Status	Finished
Started	Saturday, 23 November 2024, 4:58 PM
Completed	Saturday, 23 November 2024, 5:04 PM
Duration	5 mins 49 secs
Marks	8.80/9.00
Grade	**9.78** out of 10.00 (**97.78**%)

Information

Funciones de manejo de listas

Python ofrece algunas funciones estándar útiles para manejar listas, por ejemplo:

len(*list*)	devuelve el número de elementos de una lista (su longitud)
sum(*list*)	devuelve la suma de los elementos de una lista
max(*list*)	devuelve el valor máximo de los elementos de una lista
min(*list*)	devuelve el valor mínimo de los elementos de una lista

Lógicamente, `sum()` sólo se puede aplicar a listas con elementos numéricos, y `max()` y `min()` a listas con valores comparables entre sí.

Question 1

Complete

Mark 4.00 out of 4.00

Dada cualquier lista t de números. por ejemplo:

```
t = [21, 22, 25, 25, 24, 21, 20]
```

Complete las siguientes operaciones para cualquier lista t usando las funciones adecuadas:

```
average = sum(t) / len(t)    # Media de los elementos de t
minimum = min(t)             # Mínimo de los elementos de t
maximum = max(t)             # Máximo de los elementos de t
```

Information

Función `list()`

La función **`list()`** crea una lista con los elementos de una secuencia.

```
a = list(range(1, 10))  # [1, 2, 3, 4, 5, 6, 7, 8, 9]
b = list('Python')      # ['P', 'y', 't', 'h', 'o', 'n']
c = list()              # []
d = ('milk', 'bread', 'butter','honey')  # tupla
e = list(d)             # ['milk', 'bread', 'butter', 'honey']
f = list(a)
```

En el último ejemplo, `f` recibe una **nueva** lista, copiada de la lista a.

Question 2

Complete

Mark 1.00 out of 1.00

La función `list()` devuelve una nueva lista con elementos de la secuencia pasada como parámetro

Select one:

- True
- False

Information

Métodos de manejo de listas sin modificación

En Python los objetos de tipo `list` también admiten algunos *métodos*, que son funciones especiales que han de ser llamadas indicando el objeto sobre el que se aplican, seguido de un punto: por ejemplo, dada la lista `mylist`, los siguientes dos métodos proporcionan información sobre la misma (sin cambiar su contenido):

```
a = mylist.count(x)              # Devuelve el número de veces que x aparece en mylist
b = mylist.index(x)              # Devuelve la posición de la primera ocurrencia de x en mylist
c = mylist.index(x, start, end)  # Devuelve la posición de la primera ocurrencia de x en mylist
                                 # en el rango de índices start y end
```

Ejemplo:

```
fruits = ['banana', 'apple', 'pear', 'cherry', 'banana', 'banana']
print(fruits.count('banana'))                 # 3
print(fruits.index('banana'))                 # 0
print(fruits.index('banana', 3, len(fruits)))  # 4
```

Debemos tener cuidado al usar el método `index`, ya que si el valor buscado no se encuentra en la lista se lanza la excepción **`ValueError`**. Podemos prevenirlo usando el operador *in*:

```
if 'peach' in fruits:
    print(fruits.index('peach'))
else:
    print('There is no "peach" in this list.')
```

Question 3

Complete

Mark 1.00 out of 1.00

¿Qué valores muestra el siguiente código?

```
a = list('abcd') * 2

print(len(a))
print(a.count('a'))
print(a.index('b'))
```

Select one:

- Ninguno, se produce un *ValueError*
- 8, 2, 2
- 8, 2, 1
- 4, 1, 1

Information

Métodos que modifican listas

A los siguientes métodos se les llama (del inglés) "*mutables*", pues modifican el contenido de la lista sobre la que se aplican (no crean una nueva lista).

