Exámenes de Programación con Python

Grado en Inteligencia de los Negocios

Alfredo Sánchez Alberca
asalber@ceu.es
https://aprendeconalf.es

Tabla de contenidos

Prefacio

Colección de exámenes de la asignatura de Programación con Python del grado en Inteligencia de los Negocios.

Cada examen incluye el enunciado de los ejercicios y una solución desplegable. Las soluciones completas, junto con los ficheros de datos, están disponibles como cuadernos Jupyter en la carpeta `soluciones` del repositorio.

1 2019-03-27 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 27 de Marzo de 2019

Ejercicio 1.1.

1. Clonar con git el repositorio con la url `https://github.com/asalberceu/examen-fundamentos-programacion-2019-03-27-A.git`.
2. Crear una rama con los apellidos del alumno en mayúsculas y separados por un guión, es decir, `APELLIDO1-APELLIDO2`, y convertir esta rama en la rama activa.
3. Crear la carpeta `respuestas`, y dentro de ella el fichero `ejercicio1.3.txt` que contenga la salida que da el comando de Git para mostrar todas diferencias entre la última versión de la rama actual y la anterior.
4. Añadir todos los cambios a la zona temporal de intercambio y hacer un commit con el mensaje "Añadida respuesta ejercicio 1.3."
5. Crear dentro de la carpeta `respuestas` el fichero `ejercicio1.5.txt` que contenga la salida que da el comando de Git para mostrar todos los commits del repositorio (una línea por commit), incluyendo todas las ramas.
6. Añadir todos los cambios a la zona temporal de intercambio y hacer un commit con el mensaje "Añadida respuesta ejercicio 1.5."

Solución

```
> git clone
 ↪  https://github.com/asalberceu/examen-fundamentos-programacion-2019-03-27-A.git
> cd examen-fundamentos-programacion-2019-03-27-A
> git checkout -b SANCHEZ-ALBERCA
> mkdir respuestas
> git diff HEAD~1 > respuestas/ejercicio1.3.txt
> git add .
> git commit -m "Añadida respuesta ejercicio 1.3"
> git log --oneline --all > respuestas/ejercicio1.5.txt
> git add .
> git commit -m "Añadida respuesta ejercicio 1.5"
```

Ejercicio 1.2. Escribir un programa que realice la devolución de una cantidad dada por el usuario en monedas.

El programa debe cumplir los siguientes requisitos:

- Solo se disponen de tres tipos de monedas: 5, 2 y 1 €. Crear una lista que contenga estos tres tipos de moneda y usar la lista en la solución.
- El programa debe preguntar al usuario por una cantidad entera de euros.
- El programa debe mostrar por pantalla el mínimo número de monedas necesarias para sumar la cantidad introducida por el usuario y cuántas monedas de cada tipo se necesitan para ello. El número de monedas de cada tipo debe guardarse en otra lista.
- El programa debe guardarse dentro de la carpeta `respuestas` con el nombre `ejercicio2.py`.
- Cuando el programa esté terminado, añadir el fichero `ejercicio2.py` a la zona de intercambio temporal y hacer un commit con el mensaje "Añadida respuesta ejercicio 2".

Solución

```
coins = [5, 2, 1]
change = [0, 0, 0]
amount = int(input("Introduce una cantidad entera de euros: "))
print('Para sumar', amount, '€ se necesitan ', end='')
for i in range(len(coins)):
    while amount >= coins[i]:
        amount -= coins[i]
        change[i] += 1
print(sum(change), 'monedas:')
for i in range(len(coins)):
    print(change[i], 'monedas de ', coins[i], '€')
```

```
Introduce una cantidad entera de euros:  48
Para sumar 48 € se necesitan 11 monedas:
9 monedas de  5 €
1 monedas de  2 €
1 monedas de  1 €
```

Ver notebook de la solución en Colab

Ejercicio 1.3. Escribir un programa que simule el famoso juego del ahorcado.

El programa debe cumplir los siguientes requisitos:

- El programa debe preguntar al usuario la palabra a adivinar. A partir de la palabra introducida debe crear una lista con los caracteres de la palabra.
- Después debe ir preguntando al usuario por letras hasta un máximo de 5 fallos o hasta que no queden letras en la lista. En ambos casos el programa terminará pero mostrará el mensaje "Perdiste" si se comenten 5 fallos y el mensaje "Ganaste" si no quedan palabras en la lista.
- Cada vez que el usuario introduzca una nueva letra, si la letra está en la lista la eliminará y mostrará el mensaje "Acierto", mientras que si la letra no está en la lista mostrará el mensaje "Fallo". Si la letra está más de una vez en la lista, sólo se eliminará la primera instancia que aparezca.
- El programa debe guardarse dentro de la carpeta `respuestas` con el nombre `ejercicio3.py`.
- Cuando el programa esté terminado, añadir el fichero `ejercicio3.py` a la zona de intercambio temporal y hacer un commit con el mensaje "Añadida respuesta ejercicio 3".

Requisito adicional para un punto extra:

- Cada vez que el usuario acierte una letra debe mostrar la palabra a adivinar con las letras acertadas hasta el momento y el resto reemplazadas por asteriscos.

Solución

Solución 1

```
word = list(input('Introduce la palabra a adivinar: '))
failures = 0
while word and failures < 5:
    letter = input('Introduce una letra: ')
    if letter in word:
        print('Acierto')
        word.remove(letter)
    else:
        print('Fallo')
        failures += 1
if failures == 5:
    print('Perdiste')
else:
    print('Ganaste')
```

```
Introduce la palabra a adivinar:  python
Introduce una letra:  p
Acierto
Introduce una letra:  n
```

```
Acierto
Introduce una letra:  a
Fallo
Introduce una letra:  h
Acierto
Introduce una letra:  t
Acierto
Introduce una letra:  y
Acierto
Introduce una letra:  o
Acierto
Ganaste
```

Solución 2

```python
word = input('Introduce la palabra a adivinar: ')
word = list(word)
solution = list('*' * len(word))
failures = 0
while any(word) and failures < 5:
    letter = input('Introduce una letra: ')
    if letter in word:
        print('Acierto')
        i = word.index(letter)
        word[i] = False
        solution[i] = letter
    else:
        print('Fallo')
        failures += 1
    print(solution)
if failures == 5:
    print('Perdiste')
else:
    print('Ganaste')
```

```
Introduce la palabra a adivinar:  anaconda
Introduce una letra:  n
Acierto
['*', 'n', '*', '*', '*', '*', '*', '*']
Introduce una letra:  a
Acierto
['a', 'n', '*', '*', '*', '*', '*', '*']
Introduce una letra:  c
```

```
Acierto
['a', 'n', '*', 'c', '*', '*', '*', '*']
Introduce una letra:  h
Fallo
['a', 'n', '*', 'c', '*', '*', '*', '*']
Introduce una letra:  o
Acierto
['a', 'n', '*', 'c', 'o', '*', '*', '*']
Introduce una letra:  o
Fallo
['a', 'n', '*', 'c', 'o', '*', '*', '*']
Introduce una letra:  n
Acierto
['a', 'n', '*', 'c', 'o', 'n', '*', '*']
Introduce una letra:  d
Acierto
['a', 'n', '*', 'c', 'o', 'n', 'd', '*']
Introduce una letra:  a
Acierto
['a', 'n', 'a', 'c', 'o', 'n', 'd', '*']
Introduce una letra:  a
Acierto
['a', 'n', 'a', 'c', 'o', 'n', 'd', 'a']
Ganaste
```

Ver notebook de la solución en Colab

2 2019-05-27 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 27 de Mayo de 2019

Ejercicio 2.1.

1. Clonar con git el repositorio con la url `https://github.com/asalberceu/examen-fundamentos-computadores-2019-05-27-parcial1.git`.
2. Crear una rama con los apellidos del alumno en mayúsculas y separados por un guión, es decir, `<APELLIDO1-APELLIDO2>`, y convertir esta rama en la rama activa.
3. Crear el fichero `ejercicio1.3.txt` que contenga la salida del comando de Git que muestra para cada línea del fichero `ejercicio2.py` la información sobre el commit en que se realizó el último cambio en esa línea.
4. Crear un fichero `ejercicio1.4.txt` que contenga la salida del comando que muestra las diferencias entre el primer y el segundo commits del repositorio.
5. Crear un fichero `ejercicio1.5.txt` que contenga la salida del comando de Git que muestra el estado actual del repositorio.
6. Añadir todos los cambios a la zona temporal de intercambio y hacer un commit con el mensaje "Añadidas respuestas."

Solución

```
> git clone
 ↪  https://github.com/asalberceu/examen-fundamentos-computadores-2019-05-27-parcial1.g
> cd examen-fundamentos-computadores-2019-05-27-parcial1.git
> git checkout -b SANCHEZ-ALBERCA
> git annotate ejercicio2.py > ejercicio1.3.txt
> git diff afce5 b4a76 > ejercicio1.4.txt
> git status > ejercicio1.5.txt
> git add .
> git commit -m "Añadida respuestas."
```

Ejercicio 2.2. Escribir un programa que pida al usuario un número entero n y muestre por pantalla un triángulo el de más abajo.

```
    *
   ***
  *****
 *******
*********
```

donde n es el número de filas del triángulo.

Solución

Solución 1

```
# Preguntamos el número de filas del triángulo y lo convertimos en u
 ↪  n entero
n = int(input('Introduce el número de filas del triángulo: '))
# Recorremos con un bucle cada fila del triángulo
for i in range(n):
    # Añadimos con un bucle el número de espacios de cada línea para
     ↪  que los asteriscos aparezcan centrados
    for j in range(n-i-1):
        print(' ', end='')
    # Añadimos con otro bucle el número de asteriscos de cada línea
    for j in range(2*i+1):
        print('*', end='')
    print()
```

```
Introduce el número de filas del triángulo:  10
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

Solución 2

```python
# Preguntamos el número de filas del triángulo y lo convertimos en u
 ↪  n entero
n = int(input('Introduce el número de filas del triángulo: '))
# Recorremos con un bucle cada fila del triángulo
for i in range(n):
    # Añadimos el número de espacios de cada línea para que los aste
     ↪  riscos aparezcan centrados
    print(' '*(n-i-1), end='')
    # Añadimos el número de asteriscos de cada línea.
    print('*'*(2*i+1))
```

```
Introduce el número de filas del triángulo:  10
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
```

Ver notebook de la solución en Colab

Ejercicio 2.3. Escribir un programa para ver los beneficios de una empresa en un periodo de años.

El programa debe cumplir los siguientes requisitos:

- El programa tiene que preguntar al usuario por un año inicial y otro final, y después preguntar por los ingresos y los gastos de cada año desde el año inicial hasta el año final.
- Con los datos introducidos se deben crear dos listas, una con los ingresos y otra con los gastos, de manera que los ingresos y los gastos de cada año aparezcan en la misma posición de las listas.
- El programa debe crear otra lista con el beneficio de cada año (ingresos menos gastos) y mostrarla por pantalla.
- El programa debe crear otra lista booleana que indique para cada año si ha habido beneficios o no y mostrarla por pantalla.
- Finalmente el programa debe mostrar por pantalla la lista de los años con beneficios la lista de los años con pérdidas.

Solución

```
# Pregunta por los años inicial y final
inicio = int(input('Introduce el año inicial: '))
fin = int(input('Introduce el año final: '))
# Definimos las listas de años, ingresos y gastos vacías
años = []
ingresos = []
gastos = []
# Preguntamos por los ingresos y gastos de cada año y los añadimos a
 ↪  las listas correspondientes
for i in range(inicio, fin+1):
    años.append(i)
    ingresos.append(float(input('Ingresos del año ' + str(i) + ': '
 ↪  )))
    gastos.append(float(input('Gastos del año ' + str(i) + ': ')))
# Creamos la lista de beneficios vacía
beneficios = []
# Recorremos las listas de ingresos y gastos, calculamos la diferenc
 ↪  ia para los elementos que ocupan la misma posición y lo añadimos
 ↪  a la lista de beneficios.
for i in range(len(ingresos)):
    beneficios.append(ingresos[i] - gastos[i])
print(beneficios)
# Creamos la lista vacía para ver qué años ha habido beneficios
hay_beneficio = []
# Recorremos la lista de beneficios y añadimos a la lista hay_benefi
 ↪  cio True si el beneficio es positivo y False en caso contrario
for i in range(len(beneficios)):
    hay_beneficio.append(beneficios[i] > 0)
print(hay_beneficio)
# Creamos dos listas vacías para los años con beneficios y los años
 ↪  con pérdidas
años_beneficios = []
años_perdidas = []
# Recorremos la lista hay_beneficio y dependiendo de si el valor es
 ↪  True o False añadimos el año correspondiente a la lista de años
 ↪  con beneficio o a la de años con pérdidas.
for i in range(len(hay_beneficio)):
    if hay_beneficio[i]:
        años_beneficios.append(años[i])
    else:
        años_perdidas.append(años[i])
print('Años con beneficios: ', años_beneficios)
print('Años con pérdidas: ', años_perdidas)
```

```
Introduce el año inicial:  2015
Introduce el año final:  2018
Ingresos del año 2015:  100000
Gastos del año 2015:  80000
Ingresos del año 2016:  120000
Gastos del año 2016:  125000
Ingresos del año 2017:  140000
Gastos del año 2017:  100000
Ingresos del año 2018:  150000
Gastos del año 2018:  125000
[20000.0, -5000.0, 40000.0, 25000.0]
[True, False, True, True]
Años con beneficios:  [2015, 2017, 2018]
Años con pérdidas:  [2016]
```

Ver notebook de la solución en Colab

Ejercicio 2.4. Definir una función que reciba un número entero entre 0 y 999, y devuelva una cadena con la cantidad introducida en palabras. Por ejemplo, si se introduce 647 debe devolver la cadena "seiscientos cuarenta y siete".

La función debe cumplir los siguientes requisitos:

- El único parámetro de entrada de la función es un número entero entre 0 y 999.
- Deben usarse diccionarios para emparejar cada dígito con la palabra correspondiente para las unidades, decenas y centenas.
- Debe devolver una cadena con la cantidad introducida en palabras.

Solución

Solución 1

```python
def enuncia_cantidad(n):
    """
    Función que recibe una cantidad n y devuelve una cadena con la m
    isma cantidad expresada en palabras.
    Parámetros:
        - n es un número entero entre 1 y 999.
    Devuelve:
        Una cadena con la cantidad n expresada en palabras.
    """
    # Diccionarios con las traducciones de número a palabra para las
     unidades, decenas y centenas.
    unidades = {0:'', 1:'uno', 2:'dos', 3:'tres', 4:'cuatro', 5:'cin
    co', 6:'seis', 7:'siete', 8:'ocho', 9:'nueve'}
    decenas = {0:'', 1:'diez', 2:'veinte', 3:'treinta', 4:'cuarenta'
    , 5:'cincuenta', 6:'sesenta', 7:'setenta', 8:'ochenta', 9:'noven
    ta'}
    centenas = {0:'', 1:'ciento', 2:'doscientos', 3:'trescientos',
    4:'cuatrocientos', 5:'quinientos', 6:'seiscientos', 7:'setecient
    os', 8:'ochocientos', 9:'novencientos'}
    # Obtenemos las centenas mediante división entera por 100
    cen = n // 100 # Otra alternativa es cen = int(str(n)[0])
    # Obtenemos las decenas mediante división entera por 10 y luego
     tomando el resto de la división entera por 10
    dec = (n // 10) % 10 # Otra alternativa es dec = int(str(n)[1])
    # Obtenemos las unidades mediante el resto de la visión entera p
    or 10.
    uni = n % 10 # Otra alternativa es cen = int(str(n)[2])
    # Definimos el nexo entre las decenas y las unidades
    if not uni:
        # Si no hay unidades no hay nexo.
        nexo = ''
    elif not dec:
        # Si hay unidades pero no decenas no hay nexo.
        nexo = ''
    else:
        # Si hay unidades y decenas el nexo es 'y'
        nexo = ' y '

    return centenas[cen] + decenas[dec] + nexo + unidades[uni]

# Varias llamadas a la función de prueba
print(647, ':', enuncia_cantidad(647))
print(30, ':', enuncia_cantidad(30))
print(6, ':', enuncia_cantidad(6))
print(100, ':', enuncia_cantidad(100))
```

```
647 : seiscientoscuarenta y siete
30 : treinta
6 : seis
100 : ciento
```

Solución 2

```python
def enuncia_cantidad(n):
    """
    Función que recibe una cantidad n y devuelve una cadena con la m
    isma cantidad expresada en palabras.
    Parámetros:
        - n es un número entero entre 1 y 999.
    Devuelve:
        Una cadena con la cantidad n expresada en palabras.
    """
    # Diccionarios con las traducciones de número a palabra para las
     unidades, decenas y centenas.
    unidades = {0:'', 1:'uno', 2:'dos', 3:'tres', 4:'cuatro', 5:'cin
    co', 6:'seis', 7:'siete', 8:'ocho', 9:'nueve'}
    decenas = {0:'', 1:'diez', 2:'veinte', 3:'treinta', 4:'cuarenta'
    , 5:'cincuenta', 6:'sesenta', 7:'setenta', 8:'ochenta', 9:'noven
    ta'}
    centenas = {0:'', 1:'ciento', 2:'doscientos', 3:'trescientos',
    4:'cuatrocientos', 5:'quinientos', 6:'seiscientos', 7:'setecient
    os', 8:'ochocientos', 9:'novencientos'}
    # Obtenemos las centenas mediante división entera por 100
    cen = n // 100  # Otra alternativa es cen = int(str(n)[0])
    # Obtenemos las decenas mediante división entera por 10 y luego
     tomando el resto de la división entera por 10
    dec = (n // 10) % 10  # Otra alternativa es dec = int(str(n)[1])
    # Obtenemos las unidades mediante el resto de la visión entera p
    or 10.
    uni = n % 10  # Otra alternativa es cen = int(str(n)[2])
    # Definimos el nexo entre las decenas y las unidades
    if not uni:
        # Si no hay unidades no hay nexo.
        nexo = ''
    elif not dec:
        # Si hay unidades pero no decenas no hay nexo.
        nexo = ''
    else:
        # Si hay unidades y decenas el nexo es 'y'
        nexo = ' y '

    return centenas[cen] + decenas[dec] + nexo + unidades[uni]

# Varias llamadas a la función de prueba
print(647, ':', enuncia_cantidad(647))
print(30, ':', enuncia_cantidad(30))
print(6, ':', enuncia_cantidad(6))
print(100, ':', enuncia_cantidad(100))
```

```
647 : seiscientoscuarenta y siete
30 : treinta
6 : seis
100 : ciento
```

Ver notebook de la solución en Colab

Ejercicio 2.5. El fichero `cotizacion.csv` contiene las cotizaciones de las empresas del IBEX35 con las siguientes columnas: `nombre` (nombre de la empresa), `Final` (precio de la acción al cierre de bolsa), `Máximo` (precio máximo de la acción durante la jornada), `Mínimo` (precio mínimo de la acción durante la jornada), `volumen` (Volumen al cierre de bolsa), `Efectivo` (capitalización al cierre en miles de euros).

1. Construir una función que abra un fichero con el formato anterior y devuelva un diccionario con los datos del fichero por columnas.

   La función debe cumplir los siguientes requisitos:

   - La función recibirá como único parámetro la ruta del fichero.
   - Debe realizarse un preprocesado de los datos que reemplace la coma por el punto como separador de decimales.
   - Debe realizarse el control de errores mediante excepciones para el caso de que el fichero no exista en la ruta indicada.
   - Debe devolver un diccionario con tantos elementos como columnas tenga el fichero, donde la clave de cada par sea el nombre de la columna y el valor la lista de datos de la columna.

2. Construir una función que reciba el diccionario devuelto por la función anterior y cree un fichero en formato csv con el mínimo, el máximo y la media de dada columna.

   La función debe cumplir los siguientes requisitos:

   - La función recibirá como parámetros el diccionario con los datos de cotización y la ruta del fichero a crear.
   - El fichero generado tendrá las mismas columnas que el fichero `cotizacion.csv` con los mismos nombres de columnas, y tres líneas correspondientes al mínimo, máximo y media de los datos de cada columna. En la columna `nombre` en lugar del nombre de la empresa debe aparecer la medida calculada en esa línea (mínimo, máximo o media). Los datos deben estar separados por punto y coma.

Solución

```python
def limpiar(cifra):
    """
    Función que elimina los puntos de separación de miles y cambia l
    as comas de separación de decimales por puntos.
    Parámetros:
        - cifra: Es una cadena con una cifra
    Devuelve:
        Un real con la cifra de la cadena después de eliminar el sep
    arador de miles y cambiar el separador de decimales por punto.
    """
    cifra = cifra.replace('.', '')
    cifra = cifra.replace(',','.')
    return float(cifra)

def preprocesado(ruta):
    """
    Función que preprocesa los datos contenidos en un fichero con fo
    rmato csv y devuelve un diccionario con los nombres de las colum
    nas como claves y las listas de valores asociados a ellas.
    Parámetros:
        - ruta: Es una cadena con la ruta del fichero.
    Devuelve:
        Un diccionario con pares formados por los nombres de las col
    umnas y las listas de valores en las columnas.
    """
    try:
        # Abrimos el fichero en modo lectura
        f = open(ruta, 'r')
    except FileNotFoundError:
        print('El fichero no existe.')
        return
    # Leemos el fichero por líneas en una lista
    lines = f.readlines()
    # Cerramos el fichero
    f.close()
    # Leemos las claves del primer elemento de la lista y eliminamos
     el cambio de línea que aparece al final
    claves = lines[0]
    # Eliminamos el cambio de línea que aparece al final y dividimos
     la cadena por el punto y coma
    claves = claves[:-1].split(';')
    # Creamos el diccionario
    cotizaciones = {}
    # Inicializamos el diccionario con listas vacías
    for i in claves:
        cotizaciones[i] = []
    # Recorremos la lista línea a línea
    for linea in lines[1:]:
        # Eliminamos el cambio de línea que aparece al final y divid
         imos la cadena por el punto y coma
        linea = linea[:-1].split(';')
        cotizaciones[claves[0]].append(linea[0])
        # Añadimos cada dato a la lista correspondiente del dicciona
         rio
        for i in range(1, len(cotizaciones)):
            cotizaciones[claves[i]].append(limpiar(linea[i]))
```

Ver notebook de la solución en Colab

Ejercicio 2.6. Definir una función que reciba una lista de facturas, un NIF y un mes, y devuelva un diccionario con el número de facturas emitidas a ese NIF en el mes indicado y el total facturado en ese mes.

La función debe cumplir los siguientes requisitos:

- Los parámetros de entrada serán una lista de facturas, una cadena con un NIF y otra cadena con el mes.
- Cada factura se representarán mediante un diccionario con las claves `nif` (NIF del cliente), `mes` (mes de emisión de la factura), `cantidad` (cantidad facturada sin IVA), `iva` (porcentaje de IVA a aplicar).
- Se debe crear una lista con el total de cada factura (una vez aplicado el IVA) para el NIF y el mes indicados utilizando programación funcional o comprensión de listas.
- La función debe devolver un diccionario con el número de facturas y el total facturado al NIF en el mes indicado.

Solución

```python
def facturacion(facturas, nif, mes):
    """
    Función que recibe una lista de facturas y devuelve el número de facturas y el total facturado correspondiente a un nif y mes indicados.
    Parámetros:
        - facturas: Es una lista de facturas, donde cada factura es un diccionario con las claves nif, mes, cantidad e iva.
        - nif: Es una cadena con un nif.
        - mes: Es una cadena con el nombre de un mes.
    Devuelve:
        Un diccionario con el número de facturas y el total facturado (incluído el iva) al nif indicado en el mes indicado.
    """
    # Filtramos la lista para quedarnos con la lista de las cantidades de cada diccionario que tenga el nif y el mes indicados
    filtro = [factura['cantidad'] * (1 + factura['iva'] / 100) for factura in facturas if (factura['nif'] == nif and factura['mes'] == mes)]
    # Devolvemos un diccionario con la longitud de la lista (número de facturas) y la suma de sus valores (total facturado)
    return {'Num facturas': len(filtro), 'Total facturado': sum(filtro)}


