Nombre del bloque PROGRAMACIÓN II

Año de cursada 1° AÑO

Clase Nº 1: TIPO DE DATOS COMPLEJOS EN PYTHON

Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Secuencias.
- Listas.
- Tuplas.
- Diccionarios

Presentación:

Bienvenidos a la primera clase de este trayecto formativo denominado Programación II 🖐. Estoy muy entusiasmado en que nos podamos acompañar durante el mismo, en el cual vamos a estar explorando en lo que en mi opinión es un mundo apasionante como es el de la programación 🙌.

En los últimos años hemos sido testigos de un crecimiento exponencial del uso de la tecnología para realizar nuestras tareas de la vida cotidiana, desde alquilar una casa para vacaciones, pagar servicios, solicitar turnos médicos, estudiar, comprar pasajes, ver nuestras series favoritas, pagar nuestros impuestos, y varias actividades más que las realizamos a través de una herramienta tecnológica. Estas herramientas por un lado tienen programadores detrás que escriben código para que puedan transformarse en realidad. Por otro lado, con el uso de las mismas por todos nosotros se van generando una enorme cantidad de datos como nunca antes en la historia. El análisis y estudio de estos datos nos ayuda a evaluar y

predecir comportamientos de los individuos o grupo de individuos lo que termina generando una mejora en la calidad de las mismas y ajustar las preferencias y necesidades desarrollando ciclos de mejora continua, logrando mejor rentabilidad en las empresas, mejores servicios en el sector público y una mejora en la calidad de vida de la sociedad en su conjunto.

Durante el recorrido de este trayecto formativo vamos a trabajar con Python, un lenguaje de programación multipropósito, muy versátil y que nos permite utilizarlo para trabajar con ciencia de datos 💪.

En las primeras clases estaremos repasando conceptos que ya han trabajado en Programación I, lo veremos con mayor profundidad y trabajaremos mucho en la práctica 😃. En la clase de hoy vamos a trabajar con los tipos de datos complejos del lenguaje. Ya vieron que a través de las variables podemos almacenar datos, pero ¿qué pasa si queremos almacenar varios datos?,

¿deberíamos declarar tantas variables como datos quisiéramos almacenar?. Vamos a ver que a través de los tipos de datos complejos Python nos ofrece una solución a este problema💪.

Les deseo a todos y todas que tengan una excelente cursada, estoy a disposición para facilitarles lo que necesitan y sin más preámbulo los invito a recorrer la primera clase. Exitos!!

Desarrollo y Actividades:

Secuencias

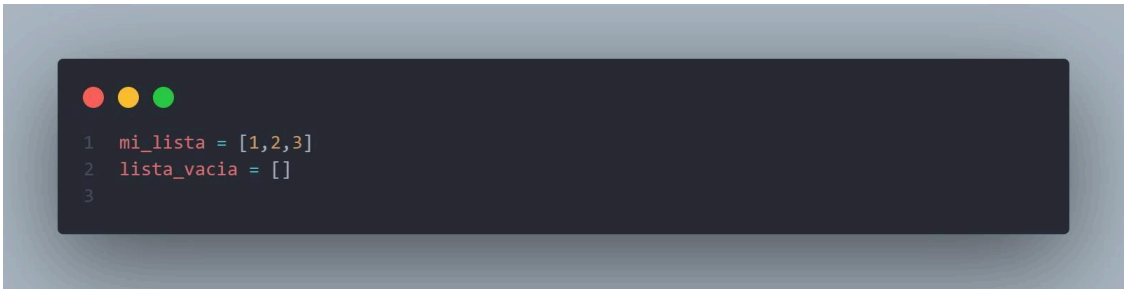
De forma abstracta y simple, se podría decir que los tipos secuencia son como un mueble que tiene N cajones, y todos están dispuestos uno detrás de otro. Cada cajón puede almacenar un tipo distinto de elemento y, según la naturaleza del mueble, que posee unas características propias, se podría expandir dinámicamente o tendría una longitud fija o se podría iterar de una determinada manera.

Listas

Las listas son secuencias de elementos de cualquier tipo y sin límite de longitud. El constructor y el tipo de dato es list en Python. Una secuencia de elementos separados por comas y rodeados por corchetes

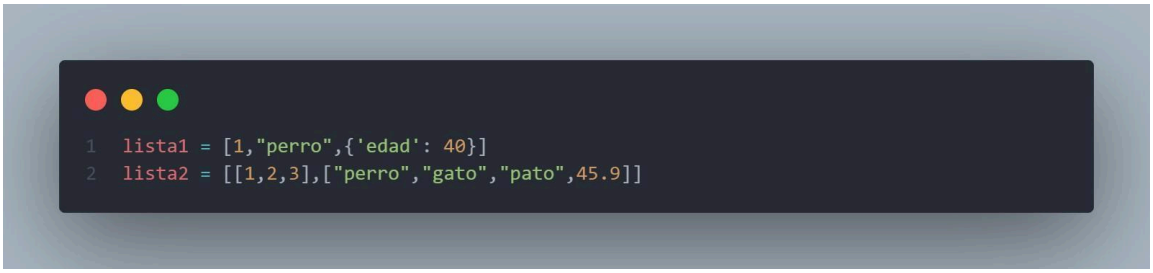
La lista de Python es una de las estructuras de datos más utilizadas, junto con los diccionarios.

Cómo crear una lista



Las listas contienen objetos normales de Python, separados por comas y rodeados por corchetes. Los elementos de una lista pueden tener cualquier tipo de datos y se pueden mezclar. Incluso puede crear una lista de listas.

Las siguientes listas son válidas:



```
lista1 = [1,"perro",{'edad': 40}]
lista2 = [[1,2,3],["perro","gato","pato",45.9]]
```

Uso de la función list()

Las listas de Python, como todos los tipos de datos en Python, son objetos.

