



Laboratorio 1A - Programas Secuenciales en Python

Objetivos

Introducción

Conociendo el IDLE de Python:

Escribiendo tu primer programa:

Parte 1: Operadores y Precedencia

Actividad 1: Operadores Aritméticos

Actividad 2: Significado de los Operadores

Actividad 3: Precedencia de los Operadores

Actividad 4: Corrigiendo errores de precedencia

Parte 2: Variables

Actividad 1: Declaración de variables

Actividad 2: Operaciones con variables

Actividad 3: Conversión de Tipos de Datos

Actividad 4: Concepto de bibliotecas

Actividad 5: Usando bibliotecas

Parte 3: Desafío

Actividad 1: El Viaje Interplanetario 🚀

Actividad 2: El problema del oxígeno ☁️

Objetivos

1. Realizar cálculos aritméticos utilizando variables y operadores básicos.
-

Introducción

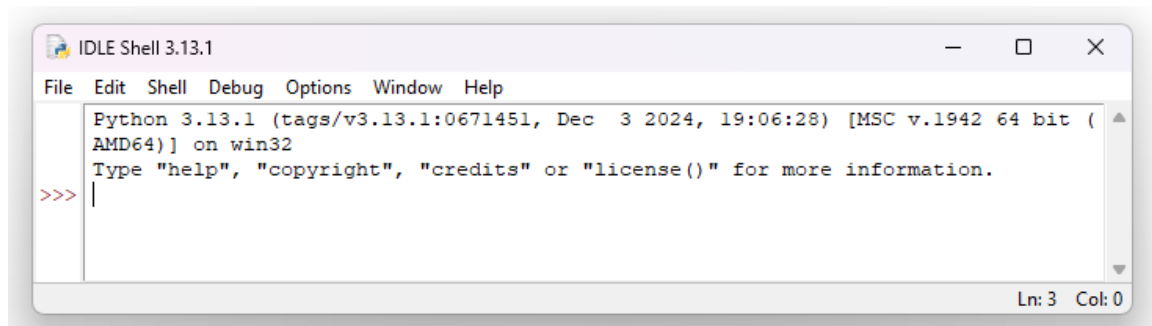
Estos Laboratorios son instancias prácticas diseñadas para que puedas aplicar los conceptos aprendidos en clase mediante la resolución de ejercicios y distintos tipos de desafíos.

Conociendo el IDLE de Python:

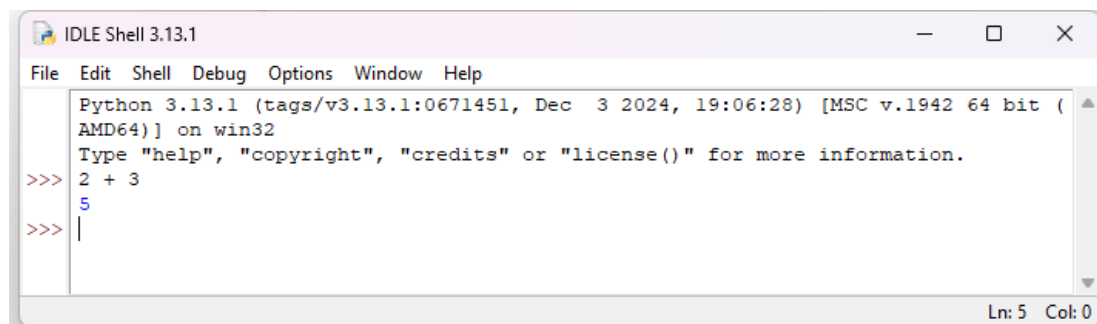
En este curso, utilizaremos **Python 3** como lenguaje de programación. Para ejecutar tus programas, utilizarás la **IDLE de Python**. La IDLE (abreviatura de *Integrated Development and Learning Environment*, en español: Entorno de

desarrollo y aprendizaje integrado) es una herramienta que permite escribir y ejecutar código de manera sencilla. Aquí están sus principales partes:

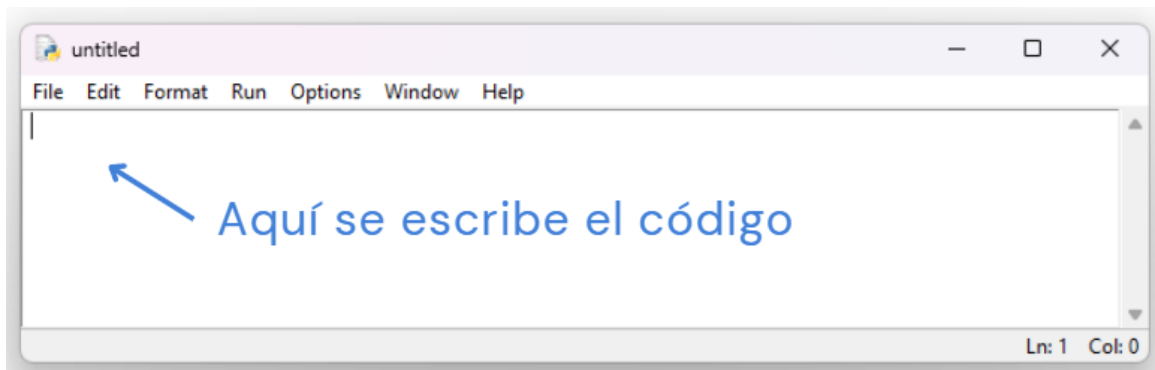
1. Shell de Python (o Consola)



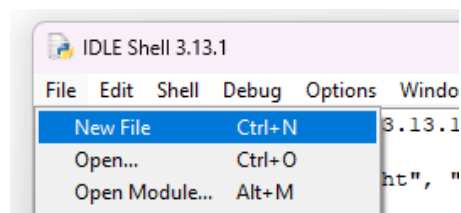
- Es la ventana principal que aparece al abrir IDLE.
- Permite escribir y ejecutar código Python línea por línea de manera interactiva.
- Se reconoce porque muestra el prompt `>>>`, indicando que está lista para recibir comandos.
- Es útil para probar pequeñas porciones de código rápidamente. Por ejemplo, podemos probar la suma de 2 números directamente, y nos mostrará el resultado:



2. Editor de Archivos (File Editor)

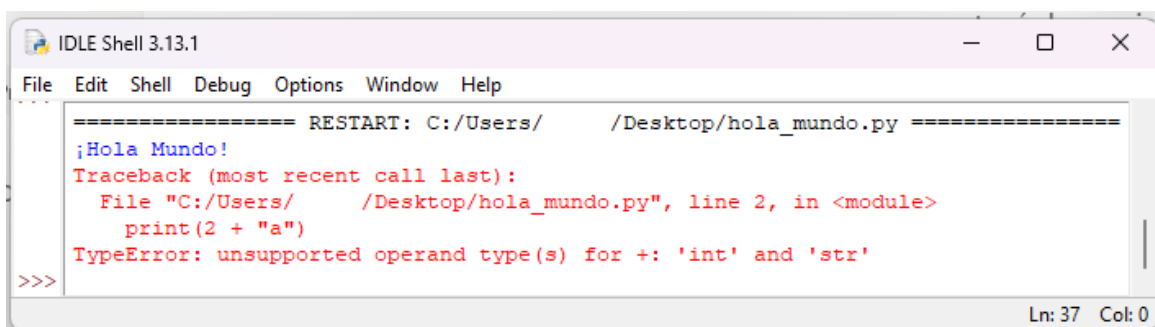


- Se usa para escribir y guardar programas en archivos con extensión `.py`.
- Para abrirlo, selecciona en el menú **File** → **New File**.



- Permite ejecutar el código guardado presionando **F5** o seleccionando **Run** → **Run Module**.
- Tiene características como resaltado de sintaxis y autocompletado.