# Lista de facturas de prueba
facturas = [{'nif': 'B12345678', 'mes': 'marzo', 'cantidad':1000, 'iva': 10},
            {'nif': 'A98765432', 'mes': 'julio', 'cantidad':500, 'iva': 21},
            {'nif': 'B12345678', 'mes': 'marzo', 'cantidad':2000, 'iva': 21},
            {'nif': 'C02040506', 'mes': 'enero', 'cantidad':200, 'iva': 4},
            {'nif': 'A98765432', 'mes': 'julio', 'cantidad':1500, 'iva': 10},
            {'nif': 'B12345678', 'mes': 'abril', 'cantidad':600, 'iva': 10},
            {'nif': 'B12345678', 'mes': 'marzo', 'cantidad':100, 'iva': 4}]

# Llamadas a la función de prueba
print(facturacion(facturas, 'B12345678', 'marzo'))
print(facturacion(facturas, 'B12345678', 'abril'))
print(facturacion(facturas, 'B12345678', 'agosto'))
```

```
{'Num facturas': 3, 'Total facturado': 3624.0}
{'Num facturas': 1, 'Total facturado': 660.0}
{'Num facturas': 0, 'Total facturado': 0}
```

Ver notebook de la solución en Colab

3 2019-06-18 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 18 de Junio de 2019

Ejercicio 3.1.

1. Clonar con git el repositorio con la url `https://github.com/asalberceu/examen-fundamentos-computadores-2019-06-18-parcial1.git`.
2. Crear una rama con los apellidos del alumno en mayúsculas y separados por un guión, es decir, `<APELLIDO1-APELLIDO2>`, y convertir esta rama en la rama activa.
3. Crear el fichero `ejercicio1.3.txt` que contenga la salida del comando de Git que muestra el historial de commits del repositorio, mostrando un commit por línea.
4. Crear un fichero `ejercicio1.4.txt` que contenga la salida del comando que muestra las diferencias entre el primer y el tercer commits del repositorio.
5. Crear un fichero `ejercicio1.5.txt` que contenga la salida del comando de Git que muestra los repositorios remotos configurados junto con sus urls.
6. Añadir todos los cambios a la zona temporal de intercambio y hacer un commit con el mensaje "Añadidas respuestas ejercicio 1."

Solución

```
> git clone
 ↪  https://github.com/asalberceu/examen-fundamentos-computadores-2019-06-18-parcial1.g
> cd examen-fundamentos-computadores-2019-06-18-parcial1.git
> git checkout -b SANCHEZ-ALBERCA
> git log --oneline > ejercicio1.3.txt
> git diff f9ed835 a485ced > ejercicio1.4.txt
> git remote -v > ejercicio1.5.txt
> git add .
> git commit -m "Añadida respuestas."
```

Ejercicio 3.2. A lo largo de un curso se realizan dos exámenes parciales. Para aprobar el curso la nota media debe ser mayor o igual que 5 siempre y cuando en ambos parciales se tenga al menos un 4. Escribir un programa que pregunte al usuario la nota de los dos

parciales y muestre por pantalla si el alumno ha aprobado el curso o si no, y en caso de no haber aprobado, qué parcial tiene que repetir por tener menos de 4 en él.

Solución

```
# Pedimos al usuario que introduzca las notas de las asignaturas y l
 ↪  as convertimos en números reales
nota1 = float(input('Introduce la nota de la primera parte: '))
nota2 = float(input('Introduce la nota de la segunda parte: '))
# Comprobamos si ambas notas son mayores o iguales que 4 y si la med
 ↪  ia es mayor o igual que 5
if nota1 >= 4 and nota2 >= 4 and ((nota1 + nota2) / 2) >= 5:  # Las
 ↪  dos notas son mayores que 4 y la media mayor o igual que 5
    print('Has aprobado')
else: # La nota media es menor que 5 o alguna nota es menor que 4 (s
 ↪  uspenso)
    print('Has suspendido')
    # Comprobamos si la primera nota es menor que 4
    if nota1 < 4:
        print('Tienes que repetir el primer parcial')
    # Comprobamos si la segunda nota es menor que 4
    if nota2 < 4:
        print('Tienes que repetir el segundo parcial')
```

```
Introduce la nota de la primera parte:  3
Introduce la nota de la segunda parte:  8
Has suspendido
Tienes que repetir el primer parcial
```

Ver notebook de la solución en Colab

Ejercicio 3.3. Un n-grama es una secuencia de n caracteres consecutivos de una cadena. Por ejemplo, los 3-gramas de la cadena `'Python'` son `'Pyt'`, `'yth'`, `'tho'` y `'hon'`. Escribir un programa que pregunte por una cadena y un número entero positivo n y muestre por pantalla todos los n-gramas de la cadena.

Solución

```
# Pedimos al usuario que introduzca una palabra y un número entero p
 ↪  ositivo
palabra = input('Introduce una palabra: ')
n = int(input('Introduce un número entero positivo: '))
# Recorremos las posiciones de la palabra desde el inicio hasta la l
 ↪  ongitud de la palabra menos n + 1.
# No tiene sentido seguir ya que las subcadenas después de esa posic
 ↪  ión tendrían longitud menor que n.
for i in (range(len(palabra) - n + 1)):
    # Mostramos el n-grama que empieza en la posición i
    print(palabra[i:i+n])
```

```
Introduce una palabra:  Python
Introduce un número entero positivo:  3
Pyt
yth
tho
hon
```

Ver notebook de la solución en Colab

Ejercicio 3.4. Escribir un programa que elimine de una lista dada todos los elementos repetidos y muestre por pantalla los elementos de la lista sin repeticiones.

Solución

La solución de este ejercicio no está disponible en el repositorio de origen.
Carpeta de soluciones del examen

Ejercicio 3.5. Definir funciones para codificar y decodificar mensajes en código morse.

1. Definir una función para codificar una palabra en código morse. Debe cumplir los siguientes requisitos:

   - Debe usarse el diccionario que se da.
   - El único parámetro de entrada de la función es una cadena con una palabra.
   - Debe devolver una cadena con el código morse correspondiente a la palabra, separando los bloques de código correspondientes a cada letra por punto y coma `;`.

2. Definir una función para decodificar una palabra en código morse. Debe cumplir los siguientes requisitos:

   - A partir del diccionario que se da se debe crear el diccionario invertido, es decir, un diccionario cuyas claves son los códigos morse y sus valores las letras correspondientes. Se valorará especialmente el uso de comprensión de diccionarios.
   - El único parámetro de entrada de la función es una cadena de código morse, donde los bloques de código correspondientes a cada letra van separados por puntos y coma `;`.
   - Debe devolver una cadena con la palabra decodificada.

3. Definir una función para codificar un mensaje en código morse. Debe cumplir los siguientes requisitos:

   - Debe usarse la función anterior para codificar palabras.
   - El único parámetro de entrada de la función es una cadena con un mensaje (palabras separadas con espacios).
   - Debe devolver una cadena con las palabras del mensaje codificadas y separadas por espacios.
   - Se valorará especialmente el uso de programación funcional.

4. Definir una función para decodificar un mensaje en código morse. Debe cumplir los siguientes requisitos:

   - Debe usarse la función anterior para decodificar palabras.
   - El único parámetro de entrada de la función es una cadena con un mensaje en código morse (letras separadas por punto y coma, y palabras separadas con espacios).
   - Debe devolver una cadena con las palabras del mensaje decodificadas y separadas por espacios.
   - Se valorará especialmente el uso de programación funcional.

Solución

```python
# Diccionario con la codificación morse
morse = {'A': '.-',     'B': '-...',   'C': '-.-.',
         'D': '-..',    'E': '.',      'F': '..-.',
         'G': '--.',    'H': '....',   'I': '..',
         'J': '.---',   'K': '-.-',    'L': '.-..',
         'M': '--',     'N': '-.',     'O': '---',
         'P': '.--.',   'Q': '--.-',   'R': '.-.',
         'S': '...',    'T': '-',      'U': '..-',
         'V': '...-',   'W': '.--',    'X': '-..-',
         'Y': '-.--',   'Z': '--..'}


def codificar_palabra(palabra):
    """
    Función que codifica una palabra en morse.
    Parámetros:
        palabra: Una cadena con una palabra.
    Devuelve:
        El código morse correspondiente a la palabra con los bloques
 ↪  de código de cada letra separados por punto y coma.
    """
    # Creamos una cadena vacía para ir añadiendo las letras codifica
     ↪  das
    palabra_codificada = ''
    # Recorremos los caracteres de la palabra
    for i in palabra:
        # Accedemos al diccionario con la clave correspondiente al c
         ↪  aracter y añadimos el valor (el código morse) a la palab
         ↪  ra codificada
        palabra_codificada += morse[i.upper()] + ';' # Añadimos un p
 ↪  unto y coma entre caracter y caracter.
    return palabra_codificada


def decodificar_palabra(palabra):
    """
    Función que decodifica una palabra en morse.
    Parámetros:
        palabra: Es una cadena con bloques de código morse separados
 ↪  por el caracter ;.
    Devuelve:
        La palabra decodificada.
    """

    # A partir del diccionario morse construimos otro diccionario in
     ↪  vertido, es decir, cuyas claves sean los códigos morse y cuy
     ↪  os valores sean las letras asociadas.
    # Recorremos los pares del diccionario morse e invertimos las cl
     ↪  aves y los valores y devolvemos un diccionario mediante comp
     ↪  rensión de diccionarios.
    morse_invertido = {value:key for key, value in morse.items()}
    # Creamos una cadena vacía para ir añadiendo las letras decodifi
     ↪  cadas.
    palabra_decodificada = ''
    # Dividimos el código morse por el punto y coma y recorremos la
     ↪  lista de códigos resultante.
```

```
.--.;-.--;-;....;---;-.;
PYTHON
....;---;.-..;.-; .--.;-.--;-;....;---;-.;
HOLA PYTHON
```

Ver notebook de la solución en Colab

Ejercicio 3.6. La url https://datos.madrid.es/egob/catalogo/300117-0-arrendamiento-programas.csv apunta a un fichero en formato csv con datos de los arrendamientos de viviendas de la Empresa Municipal de la Vivienda del Ayuntamiento de Madrid.

1. Construir una función que abra un fichero con el formato anterior y devuelva una lista cuyos elementos son a su vez las listas que contienen los datos de cada línea del fichero menos la primera línea. Debe cumplir los siguientes requisitos:

   - La función recibirá como único parámetro la url del fichero.
   - Debe leer el fichero por líneas y para cada línea debe dividir la línea por el separador de campos (punto y coma) y guardar los datos en una lista.
   - Debe devolver la lista con las listas de datos obtenidas a partir de cada línea.

2. Construir una función que reciba una lista de listas como la que devuelve la función anterior y devuelva otra lista con los nombres de los distritos contenidos en la lista. Debe cumplir los siguientes requisitos:

   - La función recibirá como único parámetro una lista de listas con las viviendas arrendadas por distrito.
   - Debe recorrer la lista de listas y para cada lista debe extraer el nombre del distrito y añadirlo a una lista con los distritos.
   - Debe devolver la lista de distritos.

3. Construir una función que reciba una lista de listas como la que devuelve la primera función y una lista de nombres de distritos y devuelva la lista con las listas correspondientes a los distritos indicados. Debe satisfacer los siguientes requisitos:

   - La función recibirá como parámetros una lista de listas con las viviendas arrendadas por distrito y otra lista con nombres de distritos.
   - Debe recorrer la lista de viviendas arrendadas y añadir a otra lista nueva las líneas correspondientes a los distritos indicados en la segunda lista.
   - Debe devolver la nueva lista con las listas correspondientes a los distritos indicados.

4. Construir una función que reciba una lista como la que devuelve la primera función y devuelva un diccionario cuyas claves sean los nombres de distrito y cuyos valores

sean el total de viviendas arrendadas en el distrito. Debe cumplir los siguientes requisitos:

- La función recibirá como único parámetro la lista con las viviendas arrendadas por distrito.
- Debe recorrer la lista de listas y para cada lista extraer el nombre del distrito y el total de viviendas arrendadas en el distrito y añadir el par a un diccionario.
- Debe devolver un diccionario con un par para cada lista de la lista, cuya clave sea el nombre del distrito y cuyo valor sea el número total de viviendas arrendadas en ese distrito.

Solución

Solución 1 Usando comprensión de listas y diccionarios.

```
def leer_fichero(url):
    """
    Función que abre un fichero de texto desde una url y devuelve un
 ↪  a lista de listas con los datos del fichero.
    Parámetros:
        url: URL del fichero de texto en formato csv donde cada regi
 ↪  stro aparece en una línea y los campos están separados por punto
 ↪  y coma `;`.
    Devuelve:
        Una lista cuyos elementos son a su vez listas que contienen
 ↪  los datos de cada línea del fichero menos la primera línea.
    """
    # Carga del módulo necesario para acceder a un fichero de intern
     ↪  et.
    from urllib import request
    from urllib.error import URLError
    # Abrimos el fichero con la url dada.
    try:
        f = request.urlopen(url)
    except URLError: # Si se produce un error al acceder a la url in
     ↪  formamos devolvemos una cadena informando de ello.
        return('¡La url ' + url + ' no existe!')
    # Leemos los datos del fichero en una cadena.
    datos = f.read()
    # Decodificamos los datos con la codificación latin1.
    datos = datos.decode('latin1')
    # Creamos una lista con las líneas del fichero dividiendo la cad
     ↪  ena por el cambio de línea.
    datos = datos.split('\n')
    # Eliminamos la primera línea que no contiene información intere
     ↪  sante.
    datos.pop(0)
    # Recorremos las líneas de la lista y para cada línea creamos un
     ↪  a lista con los datos separando por el punto y coma.
    # Devolvemos la lista de listas mediante comprensión de listas.
    datos_viviendas = [i.split(';') for i in datos]
    return datos_viviendas


def distritos(datos_viviendas):
    """
    Función recibe una lista de listas con los datos de la viviendas
 ↪  arrendadas y devuelve los distritos.
    Parámetros:
        datos_viviendas: Es una lista de listas como la que devuelve
 ↪  la función leer_fichero donde cada lista contienen los datos de
 ↪  viviendas arrendadas de un distrito.
    Devuelve:
        Una lista con los distritos correspondientes a cada lista.
    """
    # Recorremos la lista de listas y para cada lista accedemos el p
     ↪  rimer elemento que contiene el nombre del distrito.
    # Devolvemos la lista de distritos mediante comprensión de lista
     ↪  s.
    distritos = [i[0] for i in datos_viviendas[1:]]
    return distritos
```

```
['ARGANZUELA', 'BARAJAS', 'CARABANCHEL', 'CENTRO', 'CHAMBERI',
 ↪  'CIUDAD LINEAL', 'FUENCARRAL-EL PARDO', 'HORTALEZA', 'LATINA',
 ↪  'MONCLOA', 'MORATALAZ', 'PUENTE DE VALLECAS', 'RETIRO',
 ↪  'SALAMANCA', 'SAN BLAS', 'TETUAN', 'USERA', 'VALLECAS VILLA',
 ↪  'VICALVARO', 'VILLAVERDE', 'TOTAL', '', '']
{'CHAMBERI': '3', 'HORTALEZA': '12'}
```

Solución 2 Sin comprensión de listas y diccionarios.

```
def leer_fichero(url):
    """
    Función que abre un fichero de texto desde una url y devuelve un
 ↪  a lista de listas con los datos del fichero.
    Parámetros:
        url: URL del fichero de texto en formato csv donde cada regi
 ↪  stro aparece en una línea y los campos están separados por punto
 ↪  y coma `;`.
    Devuelve:
        Una lista cuyos elementos son a su vez listas que contienen
 ↪  los datos de cada línea del fichero menos la primera línea.
    """
    # Carga del módulo necesario para acceder a un fichero de intern
     ↪  et.
    from urllib import request
    from urllib.error import URLError
    # Abrimos el fichero con la url dada.
    try:
        f = request.urlopen(url)
    except URLError: # Si se produce un error al acceder a la url in
     ↪  formamos devolvemos una cadena informando de ello.
        return('¡La url ' + url + ' no existe!')
    # Leemos los datos del fichero en una cadena.
    datos = f.read()
    # Decodificamos los datos con la codificación latin1.
    datos = datos.decode('latin1')
    # Creamos una lista con las líneas del fichero dividiendo la cad
     ↪  ena por el cambio de línea.
    datos = datos.split('\n')
    # Eliminamos la primera línea que no contiene información intere
     ↪  sante.
    datos.pop(0)
    # Creamos una lista vacía para ir añadiendo los datos de cada lí
     ↪  nea.
    datos_viviendas = []
    # Recorremos las listas de la lista
    for i in datos:
        # Para cada línea creamos una lista con los datos separando
         ↪  por el punto y coma y la añadimos a la lista principal.
        datos_viviendas.append(i.split(';'))
    return datos_viviendas


def distritos(datos_viviendas):
    """
    Función recibe una lista de listas con los datos de la viviendas
 ↪  arrendadas y devuelve los distritos.
    Parámetros:
        datos_viviendas: Es una lista de listas como la que devuelve
 ↪  la función leer_fichero donde cada lista contienen los datos de
 ↪  viviendas arrendadas de un distrito.
    Devuelve:
        Una lista con los distritos correspondientes a cada lista.
    """
    # Creamos una lista vacía para ir añadiendo los distritos.
    distritos = []
```

```
['ARGANZUELA', 'BARAJAS', 'CARABANCHEL', 'CENTRO', 'CHAMBERI',
 ↪  'CIUDAD LINEAL', 'FUENCARRAL-EL PARDO', 'HORTALEZA', 'LATINA',
 ↪  'MONCLOA', 'MORATALAZ', 'PUENTE DE VALLECAS', 'RETIRO',
 ↪  'SALAMANCA', 'SAN BLAS', 'TETUAN', 'USERA', 'VALLECAS VILLA',
 ↪  'VICALVARO', 'VILLAVERDE', 'TOTAL', '', '']
{'CHAMBERI': '3', 'HORTALEZA': '12'}
```

Ver notebook de la solución en Colab

4 2020-05-27 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 27 de Mayo de 2020

Ejercicio 4.1. Escribir un programa al que al introducirle la altura de una línea sea capaz de dibujarla en diagonal con asteriscos. Por ejemplo, si introducimos `altura = 5` dibujaría lo siguiente:

```
    *
   *
  *
 *
*
```

Solución

Solución 1

```
altura = int(input('Altura de la línea: '))
for i in range(altura, 0, -1):
    print(' ' * (i-1) + '*')
```

```
*
   *
  *
 *
*
```

Solución 2

```
altura = int(input("Altura de la línea: "))
for i in range(altura, 0, -1):
    for j in range(i - 1):
        print(" ", end="")
    print("*")
```

```
*
   *
  *
 *
*
```

Ver notebook de la solución en Colab

Ejercicio 4.2. Escribir una función que cuente las palabras que hay en una frase y las devuelva dentro de un diccionario. También tiene que devolver una lista con las palabras que aparecen más de una vez. Por ejemplo si se le pasa la frase: `La caracola está enterrada al lado de otra caracola de color` la función debe devolver el diccionario y la lista siguientes:

```
{'La': 1, 'caracola': 2, 'está': 1, 'enterrada': 1, 'al': 1, 'lado': 1,
 ↪  'de': 2, 'otra': 1, 'color': 1}
['caracola', 'de']
```

Solución

```python
def contar_palabras(frase):
    ''' Función que recibe una frase y devuelve la frecuencia de las palabras en la frase y las palabras que se repiten.

    Parámetros:
        - frase: Es una cadena de caracteres.
    Devuelve: Una tupla con un diccionario con las palabras y su frecuencia, y una lista con las palabras que se repiten.
    '''
    # Dividimos la frase por el espacio en blanco y guardamos las palabras en una lista.
    palabras = frase.strip().split()
    # Creamos un diccionario vacío para guardar las frecuencias de las palabras.
    frecuencias = {}
    # Recorremos la lista de palabras y vamos contando sus frecuencias.
    for palabra in palabras:
        # Si la palabra no está todavía en le diccionario, la añadimos con frecuencia 0.
        if palabra not in frecuencias:
            frecuencias[palabra] = 0
        # Incrementamos la frecuencia de la palabra en el diccionario.
        frecuencias[palabra] += 1
    # Creamos una lista vacía para guardar las palabras repetidas.
    repetidas=[]
    # Recorremos el diccionario de frecuencias
    for clave, valor in frecuencias.items():
        # Si la frecuencia de una palabra es mayor que 1 la añadimos a la lista.
        if valor > 1: repetidas.append(clave)
    # Devolvemos el diccionario de frecuencias y la lista de palabras repetidas.
    return frecuencias, repetidas

# Ejemplo
freq, rep = contar_palabras('La caracola está enterrada al lado de otra caracola de color')
print('Frecuencia de palabras:', freq)
print('Palabras repetidas: ', rep)
```

```
Frecuencia de palabras: {'La': 1, 'caracola': 2, 'está': 1,
 ↪  'enterrada': 1, 'al': 1, 'lado': 1, 'de': 2, 'otra': 1, 'color':
 ↪  1}
Palabras repetidas:  ['caracola', 'de']
```

Solución 2

```python
def contar_palabras(frase):
    ''' Función que recibe una frase y devuelve la frecuencia de las
     ↪  palabras en la frase y las palabras que se repiten.

