Paula
10	2138	Cordero Declara, Carla
11	2119	Cortés Acha, Emilio
12	2102	Cortés Palomino, Rodrigo
13	2151	Delgado Berenguer, Ángel
14	2136	Díaz González, Alejandro
15	2100	Díaz Hack, Abigail
16	2101	Durán Araque, Paula
17	2088	Durán Muñoz, Eneas
18	2143	Escribano Martínez, Lara
19	2108	García Cercas, Elvira
20	2130	García Recuero, Celia
21	2099	Gil Martín, Pablo Eloy
22	2095	Martos Mostazo, Jaime
23	2093	Mostazo Quirós, Pablo
24	2094	Pérez Crego, Nicolás
25	2121	Solís Peña, Marta

Las **listas** son colecciones de **elementos** que se parecen entre sí y que se almacenan de forma secuencial y ordenada. Como puedes notar, no tendría sentido escribir el nombre de una universidad en la lista de Spotify. Sin embargo, las características de cada lista son muy parecidas entre sí.

Cada elemento de una lista tiene dos partes muy importantes: el **índice** o posición, y el **valor**.

Piensa en una lista de clase. Al iniciar el año escolar, se asigna un número a cada estudiante. Ese número es su posición en la lista. Los valores corresponden al nombre, las calificaciones, las faltas de asistencia y demás información de la que se lleve registro y que esté asociada a cada número de lista, es decir, a cada posición.

En la *Figura 2*, el primer recuadro encierra las posiciones y el segundo encierra los valores.

Según la imagen del ejemplo:

¿?

¿Qué crees que retornaría el código print(alumno[24])?
¿Y print(alumno[3])?
¿Y print(expediente[17])?

En programación, se usa la notación **lista[índice]** para acceder a los elementos de una lista. Y se lee “lista en índice".

Reconocer que todos los elementos de una lista tienen un índice (o posición) y un valor, es un concepto fundamental de la programación y permitirá utilizar **estructuras de datos** de una manera más fácil.

Glosario

Lista: es una colección ordenada de elementos que pueden modificarse y accederse mediante índices.

Estructura de datos: es una forma de organizar y almacenar datos para facilitar su acceso y manipulación eficiente.

Manos a la obra

Desconectadas

Esta sección corresponde al 80% de avance de la sesión

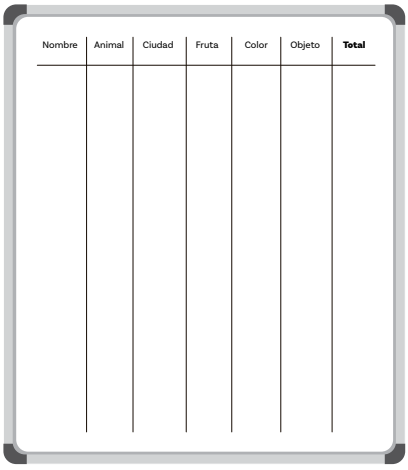
A continuación, tu docente dirigirá una actividad para practicar los conceptos asociados a las listas. Como clase, deben completar las siguientes listas escribiendo los valores en el tablero. Ten en cuenta que las personas pueden ser reales o ficticias, por tanto, los superhéroes y personajes de historias son válidos.

Índice	Sustantivo
0	
1	
2	
3	
4	
5	
6	
7	
8	ninja
9	
10	
11	

Índice	Cualidad
0	famoso
1	
2	
3	
4	
5	
6	
7	
8	
9	extravagante
10	
11	

Índice	Lugar
0	
1	
2	Espacio exterior
3	
4	
5	
6	
7	
8	
9	
10	
11	

Índice	Persona
0	
1	Betty
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	



Nota

Ten en cuenta que en las listas la posición inicial es la 0. Por tanto, si obtienes 12 al lanzar los dados, debes anotar el valor que se encuentra en la posición 11 de la tabla.

Una vez estén completas las 4 listas, responde:

¿Qué retornaría el código print ***(cualidad[0])****?*
*¿Qué retornaría print****(lugar[2])****?*
*¿Qué retornaría print****(persona[1])****?*
¿Cómo se escribiría la instrucción para imprimir ***ninja****?*

En una ronda de pensar rápido tu docente o una compañera o compañero asignado, irán pidiendo elementos y el resto del grupo debe decirlos en voz alta. Por ejemplo, "sustantivo[6], cualidad[8], persona[7]" y así sucesivamente. Además, se puede alternar y mencionar el valor de una lista y el grupo debe decir la instrucción requerida para imprimir esa respuesta. Por ejemplo, si quien lidera dice "Betty", el grupo debe responder print ("persona[1]").

Ahora, vas a utilizar las listas para crear el nombre de una película. Utilizando dos dados, vas a hacer cuatro lanzamientos y a anotar los resultados.

Primer_lanzamiento: _____

Segundo_lanzamiento: _____

Tercer_lanzamiento: _____

Cuarto_lanzamiento: _____

Ahora, sigue la fórmula para obtener el título de una historia:

El/La **sustantivo[primer_lanzamiento]** + **cualidad[segundo_lanzamiento]** + va a + **lugar[tercer_lanzamiento]** + junto a su fiel compañera + **persona[cuarto_lanzamiento]**.

Compara con los resultados de tu grupo.

¿Qué obtuvieron? ¿Se repitió algún título? ¿Cómo se verían los lanzamientos de los dados guardados en una lista? ¿Cómo cambiaría la fórmula para obtener el título de la historia?

La *Figura 3* tiene un propósito similar. Convierte la información en listas y escribe la fórmula para obtener el título de la película de tu vida. Guíate del ejemplo anterior.

Figura 3. Elementos para elaborar un guión de película

Listas

Fórmula

Al finalizar, sigue las instrucciones de tu docente para comparar las semejanzas y diferencias que tengas con la fórmula y listado creado por tus compañeras o compañeros.

Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1. ¿Puedes reconocer las listas como un tipo de variable que contiene varios elementos?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2. ¿Puedes extraer los valores e índices de los elementos de una lista?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3. ¿Puedes utilizar elementos de una lista en una función?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

> **Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", revisa nuevamente los contenidos de la sesión y consulta las inquietudes que todavía tienes con tu docente.**

Para afianzar tus aprendizajes e identificar las dudas y resolverlas, te proponemos una rutina de pensamiento llamada 3-2-1 Puente.

Completa el siguiente diagrama escribiendo tres ideas o pensamientos sobre lo visto en la sesión, dos preguntas que tengas sobre el tema y una analogía o ejemplo que te permita recordar lo aprendido. Guarda tu diagrama porque lo volverás a utilizar en las próximas sesiones.

