Status	Finished
Started	Friday, 22 November 2024, 11:30 AM
Completed	Friday, 22 November 2024, 11:33 AM
Duration	3 mins 4 secs
Marks	4.00/4.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Tuplas

Los tipos básicos de datos (int, float, bool) representan valores monolíticos, que no se pueden separar en componentes individuales con entidad propia. Por su parte las strings (str) representan valores compuestos, formados por una secuencia de caracteres, aunque el sentido de lo que representan suele venir dado por el conjunto, más que por los caracteres individuales. Los números complejos (complex) también son valores compuestos, formados por una pareja de números reales.

Hay muchas clases de información que se pueden entender como formadas por datos que son un agregado de datos más simples con entidad propia. Ejemplos: un nombre español completo está formado habitualmente por tres strings: nombre de pila, primer apellido y segundo apellido; un punto en el espacio bidimensional se representa por dos coordenadas, que son números reales; una fecha está formada por día, mes y año; etc.

En Python este tipo de información compuesta por un agregado finito de datos se puede representar de manera unitaria usando tuplas (tipo **`tuple`**). Podemos definir una n-tupla como una secuencia de n valores. En Python podemos construir una tupla encerrando una secuencia de valores, separados por coma, entre paréntesis.

```
name = ("Pedro", "Martel", "Parrilla")
point2D = (3.5, 8.3)
vector3D = (3.5, 8.2, 4.6)
dni = "42387423H"
age = 16
address = "Calle del Pino, nº 2. Las Palmas."
person_id = (dni, name, age, address)
```

El ejemplo anterior construye cuatro tuplas, referenciadas como: `name`, `point2D`, `vector3D` y `person_id`. Las variables `dni` y `address` referencian strings, no tuplas, y la variable age referencia un valor de tipo `int`. La variable `person_id` ilustra que:

1. Los elementos de una tupla pueden ser de tipos diferentes (`name` está formada solo por strings y `point2D` y `vector3D` solo por números reales, mientras que `person_id` está formada por dos strings (dni y address), un número (age) y una tupla (name)
2. Una tupla puede estar formada por elementos de cualquier tipo, incluso tuplas.

Nótese, además, que esta última tupla asignada a `person_id` **no** contiene las variables dni, name, etc. sino que referencia a los valores que en ese momento estaban referenciando dichas variables. Si posteriormente asignáramos nuevos valores a dichas variables, ello no afectará a la mencionada tupla. De hecho, en general, al construir una tupla podemos indicar para cada elemento una expresión cuyo valor resultante en ese momento será el que corresponda a dicho elemento:

```
a = 2
b = 3
t = (b, 1-b, a+b, a-b)
b = 7     # modificamos b, pero ello no afecta a la tupla
print(t)  # muestra (3, -2, 5, -1)
```

En fin, en el caso de que queramos formar una tupla con un solo elemento, debemos poner una coma detrás del elemento:

```
name = ("Pedro",)
```

Question 1

Complete

Mark 1.00 out of 1.00

¿Cuáles de las siguientes afirmaciones son ciertas?

Select one or more:

- [x] Una tupla puede incluir elementos que sean tuplas
- [ ] Los elementos de una tupla no pueden ser tuplas
- [ ] Una tupla es una secuencia de elementos del mismo tipo
- [x] Una tupla puede incluir elementos de diferentes tipos

Information

Acceso a los elementos individuales de una tupla

Una tupla es una secuencia finita de valores:

```
name = ("Pedro", "Martel", "Parrilla")
```

Una secuencia establece un orden en el sentido de que cada elemento ocupa una posición concreta:

```
Posiciones:   1ª       2ª        3ª
name      = ("Pedro", "Martel", "Parrilla")
```

En Python las posiciones de una secuencia se identifican mediante índices secuenciales, que son números enteros. El índice de la primera posición es el 0.

```
Posiciones:   1ª       2ª        3ª
name      = ("Pedro", "Martel", "Parrilla")
Índices   :   0        1         2
```

Para acceder a un elemento individual, basta con poner la tupla seguida del índice de la posición a la que queremos acceder, encerrado entre corchetes.

```
first  = name[0]  # Pedro
second = name[1]  # Martel
third  = name[2]  # Parrilla
```

El índice de la última posición es igual a la longitud de la tupla (número de elementos) menos 1. Recordando esto, también se puede acceder a los elementos de una tupla usando índices negativos relativos a la longitud de la tupla:

```
first  = name[-3]  # Pedro
second = name[-2]  # Martel
third  = name[-1]  # Parrilla (-1 nos sirve siempre como último índice)
```

Question 2

Complete

Mark 1.00 out of 1.00

Basándonos en la siguiente tupla:

tupla = (23, 45, 12, 56, 76, 67, 89, 25)

¿Cuáles de las siguientes afirmaciones son ciertas?

