Ejercicios de Programación con Python

Ejercicios de programación con Python con soluciones

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

Tabla de contenidos

Prefacio

¡Bienvenido a los Ejercicios de Programación con Python!

Este libro contiene una colección de ejercicios resueltos de programación con Python. La mayor parte de ellos son ejercicios aplicados a la economía, pero también hay ejercicios genéricos para cualquier disciplina.

Los ejercicios están clasificados por temas y siguen el orden más o menos habitual en el aprendizaje de este lenguaje. Cada ejercicio incluye su solución en un desplegable, junto con un enlace para abrirla y ejecutarla en Google Colab.

Este libro es un complemento ideal del Manual de Python, donde se explican los conceptos necesarios para resolver los ejercicios.

Capítulos

Licencia

1. Tipos de Datos Simples

Ejercicio 1.1. Escribir un programa que muestre por pantalla la cadena `¡Hola Mundo!`.

Solución

```
print("¡Hola Mundo!")
```

Abrir la solución en Google Colab

Ejercicio 1.2. Escribir un programa que almacene la cadena `¡Hola Mundo!` en una variable y luego muestre por pantalla el contenido de la variable.

Solución

```
mensaje = "¡Hola Mundo!"
print(mensaje)
```

Abrir la solución en Google Colab

Ejercicio 1.3. Escribir un programa que pregunte el nombre del usuario en la consola y después de que el usuario lo introduzca muestre por pantalla la cadena `¡Hola <nombre>!`, donde `<nombre>` es el nombre que el usuario haya introducido.

Solución

```
nombre = input("Introduce tu nombre: ")
print("¡Hola " + nombre + "!")
```

Abrir la solución en Google Colab

Ejercicio 1.4. Escribir un programa que muestre por pantalla el resultado de la siguiente operación aritmética $\left(\frac{3+2}{2\cdot 5}\right)^2$.

Solución

```
print(((3 + 2) / (2 * 5)) ** 2)
```

Abrir la solución en Google Colab

Ejercicio 1.5. Escribir un programa que pregunte al usuario por el número de horas trabajadas y el coste por hora. Después debe mostrar por pantalla la paga que le corresponde.

Solución

```
horas = float(input("Introduce tus horas de trabajo: "))
coste = float(input("Introduce lo que cobras por hora: "))
paga = horas * coste
print("Tu paga es", paga)
```

Abrir la solución en Google Colab

Ejercicio 1.6. Escribir un programa que lea un entero positivo, n, introducido por el usuario y después muestre en pantalla la suma de todos los enteros desde 1 hasta n. La suma de los n primeros enteros positivos puede ser calculada de la siguiente forma:

$$\text{suma} = \frac{n(n+1)}{2}$$

Solución

```
n = int(input("Introduce un número entero: "))
suma = n * (n + 1) / 2
print("La suma de los primeros números enteros desde 1 hasta " +
 ↪  str(n) + " es " + str(suma))
```

Abrir la solución en Google Colab

Ejercicio 1.7. Escribir un programa que pida al usuario su peso (en kg) y estatura (en metros), calcule el índice de masa corporal y lo almacene en una variable, y muestre por pantalla la frase `Tu índice de masa corporal es <imc>` donde `<imc>` es el índice de masa corporal calculado redondeado con dos decimales.

Solución

```
peso = input("¿Cuál es tu peso en kg? ")
estatura = input("¿Cuál es tu estatura en metros?")
imc = round(float(peso)/float(estatura)**2,2)
print("Tu índice de masa corporal es " + str(imc))
```

Abrir la solución en Google Colab

Ejercicio 1.8. Escribir un programa que pida al usuario dos números enteros y muestre por pantalla la `<n> entre <m> da un cociente <c> y un resto <r>` donde `<n>` y `<m>` son los números introducidos por el usuario, y `<c>` y `<r>` son el cociente y el resto de la división entera respectivamente.

Solución

```
n = input("Introduce el dividendo (entero): ")
m = input("Introduce el divisor (entero): ")
print(n + " entre " +  m + " da un cociente " + str(int(n) //
 ↪  int(m)) + " y un resto " + str(int(n) % int(m)))
```

Abrir la solución en Google Colab

Ejercicio 1.9. Escribir un programa que pregunte al usuario una cantidad a invertir, el interés anual y el número de años, y muestre por pantalla el capital obtenido en la inversión.

Solución

```
cantidad = float(input("¿Cantidad a invertir? "))
interes = float(input("¿Interés porcentual anual? "))
años = int(input("¿Años?"))
print("Capital final: " + str(round(cantidad * (interes / 100 + 1)
 ↪  ** años, 2)))
```

Abrir la solución en Google Colab

Ejercicio 1.10. Una juguetería tiene mucho éxito en dos de sus productos: payasos y muñecas. Suele hacer venta por correo y la empresa de logística les cobra por peso de cada paquete así que deben calcular el peso de los payasos y muñecas que saldrán en cada

paquete a demanda. Cada payaso pesa 112 g y cada muñeca 75 g. Escribir un programa que lea el número de payasos y muñecas vendidos en el último pedido y calcule el peso total del paquete que será enviado.

Solución

```
peso_payaso = 112
peso_muñeca = 75
payasos = int(input("Introduce el número de payasos a enviar: "))
muñecas = int(input("Introduce el número de muñecas a enviar: "))
peso_total = peso_payaso * payasos + peso_muñeca * muñecas
print("El peso total del paquete es " + str(peso_total))
```

Abrir la solución en Google Colab

Ejercicio 1.11. Imagina que acabas de abrir una nueva cuenta de ahorros que te ofrece el 4 % de interés al año. Estos ahorros debido a intereses, que no se cobran hasta finales de año, se te añaden al balance final de tu cuenta de ahorros. Escribir un programa que comience leyendo la cantidad de dinero depositada en la cuenta de ahorros, introducida por el usuario. Después el programa debe calcular y mostrar por pantalla la cantidad de ahorros tras el primer, segundo y tercer años. Redondear cada cantidad a dos decimales.

Solución

```
inversion = float(input("Introduce la inversión inicial: "))
interes = 0.04
balance1 = inversion * (1 + interes)
print("Balance tras el primer año:" + str(round(balance1, 2)))
balance2 = balance1 * (1 + interes)
print("Balance tras el segundo año:" + str(round(balance2, 2)))
balance3 = balance2 * (1 + interes)
print("Balance tras el tercer año:" + str(round(balance3, 2)))
```

Abrir la solución en Google Colab

Ejercicio 1.12. Una panadería vende barras de pan a 3.49€ cada una. El pan que no es el día tiene un descuento del 60 %. Escribir un programa que comience leyendo el número de barras vendidas que no son del día. Después el programa debe mostrar el precio habitual de una barra de pan, el descuento que se le hace por no ser fresca y el coste final total.

Solución

```
barras = int(input("Introduce el número de barras vendidas que no
 ↪  son frescas: "))
precio = 3.49
descuento = 0.6
coste = barras * precio * (1 - descuento)
print("El coste de una barra fresca es " + str(precio) + "€")
print("El descuento sobre una barra no fresca es " + str(descuento *
 ↪  100) + "%")
print("El coste final a pagar es " + str(round(coste, 2)) + "€")
```

Abrir la solución en Google Colab

2. Cadenas

Ejercicio 2.1. Escribir un programa que pregunte el nombre del usuario en la consola y un número entero e imprima por pantalla en líneas distintas el nombre del usuario tantas veces como el número introducido.

Solución

```
nombre = input("¿Cómo te llamas? ")
n = input("Introduce un número entero: ")
print((nombre + "\n") * int(n))
```

Abrir la solución en Google Colab

Ejercicio 2.2. Escribir un programa que pregunte el nombre completo del usuario en la consola y después muestre por pantalla el nombre completo del usuario tres veces, una con todas las letras minúsculas, otra con todas las letras mayúsculas y otra solo con la primera letra del nombre y de los apellidos en mayúscula. El usuario puede introducir su nombre combinando mayúsculas y minúsculas como quiera.

Solución

```
name = input("¿Cómo te llamas? ")
print(name.lower())
print(name.upper())
print(name.title())
```

Abrir la solución en Google Colab

Ejercicio 2.3. Escribir un programa que pregunte el nombre del usuario en la consola y después de que el usuario lo introduzca muestre por pantalla `<NOMBRE> tiene <n> letras`, donde `<NOMBRE>` es el nombre de usuario en mayúsculas y `<n>` es el número de letras que tienen el nombre.

Solución

```
nombre = input("¿Cómo te llamas? ")
print(nombre.upper() + " tiene " + str(len(nombre)) + " letras")
```

Abrir la solución en Google Colab

Ejercicio 2.4. Los teléfonos de una empresa tienen el siguiente formato `prefijo-número-extension` donde el prefijo es el código del país `+34`, y la extensión tiene dos dígitos (por ejemplo `+34-913724710-56`). Escribir un programa que pregunte por un número de teléfono con este formato y muestre por pantalla el número de teléfono sin el prefijo y la extensión.

Solución

```
tel = input("Introduce un número de teléfono con el formato
 ↪  +xx-xxxxxxxxx-xx: ")
print('El número de teléfono es ', tel[4:-3])
```

Abrir la solución en Google Colab

Ejercicio 2.5. Escribir un programa que pida al usuario que introduzca una frase en la consola y muestre por pantalla la frase invertida.

Solución

```
frase = input("Introduce una frase: ")
print(frase[::-1])
```

Abrir la solución en Google Colab

Ejercicio 2.6. Escribir un programa que pida al usuario que introduzca una frase en la consola y una vocal, y después muestre por pantalla la misma frase pero con la vocal introducida en mayúscula.

Solución

```
frase = input("Introduce una frase: ")
vocal = input("Introduce una vocal en minúscula:  ")
print(frase.replace(vocal, vocal.upper()))
```

Abrir la solución en Google Colab

Ejercicio 2.7. Escribir un programa que pregunte el correo electrónico del usuario en la consola y muestre por pantalla otro correo electrónico con el mismo nombre (la parte delante de la arroba `@`) pero con dominio `ceu.es`.

Solución

```
email = input("Introduce tu correo electrónico: ")
print(email[:email.find('@')] + '@ceu.es')
```

Abrir la solución en Google Colab

Ejercicio 2.8. Escribir un programa que pregunte por consola el precio de un producto en euros con dos decimales y muestre por pantalla el número de euros y el número de céntimos del precio introducido.

Solución

```
precio = input("Introduce el precio del producto con dos decimales:
 ↪  ")
print(precio[:precio.find('.')], 'euros y',
 ↪  precio[precio.find('.')+1:], 'céntimos.')
```

Abrir la solución en Google Colab

Ejercicio 2.9. Escribir un programa que pregunte al usuario la fecha de su nacimiento en formato `dd/mm/aaaa` y muestra por pantalla, el día, el mes y el año. Adaptar el programa anterior para que también funcione cuando el día o el mes se introduzcan con un solo carácter.

Solución

1

```
fecha = input("Introduce la fecha de tu nacimiento en formato
 ↪  dd/mm/aaaa: ")
print('Día', fecha[:2])
print('Mes', fecha[3:5])
print('Año', fecha[6:])
```

Adaptar el programa anterior para que también funcione cuando el día o el mes se introduzcan con un solo carácter.

```
fecha = input("Introduce la fecha de tu nacimiento en formato
 ↪  día/mes/año: ")
dia = fecha[:fecha.find('/')]
mesaño = fecha[fecha.find('/')+1:]
mes = mesaño[:mesaño.find('/')]
año = mesaño[mesaño.find('/')+1:]
print('Día', dia)
print('Mes', mes)
print('Año', año)
```

Abrir la solución en Google Colab

Ejercicio 2.10. Escribir un programa que pregunte por consola por los productos de una cesta de la compra, separados por comas, y muestre por pantalla cada uno de los productos en una línea distinta.

Solución

1

```
cesta = input('Introduce los productos de la cesta de la compra
 ↪  separados por comas: ')
print(cesta.replace(',', '\n'))
```

2

```
cesta = input('Introduce los productos de la cesta de la compra
 ↪  separados por comas: ')
print('\n'.join(cesta.split(',')))
```

Abrir la solución en Google Colab

Ejercicio 2.11. Escribir un programa que pregunte el nombre el un producto, su precio y un número de unidades y muestre por pantalla una cadena con el nombre del producto seguido de su precio unitario con 6 dígitos enteros y 2 decimales, el número de unidades con tres dígitos y el coste total con 8 dígitos enteros y 2 decimales.

Solución

```
producto = input('Introduce el nombre del producto: ')
precio = float(input('Introducde el precio unitario: '))
unidades = int(input('Introduce el número de unidades: '))
print('{producto}: {unidades:3d} unidades x {precio:09.2f}€ =
 ↪  {total:011.2f}€'.format(producto = producto, unidades =
 ↪  unidades, precio = precio, total = unidades * precio))
```

Abrir la solución en Google Colab

3. Condicionales

Ejercicio 3.1. Escribir un programa que pregunte al usuario su edad y muestre por pantalla si es mayor de edad o no.

Solución

```
age = int(input("¿Cuál es tu edad? "))
if age < 18:
    print ("Eres menor de edad.")
else:
    print("Eres mayor de edad.")
```

Abrir la solución en Google Colab

Ejercicio 3.2. Escribir un programa que almacene la cadena de caracteres `contraseña` en una variable, pregunte al usuario por la contraseña e imprima por pantalla si la contraseña introducida por el usuario coincide con la guardada en la variable sin tener en cuenta mayúsculas y minúsculas.

Solución

```
key = "contraseña"
password = input("Introduce la contraseña: ")
if key == password.lower():
    print("La contaseña coincide")
else:
    print("La contraseña no coincide")
```

Abrir la solución en Google Colab

Ejercicio 3.3. Escribir un programa que pida al usuario dos números y muestre por pantalla su división. Si el divisor es cero el programa debe mostrar un error.

Solución

```
n = float(input("Introduce el dividendo: "))
m = float(input("Introduce el divisior: "))
if m == 0:
    print("¡Error! No se puede dividir por 0.")
else:
    print(n/m)
```

Abrir la solución en Google Colab

Ejercicio 3.4. Escribir un programa que pida al usuario un número entero y muestre por pantalla si es par o impar.

Solución

1

```
n = int(input("Introduce un número entero: "))
if n % 2 == 0:
    print("El número " + str(n) + " es par")
else:
    print("El número " + str(n) + " es impar")
```

Abrir la solución en Google Colab

Ejercicio 3.5. Para tributar un determinado impuesto se debe ser mayor de 16 años y tener unos ingresos iguales o superiores a 1000 € mensuales. Escribir un programa que pregunte al usuario su edad y sus ingresos mensuales y muestre por pantalla si el usuario tiene que tributar o no.

Solución

1

```
age = int(input("¿Cuál es tu edad? "))
income = float(input("¿Cuales son tus ingresos mensuales?"))
if age > 16 and income >= 1000:
    print("Tienes que cotizar")
else:
    print("No tienes que cotizar")
```

2

```
age = int(input("¿Cuál es tu edad? "))
income = float(input("¿Cuales son tus ingresos mensuales?"))
if age <= 16 or income < 1000:
    print("No tienes que cotizar")
else:
    print("Tienes que cotizar")
```

Abrir la solución en Google Colab

Ejercicio 3.6. Los alumnos de un curso se han dividido en dos grupos A y B de acuerdo al sexo y el nombre. El grupo A esta formado por las mujeres con un nombre anterior a la M y los hombres con un nombre posterior a la N y el grupo B por el resto. Escribir un programa que pregunte al usuario su nombre y sexo, y muestre por pantalla el grupo que le corresponde.

