Status	Finished
Started	Monday, 2 December 2024, 6:13 PM
Completed	Monday, 2 December 2024, 6:20 PM
Duration	6 mins 7 secs
Marks	3.40/4.00
Grade	**8.50** out of 10.00 (**85**%)

Information

Texto con formato

Compare las dos capturas de pantalla que se muestran a continuación:

CAPTURA1

```
Aceite 9
Arroz 8
Azucar 10.5
Pollo 30.25
------------
Total 57.75
```

CAPTURA2

```
Aceite  9.00
Arroz   8.00
Azucar 10.50
Pollo  30.25
------------
Total  57.75
```

Ambas capturas muestran la misma información, pero la segunda muestra el texto formateado, alineando por un lado los productos y por el otro los precios, y unificando el número de dígitos decimales de estos últimos. El texto con formato resulta más legible y, en consecuencia, es más fácil interpretar la información que suministra.

Information

Interpolación de strings: *f-strings*

Desde la versión 3.6 de Python, el método preferente para formatear texto son los llamados "literales de string formateados" o *f-strings*, que usan como técnica la interpolación de expresiones:

```
name = "John"
age = 24
print(f'{name} tiene { age } años')  # los espacios dentro de llaves no tienen efecto
```

El código previo muestra el mensaje:

```
John tiene 24 años
```

Al anteponer una `f`, o una `F`, a un literal de tipo string, indicamos que se trata de una f-string. Una f-string es un literal string en el que se pueden intercalar expresiones de cualquier tipo, encerradas entre llaves, que se evaluarán, se convertirán a string y se insertarán en la f-string en la posición correspondiente.

```
num1 = 12
num2 = 10
print(f"sumar {num1} y {num2} da {num1 + num2}")
```

```
sumar 12 y 10 da 22
```

Si queremos incluir los caracteres de llaves en una f-string, habrá que doblarlas:

```
print(f"La palabra {{llaves}} se muestra entre llaves")
```

```
La palabra {llaves} se muestra entre llaves
```

Las f-strings, una vez evaluadas, dan como resultado una string normal, por lo que p.e. se pueden asignar a variables:

```
name = "John"
age = 24
message = f'{name} tiene {age} años'
print(message)
```

```
John tiene 24 años
```

Question 1

Complete

Mark 1.00 out of 1.00

¿Qué valor muestra el siguiente código?

```
name = "Alberto"
print(f"Hola, mi nombre es {name} y tengo {5 * 5} años")
```

Answer: Hola, mi nombre es Alberto y tengo 25 años

Information

Modificadores de formato

La captura de pantalla que se muestra a continuación:

```
Aceite  9.00
Arroz   8.00
Azucar 10.50
Pollo  30.25
------------
Total  57.75
```

ha sido generada con el siguiente código, donde `prices` es un diccionario cuyas claves son nombres de productos, y los valores son sus correspondientes precios:

```
total = 0.0
for product, price in prices.items():
    print(f'{product:7}{price:5.2f}')
    total += price
print("------------")
print(f"Total  {total:5.2f}")
```

Como se puede observar, se usa una f-string en la que se interpolan las variables cuyos valores se quiere mostrar:

```
f'{product:7}{price:5.2f}'
```

pero, en lugar de estar sólo los nombres de las variables, se han añadido modificadores de formato, que son los textos que están entre los dos puntos y el cierra llaves. Los modificadores de formato controlan cómo va a aparecer el resultado de la expresión a la izquierda de los dos puntos.

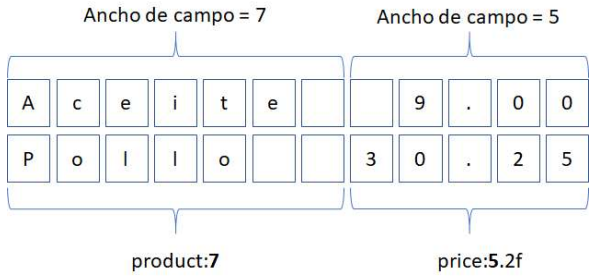
Information

Modificadores de formato - ancho de campo

Los modificadores de formato controlan cómo va a aparecer el resultado de la expresión a la izquierda de los dos puntos.

```
f'{product:7}{price:5.2f}'
```

El primer número del modificador de formato indica el "ancho del campo" (número de caracteres) en el que se va a mostrar el valor.



