

Cómo escribir buen pseudocódigo

IIC2233 — Programación Avanzada

Resumen elaborado por el equipo de ayudantía, en base a material de referencia externo



Felipín se perdió de nuevo. Esta vez no fue por usar `list` para todo: Felipín, apenas se le ocurre una idea, **empieza a escribir Python directamente**, sin pensar antes los pasos. El resultado: funciones a medio hacer, casos que olvidó revisar, y un código imposible de seguir.

No seas como Felipín. Antes de escribir una sola línea de Python, conviene **planificar los pasos del algoritmo** en pseudocódigo: una forma de describir la lógica sin preocuparse (todavía) de la sintaxis exacta.

! Cómo leer este resumen

Las Secciones 3 a 5 son secciones bonus que explican de forma más detallada la importancia y la forma general de un pseudocódigo.

Tabla de Contenidos

1 ¿Qué es el pseudocódigo? 1
1.1 Esto NO es pseudocódigo . . . 2
1.2 Esto SI es pseudocódigo . . . 2

2 La convención de este curso 3
2.1 Ejemplo: validador de contraseña . . . 4

3 Las construcciones principales (BONUS) 5
3.1 Ejemplo mínimo: `IF-THEN-ELSE` . . . 6

4 Pseudocódigo y diagramas de flujo (BONUS) 6

5 ¿Por qué vale la pena usarlo? (BONUS) 7

Bibliografía y material de referencia 7

1 ¿Qué es el pseudocódigo?

El **pseudocódigo** es una forma de describir los pasos de un algoritmo usando lenguaje cotidiano, pero **organizado como si fuera código**. No es un lenguaje de programación: no existe un "compilador de pseudocódigo". Es, más bien, un **punto intermedio** entre una idea en palabras y su implementación real en un lenguaje como Python.

1.1 Esto NO es pseudocódigo

Antes de ver qué sí es pseudocódigo, es útil ver algunos casos que **no lo son**, aunque a veces se confundan con él:

Caso 1: una frase demasiado vaga

> *"Recorrer la lista y ver los números buenos, después mostrar el resultado."*

Esto no es pseudocódigo: no dice qué significa "bueno", no define ninguna condición, ni un orden claro de pasos. El pseudocódigo debe ser **preciso en la lógica**, aunque sea informal en la sintaxis.

Caso 2: solo una descripción en prosa, sin estructura

> *"El algoritmo revisa cada elemento de la lista y compara con el máximo actual; si es mayor, lo reemplaza; al final, muestra el máximo."*

Aunque la lógica esté correcta, escribirla como un párrafo largo (sin líneas separadas, sin indentación, sin condicionales explícitos) dificulta seguir el flujo del algoritmo y traducirlo después a código. El pseudocódigo se beneficia de estar **estructurado en pasos**, como en la Sección 3.

1.2 Esto SI es pseudocódigo

Caso 1 corregido: esto SÍ es pseudocódigo

Retomando el mismo problema de antes, pero ahora definiendo qué significa "bueno" y con un orden de pasos explícito, usando sintaxis de Python:

```
lista_buenos = []

for numero in lista:
    if numero > 0 and numero es par:
        agregar numero a lista_buenos

escribir(lista_buenos)
```

Aquí la condición de "bueno" quedó explícita (mayor que cero y par), y cada paso está separado y ordenado: primero se recorre, después se decide, y al final se muestra. Nótese que se usa `for` e `if` tal como se explicó en la Sección 2, pero como quizás no recordamos el método exacto para agregar a una lista, simplemente se describe la idea (`agregar ... a ...`) en vez de detenerse a buscarlo.

