Integrantes: Fernando Huilca, Gregory Salazar, Mateo Simbaña

Curso: GR2SW

Fecha: 8 de noviembre de 2025

Aplicando principios de código limpio en proyectos reales

1. Introducción

- **¿Qué es el Código Limpio?**
  Primero hay que aclarar que este concepto no hace referencia a reglas estrictas, consiste en un conjunto de principios para producir código entendible y sencillo de modificar. Permite que la ejecución del sistema sea lógica y con una estructura simple, las relaciones entre los componentes sean obvias y visibles además de que las tareas de cada clase, función, método y variable son visiblemente comprensibles [1]. Además, no implica que se deberá escribir más o menos código, todo dependerá del contexto del sistema [2].

- **¿Por qué es importante? (mantenibilidad, escalabilidad, trabajo en equipo)**
  Permite que el código sea simple de modificar y ampliar, ayudando a reducir la gravedad de posibles errores no considerados. Por tanto, se permite tener un código sencillo de mantener. Además, permite que el código se vuelva independiente del desarrollador que lo creó y que cualquier otra persona pueda trabajar con el programa [1]. Entonces, el objetivo es entender que cualquiera puede escribir código que una computadora entienda, pero solo los buenos programadores escriben código que los humanos puedan entender [2].

- **Ejemplo de código sucio vs código limpio**
  Un ejemplo corto sería en los nombres de variables. Estos deben ser comprensibles y lo que diferencia un código sucio de uno limpio. No es lo mismo que una variable entera se llame "**d**" en comparación con el nombre "**tiempoTranscurridoEnDías**" [1].

- **Principios clave:**
  - **Reglas generales de diseño:** Usar separación de responsabilidades, alta cohesión y bajo acoplamiento, encapsulación, composición sobre herencia y la ley de Deméter (un objeto debe comunicarse solo con colaboradores inmediatos para evitar cadenas largas de sub-llamadas de getters) [3].

  - **Comprensibilidad y nombres:** Siempre usar nombres que otros desarrolladores puedan comprender [1]. Además, se deben usar los estándares de escritura de cada lenguaje de programación [2].

- **Funciones y comentarios:** Escribir funciones cortas que solo realicen una sola tarea. Por otro lado, no se deben comentar cosas obvias. Se debe realizar comentarios en partes complejas del código y que realmente aporten valor de entendimiento [2].

- **Organización de código, objetos y estructuras:** Clases y métodos son compactados a una sola tarea clara en la medida de lo posible [1].

- **Pruebas y olores de código:** El código implementa pruebas unitarias [1].

2. **Análisis de archivos**
   a. **binary_search.py**
   Enlace del archivo:
   https://github.com/keon/algorithms/blob/main/algorithms/search/binary_search.py

   Análisis del código

   Nombres: En general, los nombres son entendibles, pero existen algunas inconsistencias.

   Funciones: Ambas funciones (binary_search() y binary_search_recur()) cumplen con el principio de responsabilidad única. Cada una implementa el mismo algoritmo mediante un enfoque diferente (iterativo y recursivo), sin incluir tareas adicionales que las sobrecarguen.

   Comentarios: El archivo contiene una mezcla de comentarios útiles y otros redundantes.

   Estructura y organización: El código está ordenado de manera lógica, con la versión iterativa antes de la recursiva. No obstante, la gran cantidad de comentarios iniciales sobre teoría interrumpe la lectura fluida del código.

   Olores de código: Se identifican algunos olores leves. El primero es la duplicación de lógica entre las dos funciones, puesto que ambas repiten la misma operación con pequeñas variaciones. Además, el uso de nombres poco descriptivos como “val” o abreviaciones como “recur” puede considerarse un olor de tipo “nombres ambiguos”.

   Propuestas de mejora

Mejora 1: Separar la parte teórica del código. La explicación sobre la recurrencia y la aplicación del Teorema Maestro puede trasladarse al README o a un archivo de documentación, de manera que el código fuente quede más limpio y centrado en su implementación.