Solución

1

```
name = input("¿Cómo te llamas? ")
gender = input("¿Cuál es tu sexo (M o H)? ")
if gender == "M":
    if name.lower() < "m":
        group = "A"
    else:
        group = "B"
else:
    if name.lower() > "n":
        group = "A"
    else:
        group = "B"
print("Tu grupo es " + group)
```

2

```
name = input("¿Cómo te llamas? ")
gender = input("¿Cuál es tu sexo (M o H)? ")
if (gender == "M" and name.lower() < 'm') or (gender == "H" and
 ↪  name.lower() > 'n'):
    group = "A"
else:
    group = "B"
print("Tu grupo es " + group)
```

Abrir la solución en Google Colab

Ejercicio 3.7. Los tramos impositivos para la declaración de la renta en un determinado país son los siguientes:

Renta	Tipo impositivo
Menos de 10000€	5 %
Entre 10000€ y 20000€	15 %
Entre 20000€ y 35000€	20 %
Entre 35000€ y 60000€	30 %
Más de 60000€	45 %

Escribir un programa que pregunte al usuario su renta anual y muestre por pantalla el tipo impositivo que le corresponde.

Solución

```
# Preguntar al usuario por la renta
renta = float(input("¿Cuál es tu renta anual? "))
# Condicional para determinar el tipo impositivo dependiendo de la
 ↪  renta
if renta < 10000:
    tipo = 5
elif renta < 20000:
    tipo = 15
elif renta < 35000:
    tipo = 20
elif renta < 60000:
    tipo = 30
else:
    tipo = 45
# Mostrar por pantalla el producto de la renta por el tipo
 ↪  impositivo
print("Tienes que pagar ", renta * tipo / 100, "€", sep='')
```

Abrir la solución en Google Colab

Ejercicio 3.8. En una determinada empresa, sus empleados son evaluados al final de cada año. Los puntos que pueden obtener en la evaluación comienzan en 0.0 y pueden ir aumentando, traduciéndose en mejores beneficios. Los puntos que pueden conseguir los empleados pueden ser 0.0, 0.4, 0.6 o más, pero no valores intermedios entre las cifras mencionadas. A continuación se muestra una tabla con los niveles correspondientes a cada puntuación. La cantidad de dinero conseguida en cada nivel es de 2.400€ multiplicada por la puntuación del nivel.

Nivel	Puntuación
Inaceptable	0.0
Aceptable	0.4
Meritorio	0.6 o más

Escribir un programa que lea la puntuación del usuario e indique su nivel de rendimiento, así como la cantidad de dinero que recibirá el usuario.

Solución

```
bonificacion = 2400
inaceptable = 0
aceptable = 0.4
meritorio = 0.6
puntos = float(input("Introduce tu puntuación: "))
# Clasifiación por niveles de rendimiento
if puntos == inaceptable:
    nivel = "Inaceptable"
elif puntos == aceptable:
    nivel = "Aceptable"
elif puntos >= 0.6:
    nivel = "Meritorio"
else:
    nivel = ""
# Mostrar nivel de rendimiento
if nivel == "":
    print("Esta puntuación no es válida")
else:
    print("Tu nivel de rendimiento es %s" % nivel)
    print("Te corresponde cobrar %.2f€" % (puntos * bonificacion))
```

Abrir la solución en Google Colab

Ejercicio 3.9. Escribir un programa para una empresa que tiene salas de juegos para todas las edades y quiere calcular de forma automática el precio que debe cobrar a sus clientes por entrar. El programa debe preguntar al usuario la edad del cliente y mostrar el precio de la entrada. Si el cliente es menor de 4 años puede entrar gratis, si tiene entre 4 y 18 años debe pagar 5€ y si es mayor de 18 años, 10€.

Solución

```
edad = int(input("Introduce tu edad: "))
# Decisión del precio en función de la edad
if edad < 4:
    precio = 0
elif edad <= 18:
    precio = 4
else:
    precio = 10
# Mostrar precio
print("El precio de la entrada es", precio, "€.")
```

Abrir la solución en Google Colab

Ejercicio 3.10. La pizzería Bella Napoli ofrece pizzas vegetarianas y no vegetarianas a sus clientes. Los ingredientes para cada tipo de pizza aparecen a continuación.

- Ingredientes vegetarianos: Pimiento y tofu.
- Ingredientes no vegetarianos: Peperoni, Jamón y Salmón.

Escribir un programa que pregunte al usuario si quiere una pizza vegetariana o no, y en función de su respuesta le muestre un menú con los ingredientes disponibles para que elija. Solo se puede eligir un ingrediente además de la mozzarella y el tomate que están en todas la pizzas. Al final se debe mostrar por pantalla si la pizza elegida es vegetariana o no y todos los ingredientes que lleva.

Solución

```
# Presentación del menú con los tipos de pizza
print("Bienvenido a la pizzeria Bella Napoli.\nTipos de pizza\n\t1-
 ↪  Vegetariana\n\t2- No vegetariana\n")
tipo = input("Introduce el número correspondiente al tipo de pizza
 ↪  que quieres:")
# Decisión sobre el tipo de pizza
if tipo == "1":
    print("Ingredientes de pizzas vegetarianas\n\t 1- Pimiento\n\t2-
     ↪  Tofu\n")
    ingrediente = input("Introduce el ingrediente que deseas: ")
    print("Pizza vegetariana con mozzarella, tomate y ", end="")
    if ingrediente == "1":
        print("pimiento")
    else:
        print("tofu")
else:
    print("Ingredientes de pizzas no vegetarianas\n\t1-
     ↪  Peperoni\n\t2- Jamón\n\t3- Salmón\n")
    ingrediente = input("Introduce el ingrediente que deseas: ")
    print("Pizza no vegetarina con mozarrella, tomate y ", end="")
    if ingrediente == "1":
        print("peperoni")
    elif ingrediente == "2":
        print("jamón")
    else:
        print("salmón")
```

Abrir la solución en Google Colab

4. Bucles

Ejercicio 4.1. Escribir un programa que pida al usuario una palabra y la muestre por pantalla 10 veces.

Solución

```
word = input("Introduce una palabra: ")
for i in range(10):
    print(word)
```

Abrir la solución en Google Colab

Ejercicio 4.2. Escribir un programa que pregunte al usuario su edad y muestre por pantalla todos los años que ha cumplido (desde 1 hasta su edad).

Solución

```
age = int(input("¿Cuántos años tienes? "))
for i in range(age):
    print("Has cumplido " + str(i+1) + " años")
```

Abrir la solución en Google Colab

Ejercicio 4.3. Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla todos los números impares desde 1 hasta ese número separados por comas.

Solución

```
n = int(input("Introduce un número entero positivo: "))
for i in range(1, n+1, 2):
    print(i, end=", ")
```

Abrir la solución en Google Colab

Ejercicio 4.4. Escribir un programa que pida al usuario un número entero positivo y muestre por pantalla la cuenta atrás desde ese número hasta cero separados por comas.

Solución

```
n = int(input("Introduce un número entero positivo: "))
for i in range(n, -1, -1):
    print(i, end=", ")
```

Abrir la solución en Google Colab

Ejercicio 4.5. Escribir un programa que pregunte al usuario una cantidad a invertir, el interés anual y el número de años, y muestre por pantalla el capital obtenido en la inversión cada año que dura la inversión.

Solución

```
amount = float(input("¿Cantidad a invertir? "))
interest = float(input("¿Interés porcentual anual? "))
years = int(input("¿Años?"))
for i in range(years):
    amount *= 1 + interest / 100
    print("Capital tras " + str(i+1) + " años: " + str(round(amount,
    ↪  2)))
```

Abrir la solución en Google Colab

Ejercicio 4.6. Escribir un programa que pida al usuario un número entero y muestre por pantalla un triángulo rectángulo como el de más abajo, de altura el número introducido.

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

Solución

1

```
n = int(input("Introduce la altura del triángulo (entero positivo):
↪  "))
for i in range(n):
    for j in range(i+1):
        print("*", end="")
    print("")
```

2

```
n = int(input("Introduce la altura del triángulo (entero positivo):
↪  "))
for i in range(n):
   print("*"*(i+1))
```

Abrir la solución en Google Colab

Ejercicio 4.7. Escribir un programa que muestre por pantalla la tabla de multiplicar del 1 al 10.

Solución

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i*j, end="\t")
    print("")
```

Abrir la solución en Google Colab

Ejercicio 4.8. Escribir un programa que pida al usuario un número entero y muestre por pantalla un triángulo rectángulo como el de más abajo.

```
1
3 1
5 3 1
7 5 3 1
9 7 5 3 1
```

Solución

```
n = int(input("Introduce la altura del triángulo (entero positivo):
↪  "))
for i in range(1, n+1, 2):
    for j in range(i, 0, -2):
        print(j, end=" ")
    print("")
```

Abrir la solución en Google Colab

Ejercicio 4.9. Escribir un programa que almacene la cadena de caracteres `contraseña` en una variable, pregunte al usuario por la contraseña hasta que introduzca la contraseña correcta.

Solución

```
key = "contraseña"
password =""
while password != key:
    password = input("Introduce la contraseña: ")
print("Contraseña correcta")
```

Abrir la solución en Google Colab

Ejercicio 4.10. Escribir un programa que pida al usuario un número entero y muestre por pantalla si es un número primo o no.

Solución

1

```
n = int(input("Introduce un número entero positivo mayor que 2: "))
i = 2
while n % i != 0:
    i += 1
if i == n:
    print(str(n) + " es primo")
else:
    print(str(n) + " no es primo")
```

2

```
n = int(input("Introduce un número entero positivo mayor que 2: "))
for i in range(2, n):
    if n % i == 0:
        break
if (i + 1)  == n:
    print(str(n) + " es primo")
else:
    print(str(n) + " no es primo")
```

Abrir la solución en Google Colab

Ejercicio 4.11. Escribir un programa que pida al usuario una palabra y luego muestre por pantalla una a una las letras de la palabra introducida empezando por la última.

Solución

```
word = input("Introduce una palabra: ")
for i in range(len(word)-1, -1, -1):
    print(word[i])
```

Abrir la solución en Google Colab

Ejercicio 4.12. Escribir un programa en el que se pregunte al usuario por una frase y una letra, y muestre por pantalla el número de veces que aparece la letra en la frase.

Solución

```
frase = input("Introduce una frase: ")
letra = input("Introduce una letra")
contador = 0
for i in frase:
    if i == letra:
        contador += 1
print("La letra '%s' aparece %2i veces en la frase '%s'." % (letra,
 ↪  contador, frase))
```

Abrir la solución en Google Colab

Ejercicio 4.13. Escribir un programa que muestre el eco de todo lo que el usuario introduzca hasta que el usuario escriba “salir” que terminará.

Solución

```
while True:
    frase = input("Introduce algo: ")
    if frase == "salir":
        break
    print(frase)
```

Abrir la solución en Google Colab

5. Listas y Tuplas

Ejercicio 5.1. Escribir un programa que almacene las asignaturas de un curso (por ejemplo Matemáticas, Física, Química, Historia y Lengua) en una lista y la muestre por pantalla.

Solución

```
subjects = ["Matemáticas", "Física", "Química", "Historia",
 ↪  "Lengua"]
print(subjects)
```

Abrir la solución en Google Colab

Ejercicio 5.2. Escribir un programa que almacene las asignaturas de un curso (por ejemplo Matemáticas, Física, Química, Historia y Lengua) en una lista y la muestre por pantalla el mensaje `Yo estudio <asignatura>`, donde `<asignatura>` es cada una de las asignaturas de la lista.

Solución

```
subjects = ["Matemáticas", "Física", "Química", "Historia",
 ↪  "Lengua"]
for subject in subjects:
    print("Yo estudio " + subject)
```

Abrir la solución en Google Colab

Ejercicio 5.3. Escribir un programa que almacene las asignaturas de un curso (por ejemplo Matemáticas, Física, Química, Historia y Lengua) en una lista, pregunte al usuario la nota que ha sacado en cada asignatura, y después las muestre por pantalla con el mensaje `En <asignatura> has sacado <nota>` donde `<asignatura>` es cada una des las asignaturas de la lista y `<nota>` cada una de las correspondientes notas introducidas por el usuario.

Solución

```
subjects = ["Matemáticas", "Física", "Química", "Historia",
↪  "Lengua"]
scores = []
for subject in subjects:
    score = input("¿Qué nota has sacado en " + subject + "?")
    scores.append(score)
for i in range(len(subjects)):
    print("En " + subjects[i] + " has sacado " + scores[i])
```

Abrir la solución en Google Colab

Ejercicio 5.4. Escribir un programa que pregunte al usuario los números ganadores de la lotería primitiva, los almacene en una lista y los muestre por pantalla ordenados de menor a mayor.

Solución

1

```
awarded = []
for i in range(6):
    awarded.append(int(input("Introduce un número ganador: ")))
awarded.sort()
print("Los números ganadores son " + str(awarded))
```

Abrir la solución en Google Colab

Ejercicio 5.5. Escribir un programa que almacene en una lista los números del 1 al 10 y los muestre por pantalla en orden inverso separados por comas.

Solución

1

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
for i in range(1, 11):
    print(numbers[-i], end=", ")
```

2

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers.reverse()
for number in numbers:
    print(number, end=", ")
```

Abrir la solución en Google Colab

Ejercicio 5.6. Escribir un programa que almacene las asignaturas de un curso (por ejemplo Matemáticas, Física, Química, Historia y Lengua) en una lista, pregunte al usuario la nota que ha sacado en cada asignatura y elimine de la lista las asignaturas aprobadas. Al final el programa debe mostrar por pantalla las asignaturas que el usuario tiene que repetir.

Solución

1

```
subjects = ["Matemáticas", "Física", "Química", "Historia",
 ↪  "Lengua"]
passed = []
for subject in subjects:
    score = float(input("¿Qué nota has sacado en " + subject + "?"))
    if score >= 5:
        passed.append(subject)
for subject in passed:
    subjects.remove(subject)
print("Tienes que repetir " + str(subjects))
```

2

```
subjects = ["Matemáticas", "Física", "Química", "Historia",
 ↪  "Lengua"]
for i in range(len(subjects)-1, -1, -1):
    score = float(input("¿Qué nota has sacado en " + subjects[i] +
 ↪  "?"))
    if score >= 5:
        subjects.pop(i)
print("Tienes que repetir " + str(subjects))
```

Abrir la solución en Google Colab

Ejercicio 5.7. Escribir un programa que almacene el abecedario en una lista, elimine de la lista las letras que ocupen posiciones múltiplos de 3, y muestre por pantalla la lista resultante.

Solución

1

```
alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
 ↪  'l', 'm', 'n', 'ñ', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
 ↪  'x', 'y', 'z']
for i in range(len(alphabet), 1, -1):
    if i % 3 == 0:
        alphabet.pop(i-1)
print(alphabet)
```

2

```
alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
 ↪  'l', 'm', 'n', 'ñ', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
 ↪  'x', 'y', 'z']
for i in range(len(alphabet), 1, -1):
    if i % 3 == 0:
        alphabet.pop(i-1)
print(alphabet)
```

Abrir la solución en Google Colab

Ejercicio 5.8. Escribir un programa que pida al usuario una palabra y muestre por pantalla si es un palíndromo.

Solución