Respuesta inicial

3 ideas

2 preguntas

1 analogía

Grado 10° Guía 2

TIC

Sesión 3

Aprendizajes esperados

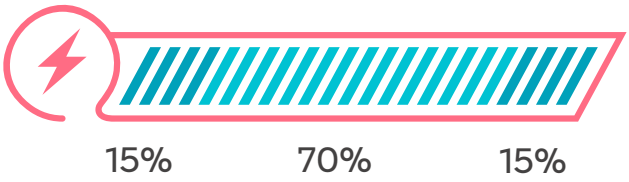
Al final de esta sesión se espera que puedas:

- Utilizar listas en un lenguaje de programación y realizar operaciones sencillas con sus elementos.
- Fortalecer las habilidades de depuración de errores de lógica y sintaxis en programas sencillos en *Python*.

Material para la clase

- Anexo 3.1.

Duración sugerida



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

Las **estructuras de datos** son una parte fundamental de la informática que nos permiten organizar y almacenar datos de manera eficiente para que podamos acceder a ellos y manipularlos fácilmente.

En la sesión anterior aprendiste sobre las listas. Las listas son estructuras de datos que permiten guardar varios elementos de manera secuencial.

Lista = | 0 | 20 | -6 | 13 | 41 | -8 |

Como aprendiste, es posible acceder a cada uno de los elementos utilizando su índice o posición.

¿Cómo podrías acceder al -6 de la imagen?
¿Qué valor retornaría Lista[1]?
¿Cuál dirías que es la longitud de esta lista?

Existen muchas operaciones que se pueden hacer sobre las listas. En esta sesión vas a aprender cómo crear listas, agregar, insertar y eliminar elementos. Además, vas a aprender cómo conocer algunas propiedades de una lista

Para crear una lista en *Python*, debes colocar los elementos dentro de corchetes [ ], separados por comas. Veamos dos ejemplos:

```
# Ejemplo de una lista de números
numeros = [1, 2, 3, 4, 5]

# Ejemplo de una lista de palabras
palabras = ["manzana", "banana", "cereza"]
```

¿Qué tienen en común los ejemplos? ¿En qué se diferencian?
¿Qué significan los diferentes colores?
¿Qué pasaría si olvidas utilizar las comillas para escribir los elementos de la segunda lista?

Comenta tus respuestas con la persona que tienes al lado.
Ten en cuenta que, en los lenguajes de programación, así como en los idiomas, es muy importante la ortografía. En programación se le llama **sintaxis**. Por eso, cuando olvidas un símbolo, o no utilizas los símbolos adecuados, *Python* te alerta mostrando un mensaje de SyntaxError como el de la imagen.

```
1  a = 2
2  if(a == 2)
3  print("True")
```

```
File "<ipython-input-4-ffea50a12548>", line 2
  if(a == 2)

SyntaxError: invalid syntax
```

Cada vez que veas un error, vuelve a revisar tu código detenidamente. Puede ser que hayas confundido un paréntesis (con un corchete [, que hayas olvidado cerrar una expresión, o combinado dos tipos de comillas diferentes (“,‘).

Manos a la obra
Desconectadas

Esta sección corresponde al 85% de avance de la sesión

Ingresa a tu editor de *Python* y trabaja en la siguiente actividad.

Imagina que quieres hacer una lista de compras. Al principio no sabes exactamente qué necesitas comprar, por lo que debes revisar tu nevera. A medida que revisas la nevera, irás agregando nuevos elementos a tu lista, que inicialmente está vacía.

Para esto vas a utilizar el método **append**. Este método se usa para añadir un elemento al final de una lista. Es especialmente útil cuando no sabes cuántos elementos necesitarás inicialmente y prefieres ir agregándolos según los vayas obteniendo.

Observa el siguiente código de *Python* y predice qué crees que hará cuando lo ejecutes. ¿Cómo crees que se irá llenando la lista? ¿Qué sucederá cuando añadas elementos nuevos?

```
compras = []

compras.append("manzana")
compras.append("tomate")
compras.append("leche")

print(compras)
```

Analiza el código y responde:

¿?

¿Qué función cumple el método append en este código?
¿Cómo se vería la lista después de ejecutar las primeras tres líneas de código?
¿Qué sucede si agregas más elementos usando append?

Ahora ingresa el código en tu editor de *Python* (puedes usar *Anaconda*, *Google Colab* o el editor que tu docente te indique) y ejecútalo. Una vez veas los resultados, responde:

¿?

¿Qué pasó cuando ejecutaste el código?
¿Cómo puedes acceder a cada elemento de la lista usando índices?
¿Qué pasa si cambias los corchetes de la lista y los reemplazas por paréntesis?
¿Qué pasaría si olvidas escribir los elementos entre comillas ("")?
¿Qué pasaría si escribes Append con mayúscula inicial?

Haz diferentes pruebas y toma nota de tus hallazgos.

Ahora accede a cada elemento de la lista.

¿Cómo lo harías?

En *Python*, además de acceder a los elementos de una lista utilizando índices positivos (empezando desde 0), también puedes usar índices negativos. Los índices negativos te permiten acceder a los elementos de una lista comenzando desde el final. Esto puede ser muy útil cuando quieres trabajar con los últimos elementos de una lista sin tener que conocer su longitud exacta.

- El índice -1 se refiere al último elemento de la lista.
- El índice -2 se refiere al penúltimo elemento.
- Y así sucesivamente.

Considera la siguiente lista de frutas:

```
frutas = ["manzana", "banana", "cereza",
"durazno","pera"]
```

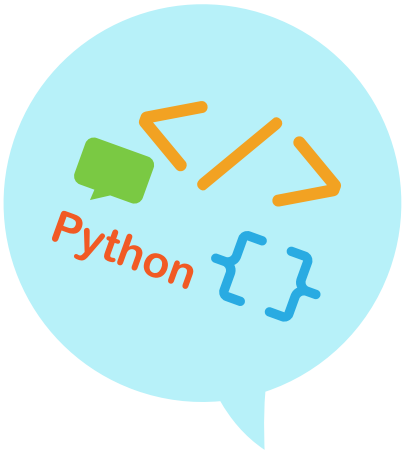
- El primer elemento (frutas[0]) es "manzana".
- El último elemento (frutas[-1]) es "pera".
- El penúltimo elemento (frutas[-2]) es "durazno".

