«Talento Tech»

Iniciación a la

Programación con Python

Clase 02

Buenos Aires Ciudad

Clase N° 2 | Variables y E/S de datos

Temario:

- Variables y tipos de datos simples.
- Operador de asignación.
- Operadores aritméticos.
- Funciones print() e input().
- Conversión entre tipos de datos simples.
- Programas con entrada, procesamiento y salida de datos.

Objetivos de la clase

El objetivo principal de esta clase es introducir al grupo de estudiantes en el manejo de variables, una herramienta fundamental para almacenar y gestionar información dentro de un programa. Aprenderán a utilizar los tipos de datos básicos y el operador de asignación para trabajar con información dinámica de manera eficiente.

Además, se presentarán las funciones print() e input(), esenciales para la interacción entre el programa y quien lo utilice. A través de estas herramientas, comprenderán cómo implementar el modelo Entrada-Proceso-Salida para crear programas simples y funcionales.

Finalmente, se explorará la conversión de tipos de datos, destacando su importancia para asegurar que los programas puedan procesar la información correctamente. Estos conceptos sentarán las bases para construir aplicaciones más complejas en las próximas clases, aplicando lógica y estructura al desarrollo de software.

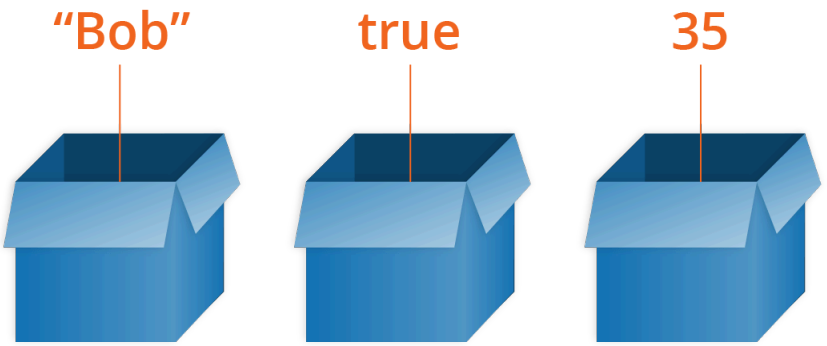
Cambia, todo cambia

¡Recibiste un correo de **TechLab!** Es un mensaje de **Mariana**, la Gerente de Proyectos, que te recuerda que como parte del equipo necesitás aprender a organizar y gestionar información con precisión. Por eso hoy habrá un proyecto orientado específicamente al uso de las **variables.**

Luis, que es desarrollador senior, te explica que una de tus primeras tareas es registrar los datos de los y las clientes de manera ordenada. Para eso, necesitas entender cómo almacenar información (utilizando variables) y cómo mostrarla por pantalla.

Leé con atención lo que sigue…

Variables en Python



Variables en Python

El concepto de variable es uno de los pilares de la programación y es común a todos los lenguajes por lo que entenderlo será fundamental para todas las personas que deseen iniciarse en el desarrollo de software. Las mismas son centrales ya que nos permiten que los programas sean dinámicos y flexibles. ¡Comencemos!

¿Qué es una variable?

En programación, una variable es como una "caja" donde podremos ingresar datos de distinto tipo y origen. Es decir, se tratará de un contenedor en la memoria de tu computadora donde almacenaremos datos temporales que tu programa pueda usar y modificar mientras se ejecuta. Una variable es en resumidas cuentas un contenedor que tiene una etiqueta con un nombre único; esta etiqueta te permite acceder al contenido de la caja o cambiarlo.

Por ejemplo, si yo estoy desarrollando la programación de un videojuego crearé o *“declararé”* (en términos técnicos) la variable **“puntos”**, donde mi programa guardará los puntos que cada equipo vaya acumulando. Esta misma variable (que guardará un tipo de dato numérico) me servirá para, por ejemplo, finalizar el juego. Imaginemos que nuestro videojuego es de volleyball, el pseudocódigo para ganar el set podría ser algo como: *“si puntos es igual a 25, entonces finalizar juego…”*

¿Cómo funcionan las variables en Python?