```
word = input("Introduce una palabra: ")
reversed_word = word
word = list(word)
reversed_word = list(reversed_word)
reversed_word.reverse()
if word == reversed_word:
    print("Es un palíndromo")
else:
    print("No es un palíndromo")
```

Abrir la solución en Google Colab

Ejercicio 5.9. Escribir un programa que pida al usuario una palabra y muestre por pantalla el número de veces que contiene cada vocal.

Solución

```
word = input("Introduce una palabra: ")
vocals = ['a', 'e', 'i', 'o', 'u']
for vocal in vocals:
    times = 0
    for letter in word:
        if letter == vocal:
            times += 1
    print("La vocal " + vocal + " aparece " + str(times) + " veces")
```

Abrir la solución en Google Colab

Ejercicio 5.10. Escribir un programa que almacene en una lista los siguientes precios, 50, 75, 46, 22, 80, 65, 8, y muestre por pantalla el menor y el mayor de los precios.

Solución

```
prices = [50, 75, 46, 22, 80, 65, 8]
min = max = prices[0]
for price in prices:
    if price < min:
        min = price
    elif price > max:
        max = price
print("El mínimo es " + str(min))
print("El máximo es " + str(max))
```

Abrir la solución en Google Colab

Ejercicio 5.11. Escribir un programa que almacene los vectores (1,2,3) y (-1,0,2) en dos listas y muestre por pantalla su producto escalar.

Solución

```
a = (1, 2, 3)
b = (-1, 0, 2)
product = 0
for i in range(len(a)):
    product += a[i]*b[i]
print("El producto de los vectores" + str(a) + " y " + str(b) + " es
 ↪  " + str(product))
```

Abrir la solución en Google Colab

Ejercicio 5.12. Escribir un programa que almacene las matrices

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad y \quad B = \begin{pmatrix} -1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

en una lista y muestre por pantalla su producto.
Nota: Para representar matrices mediante listas usar listas anidadas, representando cada vector fila en una lista.

Solución

```
a = ((1, 2, 3),
     (4, 5, 6))
b = ((-1, 0),
     (0, 1),
     (1,1))
result = [[0,0],
          [0,0]]
for i in range(len(a)):
    for j in range(len(b[0])):
        for k in range(len(b)):
            result[i][j] += a[i][k] * b[k][j]
for i in range(len(result)):
    result[i] = tuple(result[i])
result = tuple(result)
for i in range(len(result)):
    print(result[i])
```

Abrir la solución en Google Colab

Ejercicio 5.13. Escribir un programa que pregunte por una muestra de números, separados por comas, los guarde en una lista y muestre por pantalla su media y desviación típica.

Solución

```
sample = input("Introduce una muestra de números separados por
 ↪  comas: ")
sample = sample.split(',')
n = len(sample)
for i in range(n):
    sample[i] = int(sample[i])
sample = tuple(sample)
sum = 0
sumsq = 0
for i in sample:
    sum += i
    sumsq += i**2
mean = sum/n
stdev = (sumsq/n-mean**2)**(1/2)
print('La media es', mean, ', y la desviación típica es', stdev)
```

Abrir la solución en Google Colab

6. Diccionarios

Ejercicio 6.1. Escribir un programa que guarde en una variable el diccionario `{'Euro':'€', 'Dollar':'$', 'Yen':'¥'}`, pregunte al usuario por una divisa y muestre su símbolo o un mensaje de aviso si la divisa no está en el diccionario.

Solución

1

```
monedas = {'Euro':'€', 'Dollar':'$', 'Yen':'¥'}
moneda = input("Introduce una divisa: ")
print(monedas.get(moneda.title(), "La divisa no está."))
```

2

```
monedas = {'Euro':'€', 'Dollar':'$', 'Yen':'¥'}
moneda = input("Introduce una divisa: ")
if moneda.title() in monedas:
    print(monedas[moneda.title()])
else:
    print("La divisa no está.")
```

Abrir la solución en Google Colab

Ejercicio 6.2. Escribir un programa que pregunte al usuario su nombre, edad, dirección y teléfono y lo guarde en un diccionario. Después debe mostrar por pantalla el mensaje `<nombre> tiene <edad> años, vive en <dirección> y su número de teléfono es <teléfono>`.

Solución

```
nombre = input('¿Cómo te llamas? ')
edad = input('¿Cuántos años tienes? ')
direccion = input('¿Cuál es tu dirección? ')
telefono = input('¿Cuál es tu número de teléfono? ')
persona = {'nombre': nombre, 'edad': edad, 'direccion': direccion,
 ↪  'telefono': telefono}
print(persona['nombre'], 'tiene', persona['edad'], 'años, vive en',
 ↪  persona['direccion'], 'y su número de teléfono es',
 ↪  persona['telefono'])
```

Abrir la solución en Google Colab

Ejercicio 6.3. Escribir un programa que guarde en un diccionario los precios de las frutas de la tabla, pregunte al usuario por una fruta, un número de kilos y muestre por pantalla el precio de ese número de kilos de fruta. Si la fruta no está en el diccionario debe mostrar un mensaje informando de ello.

Fruta	Precio
Plátano	1.35
Manzana	0.80
Pera	0.85
Naranja	0.70

Solución

```
frutas = {'Plátano':1.35, 'Manzana':0.8, 'Pera':0.85, 'Naranja':0.7}
fruta = input('¿Qué fruta quieres? ').title()
kg = float(input('¿Cuántos kilos? '))
if fruta in frutas:
    print(kg, 'kilos de', fruta, 'valen', frutas[fruta]*kg, '€')
else:
    print("Lo siento, la fruta", fruta, "no está disponible.")
```

Abrir la solución en Google Colab

Ejercicio 6.4. Escribir un programa que pregunte una fecha en formato `dd/mm/aaaa` y muestre por pantalla la misma fecha en formato `dd de <mes> de aaaa` donde `<mes>` es el nombre del mes.

Solución

```
meses = {1:'enero', 2:'febrero', 3:'marzo', 4:'abril', 5:'mayo',
 ↪  6:'junio', 7:'julio', 8:'agosto', 9:'septiembre', 10:'octubre',
 ↪  11:'noviembre', 12:'diciembre'}
fecha = input('Introduce una fecha en formato dd/mm/aaaa: ')
fecha = fecha.split('/')
print(fecha[0], 'de', meses[int(fecha[1])], 'de', fecha[2])
```

Abrir la solución en Google Colab

Ejercicio 6.5. Escribir un programa que almacene el diccionario con los créditos de las asignaturas de un curso `{'Matemáticas': 6, 'Física': 4, 'Química': 5}` y después muestre por pantalla los créditos de cada asignatura en el formato `<asignatura> tiene <créditos> créditos`, donde `<asignatura>` es cada una de las asignaturas del curso, y `<créditos>` son sus créditos. Al final debe mostrar también el número total de créditos del curso.

Solución

```
curso = {'Matemáticas': 6, 'Física': 4, 'Química': 5}
total_creditos = 0
for asignatura, creditos in curso.items():
    print(asignatura, 'tiene', creditos, 'créditos')
    total_creditos += creditos
print('Número total de créditos del curso: ', total_creditos)
```

Abrir la solución en Google Colab

Ejercicio 6.6. Escribir un programa que cree un diccionario vacío y lo vaya llenado con información sobre una persona (por ejemplo nombre, edad, sexo, teléfono, correo electrónico, etc.) que se le pida al usuario. Cada vez que se añada un nuevo dato debe imprimirse el contenido del diccionario.

Solución

```
persona = {}
continuar = True
while continuar:
    clave = input('¿Qué dato quieres introducir? ')
    valor = input(clave + ': ')
    persona[clave] = valor
    print(persona)
    continuar = input('¿Quieres añadir más información (Si/No)? ')
    == "Si"
```

Abrir la solución en Google Colab

Ejercicio 6.7. Escribir un programa que cree un diccionario simulando una cesta de la compra. El programa debe preguntar el artículo y su precio y añadir el par al diccionario, hasta que el usuario decida terminar. Después se debe mostrar por pantalla la lista de la compra y el coste total, con el siguiente formato

Lista de la compra	
Artículo 1	Precio
Artículo 2	Precio
Artículo 3	Precio
...	...
Total	Coste

Solución

```
cesta = {}
continuar = True
while continuar:
    item = input('Introduce un artículo: ')
    precio = float(input('Introduce el precio de ' + item + ': '))
    cesta[item] = precio
    continuar = input('¿Quieres añadir artículos a la lista (Si/No)?
 ↪  ') == "Si"
coste = 0
print('Lista de la compra')
for item, precio in cesta.items():
    print(item, '\t', precio)
    coste += precio
print('Coste total: ', coste)
```

Abrir la solución en Google Colab

Ejercicio 6.8. Escribir un programa que cree un diccionario de traducción español-inglés. El usuario introducirá las palabras en español e inglés separadas por dos puntos, y cada par `<palabra>:<traducción>` separados por comas. El programa debe crear un diccionario con las palabras y sus traducciones. Después pedirá una frase en español y utilizará el diccionario para traducirla palabra a palabra. Si una palabra no está en el diccionario debe dejarla sin traducir.

Solución

```
diccionario = {}
palabras = input("Introduce la lista de palabras y traducciones en
 ↪  formato palabra:traducción separadas por comas: ")
for i in palabras.split(','):
    clave, valor = i.split(':')
    diccionario[clave] = valor
frase = input('Introduce una frase en español: ')
for i in frase.split():
    if i in diccionario:
        print(diccionario[i], end=' ')
    else:
        print(i, end=' ')
```

Abrir la solución en Google Colab

Ejercicio 6.9. Escribir un programa que gestione las facturas pendientes de cobro de una empresa. Las facturas se almacenarán en un diccionario donde la clave de cada factura será el número de factura y el valor el coste de la factura. El programa debe preguntar al usuario si quiere añadir una nueva factura, pagar una existente o terminar. Si desea añadir una nueva factura se preguntará por el número de factura y su coste y se añadirá al diccionario. Si se desea pagar una factura se preguntará por el número de factura y se eliminará del diccionario. Después de cada operación el programa debe mostrar por pantalla la cantidad cobrada hasta el momento y la cantidad pendiente de cobro.

Solución

```
facturas = {}
cobrado = 0
pendiente = 0
more = ''
while more != 'T':
    if more == 'A':
        clave = input('Introduce el número de la factura: ')
        coste = float(input('Introduce el coste de la factura: '))
        facturas[clave] = coste
        pendiente += coste
    if more == 'P':
        clave = input('Introduce el número de la factura a pagar: ')
        coste = facturas.pop(clave, 0)
        cobrado += coste
        pendiente -= coste
    print('Recaudado:', cobrado)
    print('Pendiente de cobro: ', pendiente)
    more = input('¿Quieres añadir una nueva factura (A), pagarla (P)
↪   o terminar (T)? ')
```

Abrir la solución en Google Colab

Ejercicio 6.10. Escribir un programa que permita gestionar la base de datos de clientes de una empresa. Los clientes se guardarán en un diccionario en el que la clave de cada cliente será su NIF, y el valor será otro diccionario con los datos del cliente (nombre, dirección, teléfono, correo, preferente), donde preferente tendrá el valor `True` si se trata de un cliente preferente. El programa debe preguntar al usuario por una opción del siguiente menú: (1) Añadir cliente, (2) Eliminar cliente, (3) Mostrar cliente, (4) Listar todos los clientes, (5) Listar clientes preferentes, (6) Terminar. En función de la opción elegida el programa tendrá que hacer lo siguiente:

1. Preguntar los datos del cliente, crear un diccionario con los datos y añadirlo a la base de datos.

2. Preguntar por el NIF del cliente y eliminar sus datos de la base de datos.
3. Preguntar por el NIF del cliente y mostrar sus datos.
4. Mostrar lista de todos los clientes de la base datos con su NIF y nombre.
5. Mostrar la lista de clientes preferentes de la base de datos con su NIF y nombre.
6. Terminar el programa.

Solución

```
clientes = {}
opcion = ''
while opcion != '6':
    if opcion == '1':
        nif = input('Introduce NIF del cliente: ')
        nombre = input('Introduce el nombre del cliente: ')
        direccion = input('Introduce la dirección del cliente: ')
        telefono = input('Introduce el teléfono del cliente: ')
        email = input('Introduce el correo electrónico del cliente:
↪  ')
        vip = input('¿Es un cliente preferente (S/N)? ')
        cliente = {'nombre':nombre, 'dirección':direccion,
↪  'teléfono':telefono, 'email':email, 'preferente':vip=='S'}
        clientes[nif] = cliente
    if opcion == '2':
        nif = input('Introduce NIF del cliente: ')
        if nif in clientes:
            del clientes[nif]
        else:
            print('No existe el cliente con el nif', nif)
    if opcion == '3':
        nif = input('Introduce NIF del cliente: ')
        if nif in clientes:
            print('NIF:', nif)
            for clave, valor in clientes[nif].items():
                print(clave.title() + ':', valor)
        else:
            print('No existe el cliente con el nif', nif)
    if opcion == '4':
        print('Lista de clientes')
        for clave, valor in clientes.items():
            print(clave, valor['nombre'])
    if opcion == '5':
        print('Lista de clientes preferentes')
        for clave, valor in clientes.items():
            if valor['preferente']:
                print(clave, valor['nombre'])
    opcion = input('Menú de opciones\n(1) Añadir cliente\n(2)
↪  Eliminar cliente\n(3) Mostrar cliente\n(4) Listar clientes\n(5)
↪  Listar clientes preferentes\n(6) Terminar\nElige una opción:')
```

Abrir la solución en Google Colab

Ejercicio 6.11. El directorio de los clientes de una empresa está organizado en una cadena de texto como la de más abajo, donde cada línea contiene la información del nombre, email, teléfono, nif, y el descuento que se le aplica. Las líneas se separan con el carácter de cambio de línea `\n` y la primera línea contiene los nombres de los campos con la información contenida en el directorio.

