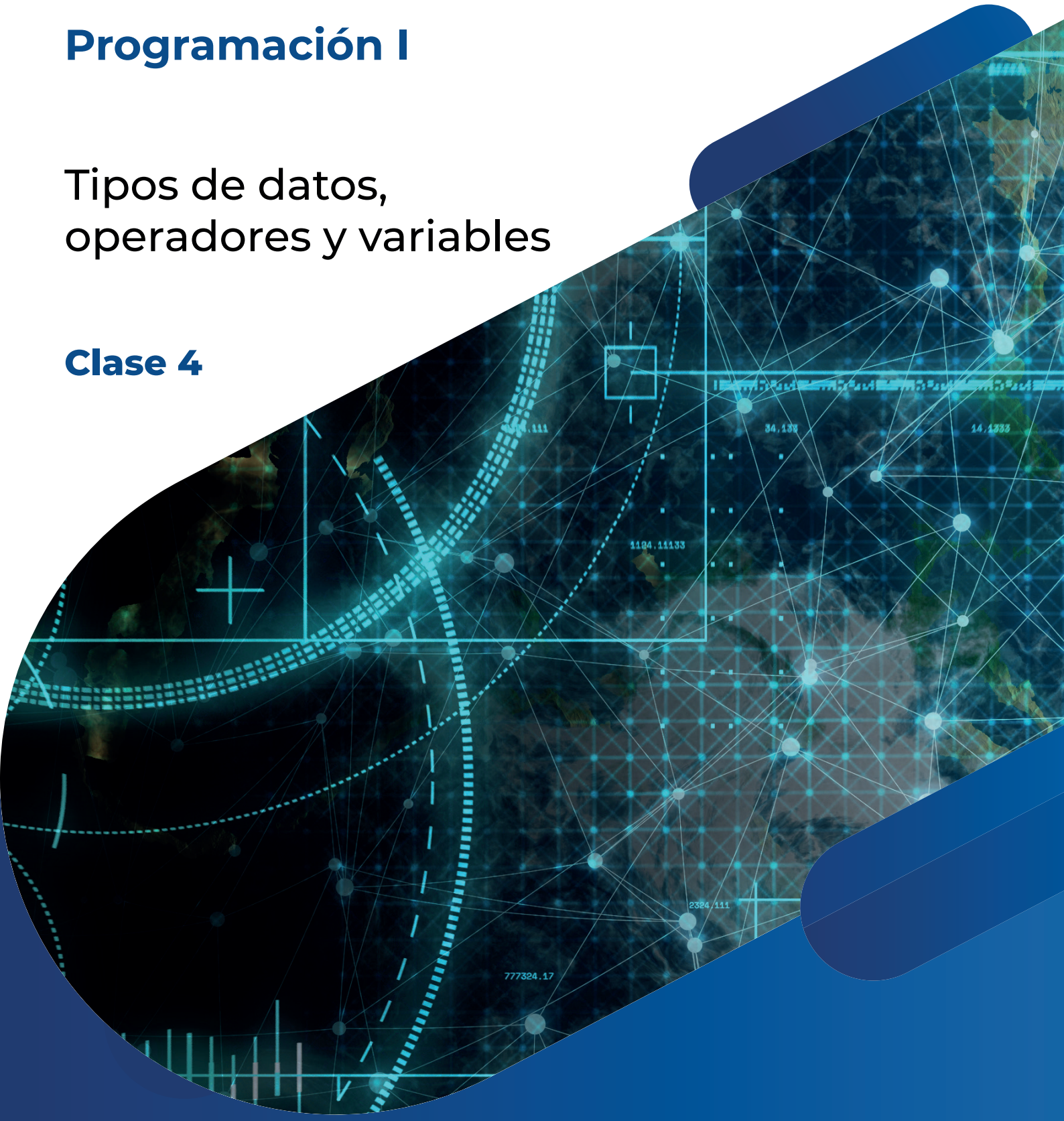
# Programación I

## Tipos de datos, operadores y variables

## Clase 4



**Ingeniería en ciberseguridad**

La excelencia no se improvisa

## 1. INTRODUCCIÓN DE LA CLASE

¡Bienvenidos a la Clase 4! Hoy continuaremos explorando los fundamentos de la programación en Python, centrándonos en los operadores aritméticos y lógicos que permiten transformar y evaluar datos de manera efectiva. Imagina que en la clase anterior aprendimos a manejar tipos de datos y declarar variables; ahora, veremos cómo aplicar estas herramientas para realizar cálculos, comparar valores y controlar la lógica de nuestros programas.

