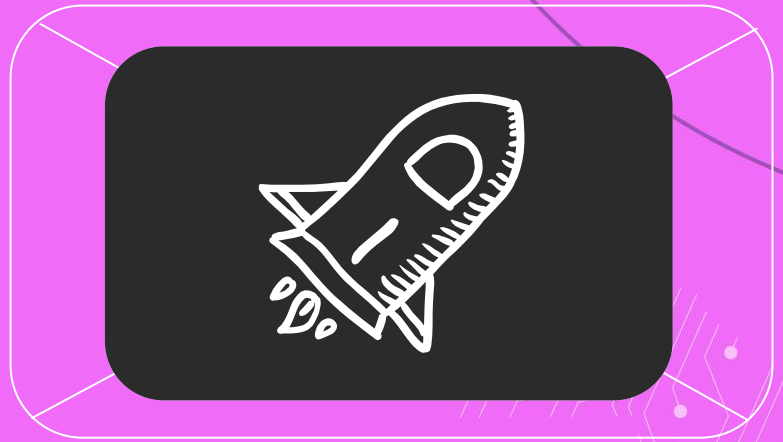
Gobierno del CHACO | Ministerio de la Producción y el Desarrollo Económico Sostenible

INFORMATORIO

Semana 3
¡Bienvenidos!



Temas de hoy

1. Python - Estructuras de control : Condicionales (if, else, elif)
2. Python - Estructuras de control : Bucles (for , while)

1

Condicionales: if-elif-else

En la semana 2 vimos, con Pseudocódigo, cómo elaborar un algoritmo que permita tomar decisiones en base a condiciones: Si, Sino Si y Sino.

Ahora bien, llevando este conocimiento a Python, les presentamos nuevamente a: if, elif y else.

Vamos a ver qué podemos hacer con ellos? 

Condicionales: if

Condicionales en Python

Los condicionales son las estructuras que nos permiten tomar decisiones en nuestros programas.

En Python, estas estructuras son muy legibles y se parecen mucho al lenguaje natural. Dejamos por acá los elementos las palabras clave:

if: Utilizado para evaluar una condición inicial.

elif: Abreviatura de "else if", se usa para evaluar condiciones adicionales si la primera if es falsa.

else: Captura cualquier caso que no cumpla ninguna de las condiciones anteriores.

```
1   color_1 = 'Blanco'
2   color_2 = 'Negro'
3   color_3 = 'Blanco'
4
5   if color_1 == color_2:
6       print('Los colores 1 y 2 son iguales.')
7   elif color_1 == color_3:
8       print('Los colores 1 y 3 son iguales.')
9   else:
10      print('Los colores 2 y 3 son iguales.')
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

on.exe c:/Users/Pc/Documents/Info/2.py
Los colores 1 y 3 son iguales.
```

¿Se acuerdan de este ejemplo? Lo utilizaremos para remarcar aspectos clave a la hora de escribir el código

Sintaxis y Puntuación:

Dos puntos (:): Se usan al final de la línea de una condición para indicar el inicio de un bloque de código.

Indentación: Muy importante en Python. Cada bloque de código bajo una condición debe estar indentado con la misma cantidad de espacios o tabulaciones.

Condicionales: Operadores

Operadores en Python

En la semana anterior vimos cuales son los operadores para comparar y también los operadores lógicos.
Hoy veremos lo mismo pero en nuestro lenguaje estrella 🐍

```
#Operadores de Comparación:
# ==: Igual a.
# !=: Distinto de.
# >: Mayor que.
# <: Menor que.
# >=: Mayor o igual que.
# <=: Menor o igual que.
```

```
# Operadores Lógicos:
# and: Y lógico.
# or: O lógico.
# not: Negación lógica.
```

En la siguiente página vamos a ver un ejemplo de nuestro ejercicio anterior pero agregando otros operadores. Y luego en clase veremos más ejemplos de operadores para que los empieces a incorporar en tu código! 🙂

Condicionales: Operadores

Operadores

```
color_1 = 'Blanco'
color_2 = 'Negro'
color_3 = 'Blanco'

if color_1 != color_2 and color_2 != color_3:
    print('Los colores 1 y 3 son iguales.')
```

Como podés ver, en este ejemplo evaluamos dos expresiones en una misma sentencia.
Por un lado, que el color 1 y 2 sean distintos "and" (en español "Y") que color 2 y 3 sean distintos.
Ambas expresiones deben ser VERDADERAS para que el código que está en la siguiente línea sea ejecutado.

