Status	Finished
Started	Saturday, 23 November 2024, 4:42 PM
Completed	Saturday, 23 November 2024, 4:49 PM
Duration	7 mins 52 secs
Marks	6.00/6.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Esquemas de recorrido y acumulación

Muchísimos problemas requieren el tratamiento de secuencias de valores aplicando una acción de forma repetida a cada uno de los valores de la secuencia, lo que se hace siguiendo los esquemas básicos determinados por un bucle *while* o un bucle *for*:

Bucle for

```
for item in range numbers:
    Hacer algo
```

Bucle while

```
Inicializaciones
```

```
while condición:
    Hacer algo
    Avanzar
```

Sobre este esquema básico se pueden hacer múltiples variaciones. A modo de ilustración, el Ejemplo 1 muestra un **esquema de recorrido** y el Ejemplo 2 un **esquema acumulador**.

Ejemplo 1: esquema de recorrido

Bucle for

```
numbers = (21, 13, 24, 42, 12)
for item in numbers:
    print(item)
```

Bucle while

```
numbers = (21, 13, 24, 42, 12)
index = 0
```

```
while index < len(numbers):
    print(numbers[index])
    index += 1
```

Ejemplo 2: esquema de acumulación

Bucle for

```
numbers = (21, 13, 24, 42, 12)
addition = 0
for item in numbers:
    addition += item


print(addition)
```

Bucle while

```
numbers = (21, 13, 24, 42, 12)
index = 0
addition = 0
```

```
while index < len(numbers):
    addition += numbers[index]
    index += 1

print(addition)
```

La diferencia entre el primer y segundo esquema es que el de recorrido trata una secuencia realizando una acción con cada uno de sus elementos, acción que sólo depende del elemento en cuestión y no de ningún otro elemento de la secuencia, mientras que en el esquema de acumulación la acción va combinando los elementos para obtener un valor final, una vez acabado el bucle. Habitualmente se asocia el esquema de acumulación con cualquier operación que tenga elemento neutro (para inicializar el proceso) y sea asociativa, como la adición o la multiplicación numéricas, o la concatenación de strings; pero podríamos generalizarlo para cubrir cualquier algoritmo en que se calcule un dato en función del conjunto de elementos de una secuencia, combinando los valores de los elementos de alguna forma.

Combinando la estructura de repetición con otras sentencias de control se pueden obtener esquemas más sofisticados.

Question 1

Complete

Mark 1.00 out of 1.00

Relacione cada operación con su elemento neutro.

1.0	multiplicación de números reales
0.0	adición de números reales
""	concatenación de strings
0	adición de números enteros
1	multiplicación de números enteros

Information

Esquema de selección

Combinando un bucle con una sentencia de selección se obtiene un **esquema de selección** como el del siguiente ejemplo, que muestra sólo los valores impares de una tupla de números enteros.

Bucle for

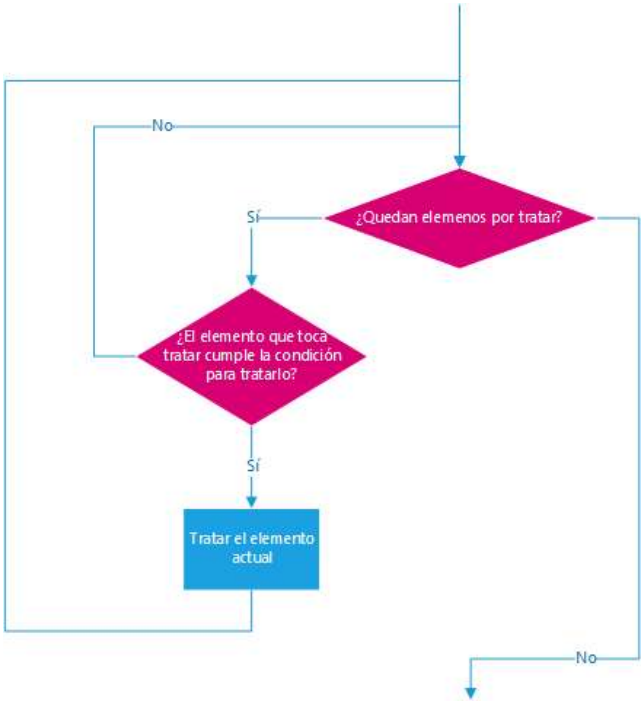
```
numbers = (21, 13, 24, 42, 12)
for number in numbers:
    if number % 2 == 1:
        print(number)
```

