

Esta clase va a ser

- grabada

Clase 05. PYTHON

Controladores de Flujo II

Temario

04

Controladores de Flujo I

- ✓ Condicional
- ✓ Else
- ✓ Elif

05

Controladores de Flujo II

- ✓ [Sentencias iterativas](#)
- ✓ [While](#)
- ✓ [For](#)

06

Conjuntos y diccionarios

- ✓ Conjuntos
- ✓ Diccionarios

Objetivos de la clase

- **Identificar** el proceso de iteración en programación
- **Diferenciar** sentencia while de sentencias while-else
- **Implementar** instrucción break
- **Diferenciar** entre instrucción continue y pass
- **Implementar** sentencia for, range y for-else-break-continue-pass

Sentencias iterativas

Repetir

Veremos cómo controlar el flujo con Python, pero antes de eso, ¿Qué es el flujo?. El flujo es una forma de entender la sucesión de las instrucciones de un programa, estas instrucciones se ejecutan una después de otras de forma ordenada y suelen tener el objetivo final de manipular información.



Iterar

En matemática, se refiere al proceso de iteración de una función, es decir, aplicando la función repetidamente, usando la salida de una iteración como la entrada a la siguiente.

En programación, Iteración es la **repetición** de un **segmento de código** dentro de un programa de computadora. Puede usarse tanto como un término genérico (como sinónimo de repetición) como para describir una forma específica de repetición.



```
31 self.file = None
32 self.fingerprints = {}
33 self.logdebugs = True
34 self.debug = debug
35 self.logger = logging.getLogger(__name__)
36
37 if path:
38     self.file = open(os.path.join(path, 'requests.log'),
39                     'a')
40     self.fingerprints.update({path: self.file})
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('SUPERSTOCK_DEBUG')
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
54
55 def request_fingerprint(self, request):
56     return request_fingerprint(request)
```



PARA RECORDAR

Tenemos una base de datos enorme y queremos encontrar un dato en especial para consultar.

Como **no existe una forma mágica** para encontrar el dato directamente, el programa deberá recorrer los datos uno a uno y compararlos hasta dar con el que buscamos iterando o repitiendo el mismo proceso desde el inicio comparando a ver si es el que queremos o no, **así hasta que encuentre el que queremos.**



PARA RECORDAR

Existen algoritmos que permiten ahorrarnos tiempo e iteraciones, pero en esencia sigue recorriendo uno a uno, la diferencia es que nosotros tardaríamos muchas horas, y el programa unos segundos.

¡Así que vamos a aprovecharnos de esto!

**While (no sabes la
cantidad)**

Sentencia **While**

Vamos a comenzar con la sentencia iterativa más básica **While** (mientras).

Se basa en repetir un bloque de código a partir de evaluar una **condición lógica**, siempre que esta sea **True** (al igual que la sentencia **if**).

Como programadores debemos decidir el momento en que la condición cambie a **False** para hacer que el **While** finalice su ejecución y así salir de la iteración, de lo contrario estaríamos frente a un bucle **infinito**.

Flujo de ejecución

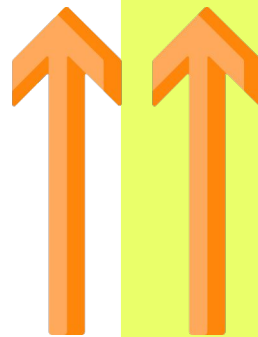
Más formalmente, el flujo de ejecución de una sentencia while es el siguiente:

1. Evalúa la condición, devolviendo False o True.
2. Si la condición es False, se sale de la sentencia While y continúa la ejecución con la siguiente sentencia.
3. Si la condición es True, ejecuta cada una de las sentencias en el bloque de código y regresa al paso 1.

Bucle

Este tipo de flujo se llama bucle porque el tercer paso del bucle vuelve arriba.

👁 Si la condición es falsa la primera vez que se pasa el bucle, las sentencias del interior del bucle no se ejecutan nunca.





Ejemplo

Analicemos qué pasó:

```
num = 5
while num > 0:
    print(f'{num}')
    num -= 1
print('Terminó el conteo!')
```

1. Declaramos una variable num y le asignamos el valor int 5.
2. Usamos la sentencia while para indicar que mientras que num sea mayor a 0 entremos al bloque de código.
3. Al evaluar num contra 0 nos indica que es True.



Ejemplo: ¿Qué pasó?