En Python no necesitás declarar explícitamente el tipo de dato que una variable va a almacenar (como números, texto, listas, etc.). Esto hace que trabajar con variables sea muy fácil.

Veamos un ejemplo:

```
mensaje = "Hola, Mundo"
```

En este ejemplo, **“mensaje”** es una variable que contiene el texto *"Hola, Mundo"*. El **operador de asignación** en Python es el símbolo =: se utiliza para *asignar un valor a una variable.* La forma en que funciona es bastante sencilla pero fundamental para casi cualquier programa. La estructura básica de una asignación es:

```
variable = valor
```

Donde variable es el nombre de la variable que estás creando o actualizando, y valor es el dato que deseas almacenar en dicha variable.

En el contexto de la programación, el significado del signo “=” (operador de asignación) es diferente al significado que tiene en, por ejemplo, la matemática (igualdad).

Formas de asignación.

Asignación única: Cada vez que usás el operador de asignación podés establecer o cambiar el valor de una variable. Si la variable ya tiene un valor determinado, el mismo será reemplazado por el nuevo valor asignado.

En el primer ejemplo, el número **30** se asigna a la variable **edad.** En el segundo ejemplo, la cadena de caracteres **"Juan"** se asigna a la variable **nombre:**

```
edad = 30
nombre = "Juan"
```

Asignaciones múltiples: Python también permite la asignación múltiple, lo que te permite asignar valores a varias variables en una sola línea de código. Veamos cómo se vería el código de las asignaciones múltiples de varias variables:

```
x, y, z = 1, 2, 3
```

Aquí, **x** recibe el valor **1**, **y** el valor **2**, y **z** el valor **3**.

Cambiando el valor de una variable

También podés modificar el valor actual de una variable. ¿Cómo? En el ejemplo siguiente, se establece el valor de la variable contador a “0”. En la siguiente línea de código **contador** recibe un nuevo valor, **1**, por lo que el valor final de la variable será **1**. Esto es útil para actualizar el contenido de una variable cuando sea necesario:

```
contador = 0
contador = 1 # Actualiza el valor de contador a 1
```

El operador de asignación es crucial porque permite:

- **Almacenar valores.** Permite guardar valores para usarlos más tarde en tu programa.
- **Actualizar valores.** Da lugar a cambios en los valores de las variables a medida que tu programa se ejecuta, lo que es esencial para la lógica de programación dinámica y los cálculos.

Además en Python, una variable puede comenzar almacenando un tipo de dato, como un número, y luego cambiar para almacenar un tipo de dato diferente, como una cadena de caracteres o una lista. Esto es posible gracias al **tipado dinámico** de Python. Por ejemplo:

```
dato = 100
dato = "Cien"
```

En este ejemplo, “dato” inicialmente almacena un valor numérico (100), pero luego cambia para almacenar una cadena de caracteres ("Cien").

El operador de asignación **=** es esencial, porque te permite la manipulación y gestión de datos dentro de un programa.

Nombres de las variables

Elegir nombres adecuados para las variables es super importante en la programación porque mejora la legibilidad del código y facilita su mantenimiento. Al nombrar variables en Python, hay varias reglas y buenas prácticas que te aconsejamos seguir para asegurarte de que tu código sea claro y fácil de entender.

1. **Caracteres permitidos:** Los nombres de las variables pueden incluir letras (mayúsculas y minúsculas), números y guiones bajos (_). Sin embargo, deben comenzar con una letra o un guión bajo. Por ejemplo, _miVariable o miVariable son válidos, pero 1miVariable no lo es.
2. **Palabras reservadas:** Python reserva un conjunto de palabras para su sintaxis y estructura de lenguaje, conocidas como palabras reservadas. Estas no pueden ser utilizadas como nombres de variables. Algunas de estas palabras incluyen *if, for, class, return, global,* entre otras. Intentar usar una palabra reservada como nombre de variable resultará en un error de sintaxis.
3. **Sensibilidad a mayúsculas y minúsculas:** Python diferencia entre mayúsculas y minúsculas. Esto significa que **variable**, **Variable** y **VARIABLE** serían consideradas tres variables distintas. Aunque técnicamente es posible tener variables con nombres tan similares, esto no es recomendable ya que puede prestarse a la confusión.

