

## PROGRAMACIÓN II

### 1º AÑO

#### Clase N° 1: Tipo de datos complejos en Python

##### Contenido:

En la clase de hoy trabajaremos los siguientes temas:

- Secuencias.
- Listas.
- Tuplas.
- Diccionarios

#### 1. 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 en nuestra vida cotidiana. Desde alquilar casas hasta pagar impuestos, todos estos procesos dependen de programadores que escriben el código necesario. A través de este curso,

trabajaremos con Python, un lenguaje multipropósito muy versátil, ideal para la ciencia de datos.

Hoy, abordaremos los tipos de datos complejos en Python. ¿Qué pasa si queremos almacenar múltiples datos? ¡Python tiene la solución! Durante esta clase, exploraremos cómo utilizar listas, tuplas y diccionarios para manejar información de manera efectiva. 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 😊

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!!



## **2. Desarrollo y Actividades:**

### **¿Cómo se relaciona la IA con la programación?**

En programación, la IA se implementa a través de algoritmos y modelos que pueden aprender de los datos.

Python es uno de los lenguajes más populares para trabajar con IA debido a su sintaxis sencilla y a la gran cantidad de bibliotecas disponibles.

### **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**

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

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:

```
1 lista1 = [1,"perro",{ 'edad': 40}]
2 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**

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

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



### 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.

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

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

```
1 mi_lista = [1,2,3]
2 mi_lista[4]
3 -----
4 IndexError                                Traceback (most recent call last)
5 c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\coleccion.es.ipynb Celda 1 in 2
6     1 mi_lista = [1,2,3]
7     ----> 2 mi_lista[4]
8
9 IndexError: list index out of range
```

## Agregar elementos a una lista

Las listas de Python tiene tres métodos para agregar elementos:

### Append

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

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.

```
1 lista1.append(4)
2 print(lista1)
3 [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 índice.

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

## Extend

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

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

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

```
1 lista1.pop(0)
2 1
3 print(lista1)
4 [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.

```
1 list1 =[1,2,3,4,5]
2 del(list1[0])
3 print(list1)
4 [2, 3, 4, 5]
```



# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



```
1 del(list1)
2 print(list1)
3 -----
4 NameError                                Traceback (most recent call last)
5 c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\coleccion.es.ip
  ynb Celda 7 in 2
6     1 del(list1)
7 ----> 2 print(list1)
8
9 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()**

```
1 list1 = [1,2,3]
2 list1.clear()
3 print(list1)
4 []
```

## 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

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

## Contar ocurrencias de un valor

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

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

## 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

```
1 list1 = [10,15,2,4,1]
2 list1.sort()
3 print(list1)
4 [1, 2, 4, 10, 15]
```

### Descendente

```
1 list1 = [10,15,2,4,1]
2 list1.sort(reverse = True)
3 print(list1)
4 [15, 10, 4, 2, 1]
```

## 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

```
1 list1 = [10,15,2,4,1]
2 sorted(list1)
3 [1, 2, 4, 10, 15]
4 print(list1)
5 [10, 15, 2, 4, 1]
```

## Descendente

```
1 list1 = [10,15,2,4,1]
2 sorted(list1,reverse=True)
3 [15, 10, 4, 2, 1]
4 print(list1)
5 [10, 15, 2, 4, 1]
```

## 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

```
1 list1 = [10,'a',2,'c',1]
2 list1.sort()
3 -----
4 TypeError                                Traceback (most recent call last)
5 c:\Politecnico Malvinas\Técnicatura en Ciencia de Datos\Programación II\coleccion.es.ip
  ynb Celda 9 in 2
6     1 list1 = [10,'a',2,'c',1]
7     ----> 2 list1.sort()
8
9 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 sintaxis 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

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



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.

```
1 list1 = [1,2,3,4,5,6,7,8]
2 list1[0:3] # muestra los primeros tres elementos de la lista
3 [1, 2, 3]
4
5 list1[:3] # comienza en el primer elemento de la lista, en la posición 0
6 [1, 2, 3]
7
8 list1[4:] # comienza en el elemento índice 4 y va hasta el final de la lista
9 [5, 6, 7, 8]
10
11 list1[::2] # va desde el inicio hasta el final de la lista recorriendo de 2 en 2
12 [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.