- 4. Ingresamos al bloque de código, imprimimos num y le restamos 1 a num
- 5. Volvemos a repetir desde el paso 2 hasta que num deje de ser mayor a 0
- 6. Cuando la operación relacional de False saldremos del bucle
- 7. Imprimimos por pantalla Terminó el conteo!
- 8. Termina nuestro programa



Más ejemplos

```
n = 0
while n <= 5:
    n += 1
print('N vale ', n)
```

¿Y un bucle infinito?

```
while True:
    print("Esto es un bucle infinito!!!!")
```

Para escapar un bucle infinito generalmente se usa ctrl + c 🤪

También podemos usar la opción de Restart Kernel! En Jupyter Notebook



While – Else



Sentencia **While-else**

Veamos un ejemplo:

```
chance = 1
while chance <= 3:
    txt = input("Escribe SI: ")
    if txt == "SI":
        print("Ok, lo conseguiste en el intento",
chance)
        break
    chance += 1
else:
    print("Has agotado tus tres intentos")
```

Este else sirve para ejecutar un bloque de código cuando el bucle while tenga una condición False o haya terminado y no haya sido forzado a salir mediante un break.

¿Qué ha pasado?



Sentencia **While-else**

1. Declaramos una variable chance y le asignamos el valor int 1.
2. Usamos la sentencia while para indicar que mientras que chance sea menor a 3 entremos al bloque de código.
3. Le pedimos al usuario que ingrese una palabra con input
4. Si la palabra es "SI" ingresa al condicional if
5. Si ingresa, imprime que lo consiguió en el intento tal y rompe el bucle con break
6. Si la condicional es False vamos a sumar uno a las chances y repetir desde el paso 2
7. Si chance es mayor a 3, entramos en el else e imprimimos



¡Solicitud de números al usuario!

Trabajaremos con el [notebook](#) de la sección, específicamente sobre los ejemplos previstos sobre Bucles While

Duración: 10 a 15 minutos



ACTIVIDAD EN CLASE

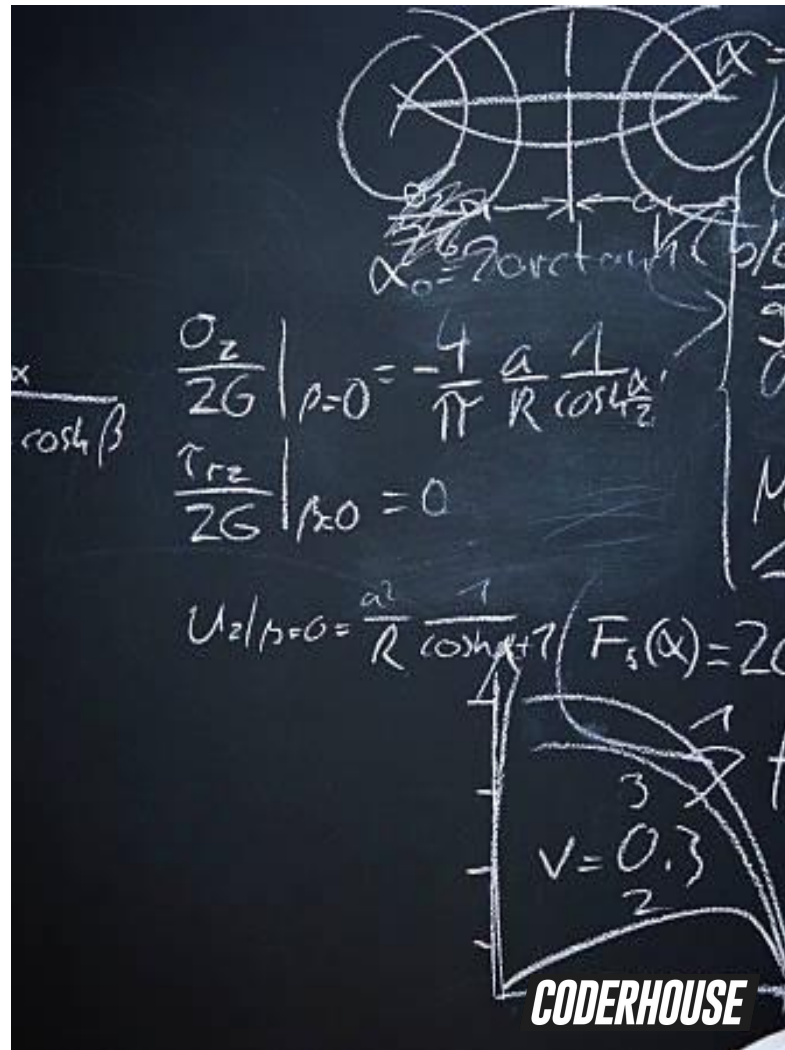
¡Solicitud de nros al usuario!

