**Módulo 2.1**

**Estructuras de datos Simples en Python**

**Introducción**

En Python, las estructuras de datos simples son tipos de datos fundamentales que constituyen la base de cualquier programa. Estas estructuras incluyen enteros (int), flotantes (float), booleanos (bool) y cadenas de texto (str). Cada una de estas estructuras tiene características particulares en cuanto a cómo se almacenan en memoria, su comportamiento en términos de mutabilidad e inmutabilidad, y su uso en la programación. A lo largo de este documento, exploraremos estas estructuras en detalle, proporcionando una base sólida para estudiantes universitarios con un nivel avanzado de comprensión.

|  |
| --- |
| **Tipos de Datos Simples** |

.

|  |  |  |
| --- | --- | --- |
| 1. **Enteros (**int**)**  El tipo de dato int en Python se utiliza para representar números enteros, tanto positivos como negativos, sin parte decimal. Python permite trabajar con enteros de tamaño arbitrario, lo que significa que no hay un límite práctico en la cantidad de dígitos que un número entero puede tener, solo está limitado por la memoria disponible.  **Ejemplo**:   |  |  | | --- | --- | | Código Python   |  | | --- | | variable\_entero = 1 |   · |   .  **Almacenamiento en Memoria**  En versiones más antiguas de Python (Python 2.x), existían dos tipos de enteros: int (de tamaño fijo, basado en la arquitectura de la máquina) y long (de tamaño ilimitado). Sin embargo, en Python 3.x, estos dos tipos se unificaron bajo int, que puede manejar enteros de tamaño ilimitado. Internamente, Python gestiona el almacenamiento de enteros en memoria mediante una estructura que incluye:   * Un puntero que indica la ubicación del objeto en la memoria. * Un contador de referencia para el recolector de basura. * Un campo que almacena el valor del entero.   Cuando un entero es pequeño, se almacena directamente en la memoria en un bloque específico. Si es grande, Python utiliza una representación más compleja, dividiendo el número en "dígitos" que se almacenan en bloques de memoria consecutivos.  En Python, un entero se almacena como un objeto completo, lo que incluye:   * Ob\_header: Un encabezado que contiene información como el tipo del objeto y el contador de referencias. * Ob\_digit: Una representación del valor entero, que puede ocupar múltiples bytes dependiendo del tamaño del número.   Python utiliza una representación interna optimizada para números pequeños (por debajo de cierta magnitud), conocidos como "small integers". Estos se almacenan en una tabla de valores preasignados para mejorar el rendimiento. |

**.**

|  |  |  |
| --- | --- | --- |
| 2. **Flotantes (**float**)**  El tipo float se utiliza para representar números reales que tienen una parte decimal. Estos números se almacenan en un formato de punto flotante de precisión doble, siguiendo el estándar IEEE 754, lo que significa que tienen una precisión de aproximadamente 15-17 dígitos decimales significativos.  **Ejemplo**:   |  |  | | --- | --- | | Código Python   |  | | --- | | variable\_real = 123.321 |   · |   .  **Almacenamiento en Memoria**  Un número flotante se almacena en memoria como un número binario en notación científica. La memoria se divide en tres partes:   * ****Signo** (1 bit):** Indica si el número es positivo o negativo. * **Exponente (11 bits)**: Representa la magnitud de la base (2). * **Mantisa (52 bits):** Contiene el valor numérico real.   El uso del estándar IEEE 754 permite a los float en Python manejar un rango extremadamente amplio de valores, aunque con ciertas limitaciones en precisión, especialmente al realizar operaciones aritméticas entre números que son muy cercanos entre sí.  Esta representación permite que los números flotantes cubran un rango muy amplio de valores, tanto extremadamente grandes como extremadamente pequeños, aunque a costa de posibles problemas de precisión cuando se manejan números muy cercanos entre sí. |

.

|  |  |  |
| --- | --- | --- |
| 3. **Booleanos (**bool**)**  El tipo bool en Python tiene dos valores posibles: True y False. Es una subclase del tipo int, donde True se evalúa como 1 y False como 0.  **Ejemplo**:   |  |  | | --- | --- | | Código Python   |  | | --- | | variable\_verdadero = True  variable\_falso = False |   · |   .  **Almacenamiento en Memoria**  Un bool se almacena internamente de manera similar a un entero, con una optimización específica para ahorrar espacio. Aunque un bool solo necesita un bit para representar True o False, en la práctica se utiliza al menos un byte de memoria debido a la alineación de datos en la mayoría de los sistemas |

.

|  |  |  |
| --- | --- | --- |
| **4.Cadenas de Texto (str)** El tipo str en Python se utiliza para representar secuencias de caracteres. Aunque las cadenas pueden parecer similares a colecciones como listas debido a que son iterables y tienen índices, en Python se consideran estructuras simples debido a su inmutabilidad y su enfoque en la representación de texto.  **Ejemplo**:   |  |  | | --- | --- | | Código Python   |  | | --- | | cadena = "Hola, Mundo" |   · |   . **Almacenamiento en Memoria**  Las cadenas de texto en Python se almacenan como objetos inmutables, lo que significa que cualquier operación que modifique una cadena realmente crea un nuevo objeto en la memoria. Internamente, una cadena de texto se almacena como una secuencia de valores Unicode, lo que permite a Python manejar una amplia gama de caracteres y símbolos de diferentes idiomas y conjuntos de caracteres. **La memoria utilizada por una cadena incluye:**   * Ob\_header: Información del objeto, como el tipo y el contador de referencias. * Ob\_size: La longitud de la cadena. * Ob\_digit: La secuencia de caracteres codificada.   Cada carácter en una cadena puede ocupar 1, 2, o más bytes dependiendo de la codificación utilizada (UTF-8, UTF-16, etc.). Esta flexibilidad permite a Python trabajar eficientemente con texto en un entorno globalizado. |

