

Esta clase va a ser

- grabada



COMISIÓN N°####

# Presentación del equipo

✓ Profesor/a responsable: Juan Pérez

✓ Coordinador/a: Juan Pérez

✓ Tutores y tutoras:

○ Juan Pérez

○ ...

○ ...

○ ...

○ ...

○ ...

○ ...

○ Juan Pérez

○ ...

○ ...

○ ...

○ ...

○ ...

○ ...

# ¿Dudas sobre el onboarding?

Mira los vídeos de  
Onboarding en la  
Plataforma



# Clase 0

En la clase anterior realizamos una [actividad de repaso](#) sobre los temas más importantes.

Te recomendamos que la realices de manera asincrónica para reforzar tus conocimientos.



# Python y Google Colab

¿Recuerdas que la clase pasada te pedimos que instales Python y configures Google Colab?

¡Comenzaremos a poner manos a la obra! 😎

Psst... en caso de que no lo hayas hecho, puedes hacerlo ahora mientras repasamos los puntos más importantes del onboarding.

[Ver tutorial](#)





## ACUERDOS Y COMPROMISOS

# Equipo

- ✓ **¡Participa de los After Class!** Son un gran espacio para atender dudas y mostrar avances.
- ✓ Intercambia ideas por **el chat de la plataforma.**
- ✓ Siempre **interactúa respetuosamente.**
- ✓ No te olvides de **valorar tu experiencia** educativa y de contarnos cómo te va.

**After Class**

## AFTER CLASS

# ¿Qué son?

Te acompañamos para resolver tus consultas sobre el contenido en estos espacios.

Si hay **temas que no se entendieron o necesitan refuerzo** se trabajarán en una clase de 1 hs que opera como **espacio de consulta**.

No son obligatorias ni se toma asistencia, pero son el espacio uno a uno con tu profesor/a\*\* para responder dudas puntuales o reforzar conceptos.

Tu profesor/a está comprometido con tu educación, por lo tanto:

- ✓ Se responderán **dudas puntuales** que hayan quedado sobre los temas dados. ¡Vení preparado, queremos escucharte!
- ✓ Se verán temas de **conocimientos básicos** para la nivelación de saberes.

\*\*Los/as tutores/as también serán protagonistas, liderando 5 veces este espacio en todo el curso.



# Instancias prácticas



## Actividades de clase

Ayudan a poner en práctica los conceptos y la teoría vista en clase. No deben ser subidas a la plataforma y se desarrollan en la clase sincrónica.



## Guía de Actividades

Actividades relacionadas con el Proyecto Final. No son entregables pero su resolución es muy importante para llegar con mayor nivel de avance a las preentregas obligatorias. Se desarrollan de forma asincrónica.



## Pre-entregas

Entregas con el estado de avance de tu proyecto final que deberás subir a la plataforma a lo largo del curso y hasta 7 días luego de la clase, para ser corregidas por tu tutor/a.



# Instancias prácticas

## ALERTAS

# ¿Qué son y cuándo aparecen?



### CoderAlert

Son avisos creados para comunicar cuándo los temas de una clase están directamente relacionados con alguna **pre-entrega** y con el **proyecto final** de modo que puedas ir construyendo con antelación parte de la consigna. Lo conseguirás usualmente al final de la presentación de la clase.



### Coder Training

Son alertas que te indicarán que el contenido de una clase puede ser ejercitado mediante a través de actividades presentes en la Guía de Actividades. Son totalmente opcionales y cumplen la función de espacio práctico asincrónico.

**NOTA:** En ambos casos podrás usar la Guía de Actividades para practicar.

## 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.

INSTANCIAS PRÁCTICAS

# Hoja de Ruta



Dentro de tu carpeta de cursada encontrarás el archivo de “Hoja de ruta”, este espacio fue creado para que puedan visualizar en un mismo lugar, de manera rápida y ágil, todas las pre-entregas y entrega del Proyecto Final.

**Te recomendamos utilizar esta herramienta para organizar la cursada y la construcción de tu proyecto final.**





# ¡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?



# ¿Qué es una Web Playground?

Un playground es un **lugar de pruebas**, donde podremos hacer, deshacer, refactorizar, romper, etc.

