



Buenas Practicas de Programación

Lección 1: Control de errores

Indice

Introducción.....	3
Descripción del código.....	4
Librerías empleadas.....	4
Abordaje del problema.....	4
Descripción de las Funciones.....	4
__init__(self, file).....	4
verify(self).....	5
clean(self).....	6
get_month_spend(self).....	7
get_month_income(self).....	8
get_month_save(self).....	8
gen_graph(self).....	9
selec_case(self, case) & select_title(self, case).....	10
Respondiendo a las cuestiones planteadas.....	11

Introducción

En esta practica se pretende poner a prueba los conocimientos acerca de control de errores, pruebas y validación de datos. Para ello se empleará un archivo csv en el que se han registrado una serie de datos simulando los gastos e ingresos de un usuario. En el archivo se hallan varias entradas no numéricas que generarán errores en la ejecución del código además, el programa deberá cerciorarse de que no existen meses sin datos y de que el archivo tiene una columna por cada mes.

Descripción del código

Librerías empleadas

Para la lectura del código se ha empleado la librería Pandas ya que es una de las librerías aprendidas durante este curso además resulta una librería muy cómoda a la hora de trabajar con Data Frames y grandes cantidades de información.

Por la parte del ejercicio opcional se ha empleado la librería plotly ya que resulta cómoda y sencilla y ya se ha empleado anteriormente por lo que la implementación de código de esta librería ya es conocida.

Abordaje del problema

Para la resolución del problema se ha decidido crear una clase llamada CSV, esto se debe a que las clases resultan útiles cuando se tiene que manejar la misma información desde diversas funciones. Al emplear una clase no resulta necesario leer el data frame en cada función que lo necesite, basta con leerlo en la función `__init__()` y guardarlo en una instancia de forma que se ahorra espacio en memoria y tiempo de ejecución. A la hora de tratar con las excepciones se han empleado los métodos `try` y `except` de forma que el código no se vea interrumpido por posibles errores en el archivo csv.

Descripción de las Funciones

`__init__(self, file)`

En este bloque se crea el Data Frame (`self.df`) que será el objeto principal del programa, tras ello se llama a las funciones que verifican y limpian el Data Frame, a continuación se llama a las funciones que procesan la información y devuelven los datos relevantes, se imprimen estos y finalmente se llama a la función que crea el grafico.

```
def __init__(self, file):
    self.df = pd.read_csv(file, delimiter='\t')
    self.verify()
    self.clean()
    self.get_month_spend()
    self.get_month_income()
    self.get_month_save()
    print(
        f'\nLa media de gasto Mensual ha sido: {self.sp_avg} \
\nEl gasto total a lo largo del año ha sido: {self.sp_t}\
\nEl ingreso total a lo largo del año ha sido: {self.ic_t}\
\nEl valance al finalizar el año es: {self.ic_t - self.sp_t}\n')
    input('Presione "Enter" para continuar...')
    self.gen_graph()
```

verify(self)

Este es el bloque de código encargado de verificar el Data Frame, para ello hace uso de las variables year y values.

La variable values representa una lista de ceros creada empleando la ‘comprensión de listas’, la year, por su parte, constituye una lista con los nombres de los meses, esta se recorre empleando sus elementos para llamar a las columnas del Data Frame con el mismo nombre, en caso de no existir dicha columna, se creará un error el cual será recogido en una excepción mediante la que se creará dicha columna con valor CERO empleando la variable values (es importante remarcar que la columna se crea con valor CERO y no NULO ya que el código crea errores para las columnas de valor NULO) tras imprimir por pantalla un mensaje indicando la acción correctiva adoptada por el código. Al tiempo que se comprueba que el Data Frame consta de doce columnas, se verifica que ninguna de ellas es NULA. En caso contrario se levanta un error de tipo ‘Exception’, el error es recogido por una excepción que muestra por pantalla un mensaje informando de que se ha hallado una columna NULA.

```
def verify(self):
    year = [
        'Enero', 'Febrero', 'Marzo',
        'Abril', 'Mayo', 'Junio',
        'Julio', 'Agosto', 'Septiembre',
        'Octubre', 'Noviembre', 'Diciembre']
    values = [0 for i in range(100)]
    for month in year:
        try:
            if self.df[month].dropna().empty:
                raise Exception
        except KeyError:
            print(f'Falta {month}, se tomará como nulo.')
            self.df.insert(
                year.index(month),
                month,
                values
            )
        except Exception:
            print(f'Faltan valores en {month}, se tomara como nulo.')
```

clean(self)