3. Salida de Errores



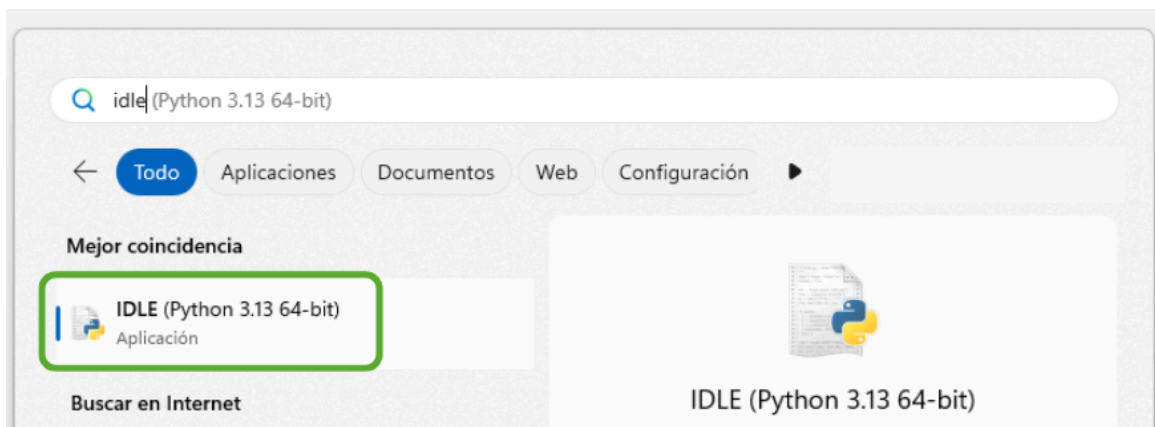
- Cuando hay errores en el código, al ejecutar el programa se muestran en la Consola.
- Aparecen mensajes con detalles del error y la línea donde ocurrió.
- Veremos un ejemplo de error más adelante.

Escribiendo tu primer programa:

Ya que conoces el entorno de desarrollo de Python, vamos a crear y ejecutar tu primer programa que mostrará el mensaje "¡Hola Mundo!" por consola. Sigue los siguientes pasos:

1. Abre IDLE de Python:

- Si estás en **Windows**, busca "IDLE" en el menú de inicio y ábrelo.
- Si estás en **Mac o Linux**, abre una terminal y escribe `idle3`, luego presiona **Enter**.



2. Crea un Nuevo Archivo en IDLE

Para escribir un programa en Python, debemos crear un archivo con extensión `.py`:

- En la ventana de IDLE, haz clic en **File > New File**.
- Se abrirá una ventana en blanco donde podrás escribir tu código.

3. Escribe el Código

En la nueva ventana, escribe el siguiente código

```
print("¡Hola Mundo!")
```

Explicación del código:

- `print()` es una función de Python que muestra un mensaje en la pantalla.
- `"¡Hola Mundo!"` es una cadena de texto (string) que será impresa.
- Las comillas (`" "`) indican que estamos trabajando con texto.

4. Guarda el Archivo

Antes de ejecutar el programa, debemos guardarlo:

- a. Haz clic en **File > Save As....**
- b. Elige una carpeta donde quieras guardar el archivo.
- c. Escribe un nombre para tu programa, por ejemplo: `hola_mundo.py`.
- d. Asegúrate de que la extensión del archivo sea **.py** y haz clic en **Save**.



Importante: siempre que programes recuerda **ir guardando tu archivo periódicamente**, para que no pierdas tu avance en caso de que haya algún problema con tu computador.

5. Ejecuta el Programa



Ejecutar un programa significa **hacer que el código escrito se procese y se convierta en acciones reales** en la computadora.

Ahora que el archivo está guardado, podemos ejecutarlo:

- a. En la ventana del editor de código, haz clic en **Run > Run Module** o presiona la tecla **F5**.
- b. Se abrirá la **Python Shell (o Consola)** y verás el siguiente resultado:

```
¡Hola Mundo!
```

¿Qué sucedió?

Python ejecutó la instrucción `print("¡Hola, Mundo!")` y mostró el texto en la pantalla. ¡Felicidades! Acabas de ejecutar tu primer programa en Python. 🎉

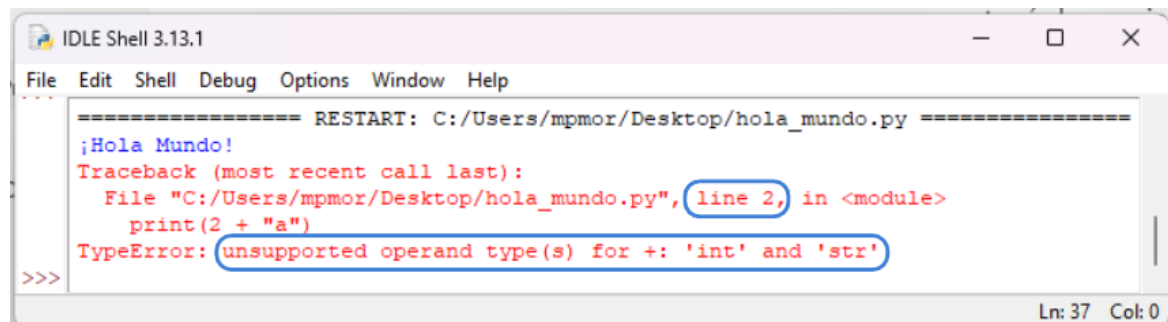
6. Hagamos un error:

Probemos cómo se vería en la consola si nuestro programa tuviese un error. Vamos a agregar una nueva línea en el programa que sabemos que va a fallar: imprimiremos la suma de un número con un string.