Descripción de la actividad.

Calcular la suma de una cantidad de números enteros ingresados por el usuario directamente utilizando la función input ().

Para finalizar la ejecución del programa, el usuario debe escribir la palabra exit(). El programa debe validar dicha acción.

Finalmente, el algoritmo debe mostrar la suma parcial y total obtenida.



Instrucciones

Bucles en Python

Usar bucles en Python nos permite automatizar y repetir tareas de manera eficiente. Sin embargo, a veces, es posible que un factor externo influya en la forma en que se ejecuta su programa.



Instrucciones

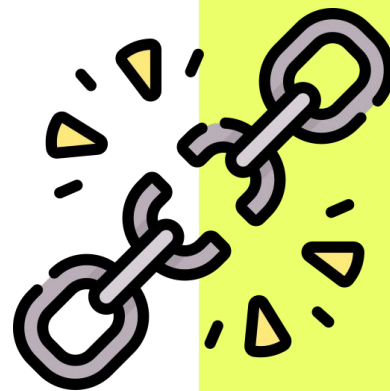
Cuando esto sucede, es posible que prefiramos que nuestro programa cierre un bucle por completo, omita parte de un bucle antes de continuar o ignore ese factor externo.

Para hacer estas acciones Python nos brinda las instrucciones **break**, **continue** y **pass**.

Break

Break

Comencemos por uno de los más sencillos y más utilizados: break. En Python, la instrucción break le proporciona la oportunidad de cerrar un bucle cuando se activa una condición externa. Debe poner la instrucción break dentro del bloque de código bajo la instrucción de su bucle, generalmente después de una sentencia if condicional.





Ejemplo

Salimos del bucle cuando **i** es igual a **3**

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```



Sentencia break

Es hora de programar mentalmente.

Duración: **10 minutos**



ACTIVIDAD EN CLASE

Sentencia break

Descripción de la actividad.

Observa el código y busca comprender qué sucedió en este caso:

```
x = 5
while True:
    x -= 1
    print(x)
    if x == 0:
        break
    print("Fin del bucle")
```

Una vez que el profesor se los solicite, explíquenlo mediante el chat o activando micrófono

Continue

Continue

La instrucción continue da la opción de omitir la parte de un bucle en la que se activa una condición externa, pero continuar para completar el resto del bucle. Es decir, la iteración actual del bucle se interrumpirá, pero el programa volverá a la parte superior del bucle.

Debe poner la instrucción continue dentro del bloque de código bajo la instrucción de su bucle, generalmente después de una sentencia if condicional.





Continue

Veamos un ejemplo:

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Ejecuta el código para analizar que fue lo que pasó

Pass

Pass

Cuando se activa una condición externa, la instrucción `pass` permite manejar la condición sin que el bucle se vea afectado de ninguna manera; todo el código continuará leyéndose a menos que se produzca la instrucción `break` u otra instrucción.

Debes poner la instrucción `pass` dentro del bloque de código bajo la instrucción de su bucle, generalmente después de una sentencia `if` condicional.

#Ejemplo 1

```
n = 0
while n < 10:
    n += 1
    if n == 2:
        pass
    print('n vale' , n)
```

#Ejemplo 2

```
n = 0
while n < 10:
    n += 1
    if n == 2:
        pass
    print('n vale' , n)
```

Ejecuta el código para analizar las diferencias 😊



Para pensar

¿Qué pasó en este ejemplo?

```
c = -3
```

```
while c < 10:
```

```
    c += 1
```

```
    if c == 2:
```

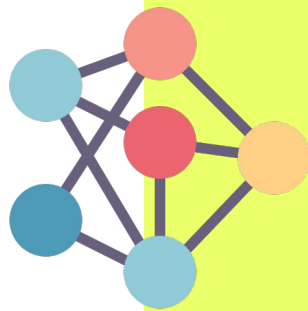
```
        pass
```

```
    print('c vale', c)
```

**For (repetir pero
sabiendo la cantidad)**

Sentencia For

Ahora seguiremos con la sentencia iterativa que podríamos decir es la más usada For (para). Se utiliza para recorrer los elementos de un objeto iterable (lista, tupla...) y ejecutar un bloque de código, o sea, tiene un número predeterminado de veces que itera.