    Parámetros:
        - frase: Es una cadena de caracteres.
    Devuelve: Una tupla con un diccionario con las palabras y su fre
 ↪  cuencia, y una lista con las palabras que se repiten.
    '''
    # Dividimos la frase por el espacio en blanco y guardamos las pa
     ↪  labras en una lista.
    palabras = frase.strip().split()
    # Recorremos la lista de palabras, contamos sus frecuencias y la
     ↪  s añadimos a un diccionario.
    frecuencias = {palabra:palabras.count(palabra) for palabra in
 ↪  palabras}
    # Recorremos el diccionario de frecuencias y generamos la lista
     ↪  con las palabras con frecuencia mayor que 1.
    repetidas = [palabra for palabra, frecuencia in
 ↪  frecuencias.items() if frecuencia > 1]
    # Devolvemos el diccionario de frecuencias y la lista de palabra
     ↪  s repetidas.
    return frecuencias, repetidas

# Ejemplo
freq, rep = contar_palabras('La caracola está enterrada al lado de o
 ↪  tra caracola de color')
print('Frecuencia de palabras:', freq)
print('Palabras repetidas: ', rep)
```

```
Frecuencia de palabras: {'La': 1, 'caracola': 2, 'está': 1,
 ↪  'enterrada': 1, 'al': 1, 'lado': 1, 'de': 2, 'otra': 1, 'color':
 ↪  1}
Palabras repetidas:  ['caracola', 'de']
```

Ver notebook de la solución en Colab

Ejercicio 4.3. El fichero horas-trabajo.csv contiene el número de horas mensuales trabajadas por los empleados de una empresa durante el primer cuatrimestre. Crear un programa que realice las siguientes operaciones sin utilizar la librería Pandas:

1. Leer el fichero de internet `https://aprendeconalf.es/docencia/python/examenes/inteligencia-negocios/soluciones/examen-2020-05-27/horas-trabajo.csv` y crear una lista con las líneas del fichero.
2. Mostrar por pantalla las horas totales del primer operario.
3. Crear un diccionario de diccionarios tal que las claves del diccionario principal serán los identificadores de los operarios y sus valores serán, a su vez, otros diccionarios cuyas claves serán los meses y sus valores las horas trabajadas en esos meses para cada operario. Es decir, un diccionario como el siguiente:

```
{'OP1': {'Enero': '180', 'Febrero': '160', 'Marzo': '140', 'Abril': '180
 ↪  '},
'OP2': {'Enero': '120', 'Febrero': '140', 'Marzo': '', 'Abril': '100'},
 ↪  ... }
```

4. Crear una función que reciba la base de datos de las horas trabajadas (puede utilizarse el diccionario del apartado anterior u otra estructura de datos), el identificador de un operario y el precio de la hora, y devuelva una tupla con el número totales de horas trabajadas y el salario de ese operario.

Solución

Solución 1

```
# Apartado 1
from urllib import request
from urllib.error import URLError
# Leemos el fichero desde la url.
try:
    f = request.urlopen('http://aprendeconalf.es/python/examenes/sol
 ↪  uciones/examen-2020-05-27/horas-trabajo.csv')
    # El fichero no existe
except URLError:
    print('¡La url no existe!')
else:
    # Guardamos cada línea en una lista
    lineas = f.read().decode('utf-8').splitlines()
    print("Lista de líneas del fichero")
    print(lineas)
```

```
Lista de líneas del fichero
```

```
['Id;Departamento;Enero;Febrero;Marzo;Abril',
 ↪  'OP1;Proveedores;180;160;140;180', 'OP2;Ventas;120;140;;100',
 ↪  'OP3;Ventas;80;90;80;80', 'OP5;IT;180;170;180;180',
 ↪  'OP6;Marketing;100;;100;', 'OP7;Ventas;160;160;160;160',
 ↪  'OP8;Proveedores;100;80;110;80', 'OP9;IT;80;80;80;80',
 ↪  'OP10;Ventas;180;160;180;180']
```

```
# Apartado 2
# Mostrar por pantalla las horas totales del primer operario.
# Dividimo la segunda línea de la lista por el separador ; y guardam
 ↪  os cada campo en una lista.
op1 = lineas[1].split(';')
# Recorremos la lista desde la posición 2 hasta el final, convertimo
 ↪  s las horas en enteros y creamos una lista con las horas.
horas_op1 = [int(i) for i in op1[2:]]
# Sumamos las horas de la lista.
print('Horas totales del primer operario:', sum(horas_op1))
```

```
Horas totales del primer operario: 660
```

```
# Apartado 3
# Crear un diccionario de diccionarios tal que las claves del diccio
 ↪  nario principal serán los identificadores de los operarios y sus
 ↪  valores serán, a su vez, otros diccionarios cuyas claves serán l
 ↪  os meses y sus valores las horas trabajadas en esos meses para c
 ↪  ada operario.
# Extraemos los nombres de columnas de la primera fila
columnas = lineas[0].split(';')
# Creamos el diccionario para guardar las horas de cada operario
operarios = {}
# Recorremos las líneas del fichero desde la primera hasta el final
for linea in lineas[1:]:
  # Creamos un diccionario para guardar las horas mensuales de cada
   ↪  operario
  operario = {}
  # Dividimos las línea en campos utilizando la coma como separador
  campos = linea.split(';')
  # Recorremos la lista de campos desde el tercer campo hasta el fin
   ↪  al (los dos primeros campos no contienen horas)
  for i in range(2, len(campos)):
    # Añadimos al diccionario las horas de cada mes
    operario[columnas[i]] = campos[i]
  # Añadimos el diccionario con las horas de un operario al dicciona
   ↪  rio de operarios
  operarios[campos[0]] = operario
print('Diccionario de diccionaros con las horas mensuales de cada op
 ↪  erario')
print(operarios)
```

```
Diccionario de diccionaros con las horas mensuales de cada operario
{'OP1': {'Enero': '180', 'Febrero': '160', 'Marzo': '140', 'Abril':
 ↪  '180'}, 'OP2': {'Enero': '120', 'Febrero': '140', 'Marzo': '',
 ↪  'Abril': '100'}, 'OP3': {'Enero': '80', 'Febrero': '90',
 ↪  'Marzo': '80', 'Abril': '80'}, 'OP5': {'Enero': '180',
 ↪  'Febrero': '170', 'Marzo': '180', 'Abril': '180'}, 'OP6':
 ↪  {'Enero': '100', 'Febrero': '', 'Marzo': '100', 'Abril': ''},
 ↪  'OP7': {'Enero': '160', 'Febrero': '160', 'Marzo': '160',
 ↪  'Abril': '160'}, 'OP8': {'Enero': '100', 'Febrero': '80',
 ↪  'Marzo': '110', 'Abril': '80'}, 'OP9': {'Enero': '80',
 ↪  'Febrero': '80', 'Marzo': '80', 'Abril': '80'}, 'OP10':
 ↪  {'Enero': '180', 'Febrero': '160', 'Marzo': '180', 'Abril':
 ↪  '180'}}
```

```python
# Apartado 4
# Crear una función que reciba la base de datos de las horas trabaja
 ↪ das (puede utilizarse el diccionario del apartado anterior u otr
 ↪ a estructura de datos), el identificador de un operario y el pre
 ↪ cio de la hora, y devuelva una tupla con el número totales de ho
 ↪ ras trabajadas y el salario de ese operario.
def salario(horas, operario, precio):
    ''' Función que recibe un diccionario de diccionarios con las ho
     ↪ ras mensuales trabajadas por los operarios de la empresa y d
     ↪ evuelve el salario de un operario dado.

    Parámetrso:
        - horas: Es un diccionari de diccionarios con las horas mens
 ↪ uales de cada operario.
        - operario: Es una cadena con el identificador del operario.
        - precio: Es un número real con el precio de la hora.
    Devuelve: El salario del opeario dado.
    '''
    # Creamos una lista con las horas correspondientes al operario.
     ↪ Las horas se convierten a enteros salvo si no hay horas.
    horas = [int(i) for i in horas[operario].values() if i != '']
    # Sumamos el total de horas
    total_horas = sum(horas)
    # Calculamos el salario multiplicando el total de horas por el p
     ↪ recio.
    salario = total_horas * precio
    return total_horas, salario

# Ejemplo operario 2
horas, salario = salario(operarios, 'OP2', 10)
print('Horas trabajadas del segundo operario:', horas)
print('Salario del segundo operario:', salario)
```

```
Horas trabajadas del segundo operario: 360
Salario del segundo operario: 3600
```

Ver notebook de la solución en Colab

Ejercicio 4.4. El fichero horas-trabajo.csv contiene el número de horas mensuales trabajadas por los empleados de una empresa durante el primer cuatrimestre. Crear un programa que realice las siguientes operaciones utilizando la librería Pandas:

1. Crear un DataFrame leyendo el fichero desde internet con la url `http://aprendec`

onalf.es/python/examenes/soluciones/examen-2020-05-27/horas-trabajo.csv. Obsérvese que el separador de campos es el punto y coma.

2. Mostrar por pantalla una serie con el número total de horas trabajadas para cada mes.
3. Mostrar por pantalla una serie con el número de operarios de cada departamento.
4. Mostrar por pantalla el número de empleados que han trabajado todos los meses, es decir, que tienen un número de horas todos los meses del cuatrimestre.
5. Convertir el DataFrame a formato largo, de manera que todas las horas aparezcan en la misma columna.
6. Mostrar por pantalla una serie con el número medio de horas trabajadas en cada departamento.
7. Mostrar por pantalla una serie con el total de horas trabajadas de cada operario.
8. Mostrar por pantalla una serie con los salarios de todos los operarios ordenados de mayor a menor.

Solución

Solución 1

```
import pandas as pd

# Apartado 1
# Crear un DataFrame leyendo el fichero desde internet.
operarios = pd.read_csv('https://aprendeconalf.es/docencia/python/ex
 ↪  amenes/inteligencia-negocios/soluciones/examen-2020-05-27/horas-
 ↪  trabajo.csv', sep = ';')
operarios
```

```
     Id Departamento  Enero  Febrero  Marzo  Abril
0   OP1  Proveedores    180    160.0  140.0  180.0
1   OP2       Ventas    120    140.0    NaN  100.0
2   OP3       Ventas     80     90.0   80.0   80.0
3   OP5           IT    180    170.0  180.0  180.0
4   OP6    Marketing    100      NaN  100.0    NaN
5   OP7       Ventas    160    160.0  160.0  160.0
6   OP8  Proveedores    100     80.0  110.0   80.0
7   OP9           IT     80     80.0   80.0   80.0
8  OP10       Ventas    180    160.0  180.0  180.0
```

```
# Apartado 2
# Mostrar por pantalla una serie con el número total de horas trabaj
 ↪  adas para cada mes.
print('Número de horas trabajadas por meses')
print(operarios[['Enero', 'Febrero', 'Marzo']].sum())
```

```
Número de horas trabajadas por meses
Enero      1180.0
Febrero    1040.0
Marzo      1030.0
dtype: float64
```

```
# Apartado 3
# Mostrar por pantalla una serie con el número de operarios de cada
 ↪  departamento.
print('Número de operarios por departamentos')
print(operarios.Departamento.value_counts())
```

```
Número de operarios por departamentos
Ventas         4
IT             2
Proveedores    2
Marketing      1
Name: Departamento, dtype: int64
```

```
# Apartado 4
# Mostrar por pantalla el número de empleados que han trabajado todo
 ↪  s los meses, es decir, que tienen un número de horas todos los m
 ↪  eses del cuatrimestre.
print('Número operarios que han trabajado todos los meses: ',
 ↪  operarios.dropna().shape[0])
```

```
Número operarios que han trabajado todos los meses:  7
```

```
# Apartado 5
# Convertir el DataFrame a formato largo, de manera que todas las ho
 ↪  ras aparezcan en la misma columna.
operarios_largo = operarios.melt(id_vars=['Id', 'Departamento'],
 ↪  var_name='Mes', value_name='Horas')
print("DataFrame en formato largo")
operarios_largo
```

```
DataFrame en formato largo
      Id Departamento      Mes  Horas
0    OP1  Proveedores    Enero  180.0
1    OP2       Ventas    Enero  120.0
2    OP3       Ventas    Enero   80.0
3    OP5           IT    Enero  180.0
4    OP6    Marketing    Enero  100.0
5    OP7       Ventas    Enero  160.0
6    OP8  Proveedores    Enero  100.0
7    OP9           IT    Enero   80.0
8   OP10       Ventas    Enero  180.0
9    OP1  Proveedores  Febrero  160.0
10   OP2       Ventas  Febrero  140.0
11   OP3       Ventas  Febrero   90.0
12   OP5           IT  Febrero  170.0
13   OP6    Marketing  Febrero    NaN
14   OP7       Ventas  Febrero  160.0
15   OP8  Proveedores  Febrero   80.0
16   OP9           IT  Febrero   80.0
17  OP10       Ventas  Febrero  160.0
18   OP1  Proveedores    Marzo  140.0
19   OP2       Ventas    Marzo    NaN
20   OP3       Ventas    Marzo   80.0
21   OP5           IT    Marzo  180.0
22   OP6    Marketing    Marzo  100.0
23   OP7       Ventas    Marzo  160.0
24   OP8  Proveedores    Marzo  110.0
25   OP9           IT    Marzo   80.0
26  OP10       Ventas    Marzo  180.0
27   OP1  Proveedores    Abril  180.0
28   OP2       Ventas    Abril  100.0
29   OP3       Ventas    Abril   80.0
30   OP5           IT    Abril  180.0
31   OP6    Marketing    Abril    NaN
32   OP7       Ventas    Abril  160.0
33   OP8  Proveedores    Abril   80.0
34   OP9           IT    Abril   80.0
35  OP10       Ventas    Abril  180.0
```

```
# Apartado 6
# Mostrar por pantalla una serie con el número medio de horas trabaj
 ↪  adas en cada departamento.
print("Media de horas mensuales trabajadas por departamentos")
print(operarios_largo.groupby('Departamento').Horas.mean())
```

```
Media de horas mensuales trabajadas por departamentos
Departamento
IT             128.750000
Marketing      100.000000
Proveedores    128.750000
Ventas         135.333333
Name: Horas, dtype: float64
```

```
# Apartado 7
# Mostrar por pantalla una serie con el total de horas trabajadas de
 ↪  cada operario.
print("Total de horas trabajadas de cada operario")
print(operarios_largo.groupby('Id').Horas.sum())
```

```
Total de horas trabajadas de cada operario
Id
OP1     660.0
OP10    700.0
OP2     360.0
OP3     330.0
OP5     710.0
OP6     200.0
OP7     640.0
OP8     370.0
OP9     320.0
Name: Horas, dtype: float64
```

```
# Apartado 8
# Mostrar por pantalla una serie con los salarios de todos los opera
 ↪  rios ordenados de mayor a menor.
print("Salarios de los operarios")
print((operarios_largo.groupby('Id'
 ↪  ).Horas.sum()*10).sort_values(ascending = False))
```

```
Salarios de los operarios
```

```
Id
OP5     7100.0
OP10    7000.0
OP1     6600.0
OP7     6400.0
OP8     3700.0
OP2     3600.0
OP3     3300.0
OP9     3200.0
OP6     2000.0
Name: Horas, dtype: float64
```

Ver notebook de la solución en Colab

5 2020-06-19 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 19 de Junio de 2020

Ejercicio 5.1. Escribe un programa en python que permita guardar las notas de un alumno conseguidas en un cuatrimestre. Guarda la información en un diccionario cuyas claves sean las asignaturas y los valores las notas de cada asignatura. El programa pedirá la asignatura y la nota para esa asignatura. Si se recibe un número negativo en la nota, el programa termina y muestra las asignaturas suspensas.

Ejemplo

```
Introduce una asignatura: matemáticas
Introduce la nota: 4
Introduce una asignatura: economía
Introduce la nota: 8
Introduce una asignatura: programación
Introduce la nota: 10
Introduce una asignatura: ninguna
Introduce la nota: -2
Las asignaturas suspensas son:
matemáticas
```

Solución

La solución de este ejercicio no está disponible en el repositorio de origen.
Carpeta de soluciones del examen

Ejercicio 5.2. Escribir una función que tome una lista de números enteros desordenados y devuelva dos listas ordenadas. La primera con los números pares y la segunda con los números impares.

Solución

```
def pares_impares(lista):
    '''Función que recibe una lista de números enteros y devuelve do
     ↪  s listas ordenadas con los números impares y pares de la lis
     ↪  ta dada.

    Parámetros:
        - lista: Es una lista de números enteros.
    Devuelve: Dos listas ordenadas con los números impares y pares d
 ↪  e la lista dada.
    '''

    # Definimos dos listas vacías para guardar los números pares e i
     ↪  mpares.
    pares = []
    impares = []
    # Bucle para recorrer la lista de números.
    for i in lista:
        # Condicional para ver si el número es par o impar.
        if i%2 == 0:
            # Si el resto de la división entera por 2 es 0, el númer
             ↪  o es par y se añade a la lista de pares.
            pares.append(i)
        else:
            # Si el resto de la división entera por 2 no es 0, el nú
             ↪  mero es impar y se añade a la lista de impares.
            impares.append(i)
    # Ordenamos las listas.
    pares.sort()
    impares.sort()
    # Devolvemos las listas de pares e impares.
    return pares, impares

# Ejemplo
lista = [3, 8, 2, 7, 5, 4, 6, 1, 9]
pares, impares = pares_impares(lista)
print('Pares: ', pares)
print('Impares: ', impares)
```

```
Pares:  [2, 4, 6, 8]
Impares:  [1, 3, 5, 7, 9]
```

Solución 2

```python
def pares_impares(lista):
    '''Función que recibe una lista de números enteros y devuelve do
     ↪  s listas ordenadas con los números impares y pares de la lis
     ↪  ta dada.

    Parámetros:
        - lista: Es una lista de números enteros.
    Devuelve: Dos listas ordenadas con los números impares y pares d
 ↪  e la lista dada.
    '''

    # Comprensión de listas para obtener la lista de números pares.
    pares = [n for n in lista if n % 2 == 0]
    # Comprensión de listas para obtener la lista de números impares
     ↪  .
    impares = [n for n in lista if n % 2 != 0]
    # Ordenamos las listas.
    pares.sort()
    impares.sort()
    # Devolvemos las listas de pares e impares.
    return pares, impares

# Ejemplo
lista = [3, 8, 2, 7, 5, 4, 6, 1, 9]
pares, impares = pares_impares(lista)
print('Pares: ', pares)
print('Impares: ', impares)
```

```
Pares:  [2, 4, 6, 8]
Impares:  [1, 3, 5, 7, 9]
```

Ver notebook de la solución en Colab

Ejercicio 5.3. Escribir un programa para gestionar las citas de una consulta médica. La base de datos de citas debe estar en un fichero de nombre `citas.csv`. Cada cita contendrá los campos `dni`, `mes`, `dia`, `hora` y `especialidad`. No es necesario que la primera fila del csv contenga los nombres de los campos. El programa debe incluir las siguientes funciones:

1. Una función que permita generar el fichero y añadir una cita a la base de datos.
2. Una función que reciba un dni y devuelva una lista con las citas de ese paciente.
3. Una función para eliminar las citas anteriores a una fecha dada.

Solución

Solución 1

```
# Ruta del fichero de citas.
fichero = 'citas.csv'

def citas(fichero=fichero):
    ''' Función que muestra por pantalla el contenido del fichero de
     ↪  citas.

    Parámetros:
        - fichero: Es una cadena con la ruta del fichero de citas.
    '''

    try:
        # Intentamos abrir el fichero de citas en modo lectura.
        f = open(fichero, mode='r')
    except FileNotFoundError:
        # Si no existe el fichero, controlamos la excepción mostrand
         ↪  o por pantalla un mensaje informando de que el fichero n
         ↪  o existe.
        print('El fichero no existe.')
    else:
        # Si el fichero existe y se puede abrir mostramos por pantal
         ↪  la una lista con sus líneas (cada línea corresponde a un
         ↪  a cita).
        print(f.readlines())
        # Cerramos el fichero.
        f.close()
    return

# Apartado 1

def añadir_cita(dni, mes, dia, hora, especialidad, fichero =
 ↪  fichero):
    ''' Función que añade una cita al fichero de citas.

    Parámetros:
        - dni: Es una cadena con el dni del paciente.
        - mes: Es una cadena con el número de mes.
        - dia: Es una cadena con el día.
        - hora: Es una cadena con la hora en formato hh:mm.
        - especialidad: Es una cadena con la especialidad de la cita
 ↪  .
        - fichero: Es una cadena con la ruta del fichero de citas.
    '''

    try:
        # Intentamos abrir el fichero de citas en modo añadir.
        f = open(fichero, mode = 'a')
    except FileNotFoundError:
        # Si no existe el fichero, controlamos la excepción mostrand
         ↪  o por pantalla un mensaje informando de que el fichero n
         ↪  o existe.
        print('El fichero no existe.')
    else:
        # Si el fichero existe y se puede abrir añadimos al final de
         ↪  l fichero una línea con la concatenación del dni, el mes
         ↪  , el día, la hora y la especialidad separados por comas.
```

```
['01234567,02,05,10:30,Cardiología\n',
 '12345678,04,18,12:00,Ginecología\n',
 '01234567,06,20,16:30,Cardiología\n',
 '12345678,03,16,9:30,Dermatología\n']
```

```python
# Apartado 2

def buscar_citas(dni, fichero = fichero):
    ''' Función que busca las citas de un paciente en el fichero de citas.

    Parámetros:
        - dni: Es una cadena con el dni del paciente.
        - fichero: Es una cadena con la ruta del fichero de citas.
    Devuelve: Una lista con las citas del paciente con el dni dado.
    '''
    try:
        # Intentamos abrir el fichero de citas en modo lectura.
        f = open(fichero, mode = 'r')
    except FileNotFoundError:
        # Si no existe el fichero, controlamos la excepción mostrando por pantalla un mensaje informando de que el fichero no existe.
        print('El fichero no existe.')
        return
    else:
        # Si el fichero existe y se puede abrir, leemos todas las líneas y las guardamos en una lista. Cada línea corresponde a una cita.
        citas = f.readlines()
        # Cerramos el fichero.
        f.close()
        # Filtramos la lista de citas para quedarnos con las del dni dado y devolvemos la lista filtrada. Para ello dividimos cada línea usando como separador de campos la coma y accedemos a la primera posición de la lista generada para obtener el dni de la cita.
        return [cita for cita in citas if cita.split(',')[0] == dni]

# Ejemplo
print(buscar_citas('12345678'))
```

```
['12345678,04,18,12:00,Ginecología\n',
 ↪  '12345678,03,16,9:30,Dermatología\n']
```

```
# Apartado 3

def eliminar_citas(mes, dia, fichero = fichero):
    ''' Función que elimina del fichero de citas las citas anteriore
     ↪  s a una fecha dada.