Buenas prácticas a la hora de nombrar tus variables:

Buenas prácticas a la hora de nombrar tus variables:

Elegir nombres adecuados para las variables es muy importante para escribir código que sea claro y fácil de mantener. Una de las prácticas más recomendadas es **usar nombres significativos y descriptivos**, ya que estos deben reflejar de manera clara los datos que contienen. Por ejemplo, optar por edad o numero_de_edad en lugar de simplemente ‘e’ o ‘n’ hace que el código sea más legible y comprensible, no solo para quien lo escribe, sino también para cualquier otra persona que trabaje con él en el futuro.

Cuando se trata de nombres largos, es común utilizar minúsculas y guiones bajos para separar las palabras. Esta convención (conocida como *snake_case)*, es la recomendada por la **guía de estilo de Python, PEP 8.** Nombres como nombre_usuario o contador_intentos son ejemplos de cómo esta práctica mejora la legibilidad del código.

Evita nombres cortos como “x” o “n", estos no proporcionan suficiente información sobre el propósito o el tipo de datos que la variable contiene, lo que puede dificultar la comprensión del código.

Además, en Python, el uso de un guión bajo (_) al principio del nombre de una variable, como en _privado, tiene un significado especial. Esto refiere al alcance de la variable (o método): **significa que ésta es privada y está destinada para uso interno dentro de una clase o módulo.** Aunque Python no aplica esta privacidad de manera estricta, es una convención ampliamente respetada entre las personas que desarrollan software.

Seguir estas reglas y buenas prácticas en la elección de nombres de variables no sólo contribuye a la claridad del código, sino que también facilita su mantenimiento y colaboración con colegas.

Tipos de datos simples en Python

Entender los tipos de datos es fundamental para manejar variables y realizar operaciones eficientemente. Los tipos de datos determinan **la clase de valor** que una variable puede almacenar, cómo se almacena ese valor en la memoria y las operaciones que son posibles realizar con esos valores. Vamos a profundizar en los tipos de datos básicos en Python y dejaremos para más adelante algunos tipos de datos más complejos.

1. Número entero (int)
Los enteros son números sin parte decimal, que pueden ser positivos o negativos. Python te permite trabajar con ellos para realizar operaciones matemáticas básicas como suma, resta, multiplicación y división, entre otras.

Ejemplos:

```
edad = 25
numero_de_estrellas = -100
años = 2023
```

Estos ejemplos muestran cómo asignar un número entero a una variable.

2. Número decimal (float)
Los números decimales o flotantes contienen una parte fraccionaria separada por un punto. Son útiles para representar valores que requieren mayor precisión, como medidas o cantidades financieras.
Ejemplos:

```
altura = 1.75
temperatura = -20.3
precio = 19.99
```

Cada ejemplo asigna un número decimal a una variable. Las operaciones con flotantes siguen las reglas de la aritmética de punto flotante.

3. Cadena de caracteres (str)
Las cadenas de caracteres son secuencias de caracteres usadas para almacenar datos textuales. Python las maneja con mucha flexibilidad, permitiendo operaciones como concatenación, multiplicación y acceso a elementos individuales.

Ejemplos:

```
nombre = "Juan"
mensaje = "Hola, Mundo"
letra = 'A'
```

Aquí, nombre, mensaje y letra son variables que almacenan cadenas de texto.

4. Booleano (bool)
Los booleanos representan dos valores: Verdadero (True) y Falso (False). Como comprobarás dentro de poco, son muy utilizados en estructuras de control.

Ejemplos:

```
es_mayor_de_edad = True
examen_aprobado = False
```

Los tipos de datos en Python son la base sobre la cual se construyen las estructuras de datos y se realizan operaciones. Es importante que los conozcas y recuerdes, ya que van a facilitar la manipulación de datos y la implementación de lógica compleja en los programas. A medida que avances, trabajaremos con tipos de datos más complejos (como las listas y diccionarios) y sus métodos asociados, lo que te permitirá construir programas más potentes y eficientes.

