Este documento tiene como objetivo introducir conceptos básicos del lenguaje Python 3 y mostrar su implementación en COLAB. Por medio de ejercicios que lo llevarán a familiarizarse con la sintaxis y lógica de programación en el lenguaje.

Índice

Qué es Python

Python es un lenguaje de programación simple con sintaxis limpia, es portable, interpretado, orientado a objetos, de código abierto, de tipado dinámico y fuertemente tipado. Su filosofía es el famoso Zen de Python

Utilizamos Python 3 porque es fácil de aprender, fácil de entender y de utilizar, sobre todo para prototipado rápido.

¿Dónde comenzar?

Existe mucha documentación en internet, conozca la página oficial https://www.python.org/
donde podrá descargar python, realizar tutoriales completos y leer la documentación oficial.

Tutoriales
Instalar Python con miniconda
The Python Tutorial
Programación en Python - Nivel básico
Curso Python desde 0
Curso de Python Básico Gratis

Libros para aprender Python
Los 7 Mejores Libros para Aprender Python
Python para todos

CONDA

Conda es un sistema de gestión de paquetes de código abierto y un sistema de gestión del entorno que se ejecuta en Windows, macOS y Linux

Conda como gestor de paquetes le ayuda a encontrar e instalar paquetes, conda también es un administrador de entorno. Podemos utilizarlo por medio de Anaconda o Miniconda.

Anaconda es una distribución de paquetes científicos, entre ellos, Python. Conda para Python incluye una variedad de paquetes muy útiles: Virtualenv, Pandas, Jupyter y más. Anaconda es la versión completa e incluye más de que 150 paquetes.

Miniconda es una versión básica, que únicamente incluye a conda, Python y otros paquetes básicos. Ideal para crear ambientes desde cero.

Guía para utilizar COLAB en el curso

Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube.

Colaboratory te permite escribir y ejecutar código, guardar y compartir tus análisis y tener acceso a recursos informáticos muy potentes, todo de forma gratuita desde el navegador.

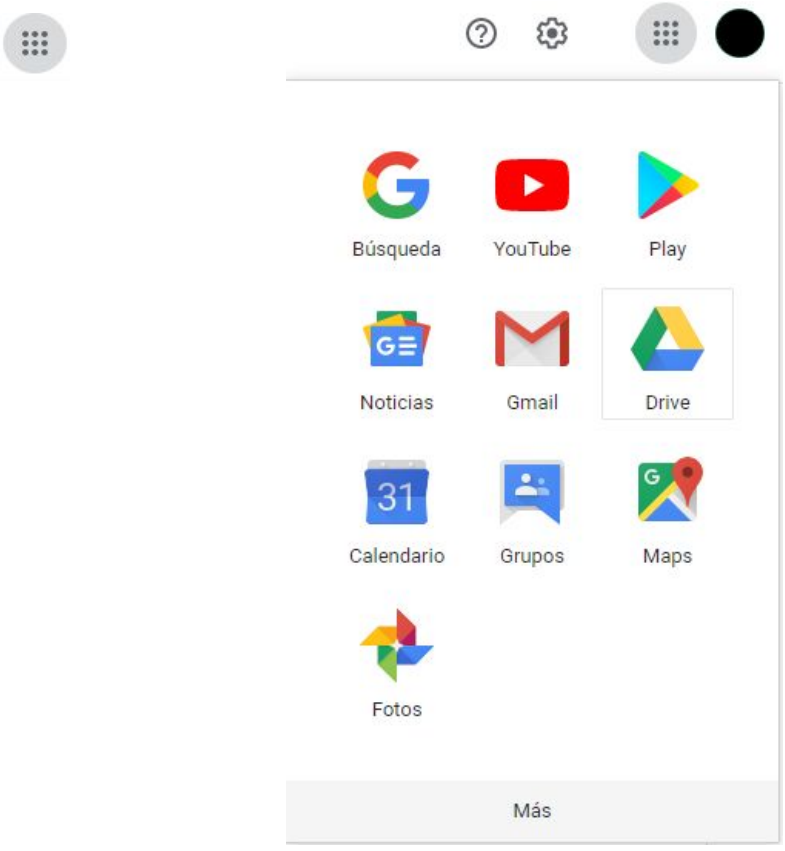
Video tutorial introductorio: https://youtu.be/n7RdjB9bDKo

Instrucciones de uso para los cursos Actumlogos

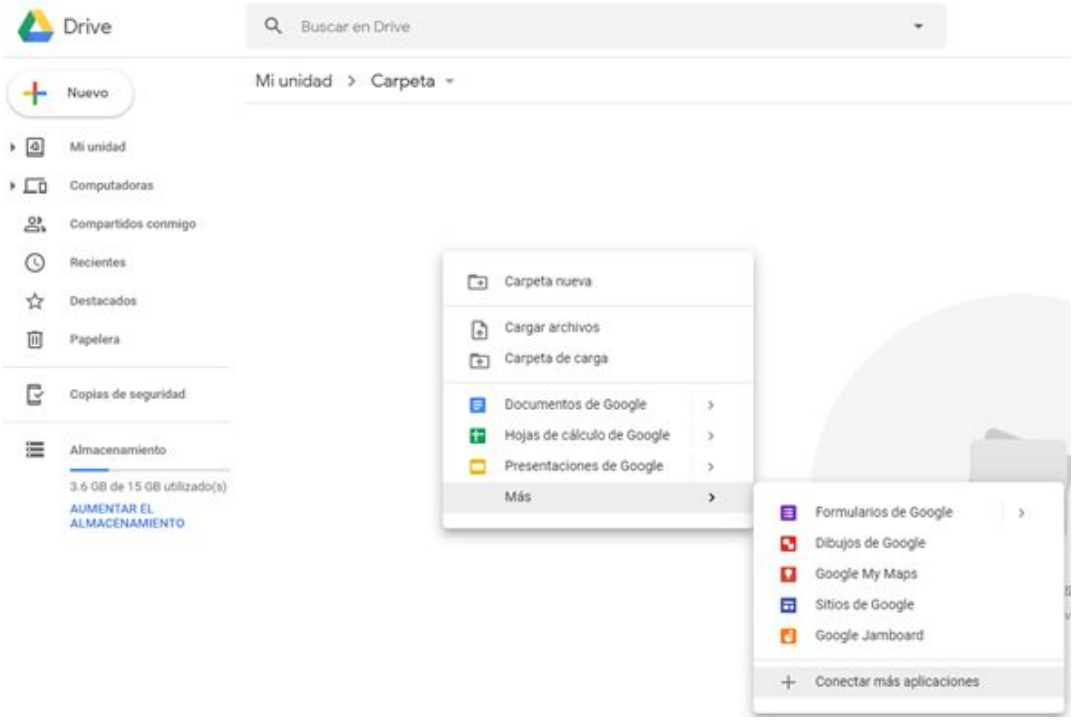
Requisito: Tener cuenta de Gmail

1) Entrar a su cuenta de Gmail

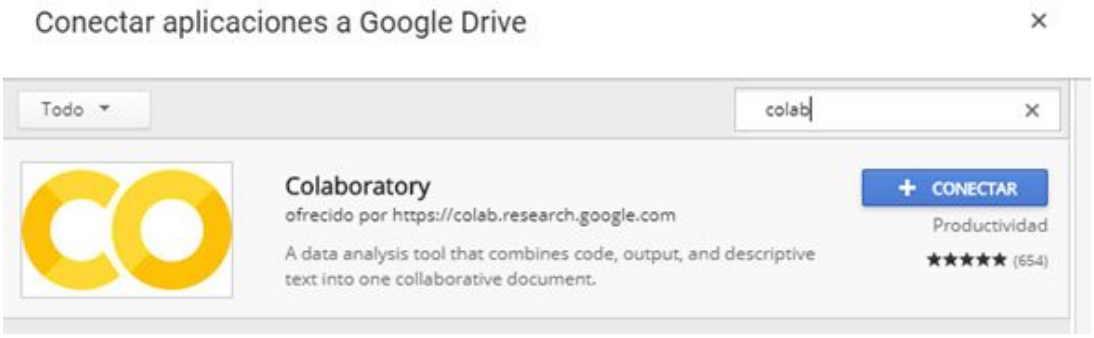
2) Acceda a su **Drive** desde el icono de Google Apps



3) Dentro de su **Drive**, de *clic derecho* sobre su unidad y seleccione del menú la opción:
Más > **Conectar más aplicaciones**

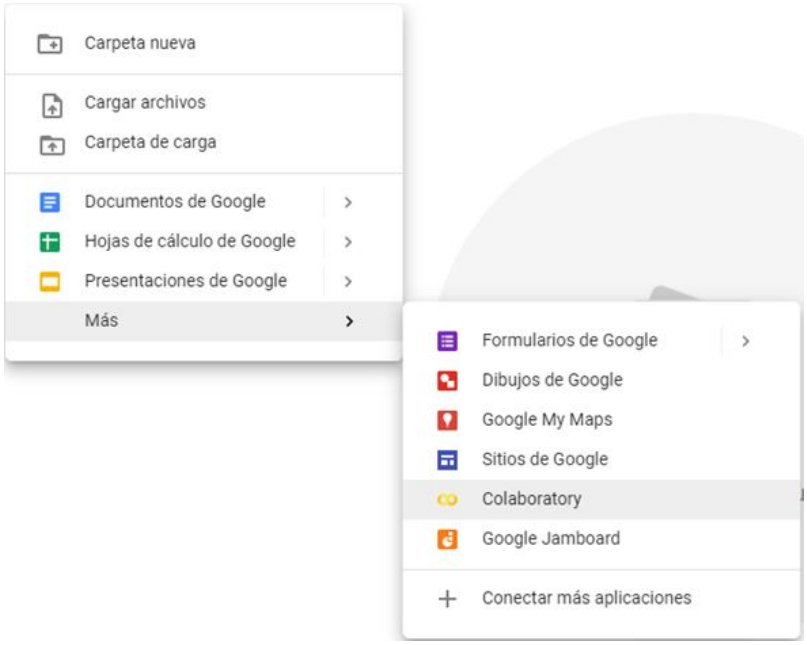


4) Escriba la palabra **colab** en la barra de búsqueda y seleccione el botón **+ CONECTAR**, acepte y cierre la ventana para continuar