    Parámetros:
        - mes: Es una cadena con el número de mes.
        - dia: Es una cadena con el día.
        - fichero: Es una cadena con la ruta del fichero de citas.
    '''
    try:
        # Intentamos abrir el fichero de citas en modo lectura.
        f = open(fichero, mode = 'r')
    except FileNotFoundError:
        # Si no existe el fichero, controlamos la excepción mostrand
         ↪  o por pantalla un mensaje informando de que el fichero n
         ↪  o existe.
        print('El fichero no existe.')
    else:
        # Si el fichero existe y se puede abrir, leemos todas las lí
         ↪  neas y las guardamos en una lista. Cada línea correspond
         ↪  e a una cita.
        citas = f.readlines()
        # Cerramos el fichero.
        f.close()
        # Abrimos de nuevo el fichero en modo escritura (se elimina
         ↪  sus contenido).
        f = open(fichero, mode='w')
        # Bucle para recorrer las citas
        for cita in citas:
            # Dividimos la línea de cada cita usando como separador
             ↪  de campos la coma y concatenamos el segundo y tercer
             ↪  elementos de la lista resultantes que corresponden a
             ↪  l mes y el día.
            fecha = ''.join(cita.split(',')[1:3])
            # Condicional para ver si la fecha de la cita es postero
             ↪  r a la fecha indicada.
            if fecha >= mes + dia:
                # Si la cita tiene una fecha posterior a la fecha in
                 ↪  dicada la escribimos en una nueva línea del fich
                 ↪  ero.
                f.write(cita)
        # Cerramos el fichero.
        f.close()
    return
```



```
# Ejemplo
eliminar_citas('04', '01')
citas()
```

```
['12345678,04,18,12:00,Ginecología\n',
 ↪  '01234567,06,20,16:30,Cardiología\n']
```

Ver notebook de la solución en Colab

Ejercicio 5.4. El fichero ipc-2020.csv contiene el IPC de las comunidades autónomas de los cinco primeros meses de 2020. Crear un programa que realice las siguientes operaciones utilizando la librería Pandas:

1. Crear un DataFrame leyendo el fichero desde internet con la url `https://aprendeconalf.es/docencia/python/examenes/inteligencia-negocios/soluciones/examen-2020-06-19/ipc-2020.csv`.
2. Mostrar por pantalla el DataFrame con los datos de las filas 10 a 15.
3. Mostrar por pantalla el DataFrame con los datos de Canarias de Mayo.
4. Mostrar por pantalla una serie con el IPC mensual máximo de cada comunidad autónoma.
5. Mostrar por pantalla una serie con la desviación típica del IPC mensual de cada grupo.
6. Mostrar por pantalla un DataFrame con las comunidades y grupos donde los precios no han subido en promedio (IPC mensual medio menor de 100).

Solución

Solución 1

```
import pandas as pd

# Apartado 1

# Leemos el DataFrame desde la url
ipc = pd.read_csv("https://aprendeconalf.es/docencia/python/examenes
 ↪  /inteligencia-negocios/soluciones/examen-2020-06-19/ipc-2020.csv
 ↪  ")
ipc
```

```
     Comunidad autónoma                                Grupo   Año
      ↪  Mes  \
0             Andalucía  Alimentos y bebidas no alcohólicas  2020
 ↪  Mayo
1             Andalucía  Alimentos y bebidas no alcohólicas  2020
 ↪  Abril
2             Andalucía  Alimentos y bebidas no alcohólicas  2020
 ↪  Marzo
```

```
3             Andalucía  Alimentos y bebidas no alcohólicas  2020
 ↪  Febrero
4             Andalucía  Alimentos y bebidas no alcohólicas  2020
 ↪  Enero
...                 ...                                 ...   ...
 ↪  ...
1135            Melilla          Otros bienes y servicios   2020
 ↪  Mayo
1136            Melilla          Otros bienes y servicios   2020
 ↪  Abril
1137            Melilla          Otros bienes y servicios   2020
 ↪  Marzo
1138            Melilla          Otros bienes y servicios   2020
 ↪  Febrero
1139            Melilla          Otros bienes y servicios   2020
 ↪  Enero

          IPC
0     107.260
1     107.332
2     105.611
3     105.435
4     105.058
...       ...
1135  101.052
1136  100.773
1137  100.986
1138  101.040
1139  100.593

[1140 rows x 5 columns]
```

```
# Apartado 2

# Serie con los datos de la fila 10 a la 15
print(ipc.iloc[9:14,])
```

```
Comunidad autónoma                          Grupo   Año      Mes
 ↪  IPC
9           Andalucía  Bebidas alcohólicas y tabaco  2020    Enero
 ↪  103.436
10          Andalucía             Vestido y calzado  2020     Mayo
 ↪  110.538
```

```
11          Andalucía             Vestido y calzado  2020    Abril
 ↪  108.010
12          Andalucía             Vestido y calzado  2020    Marzo
 ↪  97.741
13          Andalucía             Vestido y calzado  2020  Febrero
 ↪  92.678
```

```
# Apartado 3

# Ipc de Canarias de Mayo
ipc[(ipc['Comunidad autónoma'] == 'Canarias') & (ipc['Mes'] == 'Mayo
 ↪  ')]
```

```
    Comunidad autónoma
     ↪  Grupo  \
240           Canarias                 Alimentos y bebidas no
 ↪  alcohólicas
245           Canarias                       Bebidas alcohólicas y
 ↪  tabaco
250           Canarias                                  Vestido y
 ↪  calzado
255           Canarias  Vivienda, agua, electricidad, gas y otros
 ↪  comb...
260           Canarias  Muebles, artículos del hogar y artículos
 ↪  para ...
265           Canarias
 ↪  Sanidad
270           Canarias
 ↪  Transporte
275           Canarias
 ↪  Comunicaciones
280           Canarias                                      Ocio y
 ↪  cultura
285           Canarias
 ↪  Enseñanza
290           Canarias                          Restaurantes y
 ↪  hoteles
295           Canarias                       Otros bienes y
 ↪  servicios

      Año   Mes      IPC   Periodo
240  2020  Mayo  106.183  2020Mayo
```

```
245  2020  Mayo  118.690  2020Mayo
250  2020  Mayo  111.108  2020Mayo
255  2020  Mayo   99.291  2020Mayo
260  2020  Mayo   98.667  2020Mayo
265  2020  Mayo  100.323  2020Mayo
270  2020  Mayo  105.817  2020Mayo
275  2020  Mayo  104.871  2020Mayo
280  2020  Mayo   97.686  2020Mayo
285  2020  Mayo  100.905  2020Mayo
290  2020  Mayo  108.149  2020Mayo
295  2020  Mayo  103.802  2020Mayo
```

```python
# Apartado 4

# IPC máximo mensual por comunidades
ipc.groupby('Comunidad autónoma').IPC.max()
```

```
Comunidad autónoma
Andalucía                       110.538
Aragón                          110.459
Asturias, Principado de         111.028
Balears, Illes                  110.175
Canarias                        118.868
Cantabria                       113.443
Castilla - La Mancha            110.282
Castilla y León                 112.252
Cataluña                        110.784
Ceuta                           113.244
Comunitat Valenciana            111.834
Extremadura                     110.652
Galicia                         109.438
Madrid, Comunidad de            110.541
Melilla                         110.751
Murcia, Región de               111.972
Navarra, Comunidad Foral de     110.834
País Vasco                      111.777
Rioja, La                       115.892
Name: IPC, dtype: float64
```

```
# Apartado 5

# Desviación típica del IPC mensual por grupos ordenado de mayor a m
 ↪  enor
ipc.groupby('Grupo').IPC.std().sort_values(ascending = False)
```

```
Grupo
Vestido y calzado
 ↪  7.473487
Transporte
 ↪  4.021121
Bebidas alcohólicas y tabaco
 ↪  3.125740
Vivienda, agua, electricidad, gas y otros combustibles
 ↪  2.566030
Enseñanza
 ↪  1.706892
Restaurantes y hoteles
 ↪  1.505234
Muebles, artículos del hogar y artículos para el mantenimiento
 ↪  corriente del hogar    1.479133
Otros bienes y servicios
 ↪  1.314706
Alimentos y bebidas no alcohólicas
 ↪  1.274428
Comunicaciones
 ↪  1.230438
Ocio y cultura
 ↪  1.112717
Sanidad
 ↪  1.089534
Name: IPC, dtype: float64
```

```
# Apartado 6

# Comunidades y grupos con un IPC mensual medio menor de 100.
ipc_medias = ipc.groupby(['Comunidad autónoma','Grupo']).IPC.mean()
ipc_medias[ipc_medias < 100]
```

```
Comunidad autónoma           Grupo
Andalucía                    Muebles, artículos del hogar y
 ↪  artículos para el mantenimiento corriente del hogar    99.4302
```

```
                            Ocio y cultura
                             ↪ 99.1484
Aragón                      Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    98.6628
                            Ocio y cultura
                             ↪ 98.5150
Asturias, Principado de     Enseñanza
 ↪ 99.5982
                            Ocio y cultura
                             ↪ 97.8752
Balears, Illes              Ocio y cultura
 ↪ 99.4216
Canarias                    Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    98.2600
                            Ocio y cultura
                             ↪ 98.1462
Cantabria                   Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    99.6226
                            Ocio y cultura
                             ↪ 97.4698
                            Vivienda, agua, electricidad, gas y
                             ↪ otros combustibles
                             ↪ 99.9460
Castilla - La Mancha        Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    99.7986
                            Ocio y cultura
                             ↪ 98.0030
Castilla y León             Ocio y cultura
 ↪ 99.4268
Ceuta                       Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    96.6706
                            Ocio y cultura
                             ↪ 97.9256
Comunitat Valenciana        Vivienda, agua, electricidad, gas y
 ↪ otros combustibles                                  99.6172
Extremadura                 Muebles, artículos del hogar y
 ↪ artículos para el mantenimiento corriente del hogar    99.9848
                            Ocio y cultura
                             ↪ 99.1018
                            Vivienda, agua, electricidad, gas y
                             ↪ otros combustibles
                             ↪ 99.3100
```

```
Galicia                       Ocio y cultura
 ↪  98.5452
Melilla                       Ocio y cultura
 ↪  98.2606
                              Transporte
                               ↪  99.7854
Murcia, Región de             Muebles, artículos del hogar y
 ↪  artículos para el mantenimiento corriente del hogar    99.2730
                              Ocio y cultura
                               ↪  97.8524
                              Vivienda, agua, electricidad, gas y
                               ↪  otros combustibles
                               ↪  99.0846
Navarra, Comunidad Foral de   Muebles, artículos del hogar y
 ↪  artículos para el mantenimiento corriente del hogar    99.8240
                              Ocio y cultura
                               ↪  99.4248
Rioja, La                     Ocio y cultura
 ↪  99.0350
                              Vivienda, agua, electricidad, gas y
                               ↪  otros combustibles
                               ↪  99.7910
Name: IPC, dtype: float64
```

Ver notebook de la solución en Colab

6 2021-03-25 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 25 de marzo de 2021

Ejercicio 6.1. El cálculo del IRPF en la Hacienda española se define como progresivo. Hacienda divide los ingresos (tu renta) en tramos y asigna un porcentaje a pagar en cada uno de ellos. Estos tramos son los siguientes:

Tramos IRPF 2021	Tipos a aplicar
Desde 0 hasta 12.450€	19 %
De 12.450€ a 20.200€	24 %
De 20.200€ a 35.200€	30 %
De 35.200€ en adelante	37 %

Por ejemplo, para una persona con una renta de 65.000€, el cálculo del impuesto se haría así:

- Primer tramo IRPF: se paga el 19 % de 12.450 euros, es decir, 2.365,5 euros
- Segundo tramo IRPF: se paga el 24 % de 7.750 euros (la diferencia entre el primer y segundo tramo), es decir, 1.860 euros.
- Tercer tramo IRPF: se paga el 30 % de 15.000 euros (la diferencia entre el segundo y tercer tramo), es decir, 4.500 euros.
- Cuarto tramo IRPF: se paga el 37 % de 29.800 euros (la diferencia entre su renta y el límite del tercer tramo), es decir, 11.026 euros.

La suma de las anteriores cantidades es el total a pagar: 19.751,5 euros.

Escribir un programa que pregunte por la renta del usuario y muestre por pantalla el IRPF que debe pagar a Hacienda.

Solución

Solución 1

```python
# Preguntamos por la renta
renta = int(input('Introduce tu renta: '))
# Condicional. Si la renta es menor que 12450€ aplicamos un tipo del
 ↪  19%.
if (renta < 12450):
    impuesto = renta * 0.19
# Si la renta es mayor que 12450€ y menor que 20200€ aplicamos el 1
 ↪  9% a los
# primeros 12450€ y un 24% al resto.
elif (renta < 20200):
    impuesto = 12450 * 0.19 + (renta - 12450) * 0.24
# Si la renta es mayor que 20200€ y menor que 35200€ aplicamos el 1
 ↪  9% a los
# primeros 12450€, el 24% a los siguientes 7750€ y el 30% al resto.
elif (renta < 35200):
    impuesto = 12450 * 0.19 + (20200 - 12450) * 0.24 + (renta -
 ↪  20200) * 0.3
# Finalmente, si la renta es mayor que 35200€ aplicamos el 19% a los
 ↪  primeros
# 12450€, el 24% a los siguientes 7750€, el 30% a los siguientes 15
 ↪  0000€ y el
# 37% al resto.
else:
    impuesto = 12450 * 0.19 + (20200 - 12450) * 0.24 + (35200 -
 ↪  20200) * 0.3 + (renta - 35200) * 0.37
# Mostramos por pantalla el total del impuesto a pagar.
print('Tienes que pagar:', impuesto)
```

```
Tienes que pagar: 19751.5
```

Ver notebook de la solución en Colab

Ejercicio 6.2. Juan juega siempre la misma combinación a la bonoloto: 7, 13, 21, 37, 46, 49.

Construir un programa que pregunte al usuario por la combinación ganadora y diga si Juan ha ganado o, en caso contrario, muestre por pantalla la lista de los números que no ha acertado. El programa debe usar listas.

Nota: El juego de la bonoloto consiste en acertar una combinación de 6 números entre 1 y 49.

Solución

```
# Guardamos la combinación de Juan en una lista.
combinacion = [7, 13, 21, 37, 46, 50]
# Bucle iterativo para preguntar los números de la combinación ganad
 ↪  ora.
# i toma valores de 0 a 5, de manera que el bucle da 6 vueltas.
fallos = []
for i in range(6):
    # Preguntamos al usuario por el siguiente número de la combinaci
     ↪  ón ganadora.
    num = int(input('Introduce el siguiente número de la combinación
 ↪  ganadora: '))
    # Condicional. Si el número de la combinación ganadora no está e
     ↪  n la lista de
    # de Juan, lo añadimos a la lista de fallos.
    if not num in combinacion:
        fallos.append(num)
# Condicional. Si al comprobar todos los números de la combinación g
 ↪  anadora la
# lista de fallos está vacía, es que ha ganado.
if len(fallos) == 0:
    print('¡Enhorabuena, ganaste!')
# En caso contrario, Juan no ha ganado y mostramos los números que q
 ↪  uedan en la
# lista, que son los que no han salido en la combinación ganadora.
else:
    print('Los siento, no acertaste los siguientes números:',
     ↪  fallos)
```

```
Los siento, no acertaste los siguientes números: [2, 44, 38, 25]
```

Ver notebook de la solución en Colab

Ejercicio 6.3. Los productos en oferta de una tienda de informática se guardan una cadena como la de más abajo, donde cada línea (separadas por el carácter de cambio de línea `\n`) contiene el nombre del producto, el número de unidades en stock, el precio (en €) y el descuento que tiene (en porcentaje), separados por punto y coma.

```
'disco duro 500Gb;200;25;15\nmemoria ram 16Gb;500;30;20\nratón inalámbri
 ↪  co;800;12;10\ntarjeta wifi;150;20;12'
```

Construir un programa que genere, a partir de la cadena anterior, un diccionario como el

de más abajo, donde cada par corresponda un producto, siendo la clave el nombre del producto y el valor una lista con el resto de la información del producto.

```
{'disco duro 500Gb': ['200', '25', '15'], 'memoria ram 16Gb': ['500',
↪  '30', '20'], 'ratón inalámbrico': ['800', '12', '10'], 'tarjeta
↪  wifi': ['150', '20', '12']}
```

Después el programa debe recorrer el diccionario y mostrar por pantalla un listado con los nombres de todos los productos en oferta y su precio final tras aplicar el descuento.

Solución

```
# Guardamos la cadena con los productos en oferta en una
cadena_ofertas = 'disco duro 500Gb;200;25;15\nmemoria ram 16Gb;500;3
 ↪  0;20\nratón inalámbrico;800;12;10\ntarjeta wifi;150;20;12'
# Partimos la cadena por el carácter de cambio de línea '\n' y cream
 ↪  os una lista
# de subcadenas, donde cada subcadena contiene los datos de un produ
 ↪  cto.
lista_ofertas = cadena_ofertas.split('\n')
# Creamos un diccionario vacío para guardar los productos
ofertas = {}
# Bucle iterativo para recorrer la lista con las cadenas que contien
 ↪  e la
# información de los productos y crear el diccionario de productos.
# i toma como valores cada una de las subcadenas.
for i in lista_ofertas:
    # Partimos la subcadena i por el carácter ';' y creamos una list
     ↪  a de
    # subcadenas donde cada subcadena contiene una información del p
     ↪  roducto
    info = i.split(';')
    # Añadimos al diccionario la sublista desde el segundo elemento
     ↪  hasta el final,
    # usando como clave el primer elemento de la lista, que es el no
     ↪  mbre del producto.
    ofertas[info[0]] = info[1:]
# Bucle iterativo para recorrer el diccionario que hemos creado por
 ↪  pares.
# producto recorre las claves y info los valores de los pares del di
 ↪  ccionario.
for producto, info in ofertas.items():
    # Calculamos el precio final del producto aplicando su correspon
     ↪  diente descuento.
    precio = float(info[1]) * (1 - float(info[2]) / 100)
    # Mostramos por pantalla el nombre del producto y su precio fina
     ↪  l.
    print(producto, 'vale', precio, '€')
```

```
disco duro 500Gb vale 21.25 €
memoria ram 16Gb vale 24.0 €
ratón inalámbrico vale 10.8 €
tarjeta wifi vale 17.6 €
```

Ver notebook de la solución en Colab

7 2021-05-26 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 26 de mayo de 2021

Ejercicio 7.1. Dadas dos listas de números del mismo tamaño `x` e `y`, construir las siguientes funciones:

1. Una función para calcular la media de una lista de números.
2. Una función para calcular la varianza de una lista de números.
3. Una función para calcular la covarianza de dos listas de números.
4. Una función para calcular los coeficientes de la recta de regresión de `y` sobre `x`.
5. Una función que devuelva el diagrama de dispersión y la recta de regresión como la que se muestra en el siguiente ejemplo:

```
x = [1, 2, 3, 4, 5, 6]
y = [20, 18, 12, 10, 9, 9]
```

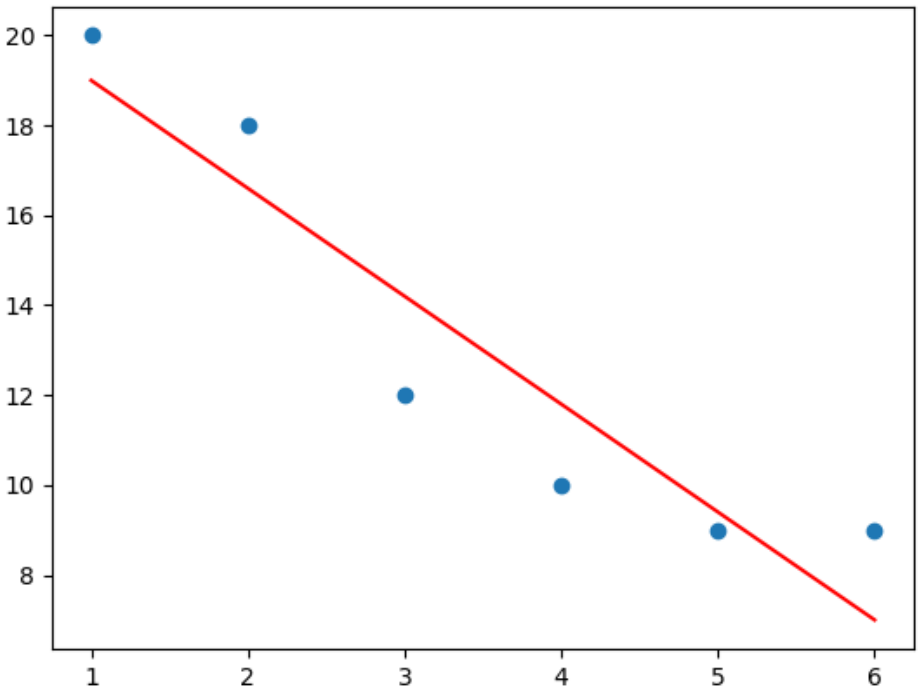

Figura 7.1: 'Recta de regresión'

Usar las siguientes fórmulas para el cálculo de los estadísticos: $\bar{x} = \frac{\sum x_i}{n}$, $s^2 = \frac{\sum x_i^2}{n} - \bar{x}^2$, $s_{xy} = \frac{\sum x_i y_j}{n} - \bar{x}\bar{y}$, $b = \frac{s_{xy}}{s_x^2}$ y $a = \bar{y} - b * \bar{x}$.

Solución

```python
import matplotlib.pyplot as plt

def media(x):
  '''Función que calcula la media de una lista de números.

  Parámetros:
    - x: Es una lista de números.

  Salida:
    La media de los números de la lista dada.
  '''
  return sum(x) / len(x)

def varianza(x):
  '''Función que calcula la varianza de una lista de números.

  Parámetros:
    - x: Es una lista de números.

  Salida:
    La varianza de los números de la lista dada.
  '''
  # Inicializamos una variable para calcular la suma de los números.
  suma = 0
  # Bucle para recorrer los números de la lista.
  for i in x:
    # Añadimos el número a la suma parcial.
    suma += i ** 2
  return suma / len(x) - media(x) ** 2

def covarianza(x, y):
  '''Función que calcula la covarianza de dos listas de números.

  Parámetros:
    - x: Es una lista de números.
    - y: Es una lista de números del mismo tamaño que x.

  Salida:
    La covarianza de las listas dadas.
  '''
  suma = 0
  # Bucle iterativo para recorrer los números de las listas x e y.
  # i toma como valores los enteros de 0 al tamaño de las listas menos 1.
  for i in range(len(x)):
    suma += x[i] * y[i]
  return suma / len(x) - media(x) * media(y)

def recta_regresion(x, y):
  '''Función que calcula los coeficientes de la recta de regresión de dos listas de números.

  Parámetros:
    - x: Es una lista de números.
    - y: Es una lista de números del mismo tamaño que x.
```

```
<Figure size 432x288 with 1 Axes>
```

Ver notebook de la solución en Colab

Ejercicio 7.2. Escribir una función que, dado un número `x`, genere una matriz cuadrada (`x*x` elementos) con el resultado de sus tablas de multiplicar desde 0 hasta `x`. El resultado debe guardarse en un fichero llamado `tabla-X.txt` donde `X` es el número introducido por el usuario.

Escribir otra función que, dado un número `x`, acceda al fichero de la tabla correspondiente y muestre la tabla por pantalla. En caso de que la tabla no exista deberá controlar la excepción para mostrar un mensaje de aviso al usuario.

Ejemplo:

```
>>> Introduce un número entre 1 y 10: 5
>>> Contenido del fichero matriz-5.txt
0    0    0    0    0    0
0    1    2    3    4    5
0    2    4    6    8    10
0    3    6    9    12   15
0    4    8    12   16   20
0    5    10   15   20   25
```

Solución

```
def generar_tabla(n):
    '''Función que crea un fichero con la tabla de multiplicar cuadr
     ↪  ada de un número dado.

    Parámetros:
      - n: Es un número entero entre 1 y 10.
    '''
    # Creamos el nombre del fichero.
    nombre_fichero = "tabla-" + str(n) + ".txt"
    # Abrimos el fichero en modo escritura.
    f = open(nombre_fichero, 'w')
    # Bucle iterativo para recorrer las filas de la tabla.
    # i toma como valores los enteros de 0 a n.
    for i in range(n+1):
        # Bucle anidado iterativo para recorrer las columnas de la t
         ↪  abla.
        # j toma como valores los enteros de 0 a n.
        for j in range(n+1):
            # Escribimos en el fichero el producto de i por j y añad
             ↪  imos un espacio o tabulador.
            f.write(str(i*j) + "\t")
        # Una vez recorridas todas las columnas escribimos en el fic
         ↪  hero un cambio de línea para pasar a la siguiente línea.
        f.write("\n")
    # Cerramos el fichero
    f.close()
    return

def leer_tabla(n):
    '''Función que lee un fichero con una tabla de multiplicar cuadr
     ↪  ada de un número dado y la muestra por pantalla.