Mejora 2: Remover los comentarios que explican la complejidad algorítmica de la búsqueda porque no explican nada significativo del código, solo teoría que se podría leer en un libro o fuente externa.

Mejora 3: Estandarizar los valores de retorno. En la versión iterativa de la función se retorna None cuando no se encuentra el elemento, mientras que en la versión recursiva se retorna -1. Es recomendable unificar ambos métodos para que devuelvan siempre el mismo tipo de valor.

Mejora 4: Mejorar nombres de varios elementos como de la función binary_search_recur() porque es una abreviación innecesaria que resulta menos clara. Además, en este mismo método, uno de los nombres es “val”, siendo un término genérico que no explica por sí mismo su propósito.

Mejora 5: Eliminar comentarios redundantes que explican acciones obvias del código, como “#Go search in the left subarray” o “#This mid will not break integer range”. Un código limpio debe ser lo suficientemente claro como para no necesitar este tipo de explicaciones.

```python
"""
Binary Search

Find an element in a sorted array (in ascending order).
"""

# For Binary Search, T(N) = T(N/2) + O(1) // the recurrence relation
# Apply Masters Theorem for computing Run time complexity of recurrence relations:
#       T(N) = aT(N/b) + f(N)
# Here
#       a = 1, b = 2 => log (a base b) = 1
# also, here
#       f(N) = n^c log^k(n)  // k = 0 & c = log (a base b)
# So,
#       T(N) = O(N^c log^(k+1)N) = O(log(N))

def binary_search(array, query):
    """
    Worst-case Complexity: O(log(n))

    reference: https://en.wikipedia.org/wiki/Binary_search_algorithm
    """

    low, high = 0, len(array) - 1
    while low <= high:
        mid = (high + low) // 2
        val = array[mid]
        if val == query:
            return mid

        if val < query:
            low = mid + 1
        else:
            high = mid - 1
    return None

#In this below function we are passing array, it's first index , last index and value to be searched
def binary_search_recur(array, low, high, val):

    Worst-case Complexity: O(log(n))

    reference: https://en.wikipedia.org/wiki/Binary_search_algorithm
    """
#Here in Logic section first we are checking if low is greater than high which means its an error condition because low index should not move ahead of high index
    if low > high:
        return -1
    mid = low + ((high-low)//2  #This mid will not break integer range
    if val < array[mid]:
        return binary_search_recur(array, low, mid - 1, val) #Go search in the left subarray
    if val > array[mid]:
        return binary_search_recur(array, mid + 1, high, val) #Go search in the right subarray
    return mid
```

Imagen 1 Código de archivo binary_search.py.

b. dijkstra.py

Enlace del archivo:
https://github.com/keon/algorithms/blob/main/algorithms/graph/dijkstra.py

Análisis del código

Nombres: En general, los nombres de variables y funciones son comprensibles, pero podrían ser más expresivos. Por ejemplo, el nombre min_dist_set() puede resultar confuso para quienes no estén familiarizados con la implementación, dado que en realidad representa el conjunto de vértices ya procesados.

Funciones: En términos generales, las funciones son concisas y cumplen su propósito. Sin embargo, dentro de dijkstra() se realizan varias tareas en una sola función (selección del nodo, actualización de distancias, marcado de vértices),

lo que podría considerarse un incumplimiento del principio de responsabilidad única.

Comentarios: Los comentarios presentes son útiles para entender las acciones principales

Estructura y organización: La estructura general es correcta, el código está encapsulado dentro de una clase (Dijkstra), lo que favorece la reutilización y la modularidad. Sin embargo, sería una buena práctica separar la creación del grafo de la lógica del algoritmo.

Olores de código: El método dijkstra() concentra demasiadas responsabilidades, lo que lo hace más difícil de probar y mantener. También hay variables poco descriptivas (target, source).

Propuestas de mejora

Mejora 1: Renombrar la variable "min_dist_set" por un nombre más claro como "visited" o "processed_vertices", puesto que describe de manera más precisa su función dentro del algoritmo.

