TUPLAS

Algoritmos y estructuras de datos I

¿QUÉ SON LAS TUPLAS?

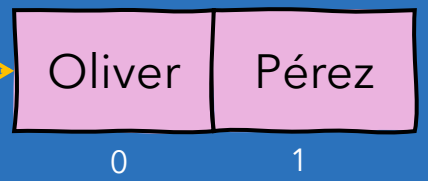
Son una secuencia de valores agrupados. Una tupla sirve para agrupar, como si fueran un único valor, varios valores que, por su naturaleza, deben ir juntos.

Las tuplas son inmutables, es decir, una tupla no puede ser modificada una vez que ha sido creada.

CREACIÓN

Para poder crear las mismas, se deben asignar a la variable los valores separados por comas y entre paréntesis.

```
coordenada = (-34.72904, -58.26374)
fecha = (2017, 8, 7)
hora = (9, 35, 0)
persona = ("Oliver", "Pérez")
```



Lic. Yaps David P.

¿QUÉ SON LAS TUPLAS?

Son una secuencia de valores agrupados. Una tupla sirve para agrupar, como si fueran un único valor, varios valores que, por su naturaleza, deben ir juntos.

Las tuplas son inmutables, es decir, una tupla no puede ser modificada una vez que ha sido creada.

CREACIÓN

También es posible crear una tupla sin el uso de los paréntesis, es decir, solo separando los valores por coma:

```
persona = 'Oliver', 'Pérez'
```

Sin embargo, para aportar mayor claridad al código, es preferible hacer uso de estos.

Una tupla puede incluir un único elemento, pero para que Python entienda que nos estamos refiriendo a una tupla es necesario escribir al menos una coma:

```
meses = (11, )
```

Cuando se crea una tupla de un solo valor, es imprescindible que se agregue la coma al final, ya que, en caso contrario, será tomada la variable como entero.

ACCIONES SOBRE TUPLAS

Para conocer la longitud de una tupla se hace uso de la función *len()*. Esta función devuelve el número de elementos de una tupla:

```
vocales = ('a', 'e', 'i', 'o', 'u')
print(len(vocales)) #5
```

Para saber si un elemento está contenido en una tupla, se utiliza el operador de pertenencia *in* (o su negación con *not* de prefijo):

```
vocales = ('a', 'e', 'i', 'o', 'u')

if ('b' in vocales):
    print("Es una vocal")

if ('b' not in vocales):
    print("No es una vocal")
```

El método *count()* cuenta el número de veces que el objeto pasado como parámetro se ha encontrado en la tupla:

```
numeros = (1, 1, 1, 3, 5)
print(numeros.count(1)) #3
```

El método *index()* busca el elemento que se le pasa como parámetro y devuelve el índice en el que se ha encontrado. En el caso de no encontrarse, se devuelve el error *ValueError*:

```
numeros = (1, 1, 1, 3, 5)
print(numeros.index(5)) #4
print(numeros.index(7)) #ValueError
```

```
notas = (10, 7, 10, 2)

for nota in notas:
    print(nota)

for i in range(0, len(notas)):
    print(notas[i])
```

ITERACIÓN SOBRE TUPLAS

Al igual que las listas, las tuplas son iterables, lo que significa que pueden recorrerse utilizando bucles o funciones que aceptan iterables como argumentos. Esta capacidad de iteración permite realizar operaciones como la búsqueda de elementos, la realización de operaciones en cada elemento individualmente, o la combinación de elementos de manera eficiente.

Además, dado que las tuplas son inmutables, su estructura y contenido permanecen intactos durante la iteración, lo que garantiza la consistencia de los datos.

CONVERSIÓN DE TUPLA A LISTA

Se puede convertir una tupla en una lista usando la función *list*, y una lista en una tupla usando la función *tuple*, permitiendo una flexibilidad adicional en el manejo de colecciones de elementos.

Cuando se convierte una tupla en una lista, cada elemento de la tupla se conserva en el mismo orden en la lista resultante. Esto abre la posibilidad de modificar los elementos de la lista, dado que las listas son mutables, algo que no es posible con las tuplas debido a su inmutabilidad.