La manipulación de datos se vuelve realmente útil cuando sabemos cómo combinar valores y tomar decisiones con base en resultados de operaciones aritméticas y condiciones lógicas.

En este encuentro descubriremos cómo los operadores aritméticos (como la suma, la resta, la multiplicación, la división y el módulo) se integran con las estructuras de control para resolver problemas reales. Además, aprenderemos a utilizar los operadores lógicos (and, or, not) para evaluar múltiples condiciones de manera simultánea. Al finalizar la clase, comprenderemos que operadores y variables funcionan en conjunto para permitirnos diseñar algoritmos más potentes y flexibles. Dominar estos operadores y su uso correcto es un paso esencial hacia la programación avanzada.

**Clase 4: “Tipos de datos, operadores y variables (Parte 2)**

**Reto # 1**

**Contenido de la Clase:**

1. **Tipos de datos, operadores y variables (Parte 2)**

En esta sección continuamos explorando los elementos esenciales de la programación, enfocándonos particularmente en dos tipos específicos de operadores: los aritméticos y los lógicos. Estos operadores constituyen herramientas fundamentales que permiten realizar tareas indispensables, tales como procesar datos, efectuar cálculos complejos o sencillos, evaluar situaciones específicas y tomar decisiones sobre el flujo que seguirá un programa. A partir de los conceptos previamente abordados sobre cómo declarar variables adecuadamente y manejar distintos tipos de datos, ahora se analizará con mayor profundidad cómo estas estructuras se combinan para transformar y evaluar información, brindando la capacidad de resolver eficientemente problemas cotidianos o específicos que puedan surgir en diferentes contextos.

Los operadores aritméticos y lógicos actúan como los elementos básicos para manipular valores almacenados en variables, y permiten que un programa pueda tomar decisiones o ejecutar cálculos precisos. Por ejemplo, mediante los operadores aritméticos es posible implementar operaciones matemáticas como sumas, restas, multiplicaciones o divisiones, facilitando el desarrollo de programas que requieren de cómputos numéricos constantes, desde simples cálculos de promedios o totales hasta análisis estadísticos más sofisticados.

Por otro lado, los operadores lógicos son esenciales para establecer condiciones que determinan qué acciones deben ejecutarse en función de los valores almacenados en variables o resultados intermedios del programa. Estos operadores permiten evaluar múltiples criterios simultáneamente, lo que es particularmente útil en escenarios donde el software necesita reaccionar de forma dinámica ante diversas situaciones. Por ejemplo, en un sistema de registro de usuarios, un programa puede utilizar operadores lógicos para comprobar si un nombre de usuario y una contraseña cumplen simultáneamente con criterios específicos antes de permitir el acceso.

Asimismo, mediante ejemplos claros y concretos, se ilustrará cómo estos operadores interactúan directamente con las variables para generar resultados útiles que apoyan el objetivo final del software desarrollado. Estos ejemplos abarcarán situaciones cotidianas que van desde la validación sencilla de entradas proporcionadas por un usuario hasta operaciones matemáticas más avanzadas, que podrían ser requeridas en aplicaciones financieras, científicas o técnicas.

En esta sección profundizamos en los operadores aritméticos y lógicos, complementando el conocimiento adquirido sobre la declaración de variables y los tipos de datos. Veremos cómo estas herramientas permiten transformar y evaluar información para resolver problemas de diversa índole, desde cálculos matemáticos hasta el control del flujo en un programa.

### 1.1. Operadores aritméticos (+, -, *, /, %).

Los operadores aritméticos son elementos esenciales en la programación, cuya función principal es realizar cálculos matemáticos utilizando los valores que previamente han sido almacenados en las variables. Gracias a estos operadores, es posible efectuar desde operaciones aritméticas básicas hasta procesos numéricos avanzados, lo que brinda al programador la capacidad de desarrollar algoritmos flexibles y funcionales. Dichos algoritmos pueden ser aplicados en múltiples contextos profesionales, tales como el análisis financiero, la gestión administrativa, la investigación científica o incluso la ciberseguridad, especialmente al realizar cálculos relacionados con análisis estadísticos complejos o procesamiento masivo de datos.

