Nombre del bloque PROGRAMACIÓN II

Año de cursada 1° AÑO

Clase Nº 5: ITERADORES E ITERABLES

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Asignaciones simples y múltiples.
- Iterables.
- Control de flujo condicional (if, elif, else)
- Control de flujo bucles (While, For)
- Control de flujo dentro de bucles (break y continue)

Objetivos de Aprendizaje:

- Comprender el concepto de iterables e iteradores en Python y su importancia en la programación.
- Familiarizarse con el control de flujo condicional (if, elif, else) y los bucles (While, For) para tomar decisiones y repetir tareas en un programa.
- Practicar la implementación de iteradores y bucles en Python a través de ejercicios prácticos.
- Mejorar la eficiencia de los programas mediante el uso adecuado de estructuras de control de flujo.

Presentación:

¡Bienvenidos a la quinta clase de este trayecto formativo denominado Programación II! 🖐 En las clases anteriores, exploramos herramientas fundamentales para el desarrollo en Python, incluyendo el sistema de control de versiones Git integrado con GitHub, así como cómo optimizar nuestro flujo de trabajo utilizando VSCode.

Además, estudiamos fascinantes herramientas de inteligencia artificial diseñadas para asistirnos en nuestro proceso de programación, como BlackBox AI y Bito Code. Estas tecnologías están transformando la forma en que desarrollamos

software al ofrecer soluciones innovadoras para mejorar la eficiencia y calidad de nuestro código. 🤖

Hoy nos adentraremos en el control de flujo en Python, una habilidad esencial para cualquier programador. El control de flujo nos permite tomar decisiones dinámicas durante la ejecución del programa, repetir secciones de código mediante bucles y realizar operaciones basadas en datos específicos. Con estas estructuras de control, podemos dirigir y gestionar el orden de ejecución de nuestras instrucciones de manera más eficiente. 💡

Sin más que agregar, vamos a por nuestra quinta clase, les repito estoy a disposición para facilitarles lo que necesitan y los invito a recorrer la quinta clase. Exitos!! 😀

Desarrollo y Actividades:

Iterables

Los objetos iterables en Python son objetos capaces de ser recorridos iterando cada uno de los elementos que contienen. Algunos ejemplos de objetos iterables son **las listas, las tuplas, las cadenas de caracteres o diccionarios**.

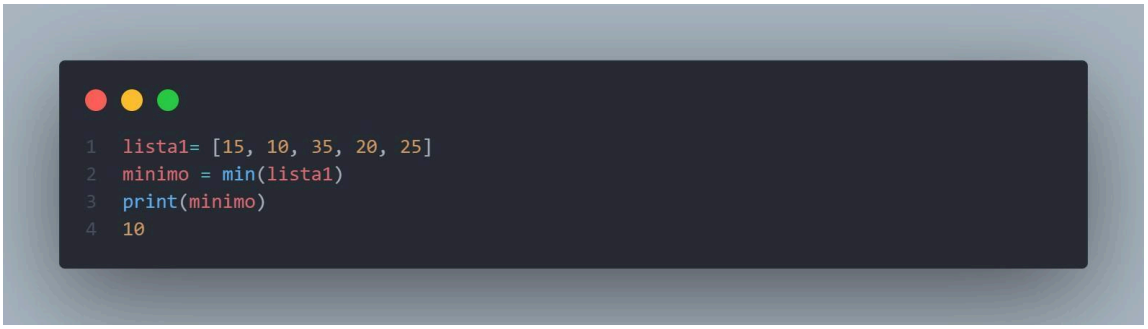
Iteradores

Los iteradores son objetos en Python que se encargan de recorrer un objeto iterable uno a uno, devolviendo un elemento a la vez. Los iteradores en Python tiene definido el método next() que devuelve el siguiente elemento en el iterable y lanza la excepción Stopiteration cuando llega al final del iterable.

Operadores para trabajar con iterables

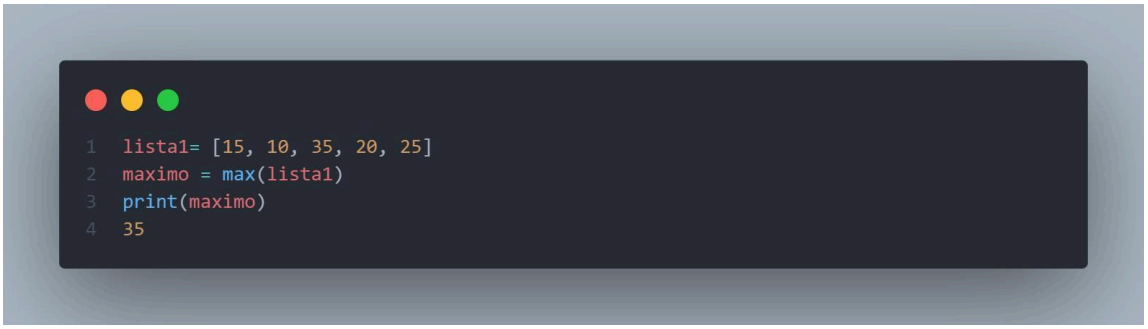
Python soporta múltiples funciones estándar para trabajar con iterables de forma fácil y sencilla e incluso convertirlos en iteradores fácilmente.