Mejora 2: Reemplazar "source" y "target" por "current_vertex""y "neighbor" para mejorar la comprensión del flujo de iteraciones.

Mejora 3: Dividir el método dijkstra() en subfunciones más pequeñas. Por ejemplo, crear métodos auxiliares para actualizar las distancias o para verificar si un vértice ya ha sido procesado. Esto haría el código más fácil de mantener y cumpliría mejor con el principio de responsabilidad única.

Mejora 4: Agregar un método add_edge(self, u, v, weight) para construir el grafo de forma más intuitiva y evitar manipular directamente la matriz de adyacencia. Esto haría que el código fuera más claro y fácil de extender a representaciones alternativas, como listas de adyacencia.

Mejora 5: Inicializar la variable "min_index" dentro del método min_distance() antes del bucle, para evitar posibles errores en casos donde no se encuentre un vértice con distancia mínima válida.

```python
class Dijkstra():
    """
    A fully connected directed graph with edge weights
    """

    def __init__(self, vertex_count):
        self.vertex_count = vertex_count
        self.graph = [[0 for _ in range(vertex_count)] for _ in range(vertex_count)]

    def min_distance(self, dist, min_dist_set):
        """
        Find the vertex that is closest to the visited set
        """
        min_dist = float("inf")
        for target in range(self.vertex_count):
            if min_dist_set[target]:
                continue
            if dist[target] < min_dist:
                min_dist = dist[target]
                min_index = target
        return min_index

    def dijkstra(self, src):
        """
        Given a node, returns the shortest distance to every other node
        """
        dist = [float("inf")] * self.vertex_count
        dist[src] = 0
        min_dist_set = [False] * self.vertex_count

        for _ in range(self.vertex_count):
            #minimum distance vertex that is not processed
            source = self.min_distance(dist, min_dist_set)

            #put minimum distance vertex in shortest tree
            min_dist_set[source] = True

            #Update dist value of the adjacent vertices
            for target in range(self.vertex_count):
                if self.graph[source][target] <= 0 or min_dist_set[target]:
                    continue
                if dist[target] > dist[source] + self.graph[source][target]:
                    dist[target] = dist[source] + self.graph[source][target]

        return dist
```

Imagen 2 Código de archivo dijkstra.py.

3. Conclusiones

- El análisis de los archivos permitió identificar varios aspectos importantes sobre la aplicación de los principios de Código Limpio. En ambos casos, el código cumple con

una estructura funcional y legible, pero presenta oportunidades de mejora en la claridad de los nombres, la organización interna de las funciones y la utilidad de los comentarios.

- Se evidenció que una buena práctica consiste en mantener funciones breves y con una sola responsabilidad, así como utilizar nombres descriptivos que comuniquen con precisión la finalidad de cada elemento. Además, mediante la revisión se comprendió que los comentarios deben aportar información relevante y no limitarse a repetir lo que el código ya expresa por sí mismo.

- Finalmente, se concluye que aplicar estos principios no solo mejora la comprensión del programa, sino que también facilita su mantenimiento, extensión y reutilización en futuros desarrollos.

4. Referencias

[1] IONOS, «IONOS,» 12 Febrero 2022. [En línea]. Disponible en: https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/clean-code-que-es-el-codigo-limpio/. [Último acceso: 7 Noviembre 2025].

[2] Codacy, «Codacy,» 19 Diciembre 2023. [En línea]. Disponible en: https://blog.codacy.com/what-is-clean-code. [Último acceso: 7 Noviembre 2025].

[3] Paradigma Digital, «Paradigma Digital,» 23 Marzo 2023. [En línea]. Disponible en: https://www.paradigmadigital.com/dev/5-reglas-diseno-software-simple/. [Último acceso: 7 Noviembre 2025].

[4] ISO, «iso2500,» [En línea]. Disponible en: https://iso25000.com/index.php/normas-iso-25000. [Último acceso: 2 Octubre 2025].

[5] ISO, «iso25000,» [En línea]. Disponible en: https://iso25000.com/index.php/normas-iso-25000/iso-25010. [Último acceso: 2 Octubre 2025].