Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Iniciación con Python

Clase 05 - “Condicionales”

Buenos Aires Ciudad

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

Clase 04.

Ruta de avance

1. Definimos los requisitos del Proyecto Integrador.
2. Menú de opciones.
3. Pedir, procesar y mostrar datos.

Clase 05.

Condicionales

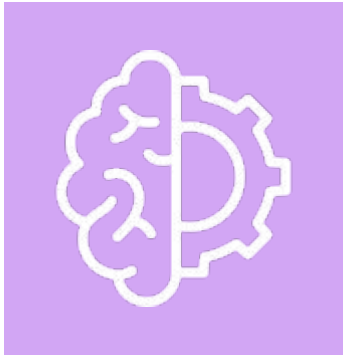
1. Cadena de caracteres.
2. Operadores lógicos.
3. Control de flujo: estructuras condicionales (if, else, elif).

Clase 06.

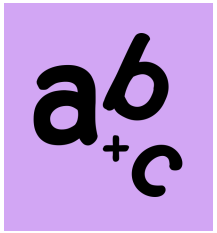
Bucles while

1. Control de flujo: bucles while.
2. Concepto y usos de los contadores.
3. Concepto y usos de los acumuladores..

Pero antes...



¡Resolvamos los “**Ejercicios Prácticos**” de la clase anterior!



Cadenas de caracteres

Cadena de caracteres

Las cadenas de caracteres (strings)no son más que secuencias de letras, números o símbolos que se juntan para formar palabras, frases o incluso datos como números de teléfono o direcciones.
Son ideales para almacenar información no numérica, como datos personales o de productos.

```
# Creación de variables del tipo String
apellido = "Gimenez"
nombre = "Ana María"        # Pueden incluir espacios en blanco.
dirección = "Mitre 123"     # Pueden incluir dígitos.
password = "$3ytR&%|"       # Pueden incluir caracteres especiales.
edad = "23"                 # Pueden tener solamente dígitos.
nada = ""                   # Esta es una cadena "nula" (vacía)
```

Cadena de caracteres

Las cadenas pueden delimitarse con comillas simples o dobles.

```
dia1 = "Lunes"   # Definición de cadenas usando comillas dobles
dia2 = 'Martes' # Definición de cadenas usando comillas simples
```

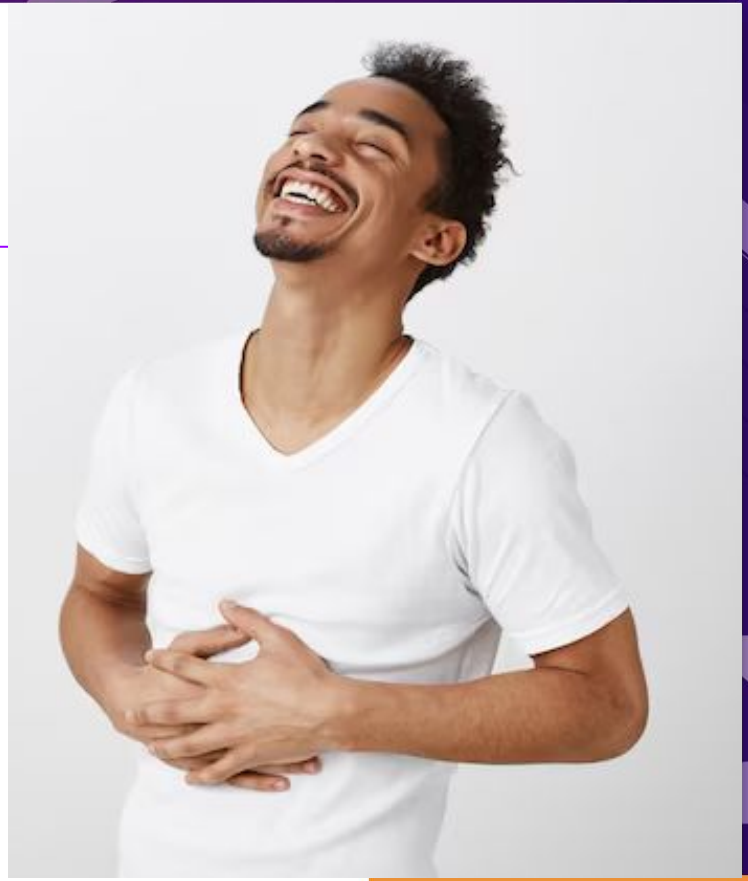
Las cadenas pueden delimitarse con comillas simples o dobles.

```
print("Mi perro 'Dinamita'") # Mi perro 'Dinamita'
print('Mi perro "Dinamita"') # Mi perro "Dinamita"
```

