Status	Finished
Started	Saturday, 23 November 2024, 4:32 PM
Completed	Saturday, 23 November 2024, 4:41 PM
Duration	8 mins 58 secs
Marks	13.00/13.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Sentencia for

Un esquema de iteración consiste en repetir una secuencia de acciones mientras se cumpla una condición. En una variante común del esquema, la condición deja de cumplirse cuando se han realizado un número de iteraciones que puede calcularse a priori con facilidad. El siguiente ejemplo muestra, de menor a mayor, los números enteros positivos menores que uno dado (límite), para lo que el bucle debe iterar límite - 1 veces:

```
actual = 1
límite = int(input("Dame un valor entero positivo: "))

while actual < límite:
    print(actual)
    actual = actual + 1
```

Dada la frecuencia de esta variante, existe una sentencia de repetición específica, que permite expresarla de forma más concisa que la sentencia while:

```
límite = int(input("Dame un valor entero positivo: "))
for actual in range(1, límite):
    print(actual)
```

```
líneas_a_escribir = int(input("¿Líneas a escribir? "))

for líneas_escritas in range(líneas_a_escribir):
    print("Hola")
```

Básicamente, una variable de control (*actual*, *líneas_escritas*) va tomando secuencialmente valores de un rango y, cada vez que toma un nuevo valor, se ejecutan las instrucciones anidadas en la sentencia *for*. Nótese que si se pone un solo valor, como en el ejemplo de líneas_a_escribir, el rango va desde cero hasta el valor previo al especificado.

```
range(10)    -> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

```
range(1, 10) -> 1, 2, 3, 4, 5, 6, 7, 8, 9
```

La función *range* define, en principio, una secuencia de números enteros consecutivos, pero admite un tercer parámetro que permite establecer un "paso" o distancia entre los valores sucesivos. El siguiente ejemplo muestra los números impares positivos menores que 15, por lo que la variable de control va saltando de 2 en 2 a partir del valor inicial:

```
for actual in range(1, 15, 2):
    print(actual, end = ', ')
```

Se muestra: 1, 3, 5, 7, 9, 11, 13,

El paso puede ser negativo, siempre que el rango esté invertido. En el siguiente ejemplo, se muestran, de mayor a menor, los números impares positivos menores o igual que uno dado (el valor inicial es top, o top - 1, si top es par, y la variable de control avanza de -2 en -2):

```
top = int(input("Dame un número entero positivo: "))
```

```
if top % 2 == 0: # Nos aseguramos de empezar con valor impar
    top -= 1
```

```
for actual in range(top, 0, -2):
    print(actual)
```

Si se introduce en la entrada el 8, se muestra:

7
5
3
1

Obsérvese que *range* puede suministrar cero valores —ejemplos: `range(0)`, `range(-4)`, `range(2, 2)`, `range(3, 2)`, `range(2, 5, -1)`—, por lo que el *for* no realizaría iteración alguna, terminando inmediatamente.

Question 1

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para mostrar los números del 0 al 15:

```
for item in range(16):
    print(item)
```

Question 2

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para mostrar los números: 8, 11, 14, 17, 20.

```
for item in range(8, 23, 3):
    print(item)
```

Information

Equivalencia entre for y while

La sentencia *for* es más específica que la *while*; es decir, todo lo que se puede hacer usando la sentencia *for* se puede hacer, aunque de forma menos concisa, usando la sentencia *while*. El siguiente ejemplo muestra la equivalencia entre la sentencia *for* y la sentencia *while*:

Sentencia for

```
for value in range(first, last, step):
    hacer algo
```

Sentencia while

```
value = first

while value < last:
    hacer algo
    value += step
```

La principal diferencia entre ambas versiones es que en el *for* la inicialización de la variable de control y su incremento al final de la iteración son automáticos, mientras que en el *while* hay que indicarlos de forma explícita.

Todo lo que se puede hacer con un *for* se puede hacer con un *while*, pero la recíproca no es cierta. Por ejemplo, el bucle *while* del siguiente ejemplo no puede sustituirse por un *for* ya que no se conoce por adelantado el rango de valores que va a tomar la variable num2 hasta llegar a cero (calcularlo costaría tanto como ejecutar el algoritmo):

