

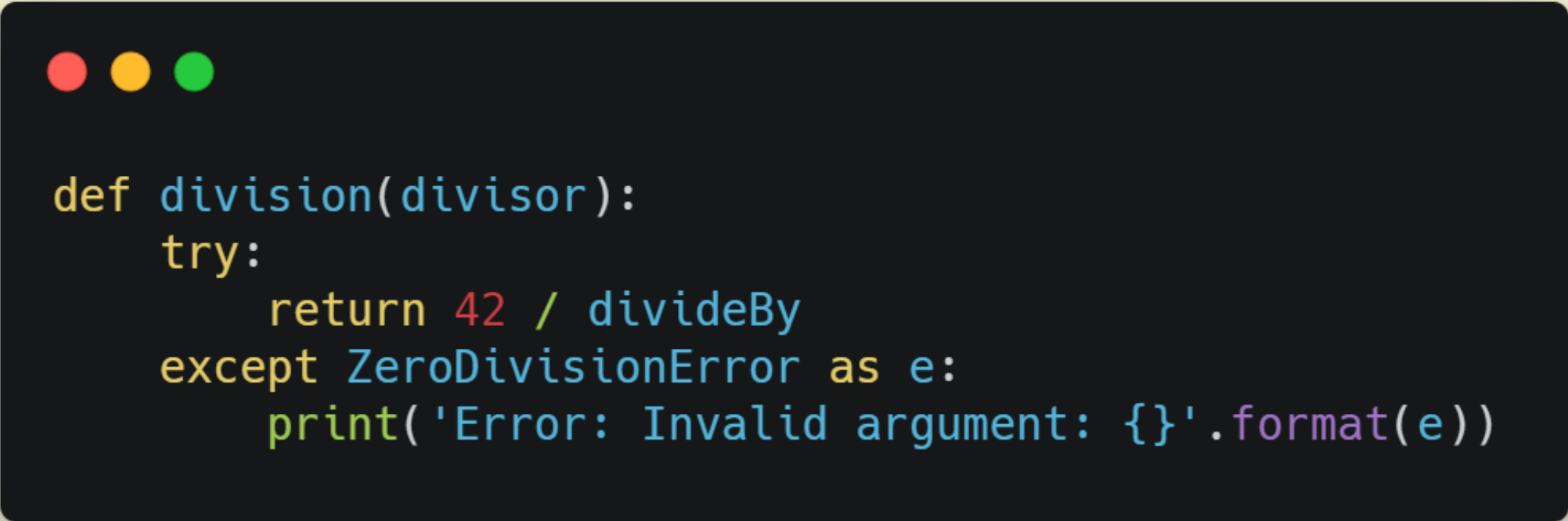


# Manejo de exepciones, numpy y pandas

Gutiérrez Cruz Abel Isaías

# Manejo de excepciones


Al momento de desarrollar un algoritmo que interactúe directamente con el usuario es altamente posible que en algún paso se tengas las condiciones específicas para obtener un error. Para esto es necesario proporcionar al usuario un mensaje que le ayude a descubrir lo que hizo mal

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a Python function definition for handling division errors.

```
def division(divisor):  
    try:  
        return 42 / divideBy  
    except ZeroDivisionError as e:  
        print('Error: Invalid argument: {}'.format(e))
```

El tipo de error se puede obtener al momento en el que se ejecuta el error. En donde se este ejecutando el código Python aparece el nombre del error.