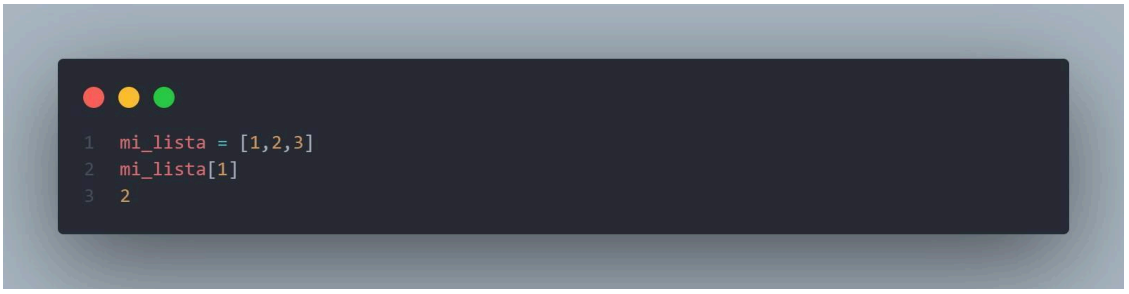
La clase de una lista se llama list, con una L minúscula.

Si queremos convertir otro objeto de Python en una lista, podemos utilizar la función list(), que en realidad es el constructor de la propia clase list

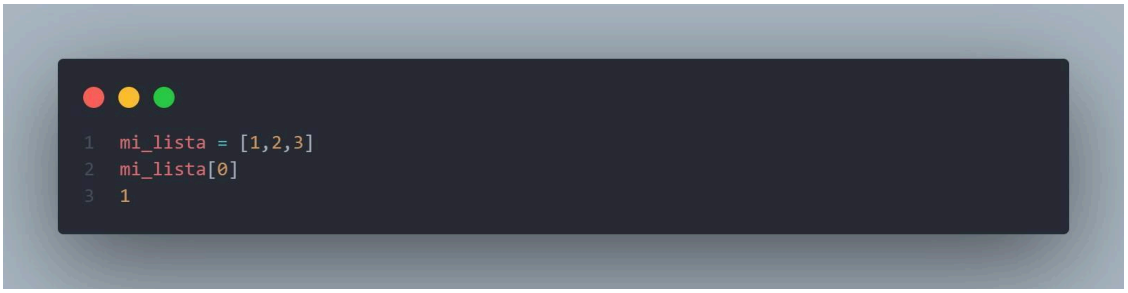
```
a = 'gonzalo'
b = list(a)
print(b)
['g', 'o', 'n', 'z', 'a', 'l', 'o']
```

Acceso a los elementos de la lista

Para acceder a un elemento de la lista, debemos conocer su posición. El primer elemento de una lista, **siempre** se encuentra en la posición 0, el segundo está en la posición 1 y así sucesivamente.



```
mi_lista = [1,2,3]
mi_lista[1]
2
```



```
mi_lista = [1,2,3]
mi_lista[0]
1
```

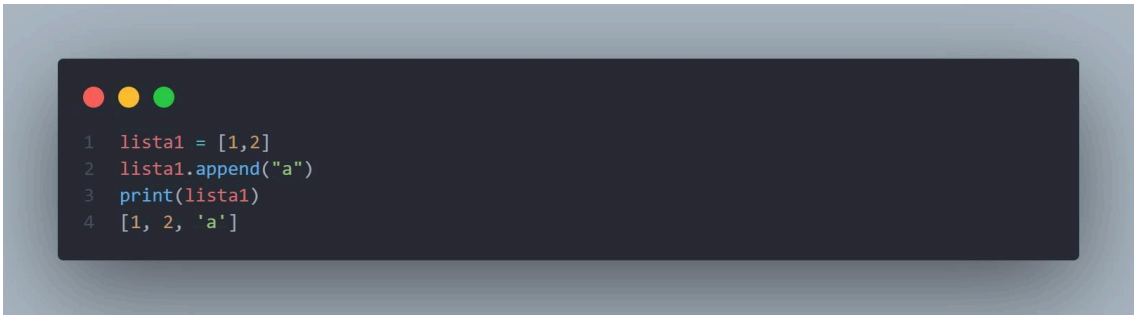
```
mi_lista = [1,2,3]
mi_lista[4]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\colecciones.ipynb Celda 1 in 2
      1 mi_lista = [1,2,3]
----> 2 mi_lista[4]

IndexError: list index out of range
```

Agregar elementos a una lista

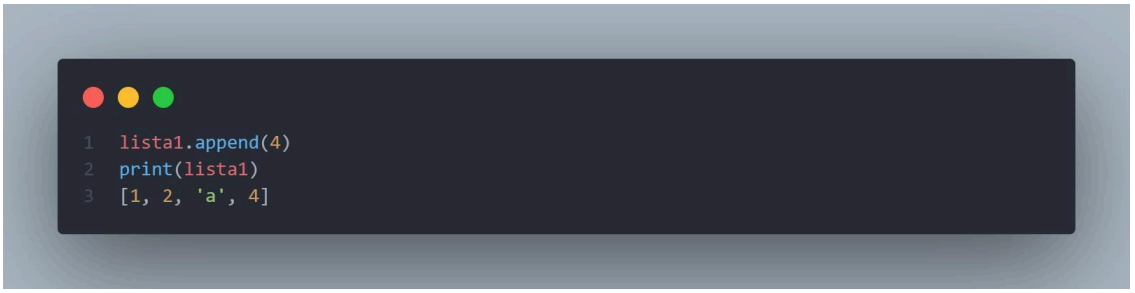
Las listas de Python tienen tres métodos para agregar elementos:

