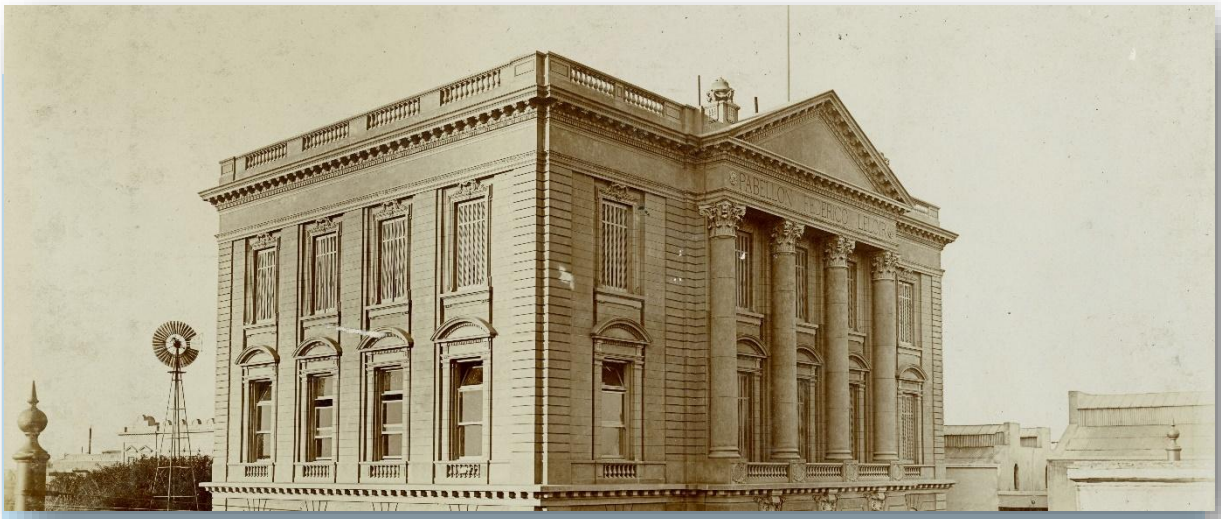
UTN

INGRESO 2025

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN A DISTANCIA

UNIVERSIDAD TECNOLÓGICA NACIONAL



CUADERNILLO 8

Método Iterativo

CURSO COMPLETO

UNIDAD I FUNDAMENTOS LOGICOMATEMÁTICOS

CUADERNILLO 1 – Teoría de conjuntos, números y sus tipos

CUADERNILLO 2 – Sistema Binario

CUADERNILLO 3 – Introducción a la lógica

CUADERNILLO 4 – Operaciones aritméticas

CUADERNILLO 5 – Números Enteros

CUADERNILLO 7 – Más de números

UNIDAD II RESOLUCIÓN DE PROBLEMAS

CUADERNILLO 6 – Análisis verbal

CUADERNILLO 8 – Método iterativo

CUADERNILLO 9 – Analogía y Patrones

CUADERNILLO 10 – Divide y conquistarás

CUADERNILLO 11 – Integración

CUADERNILLO 12 – Ensayo y Error

METODO ITERATIVO

¿CUAL ES LA IDEA?

¿QUE ES ITERACION?

Un método en el cual repetimos un proceso varias veces hasta cumplir una condición.

Muchos problemas complejos no se resuelven de golpe, sino mediante pasos repetitivos donde se va verificando el progreso. En la vida cotidiana solemos usar iteración.

Por ejemplo, pintar una pared...

Al principio no sé cuántas manos de pintura necesito, depende de la superficie, la calidad de la pintura, si uso pincel o rodillo, el acabado deseado...

Empezamos aplicando una capa de pintura (una iteración) y observamos si llegamos al acabado deseado, luego otra (segunda iteración) y de nuevo observamos, de ser necesario una tercera (otra más) hasta obtener por fin el resultado deseado (se cumplió la condición de parada).

CONDICION INICIAL

Partimos de un estado base, como un valor o una aproximación inicial. En nuestro ejemplo una pared sin pintar.

