ex_20220210

February 7, 2026

1 Exámen Programación en entorno de datos - 2022-02-10 - MUICD - UNED

Ejercicios elaborados con fines educativos, inspirados en los contenidos evaluados en el exámen del 10/02/2022 de Programación en Entorno de Datos de la UNED (convocatoria Feb-2022).

Este documento no es una copia ni una transcripción del examen oficial, sino una redacción propia de ejercicios conceptualmente equivalentes.

1.1 TEST

Cada pregunta correcta: +1 punto
Cada pregunta incorrectas: -0,3 puntos
Cada pregunta sin contestar: 0 puntos

Puntuación mínima del apartado para superar el exámen: 2 puntos
Puntuación máxima del apartado: 4 puntos

1.1.1 EX.20200210.T.1

Enunciado EX.20200210.T.1 Dado el siguiente código en NumPy:

```
import numpy as np
a = np.arange(12).reshape((2, 2, 3))
b = a.transpose((2, 1, 0))
```

¿Cuál será la forma (`shape`) del array `b`?

a) `(2, 3, 2)`

b) `(2, 2, 3)`

c) `(3, 2, 2)`

d) `(3, 4)`

Solución EX.20200210.T.1 La respuesta correcta es c) `(3, 2, 2)`.

[1]:
```
import numpy as np
# Genera 12 elementos y les da forma por bloques x filas x columnas
a = np.arange(12).reshape((2, 2, 3))
```

```
# Transpone a, indicando el nuevo orden de los índices
b = a.transpose((2, 1, 0))
# Muestra el tamaño y orden de las dimensiones de b como set
b.shape
```

[1]: `(3, 2, 2)`

1.1.2 EX.20200210.T.2

Enunciado EX.20200210.T.2 Considera el siguiente código en Python:

```
a = [1, 2, 3]
b = a * 2
```

¿Qué valor tomará la variable `b` tras ejecutar estas instrucciones?

a) `[1, 2, 3, 1, 2, 3]`

b) `[2, 4, 6]`

c) `[[1,2,3], [1,2,3]]`

d) `12`

Solución EX.20200210.T.2 La respuesta correcta es a).

El comportamiento estándar de una lista de Python con el operador * y un entero es la de duplicar los elementos por el número con el que esté acompañado.

No hay que confundirlo con el comportamiento de los `nparray`'s, en el cual si se multiplica el `nparray` por un número actualiza su contenido multiplicando cada elemento individual por un número.

```
x = np.arange(3)*2
x
```

```
array([0, 2, 4])
```

[2]:
```
a = [1,2,3]
b = a * 2
b
```

[2]: `[1, 2, 3, 1, 2, 3]`

1.1.3 EX.20200210.T.3

Enunciado EX.20200210.T.3

3. Analiza el siguiente código en Python:

```
a = [1, 2, 3]
b = a
```

```
a.append(4)
c = b[1:]
```

Dado el siguiente código en NumPy:

import numpy as np a = np.arange(12).reshape((2, 2, 3)) b = a.transpose((2, 1, 0))

¿Cuál será la forma (`shape`) del array `b`?

¿Qué valor tendrá la variable `c` al finalizar la ejecución?

a) `[2,3]`

b) `[1,2,3,4]`

c) Se produce un error en la última línea.

d) `[2,3,4]`

Solución EX.20200210.T.3 La respuesta correcta es d).

[3]:
```
a = [1,2,3]
# Se guarda la referencia de b en a. Es equivalente usar ambas variables
b = a
# Se añade un elemento a la estructura referenciada desde a/b
a.append(4)
# Se recoge de la segunda posición en adelate
c = b[1:]
c
```

[3]:
```
[2, 3, 4]
```

1.1.4 EX.20200210.T.4

Enunciado EX.20200210.T.4 Cuando dentro de una función se altera el contenido de un objeto mutable sin realizar una reasignación ni declararlo como global:

a) El cambio realizado se pierde al finalizar la función.
b) Python genera un error por no haberse declarado como global.
c) El cambio permanece una vez que la función termina.
d) No se puede modificar un objeto mutable sin recurrir a una asignación.

Solución EX.20200210.T.4 La respuesta correcta es c).

Un ejemplo de objeto mutable es una lista.