.

|  |
| --- |
| **Concepto de Mutabilidad e Inmutabilidad** |
| En Python, los objetos pueden ser mutables o inmutables. Este concepto es crucial para entender cómo los datos se gestionan y cómo el comportamiento de una variable puede cambiar según su tipo.  **Objetos Inmutables**  Los objetos inmutables son aquellos cuyo estado no puede ser modificado después de su creación. Si se intenta cambiar el valor de un objeto inmutable, se crea un nuevo objeto en la memoria y la variable se actualizará para apuntar a este nuevo objeto. Ejemplos de tipos de datos inmutables son int, float, bool, str (cadenas de texto), y tuple (tuplas).   * **Seguridad y Simplicidad**: Los objetos inmutables pueden ser compartidos entre diferentes partes de un programa sin riesgo de que sean modificados accidentalmente. * **Optimización**: Python puede optimizar el uso de la memoria reutilizando objetos inmutables en lugar de crear nuevos cada vez que se necesita un valor idéntico.   **Ejemplo**:   |  |  | | --- | --- | | Código Python   |  | | --- | | x = 5  x = 6 # Se crea un nuevo objeto '6' en memoria, y 'x' apunta a él |   · |   .  Cuando x se asigna el valor 6, Python no cambia el valor del objeto que x apuntaba inicialmente (5), sino que crea un nuevo objeto 6 y luego reasigna x para que apunte a este nuevo objeto. Este comportamiento es crucial para comprender cómo funcionan las variables y evita efectos secundarios inesperados en el código.   |  |  | | --- | --- | | Código Python   |  | | --- | | cadena = "Python"  nueva\_cadena = cadena.replace("P", "J")  # 'cadena' sigue siendo "Python", 'nueva\_cadena' es "Jython" |   · |   .  Aquí, la cadena original cadena no se modifica; en su lugar, se crea una nueva cadena nueva\_cadena.  **Objetos Mutables**  Los objetos mutables, por otro lado, pueden cambiar su estado o contenido sin necesidad de crear un nuevo objeto en la memoria. Las listas (list), conjuntos (set) y diccionarios (dict) son ejemplos de tipos de datos mutables.  Aunque los tipos de datos simples que estamos tratando (int, float, bool) son inmutables, este concepto es relevante para comprender cómo diferentes tipos de datos pueden comportarse de manera diferente cuando se manipulan en un programa. |

|  |
| --- |
| **Variables y Constantes** |
| **Variables**  En Python, una variable es un nombre que se asocia con un valor almacenado en la memoria. Las variables son "cajas" que pueden contener cualquier tipo de dato, y el valor almacenado puede cambiar a lo largo de la ejecución del programa.  **Ejemplo**:   |  |  |  | | --- | --- | --- | | Código Python   |  | | --- | | variable = 5+2+3  print (f'{variable=}') |   Salida esperada por consola   |  | | --- | | Variable = 10 |   · |   .  En este ejemplo, la variable variable se asocia con el valor 10, que es un objeto entero en la memoria. Si se reasigna un nuevo valor a variable, Python simplemente actualiza la referencia para apuntar al nuevo objeto en memoria.  **Ejemplo**:   |  |  |  | | --- | --- | --- | | Código Python   |  | | --- | | Variable = 10  variable = "Ahora soy una cadena"  print (f'{variable=}') |   Salida esperada por consola   |  | | --- | | variable = "Ahora soy una cadena" |   · |   ,  En este caso, la variable x inicialmente apunta a un entero, pero luego se reasigna a una cadena de texto. Esto es posible gracias a la naturaleza dinámica y tipada de Python.  **Constantes**  Aunque Python no tiene un mecanismo explícito para declarar constantes (valores que no deberían cambiar), se acostumbra a utilizar nombres de variables en mayúsculas para indicar que un valor no debe ser modificado.  **Ejemplo**:   |  |  |  | | --- | --- | --- | | Código Python   |  | | --- | | PI = 3.14159 # ve r que esta en mayúsculas  print (f'{PI=}') |   Salida esperada por consola   |  | | --- | | PI = 3.14159 |   · |   .  Aunque nada impide que PI se reasigne, por convención se considera que no debe ser alterado. Las constantes son útiles para valores que se mantienen fijos a lo largo del programa, lo que mejora la legibilidad y reduce errores. |

**Conclusión**

Las estructuras de datos simples en Python, como int, float, bool, y str, son los bloques fundamentales que soportan la mayoría de las operaciones y construcciones más complejas en el lenguaje. Comprender cómo se almacenan en memoria, cómo se comportan en términos de mutabilidad e inmutabilidad, y cómo se gestionan mediante variables y constantes es crucial para cualquier desarrollador que busque escribir código robusto y eficiente.

Esta comprensión también sienta las bases para explorar estructuras de datos más avanzadas y conceptos en programación, lo que es esencial para la manipulación de datos y la construcción de algoritmos en Python.