- min: Esta función devuelve el valor mínimo de los que le pasen como parámetro.



```
lista1= [15, 10, 35, 20, 25]
minimo = min(lista1)
print(minimo)
10
```

- max: Esta función devuelve el valor máximo de los que se le pasen como parámetro.



```
lista1= [15, 10, 35, 20, 25]
maximo = max(lista1)
print(maximo)
35
```

- enumerate: (iterable, start = 0) devuelve un objeto enumerate, el cual es una tupla de dos elementos en al que el primero es un entero que determina el índice (comenzando por start), y el segundo elemento es el elemento en la posición del índice iterable.

```
marcas= ['Ford', 'Renault', 'Peugeot', 'Nissan', 'Hyundai']
for indice, marca in enumerate(marcas):
    print(indice,marca)
0 Ford
1 Renault
2 Peugeot
3 Nissan
4 Hyundai
```

En el ejemplo, la lista **marcas** contiene cinco marcas de vehículos, la función enumerate() se utiliza en un bucle for para recorrer la lista y obtener un registro del índice de cada elemento. En cada iteración del bucle, la función enumerate() devuelve una tupla que contiene el índice actual y el elemento correspondiente.

- list: ([iterable]) crea una lista con los valores obtenidos tras iterar un iterable hasta llegar al final del mismo.

```
1  rango = range(1, 6)
2  numeros = list(rango)
3  print(numeros)
4  [1, 2, 3, 4, 5]
```

La función list() se puede utilizar con cualquier objeto iterable, como una cadena de texto, una tupla, un diccionario. La función list() es una forma útil de convertir cualquier objeto iterable en una lista para poder manipularlo y acceder a sus elementos de forma más fácil.

- zip: La función zip() se utiliza en Python para combinar varios iterables en un solo objeto iterable que contiene tuplas de elementos correspondiente a cada iterable. Soporta mínimo dos iterables.

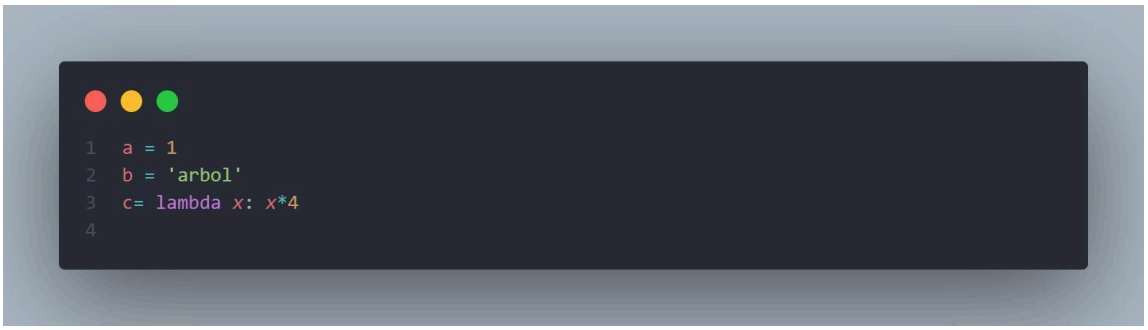
```
1  nombres = ["Gonzalo", "Manuel", "Cristian"]
2  edades = [30, 40, 35]
3  personas = zip(nombres, edades)
4  print(list(personas))
5  [('Gonzalo', 30), ('Manuel', 40), ('Cristian', 35)]
6
```

Asignaciones simples y múltiples

Las asignaciones de objetos a variables son una de las partes más utilizadas del lenguaje.

Asignaciones simples:

Este tipo de asignaciones son las más simples y también las más utilizadas.



Se componen de una sola variable en la parte izquierda y de un valor o un literal en la parte derecha de la asignación, separando las partes con un '='.

Asignaciones de múltiples variables:

Son asignaciones algo más complejas y hay que tratarlas con algo de cuidado, ya que pueden dar lugar a confusión. Principalmente se componen de varias asignaciones encadenadas en las que de izquierda a derecha se van haciendo asignaciones de un mismo objeto a variables diferentes. El valor a asignar es el que se encuentra más a la derecha (el de la última asignación) y se irá asignando a todas las variables que estén a la izquierda.

```
a=b=c=5
print('valor de a =',a,'valor de b =',b,'valor de c =',c)
valor de a = 5 valor de b = 5 valor de c = 5
```

```
d = e =[1,2,3]
print('valor de d =',d,'valor de e =',e )
valor de d = [1, 2, 3] valor de e = [1, 2, 3]
```

Asignaciones múltiples

Las asignaciones múltiples se basan en asignar valores a varias variables a la vez en una sola sentencia de código:

```
a,b,c =[1,2,3]
print(a,b,c)
1 2 3
```