Modifica tu código de la lista de compras para ver los dos últimos elementos que agregaste.

Puedes cambiar el valor de cualquier elemento de una lista accediendo a su índice y asignando un nuevo valor. Por ejemplo:

```
frutas = ["manzana", "banana", "cereza"]
print(frutas)
frutas[1] = "naranja"
print(frutas)
```

```
["manzana", "banana", "cereza"]
["manzana", "naranja", "cereza"]
```

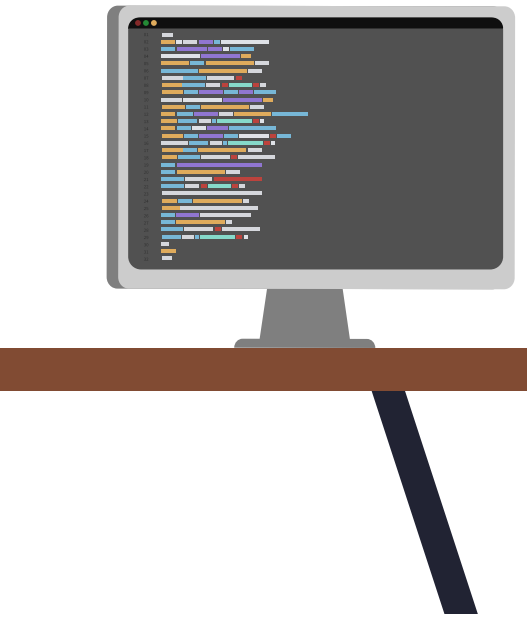


Modifica dos elementos de tu lista de compras e imprímela para ver los cambios.

Ahora, modifica tu código para insertar elementos en posiciones específicas utilizando el método **insert**. Añade las siguientes líneas a tu código y observa qué sucede:

```
compras.insert(2, "pan")
compras.insert(4, "panela")
```

¿Qué sucedió al insertar elementos en estas posiciones? ¿Cómo se reorganizó la lista?



Por último, añade la siguiente línea y descubre qué hace la instrucción **len()**:

```
print(len(compras))
```

Ahora pon en práctica lo aprendido, a partir de la siguiente situación:

Imagina que te han encargado diseñar un sistema de consulta de actividades extracurriculares de tu colegio. Este sistema permitirá a los y las estudiantes ver las actividades disponibles, consultar una actividad en particular y modificar la lista de actividades. Completa los siguientes pasos para crear este sistema:

Paso 1 - Crear el listado:

- Se requiere que el sistema contenga al menos 6 clubes o actividades de extensión disponibles. Crea una lista con las actividades extracurriculares que crees que podrían interesar a los/las estudiantes y luego imprímela.
- **Tarea:** Escribe el código para crear una lista con al menos 6 actividades extracurriculares y luego imprímela.

Paso 2 - Consultar actividades:

- Las y los estudiantes deben poder consultar las actividades disponibles de manera específica. Por ejemplo, pueden querer saber cuál es la tercera actividad del listado.
- **Tareas:** A. Escribe el código para consultar y mostrar toda la lista de actividades. B. Escribe el código para consultar y mostrar solo la tercera actividad del listado.

Paso 3 - Añadir nuevas actividades:

- Supón que llegan nuevas propuestas de actividades y se deben agregar al sistema. Añade 3 nuevas actividades al final de la lista.
- **Tarea:** Escribe el código para agregar 3 nuevas actividades al final de la lista y luego imprime el tamaño total de la lista.

Paso 4. - Consultar usando índices negativos:

- A veces, Las y los estudiantes pueden querer consultar las últimas actividades que se han agregado.
- **Tarea:** Escribe el código para consultar la penúltima actividad de la lista usando un índice negativo.

Paso 5 - Modificar actividades:

- Una de las actividades en la lista ha cambiado de nombre. Actualiza la lista reemplazando una de las actividades por el nuevo nombre.
- **Tarea:** Escribe el código para modificar una actividad en la lista e imprime la lista actualizada.

Cuanto termines, compara tu código con el de la persona que tienes al lado. Discutan las semejanzas y diferencias en la forma en que resolvieron los retos y prepárense para compartir sus conclusiones con el resto de la clase según las indicaciones de su docente.

Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1. ¿Puedes utilizar listas en un lenguaje de programación y realizar operaciones sencillas?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2. ¿Pudiste fortalecer las habilidades de depuración de errores de lógica y sintaxis en programas sencillos en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", vuelve a los contenidos de la sesión y crea un mapa mental con las operaciones con listas que practicaste hoy. Puedes incluir las operaciones con listas, su funcionalidad y ejemplos de código que te ayuden a recordar.

Anexo

Anexo 3.1

Función	Qué hace
lista.append(elemento) ejemplo: numeros.append(42)	
lista.insert(index, item) ejemplo:	
lista.pop(index) ejemplo:	
lista.remove(item) ejemplo: paises.remove('Uruguay')	
lista.index(item) ejemplo: pos = planetas.index("Marte")	
lista.count(item) ejemplo: num_pan = compras.count('pan')	
lista.reverse() ejemplo:	
lista.sort() ejemplo: nombres.sort() ejemplo: nombres.sort(reverse=True)	