Cada operador aritmético en Python cumple con un propósito específico y concreto.

- La suma (+), por ejemplo, permite la adición de dos valores numéricos, facilitando tareas fundamentales como llevar un registro acumulado de ventas, calcular totales en una factura o sumar diferentes puntuaciones en una evaluación académica.
- Por otro lado, la resta (-) proporciona un método sencillo para establecer diferencias entre dos números, resultando particularmente útil en contextos como el análisis financiero para determinar pérdidas y ganancias, en la gestión de inventarios para calcular diferencias en las cantidades almacenadas, o en procesos que impliquen medición de distancias o intervalos.
- Otro operador esencial es la multiplicación (*), que ofrece una manera eficaz de combinar cantidades a través de factores multiplicativos, siendo especialmente relevante en áreas donde los cálculos repetitivos son frecuentes, como en la determinación de costos totales cuando se compra cierta cantidad de un producto a un precio fijo, o en el manejo de datos tabulares mediante operaciones matriciales para estudios estadísticos o análisis matemáticos avanzados.
- Asimismo, la división (/) tiene una función crítica en situaciones en las que es necesario distribuir una cantidad en partes iguales o realizar promedios exactos. Cabe mencionar que en Python 3, esta operación siempre genera resultados flotantes (números decimales), aunque los operandos sean números enteros, lo cual ofrece una precisión significativa, pero requiere cuidado especial en ciertos cálculos financieros o científicos donde la exactitud decimal es crucial.
- Finalmente, el operador módulo (%) merece una mención especial, pues permite determinar el residuo resultante de una división entre dos números. Esta capacidad es particularmente útil cuando se necesita identificar patrones específicos, verificar si un número es divisible por otro (por ejemplo, determinar si es par o impar), o implementar ciclos y procesos repetitivos en programas, donde la lógica del algoritmo dependa de resultados periódicos o secuenciales.

Los operadores aritméticos en Python proporcionan las herramientas necesarias para manejar y transformar la información numérica de manera efectiva, facilitando así la creación de soluciones eficientes y adaptativas para resolver una amplia gama de problemas prácticos en distintos ámbitos profesionales.

Ejemplo en Python:

```
num1 = 10

num2 = 3



suma = num1 + num2        # 13

resta = num1 - num2       # 7

producto = num1 * num2    # 30

division = num1 / num2    # 3.333...

modulo = num1 % num2      # 1
```

En este ejemplo, cada operador se aplica a dos números enteros, pero Python también permite combinar tipos de datos (como int y float) y obtener resultados en punto flotante.

**Ventajas y Aplicaciones:**

- Manipulación directa de datos numéricos: Permite la realización de operaciones financieras, científicas o estadísticas de manera eficiente.
- Base de cálculos en un programa: Se combinan con estructuras de control (if, while, for) para resolver problemas complejos, como simulaciones o validaciones.
- Flexibilidad de tipos: Python maneja automáticamente las conversiones entre int y float al realizar operaciones, facilitando la escritura de código.

**Limitaciones:**

- Precisión en división con float: En contextos de alta exactitud (por ejemplo, cálculos monetarios), la división flotante puede introducir errores de redondeo, siendo necesario recurrir a librerías específicas.
- Módulo y divisor cero: El operador % requiere que el divisor sea distinto de cero para evitar errores en tiempo de ejecución.

Los operadores aritméticos permiten transformar datos numéricos y constituyen la base de la lógica computacional que involucra cálculos.

Tabla 1: Operadores Aritméticos en Python

| **Operador** | **Función** | **Ejemplo** | **Resultado** |
|---|---|---|---|
| + | Suma dos valores | 10 + 5 | 15 |
| - | Resta un valor de otro | 10 - 3 | 7 |
| * | Multiplica dos valores | 4 * 2 | 8 |
| / | Divide un valor por otro, devolviendo un float | 10 / 4 | 2.5 |
| % | Módulo: obtiene el resto de la división | 10 % 3 | 1 |

Damián Nicolalde Rodríguez. (2024).

