

Esta clase va a ser

- grabada

## PROYECTO FINAL

# Proyecto final

El Proyecto final se irá construyendo a partir de **las pre-entregas** que se realicen en clases puntuales según los temas trabajados hasta ese punto. Estas están conformadas por las actividades prácticas relacionadas con el Proyecto final que se encuentran en la Guía de Actividades y que se realizan de forma asincrónica.

Se debe subir a la plataforma la última clase del curso. En caso de no hacerlo **tendrás 10 días** a partir de la finalización del curso para cargarlo en la plataforma. Pasados esos días el botón de entrega se inhabilitará. Para saber más sobre el sistema de entregas puedes chequear la **hoja de ruta** del curso.



# ¡Importante!

Las pre entregas del proyecto final se deben cargar hasta **7 días** después de finalizada la clase. Y contarás con **10 días** una vez finalizado el curso para entregar tu proyecto final.  
Te sugerimos llevarlos al día.



MARTES 25/01 19:00HS - VALORACIÓN REQUERIDA

## 2. Metodologías de diseño y UX research

DESAFÍO - EXPIRA EL MARTES 01/02/2022 23:59HS

Definiendo el problema, objetivo y solución

Se bloqueará en 7 días y 11:48hs luego no se podrá entregar.

↑ Entregar



¿Cuál es nuestro  
Proyecto final?



PROYECTO FINAL

# Proyectos de nuestros estudiantes

En [este archivo](#) podrán ver el Proyecto final de Johannes Pérez, estudiante de este curso de comisiones anteriores.

**¡Esperamos que les resulte inspirador!**

Además, les compartimos el [perfil de LinkedIn](#) de Johannes.





## PROYECTO FINAL

Entrega	Fecha
1° entrega	N° de clase: 11
2° entrega	N° de clase: 15
3° entrega	N° de clase: 21
Proyecto Final	N° de clase: 25

Clase 01. PYTHON

# Números y cadenas de caracteres

# Temario

00

## Introducción a programar con Python

- ✓ Programación
- ✓ PYTHON

01

## Números y cadenas de caracteres

- ✓ [Números](#)
- ✓ [Cadenas de texto](#)
- ✓ [Variables](#)
- ✓ [String](#)

02

## Listas y Tuplas

- ✓ Listas
- ✓ Funciones
- ✓ Tuplas



# Objetivos de la clase

- **Implementar** operaciones básicas y avanzadas de números
- **Asignar** variables
- **Implementar** operaciones numéricas con variables
- **Operar** con cadena de caracteres
- **Reconocer** funcionalidades de cadenas de caracteres

Según lo visto en la clase anterior...

# ¿Pudiste instalar Visual Studio Code?

¿Crear una carpeta?

¿Crear un archivo .py?

¿Ejecutar un archivo Python?

# Primeros pasos

Vamos a usar una instrucción Python que se llama `print`. Se utiliza para mostrar información en la pantalla de la computadora. El siguiente ejemplo imprimirá un mensaje en la pantalla:

```
print("Hola, mundo!")
```

# Primeros pasos

Podemos guardar texto en una **variable**.

Una variable es un espacio en la memoria de la computadora que se utiliza para almacenar datos que pueden ser utilizados en un programa.

En Python, se pueden crear variables simplemente asignando un valor a un nombre.

El siguiente ejemplo asignará el texto "Juan" a la variable **nombre**:

```
nombre = "Juan"
```

# Primeros pasos

También vamos a usar la instrucción `input` se utiliza para solicitar información del usuario. Cuando se llama a `input()`, el programa se detiene y espera a que el usuario escriba algo. Lo que el usuario escriba se puede almacenar en una variable para su uso posterior.

El siguiente ejemplo solicitará al usuario su nombre y lo almacenará en la variable `nombre`:

```
nombre = input("Ingrese su nombre: ")
```



# Tu primer programa...

en Visual Studio Code.

Duración: **10 minutos**



ACTIVIDAD EN CLASE

# Mi primer programa en Visual Studio Code

1. Crear una carpeta llamada Coderhouse y abrirla con Visual Studio Code.
2. Crear un nuevo archivo, con el nombre: `clase_01.py`
3. Ahora, en el código: mostrar por pantalla un mensaje de bienvenida
4. Preguntar al usuario por su nombre y luego por su edad, guardando los valores en dos variables.
5. Mostrar por pantalla en un solo mensaje: "nombre" tiene "edad" años.  
Siendo nombre y edad las variables que ya tienen un valor.