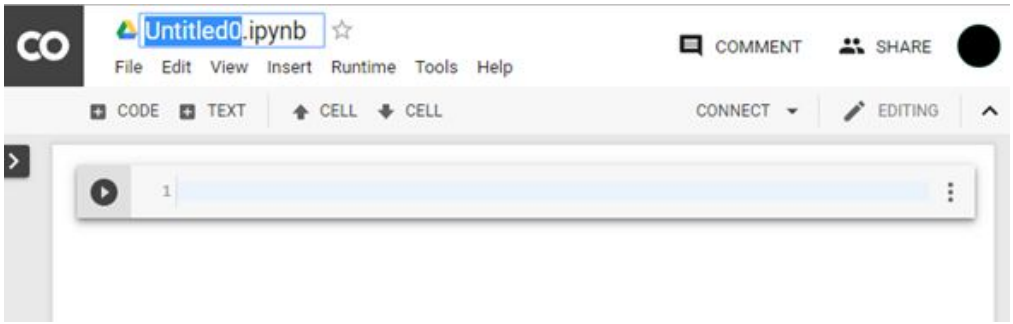


5) Ahora podrá crear archivos notebooks en su drive de la siguiente forma:
   a) Vaya nuevamente a su unidad en **Drive**

   b) De clic derecho sobre su unidad (como en el punto 3) y seleccione del menú la opción: **Más** > **Colaboratory**



   c) Modifique con doble clic el nombre del archivo para identificar su código



6) Escriba un código de prueba y de clic en el icono de play para ejecutar el código

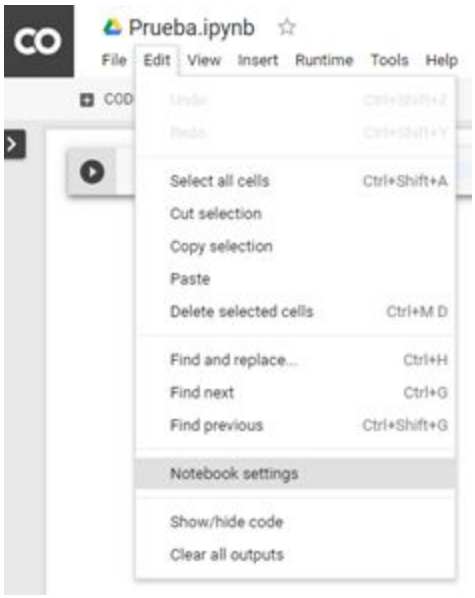
7) Para cerrar el archivo, de clic en este símbolo **CO** localizado en la parte superior izquierda del notebook

Configuración de ambientes

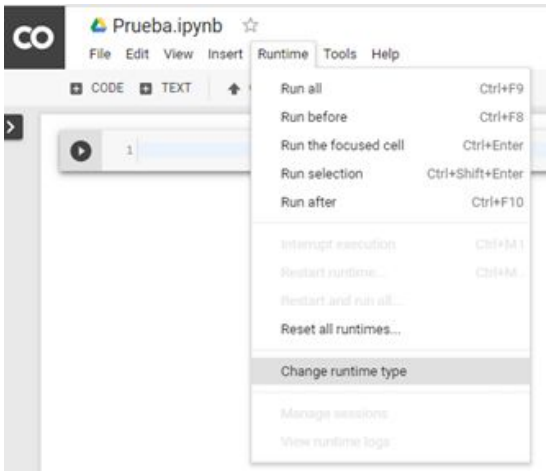
Se puede configurar el ambiente para código Python2 o Python3, así como en modo CPU, GPU o TPU, la configuración por defecto es Python3 - CPU.

Para fines prácticos verifique siempre que sus ambientes están en modo GPU, para ello realice una de las siguientes opciones:

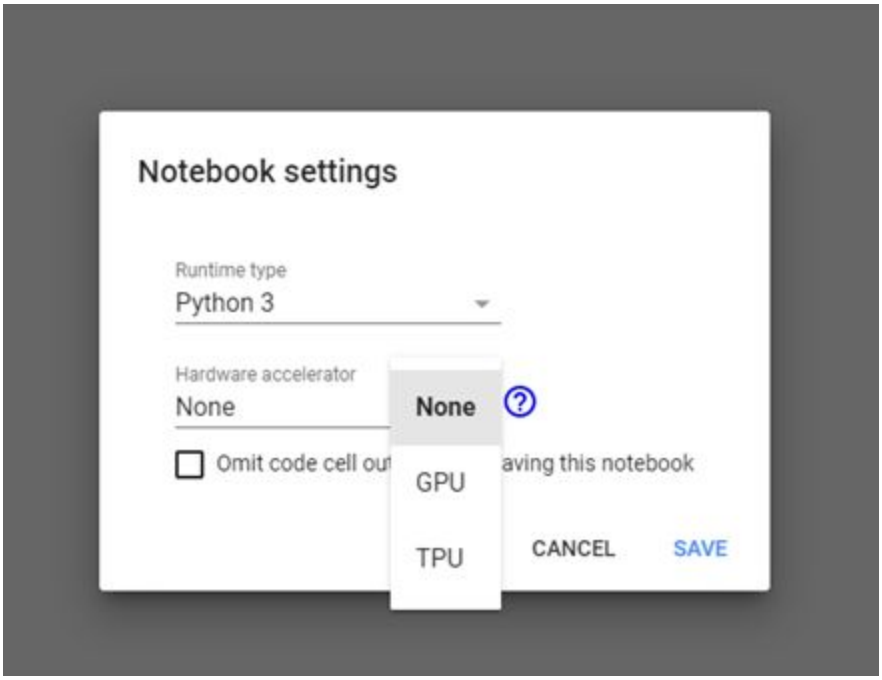
(1) Vaya a la pestaña **Edit** y seleccione "Notebook settings"



(2) Vaya a la pestaña **Runtime** y seleccione "change runtime type"



Se mostrará la siguiente ventana, seleccione en **Harware accelerator** la opción de **GPU** y salve. None equivale a utilizar CPU.



Ejercicios

La dinámica de los ejercicios siguientes le ayudará a comprender conceptos de programación de python utilizados en los cursos, deberá seguir las instrucciones en color azul y analizar cuidadosamente los códigos.

Los códigos de los ejercicios se encuentran en la carpeta compartida “30ejercicios_colab”, son notebooks .ipynb que puede editar y correr con colab.

En python los nombres de variables, objetos, funciones y más, tiene la siguiente convención:

```
module_name, package_name, ClassName, method_name, ExceptionName,
function_name, GLOBAL_CONSTANT_NAME, global_var_name,
instance_var_name, function_parameter_name, local_var_name
```

Conceptos Básicos

Ejercicio 1: Asignación de una variable

Analice con cuidado el siguiente código python

```
# Los comentarios en python comienzan con el símbolo almohadilla

print("Esta es una cadena")  # una sentencia NO termina en ;

# Esta es una asignación a una variable
variable = "Esta es otra cadena"

# Así se imprime una variable
print(variable)
```

Su salida es la siguiente:

```
Esta es una cadena
Esta es otra cadena
```

Complete el siguiente código con las indicaciones dadas por los comentarios.
Puede copiar el código en COLAB para probarlo.

NOTA: NO utilice acentos en las variables o nombres de funciones

```
cadena = "cadena"
print(cadena)

# Asigne a la variable cadena con comillas simples 'Esta es una cadena'
cadena = ____

# En la siguiente línea, imprima la variable cadena


# Asigne el número 5 a la variable numero e imprima su valor
numero = ____
```

Resultado esperado:

```
cadena
Esta es una cadena
5
```

Ejercicio 2: Asignación múltiple de variables

```
# Asignar el mismo valor a múltiples variables
a = b = c = 0
print(a, b , c)

# Asignar a múltiples variables, valores diferentes
x, y, z = 10, 3.5, "hola"
print("x=", x)
print("y=", y)
print("z=", z)
```

Su salida es la siguiente:

```
0 0 0
x= 10
y= 3.5
z= hola
```

Complete código del ejercicio02.ipynb con las indicaciones dadas por los comentarios.

```
"""
Usted tiene 50 pesos en el bolsillo, necesita comprar 3 artículos
Un café, galletas y papas para soportar el hambre
"""
mi_dinero = 50

# Asigne el valor de 24 pesos al café, 12 a las galletas y 13 a las papas
cafe, papas, galletas = _______________

# Imprima el valor de los 3 artículos de la forma
# print('artículo=', variable)

# Imprima la resta de los artículos al dinero que tiene

# Imprima cuanto le quedaría si solo compra café y galletas
```