Luis te envía esta tabla, que resume lo visto sobre tipos de datos simples:

Tipo de dato	Abreviatura	Descripción
Número entero	**int**	Los enteros son números sin parte decimal, que pueden ser positivos o negativos.
Número con decimales	**float**	Contienen una parte fraccionaria separada por un punto. Equivalen a los números racionales.
Cadena de caracteres	**str**	Son secuencias de caracteres usadas para almacenar datos de texto, como nombres o descripciones.
Lógico o Booleano	**bool**	Sólo pueden almacenar uno de estos dos valores: Verdadero (True) y Falso (False).

Entrada / Salida de datos:

La función print()

Una función es simplemente un conjunto de instrucciones que Python ya conoce y que puede ejecutar cuando se lo pedís. Cuando usas print(), le estas diciendo a Python que ejecute la función print. Podés pensar en una función como si fuese una receta, una serie de pasos que te permiten lograr un resultado, como preparar un plato de comida. En Python, las funciones hacen eso mismo: siguen una serie de pasos para realizar una tarea específica.

Aunque aún no conocemos todos los detalles, la idea principal es que una función es algo que puedes usar para realizar tareas en tu programa de manera sencilla. Dentro de poco aprenderemos a crear nuestras propias funciones, pero por ahora, pensá en print() como un ejemplo de cómo una función puede hacer que los programas sean útiles y poderosos.

La función print() es utilizada para imprimir datos en la terminal. Acepta múltiples valores entre los paréntesis, los cuales pueden ser variables, literales (textos directos), o una combinación de ambos separados por comas. Podés delimitar las cadenas de texto con comillas simples (') o dobles ("), según prefieras. Por defecto, después de imprimir, print() se

añade un salto de línea, moviendo el cursor a la línea siguiente. Si se invoca print() sin pasarle **argumentos**, simplemente imprimirá una línea en blanco.

Veamos las distintas posibilidades y cuáles serían sus resultantes o “salidas”:

```
nombre = "Mundo"
print("Hola,", nombre) # “Hola” y nombre son dos argumentos.
# Salida: Hola, Mundo
```

La línea de código de color verde, al final del código anterior, es un **comentario.** En Python, los comentarios se crean utilizando el símbolo #. Cuando escribís “#” al comienzo de una línea o en cualquier parte de ella, todo lo que sigue a la derecha del símbolo se considerará un comentario. Python ignora cualquier texto que esté precedido por #, lo que significa que esos comentarios no afectarán en absoluto el funcionamiento de tu código. Los comentarios son útiles para dejar notas, explicar lo que hace una parte del código, o para desactivar temporalmente una línea de código sin eliminarla. Por ejemplo, si deseás explicar qué hace una sección específica de tu programa podés agregar un comentario justo encima de esa sección o al final de la línea de código correspondiente. Es importante usar comentarios de manera clara para que otras personas (o vos mismo en el futuro) puedan entender fácilmente lo que el código está haciendo y el propósito detrás de ciertas decisiones. En este caso, estamos usando los comentarios para indicar qué se mostrará en la terminal al ejecutar el ejemplo propuesto.

Insertando nueva línea y tabulaciones

Si una cadena contiene los caracteres \n (Nueva línea), cuando print() muestre su contenido en la pantalla insertará un salto de línea en ese punto:

```
print("Hola\nMundo")
# Salida:
# Hola
# Mundo
```

Y si la cadena contiene los caracteres \t (Tabulación), se añade una tabulación o un conjunto fijo de espacios en blanco, algo útil para alinear texto:

```
print("Nombre:\tJuan\nEdad:\t30")
# Salida:
# Nombre: Juan
# Edad:   30
```

Consideraciones al usar \t
La tabulación (\t) inserta un número fijo de espacios, que puede variar según el entorno donde se visualice el texto pero comúnmente equivale a 4 u 8 espacios. Su efectividad para alinear texto depende de la longitud de las cadenas que lo preceden o siguen. Si las cadenas son más largas o más cortas de lo esperado la alineación puede no ser la deseada.

```
print("Producto\tPrecio")
print("Manzanas\t$ 1.00")
print("Peras\t\t$ 1.50")
# Salida:
# Producto        Precio
# Manzanas        $ 1.00
# Peras           $ 1.50
```

Más adelante aprenderás otras maneras más efectivas y flexibles de dar formato a las cadenas de texto.

