PROGRAMACIÓN ESTRUCTURADA: Un enfoque claro y organizado

Melany Tirado Organista - mtirado@unal.edu.co
Juan Diego Dimas Correa - jdimasc@unal.edu.co
Edwin Esneider Orrego Zapata - eorrego@unal.edu.co
Santiago Ortiz Alfonso - saortiza@unal.edu.co

Visión general

Introducción

Aspectos teóricos de la programación estructurada

Aplicaciones y ejemplos

Introducción

Paradigmas de la programación

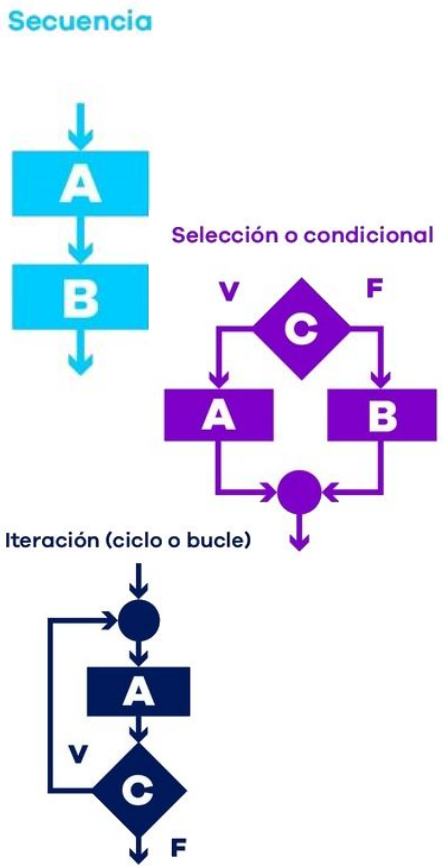
¿Qué es un paradigma?

Es un conjunto de reglas y herramientas para construir un programa de una manera específica.

¿Para qué sirven?

- Dan organización al código.
- Generan eficiencia.
- Algunos fragmentos pueden ser reutilizables

Programación estructurada



Programación lógica y secuencial

Se enfoca en mejorar la claridad, calidad y tiempo de desarrollo de un programa.

Estructuras de control

- Secuencia: La ejecución de las instrucciones es ordenada.
- Selección: Uso de condicionales.
- Iteración: Uso de iteraciones.

Comparación con otros paradigmas

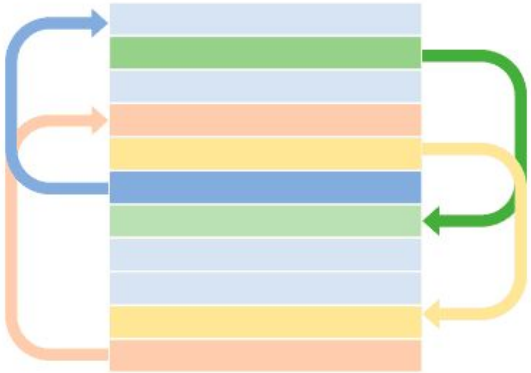
Característica	Programación Estructurada	Orientada a Objetos	Funcional	Lógica
Ventajas	Fácil de entender. Base para otros paradigmas	Reutilización de código. Permite desarrollar gráficos.	Código conciso, menos errores,.	Razonamiento lógico, resolución de problemas complejos
Desventajas	Limitada para problemas complejos. Difícil de escalar	Complejidad en diseños grandes.	No siempre es intuitiva, requiere un cambio de razonamiento.	Poco eficiente para tareas numéricas.

Aspectos teóricos

¿Por qué?

Lo propuso Edsger Dijkstra en los años con el fin de dar solución a la sentencia (GOTO) en la programación de entonces.