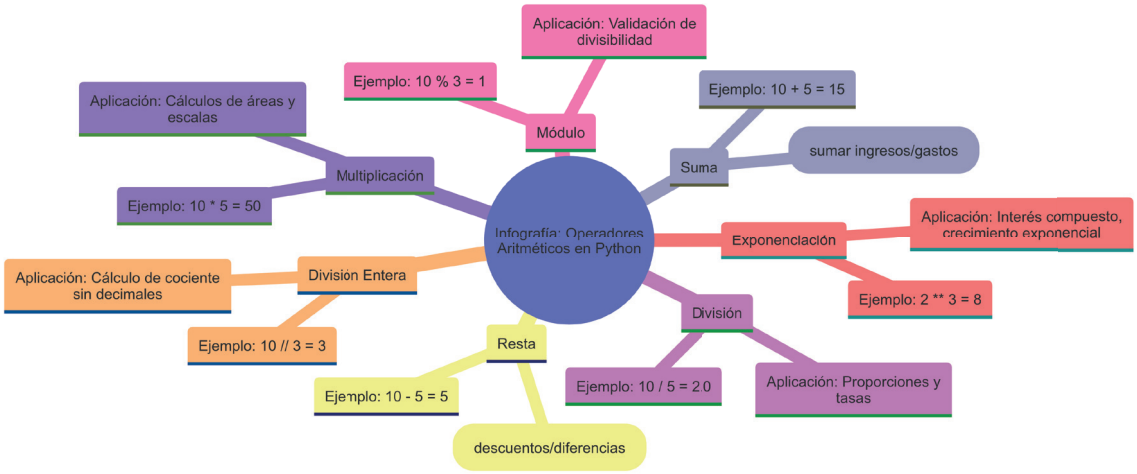



**Imagen 1: Infografía de Operadores Aritméticos**
Damián Nicolalde Rodríguez. (2024). Infografía de Operadores Aritméticos.

Para complementar y profundizar, se sugieren acceder a este recurso:

- Título del enlace relacionado: Guía Completa de Operaciones Aritméticas en Python
- **Descripción del enlace relacionado:** Este video es un tutorial detallado y práctico diseñado para dominar el uso de los operadores aritméticos en Python, esenciales para realizar cálculos matemáticos en programación.
- **Enlace: https://www.youtube.com/watch?v=ZLsHYeqVeXo**

Para ilustrar aún más el uso de los operadores aritméticos, se presenta la siguiente tabla que describe escenarios comunes donde se aplican estos operadores, destacando su relevancia en la solución de problemas reales.

Tabla 2: Casos de Uso y Ejemplos Detallados

| Caso de Uso | Operadores Involucrados | Descripción | Ejemplo de Código |
|---|---|---|---|
| **Cálculo de Promedios** | +, / | Se suman varios valores numéricos y se dividen entre la cantidad de elementos para obtener el promedio, útil en calificaciones, mediciones o análisis estadísticos. | notas = [8, 7, 9]<br><br>promedio = sum(notas) / len(notas) # 8.0 |
| **Control de Inventario** | -, +, % | Al registrar ventas o compras, se ajustan los niveles de stock (resta o suma) y, a veces, se utiliza el módulo para agrupar productos en lotes de tamaño fijo. | stock_inicial = 50<br><br>venta = 12<br><br>stock_final = stock_inicial - venta # 38<br><br>lotes = stock_final % 10 # Determina cuántos lotes sobran |

| | | | |
|---|---|---|---|
| **Cálculos Financieros** | *, /, -, % | Permiten calcular intereses, comisiones, descuentos, etc. Se combinan operaciones para obtener resultados precisos, por ejemplo, determinar cuánto se paga de impuestos. | precio_base = 100.0<br><br>impuesto = 12.5 # 12.5%<br><br>total = precio_base + (precio_base * impuesto / 100) # 112.5 |
| **Verificación de Par/Impar** | % | El operador módulo (% 2) se utiliza para determinar si un número es par (resultado 0) o impar (resultado 1), facilitando validaciones en bucles o estructuras condicionales. | numero = 13<br><br>if numero % 2 == 0:<br><br>print(“Par”)<br><br>else:<br><br>print(“Impar”) |
| **Conversión de Unidades** | +, -, *, / | En proyectos que involucran medidas (por ejemplo, metros a kilómetros), los operadores aritméticos transforman los valores de entrada para producir resultados en diferentes unidades. | metros = 1500<br><br>kilometros = metros / 1000 # 1.5 |

Damián Nicolalde Rodríguez. (2024). Casos de Uso y Ejemplos Detallados.