Append

Los objetos list tienen varios métodos integrados útiles, uno de ellos es el método *append.* Al llamar a *append* en una lista, agregamos un elemento al final de la lista.



```
lista1 = [1,2]
lista1.append("a")
print(lista1)
[1, 2, 'a']
```



```
lista1.append(4)
print(lista1)
[1, 2, 'a', 4]
```

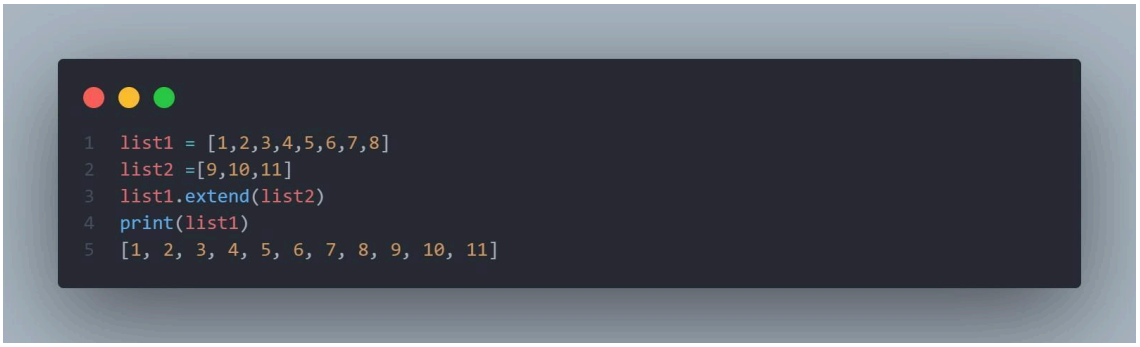
Insert

A diferencia de append, el método insert permite poder agregar elementos a la lista en una posición definida de la lista a través de su indice.

```
list1 = [1,2,3,4,5,6,7,8]
list1.insert(4,20)
print(list1)
[1, 2, 3, 4, 20, 5, 6, 7, 8]
```

Extend

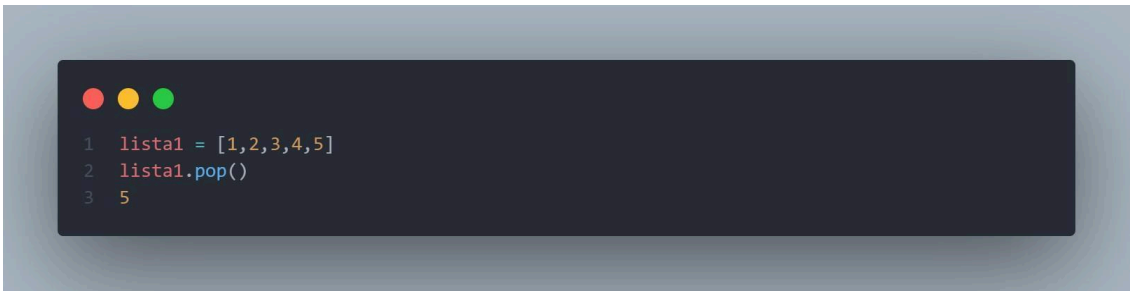
El método extend nos permite agregar una lista de elementos a un lista previamente creada, puede ser de cualquier otro iterable como ser otra lista, una tupla, un rango, etc.



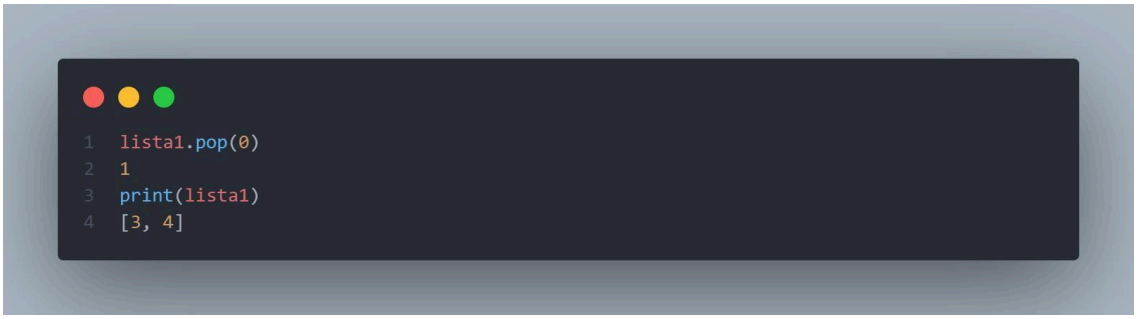
```
list1 = [1,2,3,4,5,6,7,8]
list2 =[9,10,11]
list1.extend(list2)
print(list1)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Quitar elementos de una lista

El método pop() quita y devuelve el último elemento de forma predeterminada, a menos que le demos un argumento de índice.



```
lista1 = [1,2,3,4,5]
lista1.pop()
5
```



```
lista1.pop(0)
1
print(lista1)
[3, 4]
```

Uso de del() para eliminar elementos

Tenemos varias formas de eliminar o quitar elementos de una lista. Mientras que pop devuelve el elemento que se elimina de la lista, del lo elimina sin devolvernos nada. No solo podemos eliminar un elemento con del, también podríamos eliminar toda la lista.