ITERACION

Aplicamos los mismos pasos como sumar, restar, aproximar, pintar una y otra vez.

CONDICION DE PARADA

Tras cada iteración, comprobamos si llegamos a la meta o necesitamos más repeticiones. ¿Está el acabado como quería?

O sea, si llegamos a una condición de parada.

GUIA METODO ITERATIVO

Aquí tienes una serie de consejos para plantear y resolver un problema mediante método iterativo en un software. Este enfoque funciona especialmente bien cuando no podés o no conviene hallar la solución de golpe, sino que preferís aproximarte paso a paso hasta cumplir cierto objetivo o condición.

CONSEJO 1: DEFINE CLARAMENTE EL PROBLEMA Y SUS VARIABLES

Asegurate de entender lo que querés lograr. Es útil usar el decálogo del análisis verbal de enunciados.

Preguntas clave: ¿Cuál es el objetivo final? ¿Qué valores o datos de entrada tengo?

Identificá las variables que van a cambiar (y cómo) en cada iteración.

Por ejemplo, si buscás el interés compuesto a lo largo de varios periodos, necesitamos saber cómo incrementa el interés.

CONSEJO 2: DETERMINA LA CONDICIÓN INICIAL

Elegí un punto de partida sensato para tu iteración.

Por ejemplo, si buscás el interés compuesto a lo largo de varios periodos, necesitamos saber el capital invertido.

La idea es tener un estado inicial desde donde arranque el proceso repetitivo.

CONSEJO 3: DISEÑA EL PASO ITERATIVO

Definí qué hace tu algoritmo en cada iteración.

Ejemplo: “Ajustar la aproximación restando o sumando un pequeño valor”, “Sumar un interés al capital cada año”, etc.

El paso iterativo debe mover tu sistema más cerca de la solución.

Si no, corrés riesgo de quedarte en un ciclo infinito o alejarte de la meta.

CONSEJO 4: ESTABLECE LA CONDICION DE PARADA

Necesitás definir cuándo frenar la iteración. Podés hacerlo de varias maneras:

1. Número máximo de iteraciones: repetís el proceso un número fijo de veces.
2. Umbral de error: “Cuando la diferencia entre mi aproximación y el valor real sea < 0.001...”
3. Meta alcanzada: “Hasta que el monto invertido sea ≥ $150...”.

Sin una buena condición de parada, podrías quedar iterando para siempre.

Por ejemplo, si buscás el interés compuesto a lo largo de varios periodos, necesitamos saber cuántos períodos vamos a calcular o hasta que capital final.

CONSEJO 5: REGISTRA EL PROGRESO

Para depurar y entender el proceso, conviene imprimir o guardar valores clave en cada iteración:

La aproximación actual, el error, el capital acumulado, etc.

Esto te ayuda a verificar que todo evoluciona en la dirección correcta. Si ves valores que saltan al infinito o se estancan, sabrás que hay un problema.

CONSEJO 6: TESTEA

Antes de implementarlo para casos grandes, testeá tu método iterativo con ejemplos controlados o valores pequeños:

Si sumás intereses, hacelo con números chicos para ver si llegás al resultado esperado.

Esto te permitirá refinar tu paso iterativo y tu condición de parada. Y sobre todo, SIN TESTEO CORRES EL PELIGRO DE CAER EN N BUBLE INFINITO.

CONSEJO 7: MANEJA LOS POSIBLES ERRORES O EXCEPCIONES

¿Qué pasa si el dato de entrada es 0 o negativo?

¿Qué pasa si la iteración no converge?

Pensá en esos casos antes de terminar. Puede que debas avisar al usuario o aplicar otra lógica.

CONSEJO 8: OPTMIZAR

Si el número de iteraciones es muy grande, fijate si podés mejorar tu paso iterativo (un método más rápido de converger).

A veces, métodos iterativos pueden sustituirse por fórmulas directas (aunque no siempre). Evaluá la relación entre exactitud y rendimiento (tiempo de cómputo, memoria usada, etc.).

