TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

FUNDAMENTOS DE PROGRAMACIÓN

TIPOS DE VARIABLES Y CONSTANTES EN PROGRAMACIÓN

TIPOS DE VARIABLES Y CONSTANTES EN PROGRAMACIÓN

En el desarrollo de software, comprender cómo se clasifican y utilizan las variables y constantes es fundamental para construir programas claros, seguros y organizados. Las variables permiten almacenar y manipular información, mientras que las constantes garantizan la estabilidad de ciertos valores durante toda la ejecución. Según su comportamiento, visibilidad y persistencia, las variables pueden adoptar distintas formas que influyen directamente en la estructura y lógica del código. Esta introducción aborda los principales tipos de variables —locales, globales, estáticas, de instancia y de clase—, así como el papel esencial de las constantes en distintos lenguajes de programación.

1. Tipos de variables

Las variables en programación pueden clasificarse en diferentes categorías según su alcance, visibilidad, persistencia y forma de uso. Cada tipo de variable cumple una función específica dentro de un programa y su correcta utilización contribuye a la eficiencia del código (Zohonero Martínez & Joyanes Aguilar, 2008).

Variables locales

Las variables locales son aquellas que se declaran dentro de una función, método o bloque de código y solo pueden ser utilizadas dentro de ese contexto. Una vez que la ejecución abandona dicho bloque, la variable deja de existir.

Ejemplo en Python:

En este caso, la variable resultado solo existe dentro de la función calcular_area, por lo que no puede ser accedida fuera de ella.

Figura 1. Ejemplo de variable Python

```python
def calcular_area(base, altura):
    resultado = (base * altura) / 2  # 'resultado' es una variable local
    return resultado

print(calcular_area(5, 10))  # Llamado a la función
```

Ejemplo en Java:

La variable numero es local a main(), lo que significa que no puede ser utilizada fuera de ese método.

Figura 2. Variable local - Java [IDE]

```java
public class Ejemplo {
    public static void main(String[] args) {
        int numero = 10;  // Variable local dentro del método main
        System.out.println("El número es: " + numero);
    }
}
```

Variables globales

Las variables globales son aquellas que se declaran fuera de cualquier función o método y pueden ser accedidas desde cualquier parte del código. Sin embargo, su uso excesivo puede generar problemas de mantenimiento y errores difíciles de rastrear (Flórez Fernández, 2012).

Ejemplo en JavaScript:

En este caso, la variable mensaje puede ser utilizada dentro de la función mostrarMensaje() y en cualquier otro lugar del código.

Figura 3. Variable global

```javascript
let mensaje = "Hola, mundo";  // Variable global

function mostrarMensaje() {
    console.log(mensaje);  // Se accede a la variable global dentro de la función
}

mostrarMensaje();
```

Variables estáticas

Las variables estáticas conservan su valor entre diferentes llamadas a la función o instancia en la que se declaran. En muchos lenguajes, estas variables sólo se inicializan una vez y mantienen su valor en memoria.

Ejemplo en C:

A diferencia de una variable local normal, la variable contador no se reinicia en cada llamada a la función.

Figura 4. Variable estática en C

```c
#include <stdio.h>

void contar() {
    static int contador = 0;  // Variable estática
    contador++;
    printf("Contador: %d\n", contador);
}

int main() {
    contar();
    contar();
    contar();
    return 0;
}
```

Ejemplo en Java:

Aquí, contador pertenece a la clase y conserva su valor entre llamadas a incrementar().

Figura 5. Variable estática en Java

```java
class Ejemplo {
    static int contador = 0; // Variable estática

    public static void incrementar() {
        contador++;
        System.out.println("Contador: " + contador);
    }

    public static void main(String[] args) {
        incrementar();
        incrementar();
    }
}
```

Aguilar, J. (2025). Creado con Visual Studio Code.

Variables de instancia

Las variables de instancia pertenecen a un objeto específico y su valor es único para cada instancia de la clase. Se definen dentro de una clase pero fuera de sus métodos (Flórez Fernández, 2012).

Ejemplo en Python:

Cada objeto de la clase Persona tiene su propia versión de nombre y edad.

Figura 6. Variable de instancia

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre  # Variable de instancia
        self.edad = edad      # Variable de instancia

p1 = Persona("Ana", 30)
p2 = Persona("Luis", 25)

print(p1.nombre)  # Ana
print(p2.nombre)  # Luis
```

Ejemplo en Java:

Cada objeto de Persona tiene su propio valor de nombre y edad.

Figura 7. Variable de instancia en Java

```java
class Persona {
    String nombre;  // Variable de instancia
    int edad;       // Variable de instancia

    Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}

public class Main {
    public static void main(String[] args) {
        Persona p1 = new Persona("Carlos", 28);
        Persona p2 = new Persona("Lucía", 24);

        System.out.println(p1.nombre);  // Carlos
        System.out.println(p2.nombre);  // Lucía
    }
}
```

Variables de clase

Las variables de clase son compartidas entre todas las instancias de una clase. Se definen con la palabra clave static en algunos lenguajes.

Ejemplo en Python:

Aquí, la variable total es compartida entre todas las instancias de Contador.

Figura 8. Variable de clase

```python
class Contador:
    total = 0  # Variable de clase

    def __init__(self):
        Contador.total += 1

print(Contador.total)  # 0
c1 = Contador()
c2 = Contador()
print(Contador.total)  # 2
```

Ejemplo en Java:

Cada vez que se crea un nuevo objeto Contador, la variable total se incrementa, reflejando el número total de instancias creadas.

Figura 9. Variable de clase en Java

