Soluciones

Ejercicio 1: Asignación de una variable

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

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

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

# En la siguiente línea, imprima la variable cadena
print(cadena)

# Asigne el número 5 a la variable numero e imprima su valor
numero = 5
print(numero)
```

Ejercicio 2: Asignación múltiple de variables

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 = 24, 13, 12

# Imprima el valor de los 3 artículos de la forma
# print('articulo=', variable)
print("cafe=", cafe)
print("galletas=", galletas)
print("papas=", papas)

# Imprima el valor de los 3 artículos de la forma
# print('artículo=', variable)
print(mi_dinero - cafe - papas - galletas)
```

```
# Imprima cuanto le quedaría si solo compra café y galletas
print(mi_dinero - cafe - galletas)
```

Ejercicio 3: Tipos de datos

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

```
cad = "Vehículo"
num = 250

print(cad + str(num))

# 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))
variable = 2.5 + 3
print("variable:", variable, "tipo:", type(variable))
```

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
concat = cadena_1 + cadena_2
print(concat)

# Calcule el área de un círculo de radio 3.33 e imprima su valor
area = PI * 3.33**2
print("El área es:", area)

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

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

```
x += 1
print("x=", x)
```

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
matriz.append(list_1[::-1])
matriz.append(list_2)
matriz.append(list_3[::-1])

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

# Imprima la matriz con un print
print(matriz)
```

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"]
otra_tupla = (lista, tupla)

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"
```

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 = diccionario.keys()
print("Imprimimos las llaves")
print(var_llaves)

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

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

```
# Modifica el segundo elemento
diccionario["llave2"] = 250
print(diccionario)
```

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"

```
# Defina la tupla con 10 elementos
tupla = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
lista = []  # No modificar
diccionario = {}  # No modificar

# Recorra la tupla con ciclo for y cada 2 elementos,
# haga una copia de ese elemento en una lista con append()
cont = 0
for elemento in tupla:
 if cont % 2 == 0:
   lista.append(elemento)
 cont += 1

# Imprima la lista con un for
for elemento in lista:
   print(elemento)

# Guarde los elementos de la lista con
# las llaves "uno", "dos", "tres", "cuatro", "cinco"
llaves = ["uno", "dos", "tres", "cuatro", "cinco"]
cont = 0
for elemento in lista:
   diccionario[llaves[cont]] = elemento
   cont += 1

# Imprimir el diccionario
print(diccionario)
```

Ejercicio 9: Condicionales

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**

```
# importamos el módulo para generar números aleatorios
import random

mi_lista = []
for i in range(20):
   mi_lista.append(str(i))

# Genera un número aleatorio entre [0 y 19]
indice_aleatorio = random.randint(0, 19)
mi_lista[indice_aleatorio] = "alto"

# Itere la lista con un ciclo while
contador = 0
while contador < len(mi_lista):
   if mi_lista[contador] is "alto":
       print(mi_lista[contador])
       print("Iteración actual: %d" % contador)
   contador += 1

# Itere la lista con for y detenga el ciclo al encontrar "alto"
for elemento in mi_lista:
   if elemento is "alto":
       break
   print(elemento)
```

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**

```
# Piramide
a, b, p_h = 3, 5, 2.1
```

```
# Cilindro
r, PI, c_h = 1.5, 3.14159, 2

def vol_piramide (a, b, h):
   a_base = a * b
   return (a_base * h) / 3


def vol_cilindro (r, pi, h):
   return r * pi * h

# Llame las funciones e imprima los valores
print("Voúmen de la piramide:", vol_piramide(a, b, p_h))
print("Voúmen del cilindro:", vol_cilindro(r, PI, c_h))
```

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**

```
import random
import math

lista = []
pares = []

# Ciclo for
for i in range(50):
   # Agregue un número aleatorio de {0 a 49]
   lista.append(random.randint(0, 49))

# Función
def obten_pares(x):
   nueeva_lista = []

   for item in x:
       if item % 2 == 0:
```

```
            nueeva_lista.append(item)

    return nueeva_lista

pares = obten_pares(lista)
print(pares)
var_map = map(lambda x: math.sin(x), pares)
print(list(var_map))
```