```
del(list1)
print(list1)
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\colecciones.ipynb Celda 7 in 2
      1 del(list1)
----> 2 print(list1)

NameError: name 'list1' is not defined
```

Eliminar o borrar todos los elementos de la lista

Para quitar todos los elementos de una lista, utilizamos el método clear()

```
list1 = [1,2,3]
list1.clear()
print(list1)
[]
```

Reemplazar elementos de una lista

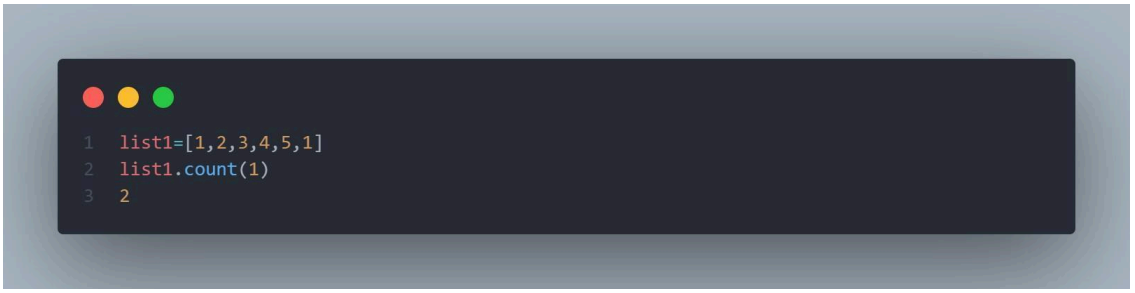
Si queremos reemplazar elementos de una lista, lo podemos hacer asignando un nuevo valor al índice determinado:***Cómo obtener la longitud de la lista***

Para poder obtener el largo de una lista, usamos la función len, la misma puede ser utilizada para obtener la longitud de cualquier objeto en Python

```
list1=[1,2,3,4,5]
len(list1)
5
```

Contar ocurrencias de un valor

La función de recuento cout, nos permite contar las ocurrencias de un valor particular dentro de una lista



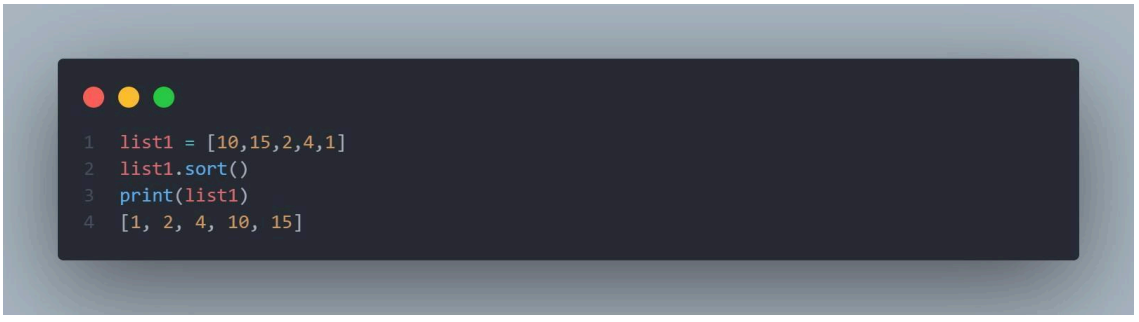
Buscar el índice de un elemento de una lista

Podemos encontrar donde está un elemento dentro de una lista con el método index. Por ejemplo en el siguiente código el 5 se encuentra en la posición 4 (recordemos que empezamos a contar en cero)

Ordenar en orden ascendente la lista

Si queremos ordenar una lista, podemos utilizar el método sort. Este método solo funciona en el caso en que tengamos listas de un mismo. Podemos ordenar tanto de orden **ascendente** como **descendente**

Ascendente



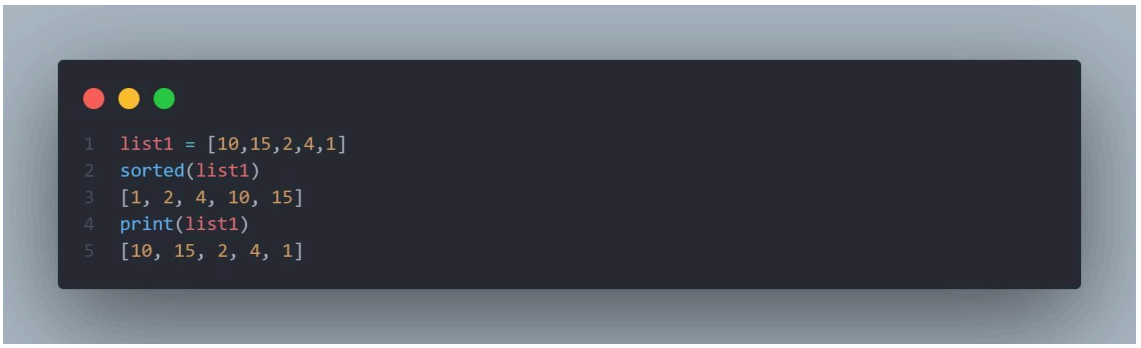
Descendente

Usando Sorted

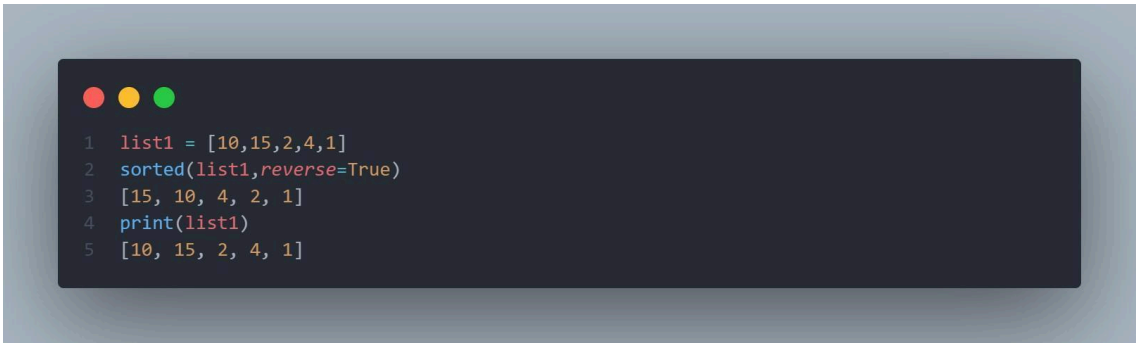
La diferencia entre el método sort y sorted, es que mientras sort modifica la lista original, sorted crea una nueva lista, sin alterar la lista original. De esta

manera yo podría usar una variable para almacenar el resultado de sorted, y de esta manera tendría dos listas, la original y la ordenada.

Ascendente



Descendente



Listas no ordenables

Como mencionamos al ver el método sort, al permitir almacenar las listas elementos de diferentes tipos, no es posible ordenar una lista, si los elementos que almacena pertenecen a tipos diferentes, ya que no los podría comparar y de intentarlo, Python lanza un TypeError

