**Escuela Politécnica Nacional**

**Ejercicios Unidad 01-B**

Nombre: Wellington Barros





Inicio

Definir función Redondear\_a\_Tres\_Dígitos(valor)

Si valor = 0 entonces

Retornar 0

Sino

Calcular redondeo a tres dígitos significativos

Retornar el valor redondeado

FinSi

FinFunción

Definir función Sumar\_Con\_Redondeo()

Definir suma = 0

Para i desde 1 hasta 10 hacer

Calcular término = 1 / (i^2)

término = Redondear\_a\_Tres\_Dígitos(término)

suma = suma + término

suma = Redondear\_a\_Tres\_Dígitos(suma)

FinPara

Retornar suma

FinFunción

Llamar a Sumar\_Con\_Redondeo()

Mostrar resultado de la suma

Fin

**Respuesta:** Este pseudocódigo muestra cómo sumar fracciones con corte a tres dígitos, primero de 1/1 a 1/100 y luego de 1/100 a 1/1. En cada paso, tanto el valor de cada fracción como la suma acumulada se redondean a tres dígitos. El método que comienza con los números pequeños (de 1/100 a 1/1) es más preciso porque los números grandes no "aplastan" los pequeños, mientras que al sumar primero los números grandes, los pequeños tienen menos impacto en el resultado final.

****

Inicializar una variable `suma1` en 0 para almacenar la suma de la primera serie (de 1/1 a 1/1000).

Inicializar una variable `suma2` en 0 para almacenar la suma de la segunda serie (de 1/1000 a 1/1).

Para i desde 1 hasta 10:

Calcular el valor de 1/i^3.

Redondear el resultado a tres dígitos (corte).

Sumar el resultado a `suma1`.

Mostrar el valor de `suma1`.

Para i desde 10 hasta 1:

Calcular el valor de 1/i^3.

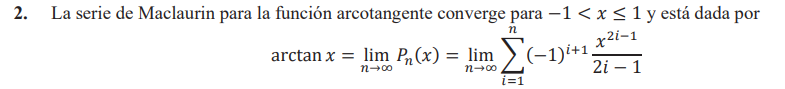
Redondear el resultado a tres dígitos (corte).

Sumar el resultado a `suma2`.

Mostrar el valor de `suma2`.

Comparar los valores de `suma1` y `suma2` y determinar cuál es más preciso.

**Respuesta:**  Calcule una suma con la fórmula, tratando de aproximar un valor relacionado con π. Luego, multiplica esa suma por 4. Resta el valor de π al resultado obtenido y toma el valor absoluto de esa diferencia (que es el error). Verifica si ese error es menor a , lo que garantiza que la aproximación es suficientemente precisa.

****



Sabemos que = 1, y por eso usaremos esta información en la serie de arctan para evaluar la precisión de nuestra aproximación.

El objetivo es sumar términos de la serie hasta que la diferencia entre y π sea menor que

Inicializar n como 1.

Inicializar suma como 0.

Inicializar error\_tolerable como

Inicializar error como un valor grande (por ejemplo, 1).

Mientras error > error\_tolerable:

1. Calcular el término actual: termino
2. Sumar termino a suma.
3. Calcular pi\_aprox como 4 x suma
4. Calcular error como Incrementar n en 1.

Mostrar el número de términos n.

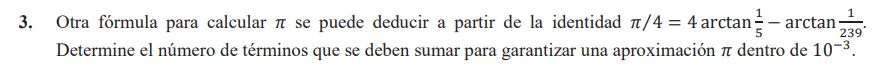
Mostrar el valor aproximado de π(pi\_aprox).

Mostrar el valor de error.

Respuesta: es el pseudocódigo directo y simple para resolver el problema que determina cuántos términos de la serie son necesarios para obtener una aproximación de π con un error menor a



Para obtener una precisión de usando la serie de Maclaurin para arctan(1) (que es equivalente a π/4), se necesitarían aproximadamente 5 millones de términos. Esto se debe a que la convergencia de la serie es bastante lenta, y se requiere sumar muchos términos para alcanzar un grado de precisión tan alto como .



1. Inicializar las variables:

pi\_aprox = 0

tolerancia = 10^(-3)

n = 1 (contador de términos)

2. Calcular el término actual usando la serie arctan(1/5) y arctan(1/239).

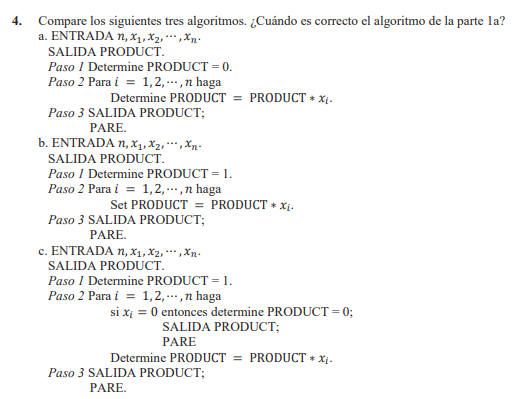
3. Mientras la diferencia entre el valor estimado de pi y el valor real de pi sea mayor que la tolerancia:

a. Sumar el siguiente término de la serie a pi\_aprox.

b. Incrementar n (el contador de términos).

