Enfoque de Arriba Abajo (Top-Down) para Resolver Problemas en Python

El **enfoque de arriba abajo** es una técnica de diseño de programas que comienza con una vista de alto nivel del problema y lo descompone en **módulos, funciones o subtareas más pequeñas** y manejables. Es una excelente práctica para escribir código **limpio, legible, reutilizable y fácil de depurar**.

La idea central es empezar definiendo la **función principal** del programa (`main`) que orquesta las llamadas a otras funciones de nivel inferior que se encargan de tareas específicas.

Código Ejemplo con Enfoque de Arriba Abajo

Este ejemplo simula un programa para calcular el área y el perímetro de un rectángulo.

En este enfoque, se inicia pensando en qué subtareas se ocupan para resolver el problema principal. Por ejemplo, puede que usted piense en dividir el problema en estas subtareas:

1. Obtener datos
2. Realizar cálculos
3. Mostrar la salida

Cada una de estas tareas se puede traducir en una o más funciones. Por ello, se empieza definiendo la función `main`, que es en donde inciaría el programa. En este punto, se llama a funciones **aún no están definidas**, que representan esas tareas planteadas anteriormente, pero que se sabe qué es lo que tienen que hacer:

```
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """

    print("Programa para calcular el área y perímetro de un
    ↪  rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
    ↪  inferior)
    area = calcular_area(largo, ancho)
```

```
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
    mostrar_resultados(area, perimetro)
```

La lógica de hacer esto es que, si ya existieran las funciones `obtener_dimensiones()`, `calcular_area(largo, ancho)`, `calcular_area(largo, ancho)` y `mostrar_resultados(area, perimetro)`, ya el problema estaría resuelto. Estas funciones son diseñadas por usted, y surgen como parte del análisis de las tareas necesarias para resolver el problema.

En realidad, se está aplicando **abstracción**, pues se ignora cómo estas funciones hacen algo, y por ahora solo interesa lo que hacen. Es **hasta que se termina de escribir** la función `main()` que se empiezan a definir el código para las funciones `obtener_dimensiones()`, `calcular_area(largo, ancho)`, `calcular_area(largo, ancho)` y `mostrar_resultados(area, perimetro)`.

Así, como ya se terminó `main()`, se procede a encontrar la primera función pendiente de implementar, que es `obtener_dimensiones()`. Por tanto, se procede a definirla abajo de la función `main()`, teniendo en cuenta que al llamarla se está asumiendo el retorno de una variable largo y otra ancho en la función `obtener_dimensiones()`. Así, se debe diseñar la función teniendo en cuenta que debe devolver estas dos variables:

```
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """

    print("Programa para calcular el área y perímetro de un
    ↪  rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
    ↪  inferior)
    area = calcular_area(largo, ancho)
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
    mostrar_resultados(area, perimetro)

def obtener_dimensiones() -> tuple:
    """
    Pide al usuario las dimensiones del rectángulo.
```

```
    Nótese que retorna una tupla, con largo y ancho.
    """
    terminado = False
    ancho = None
    largo = None
    while not terminado:
        try:
            largo = float(input("Introduzca el largo del
↪   rectángulo: "))
            ancho = float(input("Introduzca el ancho del
↪   rectángulo: "))
            if largo > 0 and ancho > 0:
                terminado = True
            else:
                print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada no válida. Por favor, introduzca un
             ↪  número.")

    return largo, ancho
```

Bien, ahora que se definió `obtener_dimensiones()` y devuelve las variables esperadas en `main()`, se procede con la siguiente función pendiente por implementar: `calcular_area(largo, ancho)`. Nótese cómo esta recibe largo y ancho como argumentos y devuelve el área calculada, pues en `main()` se atrapa esa variable al llamar a la función.

```
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """

    print("Programa para calcular el área y perímetro de un
     ↪  rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
     ↪  inferior)
    area = calcular_area(largo, ancho)
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
```

```
    mostrar_resultados(area, perimetro)

# Funciones de bajo nivel (tareas específicas)
def obtener_dimensiones() -> tuple:
    """Pide al usuario las dimensiones del rectángulo.

    Returns:
        tuple: Una tupla conteniendo el largo y ancho
    """
    terminado = False
    ancho = None
    largo = None
    while not terminado:
        try:
            largo = float(input("Introduzca el largo del
↪  rectángulo: "))
            ancho = float(input("Introduzca el ancho del
↪  rectángulo: "))
            if largo > 0 and ancho > 0:
                terminado = True
            else:
                print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada no válida. Por favor, introduzca un
             ↪  número.")

    return largo, ancho

def calcular_area(largo: float, ancho: float) -> float:
    """
    Calcula el área del rectángulo.
    """
    return largo * ancho
```