[4]:
```
def modificar_lista(lista):
    # Modifica el objeto original
    lista.append(4)

mi_lista = [1, 2, 3]
```

```
modificar_lista(mi_lista)
print(mi_lista)
# Resultado: [1, 2, 3, 4] -> La modificación se CONSERVA
```

```
[1, 2, 3, 4]
```

1.2 Problemas

Cada uno de estos problemas se puntuará sobre 3 puntos. Para superar esta parte será necesario obtener al menos 2 puntos sobre 6. Se recuerda a los estudiantes que los pequeños errores en el código (como nombres exactos de funciones, orden de los parámetros y similares) no se penalizarán.

1.2.1 EX.20200210.P.1

Enunciado EX.20200210.P.1 Una tienda desea gestionar de forma sistemática la información asociada a los productos que recibe de sus proveedores. Para ello, se plantea desarrollar un sistema que permita registrar y procesar cada una de las entregas realizadas.

Como punto de partida, se solicita a cada proveedor que, en el momento de la entrega, facilite un fichero en formato CSV con la siguiente información:

- Identificador del producto
- Número de unidades entregadas
- Fecha de caducidad del producto (común a todas las unidades incluidas en esa entrega)
- Precio por unidad

Se pide:

a) Analizar si es preciso incorporar información adicional a los ficheros proporcionados por los proveedores para poder resolver correctamente el resto de los apartados del ejercicio. En caso afirmativo, indicar qué datos serían necesarios. Describir y justificar la estructura de datos que se utilizaría para cargar en memoria la información de cada entrega, así como el procedimiento seguido para dicha carga.

b) Desarrollar el código necesario para calcular el coste total de los productos recibidos en una fecha determinada.

c) Desarrollar el código necesario para determinar, para cada mes, el proveedor al que corresponde el mayor importe a pagar.

d) Supóngase que la tienda pasa a formar parte de una gran cadena que agrupa varios establecimientos similares, y que se decide implantar este sistema de gestión de forma centralizada. Indicar qué modificaciones serían necesarias en la estructura de datos definida en el apartado a).

e) Considerando ahora el sistema centralizado para todas las tiendas, repetir los apartados b) y c) de forma que se obtenga la información tanto a nivel global como desglosada por cada

tienda. Por ejemplo, en el apartado c), se deberá identificar el proveedor al que más se paga por mes, tanto en el conjunto de la cadena como en cada tienda individual.

Solución EX.20200210.P.1

a) Indique si es necesario añadir información a cada fichero (y, en caso afirmativo, qué información) para realizar adecuadamente las operaciones que se piden en el resto de los apartados de este ejercicio. Describa, de forma justificada, la estructura que utilizaría para cargar en memoria los ficheros correspondientes a cada entrega y el proceso seguido para realizar dicha carga.

Se ve necesario añadir las siguientes columnas a la estructura de datos: - **id_proveedor**: Identificador de proveedor, puesto que es necesario poder distinguir y filtrar dependiendo del mismo. - **dia_entrega**: Día de entrega, para confirmar el día que fue entregado. - **id_tienda**: Identificador de tienda, para poder distinguir por la misma.

d) Nuestra tienda pasa a ser absorbida por una gran cadena con varias tiendas similares a la nuestra. Como les gusta nuestro sistema, plantea implantarlo de forma centralizada. ¿Qué cambios habría que realizar sobre la estructura del primer apartado?

Habría que añadir el campo `id_tienda` listado en el primer apartado, ya que habría que distinguir entre las distintas tiendas.