    Parámetros:
      - n: Es un número entero entre 1 y 10.
    '''
    # Creamos el nombre del fichero.
    nombre_fichero = "tabla-" + str(n) + ".txt"
    print("Contenido del fichero " + nombre_fichero + ":")
    try:
        # Intentamos abrir el fichero en modo lectura.
        fichero = open(nombre_fichero, 'r')
    except FileNotFoundError:
        # Si el fichero no existe mostramos por pantalla un mensaje
         ↪  de aviso.
        print("El fichero no existe")
    else:
        # Leemos el contenido del fichero y lo mostramos por pantall
         ↪  a.
        print(fichero.read())
        # Cerramos el fichero.
        fichero.close()

n = int(input("Introduce un número entero entre 1 y 10: "))
generar_tabla(n)
leer_tabla(n)
```

```
Contenido del fichero tabla-5.txt:
0   0   0   0   0   0
0   1   2   3   4   5
0   2   4   6   8   10
0   3   6   9   12  15
0   4   8   12  16  20
0   5   10  15  20  25
```

Ver notebook de la solución en Colab

Ejercicio 7.3. Crear un programa utilizando la librería Pandas y Matplotlib que realice lo siguiente:

1. Crear el siguiente DataFrame indexado: `sh  calorias  tiempo  L  420  60  M  380  40  X  390  75  J  490  55  V  300  45`
2. Calcular la media, mediana y desviación típica de ambas columnas.
3. Añadir otra columna booleana al DataFrame para ver si se ha cumplido el reto de quemar más de 400 calorías por hora. La nueva columna debe generarse aplicando una fórmula a las otras columnas. El DataFrame resultante debe ser el siguiente: `sh  calorias  tiempo  reto  L  420  60  True  M  38  0  40  True  X  390  75  False  J  490  55  True  V  300  45  False`
4. Filtrar el DataFrame y devolver otro DataFrame con las filas pares que cumplan que el número de calorías es mayor de 400.
5. Crear a partir del DataFrame una serie con los porcentajes de días que se ha conseguido el reto y los que no.
6. Crear un gráfico como el de más abajo que muestre la progresión de las calorías y tiempo durante la semana.

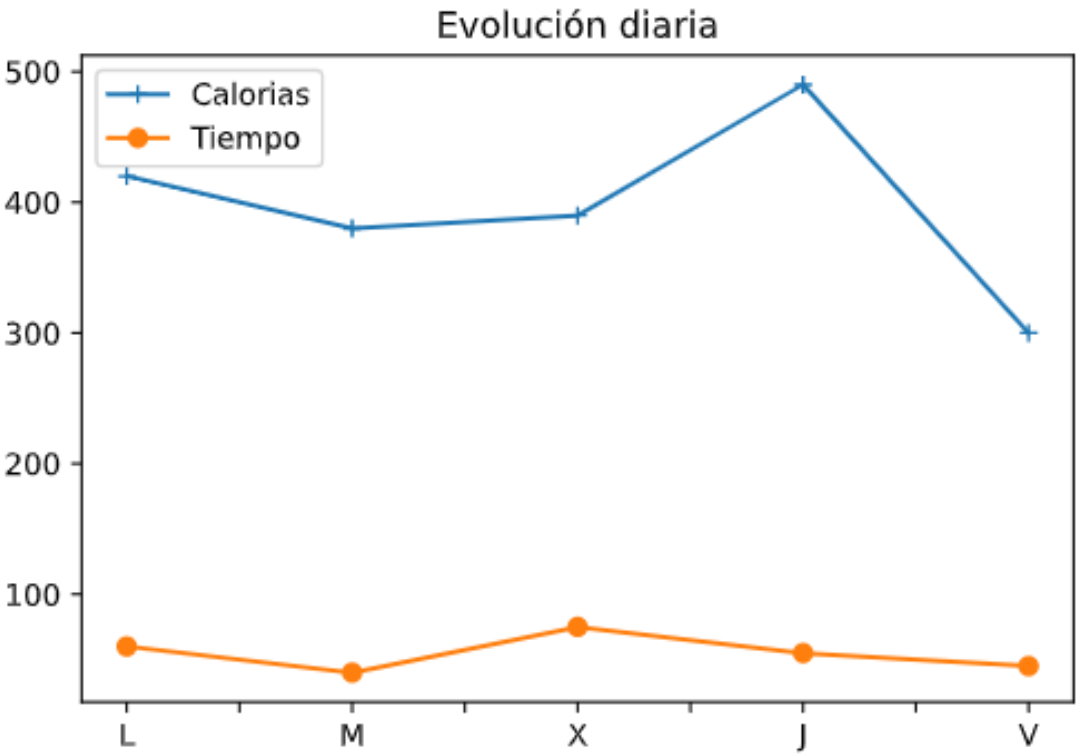

Figura 7.2: Serie evolución calorias

Solución

```
import pandas as pd
import matplotlib.pyplot as plt

# Creamos un diccionario con claves los nombres de las columnas y va
 ↪  lores las listas con los elementos de las columnas.
datos = {
  "Calorias": [420, 380, 390, 490, 300],
  "Tiempo": [60, 40, 75, 55, 45]
}
# Creamos una lista con los días de la semana para utilizarla como í
 ↪  ndice
semana = ["L", "M", "X", "J", "V"]
# Creamos el DataFrame
df = pd.DataFrame(datos, semana)
print(df)
```

```
   Calorias  Tiempo
L       420      60
M       380      40
X       390      75
J       490      55
V       300      45
```

```python
# Calculamos la media.
print('Medias:')
print(df.mean())
# Calculamos la mediana.
print('Medianas:')
print(df.median())
# Calculamos la desviación típica.
print('Desviaciones típicas:')
print(df.std())
```

```
Medias:
Calorias    396.0
Tiempo       55.0
dtype: float64
Medianas:
Calorias    390.0
Tiempo       55.0
dtype: float64
Desviaciones típicas:
Calorias    68.774995
Tiempo      13.693064
dtype: float64
```

```python
# Añadimos una nueva columna comparando las calorisas consumidas por
 ↪  hora con 400 calorias por 60 minutos.
df["eto"] = df.Calorias / df.Tiempo > 400 / 60
print(df)
```

```
   Calorias  Tiempo   reto   Reto
L       420      60   True   True
M       380      40   True   True
X       390      75  False  False
J       490      55   True   True
V       300      45  False  False
```

```
# Obtenemos la filas pares.
df_filtrado = df.iloc[range(0, df.shape[0], 2)]
# Obtenemos las filas con un número de calorías mayor de 400.
df_filtrado = df_filtrado[df_filtrado.Calorias > 400]
print(df_filtrado)
```

```
   Calorias  Tiempo  reto
L       420      60  True
```

```
# Calculamos los porcentajes de días en los que se ha cumplido el re
 ↪  to y los que no.
print(df.Reto.value_counts(normalize = True) * 100)
```

```
True     60.0
False    40.0
Name: Reto, dtype: float64
```

```
fig, ax = plt.subplots()
df.plot(y = 'Calorias', marker = "+", ax = ax)
df.plot(y = 'Tiempo', marker = "o", ax = ax)
plt.title('Evolución diaria')
plt.show()
```

```
<Figure size 432x288 with 1 Axes>
```

Ver notebook de la solución en Colab

8 2021-06-05 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 5 de junio de 2021

Ejercicio 8.1. A veces, al tratar con tuplas, podemos tener un problema en el que necesitamos extraer solo k elementos extremos, es decir, los k máximos y mínimos. Este problema puede tener aplicaciones en campos como el desarrollo web y la ciencia de datos. Desarrollar un programa que dada una tupla y un número k devuelva otra tupla con los k elementos máximos y mínimos.

Ejemplo:

```
La tupla original es: (5, 20, 3, 7, 6, 8)
La tupla con los k = 2 máximos y mínimos es: (3, 5, 8, 20)
```

Solución

Solución 1

```
# Definimos la tupla
tupla = (5, 20, 3, 7, 6, 8)
# Preguntamos por el valor de k
k = int(input('¿Cuántos máximos y mínimos quieres? '))
# Convertimos la tupla en una lista para poder eliminar elmentos.
lista = list(tupla)
# Condicional para ver si la lista contiene suficientes elementos.
if len(lista) < 2 * k:
  print('No se pueden extraer', k, 'máximos y mínimos de la tupla po
   ↪  rque solo cotinene', len(lista), 'elementos.')
else:
  # Ordenamos la lista
  lista.sort()
  # Creamos una tupla concatenando los k primeros y los k últimos el
   ↪  ementos de la lista ordenada.
  tupla2 = tuple(lista[:k] + lista[-k:])
  print(tupla2)
```

```
(3, 5, 8, 20)
```

Solución 2

```
# Definimos la tupla
tupla = (5, 20, 3, 7, 6, 8)
# Preguntamos por el valor de k
k = int(input('¿Cuántos máximos y mínimos quieres? '))
# Convertimos la tupla en una lista para poder eliminar elmentos.
lista = list(tupla)
# Condicional para ver si la lista contiene suficientes elementos.
if len(lista) < 2 * k:
  print('No se pueden extraer ', k, 'máximos y mínimos de la tupla p
   ↪  orque solo cotinene', len(lista), 'elementos.')
else:
  # Creamos una lista vacía para ir añadiendo los máximos y mínimos.
  lista2 = []
  # Bucle iterativo para realizar k iteraciones. i toma valores de 0
   ↪  a k-1.
  for i in range(k):
    # Añadimos a la lista2 el mínimo de la lista.
    lista2.append(min(lista))
    # Eliminiamos de la lista el mínimo.
    lista.remove(min(lista))
    # Añadimos a la lista 2 el máximo de la lista.
    lista2.append(max(lista))
    # Eliminiamos de la lista el máximo.
    lista.remove(max(lista))
  # Convertimos la lista2 en una tupla.
  tupla2 = tuple(lista2)
  print(tupla2)
```

```
(3, 20, 5, 8)
```

Ver notebook de la solución en Colab

Ejercicio 8.2. Construir un programa que evalúe operaciones aritméticas sencillas (sumas, restas, productos, cocientes y potencias) introducidas por el usuario. El programa preguntará por la operación a realizar y el usuario tecleará por pantalla la operación con el siguiente formato:

```
operando1 operador operando2
```

Después el programa debe mostrar por pantalla el resultado de la operación con el siguiente formato:

```
operando1 operador operando2 = resultado
```

El programa debe preguntar al usuario hasta que este introduzca la palabra “salir”. También mostrará un mensaje de error si el usuario introduce un número de valores distinto de 3 y si introduce un operador no válido.

Ejemplo:

```
Introduce la operación con el formato operando1 operador operando2: 2+3
Entrada no válida. Debes introducir exactamente tres valores separados
 ↪  por espacio.
Introduce la operación con el formato operando1 operador operando2: 2 +
 ↪  3
2 + 3 = 5.0
Introduce la operación con el formato operando1 operador operando2: 4 ^
 ↪  2
Operación no válida.
Introduce la operación con el formato operando1 operador operando2: 4
 ↪  ** 2
4 ** 2 = 16.0
Introduce la operación con el formato operando1 operador operando2:
 ↪  salir
```

Solución

```
# Bucle condicional para preguntar al usuario hasta que introduzca s
 ↪  alir.
while True:
    # Preguntamos al usuario por la operación a realizar
    operacion = input("Introduce la operación con el formato operand
 ↪  o1 operador operando2:")
    # Si el usuario introduce salir salimos del bucle y terminamos
    if operacion.lower() == 'salir':
        break
    # Dividimos la entrada por el espacio en blanco y obtenemos la l
     ↪  ista de valores
    operacion = operacion.split()
    # Condicional para ver si la lista no contiene 3 valores. En tal
     ↪  caso mostramos un mensaje de error.
    if len(operacion) != 3:
        print('Operación no válida. Se necesitan tres argumentos sep
         ↪  arados por espacio.')
    # Si entrada es válida hacemos la operación.
    else:
        # Creamos variables para los operandos y el operador
        a = float(operacion[0])
        operador = operacion[1]
        b = float(operacion[2])
        if operador == "+":
            print(a, operador, b, '=', a + b)
        elif operador == "-":
            print(a, operador, b, '=', a - b)
        elif operador == "*":
            print(a, operador, b, '=', a * b)
        elif operador == "/":
            print(a, operador, b, '=', a / b)
        elif operador == "**":
            print(a, operador, b, '=', a ** b)
        else:
            print('Operador no válido. Introduce uno de los siguient
             ↪  es operadores: + (suma), - (resta), * (producto), /
             ↪  (cociente), ** (potencia).')
```

```
Operación no válida. Se necesitan tres argumentos separados por
 ↪  espacio.
2.0 + 3.0 = 5.0
```

```
Operador no válido. Introduce uno de los siguientes operadores: +
 ↪  (suma), - (resta), * (producto), / (cociente), ** (potencia).
4.0 ** 2.0 = 16.0
```

Ver notebook de la solución en Colab

Ejercicio 8.3. El siguiente diccionario contiene pares formados por números de teléfonos y propietarios:

```
{'919654665':'Pedro', '917489210': 'Luis', '623543213':'Carmen', '674833
721':'Luis'}
```

Crear un programa que construya un diccionario con la misma información, pero tomando como claves los nombres de los usuarios y como valores los teléfonos. Como un usuario puede tener dos teléfonos, uno móvil y uno fijo, los teléfonos deben agruparse a su vez en un diccionario cuyos elementos tendrán clave "movil" o "fijo" según el teléfono empiece por 6 o no. Por ejemplo, a partir del diccionario anterior debe construirse el siguiente diccionario:

```
{'Pedro': {'fijo': '919654665'}, 'Luis': {'fijo': '917489210', 'movil':
'674833721'}, 'Carmen': {'movil': '623543213'}}
```

Después el programa debe recorrer este diccionario y mostrar por pantalla los teléfonos del listín en orden alfabético.

Ejemplo:

```
Teléfonos de Carmen
        movil : 623543213
Teléfonos de Luis
        fijo : 917489210
        movil : 674833721
Teléfonos de Pedro
        fijo : 919654665
```

Solución

```
# Definimos el diccionario con el listín telefónico.
listin = {'919654665':'Pedro', '917489210': 'Luis', '623543213':'Car
 ↪  men', '674833721':'Luis'}
# Definimos un diccionario vacío para el nuevo listín telefónico.
listin2 = {}
# Bucle iterativo para recorrer el diccionario. i toma como valores
 ↪  los valores del diccionario.
for i in listin.values():
  # Añadimos al listin nuevo un par con la clave el nombre de la per
   ↪  sona y el valor un diccionario vacío.
  listin2[i] = {}
# Bucle iterativo para recorrer el diccionario. k toma como valores
 ↪  las claves y v los valores.
for k,v in listin.items():
  # Condicional para ver si el teléfono empieza por 6.
  if k[0] == '6':
    # Si el teléfono empieza por 6 añadimos al diccionario de la per
     ↪  sona un nuevo par con la clave 'movil' y el valor el telefon
     ↪  o.
    listin2[v]['movil'] = k
  else:
    # Si el teléfono no empieza por 6 añadimos al diccionario de la
     ↪  persona un nuevo par con la clave 'fijo' y el valor el telef
     ↪  ono.
    listin2[v]['fijo'] = k
print(listin2)
# Bucle iterativo para recorrer el segundo diccionario alfabéticamen
 ↪  te. i toma como valores las claves.
for i in sorted(listin2.keys()):
  print('Teléfonos de', i)
  # Bucle iterativo anidado para recorrer el diccionario de los telé
   ↪  fonos de una persona. k toma como valores las claves y v los v
   ↪  alores.
  for k,v in listin2[i].items():
    print('\t', k, ':', v)
```

```
{'Pedro': {'fijo': '919654665'}, 'Luis': {'fijo': '917489210',
 ↪  'movil': '674833721'}, 'Carmen': {'movil': '623543213'}}
Teléfonos de Carmen
     movil : 623543213
Teléfonos de Luis
```

```
     fijo : 917489210
     movil : 674833721
Teléfonos de Pedro
     fijo : 919654665
```

Ver notebook de la solución en Colab

Ejercicio 8.4. Construir un programa para realizar las siguientes operaciones con dos números proporcionados por el usuario:

- Comprobar si uno de los dos números es divisible por el otro. Un número es divisible por otro cuando el resto de la división entera es cero.
- Calcular su Máximo Común Divisor.
- Calcular su Mínimo Común Múltiplo.

Cada una de las operaciones deben estar separadas en funciones que tengan como parámetros los 2 números y devuelvan el resultado adecuado.

Solución

```python
def div(a, b):
    '''Función que comprueba si un número es divisible por otro.

    Parámetros:
        - a: Es un número entero.
        - b: Es un número entero.

    Salida: Un booleano, True si alguno de los números es divisible
 ↪  por el otro y False en caso contrario.
'''
    return a % b == 0 or b % a == 0

def mcd(a, b):
    '''Función que devuelve el máximo común divisior de dos números
     ↪  por el algoritmo de Euclides.

    Parámetros:
        - a: Es un número entero.
        - b: Es un número entero.

    Salida: El máximo común divisor de los dos números dados.
    '''
    # Bucle condicional mientras b (el divisor) sea distinto de 0.
    while b:
        # b pasa a ser el nuevo dividendo y el resto de la división
         ↪  entera de a por b pasa a ser el nuevo divisor
        a, b = b, a%b
    # Devolvemos el último divisor distinto de 0.
    return a

def mcm(a, b):
    '''Función que devuelve el mínimo común múltiplo de dos números
     ↪  por el algoritmo de Euclides.

    Parámetros:
        - a: Es un número entero.
        - b: Es un número entero.

    Salida: El mínimo común múltiplo de los dos números dados.
    '''
    # Aprovechando que tenemos el mcd aplicamos la propiedad a * b =
     ↪  mcd(a,b) * mcm (a,b).
    return a / mcd(a,b) * b


print(div(12, 18))
print(mcd(12, 18))
print(mcm(12, 18))
```

```
False
6
36.0
```

Ver notebook de la solución en Colab

Ejercicio 8.5. El fichero `cercanias.csv` contiene información sobre las líneas de tren de cercanías de Madrid: id (identificador del tren), línea (nombre de la línea), estaciones (estaciones de origen y destino separadas por un guion). Se pide:

1. Construir una función que lea el fichero `cercanias.csv` y cree un diccionario donde la clave de cada par sea el identificador de la línea y el valor asociado una lista de dos elementos con la estación de origen y la estación de destino como el que se muestra a continuación a modo de ejemplo:

```
{'10T0001C1': ['Principe Pio', 'Aeropuerto'], '10T0002C1': ['Aeropuerto'
, 'Principe Pio'], ...}
```

La función debe recibir el nombre del fichero como parámetro. 2. Construir otra función que guarde la información del diccionario obtenido en el apartado anterior en un fichero csv separado por punto y coma con 3 columnas con los encabezados id, origen y destino. La función debe recibir como parámetros el diccionario con los trenes y el nombre del fichero resultante.

Solución

```
def importar_trenes(fichero):
    '''Función que genera un diccionario con las líneas de trenes (o
     ↪  rigen y destino) a partir de un fichero de texto.

    Parámetros:
        - fichero: Es una cadena con el nombre del fichero.

    Salida: Un diccionario con las líneas de tren. Las claves son lo
 ↪  s identificadores de las líneas y los valores son listas de dos
 ↪  elementos con el origen y el destino.
    '''
    try:
        # Intentamos abrir el fichero en modo lectura.
        f = open(fichero, 'r')
    except FileNotFoundError:
        # Si el fichero no existe captuarmos la excepción y mostramo
         ↪  s un aviso por pantalla.
        print('El fichero', f, 'no existe.')
    else:
        # Leemos las líneas del fichero en una lista.
        lineas = f.read().split('\n')
        # Cerramos el fichero.
        f.close()
        # Creamos un diccionario vacío para guardar las líneas de tr
         ↪  enes.
        trenes = {}
        # Bucle iterativo para recorrer las líneas del fichero. i to
         ↪  ma como valores cada línea menos la primera que es el en
         ↪  cabezado.
        for i in lineas[1:]:
            # Creamos una lista con los campos dividiendo la línea p
             ↪  or la coma.
            campos = i.split(',')
            # Añadimos al diccionario un nuevo par con la clave el i
             ↪  dentificador del tren y valor la lista que se obtien
             ↪  e de dividir el último campo por el guión.
            trenes[campos[0]] = campos[2].split('-')
        return trenes

def exportar_trenes(trenes, fichero):
    '''Función que crea un fichero csv separado por ; con los identi
     ↪  ficadores de las líneas de tren, su origen y su destino a pa
     ↪  rtir de un diccionario.

    Parámetros:
        - trenes: Es un diccionario cuyas claves son los identificad
 ↪  ores de los trenes y cuyos valores son listas de dos elementos c
 ↪  on el origen y el destino.
        - fichero: Es una cadena con el nombre del fichero.
    '''
    # Definimos una cadena con el encabezado del fichero.
    lineas = 'id;origen;destino'
    # Bucle iterativo para recorrer el diccionario. id recorre las c
     ↪  laves y estaciones recorre los valores.
    for id, estaciones in trenes.items():
        # Añadimos a la cadena una nueva línea con el identificador
```

```
{'10T0001C1': ['Principe Pio', 'Aeropuerto'], '10T0002C1':
 ↪  ['Aeropuerto', 'Principe Pio'], '10T0003C10': ['Villalba',
 ↪  'Aeropuerto'], '10T0004C10': ['Aeropuerto', 'Villalba'],
 ↪  '10T0005C2': ['Guadalajara', 'Chamartin'], '10T0006C2':
 ↪  ['Chamartin', 'Guadalajara'], '10T0007C3': ['Chamartin',
 ↪  'Aranjuez'], '10T0008C3': ['Aranjuez', 'Chamartin '],
 ↪  '10T0009C3a': ['Chamartin', 'El Escorial'], '10T0010C3a': ['El
 ↪  Escorial', 'Chamartin'], '10T0011C4': ['Chamartin', 'Parla'],
 ↪  '10T0012C4': ['Parla', 'Chamartin'], '10T0013C4a':
 ↪  ['Alcobendas', 'Parla'], '10T0014C4a': ['Parla', 'Alcobendas'],
 ↪  '10T0015C4b': ['Colmenar Viejo', 'Parla'], '10T0016C4b':
 ↪  ['Parla', 'Colmenar Viejo'], '10T0017C5': ['Mostoles el Soto',
 ↪  'Humanes'], '10T0018C5': ['Humanes', 'Mostoles el Soto'],
 ↪  '10T0019C7': ['Alcala de Henares', 'Principe Pio'], '10T0020C7':
 ↪  ['Principe Pio', 'Alcala de Henares'], '10T0021C8': ['Atocha',
 ↪  'Cercedilla'], '10T0022C8': ['Cercedilla', 'Atocha'],
 ↪  '10T0023C3a': ['Atocha', 'El Escorial'], '10T0024C3a': ['El
 ↪  Escorial', 'Atocha'], '10T0025C9': ['Cercedilla', 'Cotos'],
 ↪  '10T0026C9': ['Cotos', 'Cercedilla'], '10T0027C3a': ['Atocha',
 ↪  'El Escorial'], '10T0028C3a': ['El Escorial', 'Atocha']}
```

Ver notebook de la solución en Colab

Ejercicio 8.6. Construir un programa que realize las siguientes operaciones con la librería Pandas:

1. Crear un DataFrame con las siguientes columnas:
   - Nombre: Juan, Marta, Pedro, Jorge, Blas, Lisa, Antonio
   - Edad: 23,78,22,19,45,33,20
   - Género: M, F, M, M, M, F, M
   - Provincia': Burgos, Madrid, Toledo, Burgos, Madrid, Toledo, Madrid
   - Hijos: 2,0,0,3,2,1,4
   - Mascotas: 5,1,0,5,2,2,3
2. Mostrar la información básica del DataFrame.
3. Obtener los principales estadísticos de las columnas numéricas.
4. Obtener los porcentajes de hombres y mujeres por provincias.
5. Representar, mediante un diagrama de dispersión, en número de hijos frente al número de mascotas para las personas de Madrid.
6. Realizar la siguiente gráfica.

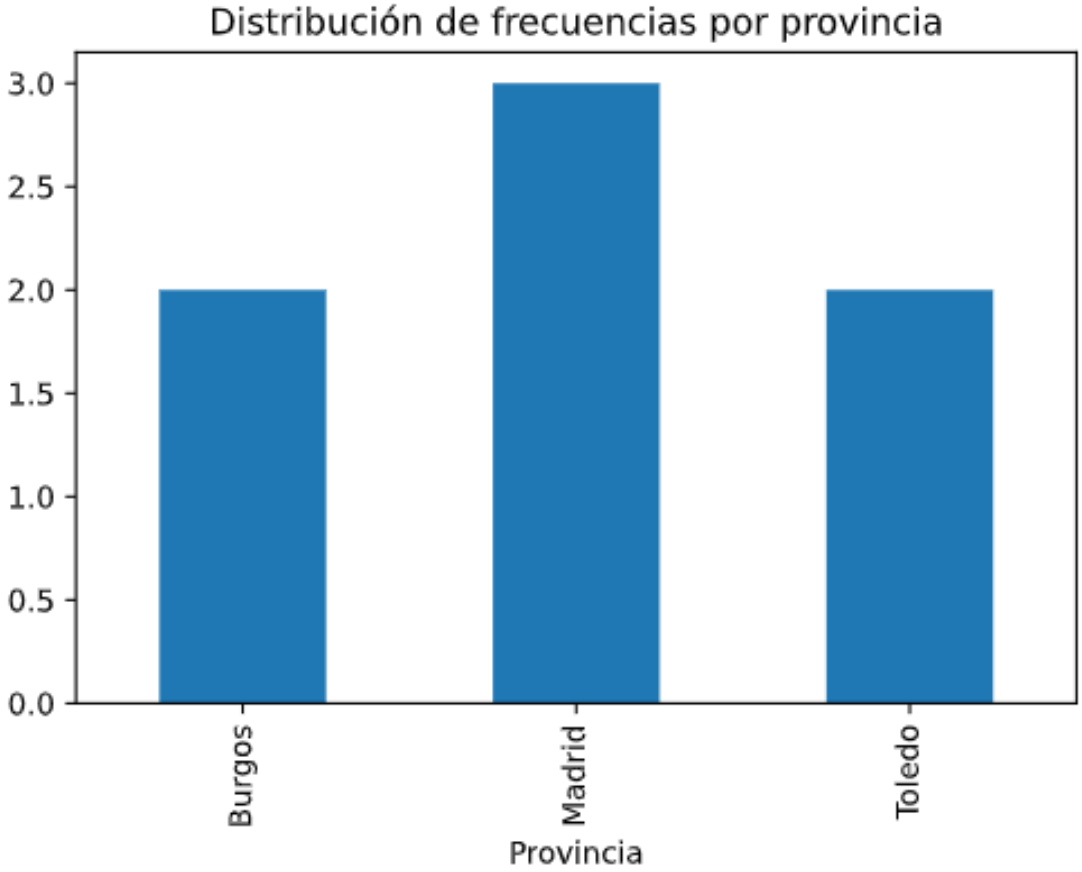

Figura 8.1: 'Recta de regresión'

Solución