En el caso de que se quiera ejecutar un fragmento de código inmediatamente después de que se encuentre un error, entonces se debe usar la siguiente estructura:



```
def division(divisor):  
    try:  
        return 42 / divisor  
    except ZeroDivisionError as e:  
        print('Error: Invalid argument: {}'.format(e))  
    finally:  
        print("-- división terminadaa --")
```

## Crear un error personalizado




```
raise TyperError("Esto es un error personalizado")
```

Al momento de llegar a esta línea de código, el Python arrojará un error, pero no seguirá con las líneas de código que faltan ejecutar

# Manejar directorios


Cuando se están manejando archivos desde un ambiente como un lenguaje de programación, se vuelve indispensable trabajar con carpetas de nuestro sistema operativo. Para esta tarea es muy útil el modulo "os" de Python

## Cargar el modulo en nuestro archivo de Python



```
import os
```

## Obtener la carpeta en la que nos encontramos ubicados



```
os.getcwd( )
```

## Establer una nueva carpeta de trabajo



```
os.chdir( '<direccion>' )
```

## Unir varios elementos para hacer una dirección de alguna carpeta



```
os.path.join( '<carpeta1>', '<carpeta2>', '<carpeta3>' )
```

# Crear una carpeta



```
os.makedirs('<direccion a la carpeta>')
```

# Direcciones absolutas vs relativas

## Dirección absoluta

Siempre comienza con la carpeta raíz

```
os.getcwd()  
# output: C:\Users\Usuario\Desktop
```

## Dirección relativa

Toma como referencia la dirección con la cual se está trabajando actualmente

```
os.getcwd()  
# output: C:\Users\Usuario\Desktop  
os.chdir("../")  
# output: C:\Users\Usuario
```





## ¿Qué es?

Es una biblioteca de Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con otra colección de funciones matemáticas de alto nivel para trabajar con ellas.

## Trabajar con arrays (vectores)

### Importar la librería

```
import numpy as np
```

## Creación de un array con valores aleatorios



```
np.random.randn(<filas>, <columnas>)
```

## Determinar el número de filas y columnas del array




```
<array>.shape
```

## Determinar las dimensiones que tiene el array



```
<array>.ndim
```

## Creación de un array a partir de una lista de Python




```
arr = np.array([1, 2, 3, 4, 5, 6])  
arr = arr.reshape((1, 6))
```

# Vectorización (operaciones aritméticas con Numpy)

Una funcionalidad muy útil de Numpy es vectorizar las operaciones aritméticas, de tal modo que se pueda operar con cada uno de los elementos del array de forma directa.

La siguiente operación con una lista de Python obtendrá un error:



```
lista = [1, 2, 3, 4, 5, 6]  
lista * 6
```

Para llevar a cabo la anterior operación se tendría que optar por la siguiente opción, lo cual es muy lento.

```
lista = [1, 2, 3, 4, 5, 6]
for i in range(0, len(lista)):
    lista[i] = lista[i]*6

# output: [6, 12, 18, 24, 30, 36]
```

Una forma eficiente de hacerlo, es con un array de numpy

```
arr = np.array([1, 2, 3, 4, 5, 6])
arr = arr.reshape((1, 6))
arr * 6
# output: array([[ 6, 12, 18, 24, 30, 36]])
```

# Vectorización



```
arr = np.array([1, 2, 3, 4, 5, 6])  
arr = arr.reshape((1,6))  
arr * 6
```

1	2	3	4	5	6
---	---	---	---	---	---

\*

6
---



1	2	3	4	5	6
---	---	---	---	---	---

6	6	6	6	6	6
---	---	---	---	---	---



6	12	18	24	30	36
---	----	----	----	----	----



Biblioteca escrita como extensión de NumPy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales

Variables

Observaciones


## Importar la libreria



```
import pandas as pd
```

## Leer archivos de Excel

Instalar xlrd



```
pip install xlrd
```





```
pd.read_excel('direccion del archivo.xlsx', '<nombre de la hoja>')
```

## Leer archivos csv



```
pd.read_csv('direccion del archivo.csv')
```

## Leer archivos con datos separados por tabulaciones



```
pd.read_csv('direccion del archivo.txt', delimiter="\t")
```

## Seleccionar filas en un dataframe de acuerdo a una condición en específico



```
<dataframe>.loc[<dataframe>['<nombre de columna' ] == <valor>]
```