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

```
def fibonacci(elementos):
    # primeros 2 elementos
    x1 = 0
    x2 = 1
    contador = 0

    while contador < elementos:
        if contador < 1:
            contador += 1
            yield 1
        else:
            n = x1 + x2
            x1 = x2
            x2 = n
            contador += 1
            yield n


iteraciones = 10
fib = fibonacci(iteraciones)

for i in range(iteraciones):
    print(next(fib))
```

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**

```
cadena_larga = "never stop LEARNING because life never stops TEACHING"
archivos = ["imagen_1.jpg", "imagen_2.jpg", "imagen_3.jpg" , "imagen_4.jpg"]

# Corte la cadena en sub cadenas separadas por espacios
cadena_cortada = str.split(cadena_larga)
print(cadena_cortada)

# Divida las cadenas contenidas en archivos con separador "."
for cadena in archivos:
   cadena_cortada = cadena.split(".")
   print("arreglo de ", len(cadena_cortada), "elementos")
   print(cadena_cortada)

# Obtenga una subcadena manualmente (sin usar split)
# De cadena larga imprima la subcadena LEARNING por medio de los índices
sub = cadena_larga[11:18]
print(sub)
print("longitud:", len(sub))
```

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**

```
datos_list = ["Buenos días a todos y todas", 999999999999999, True, 125.193565145e-8]
```

```
for dato in datos_list:
   print()
   print(dato)
   if isinstance(dato, bool):
       print("{:d}".format(dato))  # TODO: Modifique esta línea

   elif isinstance(dato, int):
       print("{:.14E}".format(dato))  # TODO: Modifique esta línea

   elif isinstance(dato,float):
       print("{:.8f}".format(dato))  # TODO: Modifique esta línea

   elif isinstance(dato, str):
       print("{:>54}".format(dato))  # TODO: Modifique esta línea

##############
# Qué operador nos crea una lista con repeticiones
lista = [5] * 20
print(lista)
```

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**

```
# Defina la clase Automovil, no use acentos
class Automovil():

   def __init__(self):
       self.tanque = 0.0  # de [0,100]
       self.velocidad = 0.0  # de [0, 160]
       self.ocupantes = 0  # de [0, 5]
       self.encendido = False # Bool
```

```
    def encender_apagar(self):
        pass

    def cargar_gasolina(self):
        pass

    def conducir(self):
        pass

    def dar_un_ray(self):
        pass

    def probar_clase(self):
        print("El carro tiene:")
        print("Atributos:", self.tanque, self.velocidad, self.ocupantes)
        print("Encendido:", self.encendido)


# No modifique las siguientes líneas
carro = Automovil()

carro.probar_clase()
print(carro.encender_apagar())
print(carro.cargar_gasolina())
print(carro.conducir())
print(carro.dar_un_ray())
```

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)

```
# Defina la clase Automovil, no use acentos
class Automovil():

    def __init__(self):
        self.tanque = 0.0  # de [0,100]
        self.velocidad = 0.0  # de [0, 160]
        self.ocupantes = 0  # de [0, 5]
        self.encendido = False

    def encender_apagar(self):
        if not self.encendido and self.tanque > 0:
            self.ocupantes = 1
            self.encendido = True
        else:
            self.ocupantes = 0
            self.encendido = False

        print("Encendido: %s, personas %d" %  (self.encendido, self.ocupantes))

    def cargar_gasolina(self, litros=0):
        gass = self.tanque + litros
        if gass > 100:
            self.tanque = 100
        else:
            self.tanque = gass

        print("Gasolina: ", self.tanque)

    def conducir(self, velocidad=0, tiempo=0):
```

```
        for i in range(int(tiempo)):
            self.tanque -= velocidad * tiempo / 8.2
            if self.tanque < 0:
                self.tanque = 0;
                break

        print("Me quedan %f litros" % self.tanque)

    def dar_un_ray(self, personas=0):
        gente = self.ocupantes + personas
        if gente > 5:
            self.ocupantes = 5
        elif gente < 1:
            self.ocupantes = 1
        else:
            self.ocupantes = gente

        print("Somos %d en el carro" % self.ocupantes)


# No modifique las siguientes líneas
carro = Automovil()

carro.cargar_gasolina(120)
carro.encender_apagar()
carro.conducir(80, 2)
carro.dar_un_ray(3)
carro.conducir(100, 2)
carro.encender_apagar()
```