Para eso agrega la segunda línea a tu programa:

```
print("¡Hola Mundo!")  
print(2 + "a")
```

Y luego guarda y ejecuta nuevamente el código. En la consola o shell te aparecerá el siguiente error en rojo:



```
===== RESTART: C:/Users/mpmor/Desktop/hola_mundo.py =====
¡Hola Mundo!
Traceback (most recent call last):
  File "C:/Users/mpmor/Desktop/hola_mundo.py", line 2, in <module>
    print(2 + "a")
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Que indica que en la línea 2 hay un error por un tipo de operando no soportado para la operación de suma en Python, porque estamos intentando sumar un entero con un string.

Es muy importante ir viendo y analizando los errores que aparezcan en tu código, ya que te ayudarán a entender qué línea falla, y por qué, para que puedas ir arreglando el programa.



Si te fijas en la imagen del error, antes de mostrar el detalle del error sí se imprimió el mensaje `¡Hola Mundo!` en la consola, ¿por qué ocurrió esto?

Python ejecuta los programas **línea por línea**, de arriba hacia abajo.

```
print("Primera línea")
print("Segunda línea")
print("Tercera línea")
```

Al ejecutarlo, se imprimirá en la consola:

```
Primera línea
Segunda línea
Tercera línea
```

Cada `print()` se ejecuta en el orden en que aparece en el archivo.

¿Qué pasa si hay un error? Se empieza a ejecutar el programa línea por línea, y si hay un error en una línea, Python **detiene la ejecución y no sigue con las siguientes líneas**.

Con este primer paso, ya sabes cómo escribir, guardar y ejecutar un programa en Python. En los próximos ejercicios, exploraremos cómo usar variables y operadores para hacer programas más complejos.

Parte 1: Operadores y Precedencia

Cada laboratorio incluirá ejercicios de diferentes niveles, desde preguntas teóricas hasta desafíos de programación más complejos. El objetivo es que apliques los conceptos de manera práctica y estructurada.

Actividad 1: Operadores Aritméticos

1. Escribe en la consola y ejecuta las siguientes expresiones una por una, y observa los resultados:

- `10 // 3`
- `10 / 3`
- `10 % 3`
- `2 * 3`
- `2 ** 3`
- `5 + 3 - 1`

2. Responde:

- ¿Cuál es la diferencia entre `//`, `/` y `%`?
- ¿Cuál es la diferencia entre `*` y `**`?
- ¿Se puede utilizar más de un operador en una misma línea?

Actividad 2: Significado de los Operadores

Los operadores aritméticos en programación permiten realizar cálculos básicos con números. Su uso es esencial para resolver problemas matemáticos y realizar operaciones en cualquier programa.

Conecta cada operador aritmético con su definición correcta:

Operador	Definición
<code>+</code>	Suma entre dos números
<code>-</code>	Resta entre dos números
<code>*</code>	Multiplicación entre dos números
<code>/</code>	División estándar, conserva los decimales

//	División entera, descarta los decimales
%	Módulo, calcula el residuo de una división
**	Potencia, eleva un número a la potencia de otro

Actividad 3: Precedencia de los Operadores

Algunos operadores tienen mayor prioridad que otros al momento de ejecutarse en una misma línea.

1. Observa las siguientes expresiones y predice el resultado antes de ejecutarlas en Python. Escribe tus respuestas y luego compáralas con los resultados reales:

- a. `5 + 2 * 3`
- b. `(5 + 2) * 3`
- c. `20 // 3 + 2 * 4`
- d. `20 // (3 + 2) * 4`
- e. `2 + 3 - 5 + 6`
- f. `3 * 4 / 5 + 1`

2. Responde:

- a. ¿Qué sucede cuando usas paréntesis?
- b. ¿Cuál operación tiene mayor precedencia: multiplicación, potencia o suma?
- c. ¿Qué pasa si dos operadores tienen la misma precedencia?

Actividad 4: Corrigiendo errores de precedencia

Las reglas de precedencia determinan el orden en que se evalúan las operaciones dentro de una expresión en Python. Considerando que las operaciones se ejecutan de izquierda a derecha, las reglas de precedencia son:

- **Paréntesis:** `()`
- **Potencia:** `**`
- **Multiplicación, División, División entera, Módulo:** `*`, `/`, `//`, `%`
- **Suma y Resta:** `+`, `-`

Corrige las siguientes expresiones para que se comporten como se solicita.

▼ 👁️ Pista

Puedes usar paréntesis donde sea necesario.

1. Quiero que la suma ocurra antes que la multiplicación:

```
5 + 2 * 3
```

2. Quiero que la resta se haga antes de elevar al cuadrado:

```
10 - 3 ** 2
```

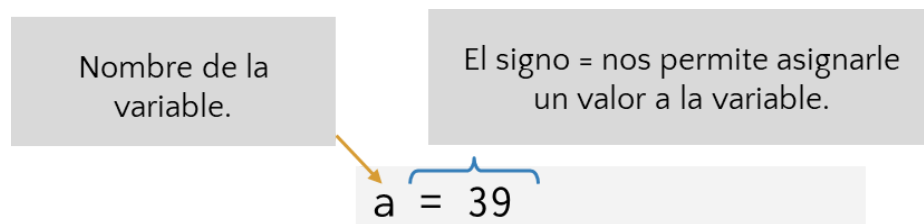
3. Quiero que primero se divida entre 3 y luego se sume todo antes de multiplicar:

```
20 // 3 + 2 * 4
```

Parte 2: Variables

Actividad 1: Declaración de variables

Las variables nos permiten almacenar valores (datos) de manera ordenada para posteriormente darles un uso.



1. Ejecuta el siguiente código y prueba diferentes valores para las variables `a` y `b`:

```
a = 10
b = 3
c = a / b
print("Resultado de la división:", c)
```

2. Responde:

- a. ¿Qué hace la función `print`?
- b. ¿Qué sucede si cambias el valor de `a` por un número flotante como `10.0`?



Podemos ponerle cualquier nombre a las variables, pero siempre se recomienda ponerles nombres descriptivos. Por ejemplo, si sabemos que una variable almacenará el nombre de una persona, es buena idea nombrar a la variable `nombre`.



Mala Práctica D:

```
h = input("Ingrese su nombre: ")
```

No es muy descriptivo.

Buena Práctica ♡



```
nombre = input("Ingrese su nombre: ")
```

Queda claro lo que representa la variable.

Poner buenos nombres a las variables ayudará a que entiendas mejor tu código

También se considera buena práctica ponerle nombres que no contengan caracteres especiales como tildes o "ñ".

Actividad 2: Operaciones con variables



Cuando usas la función `print()` con **comas** entre los valores, lo que haces es imprimir varios valores o expresiones de manera **separada** por un espacio por defecto. Cada valor que pongas dentro de `print()` será convertido en una cadena (si es necesario) y será impreso en la misma línea, con un espacio entre cada uno.

Crea dos variables: `a = 8` y `b = 4`.

Realiza las siguientes operaciones y guarda los resultados en nuevas variables:

- Suma de `a` y `b`
- Resta de `a` y `b`
- Multiplicación de `a` y `b`
- División de `a` entre `b`

Imprime los resultados de las operaciones en una misma línea.

Actividad 3: Conversión de Tipos de Datos

Hasta ahora hemos usado variables numéricas, pero pueden almacenar otros tipos de datos que utilizaremos más adelante en el curso.

int

(integer)

Números enteros

```
a = 1  
b = -42650
```

float

(punto flotante)

Números Reales

```
a = 1.0  
b = -3.75
```

bool

(booleano)

Valores lógicos

```
a = True  
b = False
```

str

(string) Texto

```
a = "Hola que tal"  
b = 'así también!'  
c = 'Ella dijo "Hola"'
```

Es posible convertir entre tipos de datos. Coloquialmente, a esta acción le llamamos *castear*. Veamos algunos ejemplos:

1. Ejecuta el siguiente código y analiza qué ocurre:

```
a = "15"  
print(int(a))  
  
b = 567.98  
print(str(b))  
  
c = "34.98"  
print(float(c))  
  
d = 456.6763  
print(int(d))
```

Responde:

- a. ¿Qué sucede cuando convertimos un *string* a un entero con `int(a)` ?
- b. ¿Por qué la conversión de `d = 456.6763` a un entero con `int(d)` pierde la parte decimal? ¿Está aproximando el número al entero más cercano, o lo está truncando?
- c. ¿Qué ocurriría si intentamos convertir una cadena con letras o caracteres especiales a un tipo numérico (por ejemplo, `int("abc")`)?

2. Analiza el siguiente código antes de ejecutarlo: ¿los valores de las variables `resultado1` y `resultado2` serán iguales o diferentes? ¿por qué?

