For en Python

El *for* en Python puede tener sus *"variantes"*, pero más que variantes son diferentes técnicas o maneras de, generalizando, acceder a índices según lo que querramos.

La diferencia entre este for y los de otros lenguajes es que podemos iterar de muchas formas este, sin ir a ejemplos genericos ni a una definición directa, creo que la mejor manera de explicar su funcionamiento y algunas variantes es con ejemplos

El **más básico** es el siguiente:

```
for variable in iterable:
    print("Inicio de la iteración")
    print(variable)
    print("Fin de la iteración")
```

En *variable* y en *iterable* pueden haber estructuras y la variable que corresponda a dicho iterable.

Los ejemplos básicos son los siguientes:

```python
# Listas
nombres_lista = ["Juan", "Carlos", "Fulano"]
for nombre in nombres_lista:
    print(nombre)

# Tupla
nombres_tupla = ("Juan", "Carlos", "Fulano")
for nombre in nombres_tupla:
    print(nombre)

# Set
nombres_set = {"Juan", "Ana", "Elena"}
for nombre in nombres_set:
    print(nombre)

# Diccionario
nombres_diccionario = {"Juan": 1, "Ana": 2, "Elena": 3}
for nombre in nombres_diccionario:
    print(nombre)
```

También podémos recorrer una cadena caracter por caracter, empezando desde su primera letra como indice 0, aunque esto no debe ser aclarado:

```
for x in "cadena":
    print(x)
```

Otra posibilidad es usar la función ***range*(valor_inicial, valor_final+1, paso).**
Es decir: el valor final esta excluido del for:

```
for i in range(0, 11):
    print(i)
```

El ***paso*** es opcional, si no ponemos nada sumará 1.
Una estructura interesante es que si tomamos como valor inicial el ***maximo valor de una estrucutura - 1,*** asignamos como valor final ***-1***, y como paso ***-1*** podrémos recorrer cualquier estructura a la inversa, sea cual sea dado que es una cuestión de pasos lógicos genericos:

```
cadena = "cadena"
for i in range(len(cadena) - 1, -1, -1):
    print(i)
```

Otra manera de iterar es usando ***comprensión***, esto nos simplificará una bararidad de código, permitiendo retornar estructuras en una sola línea de código sin problema, pudiendo incluso generar nuestras nuevas estructuras con su correspondiente condición si es que la requerimos, algo sumamente eficiente para optimizar código:

```
lista_valores = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
lista_inversa = [lista_valores[i] for i in range(len(lista_valores) -1, -1, -1)]
print(lista_inversa)
```

Hay muchas más maneras y técnicas de ***iterar*** estructuras, en este apunte solo se ven las básicas, pero en el libro de python se ven otras maneras muy interesantes también