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

```
import matplotlib.pyplot as plt
import numpy as np

# Datos:
x = np.arange(0.0, 30.0, 0.1)  # Valores de X
y = np.cos(x)  # Y es la función coseno de X
```

```
# Se crea un objeto figura
fig, ax = plt.subplots()
ax.plot(x, y)  # Ésta linea genera el gráfico

# ax es el objeto que accede a los ejes (axis) para modificar sus propiedades
ax.set(xlabel='Este es el eje x', ylabel='Este es el eje y',
       title='Este es el título del plot')
ax.grid()
# .show() es importante, sin ella no se dibuja el plot cuando no tuliza notebooks
plt.show()
```

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.

```
import matplotlib.pyplot as plt
import numpy as np

# Datos:
x = np.arange(-10.0, 10.0, .1)  # Valores de X

y1 = map(lambda x: x**3 -x**2 - x + 1, x)
y2 = map(lambda x: 2*x+ 1, x)

# Otra forma de usar los plot es:
plt.figure(figsize=(5, 5))  # Crea la figura de un tamaño definido
plt.plot(list(y1))  # Dibuja la función y1
plt.plot(list(y2))  # Dibuja la función y2
plt.xlim((-200, 400))  # Delimita los valores en el eje X
plt.show()
```

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**

```
import numpy as np

lista_python = ["A", "B", "C", "D", "E", "F"]
arrego_numpy = np.array(lista_python)

print(type(lista_python))
print(type(arrego_numpy))

numeros = np.arange(1, 13, 1)
print("numeros: ", numeros)
print("tipo(numeros) ", type(numeros))

matriz = np.reshape(numeros, (4, 3))
print(matriz)

matriz = np.reshape(numeros, (2, 6))
print()
print(matriz)

# Descomente la siguiente linea para probar
#matriz = np.reshape(numeros, (7, 2))
print()
print(matriz)
```

Ejercicio 20: OS escritura de archivos

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.

```
# Módulo para manejar carpetas en el (SO)
import os
import csv

"""
Un archivo CSV (archivo de valores separados por comas)
es un tipo de archivo de texto simple que utiliza una estructura
específica para organizar los datos tabulares. Debido a que
es un archivo de texto sin formato, solo puede contener datos
de texto reales, es decir, caracteres ASCII o Unicode imprimibles.
"""
nombre_archivo = "archivo.csv"
nombre_carpeta = "folder_1"

# Este es el directorio donde se encuentra el archivo
print(os.path.join(nombre_carpeta, nombre_archivo))

# Si existe el archivo, borrarlo
if os.path.exists(os.path.join(nombre_carpeta, nombre_archivo)):
   os.remove(os.path.join(nombre_carpeta, nombre_archivo))

# Crear carpeta
if os.path.exists("folder_1"):  # Si existe, borrar
   os.rmdir("folder_1")

os.mkdir("folder_1")  # Crea la carpeta

colores = ["rojo", "verde", "azul", "magenta", "cian",
          "amarillo", "marrón", "violeta", "naranja",
          "blanco", "negro", "gris"]
```

```
with open('folder_1/archivo.csv', "w") as csv_file:
   fieldnames  = ['número', 'color']
   csv_writer = csv.DictWriter(csv_file, fieldnames=fieldnames)

   csv_writer.writeheader()
   for i, dato in enumerate(colores):
       csv_writer.writerow({'número':i, 'color':dato})
```

