
# Mejores practicas en la programaciòn

DOJO.NET
(DIGITALGEKO)

# Prefijos que indiquen el tipo de datos

La nomenclatura húngara usa prefijos para indicar el tipo de datos o la intención de una variable. Es fundamental que estos prefijos sean claros y consistentes para que el código sea legible y fácil de entender.

**Ejemplos:**

- **strNombre** para una variable de tipo string (cadena de texto).
- **iEdad** para una variable de tipo entero (int).
- **fPrecio** para una variable de tipo flotante (float).
- **bEsActivo** para una variable de tipo booleano (bool).
- **arrNombres** para una variable que es un arreglo (array) de strings.

# Usar convenciones estándar para tipos comunes

Algunos lenguajes tienen convenciones que se pueden seguir, como:

- **i** para enteros (int): iEdad, iCuenta.
- **f** para flotantes (float): fAltura, fTemperatura.
- **b** para booleanos (bool): bEsValido, bActivo.
- **str** para cadenas de texto (string): strNombre, strDireccion.
- **arr** para arreglos o listas: arrNumeros, arrColores.

# Describir el propósito de la variable con el prefijo

Aunque el tipo de datos es importante, también lo es la **intención** o **uso** de la variable. Es una buena práctica combinar la nomenclatura húngara con una descripción adecuada del propósito de la variable.

**Ejemplos:**

- pntPosicion para un punto (point) en 2D o 3D (puede representar coordenadas).
- strNombreUsuario para un nombre de usuario.
- dtFechaNacimiento para una fecha.

# Usar un prefijo diferente para los punteros

En lenguajes como C o C++, los punteros pueden ser indicados con un prefijo **p** para que sea evidente que se trata de un puntero.

**Ejemplo:**

- pStrNombre para un puntero a una cadena de texto.
- pIntContador para un puntero a un entero.

# Usar nombres más largos y descriptivos

Aunque el uso de la nomenclatura húngara implica abreviaciones, es importante que los nombres no sean **demasiado cortos** y sean lo suficientemente descriptivos.

**Ejemplo:**

•Evitar i como nombre de variable, en su lugar, usar algo como iContador o iTotalVentas.

# Evitar sobrecargar de prefijos

No es recomendable añadir prefijos innecesarios. Es decir, **no** agregar el prefijo si no aporta claridad sobre el tipo de datos o el propósito de la variable. Si el contexto es claro, puedes omitir el prefijo.

**Ejemplo:**

- No es necesario poner str a variables de tipo string si el contexto ya lo deja claro.
- En objetos complejos, como instancias de clases, no siempre es necesario el prefijo obj (e.g., objEmpleado).

# Usar la nomenclatura húngara para funciones y procedimientos

Aunque la nomenclatura húngara es más comúnmente aplicada a las variables, también puede ser útil en funciones y procedimientos.

**Ejemplo:**

- getStrNombre() para una función que devuelve una cadena de texto.
- setIntEdad() para un setter de tipo entero.

# Manejo de variables globales

Si bien la nomenclatura húngara puede aplicarse a todas las variables, algunas personas prefieren usarla más para las variables globales o las variables de uso específico, ya que puede ayudar a identificar rápidamente si una variable es local o global. Algunos prefieren prefijos como g para variables globales.

**Ejemplo:**

- gIntTotal para una variable global que mantiene un total.

# Documentación adicional

A pesar de que la nomenclatura húngara ayuda a entender rápidamente el tipo de una variable, sigue siendo importante documentar el código de manera adecuada, especialmente en proyectos más grandes, para asegurar la comprensión del contexto.

| Tipo de dato | Prefijo | Ejemplo |
|---|---|---|
| Entero (int) | `i` | `iEdad`, `iValor` |
| Flotante (float) | `f` | `fTemperatura`, `fPrecio` |
| Booleano (bool) | `b` | `bEsActivo`, `bValido` |
| Cadena (string) | `str` | `strNombre`, `strDireccion` |
| Arreglo (array) | `arr` | `arrNumeros`, `arrColores` |
| Objeto (objeto de clase) | `obj` | `objEmpleado`, `objEstudiante` |
| Puntero | `p` | `pIntContador`, `pStrNombre` |

# Trazabilidad en el manejo de errores y excepciones

Los errores y excepciones deben ser correctamente gestionados y trazados para entender qué salió mal en la aplicación. Algunas buenas prácticas incluyen:

- **Captura de excepciones**: Asegúrate de capturar excepciones de manera adecuada y registrar información relevante, como el stack trace y el contexto de la excepción.
- **Notificación de errores críticos**: Los errores críticos deben ser notificados inmediatamente al equipo de desarrollo o a los administradores del sistema, utilizando herramientas de notificación.
- **Contexto en los errores**: Proporciona detalles adicionales en los mensajes de error (por ejemplo, el usuario afectado, los parámetros de la solicitud, etc.).