GOTO = IR A

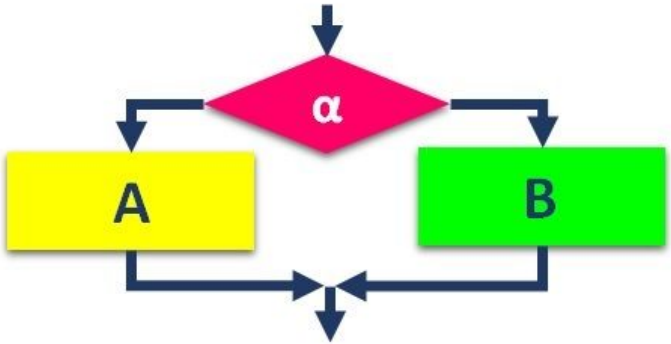


Códigos difíciles de entender y de difícil mantenimiento

Fundamentalmente, el programa se divide…

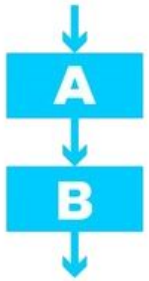
La solución al programa se debe pensar como un conjunto de soluciones más pequeñas o módulos: A, B, C …

Estas deben ser jerarquizadas, pequeñas, sencillas y legibles



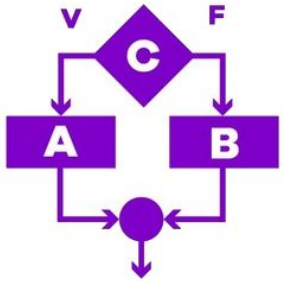
Es en estos módulos es donde se ven los aspectos de secuencia, selección e iteración…

Secuencia



Se pueden pensar como: “Primero se ejecutó el módulo A y luego el B”

Selección o condicional



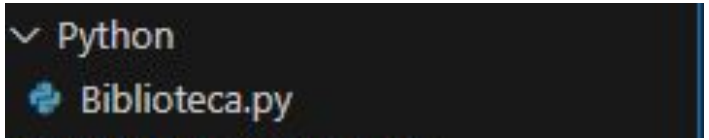
Se pueden pensar como: “Dependiendo de la respuesta del módulo C, se escoge entre ejecutar el A o el B”

Iteración (ciclo o bucle)

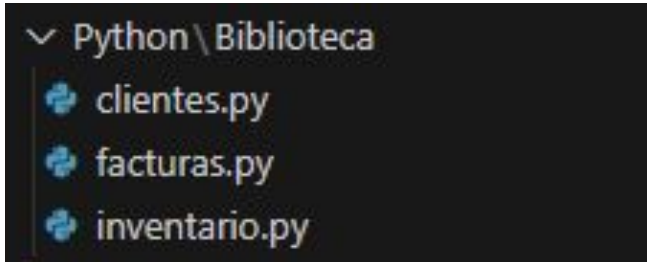
Se pueden pensar como: “Se ejecuta A y luego C, dependiendo de la salida de C, se vuelve a ejecutar A”

Se les puede llamar módulos, pero ¡No confundir con programación modular!

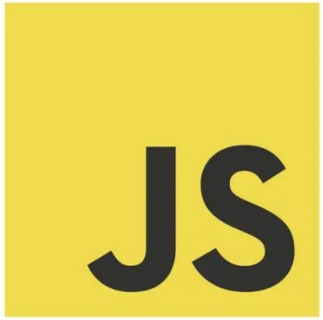
La programación estructurada se basa en la división de las soluciones o de los módulos por medio de estructuras de flujos de control y en una organización lógica del programa.



En la metodología de programación modular literalmente se separan los módulos.



Lenguajes de programación



Por la versatilidad y la fundamentación en el uso de estructuras de control, las posibilidades en lenguajes de programación asociados es amplia.

¡SÍ!	PERO.....
Claridad y simplicidad	Escalabilidad limitada
Facilita la detección de problemas y su solución	Poca abstracción del mundo real
Reduce el tiempo de ejecución	Rigidez en el manejo de los datos

