

# NIVEL 3

---

## EL TRUCO DEL CENTINELA



# PROBLEMA DEL MÁXIMO COMÚN DIVISOR

**Problema:** encontrar el máximo común divisor (MCD) de dos números


**Estrategia de solución =** Búsqueda exhaustiva:

- ✓ Tomamos como denominador el menor de los dos números
- ✓ Probamos si ese denominador es divisor de los dos números. Si lo es, lo encontramos. Si no lo es, decrementamos el denominador y volvemos a empezar (iterar)



# PASEMOS A PYTHON

Guardamos en la variable `n` el menor número entre `n1` y `n2`

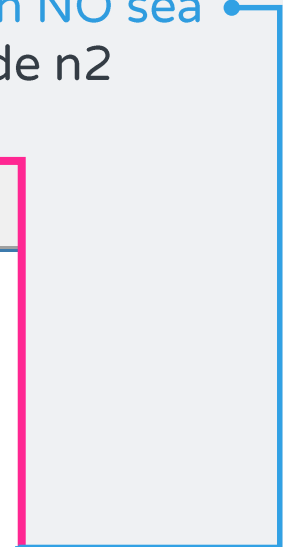
```
EjemploMCD.py   
1 def maximo_comun_divisor(n1: int, n2: int) -> int:  
2  
3     n = min(n1, n2)  
4  
5     while (n1%n!=0 or n2%n!=0):  
6         n = n - 1  
7  
8     return n  
9
```

# PASEMOS A PYTHON

Iteramos **HASTA** que **n** sea divisor de **n1** y de **n2** al mismo tiempo

Es decir, que iteramos **MIENTRAS** que **n** **NO** sea divisor de **n1** o de **n2**

```
EjemploMCD.py ✕  
1 def maximo_comun_divisor(n1: int, n2: int) -> int:  
2  
3     n = min(n1, n2)  
4  
5     while (n1%n!=0 or n2%n!=0):  
6         n = n - 1  
7  
8     return n  
9
```



# ANALICEMOS ESTAS EXPRESIONES QUE SON OPUESTAS



Iteramos **HASTA** que  $n$   
**sea divisor** de  $n1$  y de  
 $n2$  al mismo tiempo

Llegamos de la primera a la  
segunda aplicando las leyes  
de De Morgan (para negar  
expresiones lógicas)

Es decir, que iteramos  
**MIENTRAS** que  $n$  **NO** sea  
divisor de  $n1$  o de  $n2$

Si no recuerdas las leyes de De  
Morgan, repasa las lecciones  
del módulo 2



**A veces se nos facilita más pensar  
en la condición de parada (o salida)  
del ciclo, que en la condición de  
continuación (de iteración)**

Hay una solución muy útil:  
usar un CENTINELA !!!



# VEAMOS AL CENTINELA EN ACCIÓN



El centinela es una variable con la que vamos a controlar la continuación y salida del ciclo. Por eso es natural que sea una variable booleana, pero podría ser de otro tipo. En este ejemplo, la inicializamos en **False**

```
EjemploMCDConCentinela.py x
1 def maximo_comun_divisor(n1: int, n2: int) -> int:
2
3     n = min(n1, n2)
4     encontro = False
5
6     while (encontro == False):
7         if (n1%n == 0 and n2%n == 0):
8             encontro = True
9         else:
10            n = n - 1
11
12     return n
```

# VEAMOS AL CENTINELA EN ACCIÓN



Iteramos mientras el centinela siga con su valor inicial (en este ejemplo, **False**)

```
EjemploMCDConCentinela.py ✕
1 def maximo_comun_divisor(n1: int, n2: int) -> int:
2
3     n = min(n1, n2)
4     encontro = False
5
6     while (encontro == False):
7         if (n1%n == 0 and n2%n == 0):
8             encontro = True
9         else:
10            n = n - 1
11
12     return n
```



# VEAMOS AL CENTINELA EN ACCIÓN



Cuando la condición de **PARADA** del ciclo se cumple (es decir que encontramos lo que estábamos buscando), le cambiamos el valor al centinela justamente para romper el ciclo

```
EjemploMCDConCentinela.py x
1 def maximo_comun_divisor(n1: int, n2: int) -> int:
2
3     n = min(n1, n2)
4     encontro = False
5
6     while (encontro == False):
7         if (n1%n == 0 and n2%n == 0):
8             encontro = True
9         else:
10            n = n - 1
11
12     return n
```

# VEAMOS AL CENTINELA EN ACCIÓN



De lo contrario, se ejecuta el bloque normal del ciclo

```
EjemploMCDConCentinela.py ✕  
1 def maximo_comun_divisor(n1: int, n2: int)-> int:  
2  
3     n = min(n1, n2)  
4     encontro = False  
5  
6     while (encontro == False):  
7         if (n1%n == 0 and n2%n == 0):  
8             encontro = True  
9         else:  
10            n = n - 1  
11  
12     return n
```



1. Defina una función llamada `cantidad_digitos` que cuente la cantidad de dígitos de un número. Escriba el programa completo que pida el número al usuario y le informe cuántos dígitos tiene
2. Modifique la función anterior para que cuente únicamente los dígitos con valor 0 o 5. Escriba el programa completo. Pruébalo con el número 0 (cero)
3. El factorial de `n`, siendo `n` positivo, se denota con `n!`, pero no existe ningún operador en Python que permita efectuar este cálculo directamente. Sabiendo que:
  - $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n - 1) \cdot n$
  - y que  $0! = 1$ , defina una función que reciba como parámetro un número `n` y retorne `n!`
  - Escriba el programa completo, pidiendo el valor de `n` al usuario. Sólo se admiten enteros positivos

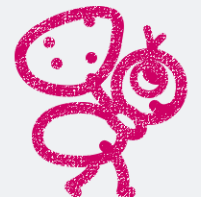




4. Escriba una función que implemente la conjetura de Collatz o el problema de Siracusa. Aquí encuentra la explicación de la conjetura:

<https://www.bbc.com/mundo/noticias-36651490>

4. En la isla del Edén vive una gran cantidad de hormigas que se reproducen a una tasa del 40% mensual; en la isla existe además un oso hormiguero que se come 7000 hormigas al final de cada mes (o todas las hormigas que haya si la población de hormigas en ese momento es inferior a 7000). Cuando la población de hormigas de la isla sobrepasa al máximo de 28000, comienza a haber problemas de alimentación, lo que hace que se reduzca la tasa de crecimiento al 31% mensual. El muestreo de la población se hace mensualmente. Escriba una función que reciba como parámetro el número de hormigas que hay en un momento dado en la isla y un número  $X$  y calcule la población de hormigas después de esos  $X$  meses





Para el ejercicio de la población de hormigas... vayamos paso a paso:

- Escriba la ecuación que describe la población de hormigas en el  $i$ -ésimo mes, a partir de la población del mes anterior ( $i-1$ ), cuando la población final del mes ( $i-1$ ) es menor que 28000, sin tener en cuenta al oso
- Escriba la ecuación que describe la población de hormigas en el mes  $i$ -ésimo a partir de la población en el mes anterior, cuando la población al final del mes anterior es superior a 2800, sin tener en cuenta el oso
- Incluya en las ecuaciones anteriores, el análisis de las hormigas que se come el oso, teniendo en cuenta el caso en que haya muchas hormigas
- Escriba la función completa en Python