# Números

**<int>   <float>   <complex>**



# Tipos de número

# Números en Python

Los números de Python están relacionados con los números matemáticos, pero están sujetos a las limitaciones de la representación numérica en las computadoras. Python distingue entre enteros, números de punto flotante y números complejos.



# Enteros

Los números enteros son aquellos que **no tienen decimales**, tanto positivos como negativos (además del cero). En Python se pueden representar mediante el **tipo int** (de integer, entero).

A diferencia de otros lenguajes de programación, los números de tipo int en Python 3 pueden ser pequeños o grandes, no tienen límite alguno.

Ejemplos:

1  
2  
525  
0  
-817

# Reales

Los **números reales** son los que tienen **decimales**, en Python se expresan mediante el tipo **float**. Desde Python 2.4 cuenta con un nuevo tipo Decimal, para el caso de que se necesite representar fracciones de forma más precisa.

Ejemplos:

0.270  
-12.1233  
989.87439124387  
-74.9349834  
.12

# Complejos

Los números complejos son una extensión del sistema numérico que permite la existencia de raíces cuadradas de números negativos.

Los números complejos se pueden crear utilizando la letra "j" para representar la unidad imaginaria. Por ejemplo,  $z = 3 + 4j$  crea un número complejo con una parte real de 3 y una parte imaginaria de 4.

En Python, este tipo se llama **complex**. Es muy probable que no lo vayas a necesitar nunca.

Ejemplo:

2, 1j  
-41, 832j  
88, 23254j

# Operaciones numéricas

# Operaciones numéricas en Python

En programación y en matemáticas, los operadores aritméticos son aquellos que manejan los datos de tipo numérico, es decir, permiten la realización de operaciones matemáticas (sumas, restas, multiplicaciones, etc.).

El resultado de una operación aritmética es un dato aritmético, es decir, si ambos valores son números enteros el resultado será de tipo entero; si alguno de ellos o ambos son números con decimales, el resultado también lo será.

## OPERADORES ARITMÉTICOS EN PYTHON

Operación	Operador	Ejemplo
Suma	+	3 + 5
Resta	-	4 - 1
Multiplicación	*	3 * 6
Potencia	**	3 ** 2
División (cociente)	/	15.0 / 2.0
División (parte entera)	//	15.0 // 2.0
División (resto)	%	7 % 5



# Precedencia o jerarquías


# Precedencia de los operadores

Al igual que ocurre en matemáticas, en programación también tenemos una **prioridad en los operadores**.

Esto significa que si una expresión matemática es precedida por un operador y seguido de otro, el operador más alto en la lista debe ser aplicado por primera vez.


# Precedencia

El orden normal de las operaciones es de izquierda a derecha, evaluando en orden los siguientes operadores:

- 
1. Términos entre paréntesis.
  2. Potenciación y raíces.
  3. Multiplicación y división.
  4. Suma y resta.

# Precedencia

En el lenguaje de programación de Python se representan los operadores con el siguiente orden:

- 
1. `()`
  2. `**`
  3. `*` `/` `%` `//`
  4. `+` `-`



# Precedencia



¿Cuál es el orden con el que Python va a resolver este cálculo?

```
((4 + 8) / 2 * 5) ** 2 - (9 + 3) / 2
```

# Cadenas de texto

`<str>`

# Cadenas de texto en Python

Las cadenas (o strings) son un **tipo de datos compuestos por secuencias de caracteres que representan texto**.

Estas cadenas de texto son de tipo `str` y se delimitan mediante el uso de comillas simples o dobles.

Ejemplo:

```
"Esto es una cadena de texto"
```

```
'Esto también es una cadena de texto'
```

# Cadenas de texto en Python

En el caso de que queramos usar comillas (o un apóstrofe) dentro de una cadena tenemos distintas opciones. La más simple es encerrar nuestra cadena mediante un tipo de comillas (simples o dobles) y usar el otro tipo dentro de la cadena. Otra opción es usar en todo momento el mismo tipo de comillas, pero usando la barra invertida (\) como carácter de escape en las comillas del interior de la cadena para indicar que esos caracteres forman parte de la cadena.

## Ejemplos:

```
"Esto es un 'texto' entre comillas dobles"
```

```
'Esto es otro "texto" entre comillas simples'
```

```
"Esto es otro \"texto\" todo en comillas dobles"
```

```
'Esto otro \'texto\' todo en comillas simples'
```



# Print

# Print

```
1 print("Esto es una cadena de texto")
2 print("Esto también es una cadena de texto")
3 print(4)
```

## ¿Para qué sirve?

La forma correcta de mostrar cadenas de texto (u otros tipos de datos) por una terminal es utilizando la función llamada **print** (imprimir).