Controlando el final de la línea con end
Si a print() le pasas un valor para **end,** le podés especificar qué se debe imprimir al final de la cadena. Por defecto, **end='\n'**, lo que significa que cada llamada a print() termina con un salto de línea. Modificar este comportamiento te puede ser útil para imprimir algo más en la misma línea o añadir un delimitador específico.

```
print("Hola,", end=' ')
print("Mundo")
# Salida: Hola, Mundo
```

La función print() es una de las herramientas más básicas y poderosas en Python, y es esencial para la salida de nuestros datos. A través de **end** y la utilización de caracteres

especiales como **\n** y **\t**, podés tener un control detallado sobre cómo se formatea y presenta la información en la pantalla.

La función input()

Esta función es una herramienta interactiva que permite a los programas utilizar **datos introducidos por el la persona que usa el programa** a través del teclado. Cuando se ejecuta, **input()** pausa la ejecución del programa y espera que la persona escriba algo, devolviendo o "retornando" lo escrito como una cadena de caracteres *(string)* luego de que se presione "enter". Este mecanismo es indispensable para crear aplicaciones interactivas que requieren la participación de nuestra persona usuaria. **Input()** tiene los siguientes propósitos:

1. **Interacción:** recoger información de quien utiliza la aplicación, como su nombre, preferencias, comandos, entre otros.
2. **Control de flujo de programas:** basar la lógica del programa en las decisiones de quien interactúa con el mismo.
3. **Entrada de datos para procesamiento:** obtener números, cadenas de caracteres o cualquier dato que necesite ser procesado posteriormente para su utilización.

Sintaxis básica
La sintaxis básica de **input()** es:

```
variable = input(prompt)
```

donde **prompt** es un *string*, una cadena de caracteres, que se muestra en la consola indicando al usuario qué debe introducir. Recordamos que este último parámetro es opcional. En el siguiente ejemplo, el prompt dirá: "Introduce tu nombre: "

Solicitar un nombre:

```
nombre = input("Introduce tu nombre: ")
print("Hola", nombre)
```

Este código pedirá a quien interactúe con él que introduzca su nombre y luego le saludará personalmente. Si alguien introduce el nombre "Mariana", se imprimirá: "Hola, Mariana".

Pero no te olvides de algo fundamental: **input() siempre devuelve cadenas de caracteres**. Si necesitamos que la persona introduzca un tipo de dato diferente, como por ejemplo un número entero, necesitamos utilizar las funciones de conversión de tipos de datos.

Conversión entre tipos de datos

La conversión de tipos de datos, también conocida como *casting*, es una operación común que consiste en transformar un valor de un tipo de dato a otro, como de un entero (int) a un flotante (float), o de un número a una cadena de caracteres (str).

Conversión entre números enteros y números decimales

Podés convertir valores entre enteros y decimales para realizar operaciones aritméticas que requieren precisión decimal o simplemente para cambiar la representación de un número. Veamos cómo:

Ejemplo de int a float

```
entero = 10
decimal = float(entero)
print(decimal) # 10.0
```

En este caso **float(entero)** utiliza la función float para convertir el número entero 10 a un número decimal 10.0.

Ejemplo de float a int

```
decimal = 3.14
entero = int(decimal)
print(entero) # 3
```

En el ejemplo precedente observamos el mismo caso a la inversa: int(decimal) convierte el número decimal 3.14 a un número entero 3, descartando cualquier valor después del punto decimal.

Conversión de números a cadenas de caracteres y viceversa

La conversión de números a cadenas de caracteres es útil para concatenar números con texto, mientras que convertir cadenas de caracteres a números es necesario para realizar operaciones matemáticas con datos que originalmente se ingresaron con input() o se proporcionaron como texto.