Caso 2 corregido: esto SÍ es pseudocódigo

La misma idea de encontrar el máximo de una lista, pero ahora estructurada en pasos en vez de un párrafo:

```
maximo = lista[0]

for elemento in lista (desde el segundo en adelante):
    if elemento > maximo:
        maximo = elemento

escribir(maximo)
```

La lógica es la misma que la versión en prosa, pero ahora cada paso vive en su propia línea, con indentación y condicionales explícitos usando `for` e `if`. Esto hace mucho más fácil seguir el flujo y detectar, por ejemplo, si faltó inicializar `maximo` antes del ciclo. Como en el ejemplo de la Sección 2, se mezcla sintaxis exacta de Python (`maximo = elemento`) con una descripción en palabras donde no recordamos cómo expresarlo de forma exacta (`desde el segundo en adelante`).

Características del pseudocódigo

- Mezcla lenguaje natural con una estructura similar a la de un programa.
- Se enfoca en la **lógica**, no en la sintaxis de un lenguaje particular.
- Es **independiente del lenguaje**: el mismo pseudocódigo puede traducirse a Python, Java, C, etc.
- Debe ser lo bastante completo como para que traducirlo a código sea, en esencia, **traducir línea por línea**.
- Lo pueden entender tanto programadores como personas sin experiencia técnica.

★ ¿Por qué molestarse en escribirlo?

Escribir pseudocódigo antes de programar permite **detectar errores de diseño temprano** (antes de que existan como `bug`), facilita **comunicar el algoritmo** a otras personas (incluso sin conocimientos de Python), y sirve como una primera versión de la **documentación** del código. En proyectos simples puede parecer una pérdida de tiempo, pero en algoritmos más largos o complejos ahorra muchísimo tiempo de depuración después.

2 La convención de este curso

En este curso, cuando se pida escribir pseudocódigo, la idea es usar como base la **sintaxis de Python**: nombres de variables, `for`, `if`, `while`, indentación con espacios, etc. Esto tiene sentido porque Python es el lenguaje con el que ya estamos trabajando, y así el pseudocódigo queda mucho más cerca del código final.

Lo que se espera

- Intentar usar las estructuras de Python (`for`, `while`, `if/elif/else`, definición de funciones) siempre que las recuerdes.
- Si **no recuerdas** el nombre exacto de un método, una función *built-in*, o la sintaxis precisa de algo, **no te detengas a buscarlo**: describe la idea en palabras, como `escribir...` o `ordenar la lista de menor a mayor...`, y sigue con el resto del algoritmo.
- Lo esencial es que la **lógica** quede clara y completa: el orden de los pasos, las condiciones, los ciclos. La sintaxis exacta es secundaria.

★ ¿Y si sí me sé la sintaxis exacta?

En este curso, si **efectivamente sabes** la sintaxis correcta de Python para un paso del algoritmo, **puedes usarla directamente como pseudocódigo**: no es necesario "disfrazar" o simplificar algo que ya sabes escribir bien. Es perfectamente válido mezclar, en un mismo pseudocódigo, líneas con sintaxis exacta de Python donde te la sabes, y líneas más descriptivas

(como `escribir...`) donde no la recuerdes. Lo importante sigue siendo que la lógica completa quede clara, no que todo el bloque tenga el mismo nivel de detalle.

! No te obsesiones con la sintaxis

El objetivo del pseudocódigo **no** es que compile. Si en medio de una prueba o una tarea no recuerdas cómo se invierte una lista en Python, no dejes esa parte en blanco: escribe algo como `invertir(lista)` y sigue avanzando. Un pseudocódigo con una idea clara pero sintaxis imperfecta vale mucho más que uno incompleto.

2.1 Ejemplo: validador de contraseña

Queremos un algoritmo que reciba una contraseña y la considere válida si tiene al menos 8 caracteres **y** contiene al menos un número. Veamos el mismo algoritmo escrito de **tres formas distintas**, para dejar clara la diferencia entre el estilo esperado en este curso y el pseudocódigo clásico (que se muestra solo como referencia, **no** como lo que se pide entregar).

a) Estilo del curso, olvidando el nombre de una función

Usamos sintaxis de Python (`def`, `for`, `if`) tal como se explicó en esta sección. Supongamos que, al escribir esto, se te olvida cómo se llama la función que entrega el largo de un *string*: en vez de detenerte a buscarla, simplemente le pones un nombre razonable a tu propia idea (por ejemplo `largo()`) y sigues avanzando.

```
def es_valida(contraseña):
    tiene_numero = False

    for caracter in contraseña:
        if caracter.isdigit():
            tiene_numero = True

    if largo(contraseña) >= 8 and tiene_numero:
        return True
    else:
        return False
```