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, ...}

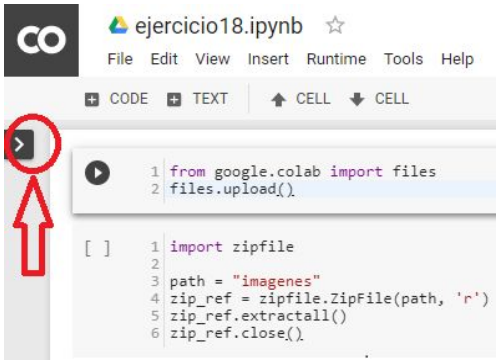
```
# Módulo para manejar carpetas en el (SO)
import csv
import os

nombre_archivo = "archivo.csv"
nombre_carpeta = "folder_1"
path = os.path.join(nombre_carpeta, nombre_archivo)
diccionario = {}

with open(path, "r") as csv_file:  # Abrir el archivo
   csv_reader = csv.reader(csv_file)
   next(csv_reader)  # le la primer línea (header)
   for fila in csv_reader:  # por cada fila en el archivo
       if len(fila) > 1:  # Si la fila tiene 2 o más datos
           print(fila)
           diccionario[int(fila[0])] = fila[1]  # guardar datos

print("Diccionario:")
print(diccionario)
```

Ejercicio 22: Cargar archivos a COLAB

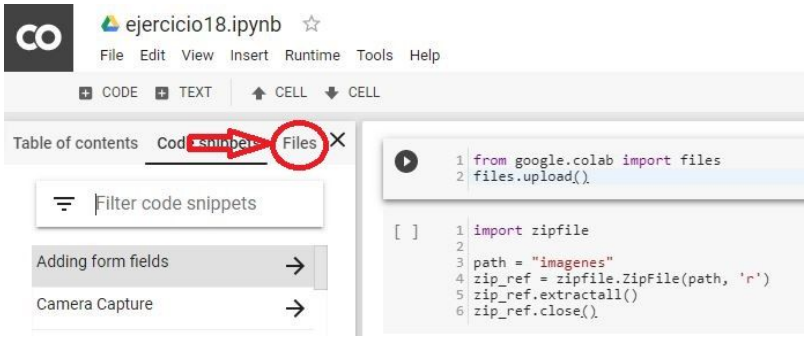
Siga las instrucciones paso a paso

Abra el código ejercicio22.ipynb en COLAB

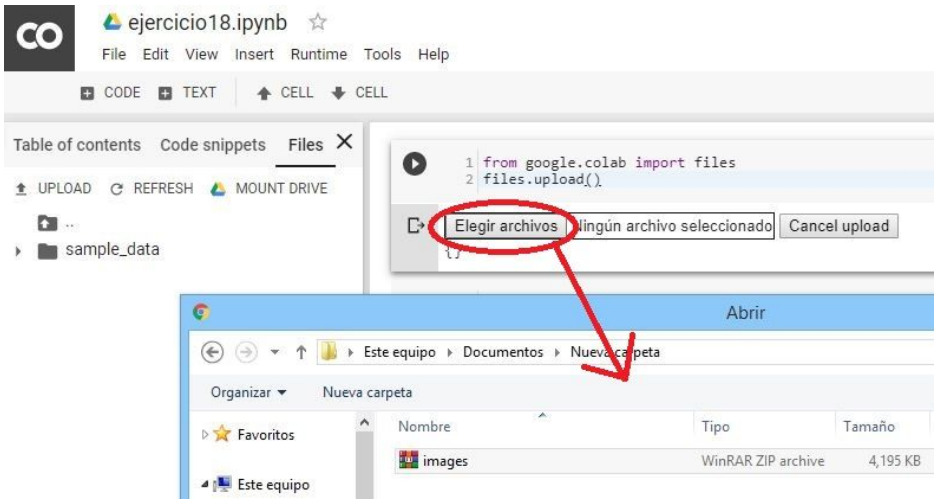
- **Despliegue la pestaña que se encuentra señalada en la imagen**