### 1.2. **Operadores lógicos (and, or, not).**

Los operadores lógicos permiten combinar expresiones booleanas para formar condiciones complejas y evaluar múltiples factores en la ejecución de un programa.

Los operadores lógicos son herramientas esenciales para combinar múltiples condiciones o criterios que se evalúan como verdaderos o falsos. Estos operadores son particularmente útiles porque permiten crear decisiones compuestas y complejas, facilitando la ejecución dinámica del código según distintas circunstancias. Con ellos, un programa puede evaluar diversas situaciones simultáneamente y determinar si estas cumplen o no ciertas reglas o requisitos específicos. Gracias a esta característica, se pueden implementar procesos inteligentes y precisos, fundamentales en diversas áreas como la seguridad informática, la validación de datos, la automatización industrial o el control de calidad.

- El operador lógico **and** es utilizado para combinar dos o más condiciones, devolviendo un valor True únicamente si todas las condiciones evaluadas se cumplen simultáneamente. Por ejemplo, en un sistema que gestiona accesos, podría utilizarse para verificar si un usuario es mayor de edad y cuenta con los permisos necesarios al mismo tiempo, permitiendo así un control más estricto y seguro sobre el acceso a ciertas funcionalidades o recursos sensibles. Este operador es especialmente importante en contextos donde la seguridad o la precisión dependen de múltiples factores simultáneos.
- Por otro lado, el operador lógico **or** proporciona mayor flexibilidad al permitir que el programa retorne True si al menos una de varias condiciones evaluadas es verdadera. Esto es sumamente útil en escenarios donde basta con que se cumpla una sola condición para proceder, como por ejemplo en el acceso alternativo a un sistema mediante varios métodos de autenticación diferentes, donde cumplir uno solo de los métodos sería suficiente para autorizar al usuario. La capacidad del operador **or** de evaluar múltiples caminos posibles aporta una gran adaptabilidad a las aplicaciones desarrolladas.
- Finalmente, el operador lógico **not** tiene una función complementaria y crítica, que es invertir el resultado de una expresión lógica. Este operador convierte una condición inicialmente verdadera en falsa, y viceversa. La capacidad del **not** para invertir condiciones es particularmente útil en contextos en los que resulta más sencillo o natural expresar una condición en negativo; por ejemplo, verificar

si una operación no se ha completado o si una contraseña ingresada no coincide con la almacenada, generando así una lógica más directa y comprensible en ciertas situaciones.

En conjunto, estos operadores permiten al desarrollador crear programas complejos que reaccionan adecuadamente ante una gran variedad de situaciones, fortaleciendo la capacidad de decisión del software y aumentando significativamente la eficiencia del código escrito. El dominio efectivo de los operadores lógicos garantiza no solo soluciones funcionales, sino que también mejora la legibilidad y la robustez del código, aspectos clave en cualquier desarrollo profesional.

Ejemplo en Python:

```
edad = 20

salario = 1200

es_estudiante = True

condicion_and = (edad > 18) and (salario > 1000)  # True

condicion_or = (edad > 25) or (es_estudiante)     # True

condicion_not = not (edad < 18)                   # True
```

En este ejemplo, se combina la comparación de la edad y el salario con el operador and, se evalúa si el usuario es mayor de 25 años o estudiante con or, y se invierte la condición de ser menor de edad con not.

**Ventajas y Aplicaciones:**

- Evaluación simultánea de condiciones: Los operadores lógicos permiten combinar múltiples factores, creando condiciones más ricas y adaptables que las evaluaciones individuales.
- Control del flujo de ejecución: Son esenciales en la implementación de estructuras condicionales (if, elif, else) y en bucles (while, for), ayudando a decidir qué se ejecuta según la combinación de condiciones.
- Validaciones de datos: Se aplican en formularios, sistemas de acceso y cualquier lógica que dependa de varios criterios (por ejemplo, permisos, rangos de valores, roles de usuario).