## ***Tuplas***

Una 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

## **Tuplas vs Listas**

- 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 ( ).

```
1 tupla1 = (1,2,3,4,5)
2 print(type(tupla1))
3 <class 'tuple'>
```

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.

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

## 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.

```
1 tupla1 = (1,2,3,4,5)
2 tupla2 = (*tupla1,6,7)
3 print(tupla2)
4 (1, 2, 3, 4, 5, 6, 7)
```

## 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.

```
1 tupla1 = (1,2,3,4,5)
2 tupla2 = (*tupla1,6,7)
3 print(tupla2)
4 (1, 2, 3, 4, 5, 6, 7)
```

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

## **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:

```
1 telefonos = {'Gonzalo':12345678,  
2             'Pedro':986455882,  
3             'Maria':445566887}  
4 diccionario_vacio = {}  
5 telefonos['Maria']  
6 445566887
```

## 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.

```
1 telefonos['Veronica']=77889977  
2 del(telefonos['Gonzalo'])  
3 telefonos  
4 {'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()`.

```
1 telefonos  
2 {'Pedro': 986455882, 'María': 445566887, 'Veronica': 77889977}  
3 telefonos['Veronica']=11111111  
4 telefonos  
5 {'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.

```
1  diccionario={'sub_diccionario':{'a':30,'b':False},
2             'lista1':[10,20,30]}
3  diccionario
4  {'sub_diccionario': {'a': 30, 'b': False}, 'lista1': [10, 20, 30]}
5  diccionario['sub_diccionario']['b']
6  False
7  diccionario['lista1'][1]
8  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.

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

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



**values()**: devuelve un objeto de tipo dict\_values del diccionario con los valores que contiene.

```
1 diccionario.values()
2 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.

```
1 diccionario.get('lista1')
2 [10, 20, 30]
3 diccionario.get('Gonzalo')
4
5 diccionario.get('precio', 30000)
6 30000
```

## Comprobar si existe una clave en un diccionario

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

```
1 'lista1' in diccionario
2 True
3 'lista1' not in diccionario
4 False
```

## 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.

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

```
1 diccionario1 = {
2     "marca": "Renault",
3     "modelo": "Koleos",
4     "anio": 2014
5 }
6 diccionario1.pop('modelo')
7 print(diccionario1)
8 {'marca': 'Renault', 'anio': 2014}
```

## **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. Agregar 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.

## Actividad: Clasificación de Frutas

**Objetivo:** Aprender a utilizar listas y diccionarios en Python para organizar información sobre frutas y simular un proceso de clasificación básico, introduciendo así el concepto de inteligencia artificial.

### Descripción de la Actividad:

1. **Crea una lista de frutas:** Define una lista que contenga nombres de frutas. Por ejemplo:

```
frutas = ["manzana", "plátano", "naranja", "kiwi", "uva"]
```

2. **Crea un diccionario de características:** Para cada fruta, crea un diccionario que contenga sus características como peso y color. Por ejemplo:

```
frutas_info = {  
  
    "manzana": {"peso": 150, "color": "rojo"},  
  
    "plátano": {"peso": 120, "color": "amarillo"},  
  
    "naranja": {"peso": 180, "color": "naranja"},  
  
    "kiwi": {"peso": 75, "color": "verde"},  
  
    "uva": {"peso": 50, "color": "morado"},  
}
```



3. **Simulación de clasificación:** Imagina que una inteligencia artificial debe clasificar una fruta basándose en sus características. Define una fruta a clasificar con un peso y un color. Luego, identifica manualmente a qué fruta corresponde en el diccionario. Escribe el código necesario para hacer esta identificación.
4. **Imprime el resultado:** Una vez que hayas clasificado la fruta, imprime el resultado en pantalla.
5. **Reflexiona sobre el proceso:** Al finalizar, reflexiona sobre cómo tomaste la decisión de clasificar la fruta. Pregúntate:
  - ¿Cómo decidiste qué fruta era?
  - ¿Qué características te ayudaron en la identificación?