Se indica lo que se desea mostrar por pantalla entre paréntesis.

# Ventajas

Usar **print** tiene sus ventajas. Por ejemplo, nos deja mostrar por pantalla caracteres especiales, como tabulación o saltos de línea.

**Ejemplo:**

```
print("Otra cadena\ncon salto de línea")
```

```
Una cadena  
con salto de línea
```

# Print

Si quisiéramos imprimir el directorio de una carpeta: sería de la siguiente forma:

```
print("C:\nombre\directorio")
```

Pero va a tomar el `\n` como carácter especial para salto de línea. Para poder ignorar estos caracteres especiales Python debemos escribir una `r` delante de lo que se va a imprimir, y Python automáticamente lo interpretará para no tomar en cuenta el carácter de escape `\n`.

```
print(r"C:\nombre\directorio")
```

# Print

Otra funcionalidad que tiene es permitir mostrar una cadena en distintas líneas, de forma que con un solo print se muestran varias líneas de cadenas.

Para lograrlo tenemos que pasarlo entre tres comillas dobles, o tres comillas simples.

## Ejemplo:

```
print("""una cadena  
otra cadena  
otra cadena más  
""")
```

# Print

La barra invertida nos sirve para ver el texto en el código de la forma que nosotros queramos, sin que se altere la salida.

## Ejemplo:

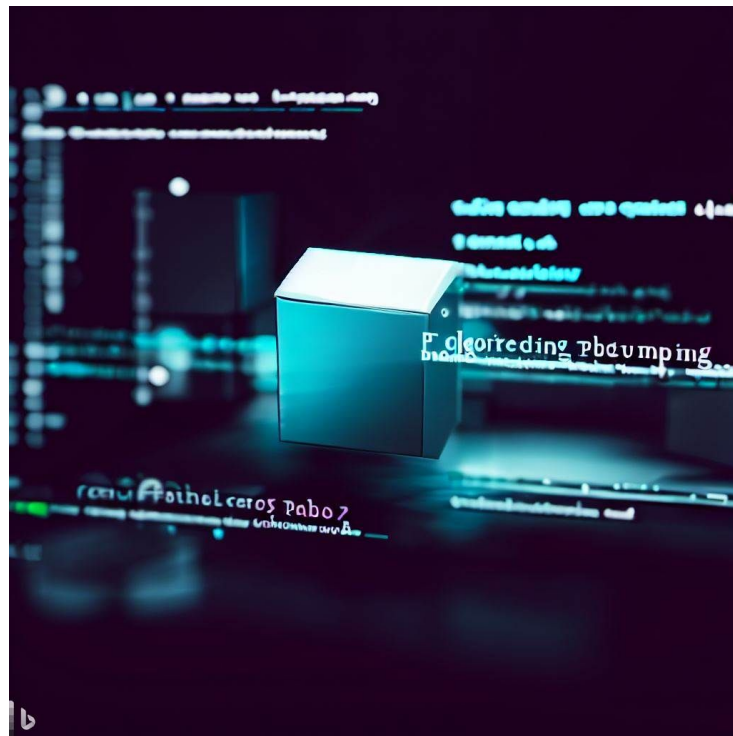
```
print("\n  
Esta es una línea \n  
de escritura.\n  
Y esta es otra.")
```

# Variables

# Variables

En algunos lenguajes de programación, las variables se pueden entender como "cajas" en las que se guardan los datos.

Pero en Python las variables son **"etiquetas"** que **hacen referencia a los datos** (que se guardan en unas "cajas" llamadas **objetos**).