- **Seleccione la pestaña Files**

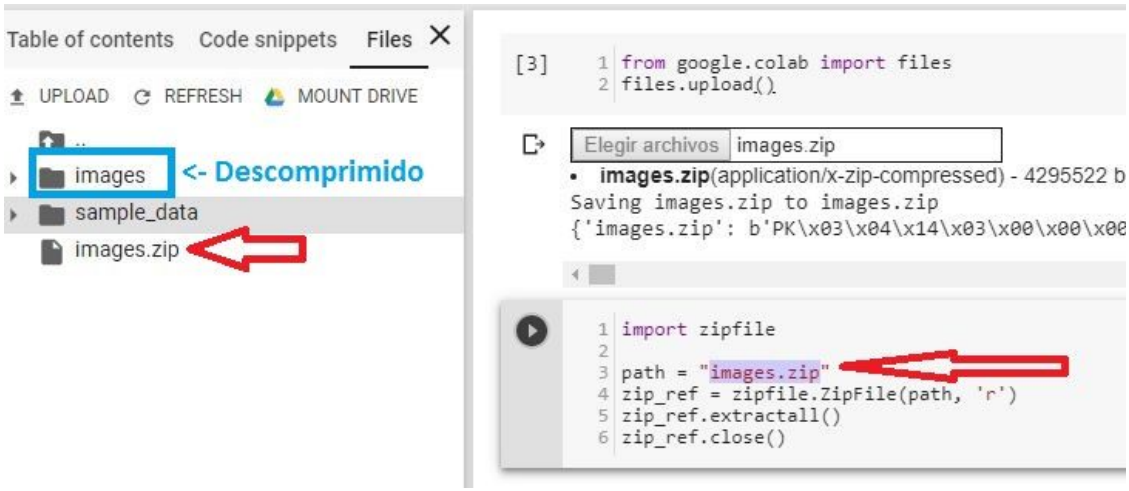


- **Ejecute la primer celda y seleccione el botón "Elegir archivos", busque la carpeta images.zip que se le compartio con este material en su equipo**



- **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”




```
from google.colab import files
files.upload()
```

```
import zipfile

path = "images.zip"
zip_ref = zipfile.ZipFile(path, 'r')
zip_ref.extractall()
zip_ref.close()
```

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**

```
import cv2
import numpy as np
import os

nombres = []
path = "images"
```

```
imagenes = []

# listar contenido de un directorio
nombres = os.listdir(path)

for archivo in nombres:
  ruta = os.path.join(path, archivo)
  print(ruta)
  imagenes.append(cv2.imread(ruta))

# Convertir a numpy array para su manipulación
imagenes = np.asarray(imagenes)
print("\nimagenes.shape = ", imagenes.shape)
```

```
for i, img in enumerate(imagenes):
  print("imagen %d shape = %s" % (i, img.shape))
```

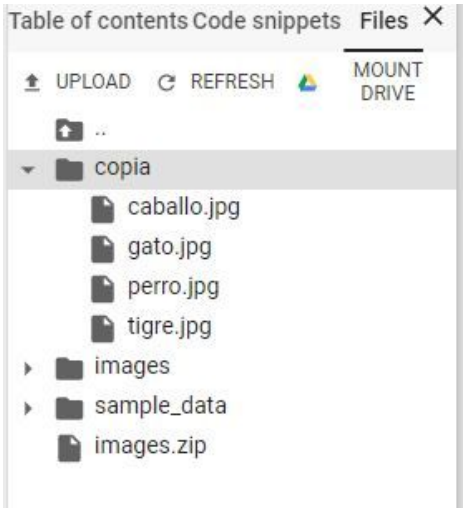
Ejercicio 24: Copiar y reescalar imágenes