En un Playground **no se abordan las cuestiones relativas a la Interfaz Gráfica (la parte visual de la App), solo trabajaremos con el Código Fuente**. La idea de generar una web playground es para que podamos aventurarnos en las funciones más avanzadas de Python y Django, de tal forma que siempre tengamos un lugar donde probar dichas funciones.



## PROYECTO FINAL

# Web Playground

### Consigna:

Desarrollarás una **web** en Django. Las entregas son individuales. Se verán temas avanzados:

### Registros:

- ✓ Login/signup
- ✓ Reset pwd
- ✓ Logout

### Perfil:

- ✓ Imagen
- ✓ Editar email/pwd

### Páginas:

- ✓ CRUD solo si está registrado

### Admin:

- ✓ Apps





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
- ✓ Metodologías Ágiles
- ✓ PYTHON

01

## Números y cadenas de caracteres

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

02

## Listas y Tuplas

- ✓ Listas
- ✓ Funciones
- ✓ Tuplas



PARA RECORDAR

# A tener en cuenta

Este curso cuenta con una [valija de recursos introductorios](#).

Podrás encontrar tutoriales, contenido audiovisual, un glosario y una propuesta gamificada

**¡Anímate a descubrirla!**



# 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





# Tu primer Colab

Creación de un nuevo notebook

Duración: **10 minutos**

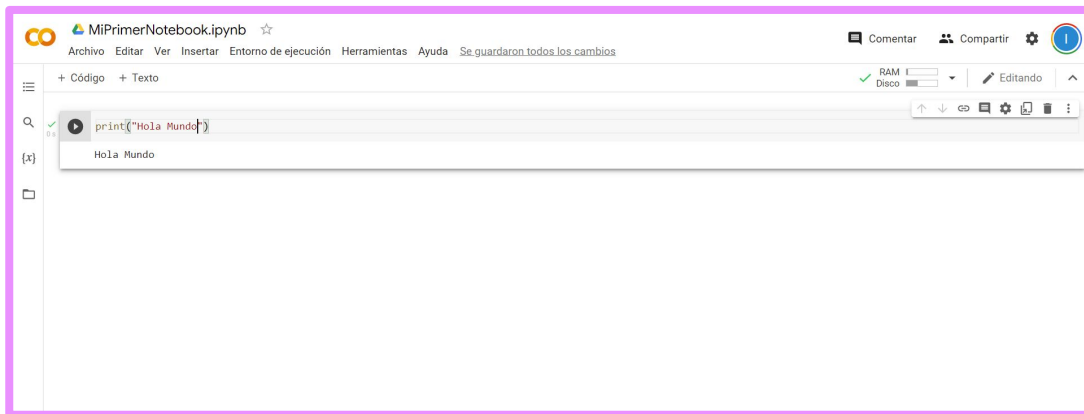


## ACTIVIDAD EN CLASE

# Mi primer Colab

Crearás un nuevo notebook en Google Colab, titulado: **MiPrimerNotebook** y deberás mostrar por pantalla la frase: **"Hola Mundo"**, obteniendo el siguiente resultado:

Si lo has intentado anteriormente, pero no lo has logrado, es momento de consultar tus dudas.



# Números

# 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).

Ejemplo: 1, 2, 525, 0, -817

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

# Long

Los números enteros largos o long en Python son iguales a los enteros, no tienen decimales, y **pueden ser positivos, negativos o cero**. Se tratan de números de cualquier tamaño. Se puede definir con una **L** al costado de nuestro número.

Ejemplo:

```
9812893712387912379123L  
897538475389475198237891249823L  
12387349587373L
```

# Float/Decimal

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.

Ejemplo:

0,270

-12,1233

989,87439124387

-74,9349834

# Complejos

Los números complejos son los que tienen parte imaginaria, es muy probable que no lo vayas a necesitar nunca. Este tipo se llama **complex**, se almacena usando reales, ya que es una extensión de dichos números.