## Breve Explicación del Concepto de Clasificación en IA

La clasificación es una tarea común en inteligencia artificial que consiste en asignar una etiqueta o categoría a un conjunto de datos basado en sus características. En otras palabras, se trata de identificar a qué clase pertenece un objeto o dato en función de su información.

Por ejemplo, en esta actividad, al clasificar frutas según su peso y color, estamos simulando cómo una inteligencia artificial podría "decidir" a qué fruta corresponde un conjunto de características. Este proceso es fundamental en muchas aplicaciones de IA, como el reconocimiento de imágenes, la detección de spam en correos electrónicos y la categorización de textos.

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



A través de la clasificación, los sistemas de IA pueden aprender a reconocer patrones y hacer predicciones sobre datos nuevos basándose en ejemplos previos. Esta capacidad de aprender de la experiencia es lo que permite a la inteligencia artificial ser tan poderosa y versátil en diversas aplicaciones.



### Foro Obligatorio

**Objetivo:** Los estudiantes deberán analizar el artículo y reflexionar sobre el papel de la inteligencia artificial en el aprendizaje de la programación.

<https://www.genbeta.com/inteligencia-artificial/me-alegro-que-ia-no-existiera-cuando-aprendi-a-programar-gen-z-que-usa-todos-dias-explica-que-ve-mal-para-aprender>

### Instrucciones:

1. **Lectura:** Lee el artículo detenidamente.
2. **Respuestas a Preguntas:** Responde a las preguntas de reflexión proporcionadas en el foro, centrándote en tu perspectiva sobre los argumentos presentados por el autor.
3. **Participación en el Foro:** Comparte tus respuestas en el foro y comenta sobre las reflexiones de tus compañeros, fomentando un debate constructivo.

### Criterios de Evaluación:

1. **Claridad y Coherencia en las Respuestas (30%):**
  - Las respuestas deben ser claras, bien estructuradas y fáciles de seguir. Se evaluará la capacidad del estudiante para expresar sus ideas de manera coherente y lógica.
2. **Profundidad del Análisis:**

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



- Se valorará la profundidad del análisis crítico del artículo. Los estudiantes deben demostrar que han comprendido los puntos clave del texto y que pueden relacionarlos con sus propias experiencias y conocimientos.

### 3. **Argumentación y Justificación:**

- Los estudiantes deben respaldar sus opiniones con argumentos sólidos y ejemplos concretos. Se evaluará la capacidad para justificar sus afirmaciones y reflexiones de manera efectiva.

### 4. **Participación y Colaboración en el Foro:**

- Se tendrá en cuenta la participación activa en el foro, incluyendo la capacidad de responder a las intervenciones de otros compañeros de manera constructiva y respetuosa.

## **ACTIVIDAD INTEGRADORA DE CIERRE**

En esta actividad vas a aplicar todos los conocimientos trabajados en la clase:

Practicar listas, tuplas y diccionarios en un solo ejercicio sencillo.

### **Consigna**

#### 1. **Lista de compañeros**

- Crear una lista llamada **compañeros** con **3 tuplas**, donde cada tupla tenga:
  - Nombre (str)
  - Edad (int)

#### 2. **Mostrar datos**

Imprimir el primer y el último compañero de la lista.

### 3. **Modificar un dato**

- Convertir la tupla del segundo compañero en lista, cambiar su edad sumando 1 año, volverla a tupla y reemplazarla en la lista.

### 4. **Diccionario de hobbies**

- Crear un diccionario llamado **hobbies** donde la **clave** sea el nombre de cada compañero y el **valor** sea su hobby favorito.

### 5. **Mostrar un hobby**

- Usar **.get()** para mostrar el hobby de uno de los compañeros.

### **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**

# PROGRAMACIÓN II

## 1º 2º Cuatrimestre



**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](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>