```
import pandas as pd
import matplotlib.pyplot as plt

# Creamos un diccionario con claves los nombres de las columnas y va
 ↪  lores las listas con los elementos de las columnas.
datos = {
  'Nombre': ['Juan', 'Marta', 'Pedro', 'Jorge', 'Blas', 'Lisa', 'Ant
   ↪  onio'],
  'Edad': [23, 78, 22, 19, 45, 33, 20],
  'Género': ['M', 'F', 'M', 'M', 'M', 'F', 'M'],
  'Provincia': ['Burgos', 'Madrid', 'Toledo', 'Burgos', 'Madrid', 'T
   ↪  oledo', 'Madrid'],
  'Hijos': [2, 0, 0, 3, 2, 1, 4],
  'Mascotas': [5, 1, 0, 5, 2, 2, 3]
}

df = pd.DataFrame(datos)
print(df)
```

```
    Nombre  Edad Género Provincia  Hijos  Mascotas
0     Juan    23      M    Burgos      2         5
1    Marta    78      F    Madrid      0         1
2    Pedro    22      M    Toledo      0         0
3    Jorge    19      M    Burgos      3         5
4     Blas    45      M    Madrid      2         2
5     Lisa    33      F    Toledo      1         2
6  Antonio    20      M    Madrid      4         3
```

```
# Información básica del DataFrame
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Nombre     7 non-null      object
 1   Edad       7 non-null      int64
 2   Género     7 non-null      object
 3   Provincia  7 non-null      object
```

```
 4   Hijos      7 non-null      int64
 5   Mascotas   7 non-null      int64
dtypes: int64(3), object(3)
memory usage: 464.0+ bytes
None
```

```
# Principales estadísticos de las columnas numéricas
print(df.describe())
```

```
            Edad     Hijos  Mascotas
count   7.000000  7.000000  7.000000
mean   34.285714  1.714286  2.571429
std    21.383126  1.496026  1.902379
min    19.000000  0.000000  0.000000
25%    21.000000  0.500000  1.500000
50%    23.000000  2.000000  2.000000
75%    39.000000  2.500000  4.000000
max    78.000000  4.000000  5.000000
```

```
# Porcentaje de hombres y mujeres por provincias
print(df.groupby('Provincia').Género.value_counts(normalize = True)
 ↪  * 100)
```

```
Provincia  Género
Burgos     M         100.000000
Madrid     M          66.666667
           F          33.333333
Toledo     F          50.000000
           M          50.000000
Name: Género, dtype: float64
```

```
# Diagrama de dispersión del número de hijos y el número de mascotas
 ↪  de Madrid.
# Inicializamos la figura y los ejes.
fig, ax = plt.subplots()
# Filtramos el data frame para quedarnos con las filas de Madrid y d
 ↪  ibujamos el diagrama de dispersión.
df[df.Provincia == 'Madrid'].plot(kind = 'scatter', x = 'Hijos', y =
 ↪  'Mascotas', ax = ax)
# Mostramos el gráfico por pantalla.
plt.show()
```

```
<Figure size 432x288 with 1 Axes>
```

```
# Diagrama de barras de las frecuencias por provincia.
# Inicializamos la figura y los ejes.
fig, ax = plt.subplots()
# Agrupamos por provincia y contamos el número de elementos de cada
 ↪  grupo. A partir de la serie con las frecuencias dibujamos el dia
 ↪  grama de barras.
df.groupby('Provincia').size().plot(kind = 'bar')
# Añadimos un título.
plt.title('Distribución de frecuencias por provincia')
# Mostramos el gráfico.
plt.show()
```

```
<Figure size 432x288 with 1 Axes>
```

Ver notebook de la solución en Colab

9 2022-03-24 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 24 de marzo de 2022

Ejercicio 9.1. Escriba un programa que permita practicar una variante simplificada de la prueba de cálculo mental *La calculadora humana* del concurso televisivo Saber y ganar. El usuario debe ir sumando todos los números de la lista de uno en uno hasta que se equivoque o termine la lista, en cuyo caso ganará.

Ejemplo usando la lista [50, 4, 28, 33, 12]:

```
SUMAR Y GANAR
Vaya sumando todos los números que le iré diciendo. Empezamos por 0.
Más 50: 50
Más 4: 54
Más 28: 72
Te has equivocado, pero has acertado 2 veces seguidas.


SUMAR Y GANAR
Vaya sumando todos los números que le iré diciendo. Empezamos por 0.
Más 50: 50
Más 4: 54
Más 28: 82
Más 33: 115
Más 12: 127
Enhorabuena, GANASTE.
```

Solución

```
# Definimos la lista de números
numeros = [50, 4, 28, 33, 12]
# Definimos un acumulador para llevar la suma
suma = 0
print("SUMAR Y GANAR")
print("Vaya sumando todos los números que le iré diciendo. Empezamos
 ↪  por 0.")
# Bucle iterativo para recorrer la lista de números
for i in range(len(numeros)):
    # Preguntamos al usuario por la siguiente suma
    respuesta = int(input("Más " + str(numeros[i]) + ": "))
    # Añadimos el siguiente número a la suma
    suma += numeros[i]
    # Si la respuesta del usuario no coincide con la suma, le avisam
     ↪  os y salimos del bucle.
    if respuesta != suma:
        print("Te has equivocado, pero has acertado", i, "veces segu
         ↪  idas.")
        break
# Si el acumulador suma contiene la suma de todos los números entonc
 ↪  es el usuario ha ganado.
if suma == sum(numeros):
    print("Enhorabuena, GANASTE.")
```

```
SUMAR Y GANAR
Vaya sumando todos los números que le iré diciendo. Empezamos por 0.
Enhorabuena, GANASTE.
```

Ver notebook de la solución en Colab

Ejercicio 9.2. Escribir un programa que calcule a partir de una fecha de un año no bisiesto el número de días que han transcurrido en ese año y el número de meses lunares completos que abarcan. Se recuerda que un mes lunar dura aproximadamente 29,53 días.

El programa debe usar diccionarios para acceder al número de días de cada mes.

Ejemplo:

```
CALENDARIO LUNAR
Este programa convierte una fecha de un año NO bisiesto a un calendario
 ↪  lunar.
Indique el día: 1
```

```
Indique el mes: henero
El mes henero no existe.

CALENDARIO LUNAR
Este programa convierte una fecha de un año NO bisiesto a un calendario
 ↪  lunar
Indique el día: -5
Indique el mes: marzo
El día -5 de marzo no existe.

CALENDARIO LUNAR
Este programa convierte una fecha de un año NO bisiesto a un calendario
 ↪  lunar
Indique el día: 29
Indique el mes: febrero
El día 29 de febrero no existe.

CALENDARIO LUNAR
Este programa convierte una fecha de un año NO bisiesto a un calendario
 ↪  lunar
Indique el día: 1
Indique el mes: febrero
El día 1 de febrero es el día 32 del año.
Habrán pasado 1.08 meses lunares.
```

Solución

```
# Definimos una lista con los meses del año en orden
meses = ["enero", "febrero", "marzo", "abril", "mayo", "junio", "jul
 ↪  io", "agosto", "septiembre", "octubre", "noviembre", "diciembre
 ↪  "]
# Definimos un diccionario con los días de cada mes
dias = {"enero":31, "febrero":28, "marzo":31, "abril":30, "mayo":31,
 ↪  "junio":30, "julio":31, "agosto":31, "septiembre":30, "octubre
 ↪  ":31, "noviembre":30, "diciembre":31}
# Definimos el número de días de un mes lunar
meslunar = 29.53
print("CALENDARIO LUNAR")
print("Este programa convierte una fecha de un año NO bisiesto a un
 ↪  calendario lunar.")
# Preguntamos al usuario por el día
dia = int(input("Indique el día: "))
# Preguntamos al usuario por el mes
mes = input("Indique el mes: ")
# Si el mes no está en la lista de meses, avisamos al usuario y term
 ↪  inamos la ejecución.
if mes not in meses:
    print("El mes", mes, "no existe.")
# Si el día no es un día válido del mes introducido, avisamos al usu
 ↪  ario y terminamos la ejecución.
elif dia < 1 or dia > dias[mes]:
    print("El día", dia, "de", mes, "no existe.")
# En caso contrario, la fecha introducida es válida y procedemos al
 ↪  cálculo de días transcurridos
else:
    # Definimos un acumulador para el número de días transcurridos
    numdias = 0
    # Bucle iterativo para recorrer la lista d emeses por valor
    for i in meses:
        # Si el mes i no coincide con el mes introducido, añadimos e
         ↪  l número de días del mes i a los días transcurridos
        if i != mes:
            numdias += dias[i]
        # Si el mes i coincide con el mes introducido, añadimos el d
         ↪  ía introducido a los días transcurridos y salimos del bu
         ↪  cle
        else:
            numdias += dia
            break
    # Mostramos por pantalla el número de días transcurridos hasta l
     ↪  a fecha introducida
    print("El día", dia, "de", mes, "es el día", numdias, "del año.
     ↪  ")
    # Mostramos por pantalla el número de meses lunares transcurrido
     ↪  s hasta la fecha introducida
    print("Habrán pasado", round(numdias / meslunar, 2), "meses luna
     ↪  res.")
```

```
CALENDARIO LUNAR
Este programa convierte una fecha de un año NO bisiesto a un
 ↪  calendario lunar.
El día 1 de febrero es el día 32 del año.
Habrán pasado 1.08 meses lunares.
```

Ver notebook de la solución en Colab

Ejercicio 9.3. Un cine tiene una sala rectangular con 5 filas y 4 columnas de butacas. Escribir un programa que permita gestionar la reserva de butacas con los siguientes requisitos:

- El programa mostrará una matriz con el carácter X para las butacas libres y O para las ocupadas y preguntará por la fila y columna de la butaca a reservar.
- Si el usuario introduce una fila o una columna no válidas, se le avisará y se le volverá a preguntar.
- Si la fila y la columna introducida corresponde a una butaca ocupada, se avisará al usuario y se le volverá a preguntar.
- Si la butaca está libre se reservará y se volverá a preguntar al usuario si quiere hacer más reservas.
- El programa terminará cuando el usuario no quiera hacer más reservas.

Ejemplo:

```
RESERVA DE BUCATAS
XXXXX
XXXXX
XXXXX
XXXXX
Introduce la fila que quieres: 2
Introduce la columna que quieres: 3
Reserva realizada.
¿Desea realizar otra reserva? (S/N): S

RESERVA DE BUCATAS
XXXXX
XXOXX
XXXXX
XXXXX
```

```
Introduce la fila que quieres: 5
Introduce la columna que quieres: 1
La fila y columna elegidas no son válidas.
¿Desea realizar otra reserva? (S/N): S

RESERVA DE BUCATAS
XXXXX
XXOXX
XXXXX
XXXXX
Introduce la fila que quieres: 2
Introduce la columna que quieres: 3
La butaca elegida está ocupada.
¿Desea realizar otra reserva?
(S/N): S

RESERVA DE BUCATAS
XXXXX
XXOXX
XXXXX
XXXXX
Introduce la fila que quieres: 1
Introduce la columna que quieres: 2
Reserva realizada.
¿Desea realizar otra reserva? (S/N): N
¡Gracias!
```

Solución

Solución 1

```python
# Número de filas
nfil = 4
# Número de columnas
ncol = 5
# Creamos una lista para representar las filas del cine
cine = []
for i in range(nfil):
    # Representamos cada fila con una cadena de "X" del tamaño del n
    ↪ úmero de columnas
    cine.append("X" * ncol)
# Bucle para preguntar al usuario por la reserva
while True:
    print("\nRESERVA DE BUTACAS")
    # Bucle para recorrer las filas del cine y mostrarlas por pantal
    ↪ la
    for i in cine:
        print(i)
    # Preguntamos al usuario por la fila que quiere
    fila = int(input("Introduce la fila que quieres: "))
    # Preguntamos al usuario por la columna que quiere
    columna = int(input("Introduce la columna que quieres: "))
    # Si el número de fila o columna son mayores de los disponibles
    ↪ avisamos al usuario
    if fila > nfil or columna > ncol:
        print("La fila y columna elegidas no son válidas.")
    # Si la fila y la columna que quiere el usuario está ocupada, le
    ↪ avisamos
    elif cine[fila-1][columna-1] == "O":
        print("La butaca elegida está ocupada.")
    # En caso contrario, reservamos la butaca escribiendo un "O" en
    ↪ la posición de la fila y la columna
    else:
        cine[fila-1] = cine[fila-1][:columna-1] + "O" +
↪   cine[fila-1][columna:]
        print("Reserva realizada.")
    # Preguntamos al usuario si quiere hacer otra reserva y si no sa
    ↪ limos del bucle
    if input("¿Desea realizar otra reserva? (S/N): ") != "S":
        break
print("¡Gracias!")
```

```
RESERVA DE BUTACAS
XXXXX
XXXXX
XXXXX
XXXXX
Reserva realizada.

RESERVA DE BUTACAS
XXXXX
XXXXX
XOXXX
XXXXX
La fila y columna elegidas no son válidas.

RESERVA DE BUTACAS
XXXXX
XXXXX
XOXXX
XXXXX
La butaca elegida está ocupada.

RESERVA DE BUTACAS
XXXXX
XXXXX
XOXXX
XXXXX
Reserva realizada.
¡Gracias!
```

Solución 2

```
# Número de filas
nfil = 4
# Número de columnas
ncol = 5
# Creamos una lista para las filas del cine
cine = []
# Bucle iterativo para recorrer las filas del cine
for i in range(nfil):
    # Creamos una lista para las butacas de una fila
    fila = []
    for j in range(ncol):
        # Ponemos el caracter X en la fila i y la columna j del cine
        fila.append("X")
    # Añadimos la fila al cine
    cine.append(fila)
# Bucle para preguntar al usuario por la reserva
while True:
    print("\nRESERVA DE BUTACAS")
    # Bucle para recorrer las filas del cine y mostrarlas por pantal
    ↪ la
    for i in cine:
        #Unimos todas las butacas de la fila i en una cadena y la mo
        ↪ stramos por pantalla
        print("".join(i))
    # Preguntamos al usuario por la fila que quiere
    fila = int(input("Introduce la fila que quieres: "))
    # Preguntamos al usuario por la columna que quiere
    columna = int(input("Introduce la columna que quieres: "))
    # Si el número de fila o columna son mayores de los disponibles
    ↪ avisamos al usuario
    if fila > nfil or columna > ncol:
        print("La fila y columna elegidas no son válidas.")
    # Si la fila y la columna que quiere el usuario está ocupada, le
    ↪ avisamos
    elif cine[fila-1][columna-1] == "O":
        print("La butaca elegida está ocupada.")
    # En caso contrario, reservamos la butaca escribiendo un "O" en
    ↪ la posición de la fila y la columna
    else:
        cine[fila-1][columna-1] = "O"
        print("Reserva realizada.")
    # Preguntamos al usuario si quiere hacer otra reserva y si no sa
    ↪ limos del bucle
    if input("¿Desea realizar otra reserva? (S/N): ") != "S":
        break
print("¡Gracias!")
```

```
RESERVA DE BUTACAS
XXXXX
XXXXX
XXXXX
XXXXX
Reserva realizada.

RESERVA DE BUTACAS
XXXXX
XXXXX
XOXXX
XXXXX
La butaca elegida está ocupada.

RESERVA DE BUTACAS
XXXXX
XXXXX
XOXXX
XXXXX
Reserva realizada.
¡Gracias!
```

Ver notebook de la solución en Colab

10 2022-06-04 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 4 de junio de 2022

Ejercicio 10.1. Realizar un programa que pregunte al usuario por un número entero impar y dibuje un rombo con el número de filas introducidas por el usuario.

Ejemplo de ejecución

```
Introduce el número de filas del rombo (un número impar): 7
   *
  ***
 *****
*******
 *****
  ***
   *
```

Solución

Solución 1

```
# Preguntamos el número de filas del rombo
n = int(input("Introduce el número de filas (un número impar): "))
# Otenemos el cociente de la división entera del número de filas del
 ↪  rombo por 2
m = n // 2
# Bucle iterativo para recorrer por posición las filas de la mitad s
 ↪  uperior del rombo. i toma como valores los valores de la secuenc
 ↪  ia de enteros desde 1 hasta la mitad de las filas del rombo.
for i in range(1, m + 1):
    # Mostramos por pantalla la cadena que contiene m-i espacios y 2
     ↪  *i-1 asteriscos. A medida que i se va incrementando, los esp
     ↪  acios disminuyen y el número de asteriscos de cada línea aum
     ↪  enta.
    print(" " * (m - i), "*" * (2 * i - 1))
# Mostramos por pantalla la línea central del rombo que tendrá tanto
 ↪  s asteriscos como el número de filas del rombo.
print("*" * n)
# Bucle iterativo para recorrer por posición las filas de la mitad i
 ↪  nferior del rombo. i toma como valores los valores de la secuenc
 ↪  ia de enteros desde m hasta 1.
for i in range(m, 0, -1):
    # Mostramos por pantalla la cadena que contiene m-i espacios y 2
     ↪  *i-1 asteriscos. A medida que i se va decrementando, los esp
     ↪  acios aumentan y el número de asteriscos de cada línea dismi
     ↪  nuye.
    print(" " * (m - i), "*" * (2 * i - 1))
```

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

Solución 2

```
# Preguntamos el número de filas del rombo
n = int(input("Introduce el número de filas (un número impar): "))
# Otenemos el cociente de la división entera del número de filas del
 ↪  rombo por 2
m = n // 2
# Bucle iterativo para recorrer por posición las filas de la mitad s
 ↪  uperior del rombo. i toma como valores los valores de la secuenc
 ↪  ia de enteros desde 1 hasta la mitad de las filas del rombo.
for i in range(m):
    # Bucle iterativo para mostrar los espacios de cada línea de la
     ↪  mitad superior del rombo. A medida que i se va incrementando
     ↪  , los espacios disminuyen.
    for j in range(m - i):
        print(" ", end = "")
    # Bucle iterativo para mostrar los asteriscos de cada línea de l
     ↪  a mitad superior del rombo. A medida que i se va incrementan
     ↪  do, los asteriscos aumentan.
    for j in range(2 * i + 1):
        print("*", end = "")
    # Cambiamos de línea
    print("")
# Mostramos por pantalla la línea central del rombo que tendrá tanto
 ↪  s asteriscos como el número de filas del rombo.
print("*" * n)
# Bucle iterativo para recorrer por posición las filas de la mitad i
 ↪  nferior del rombo. i toma como valores los valores de la secuenc
 ↪  ia de enteros desde m hasta 1.
for i in range(m, 0, -1):
    # Bucle iterativo para mostrar los espacios de cada línea de la
     ↪  mitad inferior del rombo. A medida que i se va decrementando
     ↪  , los espacios aumentan.
    for j in range(m - i + 1):
        print(" ", end = "")
    # Bucle iterativo para mostrar los asteriscos de cada línea de l
     ↪  a mitad inferior del rombo. A medida que i se va decrementan
     ↪  do, los asteriscos disminuyen.
    for j in range(2 * i - 1):
        print("*", end = "")
    # Cambiamos de línea
    print("")
```

```
   *
```

```
   ***
  *****
*******
  *****
   ***
    *
```

Ver notebook de la solución en Colab

Ejercicio 10.2. Realizar un programa que simule la operativa de una cuenta bancaria. El programa debe preguntar al usuario si desea realizar un ingreso o un reintegro hasta que el usuario decida terminar el programa.

Si el usuario selecciona la opción de ingreso, debe preguntarle por la cantidad a ingresar e incrementar el saldo en esa cantidad.

Si el usuario selecciona la opción de reintegro deberá preguntarle por la cantidad a sacar y sustraerla del saldo, siempre y cuando haya saldo suficiente. Si no hubiese saldo suficiente avisará al usuario y no realizará el reintegro.

Las cantidades de las operaciones deben guardarse en una sola lista (positivos para ingresos y negativos para reintegros), y después de cada operación debe mostrarse el saldo por pantalla.

Cuando el usuario ya no quiera hacer más operaciones, a partir de la lista de operaciones se deben crear dos listas más, una para los ingresos y otra para los reintegros y deben mostrarse por pantalla.

Ejemplo de ejecución

```
OPERATIVA CUENTA BANCARIA

¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 1
Introduce la cantidad: 1000
Saldo:  1000.0

¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 2
Introduce la cantidad: 1200
Lo siento, no hay saldo suficiente.
```

```
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 2
Introduce la cantidad: 800
Saldo:  200.0

¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 1
Introduce la cantidad: 500
Saldo:  700.0

¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 2
Introduce la cantidad: 400
Saldo:  300.0

¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Elige una opción (cualquier otra para terminar): 3

Ingresos:  [1000.0, 500.0]
Reintegros: [-800.0, -400.0]
```

Solución

```
print("OPERATIVA CUENTA BANCARIA")
# Inicializamos a 0 una variable real para guardar el saldo.
saldo = 0
# Inicializamos una lista vacía para guardar las operaciones bancari
 ↪  as.
operaciones = []
# Bucle infinito
while True:
    print("¿Qué deseas hacer?")
    print("1- Ingreso")
    print("2- Reintegro")
    # Preguntamos al usuario por la operación que quiere realizar.
    opcion = input("Elige una opción (cualquier otra para terminar):
 ↪  ")
    # Condicional para ver qué operacion ha seleccionado el usuario.
    if opcion == "1":  # Si el usuario elige la opción de realizar u
     ↪  n ingreso.
        # Preguntamos al usuario por la cantidad a ingresar y la con
         ↪  vertimos en un número real.
        cantidad = float(input("Introduce la cantidad: "))
        # Añadimos la cantidad a la lista de operaciones.
        operaciones.append(cantidad)
        # Sumamos al saldo en la cantidad del ingreso.
        saldo += cantidad
        # Mostramos el saldo por pantalla.
        print("Saldo: ", saldo)
    elif opcion == "2":  # Si el usuario elige la opción de realizar
     ↪  un reintegro.
        # Preguntamos al usuario por la cantidad a reintegrar y la c
         ↪  onvertimos en un número real.
        cantidad = float(input("Introduce la cantidad: "))
        # Condicional para ver si el hay saldo suficiente para reali
         ↪  zar el reintegro.
        if saldo < cantidad:  # No hay saldo suficiente.
            # Mostramos por pantalla un mensaje informando al usuari
             ↪  o de que no hay saldo.
            print("Lo siento, no hay saldo suficiente.")
        else:  # Hay saldo suficiente.
            # Añadimos la cantidad en negativo a la lista de operaci
             ↪  ones.
            operaciones.append(-cantidad)
            # Restamos del saldo la cantidad del ingreso.
            saldo -= cantidad
            # Mostramos el saldo por pantalla.
            print("Saldo: ", saldo)
    else:  # Si el usuario introduce cualquier otro número el bucle
     ↪  termina
        break
# Creamos dos listas vacías, una para los ingresos y otra para los r
 ↪  eintegros.
ingresos = []
reintegros = []
# Bucle iterativo para recorrer por valor los elementos de la lista
 ↪  de operaciones bancarias.
for i in operaciones:
    # Condicional para ver si la operación es un ingreso o un reinte
```