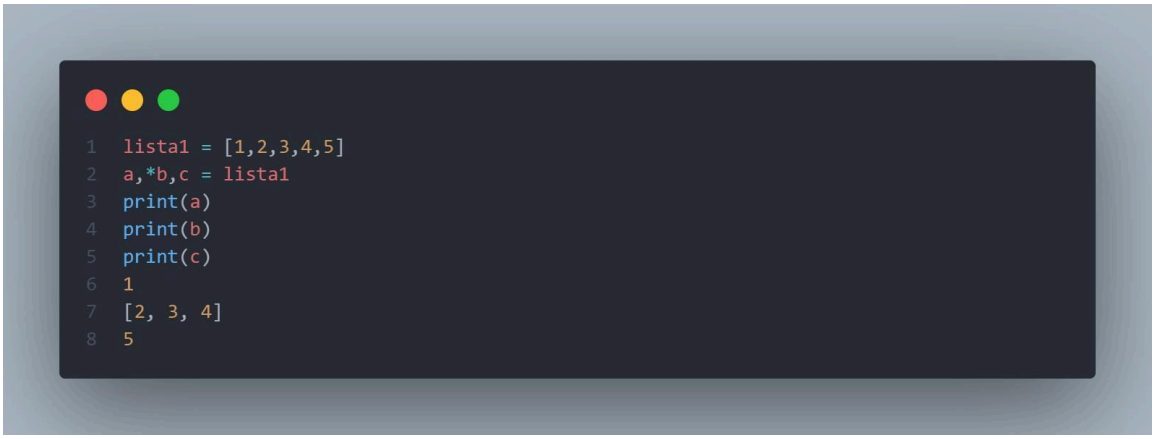
```
d,e,f='Hola'
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\diccionarios.
ipynb Celda 25 in 1
----> 1 d,e,f='Hola'

ValueError: too many values to unpack (expected 3)
```

Hay que tener en cuenta que cada elemento de las secuencias que hay a la derecha se desempaqueta uno a uno, por lo que el número de variables debe ser igual al número de elementos por asignar, de lo contrario, se elevará una excepción del tipo ValueError.

*Asignaciones slicing con **

Existe una forma más sofisticada de hacer asignaciones cuando se trabaja con secuencias. Consiste en hacer uso del operador *, el cual se utiliza ampliamente para descomponer una secuencia en cada uno de sus elementos.



```
lista1 = [1,2,3,4,5]
a,*b,c = lista1
print(a)
print(b)
print(c)
1
[2, 3, 4]
5
```

Control de flujo de ejecución

Al desarrollar programas o algoritmos basados en lenguajes de programación se utilizan sentencias que permiten encauzar el flujo de la ejecución de un programa. Pudiendo así especificar si se deben ejecutar unas operaciones u otras en función de una expresiones o funciones lógicas.

Control de flujo condicional con las sentencias if, elif y else

En Python, para controlar el flujo de la ejecución de forma condicional se utilizan las sentencias if, elif y else.

La sintaxis de la sentencia if es la siguiente:

"if" expresión ":" código

("elif" expresión ":" código)

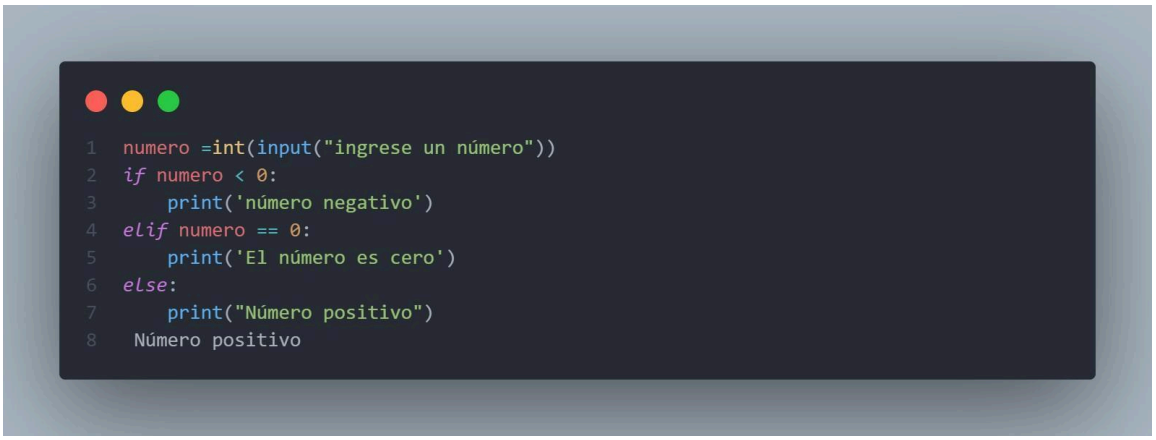
("else" ":" código)

En el condicional **if**, elif y else son opcionales, y elif puede aparecer un número indeterminado de veces.

código: hace referencia al código que se pretende ejecutar.

expresión: si esta expresión se evalúa como verdadera, se ejecutará el código asociado.

else: cuando no se evalúa como verdadera ninguna expresión de if o de elif, se ejecuta por defecto el bloque de código asociado a la sentencia else, si existe.