```
"nif;nombre;email;teléfono;descuento\n01234567L;Luis
 ↪ González;luisgonzalez@mail.com;656343576;12.5\n71476342J;Macarena
 ↪ Ramírez;macarena@mail.com;692839321;8\n63823376M;Juan José
 ↪ Martínez;juanjo@mail.com;664888233;5.2\n98376547F;Carmen
 ↪ Sánchez;carmen@mail.com;667677855;15.7"
```

Escribir un programa que genere un diccionario con la información del directorio, donde cada elemento corresponda a un cliente y tenga por clave su nif y por valor otro diccionario con el resto de la información del cliente. Los diccionarios con la información de cada cliente tendrán como claves los nombres de los campos y como valores la información de cada cliente correspondientes a los campos. Es decir, un diccionario como el siguiente

```
{'01234567L': {'nombre': 'Luis González', 'email':
 ↪ 'luisgonzalez@mail.com', 'teléfono': '656343576', 'descuento':
 ↪ 12.5}, '71476342J': {'nombre': 'Macarena Ramírez', 'email':
 ↪ 'macarena@mail.com', 'teléfono': '692839321', 'descuento': 8.0},
 ↪ '63823376M': {'nombre': 'Juan José Martínez', 'email':
 ↪ 'juanjo@mail.com', 'teléfono': '664888233', 'descuento': 5.2},
 ↪ '98376547F': {'nombre': 'Carmen Sánchez', 'email':
 ↪ 'carmen@mail.com', 'teléfono': '667677855', 'descuento': 15.7}}
```

Solución

```
# Cadena con los datos de los clientes de la empresa
datos_clientes =
 ↪  "nif;nombre;email;teléfono;descuento\n01234567L;Luis
 ↪  González;luisgonzalez@mail.com;656343576;12.5\n
 ↪  71476342J;Macarena
 ↪  Ramírez;macarena@mail.com;692839321;8\n63823376M;Juan José
 ↪  Martínez;juanjo@mail.com;664888233;5.2\n98376547F;Carmen
 ↪  Sánchez;carmen@mail.com;667677855;15.7"
# Dividimos la cadena por el caracter de cambio de línea \n y
 ↪  creamos una lista con las subcadenas
lista_clientes = datos_clientes.split('\n')
# Inicializamos el diccionario que va a contener el directorio de
 ↪  clientes a vacío.
directorio = {}
# Dividimos la cadena del primer elemento de la lista de clientes
 ↪  (que contienen los
# nombres de los campos) por el caracter ; y creamos una lista con
 ↪  los campos.
lista_campos = lista_clientes[0].split(';')
# Bucle iterativo para recorrer los elementos de la lista
 ↪  lista_clientes.
# la variable cliente recorre desde el segundo elemento hasta el
 ↪  último elemento de la lista
# (el primer elemento contiene los nombres de campo así que no
 ↪  corresponde a un cliente)
for i in lista_clientes[1:]:
    # Inicializamos el diccionario que va a contener los datos del
     ↪  cliente actual a vacío.
    cliente = {}
    # Dividimos la cadena i por el caracter ; y creamos una lista
     ↪  con las subcadenas con la
    # información del cliente
    lista_info = i.split(';')
    # Bucle iterativo para recorrer los campos y añadir los pares al
     ↪  diccionario del cliente.
    # j toma valores de 1 al número de campos menos 1. El primer
     ↪  elemento (posición 0) corresponde
    # al nif y no se añade al diccionario porque se utilizará
     ↪  después como clave en el diccionario
    # principal
    for j in range(1,len(lista_campos)):
        # Condicional. Si el campo actual es descuento convertimos
         ↪  su valor en real
        if lista_campos[j] == 'descuento':
            lista_info[j] = float(lista_info[j])
        cliente[lista_campos[j]] = lista_info[j]
    # Añadirmos un par al diccionario del directorio con la clave el
     ↪  nif del cliente y valor
    # el diccionario que acabamos de crear con el resto de sus
     ↪  datos.
    directorio[lista_info[0]] = cliente
# Mostramos el diccionario por pantalla
print(directorio)
```

Abrir la solución en Google Colab

7. Funciones

Ejercicio 7.1. Escribir una función que muestre por pantalla el saludo `¡Hola amiga!` cada vez que se la invoque.

Solución

```
def greet():
    """Función que muestra el saluco ¡Hola amiga! por pantalla."""
    print('¡Hola amiga!')
    return

greet()
```

Abrir la solución en Google Colab

Ejercicio 7.2. Escribir una función a la que se le pase una cadena `<nombre>` y muestre por pantalla el saludo `¡hola <nombre>!`.

Solución

```
def greet(nombre):
    """Función que muestra un saludo por pantalla.
    Parámetros
    nombre: Nombre del usuario
    Devuelve el saludo ¡Hola nombre!.
    """
    print('¡Hola ' + nombre +'!')
    return

greet('Alf')
greet('Python')
```

Abrir la solución en Google Colab

Ejercicio 7.3. Escribir una función que reciba un número entero positivo y devuelva su factorial.

Solución

```
def factorial(n):
    """Función que calcula el factorial de un número entero
    ↪ positivo.
    Parámetros
    n: Es un entero positivo.
    Devuelve el factorial de n.
    """
    tmp = 1
    for i in range(n):
        tmp *= i+1
    return tmp

print(factorial(4))
print(factorial(20))
```

Abrir la solución en Google Colab

Ejercicio 7.4. Escribir una función que calcule el total de una factura tras aplicarle el IVA. La función debe recibir la cantidad sin IVA y el porcentaje de IVA a aplicar, y devolver el total de la factura. Si se invoca la función sin pasarle el porcentaje de IVA, deberá aplicar un 21 %.

Solución

```
def invoice(amount, vat=21):
    """Función de aplica el IVA a una factura.
    Parametros
    amount: Es la cantidad sin IVA
    vat: Es el porcentaje de IVA
    Devuelve el total de la factura una vez aplicado el IVA.
    """
    return amount + amount*vat/100

print(invoice(1000,10))
print(invoice(1000))
```

Abrir la solución en Google Colab

Ejercicio 7.5. Escribir una función que calcule el área de un círculo y otra que calcule el volumen de un cilindro usando la primera función.

Solución

```
def circle_area(radius):
    """Función que calcula el area de un círculo.
    Parámetros
    radius: Es el radio del círculo.
    Devuelve el área del círculo de radio radius.
    """
    pi = 3.1415
    return pi*radius**2

def cilinder_volume(radius, high):
    """Función que calcula el volumen de un cilindro.
    Parámetros
    radius: Es el radio de la base del cilindro.
    high: Es la altura del cilindro.
    Devuelve el volumen del clindro de radio radius y altura high.
    """
    return circle_area(radius)*high

print(cilinder_volume(3,5))
```

Abrir la solución en Google Colab

Ejercicio 7.6. Escribir una función que reciba una muestra de números en una lista y devuelva su media.

Solución

1

```python
def mean(sample):
    """Función que calcula la media de una muestra de números.
    Parámetros
    sample: Es una lista de números
    Devuelve la media de los números en sample.
    """
    return sum(sample)/len(sample)

print(mean([1, 2, 3, 4, 5]))
print(mean([2.3, 5.7, 6.8, 9.7, 12.1, 15.6]))
```

2

```python
def mean(*sample):
    """Función que calcula la media de una muestra de números.
    Parámetros
    *sample: Secuencia de números separados por comas.
    Devuelve la media de los números en *sample.
    """
    return sum(sample)/len(sample)

print(mean(1, 2, 3, 4, 5))
print(mean(2.3, 5.7, 6.8, 9.7, 12.1, 15.6))
```

Abrir la solución en Google Colab

Ejercicio 7.7. Escribir una función que reciba una muestra de números en una lista y devuelva otra lista con sus cuadrados.

Solución

1

```python
def square(sample):
    """Función que calcula los cuadrados de una lista de números.
    Parámetros
    sample: Es una lista de números
    Devuelve una lista con los cuadrados de los números de la lista
    ↪ sample.
    """
    list = []
    for i in sample:
        list.append(i**2)
    return list

print(square([1, 2, 3, 4, 5]))
print(square([2.3, 5.7, 6.8, 9.7, 12.1, 15.6]))
```

2

```python
def square(*sample):
    """Función que calcula los cuadrados de una lista de números.
    Parámetros
    *sample: Es una secuencia de números separados por comas.
    Devuelve una lista con los cuadrados de los números de sample.
    """
    list = []
    for i in sample:
        list.append(i**2)
    return list

print(square(1, 2, 3, 4, 5))
print(square(2.3, 5.7, 6.8, 9.7, 12.1, 15.6))
```

Abrir la solución en Google Colab

Ejercicio 7.8. Escribir una función que reciba una muestra de números en una lista y devuelva un diccionario con su media, varianza y desviación típica.

Solución

1

```python
def square(sample):
    """Función que calcula los cuadrados de una lista de números.
    Parámetros
    sample: Es una lista de números
    Devuelve una lista con los cuadrados de los números de la lista
    sample.
    """
    list = []
    for i in sample:
        list.append(i**2)
    return list

def statistics(sample):
    """Función que calcula la media, varianza y desviación típica de
    una muestra de números.
    Parámetros
    sample: Es una lista de números
    Devuelve un diccionario con la media, varianza y desviación
    típica de los números en sample.
    """
    stat = {}
    stat['media'] = sum(sample)/len(sample)
    stat['varianza'] =
    sum(square(sample))/len(sample)-stat['media']**2
    stat['desviacion tipica'] = stat['varianza']**0.5
    return stat

print(statistics([1, 2, 3, 4, 5]))
print(statistics([2.3, 5.7, 6.8, 9.7, 12.1, 15.6]))
```

2

```python
def square(*sample):
    """Función que calcula los cuadrados de una lista de números.
    Parámetros
    sample: Es una secuencia de números separados por comas.
    Devuelve una lista con los cuadrados de los números de sample.
    """
    list = []
    for i in sample:
        list.append(i**2)
    return list

def statistics(*sample):
    """Función que calcula la media, varianza y desviación típica de
    ↪ una muestra de números.
    Parámetros
    sample: Es una secuencia de números separados por comas.
    Devuelve un diccionario con la media, varianza y desviación
↪   típica de los números en sample.
    """
    stat = {}
    stat['media'] = sum(sample)/len(sample)
    stat['varianza'] =
↪   sum(square(*sample))/len(sample)-stat['media']**2
    stat['desviacion tipica'] = stat['varianza']**0.5
    return stat

print(statistics(1, 2, 3, 4, 5))
print(statistics(2.3, 5.7, 6.8, 9.7, 12.1, 15.6))
```

Abrir la solución en Google Colab

Ejercicio 7.9. Escribir una función que calcule el máximo común divisor de dos números y otra que calcule el mínimo común múltiplo.

Solución

```
def mcd(n, m):
    """Función que calcula el máximo común divisor de dos números.
    Parámetros:
        - n: Es un número entero.
        - m: Es un número entero.
    Devuelve:
        El máximo común divisor de n y m.
    """
    rest = 0
    while(m > 0):
        rest = m
        m = n % m
        n = rest
    return n

def mcm(n, m):
    """Función que calcula el mínimo común múltiplo de dos números.
    Parámetros:
        - n: Es un número entero.
        - m: Es un número entero.
    Devuelve:
        El mínimo común múltiplo de n y m.
    """
    if n > m:
        greater = n
    else:
        greater = m
    while (greater % n != 0) or (greater % m != 0):
        greater += 1
    return greater

print(mcd(24,36))
print(mcm(24,36))
```

Abrir la solución en Google Colab

Ejercicio 7.10. Escribir una función que convierta un número decimal en binario y otra que convierta un número binario en decimal.

Solución

```python
def to_decimal(n):
    """Función que convierte un número binario en decimal.
    Parámetros:
        - n: Es una cadena de ceros y unos.
    Devuelve:
        El número decimal correspondiente a n.
    """
    n = list(n)
    n.reverse()
    decimal = 0
    for i in range(len(n)):
        decimal += int(n[i]) * 2 ** i
    return decimal

def to_binary(n):
    """Función que convierte un número decimal en binario.
    Parámetros:
        - n: Es un número entero.
    Devuelve:
        El número binario correspondiente a n.
    """
    binary = []
    while n > 0:
        binary.append(str(n % 2))
        n //= 2
    binary.reverse()
    return ''.join(binary)

print(to_decimal('10110'))
print(to_binary(22))
print(to_decimal(to_binary(22)))
print(to_binary(to_decimal('10110')))
```

Abrir la solución en Google Colab

Ejercicio 7.11. Escribir un programa que reciba una cadena de caracteres y devuelva un diccionario con cada palabra que contiene y su frecuencia. Escribir otra función que reciba el diccionario generado con la función anterior y devuelva una tupla con la palabra más repetida y su frecuencia.

Solución

```
def count_words(text):
    """Función que cuenta el número de veces que aparece cada
    ↪ palabra en un texto.
    Parámetros:
        - text: Es una cadena de caracteres.
    Devuelve:
        Un diccionario con pares palabra:frecuencia con las palabras
↪   contenidas en el texto y su frecuencia.
    """
    text = text.split()
    words = {}
    for i in text:
        if i in words:
            words[i] += 1
        else:
            words[i] = 1
    return words

def most_repeated(words):
    max_word = ''
    max_freq = 0
    for word, freq in words.items():
        if freq > max_freq:
            max_word = word
            max_freq = freq
    return max_word, max_freq

text = 'Como quieres que te quiera si el que quiero que me quiera no
↪   me quiere como quiero que me quiera'
print(count_words(text))
print(most_repeated(count_words(text)))
```

Abrir la solución en Google Colab

8. Programación Funcional

Ejercicio 8.1. Escribir una función que aplique un descuento a un precio y otra que aplique el IVA a un precio. Escribir una tercera función que reciba un diccionario con los precios y porcentajes de una cesta de la compra, y una de las funciones anteriores, y utilice la función pasada para aplicar los descuentos o el IVA a los productos de la cesta y devolver el precio final de la cesta.

Solución

```python
def apply_discount(price, discount):
    '''
    Función que aplica un descuento a una cantidad.
    Parámetros:
        price: Es un valor real con el precio al que aplicar el
 ↪  descuento.
        discount: Es el porcentaje a descontar.
    Devuelve:
        El precio final tras aplicar el descuento.
    '''
    return price - price * discount / 100

def apply_IVA(price, percentage):
    '''
    Función que aplica un IVA a una cantidad.
    Parámetros:
        price: Es un valor real con el precio al que aplicar el IVA.
        percentage: Es el porcentaje del IVA a aplicar.
    Devuelve:
        El precio final tras aplicar el IVA.
    '''
    return price + price * percentage / 100

def price_basket(basket, function):
    '''
    Función que calcula el precio de una cesta de la compra una vez
 ↪  aplicada una función a los precios iniciales.
    Parámetros:
        basket: Es un diccionario formado por pares
 ↪  precio:descuento.
        function: Es una función que toma dos valores reales y
 ↪  devuelve otro. Normalmente para aplicar descuentos o IVA.
    Devuelve:
        El precio final de la cesta de la compra una vez aplicada la
 ↪  función sobre los precios iniciales.
    '''
    total = 0
    for price, discount in basket.items():
        total += function(price, discount)
    return total

print('El precio de la compra tras aplicar los descuentos es: ',
 ↪  price_basket({1000:20, 500:10, 100:1}, apply_discount))
print('El precio de la compra tras aplicar el IVA es: ',
 ↪  price_basket({1000:20, 500:10, 100:1}, apply_IVA))
```

Abrir la solución en Google Colab

Ejercicio 8.2. Escribir una función que simule una calculadora científica que permita calcular el seno, coseno, tangente, exponencial y logaritmo neperiano. La función preguntará al usuario el valor y la función a aplicar, y mostrará por pantalla una tabla con los enteros de 1 al valor introducido y el resultado de aplicar la función a esos enteros.

Solución

1