Genial, se ha implementado `calcular_area(largo, ancho)`, y devuelve el área correctamente. La siguiente función pendiente en `main()` es `calcular_perimetro(largo, ancho)`. Se procede a hacer un análisis similar, de los parámetros requeridos y lo que debe devolver:

```
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """
```

```
    print("Programa para calcular el área y perímetro de un
    ↪  rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
    ↪  inferior)
    area = calcular_area(largo, ancho)
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
    mostrar_resultados(area, perimetro)

# Funciones de bajo nivel (tareas específicas)
def obtener_dimensiones() -> tuple:
    """Pide al usuario las dimensiones del rectángulo.

    Returns:
        tuple: Una tupla conteniendo el largo y ancho
    """
    terminado = False
    ancho = None
    largo = None
    while not terminado:
        try:
            largo = float(input("Introduzca el largo del
↪  rectángulo: "))
            ancho = float(input("Introduzca el ancho del
↪  rectángulo: "))
            if largo > 0 and ancho > 0:
                terminado = True
            else:
                print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada no válida. Por favor, introduzca un
            ↪  número.")

    return largo, ancho

def calcular_area(largo: float, ancho: float) -> float:
    """
    Calcula el área del rectángulo.
    """
    return largo * ancho
```

```python
def calcular_perimetro(largo: float, ancho: float) -> float:
    """
    Calcula el perímetro del rectángulo.
    """
    return 2 * (largo + ancho)
```

Finalmente, solo queda la función `mostrar_resultados(area, perimetro)` pendiente de implementar:

```python
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """

    print("Programa para calcular el área y perímetro de un
    ↪  rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
    ↪  inferior)
    area = calcular_area(largo, ancho)
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
    mostrar_resultados(area, perimetro)

# Funciones de bajo nivel (tareas específicas)
def obtener_dimensiones() -> tuple:
    """Pide al usuario las dimensiones del rectángulo.

    Returns:
        tuple: Una tupla conteniendo el largo y ancho
    """
    terminado = False
    ancho = None
    largo = None
    while not terminado:
        try:
            largo = float(input("Introduzca el largo del
↪  rectángulo: "))
            ancho = float(input("Introduzca el ancho del
↪  rectángulo: "))
            if largo > 0 and ancho > 0:
```

```
                terminado = True
            else:
                print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada no válida. Por favor, introduzca un
            ↪ número.")

    return largo, ancho

def calcular_area(largo: float, ancho: float) -> float:
    """
    Calcula el área del rectángulo.
    """
    return largo * ancho

def calcular_perimetro(largo: float, ancho: float) -> float:
    """
    Calcula el perímetro del rectángulo.
    """
    return 2 * (largo + ancho)

def mostrar_resultados(area: float, perimetro: float) -> float:
    """
    Imprime el área y el perímetro calculados."""
    print(f"\n--- Resultados ---")
    print(f"Área: {area:.2f}")
    print(f"Perímetro: {perimetro:.2f}")
```

Observe cómo la función `main` coordina el flujo. Por ello, hace falta hacer un llamado a `main()` al final de todas las definiciones para que se empiece a ejecutar el código:

```
# Función principal (alto nivel)
def main():
    """Función principal que ejecuta la lógica del programa.

    La idea es dividir el problema en tareas más pequeñas.
    """

    print("Programa para calcular el área y perímetro de un
    ↪ rectángulo.")

    # 1. Obtener datos (llamada a función de nivel inferior)
    largo, ancho = obtener_dimensiones()

    # 2. Realizar cálculos (llamadas a funciones de nivel
    ↪ inferior)
```

```
    area = calcular_area(largo, ancho)
    perimetro = calcular_perimetro(largo, ancho)

    # 3. Mostrar la salida (llamada a función de nivel inferior)
    mostrar_resultados(area, perimetro)

# Funciones de bajo nivel (tareas específicas)
def obtener_dimensiones() -> tuple:
    """Pide al usuario las dimensiones del rectángulo.

    Returns:
        tuple: Una tupla conteniendo el largo y ancho
    """
    terminado = False
    ancho = None
    largo = None
    while not terminado:
        try:
            largo = float(input("Introduzca el largo del
↪  rectángulo: "))
            ancho = float(input("Introduzca el ancho del
↪  rectángulo: "))
            if largo > 0 and ancho > 0:
                terminado = True
            else:
                print("Las dimensiones deben ser positivas.")
        except ValueError:
            print("Entrada no válida. Por favor, introduzca un
             ↪  número.")

    return largo, ancho

def calcular_area(largo: float, ancho: float) -> float:
    """
    Calcula el área del rectángulo.
    """
    return largo * ancho

def calcular_perimetro(largo: float, ancho: float) -> float:
    """
    Calcula el perímetro del rectángulo.
    """
    return 2 * (largo + ancho)

def mostrar_resultados(area: float, perimetro: float):
    """
```

```
    Imprime el área y el perímetro calculados."""
    print(f"\n--- Resultados ---")
    print(f"Área: {area:.2f}")
    print(f"Perímetro: {perimetro:.2f}")

# Llamado a la función principal
main()
```