```
numero1 = 14.56
numero2 = 6.9
resultado1 = int(numero1 * numero2)
print(resultado1)
resultado2 = int(numero1) * int(numero2)
print(resultado2)
```

Ejecuta el código y ve si lo que analizaste se cumplió o no.

3. Modifica el siguiente código para que el programa funcione y el resultado calculado en `total` se **trunque**:

```
precio_unitario = "15.5"
cantidad = "10"

total = precio_unitario * cantidad
print("Total de la compra:", total)
```

▼ 👁 Pista

Hay que castear el valor de las variables `precio_unitario`, `cantidad` y `total` a los tipos correspondientes.

Actividad 4: Concepto de bibliotecas

Una biblioteca es un conjunto de funciones predefinidas que puedes usar en tus programas para realizar tareas específicas, como operaciones matemáticas avanzadas. Para usar una biblioteca en Python, primero debes importarla con el comando `import`.



Importante: siempre debes importar la biblioteca antes de usarla. Como buena práctica, se suelen importar todas las bibliotecas al inicio del código (en las primeras líneas).

Prueba este ejemplo:

```
import math
```

```
raiz_cuadrada = math.sqrt(16)  
print("La raíz cuadrada de 16 es:", raiz_cuadrada)
```

Preguntas:

1. ¿Qué sucede si intentas usar `math.sqrt(16)` sin haber importado la biblioteca? Borra la primera línea y prueba.
2. ¿Qué otras funciones crees que podría incluir la biblioteca `math`?

Actividad 5: Usando bibliotecas



La biblioteca `math` posee varias funciones matemáticas interesantes como la función exponencial `math.exp(x)` y la función raíz cuadrada `math.sqrt(x)`. Sin embargo, posee muchas otras funciones matemáticas útiles que iremos aprendiendo a lo largo del curso.

```
import math
exponencial = math.exp(2)
raiz = math.sqrt(4)
```

También suele ser útil la biblioteca `random` que posee la función `randint(x, y)` que genera un número al azar entero entre el rango `x` e `y`:

```
import random
aleatorio = random.randint(1,10) #genera un entero aleatorio entre el 1 y el 10
```

Y hay algunas funciones matemáticas nativas de Python que se pueden utilizar sin importar ninguna biblioteca:

- Como la función valor absoluto `abs(x)` que entrega el valor absoluto del número `x` que se le entregue:

```
valor_absoluto = abs(-57) #valor_absoluto = 57
```

- Y la función de redondeo `round(x)` que redondea el número `x` al entero más cercano:

```
redondeo = round(-57.6) #redondeo = -58
```

Aunque también se le puede utilizar como `round(x, y)` para redondear el número `x` con una cantidad de cifras `y` de decimales específica:

```
redondeo = round(1.35678, 2) #redondeo = 1.36
```

Vamos a realizar un programa utilizando diferentes funciones matemáticas. Crea un programa que permita resolver las siguientes tareas:

1. Guarde un número aleatorio entre el 1 y el 5 en una variable.
2. Calcule e imprima la raíz cuadrada del número, redondeada al 3º decimal.

3. Calcule e imprima la potencia de ese número elevado a -6.
4. Calcule e imprima el valor exponencial de ese número aproximado al entero más cercano.
5. Calcule e imprima el valor exponencial de ese número truncado.

▼ 👁 Pista

Python tiene muchas funciones matemáticas listas para usar, pero debemos **importar las bibliotecas** antes de utilizarlas.