Ejemplo:

2,1j  
-41,832i  
88,23 254j

# Operaciones numéricas

# Operaciones numéricas en Python

En programación y en matemáticas, los operadores aritméticos son aquellos que manipulan 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	+	$3 + 5 = 8$
Resta	-	$4 - 1 = 3$
Multiplicación	*	$3 * 6 = 18$
Potencia	**	$3 ** 2 = 9$
División (cociente)	/	$15.0 / 2.0 = 7.5$
División (parte entera)	//	$15.0 // 2.0 = 7$
División (resto)	%	$7 \% 5 = 1$

# 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.

Las expresiones con paréntesis se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.


# 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. `X, /, %, //`
  4. `+, -`

# Cadenas de texto

# 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



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

## ¿Para qué sirve?

La forma correcta de mostrar cadenas de texto (u otros objetos) por pantalla en Python es utilizando una 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:

```
[in] print('Una cadena\tcon tabulación')
```

```
[out]Una cadena con tabulación
```

```
[in] print('Otra cadena\ncon salto de línea')
```

```
[out]Una cadena  
con salto de línea
```



# Print

Si por ejemplo 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 tiene una forma de “**printear**” cruda o **raw**. Lo indicamos con una **r** delante de lo que se va a imprimir, y Python automáticamente lo interpretará para no tomar en cuenta los caracteres especiales.

```
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  
""")
```



# Break

¡10 minutos y volvemos!

# Variables

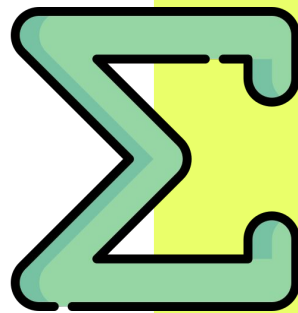
# Variables en matemáticas

Dependiendo del contexto, las variables tienen distintos significados. En el caso del Álgebra, una variable representa una cantidad desconocida que se relaciona con otras. Consideremos por ejemplo la ecuación:

**Ejemplo:**

$$x + 3 = 5$$

Fuente: [Variables](#)



# Variables en programación

# Variables en programación

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 **permiten hacer referencia a los datos** (que se guardan en unas "cajas" llamadas objetos).

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos.



# Variables en programación

Para cada dato que aparece en un programa, Python crea un objeto que lo contiene. Cada objeto tiene:

1

Un **identificador único** (un número entero, distinto para cada objeto). El identificador permite a Python referirse al objeto sin ambigüedades.

2

Un **tipo de datos** (entero, decimal, cadena de caracteres, etc.). El tipo de datos permite saber a Python qué operaciones pueden hacerse con el dato.

3

Un **valor** (el propio dato).



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. En este caso, el objeto creado será de tipo número entero y guardará el valor 2. Se asocia el nombre al objeto número entero 2 creado.



# Variables en programación

Al describir la instrucción anterior no habría que decir 'la variable **a** almacena el número entero **2**', sino que habría que decir 'podemos llamar **a** al objeto número entero **2**'.

La variable **a** es como una etiqueta que nos permite hacer referencia al objeto "2", más cómoda de recordar y utilizar que el identificador del objeto.



# Definir variables



## Ejemplo en vivo

¡Vamos a la práctica!. Definiremos las variables.

Duración: **10 minutos**



# Definir una variable

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

```
>>> 2 = mi_variable  
SyntaxError: can't assign to literal
```

Para mostrar el valor de la variable hay que escribir su nombre, o "printearlo".

```
>>> mi_variable = 2  
>>> mi_variable  
2  
>>> print(mi_variable)  
2
```



# Definir una variable

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

```
>>> x = -10
>>> y
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    y
NameError: name 'y' is not defined
```





# 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).

```
>>> nombre = "Pepito Conejo"
>>> nombre
'Pepito Conejo'
>>> print(nombre)
'Pepito Conejo'
```



# 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):

```
>>> nombre = Pepe
```

```
Traceback (most recent call last):
```

```
File "<pyshell#0>", line 1, in <module>
```

```
    nombre = Pepe
```

```
NameError: name 'Pepe' is not defined
```

```
>>> nombre = Pepito Conejo
```

```
SyntaxError: invalid syntax
```