[5]:
```
# NO incluir en exámen

PATH_CSV_P1_1 = "20220210_p1_entrega_1.csv"
ID_PROVEEDOR_ENTREGA_1 = "ACME"
FECHA_ENTREGA_1 = "2022-01-01"
CSV_TEXT_P1_1 = """codigo,unidades,fecha_caducidad,precio
1,1,2022-01-01,1
2,2,2022-01-01,2
"""

PATH_CSV_P1_2 = "20220210_p1_entrega_2.csv"
ID_PROVEEDOR_ENTREGA_2 = "Contoso"
FECHA_ENTREGA_2 = "2022-01-01"
CSV_TEXT_P1_2 = """codigo,unidades,fecha_caducidad,precio
1,4,2022-01-01,6
2,5,2022-01-01,7
"""

def crea_fichero(path, text):
    with open(path, "w") as f:
        f.write(text)

crea_fichero(PATH_CSV_P1_1, CSV_TEXT_P1_1)
crea_fichero(PATH_CSV_P1_2, CSV_TEXT_P1_2)
```

```python
# Incluir en exámen
import pandas as pd

def _carga_csv(path):
    """
    Carga del fichero
    """
    df = pd.read_csv(path)
    # Convierte a fecha
    df['fecha_caducidad'] = pd.to_datetime(df['fecha_caducidad'])

    return df

def anade_entrega(df, path_csv, id_proveedor, fecha_entrega):
    """
    a) Indique si es necesario añadir información a cada fichero (y, en caso␣
 ↪afirmativo, qué información) para realizar adecuadamente
    las operaciones que se piden en el resto de los apartados de este ejercicio.
 ↪ Describa, de forma justificada, la estructura que
    utilizaría para cargar en memoria los ficheros correspondientes a cada␣
 ↪entrega y el proceso seguido para realizar dicha carga.
    """
    df_entrega = _carga_csv(path_csv)
    # Añade columnas
    df_entrega["id_proveedor"] = id_proveedor
    df_entrega["fecha_entrega"] = fecha_entrega
    df_entrega["fecha_entrega"] = pd.to_datetime(df_entrega["fecha_entrega"])
    # Une dfs. ignore_index=true es para no repetir índices
    return pd.concat([df, df_entrega], ignore_index=True)

# a) Describa, de forma justificada, la estructura que
# utilizaría para cargar en memoria los ficheros correspondientes a cada␣
 ↪entrega y el proceso seguido para realizar dicha carga.
df = pd.DataFrame()

df = anade_entrega(df,
                   PATH_CSV_P1_1,
                   ID_PROVEEDOR_ENTREGA_1,
                   FECHA_ENTREGA_1)
df = anade_entrega(df,
                   PATH_CSV_P1_2,
                   ID_PROVEEDOR_ENTREGA_2,
                   FECHA_ENTREGA_2)
df
```

[5]:

	codigo	unidades	fecha_caducidad	precio	id_proveedor	fecha_entrega
0	1	1	2022-01-01	1	ACME	2022-01-01

```
1       2         2       2022-01-01       2          ACME    2022-01-01
2       1         4       2022-01-01       6       Contoso    2022-01-01
3       2         5       2022-01-01       7       Contoso    2022-01-01
```

[6]:
```python
def obtiene_importe_prods_fecha(df, fecha):
    """
    b) Escribir el código necesario para averiguar el coste total de los␣
 ↪productos recibidos un determinado día.
    """
    # Filtra por fecha
    df_dia = df[df["fecha_entrega"] == fecha]
    return (df_dia["unidades"] * df_dia["precio"]).sum()

# b)
fecha_consulta = "2022-01-01"
importe_prods_fecha = obtiene_importe_prods_fecha(df, fecha_consulta)
print(f"Importe total de productos en fecha: ", importe_prods_fecha)
```

```
Importe total de productos en fecha:  64
```

[7]:
```python
def obtiene_max_acreedor_por_mes(df):
    """
    c) Escribir el código necesario para conocer, por cada mes, el proveedor al␣
 ↪que más dinero hay que pagar.
    """
    # Calcula el importe por línea
    df_aux = df.copy()
    df_aux["importe"] = df_aux["unidades"] * df_aux["precio"]

    # Extrae el mes (formato YYYY-MM)
    df_aux["mes"] = df_aux["fecha_entrega"].dt.to_period("M")

    # Suma importes por mes y proveedor
    totales = (
        df_aux
        .groupby(["mes", "id_proveedor"], as_index=False)["importe"].sum()
    )

    # Para cada mes, proveedor con mayor importe
    # Devuelve un Series con un valor para cada mes
    idx = totales.groupby("mes")["importe"].idxmax()
    #return idx
    return totales.loc[idx].sort_values("mes")

# c)
mapm = obtiene_max_acreedor_por_mes(df)
# Itera en el dataframe por filas
```

```
for _, fila in mapm.iterrows():
    print(f"Máximo acreedor del {fila['mes']}: {fila['id_proveedor']}")
```

```
Máximo acreedor del 2022-01: Contoso
```