```
list1 = [10,'a',2,'c',1]
list1.sort()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\colecciones.ipynb Celda 9 in 2
      1 list1 = [10,'a',2,'c',1]
----> 2 list1.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

Slicing

Hay veces que necesitamos obtener partes de una lista. Python como ya lo vimos con las cadenas (strings) tiene una sintaxis para poder cortar, rebanar o segmentar una lista.

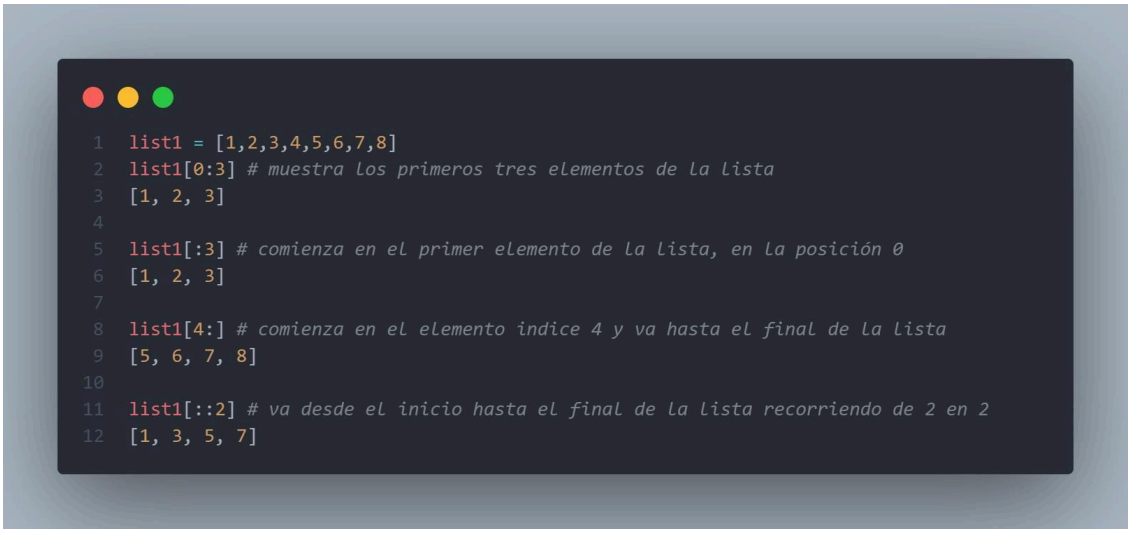
La sintáxis de segmentación es la siguiente:

mi_lista[inicio:fin:paso]

con respecto a esta sintaxis podemos detallar lo siguiente:

- inicio: es la primera posición del elemento que se incluye.
- fin: es exclusivo, lo que significa que el elemento en la posición fin no se incluirá.
- paso: es el tamaño del paso es decir si los elementos van a ir buscando de 1 en 1 desde el inicio o de 2 en 2, etc.
- inicio, fin y paso son todos **opcionales**.

También se pueden usar valores negativos.



```
list1 = [1,2,3,4,5,6,7,8]
list1[0:3] # muestra los primeros tres elementos de la lista
[1, 2, 3]

list1[:3] # comienza en el primer elemento de la lista, en la posición 0
[1, 2, 3]

list1[4:] # comienza en el elemento indice 4 y va hasta el final de la lista
[5, 6, 7, 8]

list1[::2] # va desde el inicio hasta el final de la lista recorriendo de 2 en 2
[1, 3, 5, 7]
```

Actividad N° 1 🤓

1. Crear una lista que contenga nombres de provincias Argentinas, que contenga más de 5 elementos e imprimir por pantalla
2. Imprimir por pantalla el tercer elemento de la lista
3. Imprimir por pantalla del segundo al cuarto elemento
4. Mostrar el tipo de dato de la lista
5. Mostrar los primeros 4 elementos de la lista
6. Agregar una provincia más a la lista que ya exista y otra que no.
7. Agregar una provincia, pero en la cuarta posición.
8. Extender otra lista a la ya creada.
9. Eliminar un elemento de la lista.
10. Extraer el último elemento de la lista, guardarlo en una variable e imprimirlo.

TuplasUna tupla es un tipo de secuencia similar a una lista, pero es **inmutable**, por lo que, una vez inicializada, no se puede cambiar ninguno

de sus elementos sin generar un nuevo objeto.

Una tupla de Python comparte muchas propiedades con las listas:

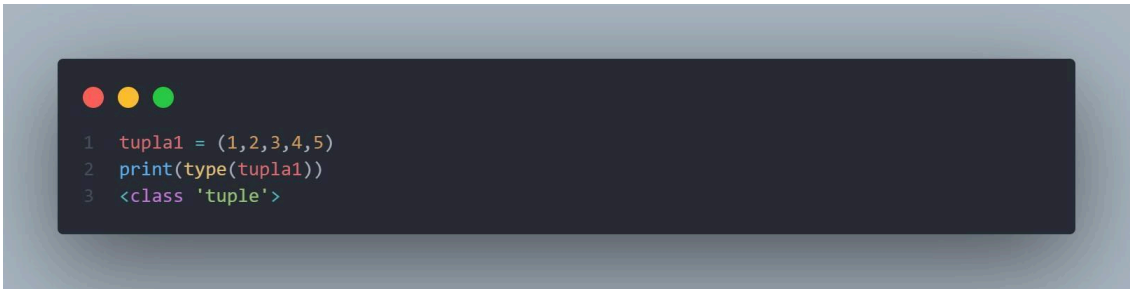
- Puede contener varios valores en una sola variable
- Una tupla puede tener valores duplicados
- Está indexado: puede acceder a los elementos numéricamente
- Una tupla puede tener una longitud

arbitraria Pero hay diferencias significativas:

- Una tupla es inmutable; No se puede cambiar una vez que la haya definido
- Una tupla se define usando paréntesis opcionales () en lugar de corchetes []

Creación de una tupla

Las tuplas se crean muy similar a la creación de una lista, en lugar de usar corchetes [], utilizamos paréntesis ().