Operador *

Hay que tener en cuenta que en Python algunos operadores tienen una función ligeramente diferente según el contexto en que se usan. Por ejemplo,hemos visto que con * podemos multiplicar números. Pero si lo usamos con una cadena y un número, en ese orden, replicamos la cadena. ¡Probalo!

```
risa = 'ja'
carcajada = risa * 5      # jajajajaja
linea = "-" * 15          # ---------------
```



Operador +

Para unir (concatenar) dos o más cadenas se utiliza el operador + (más), tal como se vé en el ejemplo de la derecha.

```
nombre = input("Ingrese su nombre: ") # Juan
saludo = "Hola " + nombre
print(saludo)    # Hola Juan
```

Este operador "+" suma números o concatena cadenas. No funciona con tipos mixtos; para evitar errores, tenemos que usar las funciones de conversión.

```
var1 = 3 + 5          # 8 (entero)
var2 = "3" + "5"      # 35 (cadena)
var3 = 3 + "5"        # TypeError
var4 = str(3) + "5"   # 35 (cadena)
var5 = 3 + int("5")   # 8 (entero)
```

Longitud y acceso

A veces es importante saber cuántos caracteres contiene una cadena. Ese valor se lo suele denominar “longitud”, y se puede obtener muy fácilmente utilizando una función llamada len():

```
nombre = 'Talento Tech Adultos'
print(len(nombre)) # se imprime 20
```

Ojo: len() cuenta incluso los caracteres “invisibles”, como los espacios en blanco.

Se puede acceder a los elementos de la cadena utilizando subíndices:

```
cadena = "Aprendemos Python"
print(cadena[0])     # A
print(cadena[5])     # d
print(cadena[-1])    # n
print(cadena[-2])    # o
```

El primer caracter tiene subíndice cero. Si usamos subíndices negativos, se cuentan desde el final de la cadena.

Slicing (rebanado)

El slicing te permite cortar porciones de una cadena de caracteres. Para hacer slicing, usamos corchetes [ ] y dentro indicamos desde qué índice queremos empezar a cortar, y hasta qué índice queremos llegar, separados por dos puntos (:). Importante: el índice final que pongas no se incluye, es como decir: "cortame desde acá hasta acá, pero sin incluir el último caracter".

```
# Definimos una cadena
frase = "Aprender Python es divertido"

# Subcadena desde el principio hasta el índice 8 (no inclusive)
subcadena = frase[0:8]

print("Subcadena desde el inicio hasta el índice 8:", subcadena)
#Imprime Subcadena desde el inicio hasta el índice 8: Aprender
```

Slicing (rebanado)

Algo interesante del slicing es que podés omitir el índice de inicio o el índice de fin, y Python automáticamente va a asumir que querés cortar desde el principio o hasta el final de la cadena, respectivamente:

```
frase = "Aprender Python es divertido"

subcadena = frase[:8] # Subcadena desde el inicio hasta el índice 8 (sin incluirlo)
print("Subcadena desde el inicio hasta el índice 8:", subcadena)
# Imprime Aprender

subcadena = frase[9:] # Subcadena desde el índice 9 hasta el final
print("Subcadena desde el índice 9 hasta el final:", subcadena)
# Imprime Python es divertido
```