```
from math import sin, cos, tan, exp, log

def apply_function(f, n):
    '''
    Función que aplica una función a los enteros desde 1 hasta n.
    Parámetros:
        f: Es una función que recibe un número real y devuelve otro.
        n: Es un número entero positivo.
    Devuelve:
        Un diccionario con los pares i:f(i) para cada valor entero i
 ↪  de 1 a n.
    '''
    functions = {'sin':sin, 'cos':cos, 'tan':tan, 'exp':exp,
 ↪  'log':log}
    result = {}
    for i in range(1, n+1):
        result[i] = functions[f](i)
    return result

def calculator():
    '''
    Función que aplica una función seleccionada por el usuario
 ↪  (seno, coseno, tangente, exponencial o logarítmo) a la lista de
 ↪  enteros desde 1 hasta n.
    Imprime por pantalla una tabla con la secuencia de enteros y el
 ↪  resultado de aplicarles la función introducida.
    Parámetros:
        f: Es una cadena con la función a aplicar (sin, cos, tan,
 ↪  exp o log).
        n: Es un entero positivo.
    '''
    f = input('Introduce la función a aplicar (sin, cos, tan, exp,
 ↪  log): ')
    n = int(input('Introduce un entero positivo: '))
    for i, j in apply_function(f, n).items():
        print (i, '\t', j)
    return

calculator()
```

2

```python
from math import sin, cos, tan, exp, log

def apply_function(f, n):
    '''
    Función que aplica una función a los enteros desde 1 hasta n.
    Parámetros:
        f: Es una función que recibe un número real y devuelve otro.
        n: Es un número entero positivo.
    Devuelve:
        Un diccionario con los pares i:f(i) para cada valor entero i
 ↪  de 1 a n.
    '''
    result = {}
    for i in range(1, n+1):
        result[i] = eval(f + '(' + str(i) + ')')
    return result

def calculator():
    '''
    Función que aplica una función seleccionada por el usuario
 ↪  (seno, coseno, tangente, exponencial o logarítmo) a la lista de
 ↪  enteros desde 1 hasta n.
    Imprime por pantalla una tabla con la secuencia de enteros y el
 ↪  resultado de aplicarles la función introducida.
    Parámetros:
        f: Es una cadena con la función a aplicar (sin, cos, tan,
 ↪  exp o log).
        n: Es un entero positivo.
    '''
    f = input('Introduce la función a aplicar (sin, cos, tan, exp,
 ↪  log): ')
    n = int(input('Introduce un entero positivo: '))
    for i, j in apply_function(f, n).items():
        print (i, '\t', j)
    return

calculator()
```

3

```python
from math import sin, cos, tan, exp, log

def calculator():
    '''
    Función que aplica una función seleccionada por el usuario
    (seno, coseno, tangente, exponencial o logarítmo) a la lista de
    enteros desde 1 hasta n.
    Imprime por pantalla una tabla con la secuencia de enteros y el
    resultado de aplicarles la función introducida.
    Parámetros:
        f: Es una cadena con la función a aplicar (sin, cos, tan,
    exp o log).
        n: Es un entero positivo.
    '''
    functions = {'sin':sin, 'cos':cos, 'tan':tan, 'exp':exp,
    'log':log}
    f = input('Introduce la función a aplicar (sin, cos, tan, exp,
    log): ')
    n = int(input('Introduce un entero positivo: '))
    results = [functions[f](x) for x in range(1, n+1)]
    for i in range(n):
        print (i + 1, '\t', results[i])
    return

calculator()
```

Abrir la solución en Google Colab

Ejercicio 8.3. Escribir una función que reciba otra función y una lista, y devuelva otra lista con el resultado de aplicar la función dada a cada uno de los elementos de la lista.

Solución

1

```python
def aplica_funcion_lista(funcion, lista):
    '''Función que aplica una función a todos los elementos de una
    ↪ lista.

    Parámetros:
        funcion: Es una función.
        lista: Es una lista con valores que se pasarán como
    ↪ argumentos a funcion.
    Devuelve:
        Una lista con el resultado de aplicar la función a los
    ↪ valores de la lista.
    '''
    l = []
    for i in lista:
        l.append(funcion(i))
    return l

def cuadrado(n):
    return n * n

print(aplica_funcion_lista(cuadrado, [1, 2, 3, 4]))
```

Abrir la solución en Google Colab

Ejercicio 8.4. Escribir una función que reciba otra función booleana y una lista, y devuelva otra lista con los elementos de la lista que devuelvan `True` al aplicarles la función booleana.

Solución

1

```python
def filtra_lista(funcion, lista):
    '''Función que filtra los elementos de una lista que devuelven
     ↪  true al aplicarle una función booleana.

    Parámetros:
        - funcion: Es una función booleana (devuleve true o false)
        - lista: Es una lista con valores que se pasarán como
 ↪  argumentos a funcion.
    Devuelve:
        Una lista con los elementos de la lista que devuelven true
 ↪  al aplicarles la función booleana.
    '''
    l = []
    for i in lista:
        if funcion(i):
            l.append(i)
    return l

def par(n):
    return n % 2 == 0

print(filtra_lista(par, [1, 2, 3, 4, 5, 6]))
```

Abrir la solución en Google Colab

Ejercicio 8.5. Escribir una función que reciba una frase y devuelva un diccionario con las palabras que contiene y su longitud.

Solución

1

```python
def length_words(sentence):
    '''
    Función que recibe una frase y devuelve un diccionario con las
 ↪  palabras que contiene y su longitud.
    Parámetros:
        sentence: Es una cadena de caracteres con una frase.
    Devuelve:
        Un diccionario con pares palabra:longitud donde palabra son
 ↪  las palabras que contiene la frase sentence.
    '''
    words = sentence.split()
    lengths = map(len, words)
    return dict(zip(words, lengths))

print(length_words('Welcome to Python'))
```

2

```python
def length_words(sentence):
    '''
    Función que recibe una frase y devuelve un diccionario con las
 ↪  palabras que contiene y su longitud.
    Parámetros:
        sentence: Es una cadena de caracteres con una frase.
    Devuelve:
        Un diccionario con pares palabra:longitud donde palabra son
 ↪  las palabras que contiene la frase sentence.
    '''
    return {word:len(word) for word in sentence.split()}

print(length_words('Welcome to Python'))
```

Abrir la solución en Google Colab

Ejercicio 8.6. Escribir una función reciba una lista de notas y devuelva la lista de calificaciones correspondientes a esas notas.

Solución

1

```
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def apply_grade(scores):
    '''
    Función que devuelve la calificación correspondiente a las notas
 ↪  de una lista dada.
    Parámetros:
        scores: Es una lista de valores reales entre 0 y 10.
    Devuelve
        La lista de calificaciones correspondiente a las notas de
 ↪  scores.
    '''
    return list(map(grade, scores))

print(apply_grade([6.5, 5, 3.4, 8.2, 2.1, 9.7, 10]))
```

```
2
```

```python
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def apply_grade(scores):
    '''
    Función que devuelve la calificación correspondiente a las notas
    de una lista dada.
    Parámetros:
        scores: Es una lista de valores reales entre 0 y 10.
    Devuelve
        La lista de calificaciones correspondiente a las notas de
    scores.
    '''
    return [grade(x) for x in scores]

print(apply_grade([6.5, 5, 3.4, 8.2, 2.1, 9.7, 10]))
```

Abrir la solución en Google Colab

Ejercicio 8.7. Escribir una función reciba un diccionario con las asignaturas y las notas de un alumno y devuelva otro diccionario con las asignaturas en mayúsculas y las calificaciones correspondientes a las notas.

Solución

1

```
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def apply_grade(scores):
    '''
    Función que recibe un diccionario de asignaturas y notas y
 ↪  devuelve otro con las asignaturas en mayúsculas y las
 ↪  calificaciones correspondientes a las notas.
    Parámetros:
        scores: Es un diccionario con pares asignatura:nota donde
 ↪  nota es un valor real entre 0 y 10.
    Devuelve
        Un diccionario con pares ASIGNATURA:calificación, donde
 ↪  calificación es la calificación correspondiente a la nota de la
 ↪  asignatura.
    '''
    subjects = map(str.upper, scores.keys())
    grades = map(grade, scores.values())
    return dict(zip(subjects, grades))

print(apply_grade({'Matemáticas':6.5, 'Física':5, 'Química':3.4,
 ↪  'Economía':8.2, 'Historia':9.7, 'Programación':10}))
```

2

```python
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def apply_grade(scores):
    '''
    Función que recibe un diccionario de asignaturas y notas y
 ↪  devuelve otro con las asignaturas en mayúsculas y las
 ↪  calificaciones correspondientes a las notas.
    Parámetros:
        scores: Es un diccionario con pares asignatura:nota donde
 ↪  nota es un valor real entre 0 y 10.
    Devuelve
        Un diccionario con pares ASIGNATURA:calificación, donde
 ↪  calificación es la calificación correspondiente a la nota de la
 ↪  asignatura.
    '''
    return {subject.upper():grade(score) for subject, score in
     ↪  scores.items()}

print(apply_grade({'Matemáticas':6.5, 'Física':5, 'Química':3.4,
 ↪  'Economía':8.2, 'Historia':9.7, 'Programación':10}))
```

Abrir la solución en Google Colab

Ejercicio 8.8. Escribir una función reciba un diccionario con las asignaturas y las notas de un alumno y devuelva otro diccionario con las asignaturas en mayúsculas y las calificaciones correspondientes a las notas aprobadas.

Solución

1

```
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def passed_subject(subject):
    '''
    Función que recibe una tupla con una asignatura y su nota y
 ↪  devuelve True si la asignatura está aprobada o False si está
 ↪  suspensa.abs
    Parámetros:
        subject: Es una tupla (asignatura, nota) donde nota es un
 ↪  valor real entre 0 y 10.
    Devuelve: True si la nota es mayor o igual que 5 y False si
 ↪  no.abs
    '''
    return (subject[1] >= 5)

def apply_grade(scores):
    '''
    Función que recibe un diccionario de asignaturas y notas y
 ↪  devuelve otro con las asignaturas en mayúsculas y las
 ↪  calificaciones correspondientes a las notas.
    Parámetros:
        scores: Es un diccionario con pares asignatura:nota donde
 ↪  nota es un valor real entre 0 y 10.
    Devuelve
        Un diccionario con pares ASIGNATURA:calificación, donde
 ↪  calificación es la calificación correspondiente a la nota de la
 ↪  asignatura.
    '''
    passed = dict(filter(passed_subject, scores.items()))
    subjects = map(str.upper, passed.keys())
    grades = map(grade, passed.values())
    return dict(zip(subjects, grades))

print(apply_grade({'Matemáticas':6.5, 'Física':5, 'Química':3.4,
 ↪  'Economía':8.2, 'Historia':9.7, 'Programación':10}))
```

2

```
def grade(score):
    '''
    Función que devuelve la calificación correspondiente a una nota.
    Parámetros:
        score: Es un valor real entre 0 y 10.
    Devuelve:
        La calificación correspondiente a la nota score.
    '''
    if score < 5:
        return 'SS'
    elif score < 7:
        return 'AP'
    elif score < 9:
        return 'NT'
    elif score < 10:
        return 'SB'
    else:
        return 'MH'

def apply_grade(scores):
    '''
    Función que recibe un diccionario de asignaturas y notas y
 ↪  devuelve otro con las asignaturas en mayúsculas y las
 ↪  calificaciones correspondientes a las notas.
    Parámetros:
        scores: Es un diccionario con pares asignatura:nota donde
 ↪  nota es un valor real entre 0 y 10.
    Devuelve
        Un diccionario con pares ASIGNATURA:calificación, donde
 ↪  calificación es la calificación correspondiente a la nota de la
 ↪  asignatura.
    '''
    return {subject.upper():grade(score) for subject, score in
     ↪  scores.items() if score >= 5}

print(apply_grade({'Matemáticas':6.5, 'Física':5, 'Química':3.4,
 ↪  'Economía':8.2, 'Historia':9.7, 'Programación':10}))
```

Abrir la solución en Google Colab

Ejercicio 8.9. Escribir una función que calcule el módulo de un vector.

Solución

1

```
def sum_square(x, y):
    '''
    Función que recibe dos valores y calcula la suma del primero más
 ↪  el cuadrado del segundo.
    Parámetros:
        x: Es un número real.
        y: Es un número real.
    Devuelve:
        x + y²
    '''
    return x + y ** 2

def module(v):
    '''
    Función que calcula el módulo de un vector.
    Parámetros:
        v: Una tupla de números reales.
    Devuelve:
        El módulo del vector v.
    '''
    from functools import reduce
    return reduce(sum_square, v, 0) ** 0.5

print(module((3, 4)))
print(module((1, 2, 3)))
```

2

```
def module(v):
    '''
    Función que calcula el módulo de un vector.
    Parámetros:
        v: Una tupla de números reales.
    Devuelve:
        El módulo del vector v.
    '''
    return sum([x ** 2 for x in v]) ** 0.5

print(module((3, 4)))
print(module((1, 2, 3)))
```

Abrir la solución en Google Colab

Ejercicio 8.10. Una inmobiliaria de una ciudad maneja una lista de inmuebles como la siguiente:

```
[{'año': 2000, 'metros': 100, 'habitaciones': 3, 'garaje': True, 'zona':
 ↪  'A'},
{'año': 2012, 'metros': 60, 'habitaciones': 2, 'garaje': True, 'zona':
 ↪  'B'},
{'año': 1980, 'metros': 120, 'habitaciones': 4, 'garaje': False, 'zona':
 ↪  'A'},
{'año': 2005, 'metros': 75, 'habitaciones': 3, 'garaje': True, 'zona':
 ↪  'B'},
{'año': 2015, 'metros': 90, 'habitaciones': 2, 'garaje': False, 'zona':
 ↪  'A'}]
```

Construir una función que permita hacer búsqueda de inmuebles en función de un presupuesto dado. La función recibirá como entrada la lista de inmuebles y un precio, y devolverá otra lista con los inmuebles cuyo precio sea menor o igual que el dado. Los inmuebles de la lista que se devuelva deben incorporar un nuevo par a cada diccionario con el precio del inmueble, donde el precio de un inmueble se calcula con las siguiente fórmula en función de la zona:

- Zona A: precio = (metros * 1000 + habitaciones * 5000 + garaje * 15000) * (1-antiguedad/100)
- Zona B: precio = (metros * 1000 + habitaciones * 5000 + garaje * 15000) * (1-antiguedad/100) * 1.5

Solución

1

```python
pisos = [{'año': 2000, 'metros': 100, 'habitaciones': 3, 'garaje':
 ↪  True, 'zona': 'A'}, {'año': 2012, 'metros': 60, 'habitaciones':
 ↪  2, 'garaje': True, 'zona': 'B'}, {'año': 1980, 'metros': 120,
 ↪  'habitaciones': 4, 'garaje': False, 'zona': 'A'}, {'año': 2005,
 ↪  'metros': 75, 'habitaciones': 3, 'garaje': True, 'zona': 'B'},
 ↪  {'año': 2015, 'metros': 90, 'habitaciones': 2, 'garaje': False,
 ↪  'zona': 'A'}]

def añadir_precio(piso):
    precio = (piso['metros'] * 1000 + piso['habitaciones'] * 5000 +
 ↪  int(piso['garaje']) * 15000) * (1 - (2020 - piso['año']) / 100)
    if piso['zona'] == 'B':
        precio *= 1.5
    piso['precio'] = precio
    return piso

def busca_piso(pisos, presupuesto):
    def filtro(piso):
        return piso['precio'] <= presupuesto

    return list(filter(filtro,map(añadir_precio, pisos)))

print(busca_piso(pisos, 100000))
```

Abrir la solución en Google Colab

Ejercicio 8.11. Escribir una función que reciba una muestra de números y devuelva los valores atípicos, es decir, los valores cuya puntuación típica sea mayor que 3 o menor que -3. Nota: La puntuación típica de un valor se obtiene restando la media y dividiendo por la desviación típica de la muestra.

Solución

1

```python
from statistics import mean, stdev

def atipico(muestra):
    media = mean(muestra)
    desviacion = stdev(muestra)
    def f(n):
        puntuacion = (n - media) / desviacion
        return (puntuacion < -3) or (puntuacion > 3)
    return f

def datos_atipicos(muestra):
    return list(filter(atipico(muestra), muestra))

print(datos_atipicos([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1000]))
```

Abrir la solución en Google Colab

9. Ficheros

Ejercicio 9.1. Escribir una función que pida un número entero entre 1 y 10 y guarde en un fichero con el nombre `tabla-n.txt` la tabla de multiplicar de ese número, done `n` es el número introducido.

Solución

1

```
n = int(input('Introduce un número entero entre 1 y 10: '))
nombre_fichero = 'tabla-' + str(n) + '.txt'
f = open(nombre_fichero, 'w')
for i in range(1, 11):
    f.write(str(n) + ' x ' + str(i) + ' = ' + str(n * i) + '\n')
f.close()
```

2