```
import math # Biblioteca de funciones matemáticas
import random # Biblioteca para generar números aleatorios
```

💡 **TIP:** Siempre importa las bibliotecas al comienzo del programa.

Parte 3: Desafío

Practicar con ejercicios con contexto hace que la programación sea más interesante y útil, ayudándote a conectar los conceptos con el mundo real y sus problemas. Muchas veces nos enfrentaremos a ejercicios que no necesariamente tienen el paso a paso de lo que hay que hacer, sino que te presentan el problema con lo que se requiere, y deberás encontrar la solución programando.

Partiremos practicando resolver problemas con contexto con pequeños desafíos, como el que viene a continuación.

Actividad 1: El Viaje Interplanetario 🚀

La Agencia Espacial enviará una misión a Marte con **cuatro astronautas** para investigar la presencia de hielo subterráneo y analizar el clima marciano. Para garantizar su seguridad, la tripulación debe calcular con precisión la duración del viaje.

Tu tarea es escribir un programa en Python que ayude a la misión a calcular el **tiempo total del viaje en horas**.

Para esto, el equipo de científicos te proporciona los siguientes datos:

- a. El tiempo de viaje depende de dos factores clave:
 - **La distancia hasta Marte** (en km) es de aproximadamente 225000000.
 - **La velocidad de la nave** (en km/h) es de aproximadamente 50000.

La distancia total del viaje será **ida y vuelta**, por lo que la distancia total se calcula así:

$$\text{Distancia total} = 2 \times \text{Distancia a Marte}$$

La fórmula para calcular el tiempo total en horas es:

$$\text{Tiempo total} = \frac{\text{Distancia total}}{\text{Velocidad de la nave}}$$

El equipo de científicos necesita el **resultado redondeado con 1 decimal**.

Un ejemplo de ejecución del programa sería:

Tiempo total de viaje: 9000.0 horas

▼ 👁 Pista

Hay ocasiones en las que deberemos programar fórmulas que se ven un poco complejas y puede que no entendamos del todo las matemáticas que hay detrás.

Sin embargo, lo que debemos hacer es **analizar cómo pasar la fórmula que está escrita en términos matemáticos a las operaciones que conocemos en Python**.

Para hacer el trabajo más fácil, recuerda usar variables para almacenar los datos iniciales, y los resultados de los cálculos. Y usar paréntesis si las operaciones son muy complejas, para asegurar que los cálculos se realizan en el orden correcto.

Actividad 2: El problema del oxígeno 🧠

Ahora que los científicos ya conocen el tiempo total que viajarán los astronautas en su viaje interplanetario, necesitan asegurarse de que tengan el oxígeno suficiente para sobrevivir todo el trayecto.

Para eso necesitan que modifiques tu programa para que además calcule:

1. **El consumo total de oxígeno necesario.**
2. **La cantidad de tanques de oxígeno requeridos.**

Para esto, el equipo de científicos te proporciona la siguiente información extra:

- a. Cada astronauta consume **0.84 kg de oxígeno al día**. Como queremos calcular el oxígeno por **hora**, dividimos este valor entre 24:

$$\text{Oxígeno por hora} = \frac{0.84 \text{ kg}}{24} = 0.035 \text{ kg/h}$$

Por lo tanto, el

oxígeno total necesario para los cuatro astronautas durante el viaje es:

$$\text{Oxígeno total} = \text{Horas de viaje} \times 0.035 \times 4$$

El equipo de científicos necesita el **resultado redondeado con 1 decimal**.

- b. Finalmente, debemos calcular cuántos tanques de oxígeno serán necesarios, asumiendo que **cada tanque tiene una capacidad de 500 kg**:

$$\text{Tanques necesarios} = \frac{\text{Oxígeno total}}{500}$$



Importante: una nave no puede llevar fracciones de tanques de oxígeno, por lo que el **resultado debe aproximarse hacia arriba**. Para esto podemos utilizar la función `math.ceil(x)` que **redondea un número hacia arriba** al entero más cercano.



Ejemplo:

```
import math

print(math.ceil(2.3)) # → 3
print(math.ceil(7.8)) # → 8
print(math.ceil(5.0)) # → 5 (no cambia si ya es un entero)
```



La función `math.ceil(x)` es útil cuando necesitas asegurarte de que un valor **no se quede corto**, como en el caso de los tanques de oxígeno: aunque necesitemos **2.25 tanques**, no podemos llevar un cuarto de tanque, y si llevamos 2 no alcanzará el oxígeno, así que **redondeamos a 3**.

Un ejemplo de ejecución del programa sería:

Tiempo total de viaje: 9000.0 horas

Consumo total de oxígeno: 1260.0 kg

Cantidad de tanques de oxígeno necesarios: 3