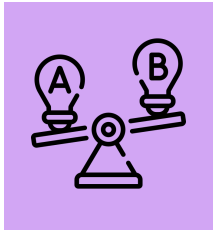
Slicing (rebanado)

Y todavía hay más. También podés hacer slicing con saltos entre caracteres, usando un tercer parámetro. Esto te permite, por ejemplo, tomar sólo cada segundo carácter de una cadena.

```
texto = "Talento Tech"
subcadena = texto[::2]
print("Subcadena obtenida:", subcadena)
# Imprime TlnoTc
```

En este ejemplo, con ::2, le indicamos a Python que debe tomar caracteres de dos en dos, a partir del primero y hasta llegar al final.

¡El slicing es una herramienta muy poderosa para cuando necesitás manipular texto en Python!.



Operadores relacionales

Decisiones

Los **operadores relacionales** en Python, que también se conocen como operadores de comparación, sirven para algo clave: comparar valores. El resultado de esta comparación siempre es un valor booleano, es decir, True (verdadero) o False (falso), dependiendo de si la condición que estás evaluando es cierta o no. Esto es súper importante, porque estos operadores son los que te permiten tomar decisiones en tus programas.

Operadores relacionales

Operador	Descripción	Ejemplo
==	Igual a: compara si dos valores son iguales.	`print(7 == 7) # True` `print(9 == 10) # False`
!=	Diferente de: compara si dos valores no son iguales.	`print(7 != 7) # False` `print(9 != 10) # True`
<	Menor que: verifica si el valor de la izquierda es menor que el de la derecha.	`print(7 < 7) # False` `print(9 < 10) # True`
>	Mayor que: verifica si el valor de la izquierda es mayor que el de la derecha.	`print(8 > 6) # True` `print(5 > 5) # False`
<=	Menor o igual que: verifica si el valor de la izquierda es menor o igual al de la derecha.	`print(8 <= 6) # False` `print(5 <= 5) # True`
>=	Mayor o igual que: verifica si el valor de la izquierda es mayor o igual al de la derecha.	`print(8 >= 8) # True` `print(4 >= 5) # False`

Comparación de cadenas

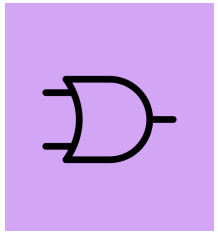
Las cadenas de texto se comparan carácter por carácter, siguiendo un orden basado en su codificación (ASCII o Unicode). Esto significa que Python va a ir chequeando cada carácter de la cadena, de izquierda a derecha, hasta que encuentre una diferencia o hasta que una de las cadenas se termine.

```
print("apple" < "banana")     # True
print("apple" > "Apple")      # True
print("apple" < "apples")     # True

print("python" == "python")   # True
print("Python" != "python")   # True

print("A" < "a")              # True
print("z" > "Z")              # True
```

Como ves, Python no sólo se fija en si las letras son iguales, sino también en si son mayúsculas o minúsculas, o si una cadena tiene más o menos caracteres. ¡Esto es súper útil cuando necesitás comparar nombres, palabras o cualquier tipo de texto en tus programas!



Operadores lógicos

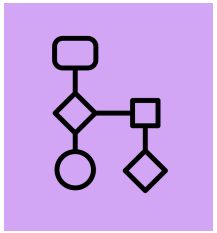
```
not a
```

Operadores lógicos

Se utilizan los operadores lógicos para tomar decisiones basada en múltiples condiciones. Los operadores lógicos utilizados en Python son and, or y not.