```
OPERATIVA CUENTA BANCARIA
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Saldo:  1000.0
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Lo siento, no hay saldo suficiente.
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Saldo:  200.0
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Saldo:  700.0
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Saldo:  300.0
¿Qué deseas hacer?
1- Ingreso
2- Reintegro
Ingresos:  [1000.0, 500.0]
Reintegros: [-800.0, -400.0]
```

Ver notebook de la solución en Colab

Ejercicio 10.3. Existe un texto que se usa frecuentemente para demostraciones dentro del ámbito del diseño gráfico. De hecho, este texto cuyo nombre es Lorem ipsum es un estándar utilizado desde el año 1500 y sirve principalmente para probar el diseño visual antes de insertar el texto definitivo. Dicho pasaje es el siguiente:

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Realizar un programa que analice dicho pasaje y realice las siguientes operaciones:

1. Crear un diccionario con la frecuencia de cada carácter sin tener en cuenta el espacio.

2. Muestre por pantalla el carácter que más se repite.
3. Muestre por pantalla las palabras que contienen el carácter más repetido y su frecuencia por palabra.

Ejemplo de ejecución

```
Carácter más repetido: i
Las palabras donde se encuentra el carácter que más se repite:
{'ipsum': 1, 'sit': 1, 'adipiscing': 3, 'elit,': 1, 'eiusmod': 1, 'incid
 ↪  idunt': 3, 'aliqua.': 1, 'enim': 1, 'minim': 2, 'veniam,': 1, 'quis'
 ↪  : 1, 'exercitation': 2, 'laboris': 1, 'nisi': 2, 'aliquip': 2, 'Duis
 ↪  ': 1, 'irure': 1, 'in': 1, 'reprehenderit': 1, 'velit': 1, 'cillum':
 ↪  1, 'fugiat': 1, 'pariatur.': 1, 'sint': 1, 'cupidatat': 1, 'proident
 ↪  ,': 1, 'qui': 1, 'officia': 2, 'mollit': 1, 'anim': 1, 'id': 1}
```

Solución

```
# Definimos una variable con la cadena a analizar.
texto = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, se
 ↪  d do eiusmod tempor incididunt ut labore et dolore magna aliqua.
 ↪  Ut enim ad minim veniam, quis nostrud exercitation ullamco labor
 ↪  is nisi ut aliquip ex ea commodo consequat. Duis aute irure dolo
 ↪  r in reprehenderit in voluptate velit esse cillum dolore eu fugi
 ↪  at nulla pariatur. Excepteur sint occaecat cupidatat non proiden
 ↪  t, sunt in culpa qui officia deserunt mollit anim id est laborum
 ↪  ."
# Definimos un diccionario vacío para ir guardando la frecuencia de
 ↪  cada carácter.
frecuencias = {}
# Bucle iterativo para recorrer por valor los caracteres de la frase
 ↪  una vez eliminados los espacios. i toma como valores los caracte
 ↪  res de la frase.
for i in texto.replace(" ", ""):
    # Condicional para ver si el carácter está ya como clave en el d
     ↪  iccionario.
    if i in frecuencias:  # El carácter es ya una clave del dicciona
     ↪  rio.
        # Incrementamos la frecuencia del carácter en 1.
        frecuencias[i] += 1
    else:  # El carácter no es una clave del diccionario.
        # Añadimos al diccionario un nuevo par con clave el carácter
         ↪  y valor 1.
        frecuencias[i] = 1
# Mostramos por pantalla el diccionario.
print(frecuencias)
# Calculamos el máximo valor del diccionario.
frec_max = max(frecuencias.values())
# Obtenemos la posición del máximo en la lista de los valores del di
 ↪  ccionario.
pos_max = list(frecuencias.values()).index(frec_max)
# Obtenemos el caracter más repetido en la misma posición de la list
 ↪  a de claves.
caracter_max = list(frecuencias.keys())[pos_max]
# Mostramos por pantalla el carácter más repetido.
print("Carácter más repetido:", caracter_max)
# Definimos un diccionario vacío para añadir las palabras que contie
 ↪  nen el caracter más repetido y su frecuencia.
palabras = {}
# Creamos una lista de palabras dividiendo el texto por el espacio.
texto  = texto.split(" ")
# Bucle iterativo para recorrer por valor las palabras de la lista.
for i in texto:
    # Condicional para ver si la palabra contiene el carácter más re
     ↪  petido y no está aún en el diccionario.
    if caracter_max in i and not i in palabras:
        # Inicializamos a 0 un contador para la frecuencia del carác
         ↪  ter en la palabra.
        freq = 0
        # Bucle iterativo para recorrer por valor los caracteres de
         ↪  la palabra.
        for j in i:
            # Condicional para ver si el carácter coincide con el má
             ↪  s repetido.
```

```
{'L': 1, 'o': 29, 'r': 22, 'e': 37, 'm': 17, 'i': 42, 'p': 11, 's':
 ↪  18, 'u': 28, 'd': 18, 'l': 21, 't': 32, 'a': 29, ',': 4, 'c':
 ↪  16, 'n': 24, 'g': 3, 'b': 3, 'q': 5, '.': 4, 'U': 1, 'v': 3,
 ↪  'x': 3, 'D': 1, 'h': 1, 'f': 3, 'E': 1}
Carácter más repetido: i
{'ipsum': 1, 'sit': 1, 'adipiscing': 3, 'elit,': 1, 'eiusmod': 1,
 ↪  'incididunt': 3, 'aliqua.': 1, 'enim': 1, 'minim': 2, 'veniam,':
 ↪  1, 'quis': 1, 'exercitation': 2, 'laboris': 1, 'nisi': 2,
 ↪  'aliquip': 2, 'Duis': 1, 'irure': 1, 'in': 1, 'reprehenderit':
 ↪  1, 'velit': 1, 'cillum': 1, 'fugiat': 1, 'pariatur.': 1, 'sint':
 ↪  1, 'cupidatat': 1, 'proident,': 1, 'qui': 1, 'officia': 2,
 ↪  'mollit': 1, 'anim': 1, 'id': 1}
```

Ver notebook de la solución en Colab

Ejercicio 10.4. Escriba un programa que obtenga el máximo y el mínimo de una serie de datos proporcionados en Hexadecimal. Para ello se le pasará una lista de datos en formato hexadecimal, y tendrá que convertirlos primero a formato Binario, y a continuación a formato Decimal. Para ello, se pide lo siguiente:

1. Crear una función de conversión de formato hexadecimal a binario en base a la siguiente tabla:

Hexadecimal	0	1	2	3	4	5	6	7
Decimal	0000	0001	0010	0011	0100	0101	0110	0111

Hexadecimal	8	9	A	B	C	D	E	F
Decimal	1000	1001	1010	1011	1100	1101	1110	1111

Ejemplo de ejecución

```
Introduce un número hexadecimal: AA55
El número hexadecimal AA55 en binario es 1010101001010101
```

2. Crear una función de conversión de formato binario a decimal. El procedimiento se detalla a continuación:

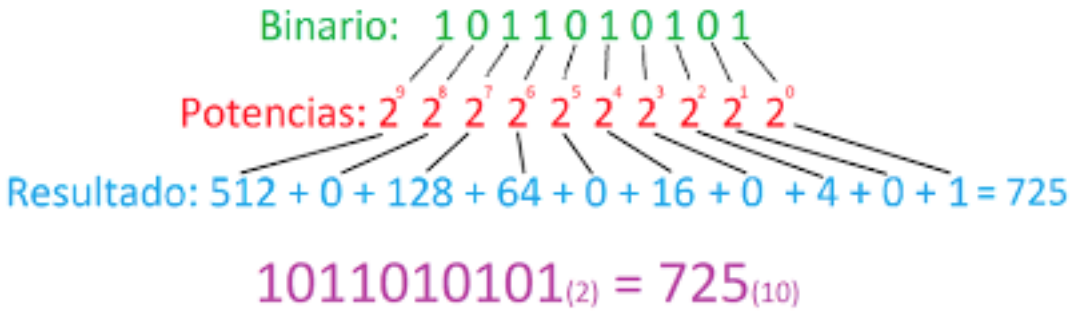

Figura 10.1: Conversión de binario a decimal

Ejemplo de ejecución `sh  Introduce un número binario: 10101010010101 01  El número binario 1010101001010101 es en decimal 43605`

3. Crear una función que reciba una lista de números hexadecimales y devuelva una tupla con el máximo y su valor decimal.

   Ejemplo de ejecución

   Usando como entrada de la última función la lista: `["AA55", "1010", "BEBE", " 0101", "0FEA"] sh  Máximo: ('BEBE', 48830)`

Solución

```python
def conversion_hex_bin(num_hex):
    """
    Función que convierte un número hexadecimal en binario.

    Parámetros:
        - num_hex: Es una cadena con un número hexadecimal.
    Salida:
        - Una cadena con el número binario correspondiente al número
 ↪  hexadecimal dado.
    """
    # Definimos un diccionario con el número binario asociado a cada
     ↪  dígito hexadecimal
    hexbin = {"0":"0000", "1":"0001", "2":"0010", "3":"0011", "4":"0
 ↪  100", "5":"0101", "6":"0110", "7":"0111", "8":"1000", "9":"1001
 ↪  ", "A":"1010", "B":"1011", "C":"1100", "D":"1101", "E":"1110", "
 ↪  F":"1111"}
    # Inicializamos la cadena para guardar el número binario a la ca
     ↪  dena vacía.
    num_bin = ""
    # Bucle iterativo para recorrer los dígitos del número hexadecim
     ↪  al por valor. i toma como valores cada uno de los caracteres
     ↪  que forman la cadena con el número hexadecimal.
    for i in num_hex:
        try:
            # Accedemos al diccionario mediante la clave correspondi
             ↪  ente al dígito hexadecimal y añadimos a la cadena co
             ↪  n el número binario los dígitos binarios correspondi
             ↪  entes al dígito hexadecimal.
            num_bin += hexbin[i]
        # Controlamos la excepción en caso de que alguno de los dígi
         ↪  tos del número hexadecimal no sea válido.
        except KeyError:
            return "El número introducido no es hexadecimal."
    # Devolvemos la cadena con el número binario.
    return num_bin

def conversion_bin_dec(num_bin):
    """
    Función que convierte un número binario en decimal.
    Parámetros:
        - num_bin: Es una cadena con un número binario.
    Salida:
        - El número decimal correspondiente al número binario dado.
    """
    # Convertimos la cadena con el número binario en una lista de ca
     ↪  racteres. Cada dígito del número binario es un elemento de l
     ↪  a lista.
    num_bin = list(num_bin)
    # Para recorrer la lista de derecha a izquierda la invertimos.
    num_bin.reverse()
    # Inicializamos a 0 una variable entera para ir guardando el núm
     ↪  ero decimal.
    num_dec = 0
    # Bucle iterativo para recorrer la lista de dígitos binarios por
     ↪  posición. i toma como valores los enteros de la secuencia de
     ↪  sde 0 al tamaño de la lista menos 1.
```

```
El número hexadecimal AA55 es 43605 en binario.
('BEBE', 48830)
```

Ver notebook de la solución en Colab

Ejercicio 10.5. El fichero coches.csv contiene información sobre los modelos de coches vendidos en USA un determinado año. Se pide:

1. Crear un DataFrame a partir del fichero anterior.
2. Eliminar las filas con valores desconocidos y mostrar el número de filas del dataframe resultante.
3. Crear una columna con el precio en euros (cambio 1$ = 0.94€)
4. Mostrar por pantalla las 10 últimas filas del DataFrame.
5. Mostrar por pantalla el número de marcas que contiene el DataFrame.
6. Mostrar por pantalla el número de modelos de cada marca que hay en el DataFrame, de mayor a menor frecuencia.
7. Mostrar por pantalla la marca y el modelo del coche más caro.
8. Mostrar por pantalla el precio medio en euros de los coches agrupando por marca y ordenando de menor a mayor precio.
9. Dibujar el diagrama de barras del porcentaje de modelos de cada marca.
10. Dibujar el diagrama de dispersión de la potencia y el precio.
    Los gráficos deben guardarse en una carpeta con el nombre gráficos y deben tener un título adecuado.

Solución

```python
import pandas as pd
import matplotlib.pyplot as plt

# Creamos el dataframe desde el fichero csv.
df = pd.read_csv("https://aprendeconalf.es/docencia/python/examenes/
 ↪  inteligencia-negocios/soluciones/examen-2022-06-04/coches.csv",
 ↪  sep = ";")
print(df)
```

```
     Marca                    Modelo   Tipo  Potencia   Precio
0    Acura                       MDX    SUV     265.0  33337.0
1    Acura           RSX Type S 2dr  Sedan     200.0  21761.0
2    Acura                  TSX 4dr  Sedan     200.0  24647.0
3    Acura                   TL 4dr  Sedan     270.0  30299.0
4    Acura                3.5 RL 4dr  Sedan     225.0  39014.0
..     ...                       ...    ...       ...      ...
423  Volvo  C70 LPT convertible 2dr  Sedan     197.0  38203.0
```

```
424  Volvo  C70 HPT convertible 2dr  Sedan     242.0  40083.0
425  Volvo               S80 T6 4dr  Sedan     268.0  42573.0
426  Volvo                    V40  Wagon     170.0  24641.0
427  Volvo                   XC70  Wagon     208.0  33112.0

[428 rows x 5 columns]
```

```python
# Eliminar las filas con valores desconocidos y mostrar el número de
 ↪  filas del dataframe resultante.
df.dropna(inplace = True)
filas, columnas = df.shape
print("Número de filas:", filas)
print("Número de columnas:", columnas)
```

```
Número de filas: 421
Número de columnas: 5
```

```python
# Crear una columna con el precio en euros (cambio 1$ = 0.94€)
df["Precio€"] = df.Precio * 0.94
# Mostrar las últimas 10 filas del dataframe
print(df.tail(10))
```

```
     Marca                   Modelo   Tipo  Potencia   Precio
      ↪  Precio€
418  Volvo              S60 2.5 4dr  Sedan     208.0  29916.0
 ↪  28121.04
419  Volvo               S60 T5 4dr  Sedan     247.0  32902.0
 ↪  30927.88
420  Volvo                S60 R 4dr  Sedan     300.0  35382.0
 ↪  33259.08
421  Volvo              S80 2.9 4dr  Sedan     208.0  35542.0
 ↪  33409.48
422  Volvo             S80 2.5T 4dr  Sedan     194.0  35688.0
 ↪  33546.72
423  Volvo  C70 LPT convertible 2dr  Sedan     197.0  38203.0
 ↪  35910.82
424  Volvo  C70 HPT convertible 2dr  Sedan     242.0  40083.0
 ↪  37678.02
425  Volvo               S80 T6 4dr  Sedan     268.0  42573.0
 ↪  40018.62
426  Volvo                    V40  Wagon     170.0  24641.0
 ↪  23162.54
```

```
427  Volvo                    XC70  Wagon     208.0  33112.0
 ↪  31125.28
```

```
# Mostrar el número de marcas
print("Número de marcas:", len(df.Marca.unique()))
```

```
Número de marcas: 38
```

```
# Mostrar una tabla con el número de coches de cada marca de mayor a
 ↪  menor frecuencia
print(df.Marca.value_counts(ascending = False))
```

```
Toyota           28
Chevrolet        27
Mercedes-Benz    26
Ford             23
BMW              19
Audi             19
Nissan           17
Honda            16
Volkswagen       15
Chrysler         14
Dodge            13
Mitsubishi       13
Volvo            12
Hyundai          12
Jaguar           12
Kia              11
Mazda            11
Pontiac          10
Subaru           10
Lexus            10
Lincoln           9
Buick             9
Mercury           9
Cadillac          8
Suzuki            8
Infiniti          8
GMC               8
Saturn            8
Saab              7
Porsche           7
```

```
Acura             7
Oldsmobile        3
Land Rover        3
MINI              2
Scion             2
Isuzu             2
Jeep              2
Hummer            1
Name: Marca, dtype: int64
```

```
# Mostrar la marca y modelo del coche más caro.
print(df[df.Precio == df.Precio.max()][["Marca", "Modelo"]])
```

```
       Marca       Modelo
334  Porsche  911 GT2 2dr
```

```
# Precio medio en euros de los coches por marcas
print(df.groupby("Marca")["Precio€"].mean().sort_values())
```

```
Marca
Scion            12135.400000
Kia              13996.941818
Suzuki           14937.775000
Hyundai          15073.213333
Saturn           15103.920000
MINI             15779.780000
Honda            18773.092500
Mazda            18980.907273
Toyota           19078.172857
Jeep             19856.560000
Oldsmobile       20444.060000
Mitsubishi       20524.683077
Ford             20635.820000
Pontiac          20878.528000
Subaru           21351.442000
Nissan           21617.622353
Chevrolet        22617.165926
Dodge            22710.472308
Isuzu            23141.860000
Chrysler         23778.105714
Mercury          24118.102222
GMC              24712.012500
```

```
Buick             26183.595556
Volkswagen        27901.706667
Infiniti          30907.200000
Volvo             32163.196667
Saab              33483.068571
Acura             36275.405714
Lincoln           36787.631111
Audi              36970.298947
BMW               37391.864211
Lexus             37501.864000
Land Rover        39339.940000
Hummer            43066.100000
Cadillac          43641.262500
Jaguar            52732.511667
Mercedes-Benz     53066.109231
Porsche           69243.085714
Name: Precio€, dtype: float64
```

```
# Diagrama de barras del porcentaje de modelos por marcas
df.Marca.value_counts(normalize = True).plot(kind = "barh", title =
 ↪  "Distribución del número de modelos por marca")
plt.show()
```

```
<Figure size 432x288 with 1 Axes>
```

```
# Diagrama de dispersión de la potencia y el precio
df.plot(x = "Potencia", y = "Precio€", kind = "scatter", title = "Di
 ↪  agrama de dispersión de la potencia frente al precio")
plt.show()
```

```
<Figure size 432x288 with 1 Axes>
```

Ver notebook de la solución en Colab

Ejercicio 10.6. El fichero bitcoin-tweets.csv contiene los tweets obtenidos en una búsqueda en Twitter sobre bitcoins (cada línea contiene información de un tweet). Se pide:

1. Crear una función que reciba la url de un fichero csv como el anterior y devuelva una lista con los tweets del fichero, donde cada tweet se represente con un diccionario con sus campos, tal y como se muestra en la siguiente salida:

```
[{'fecha': '2022-05-20 15:11:03+00:00', 'usuario': 'infobaeeconomia
 ↪  ', 'texto': 'Bill Gates contra el Bitcoin: "No aporta nada" htt
 ↪  ps://t.co/pZBlC664gw', 'likes': '496'},
{'fecha': '2022-05-21 14:26:51+00:00', 'usuario': 'DaniaGonzalz', '
 ↪  texto': 'El Salvador hoy es un referente en materia de Inclusió
 ↪  n Financiera gracias a la Ley #Bitcoin, países de la región c…
 ↪  https://t.co/P8oS2nfm36', 'likes': '554'},
{'fecha': '2022-05-20 20:42:04+00:00', 'usuario': 'LaHuellaSV', 'te
 ↪  xto': 'El asesor en criptoeconomía y fundador de Criptokuántica
 ↪  , Néstor Kreimer, afirmó que El Salvador tiene actualmente… htt
 ↪  ps://t.co/HbSqrQYk67', 'likes': '57'},
{'fecha': '2022-05-20 11:38:19+00:00', 'usuario': 'carmenaidalazo',
 ↪  'texto': 'La ironía con bitcoin es que años de bajas tasas y po
 ↪  litica monetaria expansiva alimentaron incrementos en precios…
 ↪  https://t.co/cm2mo86eKy', 'likes': '284'},
{'fecha': '2022-05-20 11:56:30+00:00', 'usuario': 'garciabanchs', '
 ↪  texto': 'Cuando a soberbios del #Bitcoin aún les cambiaban los
 ↪  pañales, ya me quemaba las pestañas en mi doctorado de Econom…
 ↪  https://t.co/82tj5Dcalp', 'likes': '74'},
{'fecha': '2022-05-21 14:27:07+00:00', 'usuario': 'DaniaGonzalz', '
 ↪  texto': '"El Salvador es la capital del #Bitcoin #Ethereum , in
 ↪  vitamos a todos los países a invertir en el futuro, por eso nue
 ↪  stro p… https://t.co/UnQVQZj7aw"', 'likes': '89'},
{'fecha': '2022-05-14 00:07:00+00:00', 'usuario': 'ActualidadRT', '
 ↪  texto': "@ActualidadRT's account has been withheld in Belgium,
 ↪  Austria, Bulgaria, Sweden, Croatia, Spain, Slovenia, Cyprus, Sl
 ↪  ovakia, Czech Republic, Romania, Portugal, Poland, Denmark, Net
 ↪  herlands, Estonia, Malta, Luxembourg, Finland, France, Lithuani
 ↪  a, Germany, Greece, Latvia, Hungary, Italy, Ireland, United Kin
 ↪  gdom in response to a legal demand. Learn more.", 'likes': '103
 ↪  7'},
{'fecha': '2022-05-14 14:01:45+00:00', 'usuario': 'ChiguireBipolar'
 ↪  , 'texto': 'Testigo de Jehová se esconde de promotor del Bitcoi
 ↪  n https://t.co/odVOELmYKu https://t.co/GOtuWLoGhT', 'likes': '1
 ↪  082'}, {'fecha': '2022-05-15 01:21:58+00:00', 'usuario': 'el_pa
 ↪  is', 'texto': 'El Salvador convirtió al Bitcoin en su moneda le
 ↪  gal. Con la caída de las criptomonedas, han caído en picado tam
 ↪  bién… https://t.co/fFlk1P1LSj', 'likes': '551'},
{'fecha': '2022-05-13 23:00:28+00:00', 'usuario': 'garciabanchs', '
 ↪  texto': 'La cantidad de dinero que han perdido fans por hacerle
 ↪  caso a celebridades en torno a la adquisición de criptomoned… h
 ↪  ttps://t.co/UOjmvj4Jxq', 'likes': '1143'}, {'fecha': '2022-05-1
 ↪  4 15:44:24+00:00', 'usuario': 'juanrallo', 'texto': 'Llevas dic
 ↪  iendo que Bitcoin es una burbuja desde que cotizaba a 100 dólar
 ↪  es. Que sepamos los patrones típicos de un… https://t.co/2inCgv
 ↪  PHnE', 'likes': '1423'}, {'fecha': '2022-05-14 16:01:48+00:00',
 ↪  'usuario': 'wallstwolverine', 'texto': 'Los mismos políticos qu
 ↪  e asesoraban a Venezuela en sus políticas económicas te dan lec
 ↪  ciones hoy de lo peligroso qu… https://t.co/wRje03EydN', 'likes
 ↪  ': '1543'},
```

```
{'fecha': '2022-05-18 06:18:46+00:00', 'usuario': 'EdgardoMulatoSV'
↪  , 'texto': 'Ya se preguntaron, ¿Por qué sube la ola de homicidi
↪  os y ataques al Gobierno cuando vienen INVERSIONISTAS al país?…
↪  https://t.co/T5PCT8RgKB', 'likes': '796'}]
```

2. Crear una función que reciba una lista de tweets como la que devuelve la función anterior, y devuelva un diccionario con los hashtags contenidos en los tweets y su frecuencia, tal y como se muestra en la siguiente salida:

```
{'#Bitcoin': 3, '#Ethereum': 1}
```

Solución

```
def parsear_tweets(url):
    """
    Función que parsea un fichero csv con información de tweets orga
 ↪  nizada por campos y devuelve una lista de diccionarios donde cad
 ↪  a diccionario contiene los campos de un tweet.