Por otro lado, al convertir una lista en una tupla, se crea una nueva tupla que contiene los mismos elementos que la lista original. Este proceso garantiza que la estructura de la tupla resultante sea inmutable, preservando así la integridad de los datos.

```
tupla = (1, 2, 3)
tuplaALista = list(tupla)
print(tuplaALista) #[1, 2, 3]

lista = [5, 8, 13]
listaATupla = tuple(lista)
print(listaATupla) #(5, 8, 13)
```

Estas conversiones son especialmente útiles en situaciones donde se necesita realizar operaciones específicas que solo son posibles en un tipo de datos u otro.

Además, proporcionan una forma conveniente de interoperar entre diferentes partes de un programa que pueden requerir diferentes tipos de colecciones de datos.

DESEMPAQUETADO

```
persona = ('Oliver', 'Pérez')
nombre, apellido = persona

print("Nombre: ", nombre, " - Apellido: ", apellido)
```

La cantidad de variables que se va a asignar el contenido de la tupla, debe ser coincidente; en caso contrario, ocurrirá un error de asignación.

Los valores individuales de una tupla pueden ser recuperados asignando la tupla a las variables respectivas. Esto se llama desempaquetar (*unpack*) la tupla.

El desempaquetado de tuplas es una técnica poderosa que permite acceder a los elementos de una tupla de forma individual y asignarlos a variables en una sola operación.

Es particularmente útil cuando se trabaja con funciones que devuelven múltiples valores como una tupla. En lugar de acceder a los valores de la tupla mediante índices, el desempaquetado permite asignarlos directamente a variables con nombres significativos, lo que mejora la legibilidad y la claridad del código.

COMPARACIÓN DE TUPLAS

Dos tuplas son iguales cuando tienen el mismo tamaño y cada uno de sus elementos correspondientes tiene el mismo valor.

Esta comparación se realiza elemento por elemento, lo que significa que, si las tuplas tienen el mismo número de elementos y cada uno de ellos es igual en posición y valor, entonces se consideran iguales en términos de contenido.

```
print((3, 5) == (33 / 11, 2 + 3)) #True
print((1, 3) == (3, 1)) #False
print((3, 3) == (3, 3)) #True
print((3, "3") == (3, 3)) #False
print((1, 2) == (0, 1, 2)) #False
```

COMPARACIÓN DE TUPLAS

Para determinar si una tupla es menor que otra, se utiliza lo que se denomina orden lexicográfico. Este método de comparación se basa en una especie de diccionario de ordenamiento, donde se considera el orden relativo de los elementos basado en sus valores y posiciones en las tuplas.

Cuando se compara el orden de dos tuplas, el proceso se realiza elemento por elemento, comenzando desde la primera posición. Si los elementos en la primera posición de ambas tuplas son distintos, ellos determinan el ordenamiento de las tuplas: el elemento que sería el primero en orden alfabético o numérico definirá la relación de orden entre las tuplas.

```
#True, ya que (1, 2, 3) es menor que (2, 3, 5, 8) al comparar el primer elemento (1 < 2).
print((1, 2, 3) < (2, 3, 5, 8))

#True, ya que (1, 2, 3) es menor que (1, 2, 4) al comparar el tercer elemento (3 < 4).
print((1, 2, 3) < (1, 2, 4))

#False, ya que (8, 13) es mayor que (0, 21) al comparar el primer elemento (8 > 0).
print((8, 13) < (0, 21))

#True, ya que (1, 2) es menor que (1, 2, 4) al tener la misma secuencia de elementos en la primera y segunda posición, y no tener un tercer elemento para comparar.
print((1, 2) < (1, 2, 4))
```

EJEMPLO DE USO: COORDENADAS

Para representar puntos en el plano, se puede usar tuplas de dos elementos (x, y). Por ejemplo, podemos crear una función distancia que recibe dos puntos y entrega la distancia entre ellos.

En el ejemplo, se crean dos variables del tipo tupla, que son utilizadas como argumentos en el llamado a la función. Dentro de la misma, se desempaquetan para realizar el cálculo de la correspondiente distancia, utilizando el teorema de Pitágoras.

```
def calcularDistancia(punto1, punto2):
    #Extraemos las coordenadas x y y de cada punto
    x1, y1 = punto1
    x2, y2 = punto2

    #Calculamos la diferencia entre las coordenadas
    diferenciaX = x2 - x1
    diferenciaY = y2 - y1

    #Calculamos la distancia utilizando el teorema de Pitágoras
    distancia = ((diferenciaX ** 2) + (diferenciaY ** 2)) ** 0.5

    return distancia


#Ejemplo de uso de la función calcularDistancia
puntoA = (-34.7166, -58.2666)
puntoB = (-34.6167, -58.3817)
print("La distancia entre el punto A y el punto B es:",
calcularDistancia(puntoA, puntoB))
```

EJEMPLO DE USO: FECHAS

Las fechas, en el contexto de la programación y la gestión de datos, se representan típicamente como tuplas que agrupan el año, el mes y el día.