Pero… ¿Qué pasa si no? 🤔

¡Muy buena pregunta! Y de seguro ya te sabes la respuesta… NADA.
Ésta claramente no es una manera óptima de escribir este ejemplo, si algunas de estas condiciones no se cumple, tenemos que evaluar alguna expresión para ver cual resultó falsa y por último una tercer sentencia por sí ninguna es verdadera. (Para probar cambiemos color_3 a 'Negro')
Quedaría algo así:

```
color_1 = 'Blanco'
color_2 = 'Negro'
color_3 = 'Negro'

if color_1 != color_2 and color_2 != color_3:
    print('Los colores 1 y 3 son iguales.')
elif color_1 == color_2:
    print('Los colores 1 y 2 son iguales.')
else:
    print('Los colores 2 y 3 son iguales.')
```

Condicionales: Tipos

Simple: Solo un “if”.

Anidados: Un “if” dentro de otro “if” como ves en el ejemplo.

Multiples condiciones: Cuando la sentencia tiene más de una expresión separados por un operador lógico.

Condicional alternativo o doble: “if” acompañado de “else”.

Condicional multiple: Se encadena un “elif” o varios para evaluar todas las alternativas posibles.

```
color_1 = 'Blanco'
color_2 = 'Negro'
color_3 = 'Negro'

if color_1 != color_2:
    if color_2 != color_3:
        print('Los colores 1 y 3 son iguales.')
    else:
        print('Los colores 2 y 3 son iguales.')
else:
    print('Los colores 1 y 2 son iguales.')
```

Ejemplo de condicionales

En este ejemplo sencillo creamos un programa que utiliza la función “input()”, de la que hablamos en clases anteriores, para pedir el ingreso de un dato, en este caso, la edad del usuario, y mediante el uso de condicionales evalúa si es mayor o menor de edad y se lo informa utilizando la función “print()”.

```
edad = int(input("Ingresa tu edad: "))
if edad >= 18:
    print("Eres mayor de edad")
else:
    print("Eres menor de edad")
```

En este segundo ejemplo, agregamos el uso de “elif” a nuestra estructura condicional, el cual nos permite evaluar la condición de “Sino Si *num* es menor a cero” para informar que el número ingresado es negativo.

```
num = int(input("Ingresa un número entero: "))
if num > 0:
    print(num, "es un número positivo")
elif num < 0:
    print(num, "es un número negativo")
else:
    print("El número es cero")
```

Ahora un ejemplo para mostrar las múltiples condiciones, en este caso, evaluamos que el número ingresado **sea múltiplo de 2 y de 3 a la vez.**

```
num = int(input("Ingresa un número entero: "))
if num % 2 == 0 and num % 3 == 0:
    print(num, "es múltiplo de 2 y de 3 a la vez")
else:
    print(num, "no es múltiplo de 2 y de 3 a la vez")
```

Ejemplo de condicionales

En este ejemplo utilizamos métodos propios de la clase string(no te preocupes, más adelante veremos que es una clase, ya que métodos vimos en la clase anterior) para evaluar el carácter ingresado por el usuario y como respuesta, decirle al usuario qué tipo de carácter es, para eso utilizamos múltiples condiciones.

👉

```python
caracter = input("Ingresa un carácter: ")
if caracter.isupper():
    print("El carácter es una letra mayúscula")
elif caracter.islower():
    print("El carácter es una letra minúscula")
elif caracter.isdigit():
    print("El carácter es un número")
else:
    print("El carácter es un carácter especial")
```

Y ahora te toca a vos practicar 🙂

2

Bucles : for - while

¿Están listos para darle una vuelta a su código y repetir hasta que se cansen? ¿Se acuerdan de “Para” y “Mientras” ?

Hoy vamos a explorar los bucles for y while (si… si… ya sé que se dan una idea), esas herramientas mágicas hacen que nuestro código haga la misma tarea una y otra vez sin quejarse.

Vamos! 🎢✨

Bucles: for - while

Bucles en Python

En Python, los bucles nos permiten repetir una secuencia de instrucciones múltiples veces. Los dos tipos principales de bucles son for y while.

for:
El bucle for se usa para iterar sobre una secuencia predefinida (como una lista, tupla, diccionario, conjunto o una cadena). Es muy útil para recorrer colecciones y ejecutar un bloque de código para cada elemento de la secuencia.

while:
El bucle while repite un bloque de código mientras una condición es verdadera.

```
1  # Iterar sobre una lista
2  numeros = [1, 2, 3, 4, 5]
3  for numero in numeros:
4      print(numero)
```

En este simple ejemplo definimos una lista de números y utilizamos una estructura de control de flujo repetitiva (otra forma de llamar a los bucles) para imprimir por consola todos los elementos en diferentes líneas.