Operador	Descripción	Ejemplo
and	Devuelve True si **ambos** operandos son True	`a and b`
or	Devuelve True si **alguno** de los operandos es True	`a or b`
not	Devuelve True si el operandos False, y viceversa	`not a`

a y b son expresiones lógicas. Cada una de ellas puede ser verdadera o falsa.
Si a y/o b son valores numéricos, se tratan como True o False según su valor sea cero o no.



Estructuras de control

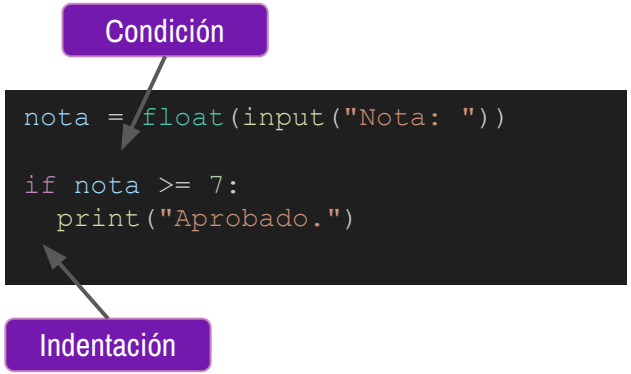
Estructuras

Las estructuras de control en Python son clave porque te permiten tener un programa que no simplemente corre una lista de instrucciones de manera secuencial, sino que puede adaptarse a distintas situaciones. Dependiendo de las condiciones que se encuentren, podés hacer que el programa ejecute una parte del código o repita una acción varias veces. Estas estructuras son lo que le da vida a los programas, haciéndolos más dinámicos e interactivos.

Condicionales | if

Las estructuras condicionales tienen como objetivo ejecutar un bloque de instrucciones u otro en base a una condición que puede ser verdadera o falsa. La palabra clave asociada a esta estructura es if.

Si la condición es True se ejecuta el bloque dentro del if. Luego, independientemente del valor de verdad de la condición.



Condicionales | if

Esa pequeña sangría al principio de la línea no es sólo por estilo; le dice a Python que esas líneas de código forman parte del bloque del if. Vamos a poner otro ejemplo para que quede bien claro:

```
edad = 18

if edad >= 18:
  print("Sos mayor de edad.")
```

En este caso, estamos verificando si alguien tiene 18 años o más. Si la condición es True, Python va a imprimir "Sos mayor de edad". Si no, simplemente va a seguir con el resto del programa, sin hacer nada especial con esa línea.

Condicionales | if...else

Ya vimos cómo if le da la capacidad a tu programa de tomar decisiones cuando una condición es verdadera.

Pero, ¿qué pasa cuando esa condición no se cumple? Ahí es donde entra en juego else. Mientras que if se encarga de ejecutar un bloque de código solo si algo es True, else viene al rescate cuando esa condición resulta ser False.

```
if edad >= 18:
  # Bloque de instrucciones que se
  # ejecuta si la condición es verdadera
  print("Puedes pasar.")
else:
  # Bloque de instrucciones que se
  # ejecuta si la condición es falsa
  print("No admitido.")
```

Preguntamos si edad es mayor o igual que 18. Si es verdad, mostramos “Puedes pasar.” y el programa continúa sin ejecutar el bloque del else. Si es falso, mostramos “No admitido.” y obviamos el bloque del if.

Condicionales | if...else

```
edad = int(input("Ingresá tu edad: "))
tiene_licencia = input("¿Tenés licencia de conducir? (S/N): ")

# Verificamos si la persona puede conducir
if edad >= 18 and tiene_licencia == "S":
   print("Podés conducir.")
else:
   print("No podés conducir.")
```

Si alguna de las dos condiciones no se cumple (por ejemplo, si la persona no tiene 18 años o si no tiene licencia), el programa saltará al else y mostrará "No podés conducir.".

Condicionales anidados

Hasta ahora, vimos cómo if y else te permiten tomar decisiones en tu programa, pero ¿qué pasa cuando necesitás evaluar más de una condición a la vez? Acá es donde entran en juego las estructuras condicionales anidadas.