1. Retornar el número de términos n.

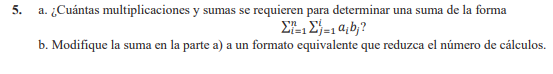
**Resultados:** El pseudocódigo primero define la estructura básica del problema, inicializando las variables necesarias para realizar los cálculos. Se basa en la serie infinita de Arc tangentes para calcular π y va sumando términos hasta que la diferencia entre el valor calculado y el valor real de π sea menor que la tolerancia de Utiliza un bucle que termina cuando se logra esta precisión, devolviendo el número de términos requeridos.



**Algoritmo a**: Es incorrecto en la mayoría de los casos, a menos que el resultado esperado sea 0 desde el principio.

**Algoritmo b**: Es correcto en todos los casos y es el enfoque estándar para calcular productos.

**Algoritmo c**: Es correcto y optimizado para conjuntos de datos donde puede haber ceros, ya que ahorra tiempo de cómputo al detenerse temprano si encuentra un cero.



1. Inicializar PRODUCTO\_TOTAL = 0

2. Para i desde 1 hasta n:

a. Inicializar SUMA\_PARCIAL = 0

b. Para j desde 1 hasta m:

- Calcular SUMA\_PARCIAL = SUMA\_PARCIAL + ai \* bj

c. Sumar SUMA\_PARCIAL a PRODUCTO\_TOTAL

3. Salida PRODUCTO\_TOTAL

**Modificación para la parte b:**

1. Inicializar PRODUCTO\_TOTAL = 0

2. Para j desde 1 hasta m:

a. Inicializar FACTOR\_COMUN = 0

b. Para i desde 1 hasta n:

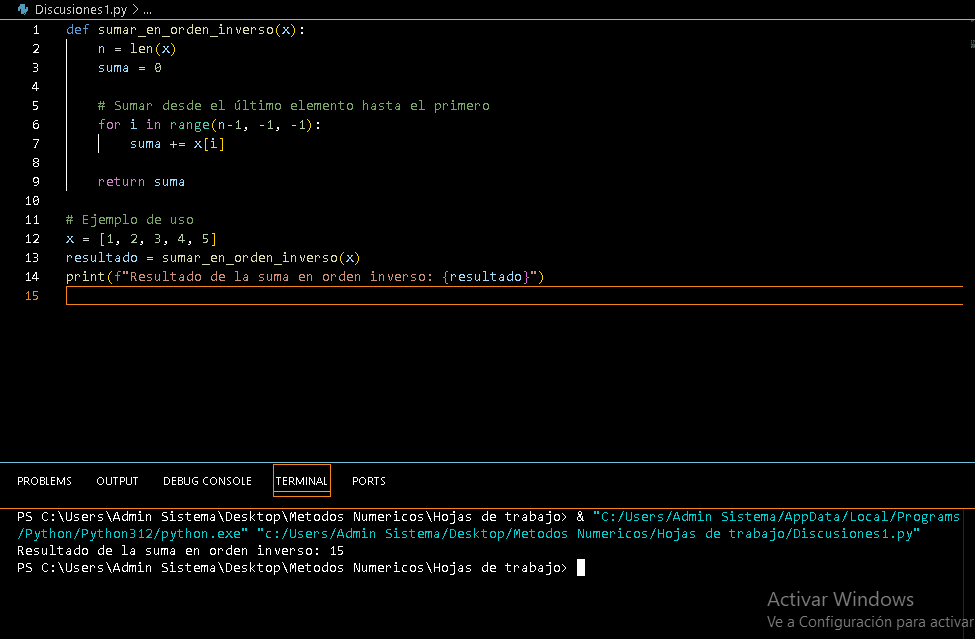
- Calcular FACTOR\_COMUN = FACTOR\_COMUN + ai

c. Sumar PRODUCTO\_TOTAL = PRODUCTO\_TOTAL + FACTOR\_COMUN \* bj

3. Salida PRODUCTO\_TOTAL

**Resultados:** En la parte a, el algoritmo realiza una suma doblemente anidada, lo que requiere n×m multiplicaciones y sumas. En la parte b, se optimiza reduciendo la cantidad de cálculos al sumar los valores de a​ una sola vez antes de realizar las multiplicaciones, lo que disminuye los cálculos necesarios de n×m a m, reduciendo la complejidad computacional.