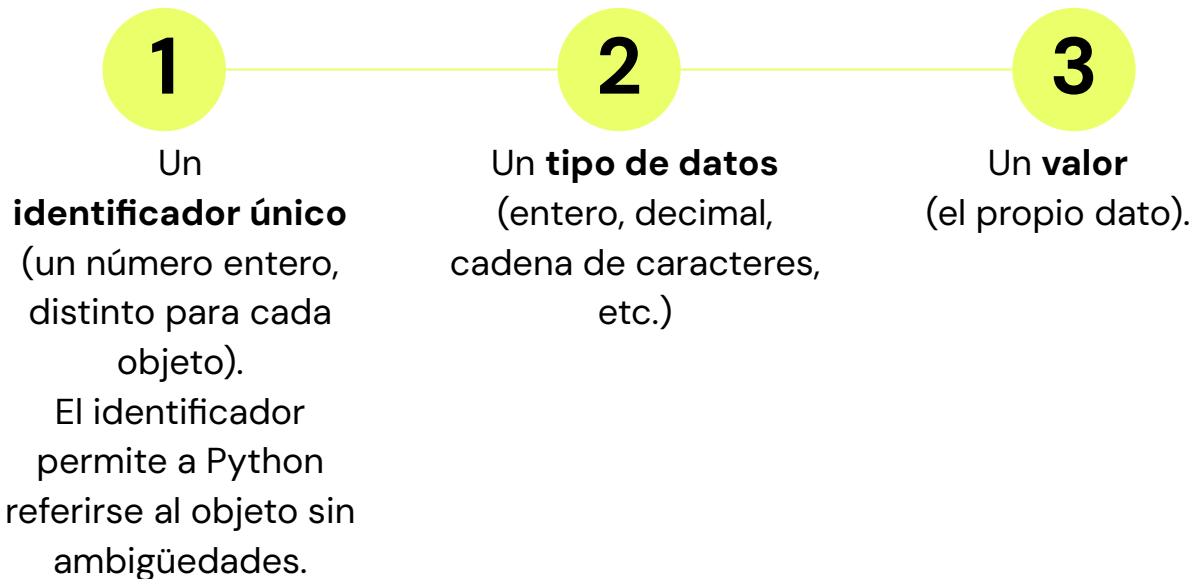




# Variables en programación

Por cada dato que se crea en un programa, Python crea un **objeto** que lo contiene.

Cada objeto tiene:





PARA RECORDAR

# Variables en Python

Las variables en Python no guardan los datos, sino que son simples nombres para **poder hacer referencia a esos objetos.**

# Variables en programación

En Python, si escribimos la instrucción:

```
a = 2
```

Se crea el objeto 2.

Ese objeto tendrá un identificador único que se asigna en el momento de la creación y se conserva a lo largo del programa.

Se asocia el nombre "a" al objeto número entero 2.



# Definir variables



# Definir una variable

Siempre se escribe a la izquierda de la igualdad, de lo contrario, Python generará un mensaje de error:

❌ `2 = mi_variable`

Para mostrar el valor de la variable hay que imprimir su nombre.

✨ `mi_variable = 2`  
`print(mi_variable)`



# Definir una variable

Si una variable no se ha definido previamente, al escribir su nombre o imprimir la variable generará un error:

```
x = -10
```

```
❌ y
```



# Definir una variable


Una variable puede almacenar números, texto o estructuras más complicadas (que se verán más adelante). Si se va a almacenar texto, debe escribirse entre comillas simples (') o dobles ("), que son equivalentes. A las variables que almacenan texto se les suele llamar cadenas (de texto).

```
mi_auto = "ford"  
print(mi_auto)
```




# Definir una variable

Si no se escriben comillas, Python supone que estamos haciendo referencia a otra variable (que, si no está definida, genera un mensaje de error):

 `mi_auto = ford`

 `ford = "Ford Mustang"`

 `mi_auto = ford`

 `print(mi_auto)`





PARA RECORDAR

# Buenas prácticas

Aunque no es obligatorio, se recomienda que el nombre de la variable esté relacionado con la **información que se almacena en ella para que sea más fácil entender el programa.**



PARA RECORDAR

# Buenas prácticas

Si el programa es trivial o mientras se está escribiendo un programa, esto no parece muy importante, pero si se consulta un programa escrito por otra persona o escrito por uno mismo hace tiempo, **resultará mucho más fácil entender el programa si los nombres están bien elegidos.**



# Veamos un ejemplo en vivo

El nombre de una variable debe empezar por una letra o por un guion bajo (\_) y puede seguir con más letras, números o guiones bajos. Los nombres de variables no pueden incluir espacios en blanco.

Se recomienda que sea todo con minúsculas, separadas las palabras con guiones bajos. Este estilo se llama **snake\_case**.

😞 **FecNac** = "27 de octubre de 1997"

😄 **fecha\_de\_nacimiento** = "27 de octubre de 1997"



# Veamos un ejemplo en vivo

Los nombres de las variables pueden contener mayúsculas, pero ten en cuenta que **Python distingue entre mayúsculas y minúsculas** (en inglés se dice que Python es case-sensitive).

```
nombre = "Pepito Conejo"  
Nombre = "Numa Nigerio"  
nomBre = "Fulanito Mengáñez"  
nombre
```

# INPUT



# Input

En Informática, la "**entrada**" o **input** de un programa son los **datos** que llegan al programa desde el exterior. Actualmente, el origen más habitual es el teclado.