Parte Práctica

```python
numero = input("Ingresa un número: ")#(1. Mostrar un mensaje que pida al usuario un número)
while numero.isdigit() == False:
    numero = input("Por favor, ingresa un número valido: ") #(1.2 Si se ingresa un valor no esperado pida al usuario que agregue un valor válido)

numero = int(numero)
resultado = numero * 2 #(2. Tome el número del usuario y multiplicarlo por 2)

#(2.1 Si el número es múltiplo de 5 Muestre un mensaje que diga "Número redondo". En caso contrario muestre un mensaje que diga "No es múltiplo d
if resultado % 5 == 0:
    print("Número redondo")
else:
    print("No es múltiplo de 5")

#(3. Ahora tome el resultado de la multiplicación y sumarle 273)
resultado_final = resultado + 273

if resultado_final > 400:
    print("Número grande") # (3.1 Si el número es mayor a 400 muestre un mensaje que diga "Número grande")
else:
    print("Un numero pequeño")#( 3.2 Si es menor a 400 muestre un mensaje que diga "numero pequeño")
```

numero = (“carácter”)

```
Ingresa un número: g
Por favor, ingresa un número valido:
```

numero =15

```
Ingresa un número: 15
Número redondo
Un numero pequeño
```

numero =400

```
Ingresa un número: 400
Número redondo
Número grande
```

```python
while continuar:
    valor = float(input("Ingrese el valor en pesos colombianos: ")) #(1. Tiene que recibir un valor en pesos colombianos)

    if valor < 100: #(1.1 Si el valor es menor a 100, ya que es un valor no válido en el peso colombiano muestre un mensaje que pida un valor may
        print("Valor no válido. Ingresa un valor mayor a 100.")
        continue

    if valor > 85000: #(2. Si el valor es superior a 85.000 le vamos a restar el 2.37% como promoción)
        descuento = valor * 0.0237
        valor -= descuento
    else:
        descuento = 0

    if valor > 92000: #(3. Sólo se sumará el iva si el valor supera los 92.000)
        iva = valor * 0.19
    else:
        iva = 0

    total = valor + iva

    print("Detalles de la operación:")
    print("Valor base: ",valor + descuento) #(4. Indique valor base)
    if iva > 0:
        print("Valor del IVA: ",iva) #(4.1 Indique el valor del iva - Solo si aplica)
    if descuento > 0:
        print("Valor del descuento por promoción:",descuento) #(4.2 Indique el valor del descuento por promocion - Solo si aplica)
    print("Total a pagar:",total) #(4.3 Indique el total)
```

valor=50

```
ombianos: 50
alor mayor a 100
ombianos:
```

valor=90000

```
Ingrese el valor en pesos colombianos: 90000
Detalles de la operación:
Valor base:  90000.0
Valor del descuento por promoción: 2133.0
Total a pagar: 87867.0
```

valor=100000

```
Ingrese el valor en pesos colombianos: 100000
Detalles de la operación:
Valor base:  100000.0
Valor del IVA:  18549.7
Valor del descuento por promoción: 2370.0
Total a pagar: 116179.7
```

```python
continuar = True

while continuar:

    temperatura = float(input("Ingrese la temperatura: ")) #(1. Pida al usuario un número que será la temperatura)
    unidad_inicial = input("Seleccione la unidad inicial (Celsius, Fahrenheit, Kelvin): ").lower() #(2. Le daremos 3 opciones para seleccionar
    #si es Celsius, fahrenheit y Kelvin )
    if unidad_inicial not in ['celsius', 'fahrenheit', 'kelvin']:
        print("Unidad inválida. Se tomará Celsius por defecto.") #(2.2 Si pone un valor no esperado, le daremos un valor por defecto que será Cel
        unidad_inicial = 'celsius'

    unidad_destino = input("Seleccione la unidad de destino (Celsius, Fahrenheit, Kelvin): ").lower()

    if unidad_destino not in ['celsius', 'fahrenheit', 'kelvin']:
        print("Unidad no válida. Se tomará Celsius por defecto.")
        unidad_destino = 'celsius'

    if unidad_inicial == 'celsius':
        if unidad_destino == 'fahrenheit':
            resultado = temperatura * 9/5 + 32
        elif unidad_destino == 'kelvin':            #(3 convertir a celsius)
            resultado = temperatura + 273.15
        else:
            resultado = temperatura
    elif unidad_inicial == 'fahrenheit':
        if unidad_destino == 'celsius':
            resultado = (temperatura - 32) * 5/9    #(3.1 Convertir fahrenheit)
        elif unidad_destino == 'kelvin':
            resultado = (temperatura - 32) * 5/9 + 273.15
        else:
            resultado = temperatura
    elif unidad_inicial == 'kelvin':
```

```python
if unidad_destino not in ['celsius', 'fahrenheit', 'kelvin']:
    print("Unidad no válida. Se tomará Celsius por defecto.")
    unidad_destino = 'celsius'

if unidad_inicial == 'celsius':
    if unidad_destino == 'fahrenheit':
        resultado = temperatura * 9/5 + 32
    elif unidad_destino == 'kelvin':            #(3 convertir a celsius)
        resultado = temperatura + 273.15
    else:
        resultado = temperatura
elif unidad_inicial == 'fahrenheit':
    if unidad_destino == 'celsius':
        resultado = (temperatura - 32) * 5/9    #(3.1 Convertir fahrenheit)
    elif unidad_destino == 'kelvin':
        resultado = (temperatura - 32) * 5/9 + 273.15
    else:
        resultado = temperatura
elif unidad_inicial == 'kelvin':
    if unidad_destino == 'celsius':
        resultado = temperatura - 273.15
    elif unidad_destino == 'fahrenheit':        #(3.2 Convertir a Kelvin)
        resultado = (temperatura - 273.15) * 9/5 + 32
    else:
        resultado = temperatura  # Kelvin a Kelvin


print("La temperatura en", unidad_destino.capitalize(), "es:", f"{resultado:.2f}")


continuar = False
```