```
num1 = int(input("Dame un número entero positivo: "))
num2 = int(input("Dame un número entero positivo menor que el anterior: "))
```

```
while num2 > 0:
    mod = num1 % num2
    num1 = num2
    num2 = mod
```

Question 3

Complete

Mark 6.00 out of 6.00

Complete el código de la derecha usando los mismos nombres de variables que el de la izquierda para que ambos sean equivalentes:

Sentencia for

```
for value in range(first, last, step):
    hacer algo
```

Sentencia while

```
value = first

while value < last :
    hacer algo
    value += step
```

Information

La sentencia for y el tratamiento de secuencias

La función *range* del siguiente ejemplo define una secuencia de números enteros que debe ir tomando la variable de control del bucle *for*:

```
for actual in range(1, 15):
    print(actual)
```

En realidad, la sentencia *for* admite cualquier clase de secuencia. En el siguiente ejemplo se muestran los días de la semana que han sido almacenados previamente en una tupla:

```
days = ("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo")

for day in days:
    print(day)
```

El siguiente ejemplo cuenta el número de letras 'e' que hay en la string referenciada por la variable text:

```
text = "Dame un número entero positivo: "
count = 0

for char in text:
    if char == 'e':
        count += 1

print(count)
```

El bucle *for* requiere que una variable de control recorra una secuencia de valores; da igual el tipo de secuencia de que se trate.

En general, las secuencias pueden contener cero elementos, en cuyo caso esta forma de *for* no realizaría iteración alguna, terminando inmediatamente.

Hay que aclarar que el operador *in*, cuando se usa como en los ejemplos anteriores indica el acceso a cada uno de los elementos de una secuencia, pero también puede usar en una condición para evaluar la pertenencia de un elemento a una secuencia/conjunto/contenedor, como en le siguiente ejemplo, que evalúa si la letra 'e' está en un texto:

```
if 'e' in text:
    print(True)
else:
    print(False)
```

Question 4

Complete

Mark 2.00 out of 2.00

Complete el siguiente código para que muestre todos los valores de la tupla months.

```
months = (
    "enero", "febrero", "marzo", "abril", "mayo", "junio", "julio",
    "agosto", "septiembre", "octubre", "noviembre", "diciembre"
)

for month in [ months ]:
    print([ month ])
```

Information

Acceso a los índices al iterar sobre una secuencia

Nótese, que al iterar sobre una secuencia se va accediendo a sus elementos pero no se tiene información sobre la posición que ocupan dichos elementos en la secuencia.

```
days = ("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo")

for day in days:
    print(day)
```

Se muestra:

lunes
martes
miércoles
jueves
viernes
sábado
domingo

Si fuese necesario conocer el índice de los elementos, se puede usar la función *enumerate*, e iterar usando una pareja de variables:

```
days = ("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo")

for num_day, day in enumerate(days):
    print(num_day, "-", day)
```

Se muestra:

0 - lunes
1 - martes
2 - miércoles
3 - jueves
4 - viernes
5 - sábado
6 - domingo

Básicamente, lo que hace *enumerate* es tomar un objeto iterable (que se puede "recorrer") y añadirle un contador, devolviendo una nueva secuencia de elementos formados por parejas índice, valor.

Se puede obtener el mismo resultado iterando sobre el rango de índices de la secuencia a recorrer, como en el siguiente ejemplo:

```
days = ("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo")

for num_day in range(len(days)):
    print(num_day, "-", days[num_day])
```

Question 5

Complete

Mark 1.00 out of 1.00

Complete el siguiente código para que muestre los días de la semana junto con el índice de la posición que ocupan en la tupla *days*.

```
days = ("lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo")

for num_day, day in enumerate(days):
    print(num_day, "-", day) # num_day representa el índice de la posición que ocupa day en days
```