Sea `mylist` una lista cualquiera:

```
mylist.append(x)          # Añade un nuevo elemento con el valor x al final de mylist
mylist.insert(i, x)       # Inserta un nuevo elemento con el valor x en la posición de mylist indicada por i
value = mylist.pop()      # Quita de mylist su último elemento y lo devuelve; se asigna a la variable value
value = mylist.pop(i)     # Quita de mylist el elemento indicado por i y lo devuelve; se asigna a la variable value
mylist.remove(x)          # Quita de mylist el primer elemento cuyo valor sea igual a x
mylist.sort()             # Ordena mylist de menor a mayor valor
mylist.sort(reverse=True) # Orena mylist de mayor a menor valor
mylist.sort(key=getKey)   # Siendo getKey el nombre de una función que toma como parámeto un elemento
                          # Ordena la lista en función de los resultados de getKey al aplicarse a cada
                          # elemento de la lista. Puede combinarse con reverse.
mylist.extend(other)      # Añade al final de mylist los elementos de la secuencia other
mylist.clear()            # Vacía mylist, quitando todos los elementos
```

Tanto `pop()` como `remove()` lanzan la excepción ValueError si no existe elemento a extraer. Si el índice de `insert` es mayor o igual que la longitud de la lista, añade al final.

Existe una función:

```
sorted(c)→ list sorted copy
```

que, siendo c una secuencia/contenedor, devuelve una lista ordenada con los elementos de c; por ejemplo:

```
c = "cabrito"
s = sorted(c)
print(s)  # ['a', 'b', 'c', 'i', 'o', 'r', 't']
```

La función sorted() admite los mismos parámetros opcionales que el método .sort() -reverse y key.

Information

Métodos que modifican listas

Ejemplo de código:

```
fruits = ['banana', 'apple', 'pear', 'cherry', 'apple', 'banana']
fruits.append('peach')
print(fruits)              # ['banana', 'apple', 'pear', 'cherry', 'apple', 'banana', 'peach']
print(fruits.pop())        # peach
print(fruits.pop(0))       # banana
fruits.insert(0, 'coconut')
fruits.remove('apple')
print(fruits)              # ['coconut', 'pear', 'cherry', 'apple', 'banana']
fruits.reverse()
print(fruits)              # ['banana', 'apple', 'cherry', 'pear', 'coconut]
fruits.sort()
print(fruits)              # ['apple', 'banana', 'cherry', 'coconut', 'pear']

def lastChar(string):      # Obtiene el último carácter de una string
    return string[-1]

fruits.sort(key=lastChar, reverse=True)  # Ordena por el último caracter, descendente
print(fruits)              # ['cherry', 'coconut', 'pear', 'apple', 'banana']
fruits.extend(['orange', 'lemmon'])
print(fruits)              # ['cherry', 'coconut', 'pear', 'apple', 'banana', 'orange', 'lemmon']
fruits.clear()
print(fruits)              # []
```

En el siguiente ejemplo se añaden elementos a una lista vacía para que contenga todos los números enteros no negativos menores que 1000 divisibles por 3 o por 5 o por ambos:

```
numbers = []
for i in range(1000):
   if i % 3 == 0 or i % 5 == 0:
      numbers.append(i)
```

Question 4

Complete

Mark 0.80 out of 1.00

Empareje las operaciones de la secuencia con el valor resultante asociado a la lista a.

`a = list(range(1, 6))`	[1, 2, 3, 4, 5]
`a.remove(3)`	[1, 2, 3, 4, 5]
`a.pop()`	[1, 2, 4]
`a.insert(-1, 0)`	[1, 2, 0, 4]
`a.append(0)`	[1, 2, 0, 4, 0]

Information

Slices

Igual que con las strings o las tuplas, podemos obtener sublistas (*slices*, segmentos) a partir de una lista. La sintaxis completa es:

```
lista[inicio : fin : paso]
```

La operación de slice no modifica la lista original, sino que crea una nueva.

Ejemplo de código:

```
cities = ['Bratislava', 'Warsaw', 'Madrid', 'Praha']
a = cities[1:3]   # ['Warsaw', 'Madrid']
b = cities[-3:]   # ['Warsaw','Madrid', 'Praha']
c = cities[:-1]   # ['Bratislava','Warsaw', 'Madrid']
d = cities[1::2]  # ['Warsaw', 'Praha']
e = cities[::-1]  # ['Praha', 'Madrid', 'Warsaw', 'Bratislava']
f = cities[:]     # crea una nueva lista ['Bratislava', 'Warsaw', 'Madrid', 'Praha']
```

Question 5

Complete

Mark 1.00 out of 1.00

Dada la lista *names*:

```
names = ['Matthew', 'Mark', 'Luke', 'John', 'Francis']
```

Empareje cada lista con la expresión que la produce.