Tras verificar que la estructura del Data Frame es correcta se pasa a la limpieza del mismo, para lo cual se recorren las columnas del Data Frame una por una y los valores recogidos en las columnas uno por uno. Para la limpieza se tiene en cuenta un primer caso general; que el dato observado sea de tipo carácter (string). En caso afirmativo se pasará a analizar el dato mas a fondo.

Primero, se comprobará que el dato no contenga comillas, en cuyo caso, se eliminarán y se sustituirá el dato por un entero (integer) con su mismo valor, en caso de no cumplirse este primer caso, se pasará a intentar reemplazar el dato por un entero de su mismo valor, si se diera un ValueError porque el dato observado es necesariamente de tipo cadena de caracteres, se recogería este error en una excepción y se sustituiría el dato por un CERO.

Finalmente, el dato no fuera un objeto de tipo cadena de caracteres y fuera un dato NULO, se sustituiría por un CERO.

```
def clean(self):
    for col in self.df.columns:
        for item in self.df[col]:
            # n += 1
            if type(item) is str:
                # print(f'Elemto {n}: {item}, Type: {type(item)}')
                if item.startswith('\'):
                    self.df[col] = self.df[col].replace(
                        [item], int(item[1:-1]))
            else:
                try:
                    self.df[col] = self.df[col].replace(
                        [item], int(item))
                except ValueError:
                    self.df[col] = self.df[col].replace([item], 0)
            elif item is None:
                self.df[col] = self.df[col].replace([item], 0)
```

get_month_spend(self)

Este es el primer bloque de código dedicado a analizar el Data Frame. Con este bloque se suma el gasto total de cada mes. El valor obtenido es almacenado en un diccionario con la estructura {mes: valor} que se referencia mediante la instancia self.spend. Además se almacena el gasto total realizado a lo largo del año para obtener finalmente la media mensual de gasto almacenada en la instancia self.sp_avg.

En esta función se incluyen las líneas de código responsables de imprimir por pantalla la información relevante acerca de los gastos mensuales.

```
def get_month_spend(self):
    self.spend = dict() # Guarda el gasto de cada mes
    self.sp_t = 0 # guarda el gasto total a lo largo del año
    for col in self.df.columns:
        spend = 0
        for item in self.df[col]:
            if item < 0:
                spend += item
        self.spend[col] = spend
        self.sp_t += spend
    self.sp_avg = round(self.sp_t / 12, 2) # Guarda la media de gasto

    print('\nGasto por Meses:')
    max = 0
    month = str()
    for i in self.spend:
        if self.spend[i] < max:
            max = self.spend[i]
            month = i
        print(f'\t{i}: {self.spend[i]}')
    print(f'Mes con mayor gasto: {month}({max})')
```

get_month_income(self)

Esta función se encarga de calcular el total por meses, almacenado en la instancia `self.income` que hace referencia a un diccionario con la misma estructura que en el empleado para los gastos y el total anual de ingresos, almacenado en la instancia `self.ic_t`.

```
def get_month_income(self):
    self.income = dict()
    self.ic_t = 0 # guarda el ingreso total a lo largo del año.
    for col in self.df.columns:
        save = 0
        for item in self.df[col]:
            if item > 0:
                save += item
        self.income[col] = save
    self.ic_t += save
```

get_month_save(self)

Esta es la ultima función encargada de analizar el Data Frame.