Python tiene una función llamada **input()** la cual permite obtener texto escrito por teclado. Al llegar a la función, el programa se detiene esperando que se escriba algo y se pulse la tecla **Intro**.

```
nombre = input()
```



# Input

Otra solución, más compacta, es aprovechar que a la función **input()** se le puede enviar un argumento que se escribe en la pantalla (sin añadir un salto de línea):

**Ejemplo:**

```
nombre = input("¿Cómo te llamas? ")
```



# Input

## Conversión de tipos

De forma predeterminada, la función `input()` convierte la entrada en una cadena, aunque escribamos un número. Si intentamos hacer operaciones, se producirá un error. Si se quiere que **Python interprete la entrada como un número entero, se debe utilizar la función `int()`** de la siguiente manera:

```
nombre = int(input("¿Qué edad tienes?"))
```



# Operaciones aritméticas con variables

# Operaciones aritméticas con variables



Podemos utilizar todos los operadores aritméticos antes vistos en las variables numéricas. Algunos ejemplos:

`a = 2`

`b = 3`

`a + b`



`a = 5`

`b = 2`

`a * b`



`a = 35`

`b = 7`

`a / b`



`a = 3`

`b = 2`

`a ** b`



# Operaciones aritméticas con variables



Podemos utilizar todos los operadores aritméticos antes vistos en las variables de **string** también. A la suma de cadenas de caracteres la llamaremos **concatenación**. Algunos ejemplos:

```
cadena = "Python"
```

```
cadena * 2
```



```
cadena = "Python"
```

```
otra_cadena = "Hola!"
```

```
otra_cadena + cadena
```





# Break

¡10 minutos y volvemos!

# Longitud de strings

## len()

# Longitud de strings

Python nos da una función llamada **len**. Esta función nos permite saber cuál es la longitud de un string.

**Ejemplo de len():**

```
palabra = "Python"  
len(palabra)
```

```
frase = "Hola, ¿cómo están? ¡Yo bien! "  
len(frase)
```

# Indexación de strings

# Indexación de las cadenas de texto

Cada uno de los caracteres de una cadena (incluidos los espacios) tiene asignado un índice. Este índice nos permite seleccionar su carácter asociado haciendo referencia a él entre corchetes ([ ]) en el nombre de la variable que almacena la cadena.





# Indexación de las cadenas de texto

Si consideramos el orden de izquierda a derecha, el **índice empieza en 0 para el primer carácter**, etc. También se puede considerar el **orden de derecha a izquierda**, en cuyo caso al último carácter le corresponde el índice -1, al penúltimo -2 y así sucesivamente.



# Indexación de las cadenas de texto

Este método es útil si por ejemplo queremos acceder a caracteres en las últimas posiciones de una cadena con muchos caracteres de la cual no conocemos su longitud.

```
cadena = 'Python'  
cadena[0] → 'P'  
cadena[-1] → 'n'
```

Caracteres :	P	y	t	h	o	n
Índice :	0	1	2	3	4	5
Índice inverso :	-6	-5	-4	-3	-2	-1



INDEX

# Slicing

# Rebanar strings (slicing)

Otra función de las cadenas que podemos usar, es seleccionar solamente una parte de las cadenas.

Para ello se usa la notación **[inicio : fin : paso]** también en el nombre de la variable que almacena la cadena

# Rebanar string (slicing)

## Inicio

Es el índice del primer carácter de la porción de la cadena que queremos seleccionar.

## Fin

Es el índice del último carácter no incluido de la porción de la cadena que queremos seleccionar.

## Paso

Indica cada cuantos caracteres seleccionamos entre las posiciones de inicio y fin.

## Ejemplo

```
cadena = "Python"
```

```
cadena[0:4:1]  
'Pyth'
```

```
cadena[2:6:2]  
'to'
```



# Desafío String

Genera una nueva variable

Duración: **7 minutos**



ACTIVIDAD EN CLASE

# Desafío String

Partiendo de:

```
cadena_1 = "versátil"  
cadena_2 = "Python"  
cadena_3 = "es un lenguaje"  
cadena_4 = "de programación"
```

Crea una nueva cadena con el siguiente contenido:

**"Python es un lenguaje de programación versátil"**

Nota: Utiliza el operador + para unir las cadenas (concatenación)



**CODERHOUSE**



# Desafío números

De manera individual desarrollarán un programa que permita calcular el promedio final de puntos de los equipos de fútbol en un torneo.