Sentencia if ternaria

En Python existe una opción simplificada para utilizar la sentencia if cuando se desea usar en una sola sentencia, se denomina **ternaria**.

```
edad = 40
persona = 'Menor' if edad <18 else 'Mayor'
print(persona)
Mayor
```

Actividad N° 1 🤓

1. Ingresar dos números por teclado a y b. Si a es mayor que b, imprimir ‘Hola Politécnico Malvinas’
2. Ingresar dos números por teclado a y b. Si a no es igual a b, imprimir ‘Hola Politécnico Malvinas’
3. Ingresar dos números por teclado a y b. Si a es igual a b, imprimir ‘son iguales’ sino imprimir ‘son distintos’
4. Ingresar dos números por teclado a y b. Si a es igual a b, imprimir 1 si a es mayor que b, imprimir 2 sino imprimir 3.
5. Ingrese cuatro números por teclado a, b, c y d. Si a es igual b y c es igual a d, imprimir ‘Hola Politécnico Malvinas’
6. Ingrese cuatro números por teclado a, b, c y d. Si a es igual b ó c es igual a d, imprimir ‘Hola Politécnico Malvinas’
7. Ingresar la edad de una persona por teclado, mostrar por pantalla si es mayor o menor de edad.
8. Ingresar un número por teclado y mostrar por pantalla si es para o

impar.

9. Ingresar por teclado tres
   números y mostrar por pantalla el mayor de los tres.
10. Ingresar por teclado un nombre, si la variable nombre, no es nula mostrar por pantalla ‘Bienvenido ‘+ nombre, sino mostrar ‘Bienvenido anónimo’

Flujo de ejecución con bucles

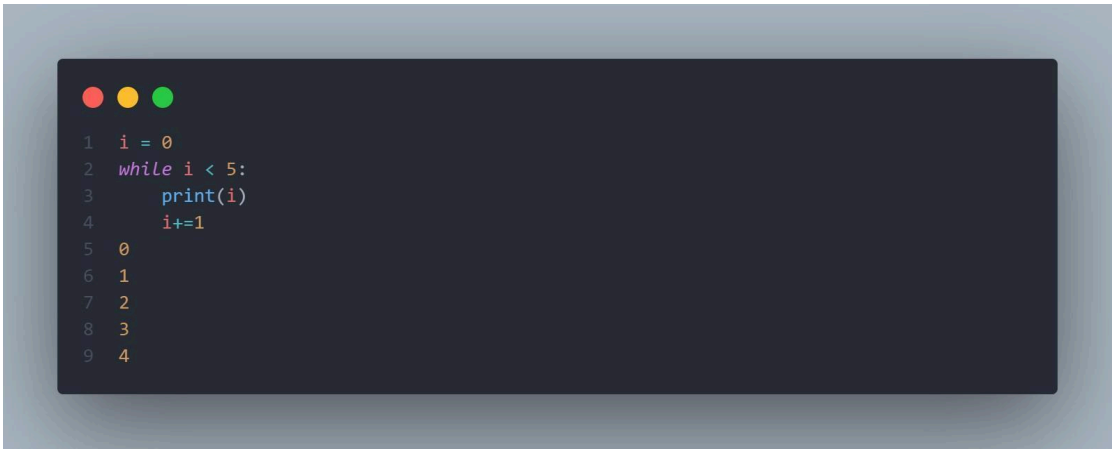
En programación suele suceder que el flujo de un algoritmo se mantiene realizando operaciones hasta que una expresión lógica cambie su valor u ocurra algún acontecimiento esperado. Para este tipo de ejecuciones se implementan los **bucles**, que son trozos de código que deben ser ejecutados hasta que una expresión determinada ocurra.

Bucle while

El bucle while permite permanecer iterando sobre un bloque del código continuamente “mientras” una expresión sea verdadera, La sintaxis básica es la siguiente:

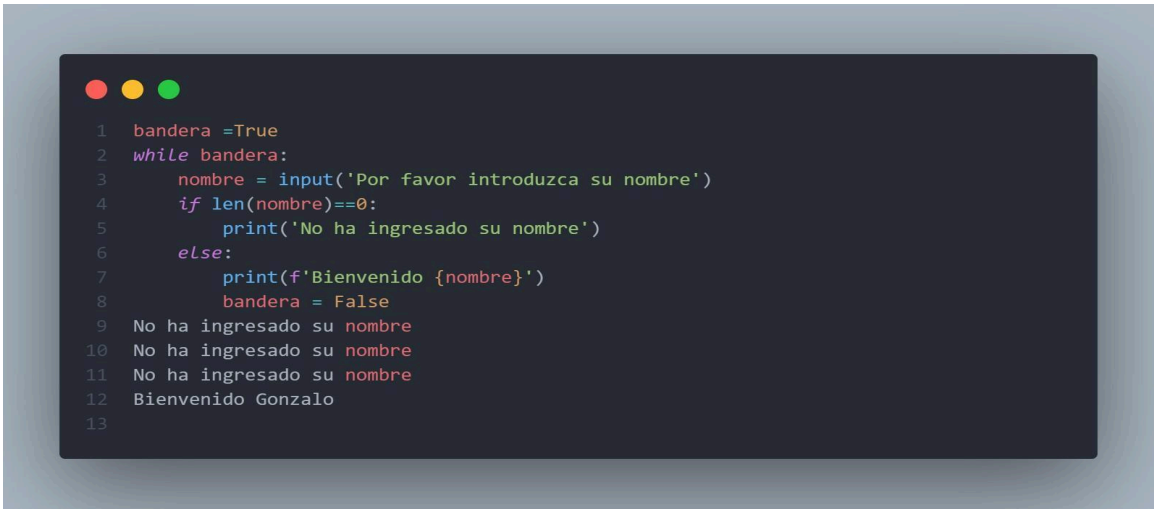
“while” expresión “:” código

Como podemos ver, el bucle while consta de una expresión que de validarse como verdadera, ejecutará el bloque de código correspondiente.



Los bucles while son famosos por generar **bucles infinitos** de manera intencionada o accidental. Este tipo de bucles son consecuencia de que la expresión utilizada siempre se evalúe como verdadera y, por lo tanto, la ejecución nunca salga de ese bloque de código.

Los bucles infinitos se usan de forma controlada cuando se pretende ejecutar constantemente un código a la espera de un evento específico. Ejemplo del uso de esta lógica podría ser la construcción en la que siempre se espera la interacción del usuario para continuar.



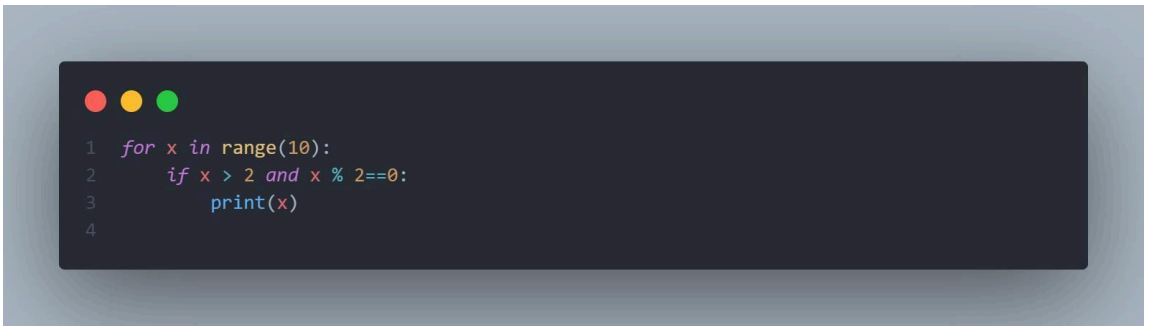
```
bandera =True
while bandera:
    nombre = input('Por favor introduzca su nombre')
    if len(nombre)==0:
        print('No ha ingresado su nombre')
    else:
        print(f'Bienvenido {nombre}')
        bandera = False
No ha ingresado su nombre
No ha ingresado su nombre
No ha ingresado su nombre
Bienvenido Gonzalo
```

En la ejecución de la imagen, el ciclo sigue pidiendo que ingrese un nombre, mientras que la variable nombre tenga una longitud 0, cuando detecta que la longitud es mayor a cero, cambia el valor de la variable que hace que el ciclo sea infinito y arroja el mensaje de bienvenida.

Bucle for

La otra forma de construir bucles en Python es haciendo uso de la sentencia **for**. La sentencia for es ampliamente utilizada tanto para el control de flujo

como para iterar por iterables, de hecho el bucle for está pensando en Python para ser utilizado por esta segunda opción.

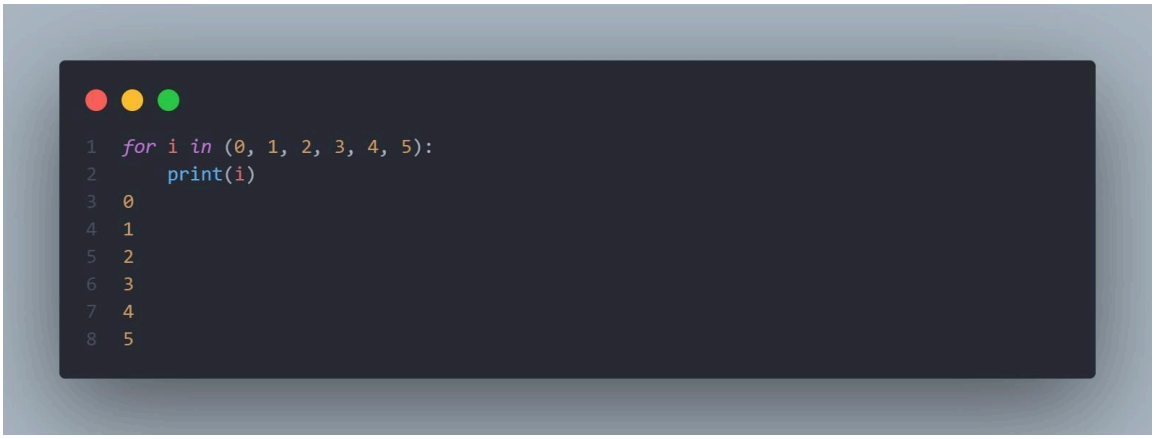


```python
for x in range(10):
    if x > 2 and x % 2==0:
        print(x)