Nuevamente se emplea un diccionario con la misma estructura que en las dos funciones anteriores {mes: valor} que se referencia mediante la instancia `self.save`, sin embargo en este caso se emplean los diccionarios generados en las funciones anteriores para ahorrar código y tiempo de ejecución. Finalmente se imprime la información relevante relativa a los ahorros por pantalla.

```
def get_month_save(self):
    self.save = dict()
    for col in self.df.columns:
        self.save[col] = self.income[col] + self.spend[col]

    print('\nAhorro por Meses:')
    max = 0
    month = str()
    for i in self.save:
        if self.save[i] > max:
            max = self.save[i]
            month = i
        print(f'\t{i}: {self.save[i]}')
    print(f'Mes con mayor ahorro: {month}({max})')
```


gen_graph(self)

Esta es la función encargada de generar el grafico.

La primera parte del código pide al usuario que seleccione que grafico desea mostrar mediante códigos numéricos y comprueba que el valor introducido sea valido, mientras no lo sea, se generan errores del tipo ValueError recogidos y tratados en una excepción.

A continuación se crea la figura del grafico en la que se establecen los títulos de los ejes, el titulo del grafico se selecciona mediante una función que simula un selectCase del lenguaje C.

Tras esto se añade un trazo que representa la información seleccionada, esta información nuevamente es seleccionada mediante una función que simula un selectCase.

```
def gen_graph(self):
    while True:
        try:
            show = int(input(
                '\nIntroduzca la grafica a mostrar(1, 2, 3):\n'
                '\n\t1) Evolución de los ingresos.\n'
                '\n\t2) Evolución de los gastos.\n'
                '\n\t3) Evolucion de los ahorros.\n'
                '\n>>>'
            ))
            if 1 > show or show > 3:
                raise ValueError
            break
        except ValueError:
            print('Caracter no valido.')

    fig = go.Figure()
    fig.update_layout(
        title=self.select_tittle(show),
        xaxis_title='Mes',
        yaxis_title='Valor'
    )
    fig.add_trace(self.select_case(show))
    fig.show()
```

selec_case(self, case) & select_title(self, case)

Ambas funciones simulan la estructura Select, Case empleada en C, en pocas palabras, haciendo uso de los diccionarios, se programan como valores de este el 'Case' y se emplean como key del diccionario los valores a evaluar, tras ello sencillamente se realiza un return en el que devuelve la pareja key, value donde key será el case y value el valor de asignado a ese caso.

```
def selec_case(self, case):
    cases = {
        1: go.Scatter(
            x=list(self.income.keys()),
            y=list(self.income.values()),
            mode='lines',
            name='Ingreso'
        ),
        2: go.Scatter(
            x=list(self.spend.keys()),
            y=list(self.spend.values()),
            mode='lines',
            name='Gasto'
        ),
        3: go.Scatter(
            x=list(self.save.keys()),
            y=list(self.save.values()),
            mode='lines',
            name='Ahorro'
        )
    }
    return cases[case]

def select_tittle(self, case):
    title = {
        1: 'Evolución de los ingresos',
        2: 'Evolución de los gastos.',
        3: 'Evolucion de los ahorros.'
    }
    return title[case]
```

Respondiendo a las cuestiones planteadas

Como ya se ha explicado en los apartados anteriores, se ha preparado el código para que imprima por pantalla la información que responderá a las preguntas planteadas por lo que se muestra la siguiente imagen:

resultados obtenidos con archivo csv con errores provocados:

```
Faltan valores en Junio, se tomara como nulo.
Falta Septiembre, se tomará como nulo.

Gasto por Meses:
    Enero: -18162
    Febrero: -19496.0
    Marzo: -29690
    Abril: -34133.0
    Mayo: -17200
    Junio: 0
    Julio: -18992.0
    Agosto: -29013.0
    Septiembre: 0
    Octubre: -7093.0
    Noviembre: -23995.0
    Diciembre: -25087.0
Mes con mayor gasto: Abril(-34133.0)

Ahorro por Meses:
    Enero: 11523
    Febrero: 1238.0
    Marzo: -7969
    Abril: -19355.0
    Mayo: 10304
    Junio: 0
    Julio: 7369.0
    Agosto: -9434.0
    Septiembre: 0
    Octubre: 7095.0
    Noviembre: 1342.0
    Diciembre: -2270.0
Mes con mayor ahorro: Enero(11523)

La media de gasto Mensual ha sido: -18571.75
El gasto total a lo largo del año ha sido: -222861.0
El ingreso total a lo largo del año ha sido: 222704.0
El balance al finalizar el año es: 44565.0
```