CONSEJO 9: DOCUMENTAR

Dejá claro por qué usás iteración y cómo funciona:

1. Describí el estado inicial.
2. Explicá las transformaciones por paso.
3. Indicá la condición para detenerte.

Esto facilita la lectura del código y la comprensión por parte de otros (¡o vos mismo en el futuro!).

CONSEJO 10: REVISION FINAL

Revisa si el resultado final tiene sentido. Una vez que el software te da un resultado, chequeá:

- ¿Es lógico el valor obtenido?
- ¿Se condice con tus predicciones o casos de prueba?

Si hay discrepancias, revisá tu lógica de iteración y la condición de parada.

RESUMEN

El método iterativo es un enfoque paso a paso donde cada ciclo te acerca a la solución. Para aplicarlo en software:

1. Entendé el problema y fijá tus variables.
2. Definí un estado inicial razonable.
3. Diseñá un paso iterativo que mejore tu aproximación en cada ciclo.
4. Asegurate de tener una condición de parada (máximo de iteraciones o un umbral de error).
5. Testeá con ejemplos y revisá si todo avanza bien.

Siguiendo estos consejos, tu método iterativo será más confiable, claro y fácil de mantener. ¡Manos a la obra!

APLICACIÓNMETODO ITERATIVO

ENUNCIADO 1

“Calcular la suma de los primeros n números naturales de manera iterativa.”

ANALISIS

Define claramente el problema y sus variables.

Objetivo: obtener la suma S de 1 + 2 + ⋯ + n.

Entrada: un entero n.

Variables: contador (de 1 a n), suma (donde acumulamos el resultado).

A esto le llamamos definición de variables.

En este punto podemos determinar que necesitaremos solo enteros.

Condición inicial.

suma = 0

contador = 1

A esto lo llamaremos inicialización de variables.

Diseño del paso iterativo.

En cada iteración:

suma += contador

contador++.

Nota: suma += contador en muchos lenguajes indica que suma = suma + contador. O sea, toma el valor de suma, le adiciona contador, vuelve a guardarlo, en suma.

Nota: contador++ indica que al valor anterior le sumamos 1. No en todos los lenguajes se puede hacer, en C es la operación más rápida.

Condición de parada.

Sabemos que contador debe llegar a n.

Dos estructuras se usan en programación para repetir la ejecución de un fragmento de código:

REPETIR MIENTRAS o WHILE: Se usa cuando no sabemos cuántas veces será necesario que el código se ejecute. El fragmento de código se repetirá mientras no se cumpla una condición que llamamos de parada. La condición de parada es contador supera n.

REPETIR HASTA o FOR: Se usa cuando sabemos cuántas veces se repetirá el código. En este caso es lo más conveniente ya que sabemos exactamente cuantas veces se repetirá, n veces. Condición de parada es contador llega a n.

Registro del progreso.

Se podrían imprimir valores parciales.

Testeo.

Para n = 3: suma final = 6.

Errores o excepciones.

n = 0 → suma = 0.

n < 0 → caso no contemplado (no hay “números naturales” negativos).

Optimizar.

Existe la fórmula n *(n + 1) / 2, pero como queremos practicar bucles lo haremos de otra manera.

Documentar.

Explicar la lógica del bucle y sus variables.

Revisión final.

Verificamos con n=5 → 15. Perfecto.

ENUNCIADO 2

"Calcular cuántos años tardará un capital de $100 millones al 10% anual en llegar o superar $200 millones."

ANALISIS

Define claramente el problema y sus variables.

Objetivo: Hallar el número de años para que el capital supere/iguale 200 millones, partiendo de 100 millones.

Entradas:

capital = 100 millones

interés = 1.1. # Equivale a aplicar un 10%

meta = 200 millones de pesos.

Variables:

capital, interes, meta y años

En este punto podemos determinar que necesitaremos enteros y float.

Nota: En la enorme mayoría de lenguajes no hay problema de multiplicar un entero con un float. En ellos primero se promociona al tipo de mayor precisión, o sea el integer a float, luego se opera dando como resultado un float.

Como sucede esto, y todavía no terminamos el análisis, o posponemos la determinación del tipo de número o directamente usamos todos float.

Determina la condición inicial.

capital = 100 millones

anios = 0

interes = 1.1 #aumenta un 10%

Diseña el paso iterativo.