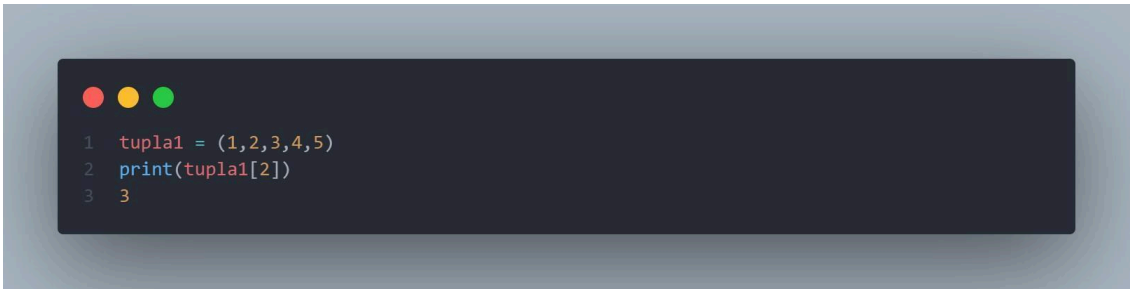


Como vemos en la imagen el constructor de la clase es tuple.

Acceder a elementos de una tupla

Para acceder a los elementos de una tupla, lo hacemos de la misma forma que lo hacemos con las listas, nombre de la tupla y entre corchetes

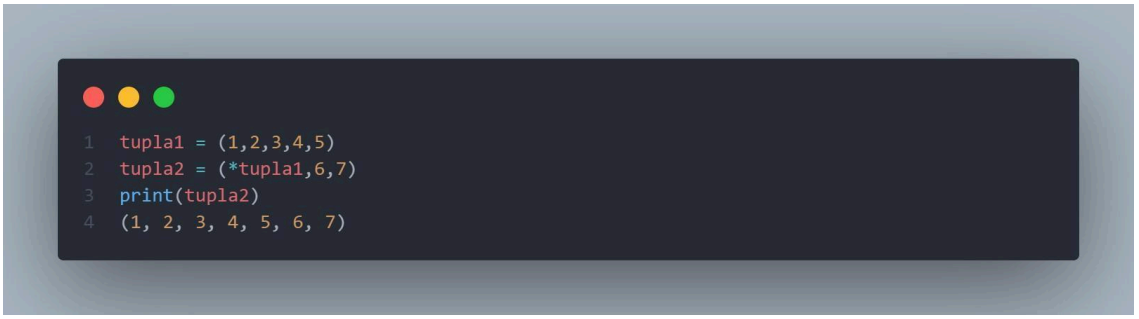
indicamos el índice del elemento.



Agregar elementos a una tupla

Como ya hemos mencionado una tupla es inmutable, no se pueden agregar elementos nuevos a una tupla después de crearla. Por esta misma razón, tampoco se pueden eliminar elementos de una tupla.

Lo que sí está permitido es crear una nueva tupla a partir de una tupla anterior y agregarle los elementos adicionales de esta manera.



El operador inicial * desempaqueta los valores en elementos individuales. Es como si los hubiéramos escrito individualmente en ese lugar.

Convertir una tupla en una lista

Como ya hemos visto, mientras las listas son objetos mutables, las tuplas son inmutables. Si es necesario, podemos convertir una tupla en una lista de

forma de poder agregar y/o
quitar elementos. Para esto, la forma más limpia y legible es utilizando el constructor list()

```
tupla1 = (1,2,3,4,5)
list(tupla1)
[1, 2, 3, 4, 5]
```

Desempaquetar una tupla

Podemos extraer los elementos de una tupla en múltiples variables, este proceso se llama desempaquetar la tupla. Este proceso resulta útil por múltiples motivos:

1. Trabajar con valores devueltos por funciones. Muchas funciones en Python devuelven múltiples valores en forma de tupla. Si los necesitamos algunos de estos valores, podemos desempaquetarlos en variables separadas para trabajar con ellos.
2. Simplificar el código. Desempaquetar una tupla puede hacer que el código sea más limpio y fácil de leer.

```
persona =('Gonzalo',44,'Programación II')
nombre, edad, materia = persona
print(nombre)
print(edad)
print(materia)
Gonzalo
44
Programación II
```

Actividad N° 2 🤓

1. Crear una tupla que
   contenga los números enteros del 1 al 20
2. Imprimir desde el índice 10 al 15 de la tupla
3. Mostrar la cantidad de veces que se encuentra un
   elemento específico dentro de la tupla y de la lista (utilizar
   count)
4. Convertir la tupla en una lista
5. Desempaquetar solo los primeros 3 elementos de la tupla en
   3 variables