Por omisión, el texto se alinea a la izquierda del campo especificado si es una string y a la derecha si es un número, pero puede especificarse qué tipo de justificación se quiere, anteponiendo al ancho de campo los símbolos < para alinear a la izquierda, ^ para centrar y > para alinear a la derecha:

```
value = "align left"
print(f"[{value:<20}]")
```

```
[align left          ]
```

```
value = "align center"
print(f"[{value:^20}]")
```

```
[    align center    ]
```

```
value = "align right"
print(f"[{value:>20}]")
```

```
[         align right]
```

Hay que tener en cuenta que el modificador de ancho de campo indica un mínimo, pero el campo se extenderá cuanto sea necesario para representar completamente el valor formateado. Si por el contrario fuera suficiente, por omisión el campo sobrante se rellena con espacios tal como hemos visto, si bien puede especificarse otro valor de relleno. En el caso de que el valor a representar sea un número, un cero a la izquierda del ancho de campo indica que se rellene con ceros:

```
for num in range(5, 11):
    print(f"{num:03}")
```

```
005
006
007
008
009
010
```

En cualquier otro caso, el carácter de relleno tiene que ir seguido de un código de alineación (<^>) antes del ancho de campo:

```
for num in range(5, 11):
    print(f"{num:*>3}") # > indica alineación derecha
```

```
**5
**6
**7
**8
**9
*10
```

Question 2

Complete

Mark 1.00 out of 1.00

¿Qué valor se asigna a la variable *test_string*?

```
test_string = f"{'hola':^10}"
```

Select one:

- `"hola      "`
- `"   hola   "`
- `"      hola"`

Information

Modificadores de formato - Precisión

Un punto seguido de un número indica la precisión con la que se quiere formatear el valor. Véanse los siguientes ejemplos.

```
import math
print(math.pi)              # 3.141592653589793
print(1000000/3)            # 333333.3333333333
print(0.0000555)            # 5.55e-05
```

Si no se especifica formato, al convertirlos a string (en los anteriores ejemplos, automáticamente, antes de ser visualizados con print) los valores *float* se representan íntegramente, reflejando la precisión con la que se almacenan internamente (obsérvese el número de cifras para π y para el resultado de 1000000/3). La notación `5.55e-05` expresa $5.55 \cdot 10^{-5}$.

```
print(f"{math.pi:.2}")      # 3.1
print(f"{0.00555:.2}")      # 0.0056
print(f"{555.5:.2}")        # 5.6e+02
```

La precisión 2 aquí se refiere a los 2 **primeros dígitos significativos** (se redondea apropiadamente). Se incluye al menos un dígito a la derecha del punto. No aplicable a valores de tipo *int*.

```
print(f"{math.pi:.2f}")     # 3.14
print(f"{0.00555:.2f}")     # 0.01
print(f"{555:.2f}")         # 555.00
```

La `f` del final indica ***f**ixed point*: la parte entera (incluso si es 0) se representará a la izquierda del punto, y a la derecha de éste se incluirán exactamente (tras el apropiado redondeo) los 2 **primeros dígitos fraccionarios** (incluso los ceros)**.**

```
print(f"{math.pi:5.2f}")    # ␣3.14
print(f"{555:5.2f}")        # 555.00
```

Lo mismo que el caso anterior, pero en un campo de ancho 5 (al menos).

```
print(f"{str(math.pi):.2}")  # 3.
```

El valor *float* del número π se ha convertido primero a un valor de tipo string, que es lo que se formatea, en este caso recortándola y quedando únicamente los 2 **primeros caracteres** de la misma (nótese la diferencia con respecto al caso en que la misma especificación de formato se aplica sobre un valor de tipo numérico).

Question 3

Complete

Mark 0.40 out of 1.00

Empareje cada valor con la sentencia *print()* que lo produce.