```
n = int(input('Introduce un número entero entre 1 y 10: '))
nombre_fichero = 'tabla-' + str(n) + '.txt'
with open(nombre_fichero, 'w') as f:
    for i in range(1, 11):
        f.write(str(n) + ' x ' + str(i) + ' = ' + str(n * i) + '\n')
```

Abrir la solución en Google Colab

Ejercicio 9.2. Escribir una función que pida un número entero entre 1 y 10, lea el fichero `tabla-n.txt` con la tabla de multiplicar de ese número, done `n` es el número introducido, y la muestre por pantalla. Si el fichero no existe debe mostrar un mensaje por pantalla informando de ello.

Solución

1

```
n = int(input('Introduce un número entero entre 1 y 10: '))
nombre_fichero = 'tabla-' + str(n) + '.txt'
try:
    f = open(nombre_fichero, 'r')
except FileNotFoundError:
    print('No existe el fichero con la tabla del', n)
else:
    print(f.read())
    f.close()
```

2

```
n = int(input('Introduce un número entero entre 1 y 10: '))
nombre_fichero = 'tabla-' + str(n) + '.txt'
try:
    with open(nombre_fichero, 'r') as f:
        print(f.read())
except FileNotFoundError:
    print('No existe el fichero con la tabla del', n)
```

Abrir la solución en Google Colab

Ejercicio 9.3. Escribir una función que pida dos números `n` y `m` entre 1 y 10, lea el fichero `tabla-n.txt` con la tabla de multiplicar de ese número, y muestre por pantalla la línea `m` del fichero. Si el fichero no existe debe mostrar un mensaje por pantalla informando de ello.

Solución

```
n = int(input('Introduce un número entero entre 1 y 10: '))
m = int(input('Introduce otro número entero entre 1 y 10: '))
nombre_fichero = 'tabla-' + str(n) + '.txt'
try:
    with open(nombre_fichero, 'r') as f:
        lineas = f.readlines()
    print(lineas[m - 1])
except FileNotFoundError:
    print('No existe el fichero con la tabla del ', n)
```

Abrir la solución en Google Colab

Ejercicio 9.4. Escribir un programa que acceda a un fichero de internet mediante su url y muestre por pantalla el número de palabras que contiene.

Solución

1

```
def contar_palabras(url):
    '''
    Función que recibe recibe la url de un fichero de texto y
 ↪  devuelve el número de palabras que contiene.
    Parámetros:
        url: Es una cadena con la url del fichero de texto.
    Devuelve:
        El número de palabras que contiene el fichero de texto daro
 ↪  por la url.
    '''
    from urllib import request
    from urllib.error import URLError
    try:
        f = request.urlopen(url)
    except URLError:
        return('¡La url ' + url + ' no existe!')
    else:
        contenido = f.read()
        return len(contenido.split())

print(contar_palabras(
 ↪  'https://www.gutenberg.org/files/2000/2000-0.txt'))
print(contar_palabras('https://no-existe.txt'))
```

2

```
def contar_palabras(url):
    '''
    Función que recibe recibe la url de un fichero de texto y
 ↪  devuelve el número de palabras que contiene.
    Parámetros:
        url: Es una cadena con la url del fichero de texto.
    Devuelve:
        El número de palabras que contiene el fichero de texto daro
 ↪  por la url.
    '''
    from urllib import request
    from urllib.error import URLError
    try:
        with request.urlopen(url) as f:
            contenido = f.read()
    except URLError:
        return('¡La url ' + url + ' no existe!')
    else:
        return len(contenido.split())

print(contar_palabras(
 ↪  'https://www.gutenberg.org/files/2000/2000-0.txt'))
print(contar_palabras('https://no-existe.txt'))
```

Abrir la solución en Google Colab

Ejercicio 9.5. Escribir un programa que abra el fichero con información sobre el PIB per cápita de los países de la Unión Europea (url:`https://ec.europa.eu/eurostat/estat-navtree-portlet-prod/BulkDownloadListing?file=data/sdg_08_10.tsv.gz&unzip=true`), pregunte por las iniciales de un país y muestre el PIB per cápita de ese país de todos los años disponibles.

Solución

1

```
def parsear_pib(url):
    '''
    Función que parsea un fichero con pibs de países.
    Parámetros:
        url: Es una cadena con la url del fichero de texto que
 ↪  contiene el pib per cápita.
    Devuelve:
        Un diccionario con pares pais:pibs donde pibs es, a su vez,
 ↪  un diccionario con los años y los pibs del país.
    '''
    from urllib import request
    from urllib.error import URLError
    try:
        with request.urlopen(url) as f:
            datos = f.read().decode('utf-8').split('\n') # Leer los
 ↪  datos y guardar cada línea en una lista.
    except URLError:
        return('¡La url ' + url + ' no existe!')
    else:
        # Obtenemos los años de la primera linea del fichero.
        años = datos.pop(0).split('\t')[1:]
        # Creamos el diccionario prinpipal para guardar los pibs de
         ↪  todos los países.
        dict_pibs = {}
        # Bucle iterativo para recorrer las líneas del fichero.
        for pais in datos:
            datos_pais = pais.split('\t')
            # Obtenemos el código del país de los dos últimos
             ↪  caracteres del primer campo de la linea.
            codigo_pais = datos_pais.pop(0)[-2:]
            # Construimos un diccionario con los años y el pib del
             ↪  pais.
            dict_pais = {}
            # Bucle iterativo para recorrer los pibs del país.
            for i in range(len(datos_pais)):
                dict_pais[años[i].strip()] = datos_pais[i].strip()
            # Añadimos el diccionario con los pib del país al
             ↪  diccionario principal
            dict_pibs[codigo_pais] = dict_pais
        return dict_pibs

def pib(pibs, pais = "ES"):
    '''
    Función que recibe un diccionario con los pibs de los países y
 ↪  muestra por pantalla los pibs de un país dado.

    Parámetros:
        - pibs: Es un diccionario con los pibs de los países como el
 ↪  que devuelve la función parsear_pibs.
        - pais: Es una cadena con el código del país.

    Salida:
        Muestra por pantalla los pibs del país indicado.
    '''

    print("Año\tPIB")
```

2

```
def pib_pais(url, pais = "ES"):
    '''
    Función que muestra por pantalla el pib per cápita un país dado
 ↪  de los años disponibles en un fichero dado.
    Parámetros:
        url: Es una cadena con la url del fichero de texto que
 ↪  contiene el pib per cápita.
        pais: Es una cadena con el código del país.
    Devuelve:
        Un diccionario con pares año:pib del país dado que hay en el
 ↪  fichero con la url dada.
    '''
    from urllib import request
    from urllib.error import URLError
    try:
        with request.urlopen(url) as f:
            datos = f.read().decode('utf-8').split('\n') # Leer los
 ↪  datos y guardar cada línea en una lista
    except URLError:
        return('¡La url ' + url + ' no existe!')
    else:
        datos = [i.split('\t') for i in datos] # Dividir cada línea
 ↪  por el tabulador
        datos = [list(map(str.strip, i)) for i in datos] # Eliminar
 ↪  espacios en blanco
        for i in datos:
            i[0] = i[0][-2:] # Obtener el código del país de los dos
 ↪  últimos caracteres del primer elemento de la lista
        datos[0][0] = 'años'
        # Creamos un diccionario con claves los códiogos de los
         ↪  países y valores la lista de sus pibs (a excepción del
         ↪  primer par que contiene los años).
        datos = {i[0]:i[1:] for i in datos}
        # Creamos y devolvemos el diccionario con los pibs del país
        return {datos['años'][i]:datos[pais][i] for i in
         ↪  range(len(datos['años']))}

pais = input('Introduce el código de un país: ')
print('Producto Interior Bruto per cápita de', pais)
print("Año\tPIB")
for i, j in pib_pais(
 ↪  "https://ec.europa.eu/eurostat/estat-navtree-portlet-prod/BulkDownloadListing?file=
 ↪  , pais).items():
    print(i, '\t', j)
```

Abrir la solución en Google Colab

Ejercicio 9.6. Escribir un programa para gestionar un listín telefónico con los nombres y los teléfonos de los clientes de una empresa. El programa incorporar funciones crear el fichero con el listín si no existe, para consultar el teléfono de un cliente, añadir el teléfono de un nuevo cliente y eliminar el teléfono de un cliente. El listín debe estar guardado en el fichero de texto `listin.txt` donde el nombre del cliente y su teléfono deben aparecer separados por comas y cada cliente en una línea distinta.

Solución

```
def get_phone(file, client):
    '''
    Función que devuelve el teléfono de un cliente de un fichero
 ↪  dado.
    Parámetros:
        file: Es un fichero con los nombres y teléfonos de clientes.
        client: Es una cadena con el nombre del cliente.
    Devuelve:
        El teléfono del cliente guardado en el fichero o un mensaje
 ↪  de error si el teléfono o el fichero no existe.
    '''
    try:
        f = open(file, 'r')
    except FileNotFoundError:
        return('¡El fichero ' + file + ' no existe!\n')
    else:
        directory = f.readlines()
        f.close()
        directory = dict([tuple(line.split(',')) for line in
 ↪  directory])
        if client in directory:
            return directory[client]
        else:
            return('¡El cliente ' + client + ' no existe!\n')

def add_phone(file, client, telf):
    '''
    Función que añade el teléfono de un cliente de un fichero dado.
    Parámetros:
        file: Es un fichero con los nombres y teléfonos de clientes.
        client: Es una cadena con el nombre del cliente.
        telf: Es una cadena con el teléfono del cliente.
    Devuelve:
        Un mesaje informando sobre si el teléfono se ha añadido o ha
 ↪  habido algún problema.
    '''
    try:
        f = open(file, 'a')
    except FileNotFoundError:
        return('¡El fichero ' + file + ' no existe!\n')
    else:
        f.write(client + ',' + telf + '\n')
        f.close()
        return('El teléfono se ha añadido.\n')

def remove_phone(file, client):
    '''
    Función que elimina el teléfono de un cliente de un fichero
 ↪  dado.
    Parámetros:
        file: Es un fichero con los nombres y teléfonos de clientes.
        client: Es una cadena con el nombre del cliente.
    Devuelve:
        Un mesaje informando sobre si el teléfono se ha borrado o ha
 ↪  habido algún problema.
    '''
```

Abrir la solución en Google Colab

Ejercicio 9.7. 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 reciba el fichero de cotizaciones y devuelva un diccionario con los datos del fichero por columnas.
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.

Solución

```python
def limpiar(cifra):
    """
    Función que elimina los puntos de separación de miles y cambia las 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 separador 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 formato csv y devuelve un diccionario con los nombres de las columnas 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 columnas y las listas de valores en las columnas.
    """
    try:
        # Abrimos el fichero en modo lectura
        with open(ruta, 'r') as f:
            # Leemos el fichero por líneas en una lista
            lineas = f.read().split('\n')
    except FileNotFoundError:
        print('El fichero no existe.')
        return

    # Leemos las claves del primer elemento de la lista y creamos una lista dividiendo la línea por el punto y coma.
    claves = lineas.pop(0).split(";")
    # Creamos el diccionario para guardar las cotizaciones
    cotizaciones = {}
    # Inicializamos el diccionario con listas vacías
    for i in claves:
        cotizaciones[i] = []
    # Bucle iterativo para recorrer la lista de lineas
    for linea in lineas:
        # Creamos una lista con los campos dividiendo la línea por el punto y coma
        campos = linea.split(';')
        # Añadimos el primer campo (el nombre de la empresa) a la lista del diccionario
        cotizaciones[claves[0]].append(campos[0])
        # Bucle iterativo para añadir el resto de los campos a las listas correspondientes del diccionario.
        # Previamente los campos se limpian del carácter de separación de miles y se sustituye la coma por el punto para el separador de decimales.
```

Abrir la solución en Google Colab

Ejercicio 9.8. El fichero calificaciones.csv contiene las calificaciones de un curso. Durante el curso se realizaron dos exámenes parciales de teoría y un examen de prácticas. Los alumnos que tuvieron menos de 4 en alguno de estos exámenes pudieron repetirlo en la al final del curso (convocatoria ordinaria). Escribir un programa que contenga las siguientes funciones:

1. Una función que reciba el fichero de calificaciones y devuelva una lista de diccionarios, donde cada diccionario contiene la información de los exámenes y la asistencia de un alumno. La lista tiene que estar ordenada por apellidos.
2. Una función que reciba una lista de diccionarios como la que devuelve la función anterior y añada a cada diccionario un nuevo par con la nota final del curso. El peso de cada parcial de teoría en la nota final es de un 30 % mientras que el peso del examen de prácticas es de un 40 %.
3. Una función que reciba una lista de diccionarios como la que devuelve la función anterior y devuelva dos listas, una con los alumnos aprobados y otra con los alumnos suspensos. Para aprobar el curso, la asistencia tiene que ser mayor o igual que el 75 %, la nota de los exámenes parciales y de prácticas mayor o igual que 4 y la nota final mayor o igual que 5.

Solución

```
def nota(cifra):
    '''Función que elimina cambia las comas de separación de
     ↪  decimales por puntos de una cifra y la convierte en un real.
    Parámetros:
        - cifra: Es una cadena con una cifra.
    Devuelve:
        Un real con la cifra de la cadena después de cambiar el
 ↪  separador de decimales por punto.
    '''
    cifra = cifra.replace(',','.')
    return float(cifra)

def calificaciones(ruta):
    '''Función que preprocesa los datos contenidos en un fichero con
     ↪  formato csv y devuelve una lista de diccionarios donde las
     ↪  claves de los diccionarios son los datos de la primera fila
     ↪  y los valores los datos de cada línea del fichero.