Sentencia For

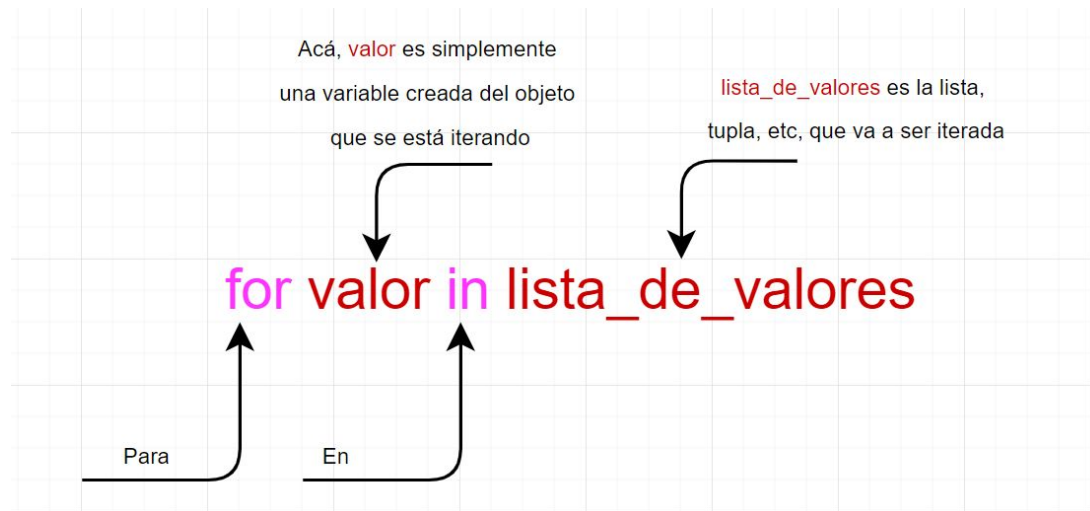
En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cual se pueden aplicar una serie de operaciones.

```
lista = [1,2,3,4,5]
for valor in lista:
    print('Soy un item de la lista y valgo', valor)
```



Ejemplo gráfico

Valor simplemente es una copia local, no afecta fuera del bucle a menos que se devuelva el valor.





EJEMPLO

```
lista = [0,1,2,3,4,5,6,7,8,9,10]
for num in lista:
    print('Soy un valor de la lista y valgo', num)
    num *= 5
    print('Soy un valor de la lista y ahora valgo',
num)
```

MODIFICANDO LA LISTA

```
indice = 0
numeros = [0,1,2,3,4,5,6,7,8,9,10]
for numero in numeros:
    numeros[indice] *= 5
    indice += 1
print(numeros)
```


Enumerate

Enumerate

La función incorporada `enumerate(lista/tupla_de_valores)` toma como argumento un objeto iterable y retorna otro cuyos elementos son tuplas de dos objetos:



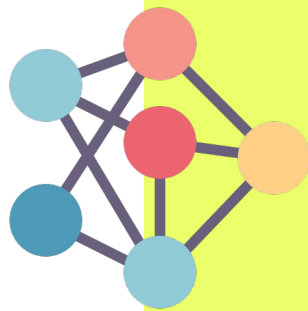
El primero de los dos indica la posición de un elemento perteneciente al objeto iterable, es decir, el índice.

El segundo, el elemento mismo.

👉 Esto se conoce como lectura secuencial de clave y valor, lo vamos a usar en el futuro

Enumerate

La función incorporada `enumerate(lista/tupla_de_valores)` toma como argumento un objeto iterable y retorna otro cuyos elementos son tuplas de dos objetos:

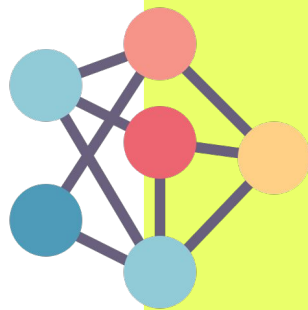




Sentencia For

Si quisiéramos recorrer un string:

```
texto = 'Hola Mundo, estoy usando for en Python'
for letra in texto:
    print(letra)
texto2 = ""
for letra in texto:
    texto2 = letra * 2
print(texto2)
```

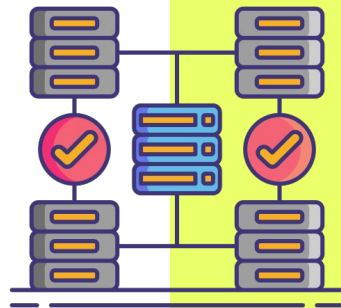


Range

¿Qué es?