[8]:
```
# e) Teniendo en cuenta que ahora trabajamos con el sistema centralizado para todas las tiendas,
# repetir los apartados b y c de manera que ofrezcan la información global y por cada tienda.
#

# Crea por tienda
dft = df.copy()
dft["tienda"] = 1

def obtiene_max_acreedor_por_mes_y_tienda(df):
    """
    Por ejemplo, para el apartado c, averiguar el proveedor al que más hay que pagar por mes en global y por cada tienda.
    """
    # Calcula el importe por línea
    df_aux = df.copy()
    df_aux["importe"] = df_aux["unidades"] * df_aux["precio"]

    # Extrae el mes (formato YYYY-MM)
    df_aux["mes"] = df_aux["fecha_entrega"].dt.to_period("M")

    # Suma importes por mes y proveedor
    totales = (
        df_aux
        .groupby(["mes", "tienda", "id_proveedor"], as_index=False)["importe"].sum()
    )

    # Para cada mes y tienda, proveedor con mayor importe
    # Devuelve un Series con un valor para cada mes
    idx = totales.groupby(["mes", "tienda"])["importe"].idxmax()
    #return idx
    return totales.loc[idx].sort_values("mes")

mapmyt = obtiene_max_acreedor_por_mes_y_tienda(dft)
mapmyt
```

[8]:

	mes	tienda	id_proveedor	importe
1	2022-01	1	Contoso	59

[ ]:

1.2.2 EX.20200210.P.2

Enunciado EX.20200210.P.2 En un centro educativo hay numerosos alumnos que están completando una colección de cromos de moda. Durante los recreos se producen intercambios frecuentes de cromos repetidos, pero los profesores han observado que los alumnos emplean mucho tiempo en encontrar compañeros con los que realizar cambios beneficiosos. Por este motivo, solicitan una forma de organizar y procesar la información disponible para facilitar dichos intercambios.

Desde el centro se proporciona un fichero de texto con la siguiente estructura:

- La primera línea indica el número total de cromos que componen la colección.
- A continuación, para cada alumno que participa en la colección, se incluyen los siguientes datos:
  - Una línea con el nombre del alumno.
  - Una línea con la lista de identificadores numéricos de los cromos que posee, separados por comas. Los valores están comprendidos entre 1 y el número total de cromos de la colección e incluyen posibles repeticiones.

Por ejemplo, si la colección consta de 5 cromos, la lista `1,4,3,5,1,3` indica que el alumno dispone de dos unidades del cromo 1, dos unidades del cromo 3 y una unidad de los cromos 4 y 5. En consecuencia, no posee el cromo 2 y tiene repetidos los cromos 1 y 3.

Se solicita desarrollar el código completo en Python que permita realizar las siguientes tareas:

a) Leer la información del fichero de texto y almacenarla en un `DataFrame` de Pandas. Cada fila representará a un alumno y contendrá `n + 1` columnas, siendo `n` el número total de cromos de la colección. La primera columna, denominada `Nombre`, almacenará el nombre del alumno. Las columnas restantes, numeradas de 1 a `n`, indicarán cuántas unidades tiene el alumno de cada cromo (0 si no lo posee, 1 si lo tiene sin repetir y un valor mayor que 1 si dispone de ejemplares repetidos). Se deberá prestar atención a la optimización del uso de memoria.

b) Implementar una función que, dado el nombre de un alumno, devuelva una tupla compuesta por:

- Una lista con los números de los cromos que le faltan para completar la colección.
- Una lista con los números de los cromos de los que dispone de más de una unidad.

c) Implementar una función que reciba una lista de cromos y un `DataFrame` con la estructura definida en el apartado a), y devuelva un nuevo `DataFrame` que incluya únicamente a los alumnos que tengan repetido al menos uno de los cromos indicados.

d) Implementar una función que reciba una lista de cromos y un `DataFrame` con la estructura definida en el apartado a), y devuelva un nuevo `DataFrame` que incluya únicamente a los alumnos a los que les falte al menos uno de los cromos de dicha lista.

e) A partir de las funciones anteriores, desarrollar un método que reciba el nombre de un alumno y devuelva la lista de nombres de los alumnos con los que podría realizar intercambios de cromos.