    Parámetros:
        - ruta: Es una cadena con la ruta del fichero.
    Devuelve:
        Una lista de diccionarios donde cada diccionario contiene
 ↪  los datos de una linea del fichero (a excepción de la primera
 ↪  línea), usando como claves los datos de la primera línea.
    '''
    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
    lineas = f.readlines()
    # Cerramos el fichero
    f.close()
    # Leemos las claves del primer elemento de la lista, eliminamos
     ↪  el cambio de línea que aparece al final y dividimos la
     ↪  cadena por el punto y coma.
    claves = lineas[0][:-1].split(";")
    # Creamos la lista de calificaciones
    calificaciones = []
    # Recorremos las líneas del fichero y para cada línea creamos un
     ↪  diccionario que añadimos a la lista de calificaciones.
    for i in lineas[1:]:
        # Eliminamos el cambio de línea del final y dividimos la
         ↪  cadena por el punto y coma.
        valores = i[:-1].split(";")
        # Creamos un diccionario vacío para ir añadiendo los datos
         ↪  de cada alumno.
        alumno = {}
        # Recorremos la lista de valores y los añadimos al
         ↪  diccionario.
        for j in range(len(valores)):
            alumno[claves[j]] = valores[j]
        # Añadimos el diccionario a la lista de calificaciones
        calificaciones.append(alumno)
```

Abrir la solución en Google Colab

10. Librería Pandas

Ejercicio 10.1. Escribir un programa que pregunte al usuario por las ventas de un rango de años y muestre por pantalla una serie con los datos de las ventas indexada por los años, antes y después de aplicarles un descuento del 10 %.

Solución

```
import pandas as pd

inicio = int(input('Introduce el año inicial: '))
fin = int(input('Introduce el año final: '))
ventas = {}
for i in range(inicio, fin+1):
    ventas[i] = float(input('Introduce las ventas del año ' + str(i)
 ↪  +': '))
ventas = pd.Series(ventas)
print('Ventas\n', ventas)
print('Ventas con descuento\n', ventas*0.9)
```

Abrir la solución en Google Colab

Ejercicio 10.2. Escribir una función que reciba un diccionario con las notas de los alumno de un curso y devuelva una serie con la nota mínima, la máxima, media y la desviación típica.

Solución

1

```python
import pandas as pd

def estadistica_notas(notas):
    notas = pd.Series(notas)
    estadisticos = pd.Series([notas.min(), notas.max(),
    ↪ notas.mean(), notas.std()], index=['Min', 'Max', 'Media',
    ↪ 'Desviación típica'])
    return estadisticos

notas = {'Juan':9, 'María':6.5, 'Pedro':4, 'Carmen': 8.5, 'Luis': 5}
print(estadistica_notas(notas))
```

2

```python
import pandas as pd

def estadistica_notas(notas):
    notas = pd.Series(notas)
    return notas.describe()

notas = {'Juan':9, 'María':6.5, 'Pedro':4, 'Carmen': 8.5, 'Luis': 5}
print(estadistica_notas(notas))
```

Abrir la solución en Google Colab

Ejercicio 10.3. Escribir una función que reciba un diccionario con las notas de los alumnos de un curso y devuelva una serie con las notas de los alumnos aprobados ordenadas de mayor a menor.

Solución

```python
import pandas as pd

def aprobados(notas):
    notas = pd.Series(notas)
    return notas[notas >= 5].sort_values(ascending=False)

notas = {'Juan':9, 'María':6.5, 'Pedro':4, 'Carmen': 8.5, 'Luis': 5}
print(aprobados(notas))
```

Abrir la solución en Google Colab

Ejercicio 10.4. Escribir programa que genere y muestre por pantalla un DataFrame con los datos de la tabla siguiente:

Mes	Ventas	Gastos
Enero	30500	22000
Febrero	35600	23400
Marzo	28300	18100
Abril	33900	20700

Solución

1

```
import pandas as pd

datos = {'Mes':['Enero', 'Febrero', 'Marzo', 'Abril'],
 ↪  'Ventas':[30500, 35600, 28300, 33900], 'Gastos':[22000, 23400,
 ↪  18100, 20700]}
contabilidad = pd.DataFrame(datos)
print(contabilidad)
```

2

```
import pandas as pd

datos = [['Enero', 30500, 22000], ['Febrero', 35600, 23400],
 ↪  ['Marzo', 28300, 18100], ['Abril', 33900,20700]]
contabilidad = pd.DataFrame(datos, columns=['Mes', 'Ventas',
 ↪  'Gastos'])
print(contabilidad)
```

Abrir la solución en Google Colab

Ejercicio 10.5. Escribir una función que reciba un DataFrame con el formato del ejercicio anterior, una lista de meses, y devuelva el balance (ventas - gastos) total en los meses indicados.

Solución

1

```
import pandas as pd

datos = {'Mes':['Enero', 'Febrero', 'Marzo', 'Abril'],
 ↪  'Ventas':[30500, 35600, 28300, 33900], 'Gastos':[22000, 23400,
 ↪  18100, 20700]}

contabilidad = pd.DataFrame(datos)

def balance(contabilidad, meses):
    contabilidad['Balance'] = contabilidad.Ventas -
 ↪  contabilidad.Gastos
    return contabilidad[contabilidad.Mes.isin(meses)].Balance.sum()

print(balance(contabilidad, ['Enero','Marzo']))
```

2

```
import pandas as pd

datos = {'Mes':['Enero', 'Febrero', 'Marzo', 'Abril'],
 ↪  'Ventas':[30500, 35600, 28300, 33900], 'Gastos':[22000, 23400,
 ↪  18100, 20700]}

contabilidad = pd.DataFrame(datos)

def balance(contabilidad, meses):
    contabilidad['Balance'] = contabilidad.Ventas -
 ↪  contabilidad.Gastos
    return contabilidad.set_index('Mes').loc[meses,'Balance'].sum()

print(balance(contabilidad, ['Enero','Marzo']))
```

Abrir la solución en Google Colab

Ejercicio 10.6. 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). Construir una función que construya un DataFrame a partir del un fichero con el formato anterior y devuelva otro DataFrame con el mínimo, el máximo y la media de dada columna.

Solución

```
import pandas as pd

def resumen_cotizaciones(fichero):
    df = pd.read_csv(fichero, sep=';', decimal=',', thousands='.',
    ↪ index_col=0)
    return pd.DataFrame([df.min(), df.max(), df.mean()],
     ↪ index=['Mínimo', 'Máximo', 'Media'])

resumen_cotizaciones('cotizacion.csv')
```

Abrir la solución en Google Colab

Ejercicio 10.7. El fichero titanic.csv contiene información sobre los pasajeros del Titanic. Escribir un programa con los siguientes requisitos:

1. Generar un DataFrame con los datos del fichero.
2. Mostrar por pantalla las dimensiones del DataFrame, el número de datos que contiene, los nombres de sus columnas y filas, los tipos de datos de las columnas, las 10 primeras filas y las 10 últimas filas
3. Mostrar por pantalla los datos del pasajero con identificador 148.
4. Mostrar por pantalla las filas pares del DataFrame.
5. Mostrar por pantalla los nombres de las personas que iban en primera clase ordenadas alfabéticamente.
6. Mostrar por pantalla el porcentaje de personas que sobrevivieron y murieron.
7. Mostrar por pantalla el porcentaje de personas que sobrevivieron en cada clase.
8. Eliminar del DataFrame los pasajeros con edad desconocida.
9. Mostrar por pantalla la edad media de las mujeres que viajaban en cada clase.
10. Añadir una nueva columna booleana para ver si el pasajero era menor de edad o no.
11. Mostrar por pantalla el porcentaje de menores y mayores de edad que sobrevivieron en cada clase.

Solución

```
import pandas as pd

# Generar un DataFrame con los datos del fichero.
titanic = pd.read_csv(
 ↪  'https://raw.githubusercontent.com/asalber/asalber.github.io/master/python/ejercici
 ↪  , index_col=0)

print(titanic)
```

```
# Mostrar por pantalla las dimensiones del DataFrame, el número de
 ↪  datos que contiene, los nombres de sus columnas y filas, los
 ↪  tipos de datos de las columnas, las 10 primeras filas y las 10
 ↪  últimas filas.
print('Dimensiones:', titanic.shape)
print('Número de elemntos:', titanic.size)
print('Nombres de columnas:', titanic.columns)
print('Nombres de filas:', titanic.index)
print('Tipos de datos:\n', titanic.dtypes)
print('Primeras 10 filas:\n', titanic.head(10))
print('Últimas 10 filas:\n', titanic.tail(10))
```

```
# Mostrar por pantalla los datos del pasajero con identificador 148
print(titanic.loc[148])
```

```
# Mostrar por pantalla las filas pares del DataFrame.
print(titanic.iloc[range(0,titanic.shape[0],2)])
```

```
# Mostrar los nombres de las personas que iban en primera clase
 ↪  ordenadas alfabéticamente.
print(titanic[titanic["Pclass"]==1]['Name'].sort_values())
```

```
# Mostrar por pantalla el porcentaje de personas que sobrevivieron y
 ↪  murieron
print(titanic['Survived'].value_counts()/titanic['Survived'].count()
 ↪  * 100)

# Alternativa
print(titanic['Survived'].value_counts(normalize=True) * 100)
```

```
#Mostrar por pantalla el porcentaje de personas que sobrevivieron en
↪ cada clase
print(titanic.groupby('Pclass')['Survived'].value_counts(normalize=
↪ True))

# Eliminar del DataFrame los pasajeros con edad desconocida.
titanic.dropna(subset=['Age'])

# Alternativa
# titanic = titanic[titanic['Age'].notna()]

# Mostrar la edad media de las mujeres que viajaban en cada clase.
print(titanic.groupby(['Pclass','Sex'])['Age'].mean().unstack()[
↪ 'female'])

# Añadir una nueva columna booleana para ver si el pasajero era
↪ menor de edad o no.
titanic['Young'] = titanic['Age'] < 18

# Mostrar el porcentaje de menores y mayores de edad que
↪ sobrevivieron en cada clase.
print(titanic.groupby(['Pclass',
↪ 'Young'])['Survived'].value_counts(normalize = True) * 100)
```

Abrir la solución en Google Colab

Ejercicio 10.8. Los ficheros emisiones-2016.csv, emisiones-2017.csv, emisiones-2018.csv y emisiones-2019.csv, contienen datos sobre las emisiones contaminates en la ciudad de Madrid en los años 2016, 2017, 2018 y 2019 respectivamente. Escribir un programa con los siguientes requisitos:

1. Generar un DataFrame con los datos de los cuatro ficheros.
2. Filtrar las columnas del DataFrame para quedarse con las columnas ESTACION, MAGNITUD, AÑO, MES y las correspondientes a los días D01, D02, etc.
3. Reestructurar el DataFrame para que los valores de los contaminantes de las columnas de los días aparezcan en una única columna.
4. Añadir una columna con la fecha a partir de la concatenación del año, el mes y el día (usar el módulo `datetime`).
5. Eliminar las filas con fechas no válidas (utilizar la función `isnat` del módulo `numpy`) y ordenar el DataFrame por estaciones contaminantes y fecha.

6. Mostrar por pantalla las estaciones y los contaminantes disponibles en el DataFrame.
7. Crear una función que reciba una estación, un contaminante y un rango de fechas y devuelva una serie con las emisiones del contaminante dado en la estación y rango de fechas dado.
8. Mostrar un resumen descriptivo (mínimo, máximo, media, etc.) para cada contaminante.
9. Mostrar un resumen descriptivo para cada contaminante por distritos.
10. Crear una función que reciba una estación y un contaminante y devuelva un resumen descriptivo de las emisiones del contaminante indicado en la estación indicada.
11. Crear una función que devuelva las emisiones medias mensuales de un contaminante y un año dados para todos las estaciones.
12. Crear un función que reciba una estación de medición y devuelva un DataFrame con las medias mensuales de los distintos tipos de contaminantes.

Solución

```
import pandas as pd
import numpy as np
import datetime as dt

# Generar un DataFrame con los datos de los cuatro ficheros
import pandas as pd

emisiones_2016 = pd.read_csv('emisiones-2016.csv', sep = ';')
emisiones_2017 = pd.read_csv('emisiones-2017.csv', sep = ';')
emisiones_2018 = pd.read_csv('emisiones-2018.csv', sep = ';')
emisiones_2019 = pd.read_csv('emisiones-2019.csv', sep = ';')
emisiones = pd.concat([emisiones_2016, emisiones_2017,
 ↪  emisiones_2018, emisiones_2019])
emisiones
```

```
# Filtrar las columnas del DataFrame para quedarse con las columnas
 ↪  ESTACION, MAGNITUD, AÑO, MES y las correspondientes a los días
 ↪  D01, D02, etc.
columnas = ['ESTACION', 'MAGNITUD', 'ANO', 'MES']
columnas.extend([col for col in emisiones if col.startswith('D')])
emisiones = emisiones[columnas]
emisiones
```

```
# Reestructurar el DataFrame para que los valores de los
 ↪  contaminantes de las columnas de los días aparezcan en una única
 ↪  columna.
emisiones = emisiones.melt(id_vars=['ESTACION', 'MAGNITUD', 'ANO',
 ↪  'MES'], var_name='DIA', value_name='VALOR')
emisiones
```

```
# Crear una nueva columna con las fechas a partir del año, mes y día
# Primero eliminamos el caracter D del comienzo de la columna de los
 ↪  días
emisiones['DIA'] = emisiones.DIA.str.strip('D')
# Concatenamos las columnas del año, mes y día
emisiones['FECHA'] = emisiones.ANO.apply(str) + '/' +
 ↪  emisiones.MES.apply(str) + '/' + emisiones.DIA.apply(str)
# Convertimos la nueva columna al tipo fecha
emisiones['FECHA'] = pd.to_datetime(emisiones.FECHA,
 ↪  format='%Y/%m/%d', infer_datetime_format=True, errors='coerce')
emisiones
```

```
# Eliminar las filas con fechas no válidas
emisiones =
 ↪  emisiones.drop(emisiones[np.isnat(emisiones.FECHA)].index)
# Ordenar el el dataframe por estación, magnitud y fecha
emisiones.sort_values(['ESTACION', 'MAGNITUD', 'FECHA'])
```

```
# Mostrar las estaciones disponibles
print('Estaciones:', emisiones.ESTACION.unique())
# Mostrar los contaminantes disponibles
print('Contaminantes:', emisiones.MAGNITUD.unique())
```

```
# Función que devuelve las emisiones de un contaminante dado en una
 ↪  estación y rango de fechas dado.
def evolucion(estacion, contaminante, desde, hasta):
    return emisiones[(emisiones.ESTACION == estacion) &
     ↪  (emisiones.MAGNITUD == contaminante) & (emisiones.FECHA >=
     ↪  desde) & (emisiones.FECHA <=
     ↪  hasta)].sort_values('FECHA').VALOR
evolucion(56, 8, dt.datetime.strptime('2018/10/25', '%Y/%m/%d'),
 ↪  dt.datetime.strptime('2019/02/12', '%Y/%m/%d'))
```

```
# Resumen descriptivo por contaminantes
emisiones.groupby('MAGNITUD').VALOR.describe()
```

```
# Resumen descriptivo por contaminantes y distritos
emisiones.groupby(['ESTACION', 'MAGNITUD']).VALOR.describe()
```

```
# Función que devuelve un resumen descriptivo de la emisiones en un
 ↪  contaminante dado en un estación dada
def resumen(estacion, contaminante):
    return emisiones[(emisiones.ESTACION == estacion) &
     ↪  (emisiones.MAGNITUD == contaminante)].VALOR.describe()

# Resumen de Dióxido de Nitrógeno en Plaza Elíptica
print('Resumen Dióxido de Nitrógeno en Plaza Elíptica:\n',
 ↪  resumen(56, 8),'\n', sep='')
# Resumen de Dióxido de Nitrógeno en Plaza del Carmen
print('Resumen Dióxido de Nitrógeno en Plaza del Carmen:\n',
 ↪  resumen(35, 8), sep='')
```

```
# Función que devuelve una serie con las emisiones medias mensuales
 ↪  de un contaminante y un mes año para todos las estaciones
def evolucion_mensual(contaminante, año):
    return emisiones[(emisiones.MAGNITUD == contaminante) &
     ↪  (emisiones.ANO == año)].groupby(['ESTACION',
     ↪  'MES']).VALOR.mean().unstack('MES')

# Evolución del dióxido de nitrógeno en 2019
evolucion_mensual(8, 2019)
```

Abrir la solución en Google Colab

11. Librería Matplotlib

Ejercicio 11.1. Escribir un programa que pregunte al usuario por las ventas de un rango de años y muestre por pantalla un diagrama de líneas con la evolución de las ventas.

Solución

```
import matplotlib.pyplot as plt
# Preguntamos por el año inicial
inicio = int(input('Introduce el año inicial: '))
# Preguntamos por el año final
fin = int(input('Introduce el año final: '))
# Definimos un diccionario vacío para guardar las ventas de cada año
ventas = {}
# Bucle iterativo para preguntar las ventas de cada año y guardarlas
 ↪  en el diccionario
# i toma como valores los años desde el año de inicio hasta el año
 ↪  final
for i in range(inicio, fin+1):
    # Preguntamos por las ventas del año i y las guardamos en el
     ↪  diccionario con la clave el año y el valor las ventas
    ventas[i] = float(input('Introduce las ventas del año ' + str(i)
 ↪  +': '))
# Definimos la figura y los ejes del gráfico con Matplotlib
fig, ax = plt.subplots()
# Dibujamos la línea con las ventas a partir del diccionario
ax.plot(ventas.keys(), ventas.values())
# Mostarmos el gráfico por pantalla
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.2. Escribir una función que reciba una diccionario con las notas de las asignaturas de un curso y una cadena con el nombre de un color y devuelva un diagrama de barras de las notas en el color dado.

