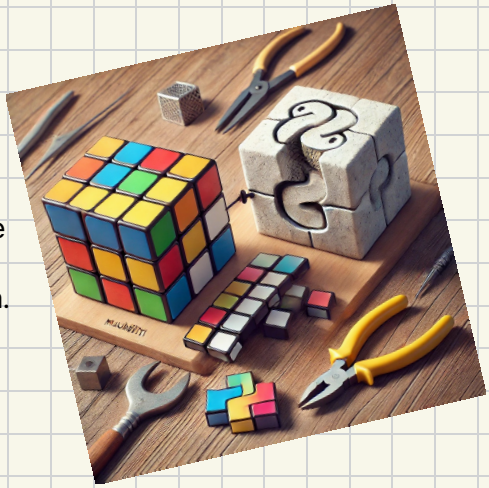
Una Ambiciosa Introducción a Python
Parte II

Vamos a definir a las secuencias en Python como un tipo capaz de almacenar más de un valor, los cuales pueden ser examinados secuencialmente como por ejemplo, las listas con las cuales terminamos de trabajar.

Mutabilidad de los objetos:
Es la propiedad que describe la capacidad de poder cambiar durante la ejecución.



Inmutabilidad:
estos datos,no pueden ser modificados una vez creados.

```
mi_lista=[4,56,7,1,-8]
>>> id(mi_lista)
>>> 364758
mi_lista.append(23)
>>>id(mi_lista)
>>>364758
mi_lista[0]=21
>>>id(mi_lista)
>>>364758
```

```
nota="soy inmutable"
>>>id(nota)
>>>73648
nota+="no puedo cambiar"
>>>id(nota)
73920
>>>nota[1]='*'
>>> error!!!
```

Una Ambiciosa Introducción a Python
Parte II

Tuplas

La palabra tupla proviene de las matemáticas, las cuales son usadas para describir una secuencia finita, ordenada de valores.
Está estructura de datos posee las mismas características que las listas:

- Almacena múltiples objetos
- Es ordenada
- Es indexada
- Se pueden anidar
- Son iterables
- **Son Inmutables**

Pero hay una que es diferente y tiene que ver con la inmutabilidad de los objetos, esto quiere decir que una vez que son creados no se puede cambiar su valor, como : los int, float, str y las tuplas.

Creación

días=('Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo')

También podemos crear una tupla usando tuple()
curso=tuple('AIPython II')

- tupla vacía, es aquella que no posee ningún valor y puede ser creada usando un par de ()
- tupla con un solo elemento, x=(1,)

¿Una tupla es un string?

Para esta respuesta, vamos a decir que la principal diferencia entre ambas secuencias es que los elementos de las tuplas pueden ser de cualquier tipo mientas que en los string solo pueden almacenar caracteres.

Algo interesante sobre las tuplas es que existe una tercera forma de creación y esta relacionado con la capacidad que tienen este tipo de secuencia y que se denomina :

Packing: Es la capacidad de poder empaquetar valores en una tupla

```
coordenadas=5.21, 9.37
```

Unpacking: Es la capacidad de poder desempaquetar una tupla en diferentes variables

```
x,y=coordenadas
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Con esto en mente, veamos como las tuplas nos permiten devolver múltiple valores en una función

```
def suma_resta(num1, num2):
    return (num1+num2, num1-num2)

resultado=suma(5,3)

>>> resultado
>>> (8,2)
```

Este es un uso común de las tuplas, esta función definida como **suma_resta** tiene dos parámetros, num1 y num2. Como se observa la función retorna una tupla. El primer elemento corresponde con la suma y el segundo con la diferencia entre ellos. Luego cuando mostramos, resultado, en la terminal vemos que es una tupla.

Sets

Otra tipo de secuencia en Python son los denominados conjuntos, sets. Un conjunto es una secuencia no ordenada que no puede contener elementos duplicados. Como las tuplas, estos pueden ser inicializados a traves de un lista.
Otra caracteristica de los conjuntos es que no tiene encuenta el orden.

Su sintaxis usa llaves para denotarlos {}. Cuando un conjuto es creado desde una secuencia que contiene elementos repetidos estos son automáticamente eliminados.

Los conjuntos son utiles porque proveen las operaciones que conocemos sobre estos, como la union (|), interseccion (&), diferencia (-) y la disyuncion exclusiva (^).

```
digitos=[0,1,1,2,3,4,5,6,7,8,9,9,9]
conjunto_digitos=set(digitos)
>>> conjunto_digitios
{0,1,2,3,4,5,6,7,8,9}
```

```
impares={1,3,5,7,9}
pares=conjunto_digitos-impares
>>>pares
{0,8,2,4,6}
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

```
primos=set([2,3,5,7])
primos_pares=prime&pares
>>>primos_pares
{2}
```

```
numeros=impares I  pares
>>> numeros
{0,1,2,3,4,5,6,7,8,9}
```

```
primeros= set([0,1,2,3,4])
segundos=set([2,3,4,5,6])
unicos= primeros ^ segundos
>>> unicos
{0,1,5,6}
```

Verificar si un elemento pertence al conjunto

conjunto={"a","b","c","d"}
a in conjunto
True

Agregar elementos podemos usar

- add() #un elemento
- update() #multiples desde una lista, tupla o set

Remover elementos

- remove() #un elemento, puede fallar
- discard()#un elemento, evita fallar sino se encuentra
- pop()#devuelve un elemento de manera arbitraria

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Un diccionario es una estructura de datos fundamental en Python que nos permite almacenar datos en pares de clave-valor. A menudo, en otros lenguajes de programación, se les conoce como tablas hash o maps. Cada clave en un diccionario debe ser única e inmutable, mientras que los valores asociados pueden ser de cualquier tipo de dato, incluidos otros diccionarios.