Resultado esperado:

```
cafe= 24
galletas= 12
papas= 13
1
14
```

Ejercicio 3: Tipos de datos

En Python podemos ver el tipo de dato de las variables con la sentencia `type()` y convertir a otros tipos de datos como se muestra en el código

```
a = 5
b = 2.55
c = "100"

print(type(a))
print(type(b))
print(type(c))

# Transformar a otro tipo de dato (Casting de variables)
flotante = float(a)  # convierte el 5 a 5.0
cadena = str(b)  # convierte 2.55 a una cadena "2.55"
entero = int(c)  # convierte la cadena "100" a un número 100

print(cadena, type(cadena))
print(flotante, type(flotante))
print(entero, type(entero))
```

Su salida es la siguiente:

```
<class 'int'>
<class 'float'>
<class 'str'>
2.55 <class 'str'>
5.0 <class 'float'>
100 <class 'int'>
```

Corrija el código de ejercicio03.ipynb para que no mande error

```
cad = "Vehículo"
num = 250

print(cad + num)  # TODO: Corrija la línea para imprimir una cadena

# A una variable se le puede asignar otro tipo de dato sin especificar de qué tipo es
variable = "Juan Carlos"
print("variable:"+ variable, "tipo:"+ type(variable))  # TODO: Corrija la línea

variable = 2.5 + 3
print(variable:, variable+ "tipo:", type variable )  # TODO: Corrija la línea
```

Resultado esperado:

```
Vehículo250
variable: Juan Carlos tipo: <class 'str'>
variable: 5.5 tipo: <class 'float'>
```

Los operadores en python son aquellos que utilizamos para manipular los datos, los más elementales son los siguientes:

Aritméticos

Descripción	**Operador**
Suma	+
Resta	-
Multiplicación	*
Potencia	**
División	/
División entera (Piso)	//
Módulo	%

Comparación

Descripción	**Operador**
Son iguales	==
Son diferentes	!=
Menor que, menor o igual	< , <=
Mayor que, mayor o igual	>, >=

Lógicos

Descripción	**Operador**
Se cumplen ambos	and
Se cumple alguno	or
Negación	not

Ejercicio 4: Operadores

Realice las siguientes operaciones

```
cadena_1 = "El oso come "
cadena_2 = 'mucha miel'

PI = 3.141592
a, b, c = 25, 8.3, 12

# Defina la variable concat y en ella concatene las cadenas 1 y 2
# Imprima concat

# Calcule el área de un círculo de radio 3.33 e imprima su valor
area = ____
print("El área es:", )

# Calcule la ecuación "(a x b)^2 / c" y el resultado guárdelo en x

# Incremente x en 1, con la expresión reducida de x = x + 1

print(x)
```

Nota: A diferencia de otros lenguajes de programación, python no cuenta con el incremento abreviado ++

Resultado esperado:

```
El oso come mucha miel
El área es: 34.8367995288
x= 3589.0208333333344
```

Colecciones

Listas

Las listas en python son

- **heterogéneas**: pueden estar conformadas por elementos de distintos tipo, incluidos otras listas.
- **mutables**: sus elementos pueden modificarse.

Métodos: append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

```
lista_numeros = [1, 2, 3, 4, 5, 6]
lista_compuesta = ["ABC", 125, .001, lista_numeros, ":)"]
print(lista_compuesta)

# Para modificar valores de una lista, se accede al índice entre corchetes
# Rango [0 a n-1]
lista_numeros[0] = 99  # Modifica el primer número de la lista
print(lista_numeros)

# Los índices negativos acceden en sentido inverso (último a primero)
# Rango [-1 a -n]
lista_numeros[-1] = 100  # Modifica el último elemento de la lista

# El operador : permite iterar en los elementos de una lista
# [inicio : m-1]
print(lista_compuesta[0:3])

# Si no se indica el valor inicial itera desde el comienzo
print(lista_compuesta[:3])  # Equivalente

# Si no se indica el valor final itera hasta el último elemento
print(lista_numeros[2:])

# Imprime los elementos de la lista desde el índice 1 y de 2 en 2
print(lista_numeros[1::2])
```

Su salida es la siguiente:

```
['ABC', 125, 0.001, [1, 2, 3, 4, 5, 6], ':)']
[99, 2, 3, 4, 5, 6]
['ABC', 125, 0.001]
['ABC', 125, 0.001]
[3, 4, 5, 100]
[2, 4, 100]
```

Ejercicio 5: Lista de listas (matriz)

Se puede anidar listas en una lista para crear una matriz

```
list_1 = [0, 1, 2]
list_2 = ["3", "4", "5"]
list_3 = [6.0, 7.0, 8.0]
matriz = []

"""
Cree una matriz de enteros de la forma
|2   1   0|
|3   4   5|
|8   7   6|
utilizando como base las listas 1 a 3
utilize la sentencia append() para crear la matriz
"""
# Crear matriz

# Convertir a enteros
matriz[1][0] = int(matriz[1][0])
matriz____
matriz____
matriz____
matriz____
matriz[2][2] =

# Imprima la matriz con un print
```

Resultado esperado:

```
[[2, 1, 0], [3, 4, 5], [8, 7, 6]]
```

Tuplas

Son muy similares a las listas y también son heterogéneas, difieren en que son

- **inmutables**: sus elementos NO pueden modificarse una vez creada.
- Utilizan ( ) en lugar de [ ] para definirlas

Métodos: count(), index()

Ejercicio 6: Tuplas

Cree una tupla y llamela "otra_tupla", inicializarla para que contenga a la lista como primer elemento y a tupla como segundo elemento. Descomente la última línea para probar que una tupla no se puede modificar.

```
tupla = (1, 2, 3, "xyz", True, [0.1, 1.0])
lista = ["gallina", "pavo", "avestruz"]
_____ = _____

print(otra_tupla)
print(otra_tupla[0][2])
print(otra_tupla[1][-1])

# Una tupla de un elemento se escribe así:
uni_tupla = (1, )  # Sin la coma seria un variable normal
print(uni_tupla, type(uni_tupla))

# Descomente la última línea
# La siguiente linea marca error, por que no se puede modificar una tupla
# otra_tupla[0] = "variable"
```

Resultado esperado:

```
(['gallina', 'pavo', 'avestruz'], (1, 2, 3, 'xyz', True, [0.1, 1.0]))
avestruz
[0.1, 1.0]
(1,) <class 'tuple'>

Traceback (most recent call last):
    otra_tupla[0] = "variable"
TypeError: 'tuple' object does not support item assignment
```

Diccionarios

Los diccionarios son una colección no ordenada de pares, estos pares son:

- **llaves**: mapean a los valores, es decir, con ellas se puede acceder a un valor dentro del diccionario. La llave es única dentro de un diccionario y debe ser de tipo inmutable (cadena, tupla, numérico).
- **Valores**: son objetos, pueden ser cualquier tipo de dato.

Los diccionarios a diferencia de las listas y tuplas, se definen con llaves { }. Los pares se relacionan entre sí con el símbolo **:** de la siguiente forma **{llave1 : valor1, llave2 : valor2, ...}**

Métodos: clear(), copy(), fromkeys(), get(), has_key(), setdefault(), update(), values()

Ejercicio 7: Diccionarios

Edite el código ejercicio07.ipynb en COLAB

- **Modifique el código para obtener las llaves de un diccionario e imprimalas**
- **Obtenga los valores del diccionario e imprima**
- **Sobreescribe el valor de llave2 por el numero 250**

```
diccionario = {"1":"primer elemento",
               "llave2":[3.5, "B"],
               100:("jugo","fruta", "pan")}

print(diccionario)

# Obtener las llaves en el diccionario
var_llaves = _____
print("Imprimimos las llaves")
print( _____)

# Obtener solamente los valores del diccionario
var_valores =  _____
print("\nImprimimos los valores")
print( _____)

# Obtener el segundo elemento del diccionario
elemento = diccionario.get("llave2")
print("\nEl elemento 2 es:")
print(elemento)

# Modifica el segundo elemento
diccionario[ _____] =  _____
print(diccionario)
```

Resultado esperado:

```
{'1': 'primer elemento', 'llave2': [3.5, 'B'], 100: ('jugo', 'fruta', 'pan')}
Imprimimos las llaves
dict_keys(['1', 'llave2', 100])

Imprimimos los valores
dict_values(['primer elemento', [3.5, 'B'], ('jugo', 'fruta', 'pan')])

El elemento 2 es:
[3.5, 'B']
{'1': 'primer elemento', 'llave2': 250, 100: ('jugo', 'fruta', 'pan')}
```

Estructuras de control

La indentación es importante en python, sustituye los corchetes que delimitan los bloques de código en otros lenguajes, lo más adecuado es utilizar 4 espacios para indentar un bloque o con tabulación, sin embargo, no se deben mezclar tabuladores con espacios en un bloque.