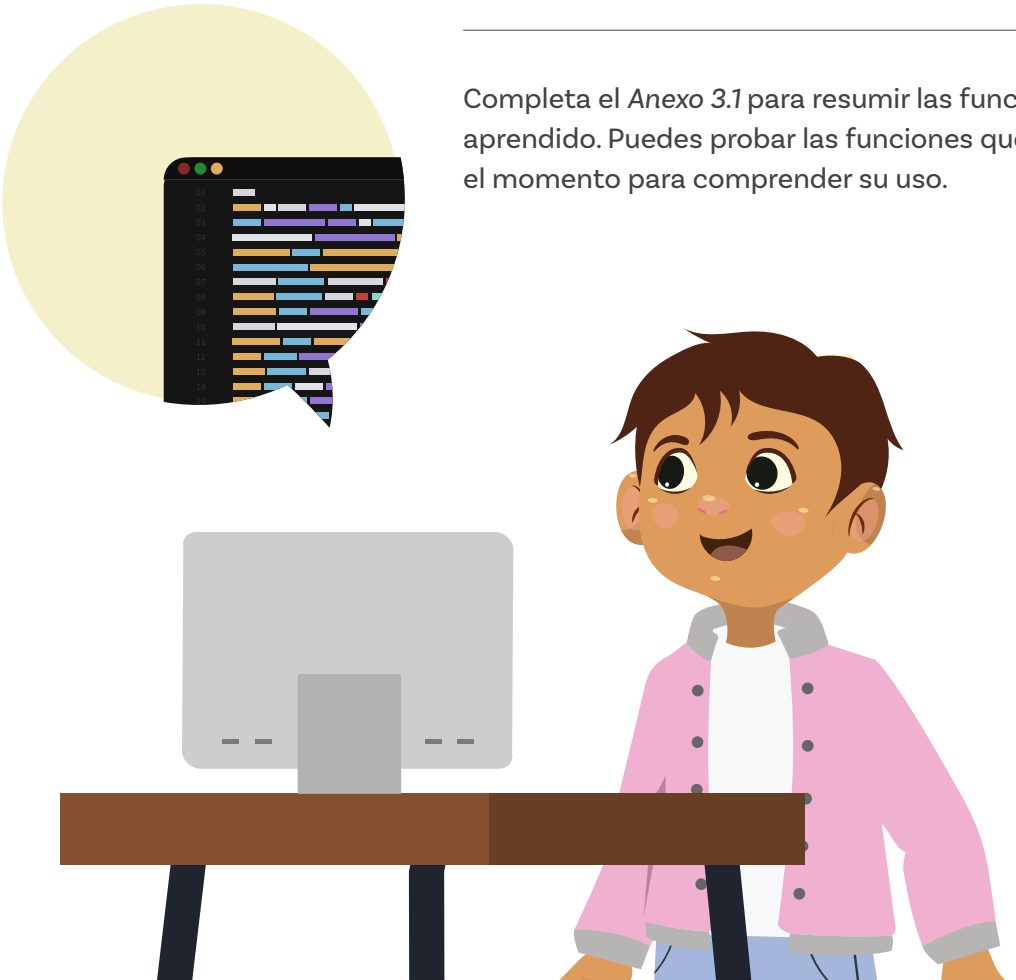
Ahora, te proponemos las siguientes preguntas de reflexión y discusión. Comenta tus respuestas con tu docente y el resto de la clase.

¿Recuerdas el ejemplo de la sesión anterior, en donde creaste nombres de historias?

¿Cómo podrías automatizarlo utilizando los conocimientos de esta sesión?

¿Puedes pensar en otros usos de las listas?

Completa el *Anexo 3.1* para resumir las funciones que has aprendido. Puedes probar las funciones que no hayas usado hasta el momento para comprender su uso.



TIC

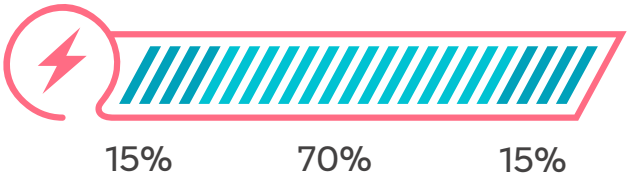
Sesión 4

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

- Utilizar listas en la creación de programas en *Python*.
- Evaluar la existencia de elementos dentro de una lista en *Python*.
- Crear soluciones interactivas que simulen procesos de autenticación en aplicaciones reales.

Duración sugerida



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

Como se ha dicho anteriormente, las listas y demás estructuras de datos son comunes en los programas que se hacen y utilizan a diario. Hasta el momento has trabajado con listas de menos de 20 elementos, por lo que es fácil inspeccionarlas y saber lo que contienen. Sin embargo, las bases de datos suelen tener cientos e incluso miles de registros, por lo que sería muy difícil encontrar un elemento de interés.

Piensa en tu lista de reproducción favorita, es probable que sea fácil encontrar una canción que te interesa escuchar.

¿? *¿Sería igual de fácil encontrarla entre miles de canciones?*

O, por ejemplo, imagina que la persona que administra la cafetería de tu colegio está preparando una lista de todo lo que debe comprar para el mes y ya no recuerda si anotó que necesitaba servilletas. Tal vez pueda leer rápidamente la lista y ver si las tenía anotadas o no. Pero un supermercado de cadena, que compra miles de cosas cada día, no podría buscarlo de la misma manera.

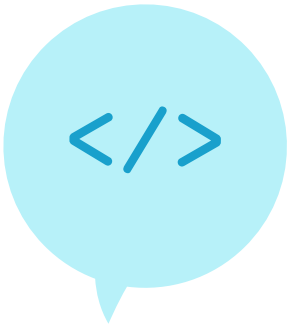
En *Python* es posible validar si un elemento se encuentra o no en una lista utilizando la función **in**. Esta función evalúa si el elemento está o no en la lista y retorna un valor booleano: verdadero (True) si el elemento se encuentra en la lista o falso (False) si no está.
Mira los siguientes ejemplos y escribe lo que se verá en pantalla al ejecutar el código.

```
dias_semana = ["lunes", "martes", "miércoles",
"jueves", "viernes"]

print("domingo" in dias_semana)
print("miércoles" in dias_semana)
print("jueves" in dias_semana)
```

¿?

¿Cómo modificarías el código para añadir el fin de semana a la lista?



Manos a la obra

Conectadas

Esta sección corresponde al 85% de avance de la sesión

Reúnete en un equipo de dos a tres personas según las indicaciones de tu docente y trabajen en el siguiente reto.

Vivimos en una era donde la información digital es extremadamente valiosa. Desde fotos personales hasta datos financieros, la mayoría de nuestra información está almacenada en computadoras y en la nube. Por esta razón, es esencial entender cómo proteger nuestros datos y por qué no todo el mundo puede acceder a todas las carpetas o información.

Los datos son importantes porque se utilizan para tomar decisiones, realizar investigaciones, gestionar operaciones comerciales y personales, y muchas otras actividades. Por eso, mantenerlos seguros es crucial.

No todos deben tener acceso a todos los datos. Aquí es donde entran en juego conceptos como la autenticación y la autorización.

- *Autenticación: es el proceso de verificar la identidad de una persona o dispositivo. Por ejemplo, cuando inicias sesión en tu cuenta de correo electrónico, estás autenticándote al ingresar tu nombre de usuario y contraseña.*

Tú y tu equipo son las personas encargadas de crear un programa que valide el ingreso de los usuarios a la base de datos de su colegio. Cuando un usuario necesite acceder a los datos, se le pedirá su usuario y contraseña. Si el usuario existe y la contraseña coincide, se mostrará un mensaje de bienvenida; de lo contrario, se mostrará un mensaje de rechazo.