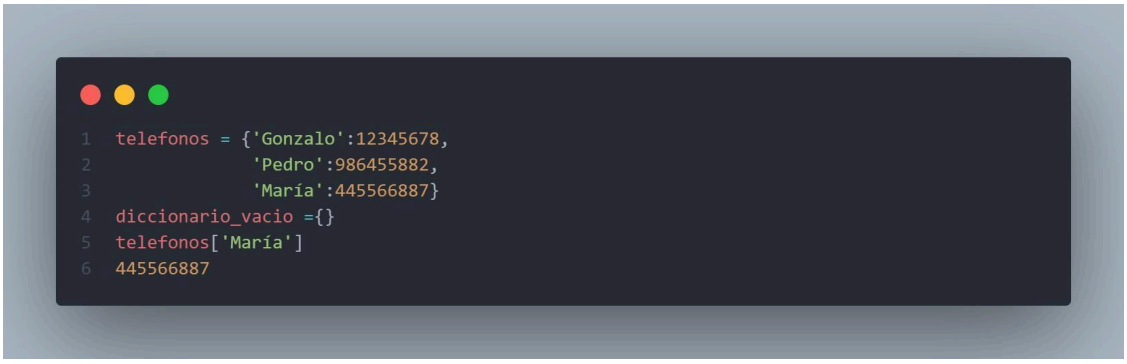
Diccionarios

Los diccionarios son uno de los tipos de datos más poderosos del lenguaje. En otros lenguajes de programación, los diccionarios también se conocen como matrices asociativas. Permiten asociar una o más claves a valores.

Si están familiarizados con JSON, es posible que lo vean similar. La sintaxis de un diccionario es muy parecida a la sintaxis de un documento JSON.

Creación de un diccionario

A continuación vamos a ver cómo podemos crear un diccionario:



Acceder y eliminar un par clave - valor

Ahora que ya hemos visto cómo crear un diccionario, vamos a ver cómo podemos agregar y eliminar entradas a uno ya creado.

```
telefonos['Veronica']=77889977
del(telefonos['Gonzalo'])
telefonos
{'Pedro': 986455882, 'María': 445566887, 'Veronica': 77889977}
```

Sobrescribir entradas de un diccionario

Para sobrescribir una entrada de diccionario, simplemente le asignamos un nuevo valor, no es necesario eliminar primero con del().

```
telefonos
{'Pedro': 986455882, 'María': 445566887, 'Veronica': 77889977}
telefonos['Veronica']=11111111
telefonos
{'Pedro': 986455882, 'María': 445566887, 'Veronica': 11111111}
```

Valores válidos en un diccionario

Los diccionarios no tienen limitación de tipo de datos válidos, podemos poner cualquier tipo de datos en un diccionario, de hecho podemos poner diccionarios y listas dentro de un diccionario y acceder a los valores de una manera muy natural.

```
diccionario ={'sub_diccionario':{'a':30,'b':False},
              'lista1':[10,20,30]}
diccionario
{'sub_diccionario': {'a': 30, 'b': False}, 'lista1': [10, 20, 30]}
diccionario['sub_diccionario']['b']
False
diccionario['lista1'][1]
20
```

El constructor de un diccionario es el método dict(). Como mencionamos anteriormente para ver cual es el constructor de un tipo de dato, lo hacemos con la función type()

Explorar valores de diccionarios

Una vez que tenemos creado el diccionario, las operaciones para explorar los diccionarios son las siguientes:

keys(): devuelve un objeto tipo dict_view del diccionario con las claves que contiene.

```
diccionario.keys()
dict_keys(['sub_diccionario', 'lista1'])
```

values(): devuelve un objeto de tipo dict_values del diccionario con los valores que contiene.

```
diccionario.values()
dict_values([{'a': 30, 'b': False}, [10, 20, 30]])
```

items(): devuelve un objeto de tipo dict_items del diccionario con los pares clave-valor de los elementos presentes en el diccionario.

get(): devuelve el valor asociado a la clave. Si no lo encuentra, devuelve None. Si se añade un valor por defecto y no encuentra la clave pedida, devuelve el valor por defecto.

```
diccionario.get('lista1')
[10, 20, 30]
diccionario.get('Gonzalo')

diccionario.get('precio',30000)
30000
```

diccionario[clave]: devuelve el valor asociado a la clave. Si no lo encuentra, eleva una excepción del tipo KeyError.

```
diccionario['lista1']
[10, 20, 30]
diccionario['precio']
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\diccionarios.ipy
nb Celda 9 in 3
      1 diccionario['lista1']
      2 [10, 20, 30]
----> 3 diccionario['precio']

KeyError: 'precio'
```

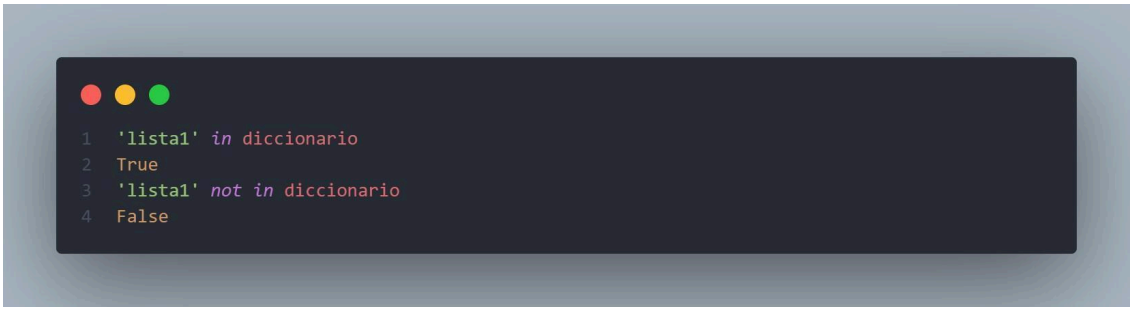
list(): devuelve una lista de todas las claves que contiene un diccionario

len(): devuelve el número de elementos que contiene el diccionario.

```
len(diccionario)
2
```