**Limitaciones:**

- Complejidad en expresiones anidadas: Expresiones con varios and, or y not pueden volverse confusas, requiriendo un entendimiento sólido de la lógica booleana para evitar resultados inesperados.
- Cortocircuito (short-circuit): En Python, and y or implementan una evaluación por cortocircuito, lo que significa que dejan de evaluar el resto de la expresión si el resultado ya se ha determinado. Esto es útil en muchos casos, pero puede causar sorpresas si no se comprende a fondo.

Los operadores lógicos permiten la evaluación simultánea de condiciones, siendo fundamentales para el control del flujo de ejecución en Python.

Tabla 3: Operadores Lógicos en Python

| Operador | Función | Ejemplo | Resultado |
|---|---|---|---|
| and | Retorna True si **ambas** condiciones son verdaderas. | (edad > 18) and (salario > 1000) | True, si ambas se cumplen |

| | | | |
|---|---|---|---|
| or | Retorna True si **al menos una** condición es verdadera. | (edad > 18) or (salario > 1000) | True, si una de ellas es True |
| not | Invierte el valor booleano de una expresión (True ↔ False). | not (edad < 18) | True, si edad < 18 es False |

Damián Nicolalde Rodríguez. (2024).

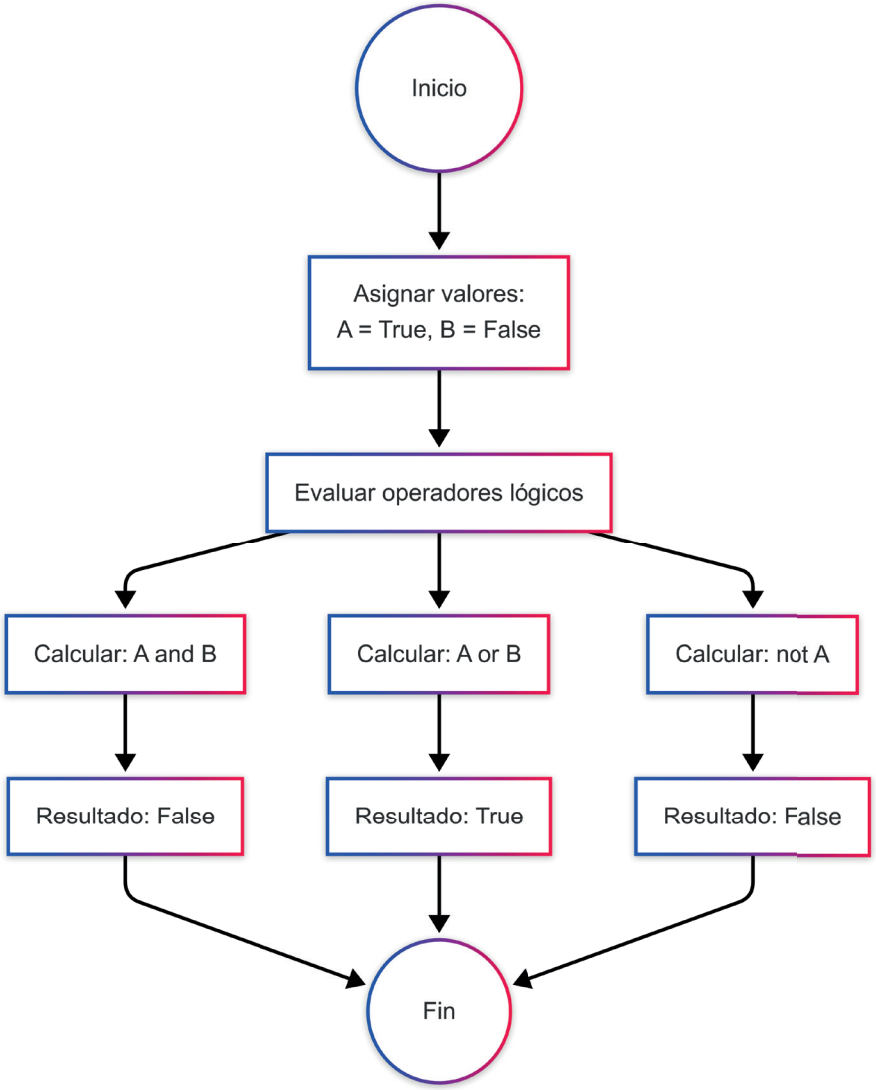



**Imagen 2: Ejemplo de Operadores Lógicos en un Diagrama**
Damián Nicolalde Rodríguez. (2024). Ejemplo de Operadores.

**De Morgan y la Lógica Booleana**

En la lógica booleana, las leyes de De Morgan establecen reglas para la conversión de expresiones cuando se aplica la negación a condiciones combinadas:

1. not (A and B) es equivalente a (not A) or (not B).