**Pseudocodigo:**

1. Inicializar SUMA = 0

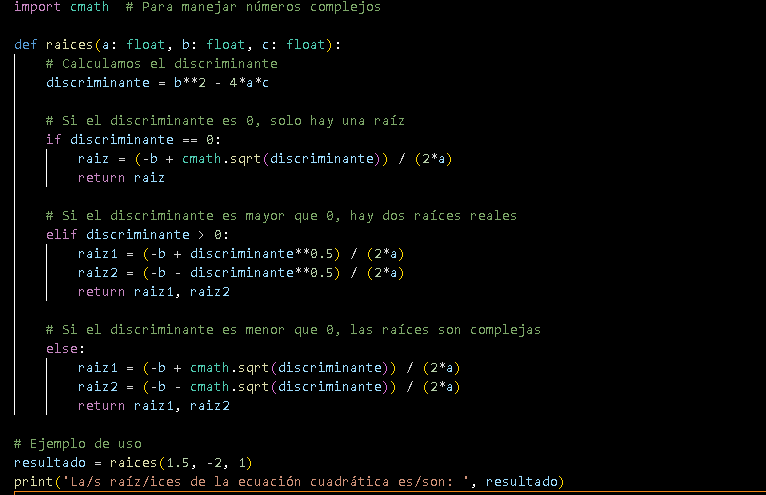
2. Para i desde n hasta 1 (en orden inverso):

a. Sumar xi a SUMA

3. Salida SUMA

**Resultados:** Este algoritmo recorre la lista xi en orden inverso, empezando desde el último elemento hasta el primero, y suma cada elemento a una variable acumuladora llamada SUMA. Finalmente, devuelve la suma total.

****

****

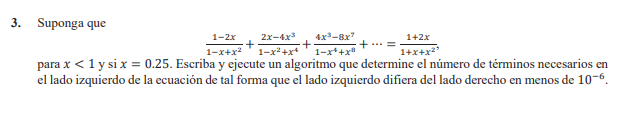
**Imagen que contiene Logotipo

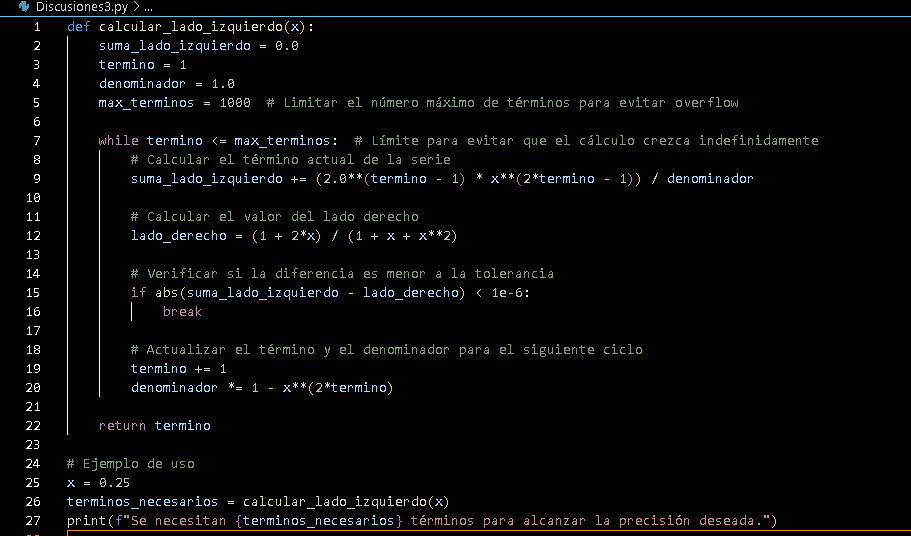
Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

Este código resuelve una ecuación cuadrática de la forma determinando las raíces (soluciones) en función del discriminante, que se calcula como . Dependiendo del valor del discriminante, el código tiene tres comportamientos: si el discriminante es 0, devuelve una única raíz real; si es mayor que 0, devuelve dos raíces reales; y si es menor que 0, devuelve dos raíces complejas conjugadas. El código usa la función cmath.sqrt para manejar correctamente las raíces cuando el discriminante es negativo (es decir, cuando las raíces son complejas).





El programa calcula el número de términos necesarios en una serie matemática para que la suma del lado izquierdo de la ecuación se acerque al valor del lado derecho con una precisión de menos de El algoritmo empieza con una suma inicial y agrega términos de la serie uno a uno, comparando cada vez la suma acumulada con el valor del lado derecho de la ecuación. Si la diferencia entre ambos es menor a la tolerancia , el bucle se detiene. Si no, sigue agregando términos hasta alcanzar un máximo definido de 1000 términos para evitar un crecimiento indefinido. En este caso, el programa indica que se necesitan 1001 términos para alcanzar la precisión deseada, lo que sugiere que el límite máximo fue alcanzado sin lograr la convergencia esperada.