## Seleccionar filas de acuerdo a su indice



```
<dataframe>.loc[<indice>]
```

## Obtener el tipo de datos de cada columna



```
<dataframe>.dtypes
```

## Seleccionar columnas de acuerdo a su nombre



```
<dataframe>[["<nombre columna 1>", "<nombre columna 2>"]]
```

## Ordenar el dataframe



```
<df>.sort_values(by = ['<nombre col>', '<nombre col2>'], ascending = <True o False>))
```

## Obtener las filas que se encuentran repetidas



```
<df>.duplicated()
```

## Remover las filas repetidas



```
<df>.drop_duplicates()
```

# Creación de dataframes

## A partir de un diccionario



```
datosDic = {"Articulos": ["Tenis", "Zapatos", "Camicetas", "Escritorio"],  
            "Sucursal": ["Guadalajara", "CDMX", "Guanajuato", "Sinaloa"]}  
data2 = pd.DataFrame(datosDic)
```

## Creación de dataframe vacío



```
pd.DataFrame(columns = ["Primer_nombre", "Apellido", "Genero"])
```

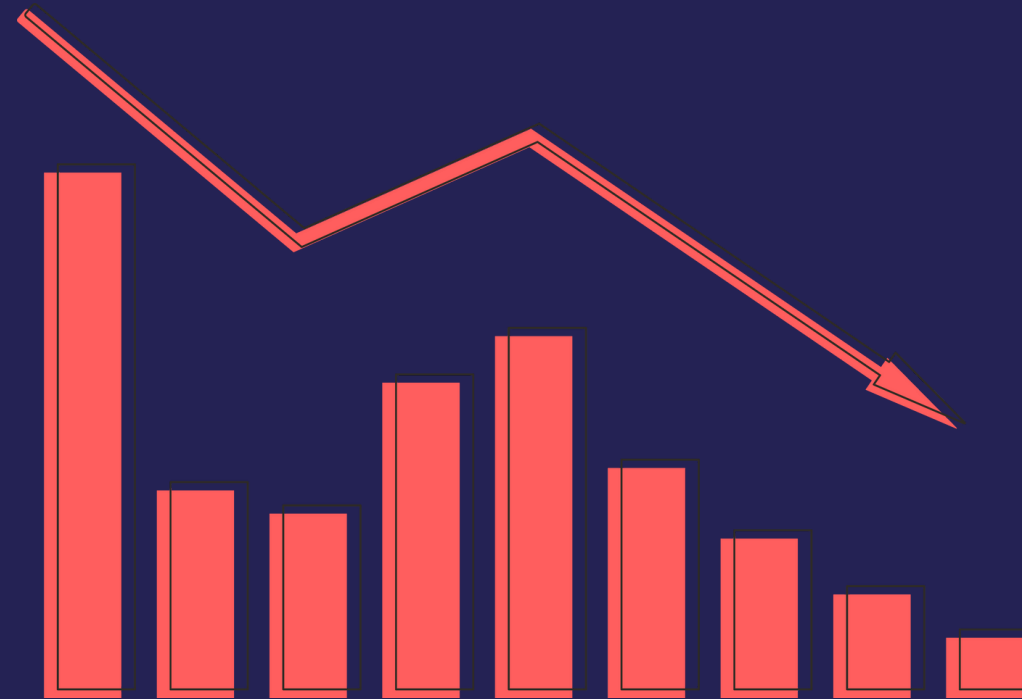
## Agregar filas a un dataframe vacío

Se agrega un diccionario con el nombre de cada columna y el valor que se desea ingresar



```
data2.append({'Primer_nombre': 'Josy', 'Apellido': 'Garcia', 'Genero': 'Mujer'}, ignore_index=True)
```

# Matplotlib



Es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje Python y su extensión matemática NumPy.



# Creación de un gráfico sencillo



```
import matplotlib.pyplot as plt

array1 = np.random.randn(10)
array2 = np.random.randn(10)

plt.plot(array1, array2)
plt.xlabel("Aleatorio 1")
plt.ylabel("Aleatorio 2")
```