Instrucciones:

Crear la lista de usuarios admitidos: Utilicen la *Tabla 1* para crear la lista de usuarios en *Python*:

Tabla 1. Lista de usuarios y contraseñas

Número de usuario	Usuario	Contraseña
1	Alicia	Al123
2	Bernardo	B3rn@rd0#
3	Carolina	PepeYMarco2
4	Diego	SanDiego!*
5	Elena	Contraseña
6	Francisco	Colegio2024
7	Gabriela	GabyLaMas
8	Hector	l@Voz
9	Isabel	IsaTKM
10	Joaquin	Joaco1990

Además, si el usuario no existe, podrá registrarse. Pero debe ingresar el código de administrador que es: ADMIN_co+ para que sea válido.

El equipo debe dedicar por lo menos 10 minutos a la planeación de la solución. Les sugerimos algunas preguntas que pueden ayudarles a descomponer el problema:

Nota

En la Guía 4 de grado 9 se explica al detalle el uso de la indentación y su importancia en las estructuras condicionales.

- ¿Cuántas listas van a crear? ¿Cuáles?
- ¿Cómo deben validar los usuarios?
- ¿Cómo van a interactuar con los usuarios?
- ¿Cómo pueden anexar a los nuevos usuarios?
- ¿Cómo pueden probar que su código funciona como es esperado?

Después de responder a las preguntas y planear su código, pueden pasar a *Python* a programar su solución. Recuerden leer con detenimiento los errores. En este caso van a utilizar condicionales por lo que deben prestar especial atención a los espacios de **indentación**.

El código que se muestra a continuación les puede servir de punto de partida para trabajar el menú.

```
print("Hola, elige una opción segun sea tu caso:")
print("1. Soy usuario registrado")
print("2. Soy usuario nuevo")

opcion = input("Ingresa el número de la opción deseada:")

if opcion == "1":
   print("Has elegido la opción 1")
   Usuario = input ("Ingresa tu nombre de usuario:")
   ## Continúa con el código

elif opcion == "2":
   print("Vamos a registrarte")
   codigo_admin = input("Ingresa el código de administrador: ")
       ## Continúa con el código
```

A medida que avanzan, pueden hacer pruebas a su programa para verificar que funciona permitiendo la consulta de datos de usuarios existentes o el registro de otros nuevos, siempre y cuando se ingrese el código correcto.

Luego, pueden trabajar con otro grupo y comparar las semejanzas y diferencias que hay en los códigos creados por ambos. Prepárense para compartir las conclusiones correspondientes cuando lo indique su docente.

Glosario

Indentación: la indentación en *Python* es el uso de espacios o tabulaciones al comienzo de una línea de código para definir la estructura del programa. Es crucial para indicar qué bloques de código pertenecen a una función, bucle, o condicional.

Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1. ¿Puedes utilizar listas en la creación de programas en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2. ¿Puedes evaluar la existencia de elementos dentro de una lista en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

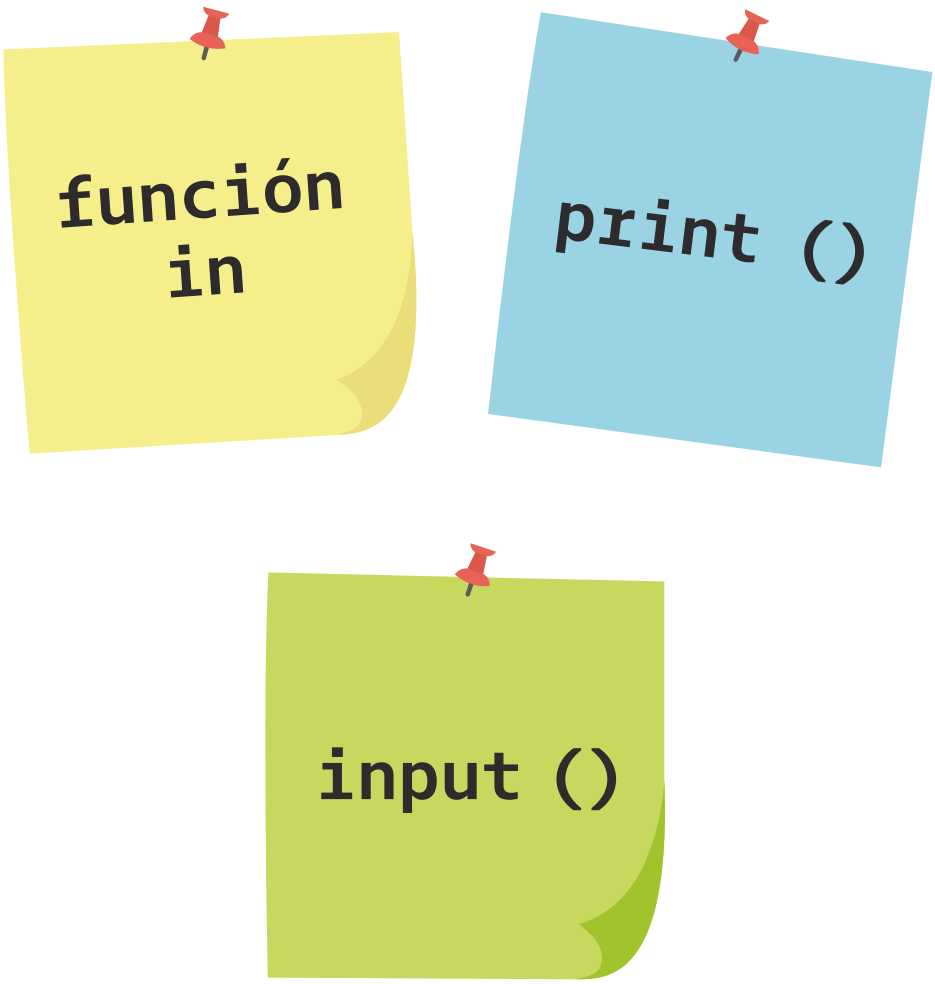
3. ¿Puedes crear soluciones interactivas que simulen procesos de autenticación en aplicaciones reales?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Puedes utilizar esta breve actividad de cierre, llamada Diálogo de colores, para fortalecer esos aprendizajes que aún están en construcción.

Para empezar, siguiendo las instrucciones de tu docente, toma 3 papeles de colores distintos. En el primer papel pon el concepto más interesante y/o útil de lo aprendido en esta sesión. En el segundo papel (de otro color) pon un desafío que enfrentaste durante la actividad y cómo lo resolviste. En el tercer papel anota cómo podrías utilizar este conocimiento a futuro.