★ Nota sobre este caso

Todo el resto del algoritmo usa sintaxis exacta de Python porque nos la sabemos (`def`, `for`, `if.isdigit()`), tal como vimos en el recuadro anterior. Solo en la línea del largo inventamos un nombre razonable, `largo()`, porque en ese momento no recordamos que la función real se llama `len`. No hace falta describirlo con una frase larga: basta con un nombre que dé a entender claramente la idea. Esto sigue siendo un pseudocódigo perfectamente válido para este curso.

b) Pseudocódigo puro (estilo clásico BONUS) — solo como referencia

Esta es la versión más "tradicional", usando las construcciones de la Sección 3 (`INICIO`, `DEFINIR`, `PARA CADA`, `SI-ENTONCES-SINO`) en vez de sintaxis de ningún lenguaje en particular.

```
DEFINIR funcion es_valida(contraseña)
INICIO
    DEFINIR tiene_numero como Falso

    PARA CADA caracter EN contraseña HACER
        SI caracter es un digito ENTONCES
```

```
            DEFINIR tiene_numero como Verdadero
        FIN SI
    FIN PARA

    SI longitud(contraseña) >= 8 Y tiene_numero ENTONCES
        RETORNAR Verdadero
    SINO
        RETORNAR Falso
    FIN SI
FIN
```

Aquí ni siquiera usamos `len`: usamos la palabra `longitud`, en español y sin paréntesis de Python, porque este estilo no pretende parecerse a ningún lenguaje real.

c) Traducción final a Python

Cualquiera de las dos versiones anteriores se traduce al mismo código real:

```python
def es_valida(contraseña):
    tiene_numero = False

    for caracter in contraseña:
        if caracter.isdigit():
            tiene_numero = True

    if len(contraseña) >= 8 and tiene_numero:
        return True
    else:
        return False
```

★ ¿Ves el paralelo?

Las tres versiones describen exactamente **la misma lógica**. La diferencia está en cuánto se acercan a la sintaxis final de Python: la versión (a) ya casi es código, con una sola línea pendiente; la versión (b) es puro pseudocódigo, independiente del lenguaje; y la versión (c) es la traducción completa.Te recomendamos usar Python tanto como sepas, y recurre a una descripción o un nombre inventado solo en lo puntual que se te olvide.

3 Las construcciones principales (BONUS)

La lógica de casi cualquier algoritmo se puede describir combinando un pequeño conjunto de **construcciones** (también llamadas palabras clave), que tradicionalmente se escriben en **MAYÚSCULAS** para distinguirlas del resto del texto.

Las seis construcciones clásicas

- `SEQUENCE`: una secuencia de pasos que se ejecutan uno después del otro.
- `IF / THEN / ELSE`: una decisión que cambia el flujo del algoritmo según una condición.
- `CASE`: una generalización de `IF-THEN-ELSE` para varias alternativas a la vez.
- `WHILE`: un ciclo que revisa la condición **antes** de cada repetición.
- `REPEAT / UNTIL`: un ciclo que revisa la condición **después** de cada repetición.
- `FOR`: otra forma de repetir pasos, típicamente sobre un rango o una colección.

A estas seis se suelen agregar dos construcciones adicionales, muy útiles en la práctica:

Construcciones adicionales frecuentes

- `CALL`: para invocar una función, método o procedimiento ya definido.
- `EXCEPTION / WHEN`: para describir el manejo de errores o casos excepcionales.

! Estas palabras no son un estándar rígido

No existe un único "lenguaje oficial" de pseudocódigo: cada persona (o cada equipo) desarrolla su propio estilo. Lo importante es **ser consistente** dentro de un mismo documento, y nunca usar estas palabras clave como nombres de variables.