Solución

```
import matplotlib.pyplot as plt

def diagrama_barras_notas(notas, color):
    '''Función que construye un diagrama de barras con las notas de
    ↪ las asignaturas de un curso.

    Parámetros:
        - notas: Es un diccionario formado por pares con clave el
    ↪ nombre de la asignaturay valor la nota.
        - color: Es una cadena con el color de las barras.

    Salida:
        - Un diagrama de barras con las notas del diccionario dado
    ↪ en el color dado.
    '''
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos las barras con las notas a partir del diccionario
    ax.bar(notas.keys(), notas.values(), color = color)
    # Devolvemos un objeto con los ejes y las barras que contienen
    return ax

notas = {'Programación':9, 'Mates':6.5, 'Economía':4, 'Historia': 8}
diagrama_barras_notas(notas, 'orange')
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.3. Escribir una función que reciba una serie de Pandas con las notas de los alumnos de un curso y devuelva un diagrama de cajas con las notas. El diagrama debe tener el título “Distribución de notas”.

Solución

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

def diagrama_caja_notas(notas):
    '''Función que construye un diagrama de cajas con las notas de
    ↪ los alumnos de un curso.

    Parámetros:
        - notas: Es una serie de Pandas con las notas de los
↪   alumnos.
    '''
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos los sectores con las verntas a partir de la serie
    notas.plot(kind = 'box', ax = ax)
    # Eliminamos las marcas del eje x
    plt.xticks([])
    #
    # Añadimos el título
    plt.title('Distribución de notas')
    # Devolvemos el objeto con los ejes y el gráfico que contienten
    return ax

notas = [4, 8, 7.5, 6, 5.5, 5.2, 3.5, 7.7, 3.2, 9, 6.8]
s_notas = pd.Series(notas)
diagrama_caja_notas(s_notas)
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.4. Escribir una función que reciba una serie de Pandas con el número de ventas de un producto durante los meses de un trimestre y un título y cree un diagrama de sectores con las ventas en formato png con el titulo dado. El diagrama debe guardarse en un fichero con formato png y el título dado.

Solución

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

def diagrama_sectores_ventas(ventas, titulo):
    '''Función que construye un diagrama de sectores con las ventas
     ↪  de un trimestre y lo guarda con un nombre dado.

    Parámetros:
        - ventas: Es una serie de Pandas con las ventas del
 ↪  trimestre, siendo el índice los meses.
        - titulo: Es una cadena con el título.
    '''
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos los sectores con las verntas a partir de la serie
    ventas.plot(kind = 'pie', ax = ax)
    # Eliminamos el título del eje y
    plt.ylabel('')
    # Añadimos el título
    plt.title(titulo)
    # Guardamos el gráfico con el nombre dado en formato png
    plt.savefig(titulo + '.png')
    return

ventas = {'Enero':200, 'Febrero':240, 'Marzo':310}
s_ventas = pd.Series(ventas)
diagrama_sectores_ventas(s_ventas, 'Ventas primer trimestre')
```

Abrir la solución en Google Colab

Ejercicio 11.5. Escribir una función que reciba una serie de Pandas con el número de ventas de un producto por años y una cadena con el tipo de gráfico a generar (lineas, barras, sectores, areas) y devuelva un diagrama del tipo indicado con la evolución de las ventas por años y con el título "Evolución del número de ventas".

Solución

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

def diagrama_evolucion_ventas(ventas, tipo):
    '''Función que construye un diagrama del tipo indicado con la
     ↪  evolución de las ventas por años.

    Parámetros:
        - ventas: Es un dataframe de Pandas con las ventas, siendo
 ↪  el índice los años.
        - tipo: Es una cadena con el tipo de gráfico a dibujar:
 ↪  lineas, barras, sectores o areas.

    Salida:
        Un gráfico del tipo indicado con la evolución de las ventas.
    '''
    # Definimos un diccionario con los tipos de gráficos
    graficos = {'lineas':'line', 'barras':'bar', 'sectores':'pie',
 ↪  'area':'area'}
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos las series de líneas con los ingresos y los gastos
    ventas.plot(kind = graficos[tipo], ax = ax)
    # Añadimos el título
    plt.title('Evolución del número de ventas')
    # Devolvemos el objeto con los ejes y el gráfico que contienten
    return ax

df_ventas = pd.Series([1200, 840, 1325, 1280, 1500], index = [2000,
 ↪  2001, 2002, 2003, 2004])
diagrama_evolucion_ventas(df_ventas, 'lineas')
plt.show()
diagrama_evolucion_ventas(df_ventas, 'area')
plt.show()
diagrama_evolucion_ventas(df_ventas, 'barras')
plt.show()
diagrama_evolucion_ventas(df_ventas, 'sectores')
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.6. Escribir una función que reciba un dataframe de Pandas con los ingresos y gastos de una empresa por meses y devuelva un diagrama de líneas con dos líneas, una para los ingresos y otra para los gastos. El diagrama debe tener una leyenda identificando la línea de los ingresos y la de los gastos, un título con el nombre "Evolución de ingresos y gastos" y el eje y debe empezar en 0.

Solución

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

def diagrama_lineas_ingresos_gastos(datos):
    '''Función que construye un diagrama de lineas con los ingresos
     ↪  y gastos de un cuatrimestre.

    Parámetros:
        - datos: Es un dataframe de Pandas con dos columnas, una
 ↪  para los ingresos y otra para los gastos, y como índice los
 ↪  meses.

    Salida:
        Un gráfico de líneas con los ingresos y los gastos dados.
    '''
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos las series de líneas con los ingresos y los gastos
    datos.plot(ax = ax)
    # Añadimos la escala del eje y
    ax.set_ylim([0, max(datos.Ingresos.max(), datos.Gastos.max()) +
 ↪  500])
    # Añadimos el título
    plt.title('Evolución de ingresos y gastos')
    # Devolvemos el objeto con los ejes y el gráfico que contienten
    return ax

datos = {'Mes':['Ene', 'Feb', 'Mar', 'Abr'], 'Ingresos':[4500, 5200,
 ↪  4800, 5300], 'Gastos':[2300, 2450, 2000, 2200]}
df_datos = pd.DataFrame(datos).set_index('Mes')
diagrama_lineas_ingresos_gastos(df_datos)
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.7. El fichero `bancos.csv` contiene las cotizaciones de los principales bancos de España con : `Empresa` (nombre de la empresa), `Apertura` (precio de la acción a la apertura de bolsa), `Máximo` (precio máximo de la acción durante la jornada), `Mínimo` (precio mínimo de la acción durante la jornada), `Cierre` (precio de la acción al cierre de bolsa), `Volumen` (volumen al cierre de bolsa). Construir una función reciba el fichero `bancos.csv` y cree un diagrama de líneas con las series temporales de las cotizaciones de cierre de cada banco.

Solución

1

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

def evolucion_cotizacion(datos, variable):
    '''Función que construye un diagrama de lineas con la evolución
     ↪  de las cotizaciones de las empresas en bolsa.

    Parámetros:
        - datos: Es un dataframe de Pandas con las columnas Empresa,
 ↪  Apertura, Mínimo, Máximo, Cierre, Volumen.
        - variable: Es una cadena con el nombre de la columna del
 ↪  dataframe a dibujar.

    Salida:
        Un gráfico de líneas con las series temporales de las
 ↪  cotizaciones de cierre de cada empresa.
    '''
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos las series de cotizaciones por empresa
    datos.groupby('Empresa').plot(x = 'Fecha', y = variable, ax =
 ↪  ax)
    # Añadimos el título
    plt.title('Evolución de las cotizaciones (' + variable + ')')
    # Añadimos la legenda
    plt.legend(df_datos.groupby('Empresa').groups.keys())
    # Devolvemos el objeto con los ejes y el gráfico que contienten
    return ax

# Creamos un dataframe a partir del fichero csv
df_datos = pd.read_csv('bancos.csv')
# Convertimos la columna Fecha a formato datetime
df_datos["Fecha"] = pd.to_datetime(df_datos["Fecha"])
# Llamamos a la función para crear el gráfico
evolucion_cotizacion(df_datos, 'Cierre')
plt.show()
```

2

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

def evolucion_cotizacion(datos, variable):
    '''Función que construye un diagrama de lineas con la evolución
     ↪ de las cotizaciones de las empresas en bolsa.

    Parámetros:
        - datos: Es un dataframe de Pandas con las columnas Empresa,
 ↪ Apertura, Mínimo, Máximo, Cierre, Volumen.
        - variable: Es una cadena con el nombre de la columna del
 ↪ dataframe a dibujar.

    Salida:
        Un gráfico de líneas con las series temporales de las
 ↪ cotizaciones de cierre de cada empresa.
    '''
    # Filtramos el data frame para quedarnos con las columnas a
     ↪ dibujar
    datos = datos[["Fecha", "Empresa", variable]]
    # Reestructuramos el data frame a formato ancho
    datos = datos.pivot(index = 'Fecha', columns = 'Empresa', values
 ↪ = variable)
    # Definimos la figura y los ejes del gráfico con Matplotlib
    fig, ax = plt.subplots()
    # Dibujamos las series de cotizaciones por empresa
    datos.plot(ax = ax)
    # Añadimos el título
    plt.title('Evolución de las cotizaciones (' + variable + ')')
    # Devolvemos el objeto con los ejes y el gráfico que contienten
    return ax

# Creamos un dataframe a partir del fichero csv
df_datos = pd.read_csv('bancos.csv')
# Convertimos la columna Fecha a formato datetime
df_datos["Fecha"] = pd.to_datetime(df_datos["Fecha"])
# Llamamos a la función para crear el gráfico
evolucion_cotizacion(df_datos, 'Cierre')
plt.show()
```

Abrir la solución en Google Colab

Ejercicio 11.8. El fichero titanic.csv contiene información sobre los pasajeros del Titanic.

Crear un dataframe con Pandas y a partir de él generar los siguientes diagramas.

1. Diagrama de sectores con los fallecidos y supervivientes.
2. Histograma con las edades.
3. Diagrama de barras con el número de personas en cada clase.
4. Diagrama de barras con el número de personas fallecidas y supervivientes en cada clase.
5. Diagrama de barras con el número de personas fallecidas y supervivientes acumuladas en cada clase.

Solución

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

# Creamos un dataframe a partir del fichero csv
df_titanic = pd.read_csv('titanic.csv')
# Creamos la figura y los ejes
fig, ax = plt.subplots()
# Diagrama de sectores de falleccidos y supervivientes
df_titanic.Survived.value_counts().plot(kind = "pie", labels =
 ↪  ["Muertos", "Supervivientes"], title = "Distribución de
 ↪  supervivientes")
plt.show()
```

```
# Histograma de edades
df_titanic.Age.plot(kind = "hist", title = "Histograma de edades")
plt.show()
```

```
# Diagrama de barras con el número de personas de cada clase
df_titanic.Pclass.value_counts().plot(kind = "bar", title = "Número
 ↪  de personas por clase")
plt.show()
```

```
# Otra forma
df_titanic.groupby("Pclass").size().plot(kind = "bar", title =
 ↪  "Número de personas por clase")
plt.show()
```

```
# Diagrama de barras con el número de personas fallecidas y
 ↪  supervivientes de cada clase
df_titanic.groupby(["Pclass",
 ↪  "Survived"]).size().unstack().plot(kind = "bar", title = "Número
 ↪  de personas fallecidas y supervivientes por clase")
plt.show()
```

```
# Diagrama de barras con el número de personas fallecidas y
 ↪  supervivientes acumuladas de cada clase
df_titanic.groupby(["Pclass",
 ↪  "Survived"]).size().unstack().plot(kind = "bar", stacked = True,
 ↪  title = "Número de personas fallecidas y supervivientes por
 ↪  clase")
plt.show()
```

Abrir la solución en Google Colab

A. Depuración

Ejercicio A.1. Corregir los errores sintácticos del siguiente programa:

```
contraseña = input('Introduce la contraseña: ")
if contraseña in ['sesamo'):
  print('Pasa')
else
  print('No pasa')
```

Solución

```
contraseña = input('Introduce la contraseña: ')
if contraseña in ['sesamo']:
    print('Pasa')
else:
    print('No pasa')
```

Abrir la solución en Google Colab

Ejercicio A.2. Detectar y corregir los errores del siguiente programa que aplica el iva a una factura:

```
base = input('Introduce la base imponible de la factura: ')
print(aplica_iva(base, iva))

def aplica_iva(base, iva = 21):
    base = base * iva
    return base
```

Solución

```
base = float(input('Introduce la base imponible de la factura: '))

def aplica_iva(base, iva = 21):
    base += base * iva / 100
    return base

print(aplica_iva(base))
```

Abrir la solución en Google Colab

Ejercicio A.3. Detectar y corregir los errores del siguiente programa que calcula el producto escalar de dos vectores:

```
u = (1, 2, 3)
v = (4, 5, 6)

def producto_escalar(u, v):
    for i in u:
        u[i+1] *= v[i+1]
    return sum(u)

print(producto_escalar(u, v))
```

Solución

```
u = [1, 2, 3]
v = [4, 5, 6]

def producto_escalar(u, v):
    for i in range(len(u)):
        u[i] *= v[i]
    return sum(u)

print(producto_escalar(u, v))
```

Abrir la solución en Google Colab

Ejercicio A.4. Detectar y corregir los errores del siguiente programa que devuelve y elimina el teléfono de un listín telefónico a través del nombre del usuario:

```
listin = {'Juan':123456789, 'Pedro':987654321}

def elimina(listin, usuario):
    del listin[usuario]
    return listin[usuario]

print(elimina(listin, 'Pablo'))
```

Solución

```
listin = {'Juan':123456789, 'Pedro':987654321}

def elimina(listin, usuario):
    return listin.pop(usuario, '')

print(elimina(listin, 'Pablo'))
```

Abrir la solución en Google Colab

Ejercicio A.5. Detectar y corregir los errores del siguiente programa que multiplica dos matrices:

```
a = ((1, 2, 3),
     (3, 2, 1))
b = ((1, 2),
     (3, 4),
     (5, 6))

def producto(a, b):
    producto = []
    for i in range(len(b)):
        fila = []
        for j in range(len(a[0])):
            suma = 0
            for k in range(len(a[0]+1)):
                suma += a[i][k] * b[k+1][j]
            fila[j] = suma
        producto[i] = tuple(fila)
    return tuple(producto)

print(producto(a, b))
```

Solución

```
a = ((1, 2, 3),
     (3, 2, 1))
b = ((1, 2),
     (3, 4),
     (5, 6))

def producto(a, b):
    producto = []
    for i in range(len(a)):
        fila = []
        for j in range(len(b[0])):
            suma = 0
            for k in range(len(a[0])):
                suma += a[i][k] * b[k][j]
            fila.append(suma)
        producto.append(tuple(fila))
    return tuple(producto)

print(producto(a, b))
```

Abrir la solución en Google Colab