

NIVEL 3

INSTRUCCIONES INTERATIVAS – FOR IN



INSTRUCCIÓN FOR - IN

Para todo **elemento** (que llamaremos variable) de una **secuencia o serie** hacer ...

```
for variable in serie de valores:  
    acción  
    acción  
    ...  
    acción
```



La instrucción for-in es una forma compacta de ciertos while, más adelante lo estudiaremos

Se ejecutan estas **acciones**. No hay límite en la cantidad de instrucciones que componen el bloque de un **for-in**

CONTEMOS LAS OCURRENCIAS DE UN CARÁCTER EN UNA CADENA CON FOR-IN

EjemploOcurrenciasCaracterConForIn.py

```
1 def ocurrencias_caracter(cadena: str, caracter: str) -> int:
2
3     ocurrencias = 0
4     for cada_caracter in cadena:
5         if (cada_caracter == caracter):
6             ocurrencias += 1
7
8     return ocurrencias
```

Para cada carácter en la cadena

Si el carácter de la cadena es igual al carácter buscado, se incrementa el contador

Resultado de la ejecución

Terminal de IPython

Terminal 2/A

```
In [5]: ocurrencias_caracter("La Casa Blanca", "a")
Out[5]: 5
```

```
In [6]: ocurrencias_caracter("La Casa Blanca", "A")
Out[6]: 0
```

```
In [7]: ocurrencias_caracter("La Casa Blanca", "b")
Out[7]: 0
```

```
In [8]: ocurrencias_caracter("La Casa Blanca", "B")
Out[8]: 1
```


CUANDO SE PUEDE USAR FOR-IN EN VEZ DE WHILE

Ciertos ciclos se ejecutan un **número de veces fijo** y conocido **a priori**. Por ejemplo, en la función que calcula la sumatoria de los **1000** primeros números utilizamos un **while** que **iteraba exactamente 1000 veces**:

```
EjemploSumatoria.py ✕  
1 def sumatoria()->int:  
2     resultado = 0  
3     i = 1  
4     while i <= 1000:  
5         resultado += i  
6         i += 1  
7  
8     return resultado  
9  
10 #PROGRAMA PRINCIPAL  
11 print("El resultado de sumar los primeros 1000 números es: ",sumatoria())
```

CUANDO SE PUEDE USAR FOR-IN EN VEZ DE WHILE

- Estos ciclos se construyen siguiendo un patrón, que es una especie de «frase hecha» del lenguaje de programación:



```
i = valor_inicial
while i <= valor_final:
    acciones
    i += 1
```

En este patrón la variable *i* suele denominarse *índice* del ciclo

CUANDO SE PUEDE USAR FOR-IN EN VEZ DE WHILE

Podemos expresar de forma compacta este tipo de ciclos con un **for-in** siguiendo este otro patrón:

```
for i in range(valor_inicial, valor_final + 1):  
    acciones
```

La función **range** devuelve una secuencia de valores entre un valor inicial y un valor final (sin incluirlo dentro de la secuencia)

EJEMPLO DE LA SUMATORIA CON FOR-IN

Al generar el rango debemos poner el valor máximo + 1

```
EjemploSumatoriaConForIn.py ✕  
1 def sumatoria()->int:  
2     resultado = 0  
3     for i in range(1, 1001):  
4         resultado += i  
5  
6     return resultado  
7  
8 #PROGRAMA PRINCIPAL  
9 print("El resultado de sumar los primeros 1000 números es: ",sumatoria())
```

1. Escriba una función que nos diga si un número (entero) es o no es primo, usando **for-in**.
Recuerde: un número primo es aquel número mayor que 1 que solo es divisible por 1 y por sí mismo



Puedes verificar tus resultados usando la terminal presente en la actividad “Manos a la obra: Número primo”

¿Cómo empezamos?

Resolvamos un problema concreto, a ver qué estrategia seguiríamos normalmente. Supongamos que deseamos saber si 7 es primo. Podemos intentar dividirlo por cada uno de los números entre 2 y 6. Si alguna de las divisiones es exacta, entonces el número no es primo:

Dividendo	Divisor	Cociente	Resto
7	2	3	1
7	3	2	1
7	4	1	3
7	5	1	2
7	6	1	1



Ninguno de los residuos dio 0, así que 7 es primo!!

2. Escriba una función que nos diga si una cadena de caracteres es palíndrome. Esto es una palabra o frase que se lee igual de izquierda a derecha que de derecha a izquierda. Ejemplos de frases palíndromes:

- «Isaac no ronca así»
- «Sometamos o matemos»



Puedes verificar tus resultados usando la terminal presente en la actividad “Manos a la obra: Palíndromos”