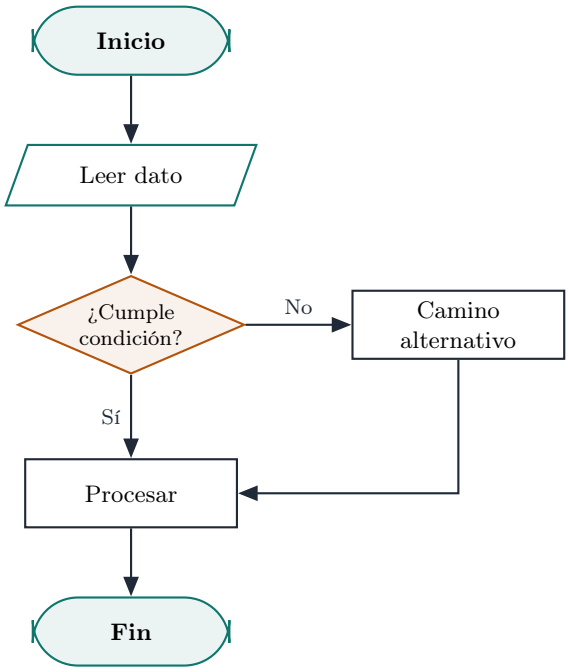
3.1 Ejemplo mínimo: `IF-THEN-ELSE`

```
SI el pedido es mayor o igual a $50 ENTONCES
    Aplicar descuento
    Mostrar "¡Descuento aplicado!"
SINO
    No aplicar descuento
    Mostrar "Agrega mas productos para el descuento"
FIN SI
```

Nótese que el pseudocódigo no dice *cómo* se representa el pedido en memoria, ni qué tipo de dato es: solo describe la **decisión** que debe tomar el algoritmo.

4 Pseudocódigo y diagramas de flujo (BONUS)

Un **diagrama de flujo** (*flowchart*) es la versión **visual** de un algoritmo: usa figuras estandarizadas conectadas por flechas para mostrar el orden de ejecución. Pseudocódigo y diagramas de flujo describen **lo mismo**, pero de formas distintas, y suelen usarse en conjunto.



Símbolos más comunes

- **Óvalo (terminal)**: marca el inicio o el fin del algoritmo.
- **Rectángulo (proceso)**: una operación que transforma datos (equivalente a una asignación o un cálculo).
- **Rombo (decisión)**: una condición que determina qué camino sigue el flujo.
- **Trapecio (entrada/salida)**: representa la lectura o la muestra de datos.
- **Flecha**: indica el orden en que se ejecutan los pasos.

★ ¿Cuándo usar cada uno?

El pseudocódigo es más **compacto** y cómodo para describir lógica detallada o algoritmos largos; el diagrama de flujo es más útil para **visualizar** decisiones complejas o explicarle el algoritmo a alguien sin experiencia técnica. No son excluyentes: muchas veces conviene usar ambos.

5 ¿Por qué vale la pena usarlo? (BONUS)

Beneficios principales

- **Comunicación**: permite explicar un algoritmo a compañeros, ayudantes o profesores sin discutir detalles de sintaxis.
- **Detección temprana de errores**: es mucho más fácil corregir un paso mal pensado en pseudocódigo que depurarlo después en código real.
- **Puente entre idea y código**: ordena el pensamiento antes de enfrentarse a la pantalla en blanco.
- **Base para documentación**: muchas veces el pseudocódigo termina siendo, casi literal, el primer borrador de los comentarios o el *docstring* de una función.

Bibliografía y material de referencia

Fuentes utilizadas en este resumen

- Metwalli, S. A. *Pseudocode: What It Is and How to Write It.* Built In. https://builtin.com/data-science/pseudocode
- Codecademy Team. *Pseudocode and Flowchart: Complete Beginner's Guide.* Codecademy. https://www.codecademy.com/article/pseudocode-and-flowchart-complete-beginners-guide
- Video complementario: https://www.youtube.com/watch?v=NJavXHU5MW8

Fin del resumen.