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 guión bajo (\_) y puede seguir con más letras, números o guiones bajos (esto en inglés se llama **snake case**).

```
>>> fecha_de_nacimiento = "27 de octubre de 1997"  
>>> fecha_de_nacimiento  
'27 de octubre de 1997'
```

Los nombres de variables no pueden incluir espacios en blanco.

```
>>> fecha de nacimiento = "27 de octubre de 1997"  
SyntaxError: invalid syntax
```



# 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  
'Pepito Conejo'  
>>> Nombre  
'Numa Nigerio'  
>>> nomBre  
'Fulanito Mengáñez'
```

# INPUT



## Ejemplo en vivo

¡Veamos INPUT en la práctica!

Duración: **10 minutos**





# Input en vivo

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**.

Ej.: `>>> nombre = input()`



# Input en vivo

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 en vivo

## 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:

**Ejemplo:** `>>> nombre = int(input("¿Que edad tenes?"))`

# Operaciones aritméticas con variables



## Ejemplo en vivo

Vamos a practicar las operaciones aritméticas con variables

Duración: **10 minutos**

# Operaciones aritméticas con variables



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

```
>>> a = 2
```

```
>>> b = 3
```

```
>>> a+b
```

5

```
>>> a = 5
```

```
>>> b = 2
```

```
>>> a * b
```

10

```
>>> a = 35
```

```
>>> b = 7
```

```
>>> a / b
```

5

```
>>> a = 3
```

```
>>> b = 2
```

```
>>> a ** b
```

9

# 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
"PythonPython"
```

```
>>> cadena = "Python"
>>> otra_cadena = "Hola!"
>>> otra_cadena + cadena
"Hola!Python"
```

# String



# 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

# Longitud de strings

# Longitud de String

Python nos da una función llamada **len**. Esta función nos permite saber cuál es la longitud de un string, sin la necesidad de contar uno a uno los caracteres que tiene. También nos sirve en el caso de que no sepamos qué valor tiene una variable, pero tenemos que sacar determinados caracteres por índice.

**Ejemplo de len:**

```
>>> palabra = 'Python'
>>> len(palabra)
6
>>> otra_palabra = 'Hola, como están? Yo bien!'
>>> len(otra_palabra)
26
```

# Slicing

# Rebanar string (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'
```

# ¿Reasignar valor?

En algún momento nos preguntaremos si es posible traer el valor de una cadena de un índice a otro, ¿eso significa que puedo cambiarle el valor de un índice a uno que yo quiera?.

**Observemos el siguiente ejemplo:**

```
>>> palabra = "Python"
```



# Para pensar

¡Cometimos un error! Debería decir Python, no Pithon

```
>>> palabra[1] = "y"
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

```
    palabra[1] = "y"
```

**TypeError: 'str' object does not support item assignment**

Pero, ¿qué pasó acá?. ¿Por qué se rompió?.

Contesta mediante el chat de Zoom



## PARA RECORDAR

En Python, las cadenas de texto o strings, son **inmutables** esto significa, que no se puede sustituir ninguno de sus caracteres individualmente. Pero esto no es un gran problema. **Python es flexible, ¡podemos modificar el string que deseemos con slicing!**

```
>>> palabra = 'Pithon'  
>>> palabra = palabra[0:1] + 'y' + palabra[2:]
```

De esta forma podremos mostrar Python y no Pithon.