Como vimos, el for en python necesita una colección de datos para poder utilizarlo, en otros lenguajes necesitamos solamente un número para indicar las iteraciones a cumplir.

Para simular estos casos Python nos provee de una función denominada range() (rango) el cual representa una colección de números inmutables.



Constructores para crear objetos Range

1

`range(fin)`: Crea una secuencia numérica que va desde 0 hasta $fin - 1$.

a. `for numero in range(10)`

2

`range(inicio, fin)`: Crea una secuencia numérica que va desde `inicio` hasta $fin - 1$.

a. `for numero in range(5, 10)`

3

`range(inicio, fin, [paso])`: Crea una secuencia numérica que va desde `inicio` hasta $fin - 1$. Si además se indica el parámetro `paso`, la secuencia genera los números de `paso` en `paso`.

a. `for numero in range(0, 20, 2)`



VENTAJAS

```
>>> range(0,10)
```

Parece ser exactamente una lista que va de 0 a 10, pero range interpreta el inicio y fin en tiempo de ejecución y eso le da ventaja contra la lista.

Si tuviéramos una lista de 0 a 10000 estaría ocupando muchísimo espacio en memoria.

```
>>> range(0,10000)
```

Ahora, si hiciéramos range(0,10000), range interpreta esto en tiempo de ejecución, es decir cuando se ejecuta el 0 se crea el 0 cuando se ejecuta el 1 se crea el 1 y se elimina el 0 y así continuamente, y esto no ocupa memoria.

For-else

Igual que en la sentencia while podemos usar un else al final de la iteración.

```
for numero in range(10):  
    print('Numero vale',numero)  
else:  
    print("Se terminó de iterar y numero vale: ", numero)
```



For-break-continue-pass

Igual que en la sentencia while podemos usar también las instrucciones break continue y pass.

```
for numero in range(10):  
    if numero == 2:  
        continue  
    elif numero == 8:  
        break  
    else:  
        print("Se terminó de iterar y numero  
vale: ", numero)
```



Canción – Me Gusta

Trabajaremos con el [notebook](#) de la sección específicamente sobre los ejemplos previstos sobre Bucles For

Duración: **10 minutos**



Canción – Me gusta

Descripción de la actividad.

Escribir la letra de la canción *Me gusta* de Manu Chao, utilizando la sentencia de iteración for:

<https://www.letras.com/manu-chao/7352/>

Importante: Deberán crear también una lista con los párrafos de la canción para poder imprimirlos correctamente por pantalla.

Ejemplo:

Me gustan los aviones, me gustas tú

Me gusta viajar, me gustas tú

Me gusta la mañana, me gustas tú

Me gusta el viento, me gustas tú

Me gusta soñar, me gustas tú

Me gusta la mar, me gustas tú



#Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar el Workbook, donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