Algunas características clave de los diccionarios incluyen:

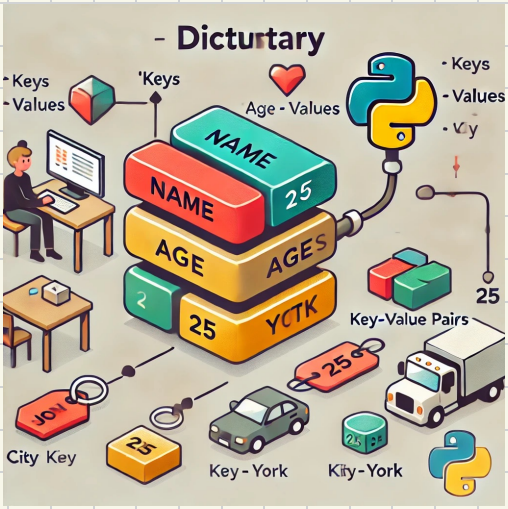
Mutabilidad: Los diccionarios pueden ser modificados después de su creación; es posible añadir, eliminar o cambiar pares clave-valor.

Acceso rápido: Las claves permiten un acceso rápido a los valores, similar a cómo un índice funciona en listas.

Unidireccionalidad: La relación clave-valor es unidireccional; es decir, puedes acceder a un valor usando su clave, pero no al revés.

Sintaxis: Los diccionarios se representan con llaves {}, y cada par clave-valor se escribe en la forma clave: valor, separados por comas.

En resumen, los diccionarios en Python se asemejan a los diccionarios de un idioma, donde las “palabras” (claves) tienen una “definición” (valor) asociada.



Ejemplo de un diccionario en Python:

```
mi_diccionario = {
    'nombre': 'Alice',
    'edad': 30,
    'ciudad': 'Madrid'
}
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Métodos y funciones de los diccionarios

```
santa_cruz={
    'CORPEN AIKE':11093,
    'DESEADO':107630,
    'GUER AIKE':113267,
    'LAGO ARGENTINO':18864,
    'LAGO BUENOS AIRES':8750,
    'MAGALLANES':9202,
    'RIO CHICO':5158
}
```

Keys : retorna una lista de todas las claves

```
for departamento in santa_cruz.keys():
    print(departamento)
```

Values : retorna una lista de todos los valores

```
for habitantes in santa_cruz.values():
    print(habitantes)
```

items : retorna una lista de tuplas, donde cada una es un par, clave y su valor

```
for departamento, habitantes in santa_cruz.items():
    print(f'{departamento} -> {habitantes}')
```

get : retorna el valor asociado a una clave, tiene un parámetro opcional si la clave no se encuentra

```
print(santa_cruz.get('CABA','No Encontrada'))
```

clear : permite eliminar todos los elementos del diccionario.

pop : Permite eliminar una clave, tiene un parámetro opcional si la clave no se encuentra puede generar un error.

popitem : Elimina y devuelve el ultimo par clave-valor, retorna una tupla.

update : Nos permite agregar, actualizar elementos clave-valor a. l diccionario, podemos pasarle una lista, tupla, etc.

clear : elimina todos los elementos.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Zip

Está función en Python permite combinar varias secuencias (listas, tuplas, cadenas, etc.) en pares o grupos de elementos. Es muy útil cuando necesitamos recorrer múltiples colecciones al mismo tiempo.

Si los iterables tienen diferentes longitudes, solo se emparejara hasta la cantidad de elementos del mas cortó.

Sintaxis

zip(iterable1, iterable2, ….)

retorna un integrador con tuplas formadas por elementos correspondiente de cada iterable

```
nombres = ["Ana", "Juan", "Pedro"]
edades = [25, 30, 22]

combinados = zip(nombres, edades)

for nombre, edad in combinados:
    print(f"{nombre} tiene {edad} años.")
```

Ana tiene 25 años.
Juan tiene 30 años.
Pedro tiene 22 años.

Podemos transformar el resultado en una lista de tuplas:

```
pares = list(zip(nombres, edades))
print(pares)
```

[('Ana', 25), ('Juan', 30), ('Pedro', 22)]

o en un diccionario:

```
diccionario = dict(zip(nombres, edades))
print(diccionario)
```

{'Ana': 25, 'Juan': 30, 'Pedro': 22}

Combinando mas de dos listas

```
ciudades = ["Buenos Aires", "Madrid", "México"]

for nombre, edad, ciudad in zip(nombres, edades, ciudades):
    print(f"{nombre}, {edad} años, vive en {ciudad}.")
```

Ana, 25 años, vive en Buenos Aires.
Juan, 30 años, vive en Madrid.
Pedro, 22 años, vive en México.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Veamos cuando los iterables tienen diferentes longitudes, zip() solo empareja hasta la cantidad de elementos del mas corto.

```
a = [1, 2, 3, 4]
b = ["A", "B"]

print(list(zip(a, b)))
```

[(1, 'A'), (2, 'B')]

Para poder dar una solución para emparejar todo, podemos usar zip_longest() del módulo itertools.

```
from itertools import zip_longest

print(list(zip_longest(a, b, fillvalue="Sin dato")))
```

[(1, 'A'), (2, 'B'), (3, 'Sin dato'), (4, 'Sin dato')]

También podemos desempaquetar con zip(*), revertimos la combinación

```
pares = [('Ana', 25), ('Juan', 30), ('Pedro', 22)]
nombres, edades = zip(*pares)

print(nombres)  # ('Ana', 'Juan', 'Pedro')
print(edades)   # (25, 30, 22)
```

Lic. Franco Herrera