El siguiente código muestra la iteración de un diccionario con un ciclo for

```
diccionario = {"1":"primer elemento",
               "llave2":[3.5, "B"],
               100:("jugo","fruta", "pan")}

# Recorre un diccionario con un ciclo for
for llave, valor in diccionario.items():
   print("llave:", llave)
   print("valor:", valor)
```

Su salida es la siguiente

```
llave: 1
valor: primer elemento
llave: llave2
valor: [3.5, 'B']
llave: 100
valor: ('jugo', 'fruta', 'pan')
```

Ejercicio 8: Ciclo for

Edite el código ejercicio08.ipynb en COLAB

- **Cree una tupla con 10 elementos**
- **Recorra la tupla con ciclo for y cada 2 elementos, haga una copia de ese elemento en una lista con append()**
- **Imprima la lista creada con un for**
- **En un diccionario guarde los elementos de la lista con las llaves:**
  “uno”, “dos”, “tres”, “cuatro”, “cinco”

Resultado esperado:

```
1
3
5
7
9
{'uno': 1, 'dos': 3, 'tres': 5, 'cuatro': 7, 'cinco': 9}
```

Ejercicio 9: Condicionales

```
lista = ["cadena", 25, 3.50000000000, False]

for item in lista:  # El operador in
   print(item)

   # is, compara ambos lados de la expresión condicional
   if item is 3.5:  # True si es el mismo objeto
       print("%.3f es Flotante" %(item)) # Print con formato
       break  # break termina el ciclo y continúa el programa

print("Se termino el ciclo for")

# in, devuelve True cuando un elemento está en una secuencia.
if lista[1] in lista:
   print("%d esta en la lista" %(lista[1]))

# not in, devuelve True cuando un elemento NO está en una secuencia.
if 10 not in lista:
   print("No se encontro 10 en la lista")
```

Su salida es la siguiente:

```
cadena
25
3.5
3.500 es Flotante
Se termino el ciclo for
25 esta en la lista
No se encontro 10 en la lista
```

Edite el código ejercicio09.ipynb en COLAB

Se tiene una lista con 20 elementos, aleatoriamente un elemento sera la cadena “alto”

- **Itere la lista utilizando while, e imprima únicamente cuando encuentre la cadena “alto”**
- **Itere la lista con ciclo for imprimiendo cada elemento, detenga el ciclo cuando encuentre la cadena “alto” sin imprimir la cadena**

Resultado esperado:

```
alto
```

```
Iteración actual: 15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Funciones

Una función es un bloque de código, recibe argumentos como entrada para realizar una tarea específica, devuelve un valor al término de su ejecución. Este bloque tiene un nombre asociado (el nombre de la función), su sintaxis es la siguiente:

```
def nombre_funcion( parametros ):
    sentencia
    ...
    sentencia
   return [expresion] # El retorno es opcional
```

```
# Una función puede tener argumentos con valores predeterminados
def funcion_suma (x, y=0):
   print("x= %.2f" % x)
   print("y= %.2f" % y)
   return x + y

# Declaración prototipo, con pass indica que no se definido la función
def foo():
   pass  # TODO: Implementar algo aquí

print(funcion_suma(3.666))
print()
print(funcion_suma(2, 3))
```

```
print(foo())
```

Su salida es la siguiente:

```
x= 3.67
y= 0.00
3.666

x= 2.00
y= 3.00
5
None
```

Ejercicio 10: Funciones

Edite el código ejercicio10.ipynb en COLAB

- **Escriba una función que calcule el volumen de una pirámide**
- **Escriba otra función que calcule el volumen de un cilindro**

Resultado esperado con los valores de prueba:

```
Voúmen de la piramide: 10.5
Voúmen del cilindro: 9.424769999999999
```

Funciones anónimas lambda

En python lambda es una expresión que crea funciones anónimas, es decir que no tienen nombre, crea un objeto de tipo función en línea. Otra manera de entender las funciones lambda, es una poder definir una función simple en una sola línea y que pueda ser argumento de otra función u objeto.

El contenido de una función anónima debe ser una única expresión en lugar de un bloque de acciones.

Sintaxis:

```
lambda parametros : expresion
```

El siguiente código utiliza la expresión lambda para realizar la suma de 2 números y para calcular el cuadrado de los valores de una lista

```
# Función que devuelve la suma de 2 números
```

```
def suma(a, b):
   resultado = a + b
   return resultado

print("función a+b =", suma(8, 2))

# Podemos realizar la misma función como una expresión lambda
temp = lambda x, y: x + y
print("lambda a+b =", temp(8, 2))

# Aplicando las funciones lambda
valores = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(valores)

# map aplica una operación (definida con lambda)
# a cada elemento de una lista
resultados = map(lambda x : x**2, valores)  # Devuelve un objeto map
# Transformamos el resultado a tipo list para visualizar
print(list(resultados))
```

Su salida es la siguiente:

```
función a+b = 10
lambda a+b = 10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Ejercicio 11: Función lambda

Edite el código ejercicio11.ipynb en COLAB

- **Cree una lista con 50 números aleatorios del 0 al 49, use for y append**
- **Con una función, obtenga los números pares de la lista y guardarlos en una nueva lista**
- **Utilice map para aplicar la función y = sin(x) a la lista guardada**
- **Imprima los valores obtenidos**

Ejemplo de salida, recuerde que **son números aleatorios**, **no espere el mismo resultado**

```
[0, 18, 42, 44, 18, 20, 22, 12, 36, 24, 36, 20, 6, 24, 24, 42, 26, 38, 38, 26, 12, 26, 16, 16,
4, 14]
[0.0, -0.7509872467716762, -0.9165215479156338, 0.017701925105413577, -0.7509872467716762,
```

```
0.9129452507276277, -0.008851309290403876, -0.5365729180004349, -0.9917788534431158,
-0.9055783620066238, -0.9917788534431158, 0.9129452507276277, -0.27941549819892586,
-0.9055783620066238, -0.9055783620066238, -0.9165215479156338, 0.7625584504796027,
0.2963685787093853, 0.2963685787093853, 0.7625584504796027, -0.5365729180004349,
0.7625584504796027, -0.2879033166650653, -0.2879033166650653, -0.7568024953079282,
0.9906073556948704]
```

Iteradores

Un iterador es un objeto que tiene una función next(), es decir, cuando se le llama, devuelve la siguiente elemento en la secuencia. Las listas, las tuplas, los diccionarios y los conjuntos son todos objetos iterables. Son contenedores iterables de los que puedes obtener un iterador.

El metodo **iter()** es usado para obtener un iterador de un objeto iterable.

```
tupla = ("tres", "dos", "uno")
iterador = iter(tupla)
print(type(tupla))
print(type(iterador))

print(next(iterador))
print(next(iterador))
print(next(iterador))
```

Su salida es la siguiente:

```
tres
dos
uno
<class 'tuple'>
<class 'tuple_iterator'>
```

Generadores

Los Generadores son usados para crear iteradores, son simples funciones las cuales devuelven un objeto de tipo iterador. Utiliza la sentencia **yield** en lugar de return que puede suspenderlo o reanudarlo en tiempo de ejecución.
En otras palabras, devuelve una secuencia de elementos cada que se le llama y se pone en suspensión hasta ser llamado nuevamente.

```
def generator_lotes(lote=2):
```

```
    i = 0
    while True:
        arr = []
        for k in range(lote):
            arr.append(i)
            i += 1
        yield arr

itera = generator_lotes()
print(type(itera))

print(next(itera))
print(next(itera))
print(next(itera))
```

Su salida es la siguiente:

```
<class 'generator'>
[0, 1]
[2, 3]
[4, 5]
```

Ejercicio 12: Generador

Edite el código ejercicio12.ipynb en COLAB

Cree un generador que devuelva la serie de fibonacci

- **Imprima los primeros 10 elementos de la secuencia**

Resultado esperado:

```
1
1
2
3
5
8
13
21
34
55
```

Cadenas

El manejo de cadenas es importante en cualquier lenguaje

Ejercicio 13: Sub cadenas

Edite el código ejercicio13.ipynb en COLAB

- **Averigue el método para cortar cadenas en python y utilícelo para separar una cadena larga en palabras**
- **Tiene una lista de archivos 4 archivos con nombres “archivo_n.jpg”, separe la cadena en dos, separarla por el punto**
- **Obtenga una subcadena manualmente utilizando acceso por índices**

Resultado esperado:

```
['never', 'stop', 'LEARNING', 'because', 'life', 'never', 'stops', 'TEACHING']
arreglo de  2 elementos
['imagen_1', 'jpg']
arreglo de  2 elementos
['imagen_2', 'jpg']
arreglo de  2 elementos
['imagen_3', 'jpg']
arreglo de  2 elementos
['imagen_4', 'jpg']
LEARNIN
longitud: 7
```

Ejercicio 14: Formato de cadenas

Revise la documentación de formateo de cadenas en python para realizar este ejercicio.

Edite el código ejercicio14.ipynb en COLAB

- **En la primer celda, formateara 4 tipos de datos (str, int, bool, float) en un print para obtener la salida deseada. Utilice la documentación mencionada**

Resultado esperado:

```
Buenos días a todos y todas
                          Buenos días a todos y todas

999999999999999
9.99999999999999E+14

True
1

1.25193565145e-06
0.00000125
```

- **En la segund acelda averigue cómo replicar el contenido de una lista sin utilizar append o ciclos for**

Resultado esperado:

```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

Clases y Objetos

Clase: Una clase es una disposición de variables y funciones en una sola entidad lógica. Funciona como una plantilla para crear objetos. Cada objeto puede usar variables de clase y funciones como sus miembros.

Objeto: El objeto es una instancia de una clase creada en tiempo de ejecución. Un objeto comprende tanto miembros de datos (variables de clase y variables de instancia) como métodos.

__init__(self) es un método especial, que se llama constructor de clase o método de inicialización al que Python llama cuando crea una nueva instancia de esta clase. En python self representa la instancia de una clase. Funciona como un controlador para acceder a los miembros de la clase, como los atributos de los métodos de clase.

Los atributos o propiedades de los objetos son las características que puede tener un objeto, como el color. Si el objeto es Persona, los atributos podrían ser: cedula, nombre, apellido, sexo, etc…

Los métodos describen el comportamiento de los objetos de una clase. Estos representan las operaciones que se pueden realizar con los objetos de la clase

```
class Persona:
   # Constructor que inicializa los atributos
   def __init__(self, nombre=None, apellido=None, edad=None, sexo=None):
       self.nombre = nombre
       self.apellido = apellido
       self.edad = edad
       self.sexo = sexo