Hora de entrenar

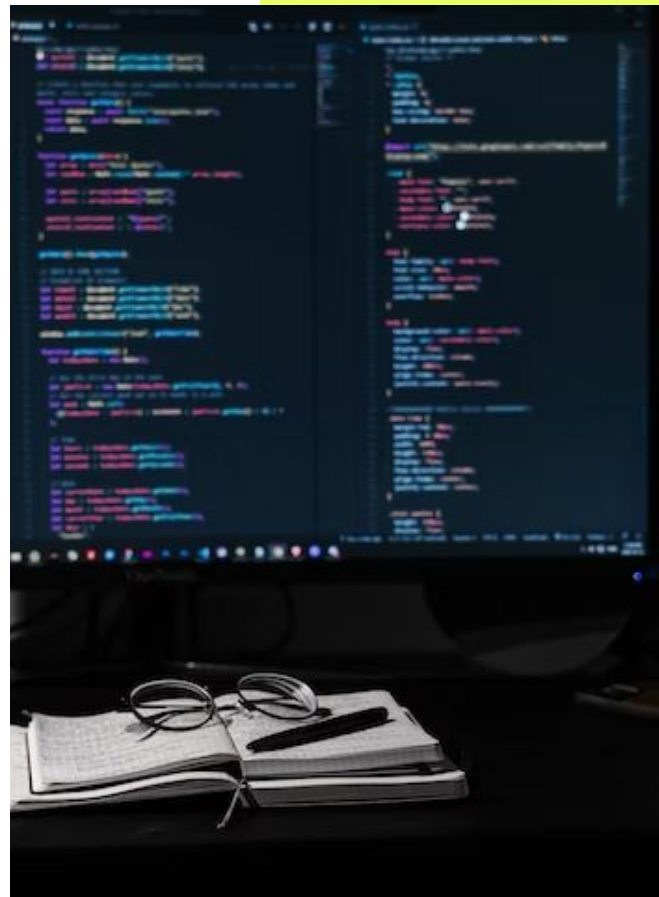
#WorkingTime

Levante la mano quienes quedaron con ganas de seguir practicando. 🙋🙋

Bueno, con el propósito de que sigan robusteciendo sus perfiles profesionales y desarrollando habilidades, les invitamos a revisar el **Workbook del curso**, donde conseguirán un ejercicio para poner en práctica lo visto en la clase de hoy.

Este material es completamente de apoyo por lo que no requerirá ser entregado al profesor/a ni a los tutores o corregido por los mismos. No obstante sí recomendamos que puedan realizar intercambio de las respuestas y logros alcanzados en el workbook, con otros estudiantes y así nutrirse de tips y formas de trabajo.

Fuente de imagen: [Kevin Canlas](#) en [Unsplash](#)





¡Instrucciones e iteración!

Consigna

- ✓ Realiza los ejercicios 1, 2, 3, 4, 5 y 6.

Formato

- ✓ Puedes completar estas consignas en un Google Docs o un link a su Colabs.



¡Instrucciones e iteración!


- 1) Escribe un programa que lea dos números por teclado y permita elegir entre 4 opciones en un menú:
 - ✓ Mostrar una suma de los dos números
 - ✓ Mostrar una resta de los dos números (el primero menos el segundo)
 - ✓ Mostrar una multiplicación de los dos números
 - ✓ Si elige esta opción se interrumpirá la impresión del menú y el programa finalizará
 - ✓ En caso de no introducir una opción válida, el programa informará de que no es correcta.



¡Instrucciones e iteración!

2) Escribe un programa que lea un número impar por teclado. Si el usuario no introduce un número impar, debe repetirse el proceso hasta que lo introduzca correctamente.

3) Escribe un programa que sume todos los números enteros impares desde el 0 hasta el 100:

 **Ayuda:** Puedes utilizar la funciones `sum()` y `range()` para hacerlo más fácil. El tercer parámetro en la función `range(inicio, fin, salto)` indica un salto de números.

4) Escribe un programa que pida al usuario cuantos números quiere introducir. Luego lee todos los números y realiza una media aritmética.



¡Instrucciones e iteración!

5) Escribe un programa que pida al usuario un número entero del 0 al 9, y que mientras el número no sea correcto se repita el proceso. Luego debe comprobar si el número se encuentra en la lista de números y notificarlo:

👉 **Ayuda:** La sintaxis "valor in lista" permite comprobar fácilmente si un valor se encuentra en una lista (devuelve True o False).

6) Utilizando la función range() y la conversión a listas, genera las siguientes listas dinámicamente:

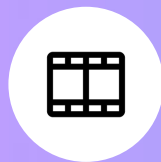
- ✓ Todos los números del 0 al 10 [0, 1, 2, ..., 10]
- ✓ Todos los números del -10 al 0 [-10, -9, -8, ..., 0]
- ✓ Todos los números pares del 0 al 20 [0, 2, 4, ..., 20]
- ✓ Todos los números impares entre -20 y 0 [-19, -17, -15, ..., -1]
- ✓ Todos los números múltiplos de 5 del 0 al 50 [0, 5, 10, ..., 50]

👉 **Ayuda:** la conversión de listas es `mi_lista=list(range(inicio,fin,salto))`

¿Preguntas?

Resumen de la clase hoy

- ✓ Iteración
- ✓ While
- ✓ Instrucciones break – continue – pass
- ✓ For



¿Quieres saber más?
**Te dejamos material
ampliado de la clase**



MATERIAL AMPLIADO

Recursos

- ✓ [Iteración](#)
- ✓ [While – Break – Continue](#)
- ✓ [For](#)
- ✓ [Range](#)
- ✓ [EjemploClase](#)

Opina y valora
esta clase

Muchas gracias.

#DemocratizandoLaEducación