Ahora, pega cada papel en un lugar diferente del salón, previamente determinado por tu docente, quien elegirá algunos de cada categoría para socializar.



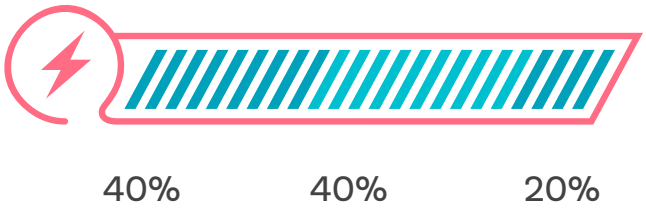
Sesión 5

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

- Iterar sobre listas y realizar comparaciones entre sus elementos.
- Interpretar los mensajes de error y usarlos para identificar y corregir los errores de lógica y sintaxis en los programas en *Python*.
- Diseñar programas que resuelvan tareas simples aplicadas a diferentes áreas de conocimiento.

Duración sugerida



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

Recuerda lo que has trabajado en las sesiones anteriores.

¿?

¿Puedes nombrar algunas funciones que has aprendido?
¿Cómo explicarías hasta el momento el concepto de lista?
¿Para qué sirve?

Para continuar aprendiendo sobre las listas, tu docente pedirá a cinco voluntarias(os) que salgan al frente. Cada persona deberá escribir en un papel pequeño una palabra secreta relacionada a una categoría. Por ejemplo, frutas. Cuando ya se tengan definidas las palabras, los cinco deberán hacer una fila sin permitir que nadie vea su palabra secreta.

Luego, otra persona será la ordenadora. Su misión será ordenar las palabras secretas en orden alfabético, con la condición de que únicamente podrá preguntar una letra a la vez.

Supongamos que la *Figura 1* muestra las palabras secretas de los primeros tres estudiantes.

Figura 1. Estudiantes y palabras secretas

Mango | Banano | Maracuyá

Ordenadora: ¿Por qué letra empieza tu palabra?
Persona[0]: Por la m.
Ordenadora: ¿Por qué letra empieza tu palabra?
Persona[1]: Por la b.

Así, la ordenadora deberá intentar ordenar todas las palabras alfabéticamente.

El resto del grupo debe estar atento a las estrategias que usa la persona. Mientras observas puedes pensar:

¿?

¿Notaste que comete algún error?
¿Crees que está usando alguna estrategia?
¿Cuántas veces le hace preguntas a la misma persona?
¿Cuánto tiempo tardaría si fueran 10 personas?

Pueden realizar varias rondas de la actividad o trabajar en pequeños grupos y luego compartir las reflexiones.

En esta actividad, tú y tus compañeras(os) practicaron la habilidad de **iterar sobre una lista**. Una de las prácticas de uso más comunes de las listas es iterar sobre ellas; es decir, recorrer cada elemento de la lista para realizar alguna acción con cada uno. Esta técnica permite manipular datos de manera repetitiva sin tener que escribir el código varias veces.

En la actividad, la persona encargada de ordenar tuvo que recorrer la fila varias veces, haciendo preguntas sobre los elementos, para lograr su objetivo.

Existen dos maneras principales de iterar sobre listas en *Python*. Compara los siguientes códigos y piensa: *¿qué crees que se verá en pantalla?*

```
lista_de_nombres = [ "Juan", "Nerea", "Camila", "Esteban" ]
for nombre in lista_de_nombres:
    print("hola", nombre )

lista_de_nombres = [ "Juan", "Nerea", "Camila", "Esteban" ]
for indice in range(len(lista_de_nombres)):
    print("hola", lista_de_nombres[indice])
```

¿En qué se diferencian los códigos? ¿Por qué usarías una forma de iterar y no la otra?

Escribe y ejecuta ambos códigos.

*¿Qué pasa si imprimes únicamente el índice? ¿Y si eliminas la función **range()**?*

Como podrás notar, ambas opciones para iterar sobre las listas son válidas. En programación, existen muchas maneras de lograr nuestros objetivos. En la siguiente actividad, podrás practicar esta habilidad.

Glosario

Iterar: repetir un proceso o conjunto de instrucciones varias veces, ya sea para alcanzar un resultado deseado, mejorar una solución o cumplir con una condición específica.

Manos a la obra

Conectadas

Esta sección corresponde al 80% de avance de la sesión

Trabaja en grupo con 2 o 3 compañeras o compañeros más, siguiendo las indicaciones de tu docente, para completar los siguientes retos. El primer reto se resolverá de manera guiada. Después, podrán elegir y resolver uno de los dos retos propuestos.

La tienda de mascotas de tu barrio quiere automatizar su inventario de productos y han solicitado tu ayuda. Actualmente, cuentan con 40 productos diferentes, con precios que varían entre 10.000 y 200.000. Tu tarea es clasificar los productos almacenando su información en dos listas diferentes: una para los que tienen precios menores o iguales a 50.000 y otra para los que superan este valor.

Ejemplo paso a paso: observen el siguiente código y predigan qué creen que hará cuando lo ejecuten. ¿Qué listas creen que se generarán y cuántos productos estarán en cada una?

```
import random

random.seed(10)
lista_precios = random.sample(range(10000, 200000), 40)
print(lista_precios)
print(len(lista_precios))
```

Ahora, ingresen el código en su editor de *Python* y ejecútenlo. Comprueben si sus predicciones eran correctas.

Después de ejecutar el código, reflexionen sobre lo siguiente:

¿Qué pasa si ejecutan varias veces el código? ¿Cambian los precios generados?
¿Qué pasa si modifican el número en ***random.seed()*** *y vuelven a ejecutar? ¿Cambian los precios generados?*
¿Qué hace la instrucción ***random.seed(10)****?*
¿Qué hace ***random.sample(range(10000, 200000), 40)****?*

1. El ejercicio menciona tres listas diferentes:

   - ○ Precios de los productos (generada con el código de arriba).
   - ○ Precios menores a 50.000.
   - ○ Precios mayores a 50.000.