Este concepto puede sonar un poco complicado al principio, pero en realidad es bastante intuitivo: se trata de poner una condición dentro de otra. Esto te permite manejar situaciones donde la decisión que vas a tomar depende de una serie de condiciones que deben evaluarse en secuencia.

```
if variable condición valor:
   print("")
   if variable condición valor:
       print("")
   else:
       if variable condición valor:
           print("")
       else:
           print("")
else:
   print("")
```

Condicionales anidados

Veamos un ejemplo:

Vamos a escribir un programa que evalúa una calificación y decide qué mensaje imprimir en base al resultado. Primero, verificamos si la persona aprobó. Si es así, después evaluamos en qué rango de calificaciones se encuentra para dar un feedback más preciso.

```
nota = 85

if nota >= 60:
   print("¡Aprobaste!")
   if nota >= 90:
       print("¡Excelente calificación!")
   else:
       if nota >= 75:
           print("Muy buen trabajo.")
       else:
           print("Buen esfuerzo, pero hay margen de
mejora.")
else:
   print("No alcanzaste la calificación mínima para
aprobar.")
```

Condicionales anidados

En el ejemplo anterior, primero chequeamos si la calificación es suficiente para aprobar. Si lo es, se imprime el mensaje de aprobación, pero ahí no termina. Dependiendo de qué tan alta sea la nota, el programa va a imprimir un mensaje adicional. Si la nota es 90 o más, el programa va a decir que es excelente. Si está entre 75 y 89, va a felicitar al estudiante por el buen trabajo. Si es menor, va a sugerir que aún hay margen para mejorar.

Los condicionales anidados son súper útiles, pero a veces la estructura asociada (indentaciones varias) se empieza a convertir en algo difícil de entender, aún para el propio programador. Afortunadamente, en algunos casos es posible reemplazar esta “cascada” de condicionales con otra estructura que provee Python: el if…elif…else.

Condicionales | if...elif...else

Este es un script que hace lo mismo que el anterior, pero usando elif.
Ahora el código se ve mucho más ordenado y fácil de seguir. Pero ¿qué es exactamente lo que hace elif?

Básicamente, es una forma de agregar más condiciones en una estructura if, pero sin necesidad de anidar los bloques de código uno dentro del otro.

```
nota = 85

if nota >= 60:
   print("¡Aprobaste!")
   if nota >= 90:
       print("¡Excelente calificación!")
   elif nota >= 75:
       print("Muy buen trabajo.")
   else:
       print("Buen esfuerzo, pero hay margen de
mejora.")
else:
   print("No alcanzaste la calificación mínima para
aprobar.")
```

¡Vamos a la práctica! 

Ejercicios Prácticos

Optativos | No entregables

Control de inventario de una tienda de videojuegos

Imaginá que estás ayudando a una tienda de videojuegos a organizar su inventario. El dueño te pide que escribas un programa que verifique si hay stock suficiente de un videojuego y, si no hay, que avise que hay que reponerlo.

El programa debería pedirle al usuario que ingrese la cantidad actual en stock y, en base a esa cantidad, mostrar si se necesita hacer un nuevo pedido o no.

Ejercicios Prácticos

Optativos | No entregables

Compra con descuentos

Escribe un programa en Python que solicite al usuario el monto total de la compra y la cantidad de artículos que está comprando. El programa debe determinar el descuento aplicable según las siguientes reglas:

- Si la cantidad de artículos comprados es mayor o igual a 5 y el monto total es mayor a $10000, aplica un descuento del 15%.
- Si la cantidad de artículos comprados es menor a 5 pero mayor o igual a 3, aplica un descuento del 10%.
- Si la cantidad de artículos comprados es menor a 3, no se aplica descuento.

Al final, el programa debe imprimir el monto total de la compra después de aplicar cualquier descuento o simplemente el monto original si no hay descuento.