```

range

Una de las iteraciones más comunes que se realizan, es la de iterar un número entre por ejemplo 0 y n. Pongamos que queremos iterar una variable i de 0 a 5.



```python
for i in (0, 1, 2, 3, 4, 5):
    print(i)
0
1
2
3
4
5
```

Como vemos en la imagen, hemos hecho uso de una tupla para poder iterar los números de 0 a 5, podríamos definir una lista de igual manera. Sin embargo Python nos brinda una forma más simple de hacer, que haciendo uso del método range()

El range() genera una secuencia de números que van desde 0 por defecto hasta el número que se pasa como parámetro menos 1. Se pueden pasar

hasta tres parámetros separados por coma, donde el primer es el inicio de la secuencia, el segundo el final y el tercero el salto que se desea entre números. Por defecto se empieza en 0 y el salto es de 1.

range(inicio, fin, salto)

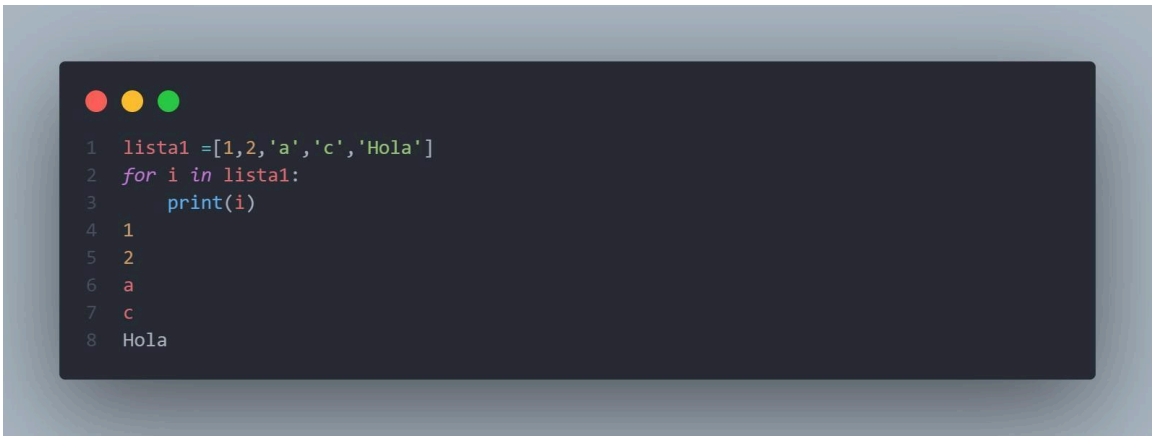
```
for i in range(3, 30, 3):
    print(i)
3
6
9
12
15
18
21
24
27
```

Se pueden generar también secuencias inversas, empezando por un número mayor y terminando en uno menor, pero para ello el salto deberá ser negativo.

```
for i in range(30, 0, -3):
    print(i)
30
27
24
21
18
15
12
9
6
3
```

como mencionamos anteriormente un bucle es un iterador y una lista,

tupla, cadena o diccionario son objetos iterables. El bucle for en Python está pensado para recorrer estas colecciones.



```
lista1 =[1,2,'a','c','Hola']
for i in lista1:
    print(i)
1
2
a
c
Hola
```

```
saludo = 'Hola Mundo'
for i in saludo:
    print(i)
H
o
l
a

M
u
n
d
o
```

Control de Flujo dentro de bucles: break y continue

Cuando se trabaja con bucles, se puede cambiar el curso de las iteraciones y hacer que algunas se salten o que se pare la ejecución de las mismas. Para este fin se crearon las sentencias **break** y **continue**.