Duración: **12 minutos**





ACTIVIDAD EN CLASE

# Desafío números

## Descripción de la actividad.

En nuestro trabajo, nos solicitan desarrollar un programa que permita calcular el promedio final de los equipos de futbol en un torneo. Para ello, debemos considerar tres aspectos:

- ✓ Jugaron 20 partidos durante el torneo.
- ✓ Los partidos poseen diferente puntaje según el resultado, los partidos ganados 3 puntos, partido empatado 1 punto, partido perdido 0 puntos.
- ✓ El promedio final resulta de la cantidad de puntos totales divididos la cantidad de partidos



ACTIVIDAD EN CLASE

# Desafío números

## Descripción de la actividad.

La cantidad de partidos que debemos considerar a un equipo para el ejemplo se detallan a continuación:

partidos\_ganados 8

partido\_empatados 7

partido\_perdidos 5

**Importante:** Cada una de las cantidades de partidos ganados, empatados o perdidos debe solicitarse al usuario utilizando la función **input()**.



# #Codertraining

Te invitamos a revisar la [Guía de Ejercicios complementarios](#), donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.

Actividad extra



# Desafío Slicing

Se tiene una cadena de texto, pero al revés. Al parecer contiene el nombre de un alumno, la nota de un examen y la materia.

```
cadena = "acitametaM ,5.8 ,otipeP ordeP"
```

1. Dar vuelta la cadena y asignarla a una variable llamada **cadena\_volteada**. Para devolver una cadena dada vuelta se usa el tercer índice negativo con slicing: **cadena[::-1]**
2. Extraer nombre y apellido, almacenarlo en una variable llamada **nombre\_alumno**



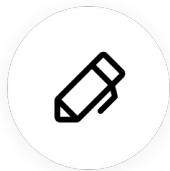
## Actividad extra



3. Extraer la nota y almacenarla en una variable llamada nota.
4. Extraer la materia y almacenarla en una variable llamada materia.
5. Mostrar por pantalla la siguiente estructura, usando la concantenación de cadenas:

**NOMBRE APELLIDO ha sacado un NOTA en  
MATERIA**





# Mi primer programa en Python

## Consigna

- ✓ Trabajas en Coderhouse y te piden crear un programa que calcule la nota final de estudiantes del curso de Python. La nota final se calcula basándonos en tres notas previas de las cuales, cada una corresponde un porcentaje distinto de la nota final. Los porcentajes se detallan a continuación:

Los porcentajes asociados que debemos considerar de cada nota se detallan a continuación:

- ✓ nota\_1 cuenta como el 20% de la nota final
- ✓ nota\_2 cuenta como el 30% de la nota final
- ✓ nota\_3 cuenta como el 50% de la nota final



# Mi primer programa en Python

## Aspectos a incluir

- ✓ Tener en cuenta los temas vistos en la clase 1: números, print, input, variables, operaciones matemáticas, cadena de texto.
- ✓ Los datos deben guardarse en variables y deben ser dinámicos por medio de input.



## PARA RECORDAR

En un **promedio pesado** o **ponderado** **no todos los valores tienen el mismo "peso" o valor.**

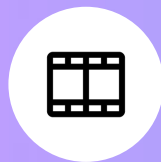
**El promedio entre 3 y 10 es:**  $(1.3 + 1.10) / 2$ , este es el promedio tradicional donde todos los valores tienen un peso de 1.

**Promedio pesado entre 3 y 10 es:**  $(13.3 + 2.10) / 15$ , aquí vemos que el peso de 3 es 13, y el peso del 10 es 2, por lo que el 3 es más importante, se divide por la suma de los pesos.

Este recordatorio te ayudará en la resolución de la actividad.







**¿Quieres saber más?**  
**Te dejamos material  
ampliado de la clase**



MATERIAL AMPLIADO

# Recursos multimedia

- ✓ [Variables Numéricas](#) | Nicolás Perez
- ✓ [Operaciones](#) | Nicolás Perez
- ✓ [Variables de Texto](#) | Nicolás Perez
- ✓ [EjemplosClase](#)

Disponible en nuestro repositorio.

# Resumen de la clase hoy

- ✓ Números
- ✓ Strings
- ✓ Print
- ✓ Variables
- ✓ Index & Slicing



**Opina y valora**  
**esta clase**

¿Preguntas?

**Muchas gracias.**

**#DemocratizandoLaEducación**