Es así que se pasó de un problema grande a resolver problemas cada vez más pequeños. De hecho, las funciones auxiliares de `obtener_dimensiones()`, `calcular_area(largo, ancho)`, `calcular_area(largo, ancho)` y `mostrar_resultados(area, perimetro)` podrían aplicar la misma lógica de separación en tareas más pequeñas. Por ejemplo, la función `mostrar_resultados(area, perimetro)` se puede dividir así:

```
def mostrar_resultados(area: float, perimetro: float):
    """
    Imprime el área y el perímetro calculados."""

    print(f"\n--- Resultados ---")
    # Mostrar el resultado del área
    mostrar_area(area)

    # Mostrar el resultado del perímetro
    mostrar_perimetro(perimetro)

def mostrar_area(area: float):
    """
    Muestra el área
    """
    print(f"El área es: {area}")

def mostrar_perimetro(perimetro: float):
    """
    Muestra el perímetro
    """
    print(f"El perímetro es: {perímetro}")
```

La decisión de si dividir una función en otras funciones o no depende de la complejidad del problema. Por ejemplo, para cosas tan sencillas como `print`, puede que no haga falta definir una función, pero para tareas más grandes o complejas (que ocupen ser aisladas para entenderlas más fácil) es mejor hacer una función.

Esta estrategia ayuda a despreocuparse de los detalles de cómo implementar algo, para ir resolviendo el problema por partes.

Ventajas del Enfoque

- Claridad: El flujo del programa es obvio al leer main.
- Modularidad: Cada función es una pieza independiente que hace una sola cosa.
- Reutilización: Las funciones como calcular_area pueden ser usadas en otras partes del código o proyectos.
- Fácil Depuración: Si hay un error, se sabe exactamente en qué función buscar (ej. si el cálculo del área es incorrecto, hay que mirar `calcular_area`).

Código Ejemplo SIN Enfoque de Arriba Abajo

Este ejemplo logra el mismo resultado, pero coloca toda la lógica directamente en el flujo principal del script sin usar funciones auxiliares o una función main claramente definida.

Todo el código está mezclado en el flujo principal...

```
print("Programa para calcular el área y perímetro de un
↪  rectángulo.")
terminado = False

# Obtener datos y validación
while not terminado:
    try:
        largo = float(input("Introduzca el largo del rectángulo:
↪  "))
        ancho = float(input("Introduzca el ancho del rectángulo:
↪  "))
        if largo > 0 and ancho > 0:
            terminado = True
        else:
            print("Las dimensiones deben ser positivas.")
    except ValueError:
        print("Entrada no válida. Por favor, introduzca un
        ↪  número.")

# Realizar cálculos
area = largo * ancho
perimetro = 2 * (largo + ancho)

# Mostrar la salida
print(f"\n--- Resultados ---")
print(f"Área: {area:.2f}")
print(f"Perímetro: {perimetro:.2f}")
```

En este caso, se está implementando todo el código seguido, y pensar en esta solución completa puede ser complejo para problemas muy grandes. Es mejor

pensar en cómo dividir un problema en tareas más pequeñas y luego preocuparse por cómo resolver esas tareas más pequeñas.

Aunque este código sin funciones se ve más corto, puede que no sea viable hacerlo en contextos que ocupan de soluciones muy complejas. Se hace confuso pensar y leer el código al no estar dividido en unidades lógicas.

Desventajas del Código Sin Estructura

- Dificultad de Lectura: El código se vuelve una "pared de texto" difícil de seguir a medida que crece.
- Acoplamiento: La lógica de entrada, cálculo y salida están fuertamente mezcladas.
- Poca Reutilización: No se ppuede usar la lógica de cálculo en otro lugar sin copiar y pegar el código.
- Mantenimiento Complicado: Un cambio en la forma de entrada podría requerir modificar partes lejanas de la lógica de cálculo o presentación.