Cuando se pretende detener la ejecución de un bucle que se está ejecutando , se usa la sentencia break.

```python
contador = 0
while True:
    if contador == 5:
        break
    print(contador)
    contador += 1
```

```
0
1
2
3
4
```

```python
for i in range(10):
    print(i)
    if i >=5:
        break
```

```
0
1
2
3
4
5
```

Cuando se pretende omitir la ejecución en curso y seguir con la siguiente volviendo a evaluar la expresión del bucle, se utiliza **continue**.

```python
for nombre in ['Gonzalo','Pedro','Victor','Manuel','Cristian','Oscar']:
    if len(nombre) ==5:
        continue
    print(nombre)
```

```
Gonzalo
Victor
Manuel
Cristian
```

como podemos ver en el código
de la imagen, al cumplirse la condición se ejecuta el **continue**, omitiendo la ejecución del print()

Actividad N° 2 🤓

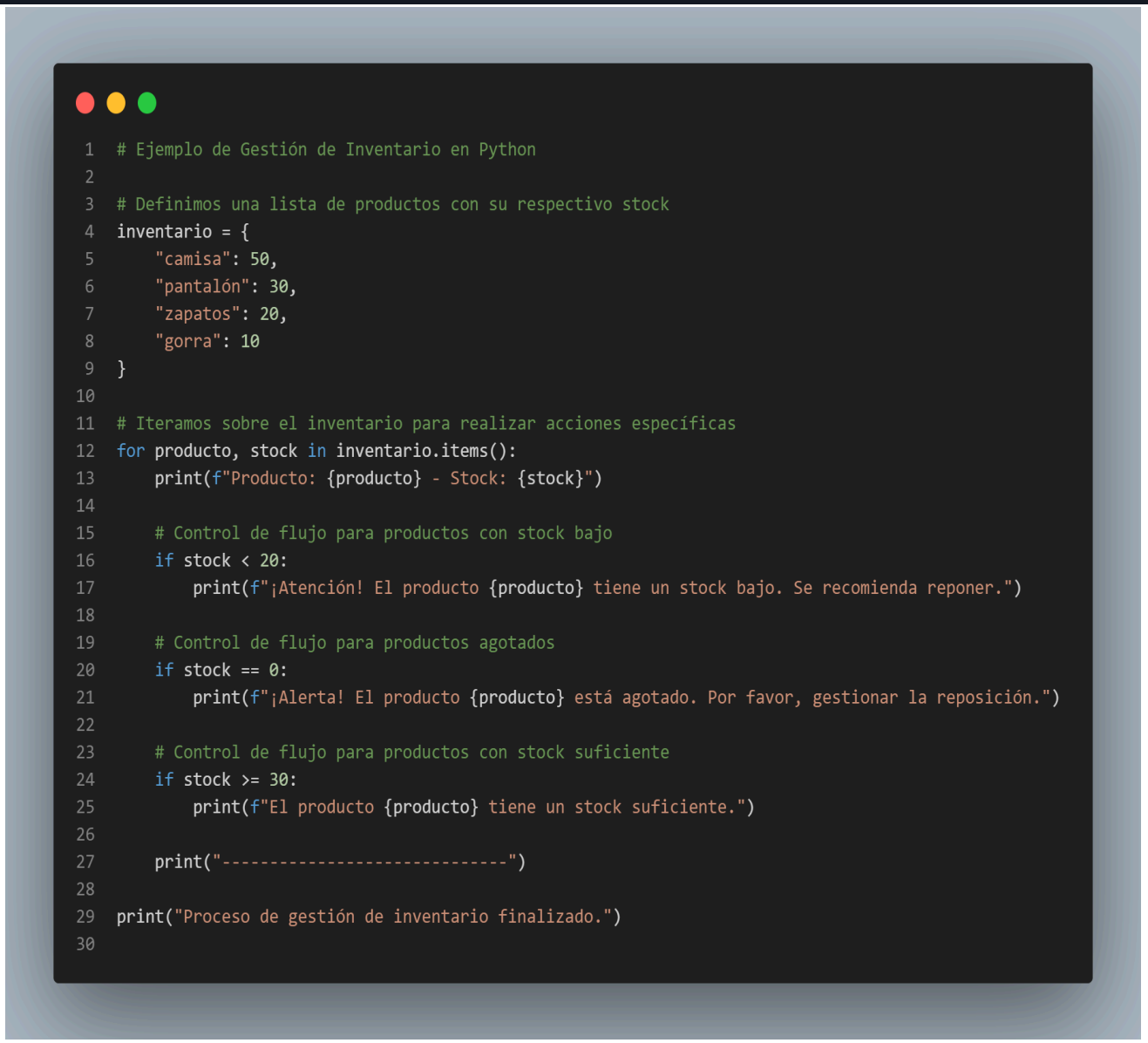
1. Imprimir los números del 1 al 20 utilizando un bucle for
2. Imprimir los números pares del 1 al 20 utilizando un bucle while
3. Imprimir los números impares del 1 al 30 utilizando un bucle for
4. Ingresar un número por teclado y mostrar su tabla de multiplicar del 1 al 10.
5. Imprimir los números del 1 al 100, para los múltiplos de 3, imprimir “múltiplo de 3”, para los múltiplos de 5 imprimir “múltiplo de 5” y para los múltiplos de 3 y de 5 imprimir “múltiplo de 3 y 5”.
6. Ingresar un número por teclado e imprimir si es número primo o no.
7. A partir de una lista vacía, utilizar un ciclo while para cargar allí números negativos del -20 al -1
8. A partir de la variable cadena, mostrar en qué posiciones aparece la letra "n" cadena = 'Hola Politécnico. Esto es una práctica programación con Python'
9. Convertir en una lista la variable "cadena" del punto 10 y luego recorrerla con un iterador
10. A partir de la siguiente lista de números, crear una lista nueva sólo si el número es divisible por 5 lista1=[18,25,29,32,35,42,50,60,63,100]
11. Imprimir los números primos del 1 al 100
12. Imprimir los números del 10 al 1 utilizando un bucle while.
13. Imprimir los números del 1 al 30, no mostrando los que son

múltiplos de 3.