Cada iteración simula 1 año:

capital = capital * 1.1 (aumento del 10% del capital).

anios++.

Establece la condición de parada.

Mientras capital < 200000000, seguir iterando.

Al llegar o superar 200000000, se detiene.

Como en este caso no sabemos cuántas veces iteraremos entonces se hace necesario un REPETIR MIENTRAS o WHILE.

Registra el progreso.

Se puede imprimir: “Año X → capital = … ”.

Testea.

Ejemplo: si en vez de 200 millones pidiéramos 150 millones. Aproximadamente en 5 años se supera esa cifra cuando partimos de 100.

Maneja posibles errores.

interes = 0:

capital no crece, nunca llega a la meta → advertir. meta < capital: ya cumplido.

Optimizar.

Existe fórmula logarítmica, pero iteramos para aprender bucles.

Documentar

Escribimos: “Iniciamos con 100 millones. Cada ‘anio’ multiplicamos capital por 1.1, sumamos 1 a ‘anios’.

Detenemos al >= 200 millones.”

Revisión final

Verificar con un caso: ¿llega a 200 millones? Sí, tras cierto número de años. Lógico.

ENUNCIADO 3

“Ajustar la temperatura con un termostato en 5 pasos. Sube 1 grado si está por debajo de 20, baja 1 grado si está por encima de 25 y se mantiene igual si está entre 20 y 25 inclusive”

ANALISIS

Define claramente el problema y sus variables

Objetivo: hallar la temperatura final tras 5 “ajustes” manteniendo un rango con un máximo y un mínimo.

Entrada:

TActual

ajustes = 5

TMax = 25

TMin = 20.

Variables:

TActual

contador

Nos preguntamos, ¿Necesitamos fracción de grado? ¿Es float o integer? Todo indica que usaremos integer.

Condición inicial

contador = 0

T actual = valor inicial

Diseña el paso iterativo

Por cada ajuste (1 paso):

Si TActual < TMin → TActual sube 1 grado

Si TActual > TMax → TActual baja 1 grado

Si TMin ≤ Tactual ≤ TMax → TActual no cambia

En los tres casos contador++

Condición de parada

Número máximo de iteraciones: 5.

Lo que tenemos clarísimo es que son 5 iteraciones, usamos un REPETIR HASTA o FOR

Registra el progreso

Podríamos imprimir Tactual tras cada ajuste.

Testea

TActual = 18 → en 2 pasos llega a 20, luego se mantiene.

TActual = 27 → baja a 25 en 2 pasos, luego se queda en 25.

Por eso importante poner, aunque después no queramos hacerlo, un imprimir temperatura en cada iteración. Muchas veces al testear necesitamos poner sentencias transitorias que luego de estar seguro, las sacamos.

Errores o excepciones

Si TActual = 1000, en 5 pasos bajará a 995, no hay tiempo para estabilizar. Sin problema.

El enunciado no promete estabilizar TActual siempre, solo hacer 5 ajustes. Pero si prometer estabilizar es importante testear la T antes de iniciar, para evitar que se ejecute todo el código para valores que nunca se estabilizaran e informar que no lo hará.

Por ejemplo: si (TActual > 15) and (TActual < 31) ejecutar iteración, caso contrario informar y salir.

Optimizar

Son solo 5 pasos; no se necesita nada más.

Documentar

“Un bucle de 5 iteraciones, cada una ajusta TActual según su rango”

Revisión final

Concluye sin contradicciones, ejemplos comprobados.

CONCLUSION GENERAL

Problema 1: Suma 1 .. n

Ejemplo clásico de acumulación, número fijo de iteraciones. Usa un for.

Problema 2: Capital en millones

No sabemos cuántas iteraciones vamos a hacer. Usa un while.

Problema 3: Termostato

Ajustes fijos donde la temperatura sube, baja o se mantiene. Usa un for.

En todos, aplicamos la iteración con los 10 consejos: definir variables, estado inicial, paso iterativo, condición de parada, registro, testeo, manejo de excepciones, optimización, documentación y revisión final