```java
class Contador {
    static int total = 0; // Variable de clase

    Contador() {
        total++;
    }

    public static void main(String[] args) {
        new Contador();
        new Contador();
        System.out.println("Total de objetos creados: " + total); // 2
    }
}
```

2. Constantes en Programación

Las constantes en programación son elementos fundamentales que permiten definir valores inmutables dentro de un código. A diferencia de las variables, las constantes mantienen su valor a lo largo de toda la ejecución del programa, lo que garantiza estabilidad y coherencia en los datos (Zohonero Martínez & Joyanes Aguilar, 2008).

El uso de constantes mejora la legibilidad, facilita el mantenimiento del código y evita errores involuntarios al prevenir la modificación de valores críticos.

Una constante es un identificador que almacena un valor fijo que no puede ser alterado después de su declaración. Su principal propósito es proporcionar seguridad en la gestión de datos y evitar cambios accidentales que podrían afectar el comportamiento del programa (Zohonero Martínez & Joyanes Aguilar, 2008).

Ejemplo en Python:

En este caso, PI es una constante utilizada para el cálculo del área de un círculo. Aunque Python no impide modificar su valor, la convención de escritura en mayúsculas indica que debe tratarse como inmutable.

Figura 10. Constante en Python

```python
PI = 3.1416  # Se define una constante en mayúsculas por convención
radio = 5
area = PI * (radio ** 2)

print(f"El área del círculo es: {area}")
```

Tabla 1. Diferencias entre Variables y Constantes

Característica	Variables	Constantes
Mutabilidad	Puede cambiar durante la ejecución	No puede ser modificada una vez definir
Almacenamiento	Puede ser reasignada	Se mantiene fija en memoria
Uso común	Almacenar datos que cambian dinámicamente	Definir datos fijos

Características de las constantes

- **Inmutabilidad:** Una vez que se asigna un valor a una constante, este no puede cambiar.
- **Legibilidad y mantenibilidad:** Usar constantes con nombres descriptivos mejora la comprensión del código.
- **Optimización:** Algunas optimizaciones del compilador son posibles cuando se detecta que un valor no cambiará.
- **Seguridad:** Evita cambios accidentales en valores críticos del programa.

Tema 1.5.4: Declaración y uso de constantes en diferentes lenguajes

Cada lenguaje de programación maneja las constantes de manera particular. A continuación, se presentan ejemplos en varios lenguajes:

Constantes en C++

C++ hereda el uso de const de C y también introduce constexpr, que permite definir constantes en tiempo de compilación.

Figura 11. Constante en C++

```cpp
constexpr double GRAVEDAD = 9.81;  // Constante evaluada en tiempo de compilación

int main() {
    const int EDAD_MINIMA = 18;
    std::cout << "La gravedad en la Tierra es: " << GRAVEDAD << " m/s²" << std::endl;
    std::cout << "La edad mínima permitida es: " << EDAD_MINIMA << std::endl;
    return 0;
}
```

Constantes en Java

En Java, las constantes se definen con la palabra clave final, lo que impide su modificación tras la inicialización.

Figura 12. Constante en Java

```java
public class Constantes {
    public static final double VELOCIDAD_LUZ = 299792458;  // en metros por segundo

    public static void main(String[] args) {
        final int MAX_INTENTOS = 5;
        System.out.println("La velocidad de la luz es: " + VELOCIDAD_LUZ + " m/s");
        System.out.println("Número máximo de intentos permitidos: " + MAX_INTENTOS);
    }
}
```

Constantes en Python

Python no posee una sintaxis específica para constantes, pero por convención, se utilizan nombres en mayúsculas para indicar que una variable no debe cambiar.

Figura 13. Constantes en Python

```python
PI = 3.14159  # Convención para constantes en Python

def calcular_circunferencia(radio):
    return 2 * PI * radio

print("Circunferencia de un círculo con radio 5:", calcular_circunferencia(5))
```

Tipos de constantes

Dependiendo del lenguaje y contexto, las constantes pueden clasificarse en diferentes tipos:

1. **Constantes numéricas:** Enteros, flotantes, números en notación científica.
    a. **Ejemplo:** const int EDAD_MINIMA = 18;
2. **Constantes de cadena:** Almacenan texto.
    a. **Ejemplo:** const string MENSAJE = "Bienvenido";
3. **Constantes booleanas:** Representan valores true o false.
    a. **Ejemplo:** const bool ACTIVO = true;
4. **Constantes definidas por el sistema:** Algunos lenguajes incluyen constantes predefinidas.
    a. **Ejemplo en PHP:** PHP_VERSION devuelve la versión del intérprete.
5. **Constantes simbólicas:** En algunos lenguajes, como Ruby, existen constantes representadas como símbolos (:nombre).

Beneficios del uso de constantes

El uso de constantes ofrece diversas ventajas en el desarrollo de software:

- ☑ **Claridad y significado:** Evita el uso de valores "mágicos" en el código, haciendo que sea más fácil de entender.
- ☑ **Facilidad de mantenimiento:** Si un valor necesita cambiarse, solo se modifica en un solo lugar.
- ☑ **Prevención de errores:** Reduce la posibilidad de cambiar accidentalmente valores críticos.
- ☑ **Optimización:** Algunos compiladores pueden realizar optimizaciones basadas en valores constantes.

Bibliografía

- Flórez Fernández, H. A. (2012). Programación orientada a objetos usando java. Ecoe Ediciones. https://elibro.net/es/lc/tecnologicadeloriente/titulos/69236
- Zohonero Martínez, I., & Joyanes Aguilar, L. (2008). Estructuras de datos en Java. McGraw-Hill España. https://elibro.net/es/lc/tecnologicadeloriente/titulos/50117