Solución EX.20200210.P.1

[9]:
```
# No exámen

PATH_FICH_P2 = "20220210_p1_entrega_1.csv"
TEXT_P2 = """6
Juan García
2,5,2,3
Ana Rodríguez
1,7,6,4
Pedro Martínez
1,5,4,5,4
"""

crea_fichero(PATH_FICH_P2, TEXT_P2)

# Exámen

import pandas as pd

def lee_fichero(path):
    """
    a) Cargar la información del fichero de texto en un DataFrame de Pandas.␣
 ↪Cada fila contendrá n+1 columnas, siendo n el número total de cromos de esa␣
 ↪colección. La primera columna se llamará Nombre y contendrá el nombre del␣
 ↪alumno que está haciendo esa colección. Las siguientes columnas, nombradas␣
 ↪de 1 a n, contendrán el número de cromos que tiene del cromo cuyo número␣
 ↪coincide con el de la columna (0 significa que no lo tiene; 1 que lo tiene,␣
 ↪pero no repetido; un número >1 que tiene cromos repetidos de ese número).␣
 ↪Procure optimizar el espacio que ocupa en memoria.
    """
    filas = []
    df = pd.DataFrame()

    with open(path, 'r') as f:
        # Lee línea 1
        linea_1 = f.readline()
        # Convierte a entero con strip para borrar \n
        total_cromos = int(linea_1.strip())
        todos_cromos = list(range(1, total_cromos + 1))

        # Inicializa estructuras para bucle
        s_nombre = None
        # Lee resto de líneas en bucle
        for i, linea in enumerate(f):
            # Comprueba si es la línea de nombre o de cromo
            if i%2 == 0:
                # Convierte a entero con strip para borrar \n
                nombre = str(linea.strip())
```

```
                s_nombre = pd.Series([nombre], index=["nombre"])
            else:
                # Procesa segunda línea con cromos
                cromos = linea.strip().split(",")
                s_cromos = (
                    pd.Series(cromos)
                    # Necesario antes de value_counts para que coincida con el tipo de todos_cromos
                    .astype(int)
                    .value_counts()
                    .reindex(todos_cromos, fill_value=0)
                    .astype("uint16")
                )
                # Unimos nombre y cromos y transponemos para que los índices sean columnas
                dfl = pd.concat([s_nombre, s_cromos]).to_frame().T
                # Añade fila a df
                filas.append(dfl)

        df = pd.concat(filas, ignore_index=True)
        # Asegura que quedan ordenadas
        df = df[["nombre"] + todos_cromos]
        # Optimiza la memoria cambiando nombre a variable categórica
        df["nombre"] = df["nombre"].astype("category")

        return df

df = lee_fichero(PATH_FICH_P2)
df
```

[9]:

	nombre	1	2	3	4	5	6
0	Juan García	0	2	1	0	1	0
1	Ana Rodríguez	1	0	0	1	0	1
2	Pedro Martínez	1	0	0	2	2	0

[10]:
```
def obtiene_faltan_y_repes_alumno(df, nombre):
    """
    b) Crear un método que reciba el nombre de un alumno y devuelva una tupla formada por:

        - Lista de los números de los cromos que le faltan para completar esa colección.
        - Lista de los números de los cromos repetidos que tiene de esa colección.
    """
    # Obtiene copia de array filtrado por alumno
    dfa = df[df["nombre"] == nombre]
```

```
    dfa = dfa.drop(columns=["nombre"])
    # Convierte única fila a Series
    fila = dfa.iloc[0]
    # Faltantes es array de nombre de columnas con valor=0
    faltan = fila[fila == 0].index.tolist()
    a = []
    # Repetidos es array de nombre de columnas con valor>0
    repetidos = fila[fila > 1].index.tolist()
    return (faltan,repetidos)

nombre_alumno = "Juan García"
a, b = obtiene_faltan_y_repes_alumno(df, nombre_alumno)
print(f"Al alumno {nombre_alumno} le falta/n {a} y tiene repetido/s {b}")
```

```
Al alumno Juan García le falta/n [1, 4, 6] y tiene repetido/s [2]
```