Al representar las fechas de esta manera, se facilita la realización de operaciones relacionales, como la comparación de fechas para determinar su secuencia temporal.

Por ejemplo, al comparar dos fechas representadas como tuplas, se pueden utilizar operadores de comparación como "<" (menor que), ">" (mayor que), "<=" (menor o igual que) y ">=" (mayor o igual que) para determinar si una fecha precede o sigue a otra en el tiempo.

```
#Definición de fechas de nacimiento
fechaNacimientoOliver = (2017, 8, 7)
fechaNacimientoGabriela = (1984, 2, 13)

#Comparación de fechas
if fechaNacimientoGabriela < fechaNacimientoOliver:
    print("Gabriela es mayor que Oliver.")
elif fechaNacimientoGabriela > fechaNacimientoOliver:
    print("Oliver es mayor que Gabriela.")
else:
    print("Ambos tienen la misma edad.")
```

EJEMPLO DE USO TABLAS

Una tabla de datos generalmente se representa como una lista de tuplas en Python.

En el contexto de la programación, especialmente en el análisis de datos y la manipulación de bases de datos, una tabla de datos se refiere a una estructura tabular que organiza la información en filas y columnas.

En Python, una forma común de representar una tabla de datos es mediante una lista de tuplas, donde cada tupla representa una fila de la tabla y los elementos de la tupla representan los valores de cada columna. Cada elemento en una tupla corresponde a un campo o atributo específico de la fila, como el nombre, el DNI, la nacionalidad, entre otros.

```
personas = [
                ('Oliver Pérez', '59111333', 'Argentina'),
                ('Gabriela García', '60333111', 'Argentina')
               ]

for nombre, DNI, nacionalidad in personas:
    print(nombre, ' - DNI: ', DNI, " (", nacionalidad, ")")
```

VENTAJAS SOBRE EL USO DE LISTAS

Las tuplas emergen como una alternativa valiosa a las listas, destacándose por su inmutabilidad, lo que garantiza la integridad de los datos y su eficiencia en términos de uso de memoria y rendimiento.
Al ofrecer seguridad de datos y capacidad de asignación múltiple, las tuplas se convierten en una elección poderosa en situaciones donde la estabilidad y la velocidad son prioritarias, destacándose como una herramienta esencial para optimizar el desarrollo de aplicaciones en Python.

Seguridad de datos

Las tuplas son inmutables, lo que significa que una vez creadas, no se pueden modificar. Esto puede ser beneficioso en situaciones donde deseas asegurarte de que los datos no cambien accidentalmente. Por ejemplo, puedes usar tuplas para representar valores constantes o para garantizar que los datos pasados a una función no se alteren dentro de ella.

Eficiencia

Las tuplas son más eficientes que las listas en términos de uso de memoria y rendimiento, especialmente para operaciones simples como acceso y recuperación de elementos. Esto se debe a que las tuplas son estructuras de datos más simples que no admiten operaciones de modificación, lo que las hace más ligeras y rápidas en algunas situaciones.

VENTAJAS SOBRE EL USO DE LISTAS

Las tuplas emergen como una alternativa valiosa a las listas, destacándose por su inmutabilidad, lo que garantiza la integridad de los datos y su eficiencia en términos de uso de memoria y rendimiento.
Al ofrecer seguridad de datos y capacidad de asignación múltiple, las tuplas se convierten en una elección poderosa en situaciones donde la estabilidad y la velocidad son prioritarias, destacándose como una herramienta esencial para optimizar el desarrollo de aplicaciones en Python.

Hashability

Las tuplas son *hashables*, lo que significa que se pueden utilizar como claves en diccionarios y como elementos de conjuntos. Esto puede ser útil en situaciones donde necesitamos estructurar datos de una manera que puedas indexar o buscar eficientemente.

Uso en asignación múltiple

Las tuplas son comúnmente utilizadas en para asignaciones múltiples, lo que significa que puedes asignar múltiples valores a múltiples variables en una sola declaración siendo una forma común de asignar valores utilizando una tupla:

```
mes, dia = 8, 7
```

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.