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




```
import cv2
import numpy as np
import os

nombres = ["tigre.jpg", "perro.jpeg", "gato.jpeg", "caballo.jpg"]
imagenes = []

for archivo in nombres:
  imagenes.append(cv2.imread(os.path.join("images", archivo)))

imagenes = np.array(imagenes)
for file in imagenes:
  print(file.shape)
```

```
nueva_carpeta = "copia"

# Crear carpeta si no exciste
if not os.path.exists(nueva_carpeta):
    os.mkdir(nueva_carpeta)

nuevas_imagenes = []
dimension = (150, 150)

# Reescalar imágenes
for imangen, nombre in zip(imagenes, nombres):
  re_img = cv2.resize(imangen, dimension, interpolation = cv2.INTER_AREA)
  nuevas_imagenes.append(re_img)
  cad = nombre.split(".")
  # Ruta donde se guardará la imagen
```

```
  path = nueva_carpeta + "/" + cad[0] + ".jpg"
  print(path)
  print(re_img.shape, type(re_img))
  cv2.imwrite(path, re_img)
```

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"**
- **En la segunda celda, haga un plot con matplotlib que muestre las 4 imágenes en forma de matriz 2x2**
- **Modifique la primer celda nuevamente para corregir los canales de las imágenes y corra nuevamente ambas celdas, busque en la documentación de CV2**

```
import cv2
import os

# listar contenido de un directorio
nombres = os.listdir("copia")
imagenes = []

for archivo in nombres:
  ruta = os.path.join("copia", archivo)
  print(ruta)
  img = cv2.imread(ruta)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
  imagenes.append(img)
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(5, 5))
plt.subplot(2, 2, 1)
plt.imshow(imagenes[0])
# TODO: Remover ejes
plt.axis('off')

plt.subplot(2, 2, 2)
plt.imshow(imagenes[1])
# TODO: Remover ejes
```

```
plt.axis('off')

plt.subplot(2, 2, 3)
plt.imshow(imagenes[2])
# TODO: Remover ejes
plt.axis('off')

plt.subplot(2, 2, 4)
plt.imshow(imagenes[3])
# TODO: Remover ejes
plt.axis('off')
```

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.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

imgname = 'images/test1.jpg'
imgCV = cv2.imread(imgname)     #Imagen para visualizar

# Modificar cajas
rect_r = ((810, 410),(950, 500))
rect_v = ((1050, 400),(1270, 510))

# Dibujar rectángulos
cv2.rectangle(imgCV, rect_r[0], rect_r[1], (0,0,255), 4)
cv2.rectangle(imgCV, rect_v[0], rect_v[1], (0,255,0), 4)

# Correjir canales
imgCV = cv2.cvtColor(imgCV, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(18, 18))
plt.imshow(imgCV)
plt.title('CV2')
plt.show()
```

Ejercicio 27: numpy save npz

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**

```
import numpy as np
from sklearn.datasets import load_boston

boston = load_boston()

print("X.shape =", boston.data.shape)
print("Y.shape =", type(boston.target.shape))

x_list = []
y_list = []

for i, (x, y) in enumerate(zip(boston.data, boston.target)):
    temp = []
    if i == 10:
        break  # Terminar el ciclo for
    print("dato %d :" % i)
    print("X= [", end = '')
    for valor in x:
        temp.append(round(valor * 100, 2))
        print(valor , end = '')

    print("]\nY= ", y)
    print("\n----")
    x_list.append(temp)
    y_list.append(y)
```

```
for x, y in zip(x_list, y_list):
    print(x, y)
    print()

np.savez("numpy_zip.npz", X=x_list, Y=y_list)
```

```
archivo_numpy = np.load("numpy_zip.npz")
```

```
n_x = archivo_numpy["X"]
n_y = archivo_numpy["Y"]

for x, y in zip(n_x, n_y):

  print("X= ",end = '')
  for val in x:
    print("{:.2f}".format(val), end = '')
  print("\nY= ", y)
```

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

```
import numpy as np
datos = np.load("datos_np.npz")