Bucles: for

Sintaxis

La sintaxis básica de un bucle ***for*** en Python es:

```
secuencia = [1,2,3,4,5]
for elemento in secuencia:
    # bloque de código
```

- *elemento*: Es una variable que toma el valor del siguiente ítem de la secuencia en cada iteración del bucle.
- *secuencia*: Es la colección de elementos que se va a recorrer. En este caso, una lista.

Ahora bien, cuando hablamos de *for*, tenemos que mencionar que existen dos herramientas que nos van a ser de gran utilidad para generar secuencias: **range()** y **enumerate()** .

range():

Se utiliza para generar una secuencia de números, que se pueden usar para iterar en un bucle *for*. Es especialmente útil cuando necesitás un número conocido de iteraciones.

Sintaxis de range()

- range() : Puede tomar hasta tres argumentos:
- range(stop) : Genera números desde 0 hasta *stop* - 1.
- range(start, stop): Genera números desde *start* hasta *stop* - 1.
- range(start, stop, step) : Genera números desde *start* hasta *stop* - 1, incrementando por *step* (este último es opcional, por defecto es 1).

enumerate():

Añade un contador a un iterable y lo devuelve en forma de un objeto enumerado. Esto es útil cuando necesitás tanto el índice como el valor del elemento en un bucle for.

Sintaxis de enumerate()

- enumerate(iterable, start=0)
  - iterable: La secuencia que querés enumerar (por ejemplo, una lista).
  - start: El valor inicial del contador (opcional, por defecto es 0).

Bucles: for

Entonces, para repasar...

El bucle *for* en Python es una herramienta poderosa para iterar sobre secuencias y ejecutar código repetitivo de manera eficiente. Con **for, range(), enumerate()**, y la capacidad de iterar sobre diferentes tipos de secuencias, podés manejar una gran variedad de tareas en tus programas.

Ahora vamos a ver algunos ejemplos para que esto nos quede un poquito más claro...

Ejemplo de bucle: for

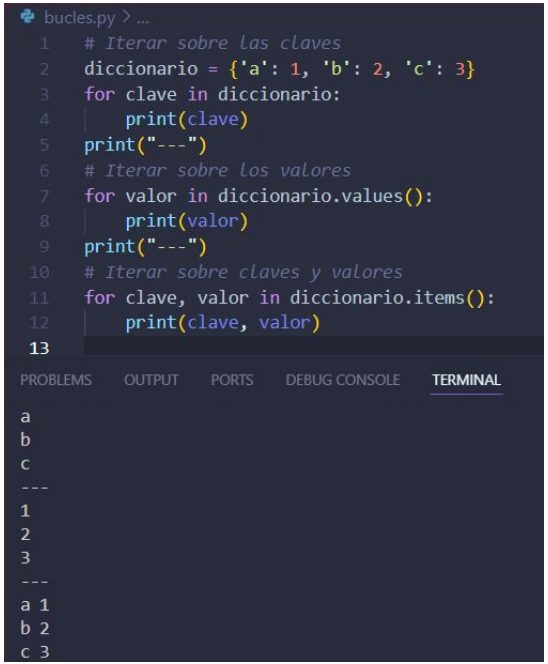
Este bucle recorre cada *letra* en la cadena *palabra* y la imprime.

El *range(5)* genera una secuencia de números del 0 al 4. El bucle *for* recorre cada número y lo imprime. Recordá que usamos la variable *i* por convención cuando usamos *for* .

Ejemplo de bucle: for

Iterar sobre un **diccionario** →
Cuando iteramos sobre un diccionario, podemos obtener las claves, los valores o ambos.

👉



¡Te lo volvemos a recordar! En mentorías podrás ver más ejemplos, ¡No te lo pierdas!

Bucles: while

El bucle ***while*** se utiliza para repetir un bloque de código mientras una condición sea **verdadera**. Este tipo de bucle es útil cuando **no** se sabe de antemano cuántas veces se debe ejecutar el bucle, ya que la ejecución se basa en una condición.

Sintaxis de while:

```
while condicion:
    # bloque de código
```

› **condicion**: Una expresión que se evalúa como *True* o *False*. Mientras sea *True*, el bloque de código se ejecutará repetidamente.

En este ejemplo, el bucle while continúa ejecutándose mientras la variable *contador* sea menor o igual a 5. En cada iteración, imprime el valor de *contador* y luego lo incrementa en 1.

Bucles: while

Uso de break y continue

Dentro de un bucle *while*, podés usar las declaraciones "**break**" y "**continue**" para controlar el flujo del bucle.

- **break**: Sale del bucle inmediatamente, independientemente de la condición.