Ejemplo de int a str:

```
edad = 25
mensaje = "Tengo " + str(edad) + " años."
print(mensaje) # Tengo 25 años.
```

Como podés ver, **str(edad)** convierte el número entero 25 a una cadena de caracteres "25", permitiendo su concatenación con otras cadenas de caracteres. Si bien pareciera decir lo mismo en ambos casos, recordá que lo importante es **cómo interpreta Python el valor 25**. Al comenzar edad es un número entero y al finalizar es una cadena de caracteres.

Ejemplo de str a int:

```
cadena_numerica = "123"
numero = int(cadena_numerica)
print(numero + 7) # 130
```

En este caso **int(cadena_numerica)** convierte la cadena de caracteres "*123*" a un número entero 123, permitiendo realizar operaciones matemáticas como la adición o suma.

Conversión entre booleanos y otros tipos de datos

Los booleanos también se pueden convertir a otros tipos de datos, y viceversa, dependiendo de las necesidades lógicas del programa. Las expresiones booleanas pueden ser traducidas a **1 (verdadero) y 0 (falso).**

Ejemplo de bool a int:

```python
valor_verdadero = True
valor_falso = False
print(int(valor_verdadero)) # 1
print(int(valor_falso))     # 0
```

Aquí, *True* se convierte a **1** y *False* a **0**, lo cual es muy útil en cálculos que dependen de condiciones lógicas.

Ejemplo de int a bool:

```python
numero = 0
print(bool(numero))     # False
numero = 5
print(bool(numero))     # True
```

Cualquier número distinto de 0 se convertirá a *True* en una conversión a booleano, **mientras que 0 se convierte a *False*.**

La conversión de tipos de datos es una herramienta poderosa en Python que permite manipular la información de manera más flexible. Luis, que sigue con atención tus avances, te alcanzar un resumen de las funciones que seguramente vas a usar todos los días:

Función	Tipos de datos aceptados	Descripción
int()	float, str (si representa un número), bool	Convierte un valor en un número entero. Si el valor es un número decimal, lo trunca (sin redondear).
float()	int, str (si representa un número), bool	Convierte un valor en un número decimal (float).

str()	int, float, bool	Convierte un valor en una cadena de texto (str).
bool()	Cualquier tipo de dato	Convierte un valor en un booleano (True o False).

La función type()

Esta función es una herramienta de desarrollo retorna el tipo de dato consultado. ¿Cómo funciona? Veámoslo con un ejemplo: Imaginate que necesitás saber cómo está interpretando Python un dato. Para resolverlo le solicitás que imprima el tipo de dato de la variable “numero”, es decir que nos devuelve su “clase” (<class>), asignando su nombre como *argumento*. Se imprimirá **“<class ‘int’>”,** detectando qué tipo de dato es el objeto “numero”: un entero. Esto incluye tipos de datos integrados como enteros, flotantes, cadenas de texto, listas, diccionarios, y más. Veamos algunos otros ejemplos para ilustrar cómo se utiliza **type()** verificando el tipo de datos básicos.

```
numero = 42
print(type(numero))          # <class 'int'>

precio = 19.99
print(type(precio))          # <class 'float'>

mensaje = "Hola, Mundo"
print(type(mensaje))         # <class 'str'>

esMayorDeEdad = True
print(type(esMayorDeEdad))   # <class 'bool'>
```

Cada llamada a **type()** te devuelve el tipo del valor asignado a la variable: **int** para enteros, **float** para flotantes, **str** para cadenas de texto y **bool** para booleanos.

Solicitar un número con input():

Retomando, vimos que que **input()** devuelve un *string*, así que si necesitas tratar ese valor como un número, primero tenés que convertirla al tipo de dato adecuado, ya sea *int* o *float*:

```
edad = input("Introduce tu edad: ")
edad = int(edad)
print("El próximo año tendrás" ,edad + 1, "años.")
```

¡Epa! ¿Qué hace ese “edad + 1” dentro del print()? En realidad, es bastante intuitivo: le suma 1 al valor que tenga la variable edad. Algo que sólo se puede hacer si “edad” es una variable de tipo numérico. Ese “+” es un operador aritmético.