# Buscamos las llaves
for key in datos.files:
  print(key)

y = datos["Y_data"]
print(y.shape)
```

```
# Histogramas
import matplotlib.pyplot as plt
import numpy as np
```

```
bins = list(range(11))
print(bins)

# () intervalo avierto, [] intevalo cerrado
# Los bins utilizan los rangos de la siguiente forma
# Bins(10): [0,1), [1, 2)... [7, 8), [8, 9]

plt.figure(figsize=(5, 5))
plt.hist(y, bins=np.arange(-0.5, 10.5), rwidth=.9)
plt.title("Histograma")
plt.xticks(range(10))

plt.show()
```

```
llaves = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

for llave in llaves:
  index = np.where(y == llave)

  print("Clase:", y[index][0])
  print("muestras = ", y[index].shape[0])
```

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)**

En la tercer celda realice:

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

En la cuarta celda:

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

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

```
# Cree un espacio de valores de [-100 a 100]
rango = np.linspace(-100, 100, 100)
# Genere un grid X, Y para plotear la superficie
X, Y = np.meshgrid(rango, rango)

# Defina la función z
def funcion_z(x, y):
  return x**2 + y**2

# Genere 10 puntos (x, y) aleatorios
puntos = []
px = np.random.randint(-100, 100, 10)
py = np.random.randint(-100, 100, 10)
print(px)
print(py)

# Cree un arreglo vertical de coordenadas (x, y)
# combinando los arreglos px y py
puntos = np.stack((px, py), axis=1)

for punto in puntos:
  print("punto %s" % punto)
  print("Z= %d" % funcion_z(punto[0], punto[1]))
```

```
# Defina la figura del plot
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111)
# Dibuje la proyeccion 2D de la superficie con contourf
plt.contourf(X, Y, funcion_z(X, Y))
# Dibuje los putos generados aleatoriamente con una lista de Comprensión
[plt.plot(punto[0], punto[1], 'ro') for punto in puntos]
plt.show()
```

```
# Realice el plot en 3D
fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(111, projection="3d")
objeto = ax.plot_wireframe(X, Y, funcion_z(X, Y))
[ax.scatter(punto[0], punto[1], funcion_z(punto[0], punto[1]), c='r', marker='o', s=100 ) for
punto in puntos]
ax.view_init(45, 15)
```

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|
- **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|
- **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**

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

imgpath = "lena_color_256.tif"
# Cargar imagen
img = cv2.imread(imgpath, 1)
# Corregir canales
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Implemente el kernel
k = np.array(([1, 0, -1], [0, 0, 0], [-1, 0, 1]), np.float32)
print("kernel:\n", k)

salida = cv2.filter2D(img, -1, k)
plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title('Imágen original')

plt.subplot(1, 2, 2)
plt.imshow(salida)
plt.title('Imágen filtrada')
plt.show()
```

```
import numpy as np
# Filtro (kernel)
filtro_k = np.array([[0 , 1, 0], [1, 1, 1], [0, 1, 0]])

sub_img = img[110:150, 110:150, :]

print(filtro_k.shape)
print(filtro_k)

plt.imshow(sub_img)
plt.show()
```

```
def convolucion2D(image, kernel, bias):
  k_x, k_y = kernel.shape

  if (k_x == k_y):
    k = k_x
    y, x = image.shape
    y = y - k + 1
    x = x - k + 1

    f_image = np.zeros((y,x))

    for i in range(y):
      for j in range(x):
        # PISTA:
        # con python los arrays pueden multiplicarse o sumarse directamente
        f_image[i][j] = np.sum(image[i:i+k, j:j+k]*kernel) + bias

    return f_image

# Provar función
convoluted = convolucion2D(sub_img[:, :, 0], filtro_k, 0)
plt.subplot(1, 2, 1)
plt.imshow(convoluted)
plt.title('Algoritmo')

# Comparación CV2
verificacion = cv2.filter2D(sub_img[:,:,1], -1, filtro_k)
plt.subplot(1, 2, 2)
plt.imshow(verificacion)
plt.title('CV2"')
plt.show()
```