# 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: **10 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()`.



# Desafío String

Genera una nueva variable

Duración: **10 minutos**





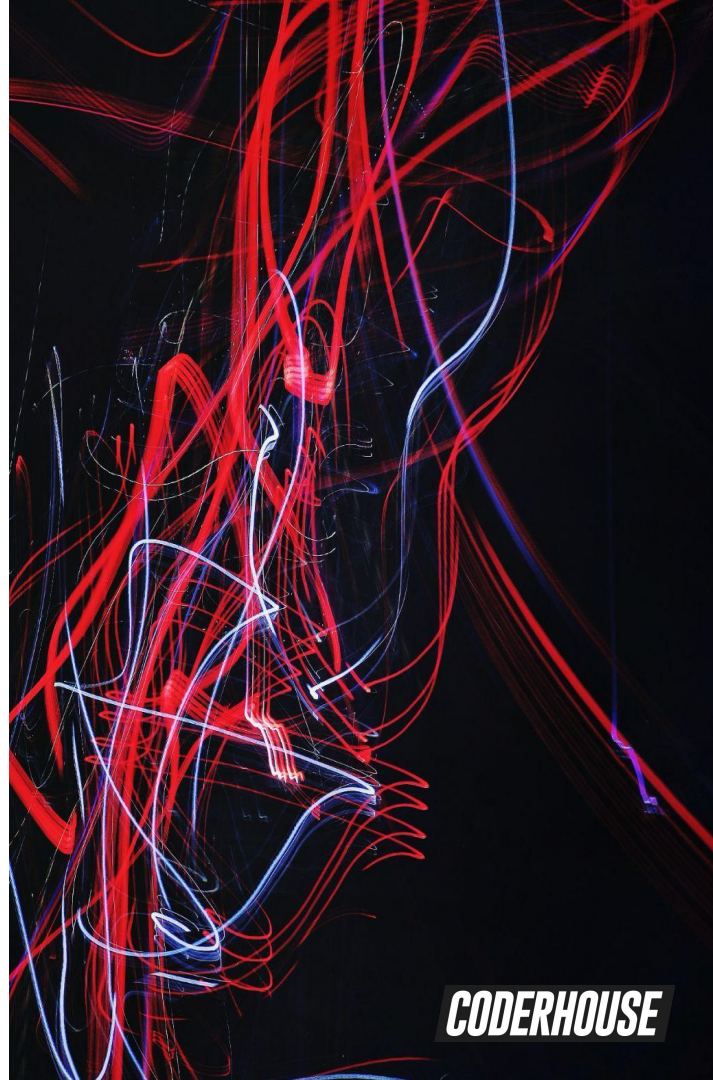
ACTIVIDAD EN CLASE

# Desafío String

**Descripción de actividad individual.**

Dadas cuatro variables con diferentes textos, genera una nueva variable con el siguiente contenido:

**Objetivo:** "Python es un lenguaje de programación versátil".



**CODERHOUSE**



ACTIVIDAD EN CLASE

# Desafío String

Descripción de actividad individual.

Partiendo de:

```
cadena_1 = "versátil"
```

```
cadena_2 = "Python"
```

```
cadena_3 = "es un lenguaje"
```

```
cadena_4 = "de programación"
```



**CODERHOUSE**



# Desafío Slicing

Dar vuelta la cadena y asignarla a una variable

Duración: **10 minutos**



## ACTIVIDAD EN CLASE

# Desafío Slicing

### Descripción de la actividad.

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.

De forma individual, realiza lo siguiente:

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 EN CLASE

4. Extraer la nota y almacenarla en una variable llamada nota.
5. Extraer la materia y almacenarla en una variable llamada materia.
6. Mostrar por pantalla la siguiente estructura:

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





## ACTIVIDAD EN CLASE

Formateando las anteriores variables en una variable llamada:

**`cadena_formateada`**

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

Para formatear ¡recuerda concatenar!

Ejemplo subido al Drive: [Desafío](#)



**CODERHOUSE**





# #Codertraining

¡No dejes para mañana lo que puedes practicar hoy! Te invitamos a revisar la [Guía de Ejercicios complementarios](#), donde encontrarás un ejercicio para poner en práctica lo visto en la clase de hoy.



## 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.





# 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:

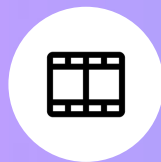
- ✓ `nota_1` cuenta como el 20% de la nota final
- ✓ `nota_2` cuenta como el 30% de la nota final
- ✓ `nota_3` cuenta como el 50% de la nota final



# 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.



**¿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.

¿Preguntas?

# Resumen de la clase hoy

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

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

**Muchas gracias.**



**#DemocratizandoLaEducación**