Select one or more:

- [ ] El índice de la última posición es 8
- [x] tupla[5] es lo mismo que tupla[-3]
- [x] tupla[5] es igual a 67
- [ ] tupla[3] es igual a 12

Information

Operaciones con tuplas

De una tupla se puede conocer su **longitud** (número de elementos) usando la función predefinida ***len***:

```
name = ("Pedro", "Martel", "Parrilla")
len_name = len(name)  # A len_name se le asigna el valor 3
```

Se pueden concatenar ("sumar") dos tuplas, obteniéndose una nueva tupla con los valores de una y otra:

```
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
tupla3 = tupla1 + tupla2  # -> (1, 2, 3, 4, 5, 6)
```

Se puede "multiplicar" una tupla por un número positivo, lo que equivale a concatenarla tantas veces como indique el número:

```
tupla1 = (1, 2, 3)
tupla2 = tupla1 * 3  # -> (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Si se "multiplica" por un número negativo o 0, se obtiene la tupla vacía, que contiene cero elementos y podemos expresar con `()`.

Se pueden comparar dos tuplas usando los operadores relacionales. La comparación se realiza comparando los elementos de principio a fin; es menor la que primero tenga un elemento menor, o, si todos los elementos son iguales hasta que se acaba una, la que tenga menos elementos.

```
tupla1 = (1, 2, 3)
tupla2 = (1, 3, 4)

iguales            = tupla1 == tupla2  # False
menor_la_1         = tupla1 < tupla2   # True
mayor_o_igual_la_1 = tupla1 >= tupla2  # False
```

Question 3

Complete

Mark 1.00 out of 1.00

¿Qué valor muestra el siguiente código?

```
t1 = (12, 13, 28, 14, 18, 15, 36)
t2 = (12, 13, 14, 18)
t3 = (28, 15, 36)
t4 = t2 + t3
print(t4 > t1)
```

Select one:

- False
- True

Information

Operaciones con tuplas (2)

Se puede construir una tupla a partir de un segmento de otra:

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
tupla2 = tupla1[3:6]  # -> (52, 26, 17)
```

El segmento deseado se identifica indicando el índice del primer elemento y el índice de la posición siguiente al último elemento. Si se omite el segundo valor, se toma el segmento desde el primer índice hasta el final:

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
tupla2 = tupla1[3:]  # -> (52, 26, 17, 81, 29)
```

Si se omite el primer valor, se toma el segmento desde el principio hasta la posición anterior a la indicada por el índice presente:

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
tupla2 = tupla1[:6]  # -> (10, 21, 13, 52, 26, 17)
```

Si se omiten ambos índices, el segmento coincide con la tupla original completa (se copia la tupla entera):

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
tupla2 = tupla1[:]  # -> (10, 21, 13, 52, 26, 17, 81, 29)
```

Se puede añadir un tercer valor entre corchetes para indicar un incremento o "paso" que permite saltarse elementos. En el siguiente ejemplo se toman los elementos desde el principio hasta el final saltando de dos en dos (índices 0, 2, 4, ···) con los que crear la nueva tupla:

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
tupla2 = tupla1[::2]  # -> (10, 13, 26, 81)
```

Nótese que todo lo dicho de las operaciones de obtención de la longitud (*len*), concatenación (+), repetición (*) y segmentación ([:] y [::]) es aplicable igualmente a las strings, porque, en realidad, son **operaciones de tratamiento de secuencias, y tanto las strings como las tuplas son secuencias**, en un caso, secuencias de caracteres y, en el otro, secuencias de elementos de cualesquiera tipos.

Question 4

Complete

Mark 1.00 out of 1.00

Dada la siguiente tupla:

```
tupla1 = (10, 21, 13, 52, 26, 17, 81, 29)
```

Empareje cada expresión con su resultado.

`tupla5 = tupla1[::3]`	(10, 52, 81)
`tupla2 = tupla1[2:5]`	(13, 52, 26)
tupla3 = tupla1[5:]	(17, 81, 29)
`tupla4 = tupla1[:3]`	(10, 21, 13)

Information

Anotación de variables tupla

Para anotar que una variable o un parámetro es de un tipo no básico, hay que importarlo desde el módulo **typing**:

```
from typing import Tuple
```

```
tupla: Tuple         # La variable tupla se anota como una tupla (tipo Tuple)
vector: Tuple[float] # La variable vector se anota como una tupla de elementos de tipo float
```

A partir de la versión 3.9 de Python, se puede anotar usando el nombre real del tipo (en minúsculas):

```
tupla: tuple         # La variable tupla se anota como una tupla (tipo tuple)
vector: tuple[float] # La variable vector se anota como una tupla de elementos de tipo float
```