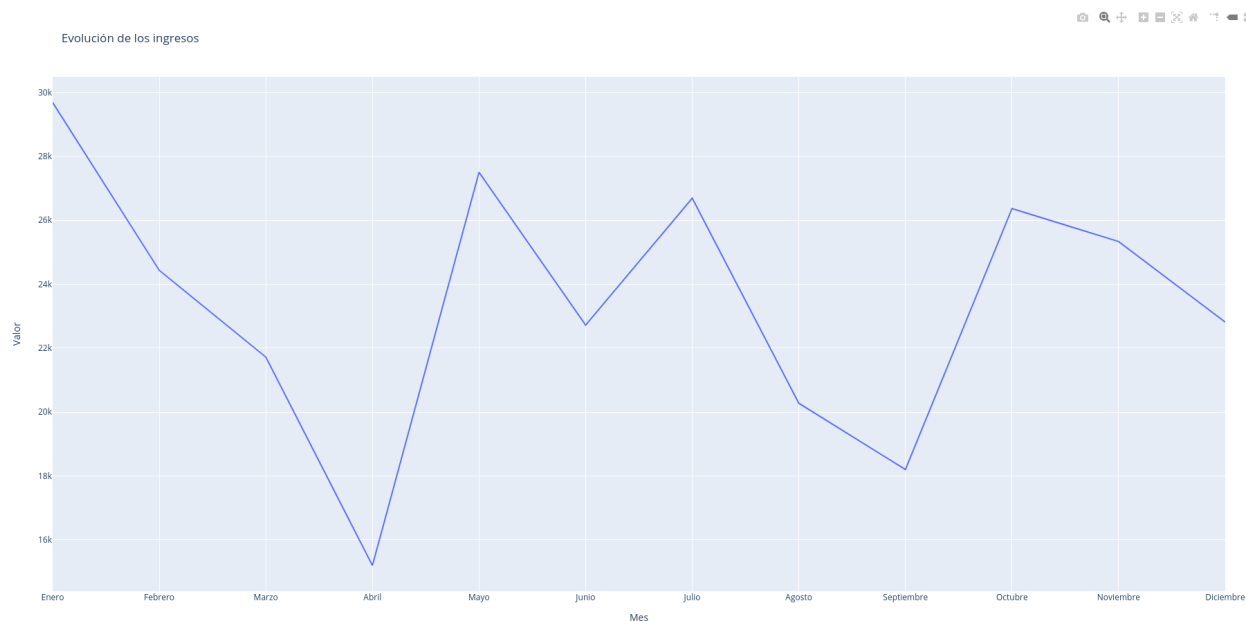
Resultados obtenidos con archivo csv original:

```
Gasto por Meses:
    Enero: -18162
    Febrero: -24398
    Marzo: -29690
    Abril: -34133
    Mayo: -17200
    Junio: -24197
    Julio: -18992
    Agosto: -29013
    Septiembre: -29151
    Octubre: -22957
    Noviembre: -24180
    Diciembre: -25861
Mes con mayor gasto: Abril(-34133)

Ahorro por Meses:
    Enero: 11523
    Febrero: 39
    Marzo: -7969
    Abril: -18933
    Mayo: 10304
    Junio: -1477
    Julio: 7698
    Agosto: -8735
    Septiembre: -10948
    Octubre: 3412
    Noviembre: 1157
    Diciembre: -3044
Mes con mayor ahorro: Enero(11523)

La media de gasto Mensual ha sido: -24827.83
El gasto total a lo largo del año ha sido: -297934
El ingreso total a lo largo del año ha sido: 280961
El valance al finalizar el año es: 578895
```

Resultado del apartado opcional:



Pese a que la imagen se muestre aquí, se adjuntara el archivo png de mayor resolución.