2. not (A or B) es equivalente a (not A) and (not B).

Estas leyes resultan útiles para simplificar o reescribir expresiones complejas, mejorando la legibilidad y facilitando la identificación de errores lógicos. Por ejemplo, si se desea evaluar la negación de una expresión and,

se puede transformar en un or negando cada uno de los operandos, lo que a veces hace el código más claro.

Para complementar y profundizar, se sugieren acceder a este recurso:

- **Título del enlace relacionado**: Guía Completa de Operaciones lógicos (and, or, not) en Python
- **Descripción del enlace relacionado:** Este video es una guía exhaustiva para entender y aplicar los operadores lógicos en Python, fundamentales para construir condiciones y tomar decisiones en programación.
- **Enlace: https://www.youtube.com/watch?v=vPbEBZb0mDY**

Para ilustrar los operadores lógicos en escenarios reales, se presenta la siguiente tabla que describe algunos casos de uso comunes, explicando cómo se aplican and, or y not para tomar decisiones:

Tabla 4: Casos de Uso de Operadores Lógicos

| Caso de Uso | Operadores Involucrados | Descripción | Ejemplo de Código |
|---|---|---|---|
| Validación de Acceso | and, or | Combina roles y permisos para decidir si un usuario puede entrar a un sistema. and garantiza que se cumplan todas las condiciones, or facilita condiciones alternativas. | rol = “admin”<br>tiene_permiso = True<br>acceso = (rol == “admin”) or (tiene_permiso) |
| Procesamiento de Formularios | and, not | Verifica que los campos requeridos no estén vacíos y que las condiciones se cumplan (por ejemplo, edad mayor de 18 y correo válido). | edad = 20<br>email_valido = True<br>form_completo = (edad >= 18) and (email_valido) |
| Control de Bucles | and, or | Se emplean para determinar si se continúa iterando (por ejemplo, mientras un contador no exceda un límite o se cumpla alguna condición). | contador = 0<br>while (contador < 5) or (usuario_quiere_seguir):<br>contador += 1 |
| Configuración de Parámetros | not | Se puede usar not para invertir condiciones (por ejemplo, si no se ha establecido una variable de entorno, asignar un valor por defecto). | config = None<br>if not config:<br>config = “default_config” |

Damián Nicolalde Rodríguez. (2024).

## INTEGRACIÓN DE CONCEPTOS Y EJEMPLO PRÁCTICO

Para consolidar los conocimientos de tipos de datos, operadores aritméticos y operadores lógicos, se presenta

el siguiente ejemplo práctico:

Ejercicio de Aplicación:

1. Solicitar al usuario su nombre, su edad y dos números enteros.
2. Calcular la suma, la resta, la multiplicación y el módulo de los números ingresados.
3. Evaluar, mediante operadores lógicos, si el usuario es mayor de edad y si alguno de los números es mayor que un umbral (por ejemplo, 50).
4. Mostrar un mensaje personalizado basado en la evaluación de las condiciones.

Ejemplo de Código:

```
# Solicitar datos al usuario
nombre = input("Ingrese su nombre: ")
edad = int(input("Ingrese su edad: "))
num1 = int(input("Ingrese el primer número: "))
num2 = int(input("Ingrese el segundo número: "))

# Operaciones aritméticas
suma = num1 + num2
resta = num1 - num2
multiplicacion = num1 * num2
modulo = num1 % num2

# Evaluación lógica
umbral = 50
mayor_de_edad = (edad >= 18)
supera_umbral = (num1 > umbral) or (num2 > umbral)
condicion_final = mayor_de_edad and supera_umbral

# Mostrar resultados
print(f"\nHola, {nombre}. Estos son los resultados de tus cálculos:")
print(f"Suma: {suma}, Resta: {resta}, Multiplicación: {multiplicacion}, Módulo: {modulo}")
```

```
if condicion_final:
    print("Eres mayor de edad y al menos uno de los números supera el umbral.")
else:
    print("No cumples con ambas condiciones al mismo tiempo.")
```