2. Es necesario crear dos listas vacías.

3. Es necesario recorrer la lista de precios. Como nos interesan los valores podemos utilizar la forma directa.

```
import random
random.seed(10)
lista_precios = random.sample(range(10000,200000), 40)
print(lista_precios)
print(len(lista_precios))

mayores = []
menores = []

for precio in lista_precios:
    print(precio)
```

Observen el código y respondan:

> *¿Qué sucede en el ciclo for?*
> *¿Cómo podrían decidir a qué lista se debe agregar cada elemento?*

4. Se debe comparar el valor con 50.000. Si es menor o igual, se anexa a la lista menores, si es mayor, se anexa a la lista de mayores.

```
import random
random.seed(10)
lista_precios = random.sample(range(10000,200000), 40)
print(lista_precios)
print(len(lista_precios))
mayores = []
menores = []

for precio in lista_precios:
    if precio > 50000:
       mayores.append(precio)
```

5 Se deben imprimir las longitudes finales de las nuevas listas.

Completen el código y pruébalo. Inviten a otro grupo para revisar y probar su código.

6 Ahora proponemos dos situaciones para poner en práctica lo que han aprendido, estas habilidades les servirán en un futuro para resolver problemas reales. Elijan un reto y diseñen su solución. Si cuentan con el tiempo, completen ambos ejercicios.

Retos propuestos:

Reto 1: Imaginen que trabajan en una empresa de análisis de datos y necesitan clasificar a los 100 empleados(as) en dos grupos: aquellos con números de identificación pares y aquellos con números impares, para así asignarles los días que pueden utilizar el parqueadero. Creen dos listas nuevas e impriman su longitud.

La lista de 100 números de identificación se puede generar así:

```
import random

random.seed(10)
identificaciones = random.sample(range(1000, 20000), 100)
```

Reto 2: Imaginen que trabajan en una estación meteorológica y necesitan organizar las temperaturas diarias registradas en tres rangos para analizar el clima en la región: temperaturas bajo cero, temperaturas entre 0 y 30 grados y temperaturas superiores a 30 grados. La lista tiene 50 registros.

```
import random

random.seed(10)
temperatura = random.sample(range(-20, 40), 50)
```

A medida que trabajan, tomen nota de lo que se les dificulta y lo que les parece interesante. Prepárense para compartir y comparar sus soluciones con las de otros grupos.

Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1. ¿Puedes iterar sobre listas y realizar comparaciones entre sus elementos?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes interpretar los mensajes de error y usarlos para identificar y corregir los errores de lógica y sintaxis en los programas en *Python*?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes diseñar programas que resuelvan tareas simples aplicadas a diferentes áreas de conocimiento?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", revisa nuevamente los contenidos de esta y las demás sesiones. Identifica los contenidos o funciones que te cuestan trabajo y pide apoyo a tu docente.

Además, te proponemos las siguientes preguntas para que reflexiones sobre lo aprendido:

¿A qué errores te enfrentaste durante la solución de los retos?

¿Qué estrategias utilizaron tu y tus compañeras y compañeros para solucionarlos?

¿Qué aprendiste en estas sesiones?

¿En qué te gustaría profundizar?

Aprovecha este espacio final para hacer un esquema en el que resumas lo que aprendiste. Puedes empezar por escribir listas en *Python* en el centro de tu hoja o cuaderno y a partir de allí dibujar ramas que representen las conexiones con los conceptos claves aprendidos a lo largo de estas sesiones.

Anexo 1.1 Uso de *Python* en *Anaconda*

Escanea el siguiente QR para visualizar el anexo:

Enlace

Anexo 1.1 Uso de Python en línea

Escanea el siguiente QR para visualizar el anexo:

Enlace

Anexo 1.1 Uso Google Colab

Escanea el siguiente QR para visualizar el anexo:

Enlace

Anexo 1.2 Retomando *Python*

1 Une la función con su uso

Yo usaría...		Cuando necesito...	
1	input	A	Realizar acciones repetidamente.
2	print	B	Comparar si dos valores son iguales.
3	while	C	Visualizar algo en la pantalla.
4	if	D	Comprobar una condición para seleccionar las acciones que se van a realizar.
5	=	E	Recibir instrucciones de las/los usuarios (con el teclado).
6	==	F	Asignar el valor a una variable

2 Lee el siguiente código. Ten en cuenta que puede haber errores en el programa y/o puede que no se comporte como se esperaba.

```
1  a = int(input())
2  b = int(input())

3  prm = a + b / 2
4  print(prm)

5  max = a
6  if b > max:
7    max = b
8    print(max)
```

Enumera las **variables** en el programa:

Identifica una **expresión aritmética** en el programa y cópiala aquí:

Identifica una expresión booleana (una **condición**) en el programa y cópiala aquí:

Escribe las líneas del programa que piden información al usuario:

Anexo 1.3 Programando en *Python* – A

Ahora vas a practicar tu programación en *Python* y deberás trabajar con alguien más. Tú recibiste el *Anexo A*, y tu compañero(a) debió recibir el *Anexo B*. La primera actividad la desarrollarán como grupo, luego se propone una actividad individual.

En grupo:

1. Se supone que este programa de *Python* calcula y muestra la edad del usuaria/o, dado su año de nacimiento.

```
1  print("¿En qué año naciste?")
2  nacimiento = input()
3  edad = 2025 – nacimiento
4  print("Tienes", edad, "años")
```

Si ejecutas el programa y escribes tu año de nacimiento cuando se solicite, aparecerá un mensaje de error en la línea 3. ¿Qué crees que lo causa?:

```
edad = 2024 – nacimiento
TypeError: unsupported operand type(s) for –: 'int' and 'str'
```

El error se debe a que input devuelve lo que el usuario ha escrito como una cadena de texto o un string. El valor de **nacimiento** es una cadena de texto, por lo que la expresión **2024 – nacimiento** no se puede evaluar.

Modifica la línea 2. Así es como la entrada del usuario se convierte en un valor entero:

```
2  nacimiento = int(input())
```

Ejecuta otra vez el programa.

Si te enfrentaste a un **mensaje** de error, estos son algunos de los errores comunes que puedes haber cometido.

- ☐ Falta uno o ambos paréntesis de la función int
- ☐ Falta uno o ambos paréntesis de la función input

Podrías mejorar el programa. Por ejemplo, este solo funciona si el año actual es el 2025. ¿Cómo podrías modificarlo para que funcione en otros años? ¿Podrías ejecutar el código para más de una persona?

Individual

2 Ahora lee la situación y planea tu respuesta. Cuando tengas una idea sobre cómo lo resolverías, debes dictarle a tu compañero(a) lo que esperas que vaya en cada línea. Él o ella debe escribir el código que tú le pidas, pero no debe leer la situación. Tu compañero(a) actuará únicamente como tus manos, podrá ayudarte a resolver los problemas, si lo necesitas, y pondrá a prueba tu código. Luego, intercambiarán roles y tú serás sus manos para resolver el ejercicio.