Resultados:

```
Ingrese la temperatura: 50
Seleccione la unidad inicial (Celsius, Fahrenheit, Kelvin): Celsius
Seleccione la unidad de destino (Celsius, Fahrenheit, Kelvin): fahrenheit
La temperatura en Fahrenheit es: 122.00
```

```
Ingrese la temperatura: 50
Seleccione la unidad inicial (Celsius, Fahrenheit, Kelvin): fahrenheit
Seleccione la unidad de destino (Celsius, Fahrenheit, Kelvin): celsius
La temperatura en Celsius es: 10.00
```

```
e la temperatura: 50
ione la unidad inicial (Celsius, Fahrenheit, Kelvin): kelvin
ione la unidad de destino (Celsius, Fahrenheit, Kelvin): fahrenheit
peratura en Fahrenheit es: -369.67
```

Desafíos y consideraciones

Desafíos y consideraciones

Complejidad

Aumenta con el tamaño del código, lo que puede dificultar su comprensión y mantenimiento.

Reusabilidad

Menos modular que otros paradigmas, lo que limita la reutilización del código.

Manejo de errores

Requiere validaciones y manejo manual, lo que puede ser propenso a errores.

Desafíos y consideraciones

Rendimiento

Generalmente eficiente para problemas simples y lineales.

Escalabilidad

Puede ser difícil de escalar para problemas más complejos debido a la falta de modularidad.

Mantenibilidad

Puede volverse difícil de mantener a medida que el código se complejiza sin una estructura modular.

Comparación con otros paradigmas

La programación estructurada es efectiva para problemas simples y lineales, donde cada paso depende directamente del anterior y no se requiere una gran modularidad.

Aunque la programación orientada a objetos ofrece ventajas en términos de organización y escalabilidad, su uso en problemas simples puede ser innecesariamente complejo.

Por lo tanto, la elección del paradigma debe basarse en la naturaleza del problema y las necesidades específicas del proyecto.

Conclusiones

- La programación estructurada nace como un paradigma de la programación para dar solución a códigos confusos y de difícil mantenimiento. Se enfoca en crear módulos para que se encarguen de pequeñas partes de la solución general del programa en función de la secuencia, selección e iteración.

- Es importante tener en cuenta las limitaciones de la programación estructurada, crear en exceso ramificaciones o módulos, crea un programa difícil de leer. Así mismo, tener cuidado para programas muy grandes que con el tiempo sean difíciles de escalar o que requieran un mejor enfoque de abstracción.

Gracias por su atención