**Explicación del Código:**

- Se declaran variables para almacenar la información ingresada por el usuario, aplicando las convenciones de nombres en minúsculas y usando snake_case (por ejemplo, mayor_de_edad, supera_umbral).
- Se utilizan operadores aritméticos para calcular la suma, resta, multiplicación y módulo.
- Se combinan operadores lógicos (and, or) para evaluar condiciones complejas, determinando si el usuario es mayor de edad y si alguno de los números supera el umbral de 50.
- Se imprime un mensaje personalizado que integra los resultados de los cálculos y la evaluación lógica.

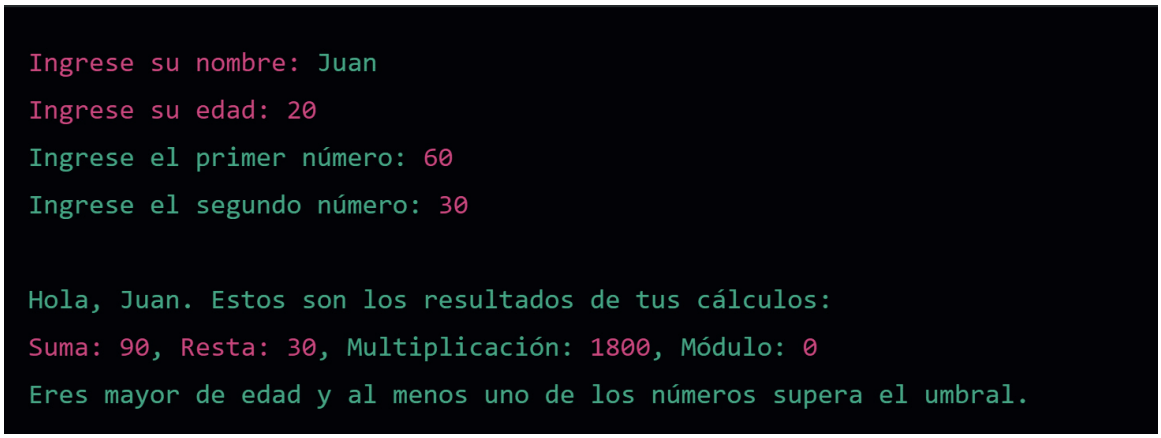



Imagen 3: Captura de Pantalla del Programa Integrado

Damián Nicolalde Rodríguez. (2024). Captura de Pantalla del Programa Integrado.

**Referencias citadas en la Clase 4.**


- https://elibro.puce.elogim.com/es/ereader/puce/230298
- https://puce.odilo.us/info/facil-aprendizaje-estructuras-de-datos-algoritmos-c-aprenda-facilmente-estructuras-de-datos-graficamente-03127232
- https://puce.odilo.us/info/aprende-c-en-un-fin-de-semana-03105596


**Definición de los términos citados en la Clase 4.**

Transformar y evaluar información: Este término hace referencia al proceso mediante el cual se toman datos de entrada (numéricos, textuales o lógicos) y se aplican operaciones (aritméticas, lógicas, de comparación, etc.) para generar nuevos resultados o conclusiones. En programación, transformar implica modificar o combinar la información (por ejemplo, sumando valores, calculando promedios o concatenando textos), mientras que evaluar consiste en analizar los datos para decidir el flujo de un programa (por ejemplo, verificar si un número excede un umbral). Este ciclo de transformación y evaluación de información es esencial para resolver problemas y diseñar algoritmos eficientes.

Expresiones booleanas: Las expresiones booleanas son aquellas que, al evaluarse, arrojan un resultado de True o False. Se construyen mediante operadores de comparación (por ejemplo, ==, >, <) y operadores lógicos (como and, or, not) aplicados a valores o variables. Estas expresiones permiten tomar decisiones en un programa, ya que sirven como condiciones en estructuras de control (if, while, for), determinando el curso de ejecución de un algoritmo en función de la lógica definida.

**Profundización Clase 4.**

Recurso_profundizacion_clase4.docx

PUCE Virtual

**La excelencia no se improvisa**

síguenos