    Parámetros:
        - url: Es una cadena con la url del fichero.
    Salida:
        Una lista de diccionarios donde cada diccionario contiene la
 ↪  información de los campos de un tweet.
    """
    from urllib import request
    from urllib.error import URLError
    # Abrimos el diccionario en modo lectura
    try:
        # Abrimos el diccionario en modo lectura
        with request.urlopen(url) as f:
             # Leemos el contenido del fichero y creamos una lista c
              ↪  on cada una de las líneas del contenido.
            lista_contenido = f.read().decode("utf-8").split("\n")
    except URLError:
        print("La url", url, "no existe.")
        return
    else:
        # Creamos una lista con los nombres de los campos que serán
         ↪  las claves de los diccionarios.
        lista_claves = lista_contenido.pop(0).split(";")
        # Creamos una lista vacía para guardar los diccionarios de l
         ↪  os tweets.
        lista_tweets = []
        # Bucle iterativo para recorrer la lista de tweets.
        for i in lista_contenido:
            # Creamos un diccionario vacío para guardar la informaci
             ↪  ón del tweet.
            dict_tweet = {}
            # Creamos la lista de los campos dividiendo la cadena co
             ↪  n la información del tweet por el separador punto y
             ↪  coma.
            lista_campos = i.split(";")
            # Bucle iterativo para recorrer la lista de campos
            for j in range(len(lista_campos)):
                # Añadimos al diccionario del tweet el par formado p
                 ↪  or el nombre del campo y la información correspo
                 ↪  ndiente.
                dict_tweet[lista_claves[j]] = lista_campos[j]
            # Añadimos el diccionario con la información del tweet a
             ↪  la lista de diccionarios.
            lista_tweets.append(dict_tweet)
        return lista_tweets

def hashtags(tweets):
    """
    Función que recibe una lista de diccionarios con información de
 ↪  tweets y devuelve una lista con los hashtags contenidos en los t
 ↪  weets.
```

```
[{'fecha': '2022-05-20 15:11:03+00:00', 'usuario':
 ↪  'infobaeeconomia', 'texto': 'Bill Gates contra el Bitcoin: "No
 ↪  aporta nada" https://t.co/pZBlC664gw', 'likes': '496'},
 ↪  {'fecha': '2022-05-21 14:26:51+00:00', 'usuario':
 ↪  'DaniaGonzalz', 'texto': 'El Salvador hoy es un referente en
 ↪  materia de Inclusión Financiera gracias a la Ley #Bitcoin ,
 ↪  países de la región c… https://t.co/P8oS2nfm36', 'likes':
 ↪  '554'}, {'fecha': '2022-05-20 20:42:04+00:00', 'usuario':
 ↪  'LaHuellaSV', 'texto': 'El asesor en criptoeconomía y fundador
 ↪  de Criptokuántica, Néstor Kreimer, afirmó que El Salvador tiene
 ↪  actualmente… https://t.co/HbSqrQYk67', 'likes': '57'}, {'fecha':
 ↪  '2022-05-20 11:38:19+00:00', 'usuario': 'carmenaidalazo',
 ↪  'texto': 'La ironía con bitcoin es que años de bajas tasas y
 ↪  politica monetaria expansiva alimentaron incrementos en precios…
 ↪  https://t.co/cm2mo86eKy', 'likes': '284'}, {'fecha': '2022-05-20
 ↪  11:56:30+00:00', 'usuario': 'garciabanchs', 'texto': 'Cuando a
 ↪  soberbios del #Bitcoin aún les cambiaban los pañales, ya me
 ↪  quemaba las pestañas en mi doctorado de Econom…
 ↪  https://t.co/82tj5Dcalp', 'likes': '74'}, {'fecha': '2022-05-21
 ↪  14:27:07+00:00', 'usuario': 'DaniaGonzalz', 'texto': '"El
 ↪  Salvador es la capital del #Bitcoin #Ethereum , invitamos a
 ↪  todos los países a invertir en el futuro, por eso nuestro p…
 ↪  https://t.co/UnQVQZj7aw"', 'likes': '89'}, {'fecha': '2022-05-14
 ↪  00:07:00+00:00', 'usuario': 'ActualidadRT', 'texto':
 ↪  "@ActualidadRT's account has been withheld in Belgium, Austria,
 ↪  Bulgaria, Sweden, Croatia, Spain, Slovenia, Cyprus, Slovakia,
 ↪  Czech Republic, Romania, Portugal, Poland, Denmark, Netherlands,
 ↪  Estonia, Malta, Luxembourg, Finland, France, Lithuania, Germany,
 ↪  Greece, Latvia, Hungary, Italy, Ireland, United Kingdom in
 ↪  response to a legal demand. Learn more.", 'likes': '1037'},
 ↪  {'fecha': '2022-05-14 14:01:45+00:00', 'usuario':
 ↪  'ChiguireBipolar', 'texto': 'Testigo de Jehová se esconde de
 ↪  promotor del Bitcoin https://t.co/odVOELmYKu
 ↪  https://t.co/GOtuWLoGhT', 'likes': '1082'}, {'fecha':
 ↪  '2022-05-15 01:21:58+00:00', 'usuario': 'el_pais', 'texto': 'El
 ↪  Salvador convirtió al Bitcoin en su moneda legal. Con la caída
 ↪  de las criptomonedas, han caído en picado también…
 ↪  https://t.co/fFlk1P1LSj', 'likes': '551'}, {'fecha': '2022-05-13
 ↪  23:00:28+00:00', 'usuario': 'garciabanchs', 'texto': 'La
 ↪  cantidad de dinero que han perdido fans por hacerle caso a
 ↪  celebridades en torno a la adquisición de criptomoned…
 ↪  https://t.co/UOjmvj4Jxq', 'likes': '1143'}, {'fecha':
 ↪  '2022-05-14 15:44:24+00:00', 'usuario': 'juanrallo', 'texto':
 ↪  'Llevas diciendo que Bitcoin es una burbuja desde que cotizaba a
 ↪  100 dólares. Que separes los patrones típicos de un
```

```
{'#Bitcoin': 3, '#Ethereum': 1}
```

Ver notebook de la solución en Colab

11 2022-06-24 Examen de Inteligencia de los Negocios

Grado: Inteligencia de los Negocios
Fecha: 24 de junio de 2022

Ejercicio 11.1. Las matemáticas financieras, resumidas en una frase, las podríamos definir como la rama de las matemáticas que estudia los flujos de dinero a través del tiempo. Básicamente se presupone que el dinero tiene menos valor en el futuro que en el presente, ya sea por un tema inflacionario o por la preferencia natural de las personas a priorizar el consumo presente.

El valor futuro es el valor alcanzado por un determinado capital al final del período determinado (para el ejemplo usaremos la fórmula del interés compuesto). Para calcularlo se utiliza la siguiente fórmula

$$VF = VA \cdot (1 + i)^n$$

El valor presente de una inversión es cuando calculamos el valor actual que tendrá una determinada cantidad que recibiremos o pagaremos en un futuro. Para calcularlo se utiliza la siguiente fórmula

$$VA = \frac{VF}{(1 + i)^n}$$

Donde VF es el valor futuro, VA es el valor actual o inicial de la inversión, i es el tipo de interés y n es número de años de la inversión.

1. Crear una función que reciba como entrada un capital, un tipo de interés y un número de años, y muestre por pantalla el valor futuro de la inversión cada año del periodo indicado.
2. Crear una función que reciba como entrada un capital, un tipo de interés y un número de años, y muestre por pantalla el valor actual del capital cada año del periodo indicado.

Ejemplo de ejecución

```
Introduce un capital: 1000
Introduce un tipo de interés: 10
Introduce un número de años: 3
VALOR FUTURO
Año 0 : 1100.0
Año 1 : 1210.0000000000002
Año 2 : 1331.0000000000005
VALOR ACTUAL
Año 0 : 909.090909090909
Año -1 : 826.4462809917354
Año -2 : 751.3148009015775
```

Solución

```python
def vf(capital, tipo, periodo):
    '''
    Función que muestra por pantalla el valor futuro de una inversió
 ↪  n cada año de un periodo dado.
    Parámetros:
        - capital: Es un real con el capital inicial de la inversión
 ↪  .
        - tipo: Es un real con el tipo de interés anual.
        - periodo: Es un entero con el número de años de la inversió
 ↪  n.
    '''
    # Bucle iterativo para recorrer la secuencia de años.
    for i in range(periodo):
        # Aplicamos la fórmula para el cálculo del valor futuro para
         ↪  cada año i.
        print("Año", i, ":", capital * (1 + tipo / 100) ** (i + 1))
    return

def va(capital, tipo, periodo):
    '''
    Función que muestra por pantalla el valor actual de una inversió
 ↪  n cada año de un periodo dado.
    Parámetros:
        - capital: Es un real con el capital final de la inversión.
        - tipo: Es un real con el tipo de interés anual.
        - periodo: Es un entero con el número de años de la inversió
 ↪  n.
    '''
    # Bucle iterativo para recorrer la secuencia de años.
    for i in range(periodo):
         # Aplicamos la fórmula para el cálculo del valor actual par
          ↪  a cada año i.
        print("Año", -i, ":", capital / (1 + tipo / 100) ** (i + 1))
    return

# Ejemplo
cantidad = float(input("Introduce un capital: "))
tipo = float(input("Introduce un tipo de interés: "))
años = int(input("Introduce un número de años: "))
print("VALOR FUTURO")
vf(cantidad, tipo, años)
print("VALOR ACTUAL")
va(cantidad, tipo, años)
```

```
VALOR FUTURO
Año 0 : 1100.0
Año 1 : 1210.0000000000002
Año 2 : 1331.0000000000005
VALOR ACTUAL
Año 0 : 909.090909090909
Año -1 : 826.4462809917354
Año -2 : 751.3148009015775
```

Ver notebook de la solución en Colab

Ejercicio 11.2. Realizar un programa que simule el juego de las tres en raya. Cada jugador solo debe colocar su ficha una vez por turno y no debe ser sobre una casilla ya ocupada. En caso de que el jugador haga trampa el ganador será el otro. Para ganar se debe conseguir realizar una línea recta (horizontal, vertical o diagonal) con la misma ficha. El tablero es de 3x3 y cualquier casilla podrá estar vacía u ocupada sólo por una ficha X o 0. El programa debe realizar las siguientes operaciones:

- Mostrar el tablero por pantalla.
- Poner una ficha en una cuadricula comprobando que no está ocupada.
- Comprobar si se produce tres en raya e indicar si es de X o de O, o si hay empate.

Ejemplo de ejecución

```
[' ', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']
Introduce la coordenada X del 1 al 3: 2
Introduce la coordenada Y del 1 al 3: 2
Introduce 0 para círculo o X para cruz: X
[' ', ' ', ' ']
[' ', 'X', ' ']
[' ', ' ', ' ']
Introduce la coordenada X del 1 al 3: 2
Introduce la coordenada Y del 1 al 3: 1
Introduce 0 para círculo o X para cruz: 0
[' ', ' ', ' ']
['O', 'X', ' ']
[' ', ' ', ' ']
Introduce la coordenada X del 1 al 3: 2
Introduce la coordenada Y del 1 al 3: 1
Introduce 0 para círculo o X para cruz: X
Esa coordenada ya está siendo utilizada, use una libre.
```

```
[' ', ' ', ' ']
['O', 'X', ' ']
[' ', ' ', ' ']
Introduce la coordenada X del 1 al 3: 3
Introduce la coordenada Y del 1 al 3: 1
Introduce O para círculo o X para cruz: X
[' ', ' ', ' ']
['O', 'X', ' ']
['X', ' ', ' ']
Introduce la coordenada X del 1 al 3: 1
Introduce la coordenada Y del 1 al 3: 2
Introduce O para círculo o X para cruz: O
[' ', 'O', ' ']
['O', 'X', ' ']
['X', ' ', ' ']
Introduce la coordenada X del 1 al 3: 1
Introduce la coordenada Y del 1 al 3: 3
Introduce O para círculo o X para cruz: X
[' ', 'O', 'X']
['O', 'X', ' ']
['X', ' ', ' ']
La partida la ha ganado el jugador con la ficha X.
```

Solución

```
# Definimos el tablero como una lista de 3 listas.
tablero = [
    [" ", " ", " "],
    [" ", " ", " "],
    [" ", " ", " "],
]

def mostrar_tablero(tablero):
    '''
    Función que muestra el tablero por pantalla.

    Parámetros:
        - tablero: Es una lista de listas con el tablero.
    '''
    # Bucle iterativo para recorrer por valor la lista del tablero.
    # fila toma como valores cada una de las listas de la lista tabl
     ↪ ero.
    for fila in tablero:
        # Mostramos la fila por pantalla. Se muestra como una lista.
         ↪
        print(fila)
    return True

def poner_ficha(tablero, x, y, ficha):
    '''
    Función que pone una ficha en el tablero.

    Parámetros:
        tablero: Es una lista de listas con el tablero.
        x: Es un entero entre 1 y 3 con la fila donde colocar la fic
 ↪ ha.
        y: Es un entero entre 1 y 3 con la columna donde colocar la
 ↪ ficha.
        ficha: Es una cadena con el símbolo a colocar.
    Salida:
        El tablero con la nueva ficha colocada si la posición dada e
 ↪ stá libre o el tablero original si no.
    '''
    # Condicional para ver si la posición indicada está vacía.
    if tablero[x-1][y-1] == " ":
        # Si la posición está vacía, colocamos la ficha en la posici
         ↪ ón.
        tablero[x-1][y-1] = ficha
    else:
        # Si no está vacía, avisamos al usuario.
        print("Esa coordenada ya está siendo utilizada, use una libr
         ↪ e.")
    return tablero

def final(tablero):
    '''
    Función para comprobar si hay ganador en tablero.

    Parámetros:
        - tablero: Es una lista de listas con el tablero.
    Salida:
```

```
[' ', ' ', ' ']
[' ', ' ', ' ']
[' ', ' ', ' ']

[' ', ' ', ' ']
[' ', 'X', ' ']
[' ', ' ', ' ']

[' ', ' ', ' ']
['O', 'X', ' ']
[' ', ' ', ' ']

Esa coordenada ya está siendo utilizada, use una libre.
[' ', ' ', ' ']
['O', 'X', ' ']
[' ', ' ', ' ']

[' ', ' ', ' ']
['O', 'X', ' ']
['X', ' ', ' ']

[' ', 'O', ' ']
['O', 'X', ' ']
['X', ' ', ' ']

[' ', 'O', 'X']
['O', 'X', ' ']
['X', ' ', ' ']
La partida la ha ganado el jugador con la ficha X.
```

Ver notebook de la solución en Colab

Ejercicio 11.3. El fichero bank-loans.csv contiene información sobre los préstamos de los clientes de un banco. Utilizando la librería Pandas, se pide:

- Crear un DataFrame importando los datos del fichero.
- Mostrar por pantalla el nombre de las columnas del DataFrame.
- Mostrar por pantalla las filas del DataFrame múltiplos de 10.
- Mostrar por pantalla el número de clientes casados con edad entre 30 y 40 años.
- Añadir al DataFrame una columna nueva con la edad en meses.
- Mostrar por pantalla las frecuencias de los oficios ordenadas de mayor a menor.
- Mostrar por pantalla las edades medias según el nivel de estudios.
- Mostrar por pantalla el porcentaje de préstamos hipotecarios (housing) según el estado civil (marital).

- Dibujar el diagrama de sectores con los porcentajes de los niveles de estudio y ponerle un título.
- Dibujar en una misma figura el histograma y el diagrama de cajas de las edades. El histograma debe tener clases de amplitud 10 desde 20 hasta 70 años, y en color rojo.

Solución

```
import pandas as pd
import matplotlib.pyplot as plt

# Crear el DataFrame a partir de la url del fichero csv
df = pd.read_csv("https://aprendeconalf.es/docencia/python/examenes/
 ↪  inteligencia-negocios/soluciones/examen-2022-06-24/bank-loans.cs
 ↪  v")
# Mostrar el índice de las columnas
print(df.columns)
# Mostrar las filas múltiplo de 10
print(df.iloc[range(0, len(df), 10)])
# Mostrar el número de clientes casados entre 30 y 40 años
print(len(df[(df.marital == "married") & (df.age > 30) & (df.age <
 ↪  40)]))
# Añadir una columna con la edad en meses.
df["months"] = df.age * 12
print(df)
# Frecuencias de oficios ordenados de mayor a menor
print(df.job.value_counts().sort_values(ascending = False))
# Calcular la edad media según el nivel de estudios
print(df.groupby("education").age.mean())
# Calcular el porcentaje de prestamos hipotecarios según el estado c
 ↪  ivil
print(df.groupby("marital").housing.value_counts(normalize = True) *
 ↪  100)
# Diagrama de sectores de nivel de estudios
fig, ax = plt.subplots()
df.education.value_counts(normalize = True).plot(kind = "pie", title
 ↪  = "Distribución del nivel de estudios")
plt.show()
# Histograma y diagrama de cajas de edades
fig, ax = plt.subplots(2)
df.age.plot(kind = "hist", ax = ax[0], bins = range(20, 70,10),
 ↪  color = "red")
df.age.plot(kind = "box", ax = ax[1], vert = False)
plt.show()
```

```
Index(['age', 'job', 'marital', 'education', 'default', 'housing',
 ↪  'loan'], dtype='object')
      age          job   marital            education  default
       ↪  housing loan
```

```
0      56    housemaid   married             basic.4y       no
 ↪  no   no
10     41  blue-collar   married              unknown  unknown
 ↪  no   no
20     30   unemployed   married          high.school       no
 ↪  no   no
30     46       admin.   married              unknown       no
 ↪  no   no
40     58   management   unknown    university.degree       no
 ↪  yes   no
...   ...          ...       ...                  ...      ...
 ↪  ...  ...
9950   49  blue-collar   married             basic.9y  unknown
 ↪  no   no
9960   51   unemployed   married          high.school  unknown
 ↪  yes   no
9970   43   technician    single  professional.course       no
 ↪  yes   no
9980   49       admin.  divorced    university.degree       no
 ↪  no   no
9990   35  blue-collar    single              unknown       no
 ↪  no   no

[1000 rows x 7 columns]
2494
      age          job   marital            education  default
       ↪  housing  \
0      56    housemaid   married             basic.4y       no
 ↪  no
1      57     services   married          high.school  unknown
 ↪  no
2      37     services   married          high.school       no
 ↪  yes
3      40       admin.   married             basic.6y       no
 ↪  no
4      56     services   married          high.school       no
 ↪  no
...   ...          ...       ...                  ...      ...
 ↪  ...
9995   39  blue-collar   married             basic.4y  unknown
 ↪  yes
```

```
9996   36   technician   married          high.school       no
 ↪  yes
9997   33     services    single             basic.6y       no
 ↪  unknown
9998   41   technician   married  professional.course  unknown
 ↪  yes
9999   40     services  divorced              unknown       no
 ↪  yes

         loan  months
0          no     672
1          no     684
2          no     444
3          no     480
4         yes     672
...       ...     ...
9995       no     468
9996       no     432
9997  unknown     396
9998       no     492
9999      yes     480

[10000 rows x 8 columns]
blue-collar      2980
admin.           2073
technician       1365
services         1172
management        678
entrepreneur      381
self-employed     340
housemaid         297
retired           273
unemployed        236
unknown           125
student            80
Name: job, dtype: int64
education
basic.4y               45.147806
basic.6y               39.448095
basic.9y               39.228571
high.school            38.313602
illiterate             45.000000
professional.course    40.367776
```

```
university.degree      39.853485
unknown                43.709278
Name: age, dtype: float64
marital   housing
divorced  no         50.940018
          yes        46.821844
          unknown     2.238138
married   no         51.987494
          yes        45.183862
          unknown     2.828644
single    no         51.931131
          yes        44.671940
          unknown     3.396929
unknown   no         64.705882
          yes        35.294118
Name: housing, dtype: float64
<Figure size 432x288 with 1 Axes><Figure size 432x288 with 2 Axes>
```

Ver notebook de la solución en Colab

Ejercicio 11.4. El fichero bank-loans.csv contiene información sobre los préstamos de los clientes de un banco. Utilizando ficheros (sin usar la librería Pandas), crear una función que tenga como parámetros la url del fichero, el nombre de una columna cualitativa y un booleano para los porcentajes (False por defecto), que devuelva un diccionario con las frecuencias absolutas de las categorías de la columna indicada. Si se pasa True como argumento del parámetro para el porcentaje, la función debe devolver el diccionario de porcentajes de las categorías.

Ejemplo de ejecución

Salida de la llamada a la función con la columna "education".

```
{'basic.4y': 1299, 'high.school': 2382, 'basic.6y': 761, 'basic.9y':
 ↪  1820, 'professional.course': 1142, 'unknown': 485, 'university.degre
 ↪  e': 2109, 'illiterate': 2}
```

Salida de la llamada a la función con la columna "housing" y porcentaje = `True`.

```
{'no': 51.88, 'yes': 45.24, 'unknown': 2.88}
```

Solución

```
# Importamos las librerías
from urllib import request
from urllib.error import URLError

def frecuencias(url, columna, porcentaje = False):
    '''
    Función que calcula la tabla de frecuencias absoluta o relativa
 ↪  de una columna de un fichero csv.

    Parámetros:
        - url: Es una cadena con la url del fichero csv.
        - columna: Es una cadena con el nombre de la columna.
        - porcentaje: Es un booleano (False por defecto) que indica
 ↪  si las frecuencias son relativas o no.

    Salida:
        Un diccionario con las categorías de la columna y sus frecue
 ↪  ncias.
    '''
    try:
        # Intentamos abrir el fichero a partir de su url.
        with request.urlopen(url) as f:
            # Leemos el contenido del fichero, lo decodificamos y cr
             ↪  eamos una lista partiendo el contenido por el carácter
             ↪  er de cambio de linea.
            contenido = f.read().decode().split("\n")
    except URLError:
        # Si no existe el fichero en la url indicada controlamos el
         ↪  error y avisamos al usuario.
        print("La url", url, "no existe")
    else:
        # Creamos una lista con los encabezados dividiendo la primer
         ↪  a línea por la coma.
        encabezado = contenido.pop(0).split(",")
        # Obtenemos la posición de la columna indicada en la lista d
         ↪  e encabezados.
        numcol = encabezado.index(columna)
        # Inicializamos un diccionario vacío para ir guardando los p
         ↪  ares con las categorías y sus frecuencias.
        dic_frec = {}
        # Bucle iterativo para recorrer por valor las líneas del fic
         ↪  hero.
        # i toma como valores las líneas del fichero.
        for i in contenido:
            # Obtenemos la clave dividiendo la línea por la coma y a
             ↪  ccediendo a la posición de la columna indicada.
            clave = i.split(",")[numcol]
            # Condicional para ver si la clave está ya en el diccion
             ↪  ario.
            if clave in dic_frec:
                # Si la clave está ya en el diccionario incrementamo
                 ↪  s la frecuencia en 1.
                dic_frec[clave] += 1
            else:
                # Si la clave no está en el diccionario, añadimos un
                 ↪  nuevo par con la clave y frecuencia 1.
```

```
{'basic.4y': 1299, 'high.school': 2382, 'basic.6y': 761, 'basic.9y':
 ↪  1820, 'professional.course': 1142, 'unknown': 485,
 ↪  'university.degree': 2109, 'illiterate': 2}
{'no': 51.88, 'yes': 45.24, 'unknown': 2.88}
```

Ver notebook de la solución en Colab