[11]:
```
lista_cromos_repes = [2,1]

def obtener_listado_alumnos_cromos_repes(df, lista_cromos_repes):
    """
    c) Crear un método que reciba una lista de cromos y un DataFrame con la␣
 ↪información descrita en el apartado a y devuelva un DataFrame que contenga␣
 ↪únicamente los datos de los alumnos del DataFrame de entrada que tienen␣
 ↪repetido alguno de los cromos de esa lista.
    """
    # Obtiene las columnas que aparecen en la lista
    df_cromos = df[lista_cromos_repes]

    # Devuelve una máscara booleana sobre si algún cromo de la lista está␣
 ↪repetido (>1)
    mask = (df_cromos > 1).any(axis=1)
    return df[mask]["nombre"]

lacr = obtener_listado_alumnos_cromos_repes(df,lista_cromos_repes)
print(f"Listado cromos repetidos en {sorted(set(lacr))}:")
print(lacr)
```

```
Listado cromos repetidos en ['Juan García']:
0    Juan García
Name: nombre, dtype: category
Categories (3, object): ['Ana Rodríguez', 'Juan García', 'Pedro Martínez']
```

[12]:
```
lista_cromos_faltan=[2,6]

def obtener_listado_alumnos_cromos_faltan(df,lista_cromos_faltan):
    """
```

```
    d) Crear un método que reciba una lista de cromos y un DataFrame con la␣
 ↪información descrita en el apartado a y devuelva un DataFrame que contenga␣
 ↪únicamente los datos de los alumnos del DataFrame de entrada a los que les␣
 ↪falte alguno de los cromos de la lista.
    """
    # Obtiene las columnas que vienen en la lista
    df_cromos = df[lista_cromos_faltan]

    # Devuelve una máscara booleana sobre si algún cromo de la lista está␣
 ↪repetido (>1)
    mask = (df_cromos == 0).any(axis=1)
    return df[mask]["nombre"]

lacf = obtener_listado_alumnos_cromos_faltan(df,lista_cromos_faltan)
print(f"Listado cromos faltantes en {sorted(set(lista_cromos_faltan))}:")
print(lacf)
```

```
Listado cromos faltantes en [2, 6]:
0       Juan García
1     Ana Rodríguez
2    Pedro Martínez
Name: nombre, dtype: category
Categories (3, object): ['Ana Rodríguez', 'Juan García', 'Pedro Martínez']
```

[13]:
```
"""
e) Utilizando los métodos anteriores, programe un método que reciba el nombre␣
 ↪de un alumno y devuelva la lista de los nombres de los alumnos con quienes␣
 ↪puede intercambiar cromos.
"""
def obtener_nombres_intercambio_alumno(df,nombre_obj):

    # Obtener listado de cromos completo
    faltan_alumno_obj, repes_alumno_obj = obtiene_faltan_y_repes_alumno(df,␣
 ↪nombre_obj)
    # nombres_posibles =␣
 ↪obtener_listado_alumnos_cromos_repes(df,faltan_alumno_obj)

    faltan_obj, repetidos_obj = obtiene_faltan_y_repes_alumno(df, nombre_obj)

    # Si no falta nada o no tiene repetidos, no puede intercambiar
    if len(faltan_obj) == 0 or len(repetidos_obj) == 0:
        return []

    # df sin el alumno objetivo
    df_otros = df[df["nombre"] != nombre_obj].copy()

    # Condición A: otros tienen repetido algo que a obj le falta
```

```
    mask_A = (df_otros[faltan_obj] > 1).any(axis=1)

    # Condición B: a otros les falta algo de lo que obj tiene repetido
    mask_B = (df_otros[repetidos_obj] == 0).any(axis=1)

    # Deben cumplirse ambas
    posibles = df_otros[mask_A & mask_B]["nombre"].tolist()
    return posibles

nombre_intercambio = "Juan García"
nombres_intercambio = obtener_nombres_intercambio_alumno(df, nombre_intercambio)

print(f"Al alumno {nombre_intercambio} le interesaría intercambiar con: ")
print(nombres_intercambio)
```

```
Al alumno Juan García le interesaría intercambiar con:
['Pedro Martínez']
```