Tu profesora de ciencias te pide que hagas un programa que lea el peso del usuario en la Tierra y calcule cuánto pesará el usuario en la luna y en diferentes planetas (Mercurio y Júpiter, por ejemplo). Investigas un poco y descubres que la gravedad en la luna equivale al 16% de la gravedad en la Tierra, por lo que algo que pesa 100 kg en la tierra, pesa 16,6 kg en la luna.

En Mercurio, la gravedad es un 37,8% la gravedad de la tierra. Y en Júpiter equivale al 253%

Pista: el símbolo para multiplicar es el asterisco (*)

Usa este espacio para planear tu solución:

3 Díctale tu código a tu compañero(a).

4 Pídele que ponga a prueba tu código, utilizando los casos de prueba que tiene en su *Anexo B*.

5 Ahora, escribe el código de tu compañero(a). Recuerda que no debes intervenir, solo escribe lo que te dicte.

6 Usa la información de los siguientes casos para probar el código de tu compañero(a). Se espera que su código te solicite un número en metros.

Caso 1 (metros)	Caso 2 (metros)	Caso 3 (metros)
Ingresa: 1	Ingresa: 0	Ingresa: -5
Resultado esperado: 3.281	Resultado esperado: 0	Resultado esperado: -16.4042

Se espera que su código te solicite una cantidad en gramos

Caso 4 (gramos)	Caso 5 (gramos)	Caso 6 (gramos)
Ingresa: 100	Ingresa: -50	Ingresa: 5000
Resultado esperado: 3.5274	Resultado esperado: -1.7637	Resultado esperado: 176.370

7 Ahora díctale tu código a tu compañero(a).

En grupo:

8 Ahora respondan en conjunto:

A *¿El código pasó todos los casos de prueba? ¿Falló en alguno?*

B *¿Lograron identificar los errores? ¿Cómo?*

C *¿Lograron corregir los errores? ¿Cómo?*

Esperen la instrucción de su docente para compartir las respuestas.

Anexo 1.3 Programando en *Python* – B

Ahora vas a practicar tu programación en Python y deberás trabajar con alguien más. Tú recibiste el *Anexo B*, y tu compañero(a) debió recibir el *Anexo A*. La primera actividad la desarrollarán como grupo, luego se propone una actividad individual.

En grupo:

1. Se supone que este programa de *Python* calcula y muestra la edad del usuaria/o,, dado su año de nacimiento.

```
1  print("¿En qué año naciste?")
2  nacimiento = input()
3  edad = 2025 – nacimiento
4  print("Tienes", edad, "años")
```

Si ejecutas el programa y escribes tu año de nacimiento cuando se solicite, aparecerá un mensaje de error en la línea 3. ¿Qué crees que lo causa?:

```
edad = 2025 – nacimiento
TypeError: unsupported operand type(s) for –: 'int' and 'str'
```

El error se debe a que input devuelve lo que el usuario ha escrito como una cadena de texto o un string. El valor de **nacimiento** es una cadena de texto, por lo que la expresión **2025 – nacimiento** no se puede evaluar.

Modifica la línea 2. Así es como la entrada del usuario se convierte en un valor entero:

```
2  nacimiento = int(input())
```

Ejecuta otra vez el programa.

Si te enfrentaste a un **mensaje** de error, estos son algunos de los errores comunes que puedes haber cometido.

- ☐ Falta uno o ambos paréntesis de la función int
- ☐ Falta uno o ambos paréntesis de la función input

Podrías mejorar el programa. Por ejemplo, este solo funciona si el año actual es el 2025. ¿Cómo podrías modificarlo para que funcione en otros años? ¿Podrías ejecutar el código para más de una persona?

Individual

2 Ahora lee la situación y planea tu respuesta. Cuando tengas una idea sobre cómo lo resolverías, debes dictarle a tu compañero(a) lo que esperas que vaya en cada línea. Él o ella debe escribir el código que tú le pidas, pero no debe leer la situación. Tu compañero(a) actuará únicamente como tus manos, podrá ayudarte a resolver los problemas, si lo necesitas, y pondrá a prueba tu código. Luego, intercambiarán roles y tú serás sus manos para resolver el ejercicio.

La dueña de la ferretería de tu barrio quiere comprar unos materiales que le faltan en el almacén, pero tiene un problema: ella sabe cuánto necesita de cada elemento en unidades métricas (gramos y metros) pero su proveedor acepta los pedidos en el sistema de Estados Unidos (onzas y pies).

Tú puedes hacer un programa que convierta las unidades para que ella pueda hacer su pedido más fácilmente. Después de investigar descubres que 1 metro equivale a 3.281 pies, y que 1 gramo equivale a 0.035 onzas.

Pista: el símbolo para multiplicar es el asterisco (*)

Usa este espacio para planear tu solución:

3 Escribe el código de tu compañero(a) quien tiene el *Anexo A*. Recuerda que no debes intervenir, solo escribe lo que te dicte.

4 Usa la información de los siguientes casos para probar el código de tu compañero(a), quien tiene la hoja A. Se espera que su código te solicite tu peso en la tierra..

Caso 1	Caso 2	Caso 3
Ingresa: 100	Ingresa: 45	Ingresa: 0
Resultados esperados:	Resultados esperados:	Resultados esperados:
Luna: 16.6	Luna: 7.4	Luna: 0
Mercurio: 37.8	Mercurio: 17	Mercurio: 0
Júpiter: 253.3	Júpiter: 113.9	Júpiter: 0

5 Ahora tú, díctale tu código a tu compañero(a).

6 Pídele que ponga a prueba tu código, utilizando los casos de prueba que tiene en su *Anexo A*.

En grupo:

7 Ahora respondan en conjunto:

A *¿El código pasó todos los casos de prueba? ¿Falló en alguno?*

B *¿Lograron identificar los errores? ¿Cómo?*

C *¿Lograron corregir los errores? ¿Cómo?*

Esperen la instrucción de su docente para compartir las respuestas.

Anexo 3.1 Operaciones en las listas

Función	Qué hace
lista.append(elemento) ejemplo: numeros.append(42)	
lista.insert(index, item) ejemplo:	
lista.pop(index) ejemplo:	
lista.remove(item) ejemplo: paises.remove("Uruguay")	
lista.index(item) ejemplo: pos = planetas.index(“Marte”)	
lista.count(item) ejemplo: num_pan = compras.count("pan")	
lista.reverse() ejemplo:	
lista.sort() ejemplo: nombres.sort() ejemplo: nombres.sort(reverse=True)	

TIC

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}