14. Ingresar una cadena por teclado y contar la cantidad de letras ‘a’ que aparecen en la misma. Imprimir la cadena y la cantidad de ‘a’ que aparecen.
15. dada una lista de marca de vehículos marcas =[‘Ford’,’Renault’,’Peugeot’,’Audi’,’Toyota’], recorrer la lista con un bucle for y en caso de que la marca sea ‘Audi’ salir del bucle.

Ejemplo de Aplicación del Control de Flujo en Python: Gestión de Inventario

Imaginemos que estamos desarrollando un programa en Python para gestionar el inventario de una tienda. Utilizaremos el control de flujo para realizar diferentes acciones según las condiciones específicas de los productos en stock.



```python
# Ejemplo de Gestión de Inventario en Python

# Definimos una lista de productos con su respectivo stock
inventario = {
    "camisa": 50,
    "pantalón": 30,
    "zapatos": 20,
    "gorra": 10
}

# Iteramos sobre el inventario para realizar acciones específicas
for producto, stock in inventario.items():
    print(f"Producto: {producto} - Stock: {stock}")

    # Control de flujo para productos con stock bajo
    if stock < 20:
        print(f"¡Atención! El producto {producto} tiene un stock bajo. Se recomienda reponer.")

    # Control de flujo para productos agotados
    if stock == 0:
        print(f"¡Alerta! El producto {producto} está agotado. Por favor, gestionar la reposición.")

    # Control de flujo para productos con stock suficiente
    if stock >= 30:
        print(f"El producto {producto} tiene un stock suficiente.")

    print("------------------------------")

print("Proceso de gestión de inventario finalizado.")
```

Instancia de Autoevaluación:

En esta instancia de autoevaluación te pedimos que puedas poner en juego lo que has aprendido hasta este momento.Recordá que siempre podrás volver a la teoría presentada en esta clase o hacer las preguntas pertinentes en los encuentros sincrónicos o las tutorías presenciales.

Es importante tener en cuenta que para aprobar esta instancia deberás:

- Entregar en tiempo y forma
- Realizar un 60% de la actividad correctamente.
- Cumplir con las consignas
- Cumplir con el formato requerido.

Cierre:

Y llegamos al final de la clase cinco 🙌. En esta quinta clase hemos visto iteradores e iterables en Python, vimos control de flujo un concepto clave en todo lenguaje de programación que nos permite que nuestros algoritmos sean eficientes en la toma de decisiones y en la ejecución de tareas repetitivas. Como pueden ver en las actividades de esta quinta clase, ya estamos integrando lo aprendido en las clases anteriores, de a poco vamos viendo cómo todo se va integrando. Las clases van incrementando la práctica, la metodología que adoptamos es "Learning by doing" es decir aprender haciendo 💪. Recuerden que la mejor manera de adquirir el aprendizaje es haciéndolo con nuestras propias manos. Por tal motivo traten de realizar y llevar a ritmo cada una de las actividades prácticas que se plantean.

La semana que viene en la clase 6, vamos a comenzar a ver un tema clave en la programación y la eficiencia del código, vamos a ver **funciones**.

Les deseo que tengan una excelente semana. Los espero con mucha energía 🔋 listos para que sigamos descubriendo Python. Saludos a todos! 😀

Bibliografía Obligatoria:

- Ramírez Jiménez, Ó. (2021). *Python a fondo*. Marcombo.

Recursos adicionales:

- ***Python Comments*. (n.d.). W3Schools. Retrieved April 7, 2023, from https://www.w3schools.com/python/python_comments.asp**
- ***El tutorial de Python — documentación de Python - 3.11.3*. (n.d.). Python Docs. Retrieved April 7, 2023, from https://docs.python.org/es/3/tutorial/**
- ***PY4E-ES - Python para todos*. (n.d.). PY4E-ES - Python para todos. Retrieved April 7, 2023, from https://es.py4e.com/lessons**
- ***Python Ya*. (n.d.). Tutoriales Ya. Retrieved April 7, 2023, from https://www.tutorialesprogramacionya.com/pythonya/index.php?inicio=45**
- *Python - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN*. (2022, November 29). MDN Web Docs. Retrieved April 7, 2023, from https://developer.mozilla.org/es/docs/Glossary/Python
- (n.d.). El Libro De Python: 🏠 Inicio. Retrieved April 11, 2023, from https://ellibrodepython.com/