Bucle while

```
numbers = (21, 13, 24, 42, 12)
index = 0
```

```
while index < len(numbers):
    number = numbers[index]
    if number % 2 == 1:
        print(number)

    index += 1
```



Un esquema de selección, como su nombre indica, aplica una acción sólo a los elementos de la secuencia que cumplen una condición determinada. También podría considerarse la variante de aplicar un tratamiento a los valores que cumplen una condición y otro distinto, a los que no la cumplen.

Bucle for

```
numbers = (21, 13, 24, 42, 12)
for number in numbers:
    if number % 2 == 1:
        print(number)
    else:
        print("*")
```

Bucle while

```
numbers = (21, 13, 24, 42, 12)
index = 0
```

```
while index < len(numbers):
    number = numbers[index]
    if number % 2 == 1:
        print(number)
    else:
        print("*")

    index += 1
```

Question 2

Complete

Mark 3.00 out of 3.00

Complete el siguiente código para que muestre todos los números pares de la secuencia *numbers*:

```
numbers = (21, 13, 24, 42, 12)
index = 0
```

```
while index < len(numbers):
    if [numbers[index]] % 2 == [0]:
        print([numbers[index]])

    index += 1
```

Information

Esquema de búsqueda

El siguiente ejemplo localiza el primer número par en una tupla de números enteros:

Bucle for

```
numbers = (19, 12, 35, 21, 12, 41, 37)
found = None

for number in numbers:
    if number % 2 == 0:
        found = number
        break

if found != None:
    print("El primer número par es:", found)
else:
    print("No hay números pares en la tupla")
```

Bucle while

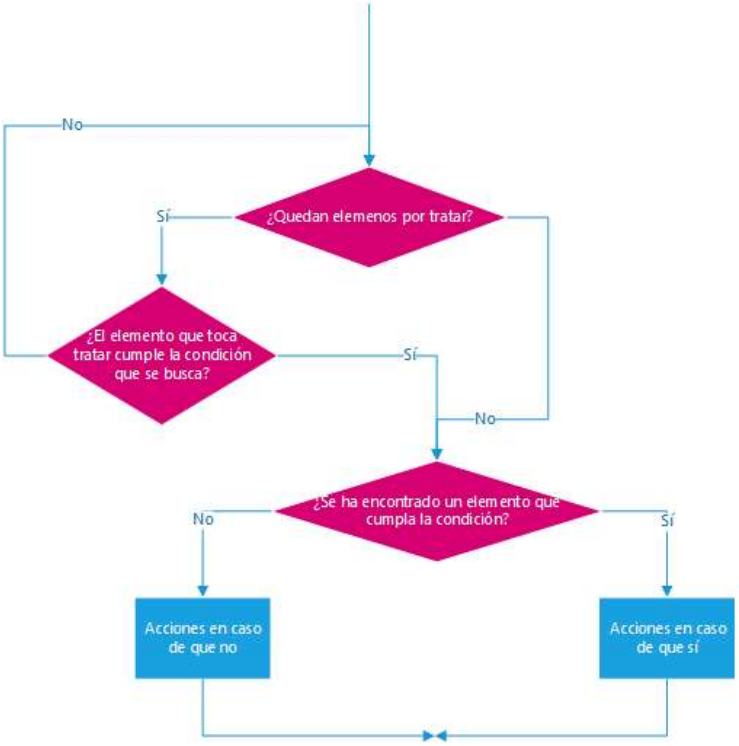
```
numbers = (19, 12, 35, 21, 12, 41, 37)
index = 0
found = None

while index < len(numbers):
    number = numbers[index]
    if number % 2 == 0:
        found = number
        break

    index += 1
```

```
if found != None:
    print("El primer número par es:", found)
else:
    print("No hay números pares en la tupla")
```

El ejemplo sigue un **esquema de búsqueda**, que es similar a un esquema de selección en cuanto que incluye una sentencia *if* para comprobar si el elemento actual de la secuencia cumple una condición, pero diferenciándose en que cuando ello ocurre se ejecuta la instrucción ***break***, la cual hace que se interrumpa la ejecución normal del bucle, dándolo por finalizado (la iteración actual termina, y ya no se realizan nuevas iteraciones).