   # Método que escribe la información de Persona
   def yo_soy(self):
       print("Hola, me llamo: ", self.nombre, self.apellido)
       print("Tengo ", self.edad, " años")
       print("Y soy", (self.sexo))

# Creación de un objeto persona
p1 = Persona("Araceli", "Acosta", 25, "Mujer")
p2 = Persona("Eduardo", "Corona", 30, "Hombre")

p1.yo_soy()
p2.yo_soy()
```

Su salida es la siguiente:

```
Hola, me llamo:  Araceli Acosta
Tengo  25  años
Y soy Mujer
Hola, me llamo:  Eduardo Corona
Tengo  30  años
Y soy Hombre
```

Ejercicio 15: Crear una Clase

Edite el código ejercicio15.ipynb en COLAB

Defina la clase automovil, la cual debe tener los atributos:

- **tanque (float) con valor inicial de 0.0**
- **velocidad (float) con valor inicial de 0.0**
- **ocupantes (int) con valor inicial de 0**
- **encendido(bool) valor inicial de False**

Y los siguientes métodos sin definir (use la palabra reservada pass):

- **encender_apagar**
- **cargar_gasolina**
- **conducir**
- **dar _un_ray**

Resultado esperado:

```
El carro tiene:
Atributos: 0.0 0.0 0
Encendido: False
None
None
None
None
```

Ejercicio 16: Definir métodos

Edite el código ejercicio16.ipynb en COLAB

El rango de valores para los atributos es
de 0 a 100 para tanque
de 0 a 160 para velocidad
de 0 a 5 para ocupantes
Controle utilizando condicional if dentro de los métodos que los ocupen

Defina el comportamiento de los métodos de la clase automovil:

- **Encender_apagar, sin argumentos ()**
  si el auto está apagado y tiene gasolina
  éste se enciende y se añade 1 ocupante (el conductor)
  si está encendido
  deberá apagarse y los ocupantes serán 0 (todos salen)

- **Cargar_gasolina. Recibe como argumento (litros(float)), la cantidad que se le va a agregar**
  - **Cargar tanque con litros, si litros + tanque > 100**
    - **Dejar el tanque a 100**
- **Conducir, recibe de argumentos(velocidad(float), tiempo(int)), el tiempo representa horas, con un máximo de 10 horas y un mínimo de 1 hora**
  - **Al conducir correrá un ciclo for de 0 hasta tiempo**
    - **La gasolina se drenara con la ecuación:**
      - **Tanque = Tanque - velocidad * ocupantes / 8.2**
- **Dar_un_ray, recibe de argumento (personas(int))**
  - **Al igual que tanque, si ocupantes + personas > 5**
    - **Ocupantes = 5**
  - **Se puede recibir un negativo (personas salen), si ocupantes + personas < 1**
    - **Ocupantes = 1 (se queda el conductor)**

Resultado esperado:

```
Gasolina:  100
Encendido: True, personas 1
Me quedan 60.975610 litros
Somos 4 en el carro
Me quedan 12.195122 litros
Encendido: False, personas 0
```

Módulos y paquetes

Los módulos:
Son principalmente los archivos (.py) que contienen funciones de definición de código de Python, clase, variables, etc. con un sufijo .py añadido en su nombre de archivo.

Pueden tener diferentes funciones, variables y clases en un archivo. También podemos llamarlos bibliotecas.
Los paquetes Python (packages):

Los paquetes permiten una estructura jerárquica de espacios de nombres de módulos, utiliza notación de puntos. De la misma manera que los módulos ayudan a evitar colisiones entre nombres de variables globales, los paquetes ayudan a evitar colisiones entre nombres de módulos.

Los módulos y paquetes pueden ser importados con la siguiente sintaxis:

```
import <nombre_del_modulo>
from <nombre_del_modulo> import <nombre, nombre,… >
import < nombre_del_modulo > as <alias>
from <nombre_del_modulo> import <nombre> as <alias>
```

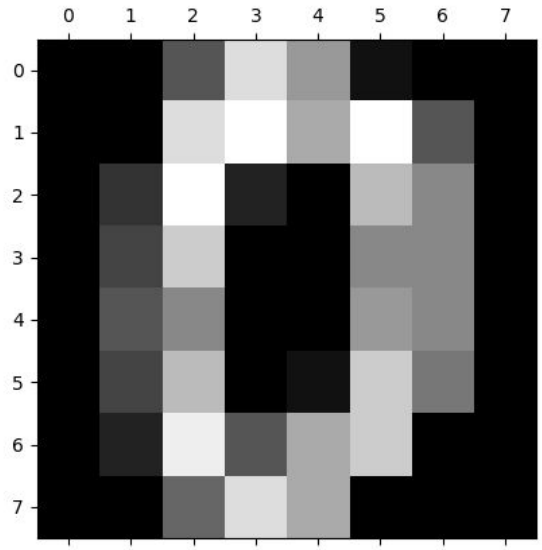
```
# De sklern/datasetes importa el método para cargar
from sklearn.datasets import load_digits

# importa el paquete para dibujar plots, y le pone un apodo
import matplotlib.pyplot as plt
# "plt" es un apodo, puede ser cualquier palabra no reservada
# ahora "plt" es lo mismo que escribir "matplotlib.pyplot"


digits = load_digits()  # asqui se cargan los datos
print(digits.data.shape)

# plt.gray() = matplotlib.pyplot.gray()
plt.gray()  # ¿Más bonito no? :)
plt.matshow(digits.images[0])
plt.show()
```

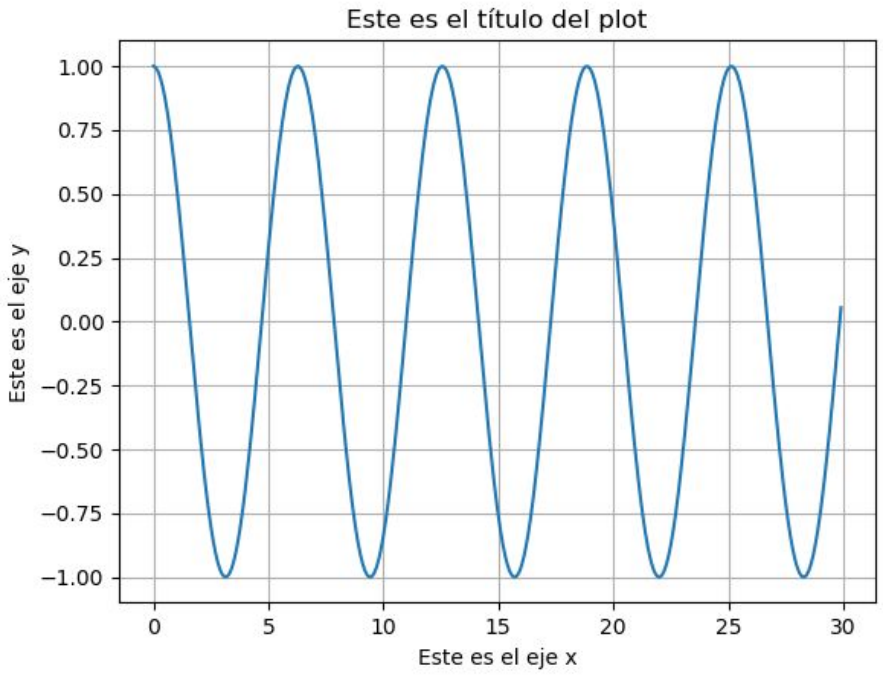
Su salida es la siguiente:



Ejercicio 17: Matplotlib

“Matplotlib se parece mucho a utilizar los plot en matlab”
Edite el código ejercicio17.ipynb en COLAB para que no marque errores, vea la documentación en la página oficial de matplotlib

Resultado esperado:

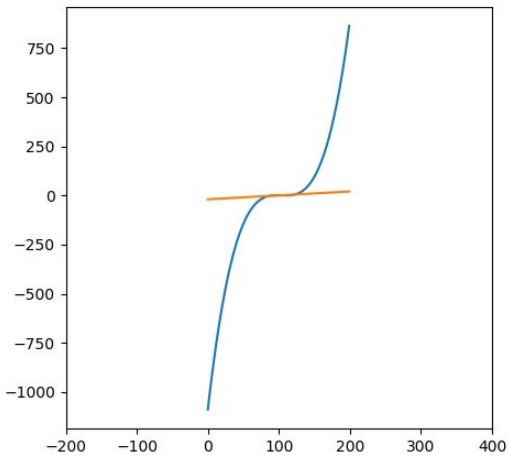


Ejercicio 18: Plot de una función

Edite el código ejercicio18.ipynb en COLAB
Con base en el ejercicio anterior, dibuje en un mismo plot las funciones:
f1(x) = 2x + 1
f2(x) = 5x^2 + 3x^2 + x + 0.5

Utilice las funciones anónimas lambda, cree el arreglo de valores x de [-10, 10] con una resolución de 0.1 para plotear ambas funciones.

Resultado esperado:



Ejercicio 19: Arreglos numpy

Los arrays Numpy son una excelente alternativa a las listas de Python. Algunas de las ventajas clave de los arrays Numpy es que son rápidos, fáciles de trabajar con ellos, y ofrece a los usuarios la oportunidad de realizar cálculos a través de arrays completos.

Edite el código ejercicio19.ipynb en COLAB

- **Convierta un arreglo de python a un arreglo numpy**
- **Cree un arreglo numpy con los números del 1 al 12, utilice arange**
- **Convierta el arreglo numpy a un a matriz de 4 filas x 3 columnas**
- **Convierta el arreglo numpy a un a matriz de 2 filas x 6 columnas**
- **Intente convertir en una matriz de 7 x 2, saque sus conclusiones**

Resultado esperado:

```
<class 'list'>
<class 'numpy.ndarray'>
numeros:  [ 1  2  3  4  5  6  7  8  9 10 11 12]
tipo(numeros)  <class 'numpy.ndarray'>
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

[[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]]
Traceback (most recent call last):
```

```
ValueError: cannot reshape array of size 12 into shape (7,2)
```

Ejercicio 20: OS escritura de archivos

El módulo os de Python le permite a usted realizar operaciones dependiente del Sistema Operativo como crear una carpeta, listar contenidos de una carpeta, conocer acerca de un proceso, finalizar un proceso, etc.

Edite el código ejercicio20.ipynb en COLAB

- **Cree una carpeta llamada folder_1, si esta ya esta creada borrela y cree una nueva**
- **Cree un archivo CSV de nombre "archivo.csv" dentro de la carpeta folder_1**
  - **Debe tener el encabezado "número", "color"**
  - **Con un ciclo for itere la lista de colores**
  - **En cada ciclo escribirá una fila en el archivo que contiene:**
    - **<el_ciclo_actual_del_for>, <el_color_actual_de_la_lista>**
- **Abra el archivo y examine su contenido.**

Resultado esperado:
Un archivo "archivo.csv" dentro de una carpeta "folder_1" con los datos

```
número,color
0,rojo
1,verde
2,azul
3,magenta
4,cian
5,amarillo
6,marrón
7,violeta
8,naranja
9,blanco
10,negro
11,gris
```

Ejercicio 21: Lectura de archivos csv

Requiere del archivo creado por el ejercicio 16.

Edite el código ejercicio21.ipynb en COLAB

- **Lea el archivo.csv creado por el ejercicio 16**
- **Saltarse el encabezado del archivo ['número', 'color']**
- **En un diccionario guarde fila por fila del archivo, donde el dato de la primer columna es la llave, y la segunda columna el valor con formato**
  Diccionario = {int:str, int:str, ...}

Resultado esperado:

```
['0', 'rojo']
['1', 'verde']
['2', 'azul']
['3', 'magenta']
['4', 'cian']
['5', 'amarillo']
['6', 'marrón']
['7', 'violeta']
['8', 'naranja']
['9', 'blanco']
['10', 'negro']
['11', 'gris']
Diccionario:
{0: 'rojo', 1: 'verde', 2: 'azul', 3: 'magenta', 4: 'cian', 5: 'amarillo', 6: 'marrón', 7:
'violeta', 8: 'naranja', 9: 'blanco', 10: 'negro', 11: 'gris'}
```

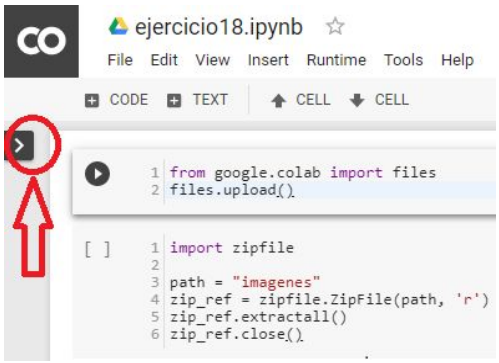
Archivos en COLAB

Ejercicio 22: Cargar archivos a COLAB

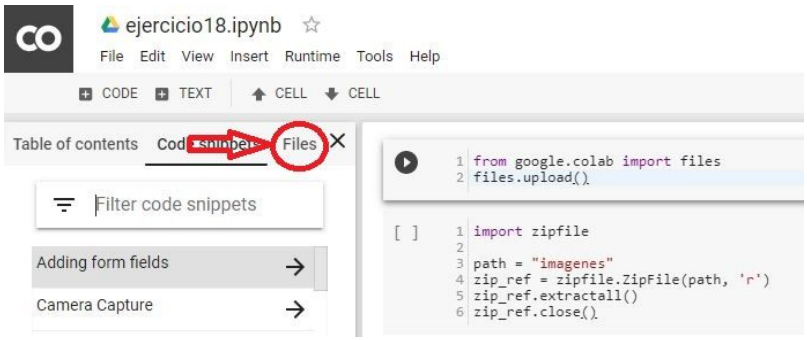
Siga las instrucciones paso a paso

Abra el código ejercicio22.ipynb en COLAB

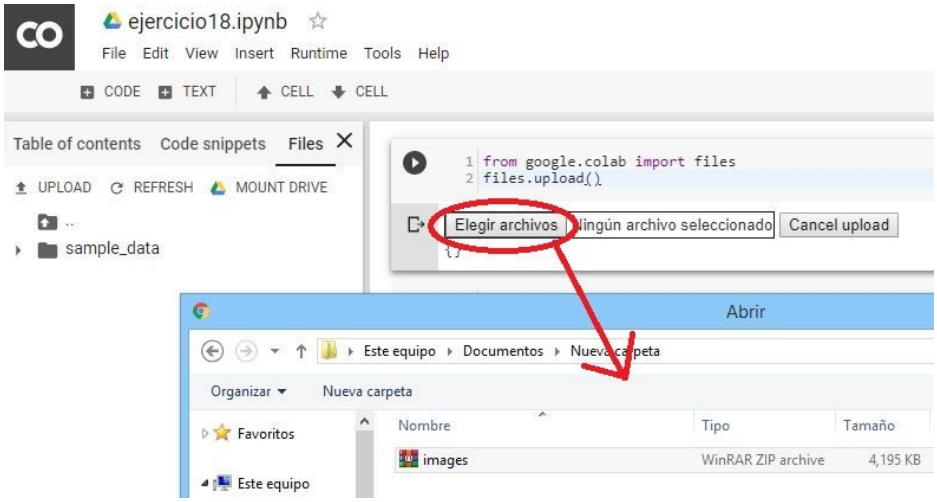
- **Despliegue la pestaña que se encuentra señalada en la imagen**



- **Seleccione la pestaña Files**

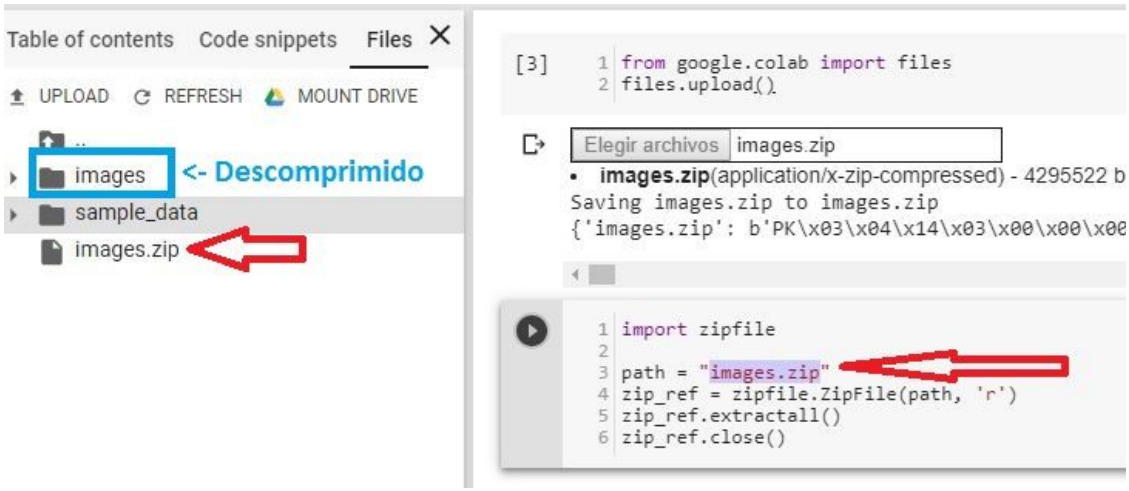


- **Ejecute la primer celda y seleccione el botón "Elegir archivos", busque la carpeta images.zip que se le compartio con este material en su equipo**



- **Ejecute la segunda celda que descomprime el archivo, verifique que el nombre en path sea el mismo que el archivo que subió**

Resultado esperado:
obtendrá la carpeta descomprimida "images"



Ejercicio 23: Open CV2

Requiere de la carpeta obtenida por el ejercicio 22.

Edite el código ejercicio23.ipynb en COLAB

- **Complete la primera celda**
- **En la segunda celda:**
  - **Guarde en una lista el contenido de la carpeta "images"**
  - **Convierta la lista a una lista numpy e imprima sus dimensiones con shape**
- **El la tercera celda imprima las dimensiones de cada imagen**

Resultado esperado Celda 2:

```
images/test4.jpg
images/test5.jpg
images/test_image.png
images/tigre.jpg
images/test0.jpg
images/gato.jpeg
images/test2.jpg
images/img_withheatmap.jpg
images/test3.jpg
images/test6.jpg
images/elephant.jpeg
images/bici1.jpg
images/perro.jpeg
images/indian_elephant.jpg
images/persona.jpg
images/caballo.jpg
images/test1.jpg

imagenes.shape =  (17,)
```

Resultado esperado Celda 3:

```
imagen 0 shape = (720, 1280, 3)
imagen 1 shape = (720, 1280, 3)
imagen 2 shape = (720, 1280, 3)
imagen 3 shape = (369, 642, 3)
imagen 4 shape = (720, 1280, 3)
imagen 5 shape = (190, 265, 3)
imagen 6 shape = (720, 1280, 3)
imagen 7 shape = (600, 899, 3)
imagen 8 shape = (720, 1280, 3)
imagen 9 shape = (720, 1280, 3)
imagen 10 shape = (600, 899, 3)
imagen 11 shape = (432, 768, 3)
imagen 12 shape = (168, 300, 3)
imagen 13 shape = (600, 1000, 3)
imagen 14 shape = (600, 800, 3)
imagen 15 shape = (1600, 2560, 3)
imagen 16 shape = (720, 1280, 3)
```

Ejercicio 24: Copiar y reescalar imágenes

Requiere de la carpeta obtenida por el ejercicio 22.

Edite el código ejercicio24.ipynb en COLAB

- **De la carpeta "images", cargue sólo las imágenes tigre.jpg, perro.jpeg, gato.jpeg y caballo.jpg**
  Tenga cuidado con la extensión, "jpeg" es diferente de "jpg"
- **En la segunda celda reescale las 4 imágenes a un tamaño de 150x150 y guardelas en una carpeta llamada "copia", si la carpeta no existe debe crearla. Guarde todas las copias con extensión jpg, utilice split para mantener los nombres originales**

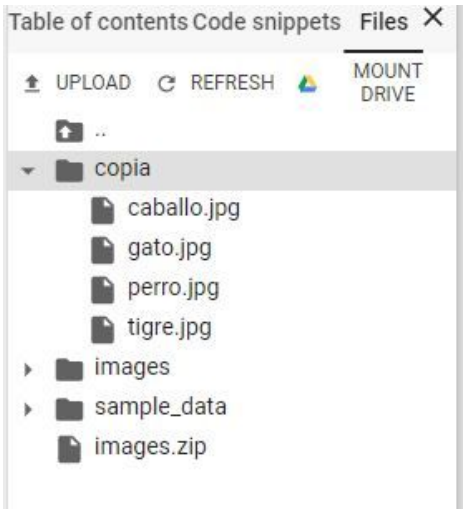
Resultado esperado Celda 1:

```
(369, 642, 3)
(168, 300, 3)
(190, 265, 3)
(1600, 2560, 3)
```

Resultado esperado Celda 2:

```
copia/tigre.jpg
(150, 150, 3) <class 'numpy.ndarray'>
copia/perro.jpg
(150, 150, 3) <class 'numpy.ndarray'>
copia/gato.jpg
(150, 150, 3) <class 'numpy.ndarray'>
copia/caballo.jpg
(150, 150, 3) <class 'numpy.ndarray'>
```

Resultado esperado en COLAB:
La carpeta copia con las 4 imágenes de tamaño 150x150



Ejercicio 25: Plot de imágenes

Requiere de la carpeta obtenida por el ejercicio 24.

Edite el código ejercicio25.ipynb en COLAB

- **En la primera celda, cargue las 4 imágenes de la carpeta "copia"**

Resultado esperado Celda 1:

```
copia/tigre.jpg
copia/gato.jpg
copia/perro.jpg
copia/caballo.jpg
```

- **En la segunda celda, haga un plot con matplotlib que muestre las 4 imágenes en forma de matriz 2x2**

Resultado esperado Celda 2:

```
(-0.5, 149.5, 149.5, -0.5)
```

- **Modifique la primer celda nuevamente para corregir los canales de las imágenes y corra nuevamente ambas celdas, busque en la documentación de CV2**

Nuevo resultado esperado Celda 2:

```
(-0.5, 149.5, 149.5, -0.5)
```

Ejercicio 26: Rectangulos con CV2

Requiere de la carpeta obtenida por el ejercicio 22.

Edite el código ejercicio26.ipynb en COLAB

Cargue la imagen "test1.jpg" de la carpeta "images". Con CV2 cree 2 rectángulos (rojo y verde), el rojo deberá encerrar al carro negro y el color verde al carro blanco.

Resultado esperado:

Ejercicio 27: numpy save npz

numpy.savez() Guarda varios arreglos en un solo archivo en formato .npz sin comprimir.
En este ejercicio guardará datos con numpy en un tipo de archivo, posteriormente recuperara la información de ese mismo archivo.

Edite el código ejercicio27.ipynb en COLAB

Cargue el conjunto de datos boston de sklearn en la primer celda y observe las dimensione de las tuplas "data" y "target". Itere los primeros 10 datos de ambas tuplas en un ciclo for al mismo tiempo.

- **En la celda 2 guarde los datos como un archivo npz**
- **En la tercer celda recupere los datos e imprima para corroborar su integridad**

Resultado esperado Celda 1:

```
X.shape = (506, 13)
Y.shape = <class 'tuple'>
dato 0 :
X= [0.0063218.02.310.00.5386.57565.24.091.0296.015.3396.94.98]
Y=  24.0
----
dato 1 :
X= [0.027310.07.070.00.4696.42178.94.96712.0242.017.8396.99.14]
Y=  21.6
----
dato 2 :
X= [0.027290.07.070.00.4697.18561.14.96712.0242.017.8392.834.03]
Y=  34.7
----
dato 3 :
X= [0.032370.02.180.00.4586.99845.86.06223.0222.018.7394.632.94]
Y=  33.4
----
dato 4 :
X= [0.069050.02.180.00.4587.14754.26.06223.0222.018.7396.95.33]
Y=  36.2
----
dato 5 :
X= [0.029850.02.180.00.4586.4358.76.06223.0222.018.7394.125.21]
Y=  28.7
----
dato 6 :
X= [0.0882912.57.870.00.5246.01266.65.56055.0311.015.2395.612.43]
Y=  22.9
----
dato 7 :
X= [0.1445512.57.870.00.5246.17296.15.95055.0311.015.2396.919.15]
Y=  27.1
----
dato 8 :
X= [0.2112412.57.870.00.5245.631100.06.08215.0311.015.2386.6329.93]
Y=  16.5
----
dato 9 :
X= [0.1700412.57.870.00.5246.00485.96.59215.0311.015.2386.7117.1]
Y=  18.9
----
```

Resultado esperado Celda 2:

```
[0.63, 1800.0, 231.0, 0.0, 53.8, 657.5, 6520.0, 409.0, 100.0, 29600.0, 1530.0, 39690.0, 498.0] 24.0

[2.73, 0.0, 707.0, 0.0, 46.9, 642.1, 7890.0, 496.71, 200.0, 24200.0, 1780.0, 39690.0, 914.0] 21.6

[2.73, 0.0, 707.0, 0.0, 46.9, 718.5, 6110.0, 496.71, 200.0, 24200.0, 1780.0, 39283.0, 403.0] 34.7

[3.24, 0.0, 218.0, 0.0, 45.8, 699.8, 4580.0, 606.22, 300.0, 22200.0, 1870.0, 39463.0, 294.0] 33.4

[6.9, 0.0, 218.0, 0.0, 45.8, 714.7, 5420.0, 606.22, 300.0, 22200.0, 1870.0, 39690.0, 533.0] 36.2

[2.99, 0.0, 218.0, 0.0, 45.8, 643.0, 5870.0, 606.22, 300.0, 22200.0, 1870.0, 39412.0, 521.0] 28.7

[8.83, 1250.0, 787.0, 0.0, 52.4, 601.2, 6660.0, 556.05, 500.0, 31100.0, 1520.0, 39560.0, 1243.0] 22.9

[14.46, 1250.0, 787.0, 0.0, 52.4, 617.2, 9610.0, 595.05, 500.0, 31100.0, 1520.0, 39690.0, 1915.0] 27.1
```

```
[21.12, 1250.0, 787.0, 0.0, 52.4, 563.1, 10000.0, 608.21, 500.0, 31100.0, 1520.0, 38663.0, 2993.0] 16.5

[17.0, 1250.0, 787.0, 0.0, 52.4, 600.4, 8590.0, 659.21, 500.0, 31100.0, 1520.0, 38671.0, 1710.0] 18.9
```

Resultado esperado Celda 3:

```
X= 0.631800.00231.000.0053.80657.506520.00409.00100.0029600.001530.0039690.00498.00
Y=  24.0
X= 2.730.00707.000.0046.90642.107890.00496.71200.0024200.001780.0039690.00914.00
Y=  21.6
X= 2.730.00707.000.0046.90718.506110.00496.71200.0024200.001780.0039283.00403.00
Y=  34.7
X= 3.240.00218.000.0045.80699.804580.00606.22300.0022200.001870.0039463.00294.00
Y=  33.4
X= 6.900.00218.000.0045.80714.705420.00606.22300.0022200.001870.0039690.00533.00
Y=  36.2
X= 2.990.00218.000.0045.80643.005870.00606.22300.0022200.001870.0039412.00521.00
Y=  28.7
X= 8.831250.00787.000.0052.40601.206660.00556.05500.0031100.001520.0039560.001243.00
Y=  22.9
X= 14.461250.00787.000.0052.40617.209610.00595.05500.0031100.001520.0039690.001915.00
Y=  27.1
X= 21.121250.00787.000.0052.40563.1010000.00608.21500.0031100.001520.0038663.002993.00
Y=  16.5
X= 17.001250.00787.000.0052.40600.408590.00659.21500.0031100.001520.0038671.001710.00
Y=  18.9
```

Extras

Histogramas

Los histogramas en python es un gráfico que muestra la distribución de frecuencias de una variable dada, para un conjunto de datos.

Bins: Son los recipientes contenedores de datos, divide el histograma completo en sub-intervalos y el valor de cada sub-intervalo se define con el tamaño de bins.

Ver documentación de numpy.histogram

Ejercicio 28: Histogramas

Edite el código ejercicio28.ipynb en COLAB

Requiere el conocimiento adquirido en los ejercicios 22, 25 y 27.

- **Suba a COLAB el archivo "datos_np.npz" que se le compartio con este material**

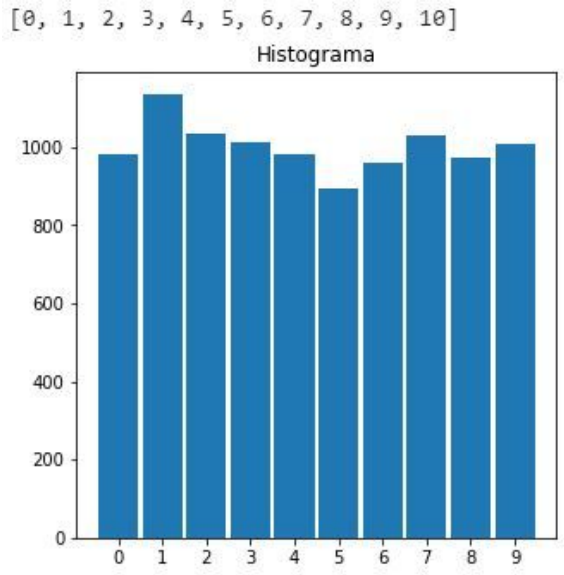
- **En la primer celda, recuerde cómo obtener datos de un archivo npz y extraiga los datos en la variable y**

Resultado esperado Celda 1:

```
Y_data
(10000,)
```

- **En la segunda celda deberá mostrar el histograma de los datos recuperados**

Resultado esperado Celda 1:



- **En la tercer celda, obtenga el número muestras de cada clase (los numeros 0, 1, … 9) e imprima la información**
  - Apóyese en numpy.where

Resultado esperado Celda 1:

```
Clase: 0
muestras =  980
Clase: 1
muestras =  1135
Clase: 2
muestras =  1032
Clase: 3
```

```
muestras =  1010
Clase: 4
muestras =  982
Clase: 5
muestras =  892
Clase: 6
muestras =  958
Clase: 7
muestras =  1028
Clase: 8
muestras =  974
Clase: 9
muestras =  1009
```

Aplicando conceptos

Los siguientes ejercicios pondrán a prueba los conocimientos adquiridos a lo largo de este documento

Ejercicio 29: plots 2D y 3D

Edite el código ejercicio29.ipynb en COLAB
Busque la información
Numpy: meshgrid, stack, linspace
Matplotlib: scatter, add_subplot, plot, plot_wireframe, contourf

Ejecute la primer celda de ejemplo para analizar el código
En la segunda celda realice:

- **Crear un espacio de valores de [-100 a 100]**
- **Genere un grid X, Y para plotear la superficie**
- **Definir la función z**
- **Genere 10 puntos (x, y) aleatorios**
- **Cree un arreglo vertical de coordenadas (x, y)**

Resultado esperado Celda 2:

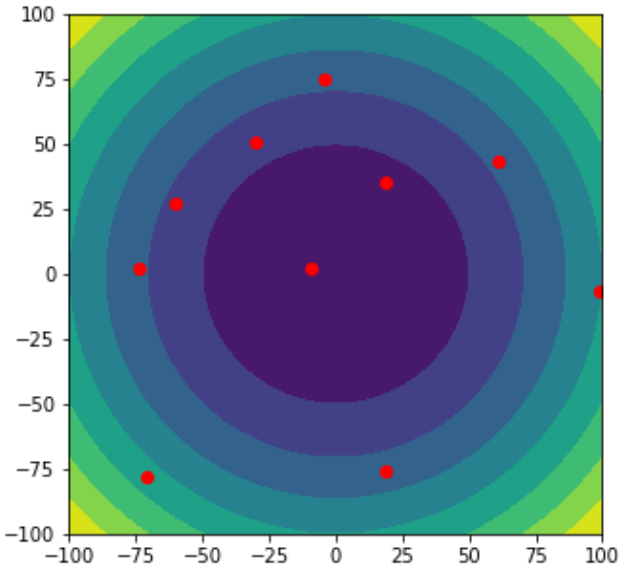
```
[ 99 -30 -74 -60 -71  -4  19  -9  61  19]
[ -7  51   2  27 -78  75  35   2  43 -76]
punto [99 -7]
```

```
Z= 9850
punto [-30  51]
Z= 3501
punto [-74   2]
Z= 5480
punto [-60  27]
Z= 4329
punto [-71 -78]
Z= 11125
punto [-4 75]
Z= 5641
punto [19 35]
Z= 1586
punto [-9  2]
Z= 85
punto [61 43]
Z= 5570
punto [ 19 -76]
Z= 6137
```

En la tercer celda realice:

- **El plot 2D de la superficie z con contourf**
- **Dibuje los 10 puntos generados sobre el mismo plot**

Resultado esperado Celda 3:

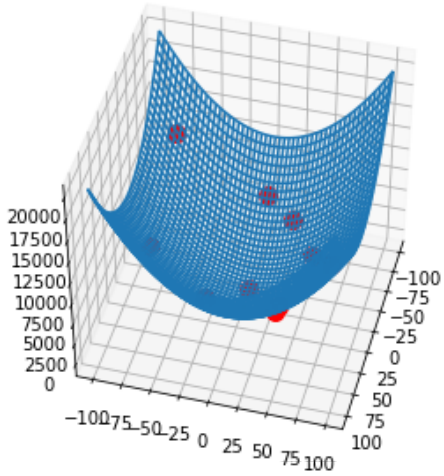


En la cuarta celda:

- **El plot 3D de la superficie z con plot_wireframe**

- **Dibuje los 10 puntos generados sobre el mismo plot**

Resultado esperado Celda 4:



Ejercicio 30: Convolución 2D

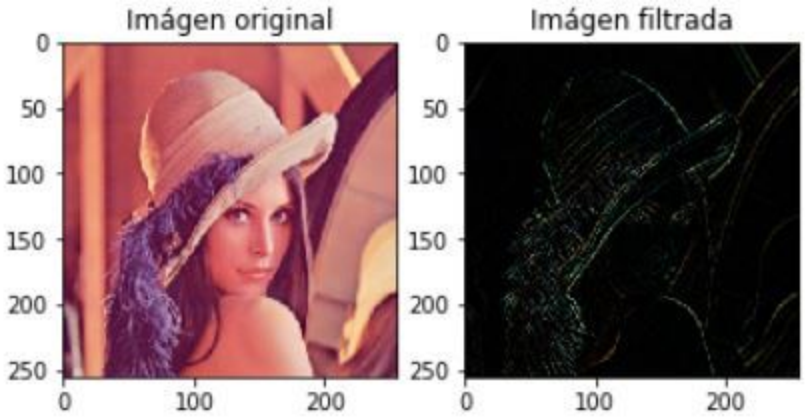
Edite el código ejercicio30.ipynb en COLAB

- **Suba la imagen "lena_color_256.tif" a COLAB**
- **En la primer celda cargue la imagen e implemente el filtro**

|1 0 -1|
|0 0 0|
|-1 0 0|

Resultado esperado Celda 1:

```
kernel:
 [[ 1.  0. -1.]
 [ 0.  0.  0.]
 [-1.  0.  1.]]
```

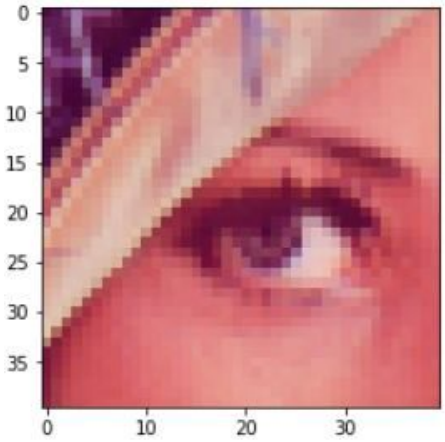


- **En la segunda celda genere una sub imagen de 40x40 con coordenada origen(110, 100) de la imagen original**
  Cree el filtro:

  |0 1 0|
  |1 1 1|
  |0 1 0|

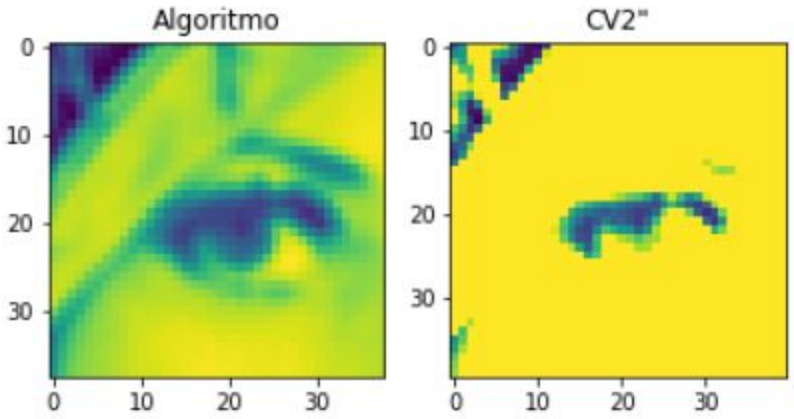
Resultado esperado Celda 2:

```
(3, 3)
[[0 1 0]
 [1 1 1]
 [0 1 0]]
```



- **En la tercer celda implemente el algoritmo de la convolución, siga las pistas del código. Imprima el resultado de la función convolucion2D y compárela con la implementación de CV2**

Resultado esperado Celda 3:


Algoritmo
CV2"
0
10
20
30
0
10
20
30
0
10
20
30
0
10
20
30