Comprobar si existe una clave en un diccionario

Podemos comprar si existe una clave dentro de un diccionario con las palabras claves in y not in:



```
'lista1' in diccionario
True
'lista1' not in diccionario
False
```

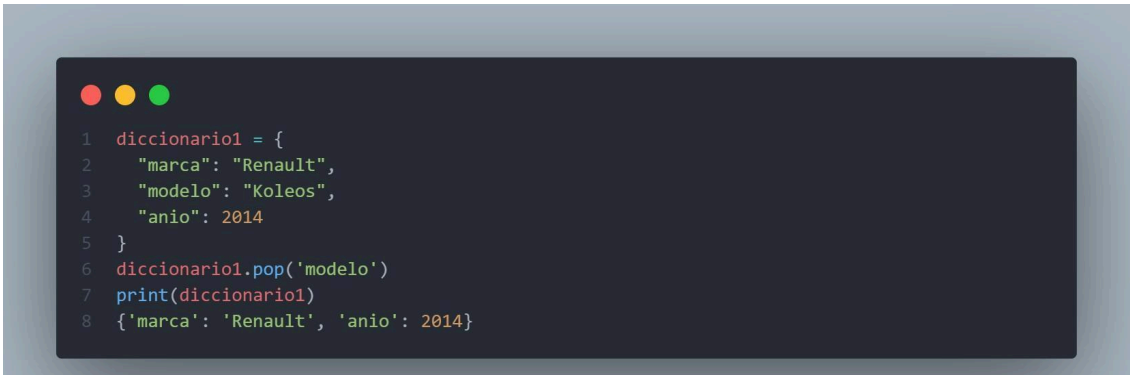
Copiar diccionarios

Si necesitamos realizar la copia de un diccionario, Python nos provee del método copy()

Eliminar elementos del diccionario

Python nos provee varios métodos para eliminar elementos de un diccionario, uno ya lo hemos mencionado que es el método del(). Vamos que otros métodos tiene Python para ofrecernos 😀.

pop(): elimina el elemento con el nombre de clave especificado.



```
diccionario1 = {
  "marca": "Renault",
  "modelo": "Koleos",
  "anio": 2014
}
diccionario1.pop('modelo')
print(diccionario1)
{'marca': 'Renault', 'anio': 2014}
```

popitem(): este método elimina el último elemento insertado.

Actividad N° 3 🤓

1. Crear un diccionario que contenga nombre, edad y carrera de un alumno y mostrar el diccionario completo.
2. Utilizar el método get para mostrar el valor de la clave nombre.
3. Cambiar la edad del alumno y mostrar el diccionario completo.
4. Agrear un par clave valor que contenga el sexo del alumno. Mostrar el diccionario completo.
5. Usar el método pop() para remover la edad del alumno. Mostrar el diccionario completo.
6. Crear un diccionario que contenga las notas de un alumno en tres materias. Las materias son programación II, ciencia de datos, programación I.
7. Mostrar todos los ítems del diccionario de notas.
8. Crear un nuevo diccionario que sea una copia del diccionario de notas del alumno. Mostrar los datos del nuevo diccionario.
9. Mostrar los valores del diccionario de notas. 10.Mostrar la longitud del diccionario de notas.
10.

Instancia de Autoevaluación:

En esta instancia de autoevaluación te pedimos que puedas poner en juego lo que has aprendido hasta este momento. Recordá que siempre podrás volver a la teoría presentada en esta clase o hacer las preguntas pertinentes en los encuentros sincrónicos o las tutorías presenciales.

Es importante tener en cuenta que para aprobar esta instancia deberás:

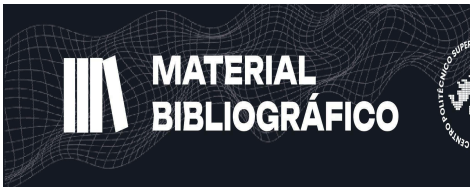
- Entregar en tiempo y forma
- Realizar un 60% de la actividad correctamente.
- Cumplir con las consignas
- Cumplir con el formato requerido.

Cierre:

Y llegamos al final de la clase uno 🙌. En esta primera clase repasado los tipos de datos complejos que nos ofrece Python para poder trabajar 🥵. Vimos las funciones o métodos con los que cuenta cada tipo para poder explotar de mejor manera. Las clases van incrementando la práctica, la metodología que adoptamos es "Learning by doing" es decir aprender haciendo 💪. Recuerden que la mejor manera de adquirir el aprendizaje es haciéndolo con nuestras propias manos. Por tal motivo traten de realizar y llevar a ritmo cada una de las actividades prácticas que se plantean.

La semana que viene en la clase 2, vamos a comenzar a ver **Herramientas de trabajo**, **ámbitos de Python y Sistema de control de versiones**. Les deseo que tengan una excelente semana. Los espero con mucha energía

🔋 listos para que sigamos descubriendo Python. Saludos a todos! 😀



Bibliografía Obligatoria:

- Ramírez Jiménez, Ó. (2021). *Python a fondo*. Marcombo.

Recursos adicionales:

- ***Python Comments*. (n.d.). W3Schools. Retrieved April 7, 2023, from https://www.w3schools.com/python/python_comments.asp**
- ***El tutorial de Python — documentación de Python - 3.11.3*. (n.d.). Python Docs. Retrieved April 7, 2023, from https://docs.python.org/es/3/tutorial/**
- ***PY4E-ES - Python para todos*. (n.d.). PY4E-ES - Python para todos. Retrieved April 7, 2023, from https://es.py4e.com/lessons**
- ***Python Ya*. (n.d.). Tutoriales Ya. Retrieved April 7, 2023, from https://www.tutorialesprogramacionya.com/pythonya/index.php?inicio=45**
- *Python - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN*. (2022, November 29). MDN Web Docs. Retrieved April 7, 2023, from https://developer.mozilla.org/es/docs/Glossary/Python