3.14	print(f"{math.pi:5.2f}")
3.1	print(f"{str(math.pi):.2}")
3.141592653589793	print(math.pi)
3.	print(f"{math.pi:.2f}")
3.14	print(f"{math.pi:5.2f}")

Information

Modificadores de formato - Notación numérica

Al final del modificador de formato se puede añadir una letra para especificar cómo queremos interpretar el tipo de datos. Véanse los siguientes ejemplos:

```
print(f"{12:5o}")
```

```
   14
```

El valor entero 12 (por omisión, en la usual base diez) se imprime en octal (base ocho, que solo usa los dígitos del 0 al 7), en un campo de tamaño 5.

```
print(f"{12:5x}")
```

```
    c
```

```
print(f"{12:5X}")
```

```
    C
```

El valor entero se imprime en hexadecimal (base dieciséis, que usa los dígitos 0-9 y a-f). Las letras 'a', 'b', 'c', 'd', 'e' y 'f' de la representación hexadecimal se muestran en mayúscula si el modificador 'X' está en mayúscula.

```
print(f"{12:5.1f}")
```

```
 12.0
```

Un valor, entero o flotante, se imprime como *float* (si el valor es entero, se añaden ceros como parte fraccionaria)

```
print(f"{-140:5.1e}")
```

```
-1.4e+02
```

```
print(f"{0.014:5.1E}")
```

```
1.4E-02
```

Un valor, entero o flotante, se imprime en notación científica (coma flotante) indicando el exponente de la potencia de diez: $-140 = -1.4\times10^{2}$, $0.014 = 1.4\times10^{-2}$. Obsérvese que se muestra "e" o "E" según el modificador.

```
print(f"{12:5d}")
```

```
   12
```

Un valor *entero* se imprime en formato decimal (la usual base diez, que usa los dígitos del 0 a 9).

Carácter de signo

Podemos controlar qué ocurre al mostrar el signo cuando el número no es negativo. Ejemplos:

```
print(f"{-12:5d}")
```

```
  -12
```

Por omisión, el carácter para el signo (− o +) sólo se muestra si el número es negativo.

```
print(f"{-12:+5d}")
```

```
  -12
```

```
print(f"{12:+5d}")
```

```
  +12
```

Si al principio de los modificadores de formato añadimos el carácter '+' se mostrará, tanto el signo '−' como el '+', en función de si el valor numérico es negativo o no.

```
print(f"{-12:-2d}")
```

```
-12
```

```
print(f"{12:-2d}")
```

```
12
```

```
print(f"{-12: 2d}")
```

```
-12
```

```
print(f"{12: 2d}")
```

```
␣12
```

Añadir un carácter '−' no afecta al resultado (sólo se muestra el signo cuando el valor es negativo), pero añadir un espacio en su lugar muestra un espacio (que aquí hemos representado por ␣) en lugar del signo cuando el valor no es negativo.

Question 4

Complete

Mark 1.00 out of 1.00

¿Qué modificador de formato se ha usado para mostrar el siguiente valor númerico?

FF2E

Select one:

- O
- x
- X
- o

Information

El método `format()`

Una alternativa a la interpolación de strings que, aunque propia de versiones de Python anteriores a la introducción de este mecanismo, sigue disponible es el método `format()`:

```
num1 = 12 / 7
print("Primero = {:4.2f}, segundo = {:2X}".format(num1, 10))
```

```
Primero = 1.71, segundo =  A
```

Como se ve en el ejemplo, es muy similar a la interpolación de strings, salvo que las expresiones cuyos valores queremos incluir en la string no se colocan en su sitio dentro de ella, sino que se pasan como parámetros del método *format()*, ocupando el lugar que tienen reservado en la string, en principio por orden de aparición.

Existe la posibilidad de dar un nombre a cada campo, lo que independiza su orden en la string del orden en que se pasan al método `format()`.

```
print("Primero = {num1:4.2f}, segundo = {num2:2X}".format(num1=num1, num2=10))
print("Segundo = {num2:2X}, primero = {num1:4.2f}".format(num1=num1, num2=10))
```