Técnica:

ITERACIÓN USANDO for

Estructura *for*

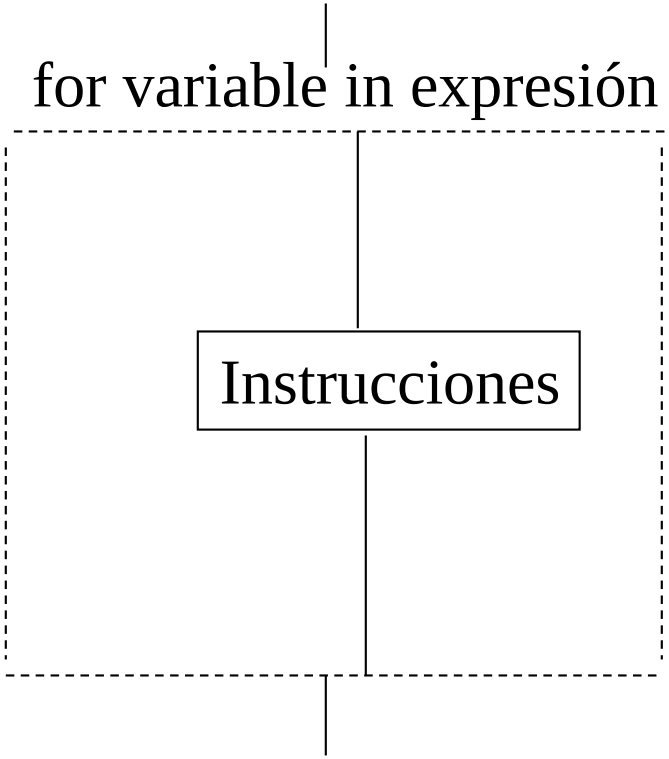
- Repite la ejecución de un bloque de instrucciones según la cantidad de elementos que tenga un objeto iterable

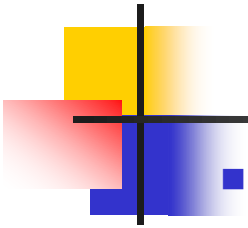
- Objeto iterable: objeto cuyos elementos se pueden acceder individualmente, por ejemplo las secuencias (listas, tuplas, strings), conjuntos, diccionarios, etc.

¿ cómo funciona ?

- El resultado de la expresión es un objeto iterable que tiene n elementos, por tanto el bloque de instrucciones se ejecuta n veces.

Representación del for mediante un diagrama de flujo

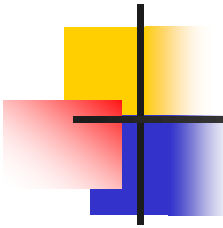




- En la primera iteración Python crea la variable del for y le asigna automáticamente el primer elemento del objeto iterable, luego ejecuta el bloque de instrucciones dentro del for, cuando finaliza la ejecución del bloque de instrucciones regresa a la instrucción donde está el for.

- En la segunda iteración le asigna el segundo elemento, luego ejecuta el bloque de instrucciones y cuando finaliza la ejecución regresa al for. Esto lo hace sucesivamente hasta el último elemento del objeto iterable.

- Cuando Python haya recorrido todos los elementos del objeto iterable el for termina automáticamente y la ejecución continúa en la siguiente instrucción que se encuentre después de la estructura del for.



- Sintaxis de la estructura en Python

for variable **in** expresión:
Instrucciones

- El bloque de instrucciones del for está indentado respecto a la palabra for
- En el bloque de instrucciones puede haber cualquier tipo de instrucción, incluyendo condicionales, iteración, asignación, etc.

- Ejemplo: función para obtener el largo de una lista (cantidad de elementos)

```
def largo(lis):
    contador = 0
    for e in lis:    #en cada iteración obtiene un elemento de la lista
        contador = contador + 1
    return contador
```

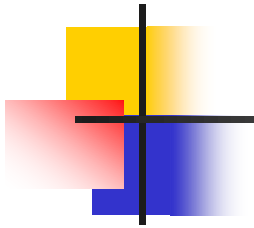
Ejemplos del funcionamiento:

```
>>> largo([10 ,5, 12])
3

>>> largo([])
0
```

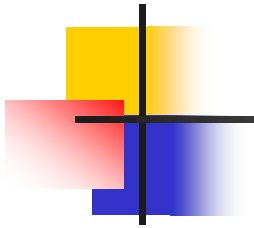
- Ejemplo: función booleana para determinar si un elemento está en una lista

```
def existe_elemento(codigo, sec):
    for elemento in sec: #en cada iteración obtiene un elemento de la lista
        if elemento == codigo:
            return True
    return False

>>> existe_elemento(10, [5, 19, 10, 8])
True
>>> existe_elemento(41, [85, 89, 65])
False
```

Función **range**



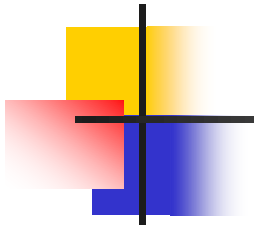
- En el for es muy común usar esta función
- range ofrece una forma sencilla de crear listas de valores enteros consecutivos
- Se puede usar de tres formas



- Forma 1: un argumento
  - Crea una lista desde 0 hasta el argumento sin incluirlo. El incremento entre cada elemento es de 1.
  - Ejemplo

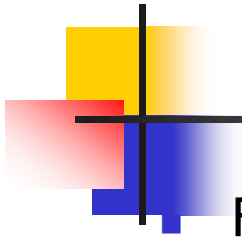
range(12)

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]



- Forma 2: dos argumentos
  - Crea una lista que contiene todos los enteros empezando en el primer argumento hasta el segundo argumento sin incluirlo. El incremento entre cada elemento es de 1.
  - Ejemplo

```
range(3, 7)
[3, 4, 5, 6]
```



- Forma 3: tres argumentos
  - Crea una lista que contiene todos los enteros empezando en el primer argumento hasta el segundo argumento sin incluirlo, y el tercer argumento indica el incremento entre los valores sucesivos
  - Ejemplos

```
range(3, 10, 2)
[3, 5, 7, 9]
```

```
range(-3, 6, 2)
[-3, -1, 1, 3, 5]
```

- En caso de que la lista de valores no pueda crearse entonces la lista no contendrá valores, será una lista vacía
  - Ejemplos

```
range(-10, 0, -1)
[]
```

```
range(15, 15)
[]
```

- Ejemplo: imprimir la tabla de multiplicación del 5 usando range

```
>>> for i in range(10):
        print(5,"x",i,"=",5*i)
```

```
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
```

Práctica

- Construya la función **factorial** usando el estatuto for. Recibe un número entero (>= 0) y retorna su factorial. Validar restricciones.

- Construya la función **tabla_multiplicar** usando el estatuto for. Recibe tres números enteros (>= 0) e imprime la tabla de multiplicar del primer argumento empezando en el segundo argumento y terminando en el tercer argumento. Validar restricciones, incluyendo que el segundo argumento <= tercer argumento. Ejemplo del funcionamiento:

```
>>> tabla_multiplicar(12, 3, 6)
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
12 x 6 = 72
```