En este ejemplo, el bucle se interrumpe cuando *contador* es igual a 3, por lo que solo se imprimen los números 1 y 2.

- **continue**: Salta el resto del bloque de código y vuelve al inicio del bucle para evaluar la condición de nuevo.

En este ejemplo, cuando *contador* es igual a 3, *continue* salta la instrucción *print()*, por lo que el número 3 no se imprime.

Bucles: while

Bucle infinito ⚠️

Un bucle ***while*** puede ser **infinito** si la condición **nunca** se vuelve *False*. Es importante asegurarse de que haya una forma de salir del bucle.

```
bucles.py > ...
1   while True:
2       respuesta = input("¿Quieres salir? (s/n): ")
3       if respuesta == 's':
4           break
5
6

PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE  TERMINAL
¿Quieres salir? (s/n): n
¿Quieres salir? (s/n): n
¿Quieres salir? (s/n): s
```

En este ejemplo, el bucle continuará ejecutándose hasta que el usuario ingrese '*s*' para salir.

Entonces decimos que:

- **while:** Ejecuta un bloque de código mientras una condición sea verdadera.
- **break:** Termina el bucle inmediatamente.
- **continue:** Salta al inicio del bucle para evaluar la condición de nuevo.
- **Bucle Infinito:** Un bucle sin una condición de terminación puede ejecutarse indefinidamente.

En caso de que te pase(esperemos que no) podes cortar la ejecución del código con Ctrl + C en la consola.

Ejemplo de bucle: while

En este ejemplo, vemos un programa que pide al usuario un número y calcula la suma de todos los números naturales del 1 hasta ese número.

```python
num = int(input("Ingresa un número: "))
i = 1
suma = 0

while i <= num:
    suma += i
    i += 1

print("La suma de los números naturales del 1 hasta", num, "es:", suma)
```

En este próximo ejemplo, vemos un programa que pide al usuario un número y luego imprime la tabla de multiplicar correspondiente a ese número del 1 al 10.

```python
num = int(input("Ingresa un número: "))
i = 1

while i <= 10:
    print(num, "x", i, "=", num*i)
    i += 1
```

Para este ejemplo tenemos un programa que pide al usuario una palabra y luego imprime la misma palabra pero con las letras en orden inverso.

```python
palabra = input("Ingresa una palabra: ")
palabra_invertida = ""

i = len(palabra) - 1

while i >= 0:
    palabra_invertida += palabra[i]
    i -= 1

print("La palabra invertida es:", palabra_invertida)
```

Ejemplo de bucle: while

En este ejemplo final, creamos un juego simple donde el usuario tiene que adivinar un número secreto entre 1 y 100.
Usamos un bucle *while* para permitir al usuario seguir adivinando hasta que acierte, y también incluimos condiciones para manejar la entrada del usuario.

Seguramente te diste cuenta que en la primer línea de código usamos las palabras reservadas *import* y *random*. Esto es parte de funciones y módulos, un tema que veremos más adelante. 😉

```python
import random

# Generar un número secreto entre 1 y 100
numero_secreto = random.randint(1, 100)
intentos = 0

while True:
    # Pedir al usuario que ingrese su adivinanza
    adivinanza = input("Introduce tu adivinanza: ")

    # Verificar si la entrada es un número
    if not adivinanza.isdigit():
        print("Por favor, introduce un número válido.")
        continue

    # Convertir la entrada a un número entero
    adivinanza = int(adivinanza)
    intentos += 1

    # Verificar la adivinanza
    if adivinanza < numero_secreto:
        print("Demasiado bajo. Intenta de nuevo.")
    elif adivinanza > numero_secreto:
        print("Demasiado alto. Intenta de nuevo.")
    else:
        print(f"¡Felicitaciones! Adivinaste el número en {intentos} intentos.")
        break

print("Gracias por jugar. ¡Hasta la próxima!")
```

Lo que vimos hoy 😉

Hasta acá pudimos ver el uso de condicionales ***if*** y sus respectivas cláusulas ***elif*** y ***else*** y el uso de bucles ***for*** y ***while***. Además, ya viste algunos ejemplos con agregados que te servirán para entender la lógica de la resolución de ejercicios con el código ya escrito en Python.

Recordá profundizar

No olvides que tenés material complementario en el campus, clases de mentorías pero sobre todo, lo que más tenes que tener, es, ¡ganas de seguir practicando y buscar por varios lugares, leer la documentación oficial de Python y ver ejemplos de fuentes confiables! 👆

Con mucha práctica vas a consolidar todo este conocimiento que vas aprendiendo clase a clase 😉

¡Nos vemos En la próxima clase!