['Matthew', 'Luke', 'Francis']	names[::2]
['John', 'Francis']	names[3::]
['Matthew', 'Mark', 'Luke']	names[:3:]
['John']	names[3:4]
['Francis', 'John', 'Luke', 'Mark', 'Matthew']	names[::-1]

Information

Asignación a slices

Cuando se toma un slice de una lista siempre se crea una lista nueva sin modificar la original.

La notación de slices puede usarse también a la izquierda de una asignación (como destino de la asignación); en ese caso, la parte derecha de la asignación (el origen de la asignación) debe ser una expresión que represente una secuencia (no obligatoriamente una lista). El funcionamiento de la asignación es como sigue:

1. Se evalúa la parte derecha de la asignación creando una lista nueva.
2. La nueva lista sustituye a la sublista representada por el slice, modificando la lista a la que el slice hace referencia.

Ejemplos en secuencia de instrucciones:

1) Se sustituyen tres elementos por otros tres.

```
names = ['Matthew', 'Mark', 'Luke', 'John', 'Francis']
names[1:4] = ['Paul', 'Peter', 'Thomas'] # ['Matthew','Paul', 'Peter', 'Thomas', 'Francis']
```

2) Tres elementos se sustituyen por dos.

```
names = ['Matthew', 'Mark', 'Luke', 'John', 'Francis']
names[1:4] = ['Paul', 'Peter'] # ['Matthew', 'Paul', 'Peter', 'Francis']
```

3) Los dos últimos elementos son sustituidos por tres.

```
names[-2:] = ['Andrew', 'James','Philip'] # ['Matthew', 'Paul', 'Andrew', 'James', 'Philip']
```

4) Los dos primeros elementos son sustituidos por una lista vacía.

```
names[:2] = [] # ['Andrew', 'James', 'Philip']
```

5) El primer elemento se sustituye por seis elementos de una lista construida a partir de una string.

```
names[:1] = 'Python' # ['P', 'y', 't', 'h', 'o', 'n', 'James', 'Philip']
```

Information

Comparación de listas

Se pueden comparar dos listas usando los operadores relacionales:

- Se van comparando de principio a fin los elementos que ocupan las mismas posiciones en las dos listas hasta que se encuentra una pareja diferente o se acaba alguna de las listas.
- Si se encuentra una pareja de elementos diferentes, es menor la lista que tenga el elemento menor.
- Si se acaba una de las listas, sin encontrar una pareja diferente, y la otra no, es menor la lista de menor longitud.
- Si se llega a la vez al final de las dos listas sin encontrar una pareja de elementos diferentes, las listas son iguales.

Ejemplos en secuencia de instrucciones:

```
a = list(range(1,5))   #  [1, 2, 3, 4]
b = list(range(1,5))   #  [1, 2, 3, 4]
a == b     # True
a != b     # False
a[1] = 0
a == b     # [1, 0, 3, 4] == [1, 2, 3, 4] da False
a < b      # [1, 0, 3, 4] < [1, 2, 3, 4] da True
```

En la lista *b*, el elemento con índice 1 es mayor que el elemento correspondiente de la lista *a*.

```
a <= b      # True
a > b       # False
a >= b      # False
[1, 2] < b  # [1, 2] < [1, 2, 3, 4] da True
```

Los primeros dos elementos en las dos listas son iguales, pero la primera lista es más corta.

Question 6

Complete

Mark 1.00 out of 1.00

Marque las expresiones que dan como resultado *True*.

Select one or more:

- [ ] [1, 2, 3] == ['mother', 'father', 'child']
- [x] [1, 2, 3] < [1, 2, 7]
- [ ] [1, 2, 3] < [1, 2]
- [x] [1, 2, 3] >= [1, 2]
- [x] [ ] == list()