Un esquema de búsqueda avanza mientras queden elementos que examinar y no se haya encontrado ninguno que cumpla la condición de búsqueda. Por tanto, el bucle puede terminar, bien porque se acabe la secuencia sin encontrar ningún elemento que cumpla la condición de búsqueda, bien porque se haya encontrado un elemento que la cumpla. Si después del bucle es necesario saber si se ha encontrado o no, habrá que usar una sentencia *if* para discernirlo.

Cuando se usa un *while*, el esquema de búsqueda puede evitar el *if* y el *break* usando una condición doble en el *while*:

```
numbers = (19, 13, 35, 21, 13, 41, 37)
index = 0
```

```
while index < len(numbers) and numbers[index] % 2 != 0:
    index += 1
```

```
if index < len(numbers):
    print("El primer número par es:", numbers[index])
else:
    print("No hay números pares en la tupla")
```

Como puede observarse, en este caso solo la variable *index* nos da la pista para saber si se ha encontrado un elemento que cumple la condición, y cuál es en ese caso. De hecho, a veces más que el valor del elemento en sí lo que nos interesa es su posición en la secuencia.

En todos los ejemplos hemos supuesto que solo nos interesa el primer elemento que cumple la condición, independientemente de que después de él haya otro(s) que también la cumplan.

Question 3

Complete

Mark 1.00 out of 1.00

Complete el siguiente código para encontrar el primer número par en la secuencia numbers.

```
numbers = (19, 12, 35, 21, 12, 41, 37)
found = None

for number in numbers:
    if number % 2 == 0:
        found = number
        break

if found != None:
    print("El primer número par es:", found)
else:
    print("No hay números pares en la tupla")
```

Information

Alteración del flujo de un bucle

Existen dos instrucciones que permiten alterar el flujo normal de ejecución de un bucle, ambas aplicables a cualquier tipo de bucle, sea *while* o *for*. La instrucción ***break*** interrumpe la ejecución del bucle terminándolo completamente de forma anticipada. El siguiente ejemplo muestra cada elemento de la tupla *numbers* emparejado con su sucesor (en la sucesión de los números enteros), hasta que se encuentre el primer elemento par, en cuyo caso en lugar del sucesor se muestra la palabra *end* y se interrumpe el bucle, no realizándose el resto de la iteración actual (que hubiera mostrado el sucesor) ni tratándose los restantes elementos de la tupla.

```
numbers = (19, 21, 35, 21, 12, 41, 28, 37)

for element in numbers:
    print(element, end = "-")

    if element % 2 == 0:
        print("end")
        break

    print(element + 1)
```

El resultado es:

19-20
21-22
35-36
21-22
12-end

La instrucción ***continue***, por el contrario, termina la iteración actual, sin ejecutar las instrucciones que falten para esta, pero no termina el bucle, sino que pasa a intentar la iteración siguiente (lo mismo que si la iteración actual hubiese concluido normalmente).

```
numbers = (19, 21, 35, 21, 12, 41, 37)

for element in numbers:
    print(element, end = "-")

    if element % 2 == 0:
        print()
        continue

    print(element + 1)
```

El resultado del ejemplo anterior es:

19-20
21-22
35-36
21-22
12-
41-42
37-38

Podemos resumir diciendo que *continue* salta al principio del bucle mientras que *break* salta fuera del bucle.

Question 4

Complete

Mark 1.00 out of 1.00

¿Qué instrucción se usa para interrumpir la ejecución de una iteración de un bucle e iniciar la siguiente?

Answer: continue