Los operadores.

Se trata de carácteres que actúan sobre una, dos o más variables y/o literales para llevar a cabo una operación de alguna clase, devolviendo un resultado.

Operadores aritméticos

Dentro de los más comunes están los **operadores aritméticos**, que son los que utilizamos habitualmente para realizar cuentas con números en nuestra vida diaria. Python posee los mismos que encontramos en una calculadora electrónica y agrega algunos más. También podemos encontrar los operadores lógicos y los relacionales, que analizaremos más adelante.
Los operadores aritméticos son fundamentales para realizar cálculos en cualquier programa de Python, y permiten sumar, restar, multiplicar, dividir y mucho más. A continuación, analizaremos cada uno de ellos con ejemplos:

Suma (+): Suma dos valores.

```
resultado = 5 + 3  # resultado = 8
```

Resta (-): Resta el segundo valor del primero.

```
resultado = 5 - 3  # resultado = 2
```

Multiplicación (*): Multiplica dos valores.

```
```

```
resultado = 5 * 3  # resultado = 15
```

División (/): Divide el primer valor entre el segundo.

```
resultado = 6 / 3  # resultado = 2.0
```

Esta operación siempre retorna un número flotante *(float)*, incluso si el resultado es un entero.

División entera (//): Divide el primer valor entre el segundo y redondea el resultado hacia abajo para obtener un número entero *(int)*.

```
resultado = 7 // 3  # resultado = 2
```

Módulo (%): Retorna o devuelve el sobrante de la división entre el primer y segundo valor.

```
resultado = 7 % 3  # resultado = 1
```

Exponenciación (**): Eleva el primer valor a la potencia del segundo.

```
resultado = 2 ** 3  # resultado = 8
```

Ejercicio práctico.

Luego de haber aprendido qué son y cómo se utilizan las variables y cómo podés interactuar con una persona a través de la terminal, ya podés resolver la tarea que mencionaron Luis y Mariana. En su siguiente correo, ella te da detalles de lo que necesita que hagas:

¡Hola!

En nuestro día a día, interactuamos con muchos clientes, y uno de los pasos iniciales es recopilar y organizar su información básica. Para eso, necesito que desarrolles un pequeño programa en Python que haga lo siguiente:

- Solicite al cliente su nombre, apellido, edad y correo electrónico.
- Almacene estos datos en variables.
- Los muestre organizados en forma de una tarjeta de presentación en la pantalla.

¡Espero ver el resultado de tu trabajo pronto!

Saludos,
Mariana”

Materiales y recursos adicionales:

El Pythonista: Variables en Python
Oregoom: Tipos de datos en Python
Tutoriales sobre ciencia y tecnología: Variables, tipos de datos, operadores aritméticos e ingreso de datos en Python
Píldoras informáticas: Curso de Python |

Preguntas para reflexionar:

1. ¿Qué relación encontrás entre las variables y el modo en que organizamos información en nuestra vida diaria?
2. ¿Por qué es fundamental elegir nombres descriptivos para las variables y qué impacto tiene esto en la legibilidad del código?

Próximos pasos:

En la próxima clase, vamos a dar un salto importante en la construcción de programas interactivos y funcionales. Aprenderemos cómo usar **estructuras condicionales** como **if** y **else**, que nos permitirán tomar decisiones dentro de nuestros programas en función de los datos ingresados. Estas herramientas son esenciales para validar la información y hacer que nuestras aplicaciones sean más robustas y seguras.

Trabajaremos también con **operadores lógicos y relacionales**, lo que nos permitirá realizar comparaciones y establecer condiciones más complejas. Además, comenzaremos a incorporar **comentarios en el código**, una práctica fundamental para organizar y documentar nuestro trabajo.

Estos conceptos serán aplicados para mejorar el programa que creaste hoy, validando que los datos ingresados sean correctos y clasificando la información según las necesidades de los usuarios y usuarias. Este es un paso clave hacia la creación de programas más inteligentes y prácticos. ¡Nos vemos en la próxima clase!

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad