UNIVERSIDAD SAN SEBASTIAN

VOCACIÓN POR LA EXCELENCIA

Taller de Programación II

Clase Inicial

Profesora: PhD. Liset González Rodríguez

El docente

Nombre:

Lázara Liset González Rodríguez

Formas de Contacto:

lagonzalez@udec.cl

- Ingeniera en Telecomunicaciones y Electrónica, con un Doctorado en Ciencias de la Ingeniería en la Universidad de Concepción.

- Actualmente, soy investigadora independiente y estoy asociada a BrainLat, en la Universidad Adolfo Ibáñez.

- A lo largo de mi carrera, me he especializado en la ciencia de datos aplicada a datos biomédicos, con un énfasis particular en el análisis de neuroimágenes.

BrainLat INSTITUTE
UNIVERSIDAD ADOLFO IBÁÑEZ

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

Bienvenidos

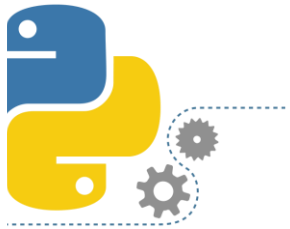
Este programa de asignatura “**Taller de Programación II**" abarca la recapitulación del conjunto de instrucciones básicas del lenguaje Python, la explotación de datos utilizando Python y la publicación de Datos en la nube.

En cada clase te presentaremos los conceptos teóricos y prácticos necesarios, con ejemplos y ejercicios de laboratorio para reforzar tu aprendizaje. Al finalizar este curso, deberías tener una comprensión sólida de los fundamentos de la programación, la capacidad de trabajar con archivos de datos y procesarlos, la habilidad para realizar análisis de datos de distintas fuentes y la visualización de datos presentados en Dashboards de información utilizando sistemas de desarrollo básicos.

Ruta de Aprendizaje

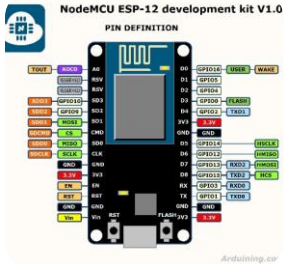
Python Básico

- Revisión de Conceptos
- Gestión de Archivos
- Procesamiento de Textos



Introducción al análisis de Datos

- Concepto de POO
- Uso de librerías Pandas
- Uso librería Matplotlib y Seaborn



Publicación de Datos

- Visualizaciones
- Conexión de datos.
- Servicios de Publicación

Streamlit

Proyecto Final usando Streamlit

Sistema de Evaluación y Fechas Importantes

- **SOLEMNE 1 (20%):** 3 de Septiembre
- **SOLEMNE 2 (30%):** 15 de Octubre
- **SOLEMNE 3 (40%):** 22 de Noviembre
- **RECUPERATIVA:** 3 de Diciembre
- **CONTROLES (10%):** *EN CLASES*

Nota mínima de aprobación 4,0

Asistencia Requerida

80%

Asistencia mínima requerida

- **Las inasistencias NO son justificables, por lo que debe cuidar en todo momento el % de asistencia a sus clases.**
- **Se considerará como presente a todo estudiante que se encuentre al principio y final de las clases.**
- **Si llega tarde se considerará como presente el estudiante que llegue antes de los 15 minutos de comenzada la clase**

Normas de Convivencia en Laboratorio

- Está prohibido el consumo de alimentos al interior del laboratorio.
- Está prohibido el consumo de bebidas y cualquier líquido al interior del laboratorio.
- El uso de los computadores y material de laboratorio está destinado al seguimiento de la clase, por lo que en ningún caso se permitirá que durante el desarrollo de la clase existan sesiones distintas a las que dirige el docente (videojuegos, transmisiones deportivas, videos musicales, etc.)
- Todo el avance que se logre durante las clases depende de su participación.
- Si un equipo o dispositivo de laboratorio no opera, avise al docente para solicitar soporte de la situación.
- Al finalizar las sesiones de aprendizaje, asegure cerrar TODAS sus sesiones abiertas y dejar apagado el computador que ha utilizado.

REGLAMENTO DE CONVIVENCIA USS

Extracto del Decreto de Rectoría N°103/2022

Artículo 14. Faltas graves

a) **Cometer fraude, engaño** o cualquier otra forma de falta a la ética **en toda especie de evaluación** tales como exámenes, **pruebas, controles u otras actividades académicas**, en actividades prácticas, clínicas asistenciales, de extensión o de vinculación con el medio, dentro o fuera de las instalaciones de la Universidad, como también facilitar o inducir a que dichas conductas se cometan. Constituyen faltas de esta especie, entre otras, el solicitar, recibir, obtener, reproducir o facilitar a cualquier título, sin autorización ni legitimación alguna, pautas, guías y cualquier otro instrumento que deba aplicarse en evaluaciones futuras.

b) **Suplantar a un estudiante o permitir ser suplantado**, por cualquier medio o en cualquier forma, en la rendición de una evaluación académica.

c) Plagiar, adulterar u ocultar el origen de la información en investigaciones, memorias, publicaciones y trabajos en general.

d) Atribuirse como propia la autoría de una obra ajena o la presentación como propias las ideas ajenas, con el objeto de obtener un provecho académico, económico, social o público, o de cualquier otra especie.

REGLAMENTO DE CONVIVENCIA USS

Extracto del Decreto de Rectoría N°103/2022

Artículo 17. Sanciones:

- ***Condicionalidad*** hasta por 4 semestres o 6 trimestres según el régimen académico del estudiante sancionado, tiempo durante el cual no deberá incurrir en falta de igual o mayor gravedad a la que motivó la aplicación de la sanción. De incumplir la condicionalidad el estudiante será suspendido de su calidad de alumno por el mismo tiempo de la condicionalidad originalmente aplicada, sin perjuicio de la sanción que le corresponda por la comisión de la nueva falta.
- ***Suspensión*** de la calidad de estudiante hasta por 2 semestres o por 3 trimestres, según el régimen académico del estudiante sancionado. La suspensión de la calidad de estudiante comprende la inhabilitación absoluta para inscribir asignaturas, asistir a clases, rendir evaluaciones, retirar material bibliográfico y otros bienes disponibles para uso de alumnos de la Universidad, ejercer cargos de ayudantías, y en general cualquier actividad académica, dentro o fuera de los recintos institucionales, durante el tiempo de dicha suspensión
- ***No renovación*** del contrato de prestación de servicios educacionales a contar del año académico inmediatamente siguiente al período en que quede ejecutoriada la resolución sancionatoria.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

REPASO DE CONTENIDOS ANTERIORES

Qué es un lenguaje de Programación

- Un lenguaje de programación es un conjunto de reglas y símbolos que permite escribir instrucciones para que una computadora las ejecute. **Utilidad**: crear programas de software, aplicaciones, scripts y sistemas operativos.
- **Ejemplos**: ensamblador (bajo nivel o cercano al lenguaje de máquina), C y C++ (medio nivel), Python, R o Java (alto nivel, más fáciles de entender y usar)
- Cada lenguaje de programación tiene su propia sintaxis (las reglas que definen cómo se deben escribir las instrucciones) y semántica (el significado de esas instrucciones).

Características de Python

- **Sintaxis Simple y Legible:** Facilita la escritura y lectura del código.
- **Portabilidad:** Multiplataforma: funciona en Windows, macOS, Linux sin cambios significativos.
- **Código Abierto:** Contribución activa de la comunidad para mejorar el lenguaje.
- **Gran Biblioteca Estándar:** Ofrece módulos y paquetes para diversas tareas sin necesidad de software adicional.
- **Interpretable:** Se ejecuta línea por línea, facilitando la depuración.
- **Integrable:** Se puede extender con C/C++ e integrar con Java, .NET, entre otros.
- **Gran Comunidad y Soporte:** Amplia disponibilidad de recursos, tutoriales y ayuda.
- **Aplicaciones Versátiles:** Utilizado en desarrollo web, análisis de datos, IA, automatización, etc.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

VARIABLES

¿Qué es una variable en Python?

¿Cómo se definen las variables en Python?

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿Qué es una variable en Python?

Una variable es un nombre que se asigna a un valor almacenado en la memoria del programa. Es un contenedor que guarda datos, como números, cadenas de texto, o estructuras de datos más sofisticadas como listas, diccionarios, etc.

Estos datos se almacenan en la memoria RAM (memoria de acceso aleatorio) del sistema, que es volátil, lo que significa que se borra cuando el programa finaliza o cuando el sistema se apaga.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

DEFINICIÓN DE VARIABLES

nombre_de_la_variable = valor_de_la_variable

- Siempre utilizarlas en minúsculas y/o usar el guión bajo para complementar la definición de esta (para que quede más claro que contiene, ejemplo:
- **Tipos de Variables numéricas**:

  Entero: cantidad_rojos = 4 o **flotante** peso_manzanas = 1,1
- **Condicionales**:

  Tipo boleano: aprobado = TRUE o FALSE
- **Cadena de caracteres alfanuméricos:**

  Tipo string: nombre = “Leonardo22”

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿ Qué hace la función “print()” en Python?

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿ Qué hace la función print() en Python?

print(“Hola mundo”)

print(nombre_variable)

Esta función se utiliza para mostrar en la consola lo que tenga contenido dentro del paréntesis. Se utiliza comillas cuando se quiere mostrar textual el texto escrito. Sin comillas considerara que su contenido es una variable previamente definida y mostrar el valor de esta en la consola.

¿ Qué hace la función input() en Python ?

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿ Qué hace la función input() en Python ?

input()

Esta función se utiliza para pedir por teclado algún valor. Si es numérico y quiere utilizar como tal se debe considerar que:

- Si es **Entero**, entonces quedaría **int(input())**
- Si es **Decimal**, entonces quedaría **float(input())**

para un tipo de valor string basta con **input()**

El valor de entrada debe almacenarse en una variable para que esta puede utilizarse a posteriori.

entero = int(input())

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿Qué son las operadores y que tipos existen?

OPERADORES RELACIONALES (DE COMPARACIÓN)

Símbolo	Significado	Ejemplo	Resultado
==	Igual que	5 == 7	Falso
!=	Distinto que	rojo != verde	Verdadero
<	Menor que	8 < 12	Verdadero
>	Mayor que	12 > 7	Falso
<=	Menor o igual que	12 <= 12	Verdadero
>=	Mayor o igual que	4 >= 5	Falso

OPERADORES LÓGICOS

Operador	Ejemplo	Resultado*	
and (y)	5 == 7 and 7 < 12	0 y 0	Falso
	9 < 12 and 12 > 7	1 y 1	Verdadero
	9 < 12 and 12 > 15	1 y 0	Falso
or (o)	12 == 12 or 15 < 7	1 o 0	Verdadero
	7 > 5 or 9 < 12	1 o 1	Verdadero
xor	4 == 4 xor 9 > 3	1 o 1	Falso
(o excluyente)	4 == 4 xor 9 < 3	1 o 0	Verdadero

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

¿Qué es una estructura de control de flujo en Python?

Las estructuras de control en Python permiten dirigir el flujo de ejecución de un programa. Estas estructuras son esenciales para tomar decisiones en el código, repetir operaciones y controlar el proceso de ejecución en función de distintas condiciones.
Pueden ser condicionales o iterativas

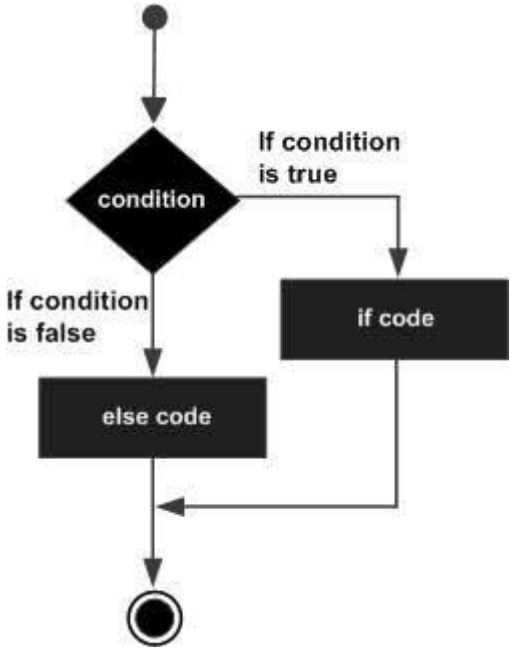
ESTRUCTURA CONDICIONAL

if (si), **else** (sino)

Si se utilizan setencias **if :** siempre irán acompañada de una condición (operadores lógicos y/o racionales).

El final de la línea debe terminar con dos puntos (:)

y las sentencias que deban ejecutarse después de cumplir con la indentación(*Indentar significa hacer espacios hacia la derecha para mover una línea de código, se puede hacer usando la barra espaciadora o con la tecla de tabulación).*




```
if condición:
    aquí van las órdenes que se ejecutan si la condición es cierta
    y que pueden ocupar varias líneas
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURA CONDICIONAL

Más de dos alternativas: if ... elif ... else ...

La estructura de control if … elif … else … permite encadenar varias condiciones. elif es una contracción de else if. La orden en Python se escribe así:

```
if condición_1:
    bloque 1
elif condición_2:
    bloque 2
else:
    bloque 3
```

- Si se cumple la condición 1, se ejecuta el bloque 1
- Si no se cumple la condición 1 pero sí que se cumple la condición 2, se ejecuta el bloque 2
- Si no se cumplen ni la condición 1 ni la condición 2, se ejecuta el bloque 3.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURA CONDICIONAL

Ejemplo 1: Realice un programa usando estructuras condicionales que determine cuál es el mayor entre 3 números que ingrese el usuario y lo imprima en pantalla

```
# Solicitar al usuario que ingrese tres números
num1 = float(input("Ingrese el primer número: "))
num2 = float(input("Ingrese el segundo número: "))
num3 = float(input("Ingrese el tercer número: "))

# Determinar el número mayor utilizando estructuras condicionales
if num1 >= num2 and num1 >= num3:
    mayor = num1
elif num2 >= num1 and num2 >= num3:
    mayor = num2
else:
    mayor = num3

# Mostrar el número mayor
print(f"El mayor de los tres números es: {mayor}")
```

ESTRUCTURA ITERATIVA

while

Permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).

Al igual que la otras estructuras al final siempre debe llevar dos puntos (:)

```
while condición:
    cuerpo del bucle
```

ESTRUCTURA DE CONTROL ITERATIVA

Ejemplo 2: Realice un programa usando un while que determine cuál es el menor entre 5 números que ingrese el usuario y lo imprima en pantalla

contador=contador+1

```
# Inicializar las variables
contador = 0
menor = float('inf')  # Inicializamos con el mayor valor posible

# Usar un bucle while para ingresar 5 números
while contador < 5:
    numero = float(input(f"Ingrese el número {contador + 1}: "))
    if numero < menor:
        menor = numero
    contador += 1

# Mostrar el número menor
print(f"El menor de los cinco números es: {menor}")
```

ESTRUCTURA ITERATIVA

for

repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

Al igual que la otras estructuras al final siempre debe llevar dos puntos (:)

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURA DE CONTROL ITERATIVA

Ejemplo 3: Realice el mismo programa usando un bucle for

```
# Inicializar la variable menor con el mayor valor posible
menor = float('inf')  # Inicializamos con el mayor valor posible

# Usar un bucle for para ingresar 5 números
for i in range(5):
    numero = float(input(f"Ingrese el número {i + 1}: "))
    if numero < menor:
        menor = numero

# Mostrar el número menor
print(f"El menor de los cinco números es: {menor}")
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURA DE CONTROL ITERATIVA

¿ Qué hace la FUNCIÓN range() en Python?

```
range(5)
      STOP
```

GENERA UNA SECUENCIA DE NÚMEROS DESDE 0 HASTA STOP - 1. EN ESTE CASO, RANGE(5) GENERARÁ LOS NÚMEROS 0, 1, 2, 3, 4.

```
range(2, 5)
         STOP
```

GENERA UNA SECUENCIA DE NÚMEROS DESDE START HASTA STOP - 1. EN ESTE CASO, RANGE(2, 5) GENERARÁ LOS NÚMEROS 2, 3, 4.

```
range(1, 10, 2)
         STOP  STEP
```

GENERA UNA SECUENCIA DE NÚMEROS DESDE START HASTA STOP - 1, INCREMENTANDO EN CADA PASO POR EL VALOR DE STEP. EN ESTE CASO, RANGE(1, 10, 2) GENERARÁ LOS NÚMEROS 1, 3, 5, 7, 9.

ESTRUCTURA DE CONTROL ITERATIVA

¿Cuántas veces se imprimirá la palabra “Hola”?

1) **for i in range(10):**
   print (“Hola”)

   Se imprime “Hola” 10 veces

2) **for i in range(10,20,3):**
   print (“Hola”)

   Se imprime “Hola” 4 veces

3) **for i in “Mundo”:**
   print (“Hola”)

   Se imprime “Hola” 5 veces

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURA ITERATIVA

for i in lista:

Lo que va aquí se repite según la cantidad de elementos que tenga la lista.

for i in [1,2,3,3,1]:

Lo que va aquí se repite 5 veces

Para todos los casos la variable **i** toma el valor de cada uno de los elementos que se recorren.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURAS DE CONTROL DE EJECUCIÓN

3. **Control de Flujo en Bucles:** `break`, `continue`, `pass`

Estos comandos se usan para controlar la ejecución dentro de los bucles.

`break`

Sale del bucle inmediatamente.

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Explicación:

- Imprime números del 0 al 4. Sale del bucle cuando `i` es igual a 5.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURAS DE CONTROL DE EJECUCIÓN

`continue`

Salta el resto del código en la iteración actual y pasa a la siguiente iteración del bucle.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Explicación:

- Imprime los números impares del 1 al 9. Salta a la siguiente iteración si el número es par.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

ESTRUCTURAS DE CONTROL DE EJECUCIÓN

`pass`

Es un marcador de posición que no hace nada. Se usa para evitar errores de sintaxis donde se requiere una declaración.

```
for i in range(10):
    if i % 2 == 0:
        pass  # No hace nada, solo sirve como marcador de posición
    else:
        print(i)
```

Explicación:

- Imprime los números impares del 1 al 9. `pass` se usa para los números pares, donde no se realiza ninguna acción.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Qué son las LISTAS en Python?

Las listas en Python son una estructura de datos que permite almacenar una colección ordenada de elementos. Estos elementos pueden ser de cualquier tipo de datos, incluyendo números, cadenas de texto, otros objetos, e incluso otras listas. Las listas son mutables, lo que significa que puedes cambiar su contenido (agregar, eliminar o modificar elementos) después de haberlas creado.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Cuál es su sintaxis?

La sintaxis para crear una lista en Python es muy sencilla. Los elementos de la lista se encierran entre corchetes ([]) y se separan por comas. Ejemplos:

```
lista_vacia = []
```

```
lista_numeros = [1, 2, 3, 4, 5]
```

```
lista_mixta = [1, "manzana", 3.14, True]
```

```
lista_de_listas = [[1, 2, 3], ["a", "b", "c"], [True, False]]
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Para qué se utilizan?

Las listas en Python se utilizan para diversas tareas, tales como:

- **Almacenamiento de Colecciones de Datos:** Puedes usar listas para almacenar una colección de datos relacionados. Por ejemplo, una lista de nombres de estudiantes o una lista de precios de productos.

```
# Lista de nombres de estudiantes
nombres_estudiantes = ["Ana", "Pedro", "Luis", "María", "Jorge"]

# Imprimir la lista completa
print("Lista de nombres de estudiantes:")
print(nombres_estudiantes)

# Acceder e imprimir el primer y último nombre en la lista
print("\nPrimer estudiante:", nombres_estudiantes[0])
print("Último estudiante:", nombres_estudiantes[-1])
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Para qué se utilizan?

Las listas en Python se utilizan para diversas tareas, tales como:

- **Iteración:** Las listas permiten iterar sobre sus elementos usando bucles for, facilitando la ejecución de operaciones repetitivas sobre cada elemento.

```
frutas = ["manzana", "banana", "cereza"]
for fruta in frutas:
    print(fruta)
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Para qué se utilizan?

Las listas en Python se utilizan para diversas tareas, tales como:

- **Acceso a Elementos:** Puedes acceder a elementos individuales de una lista usando índices, con el primer elemento teniendo un índice de 0.

```
lista = ["a", "b", "c"]
print(lista[0])  # Imprime 'a'
print(lista[2])  # Imprime 'c'
```

LISTAS

- ¿Para qué se utilizan?

Las listas en Python se utilizan para diversas tareas, tales como:

- **Modificación de Elementos:** Las listas son mutables, por lo que puedes cambiar los valores de sus elementos.

```
lista = [1, 2, 3]
lista[1] = 20
print(lista)  # Imprime [1, 20, 3]
```

LISTAS

- ¿Para qué se utilizan?

  - **Agregar y Eliminar Elementos:** Puedes agregar elementos a una lista usando métodos como append() y insert(), y eliminar elementos usando remove() o pop().

```
lista = [1, 2, 3]
lista.append(4)  # Agrega 4 al final de la lista
print(lista)     # Imprime [1, 2, 3, 4]

lista.remove(2)  # Elimina el primer valor 2 de la lista
print(lista)     # Imprime [1, 3, 4]

lista.pop()      # Elimina el último elemento de la lista
print(lista)     # Imprime [1, 3]
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

- ¿Para qué se utilizan?

- **Operaciones y Métodos de Listas:** Las listas soportan una variedad de operaciones y métodos incorporados, como concatenación, repetición, ordenación, búsqueda, entre otros.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_concatenada = lista1 + lista2
print(lista_concatenada)  # Imprime [1, 2, 3, 4, 5, 6]
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

lista = [2,35,2,5]

Otras funciones para trabajar sobre los datos en las listas son:

len()

- len(lista), retorna la cantidad de elementos en la lista

extend(iterable)

- lista.extend(*lista2*), agrega los elementos de la lista2 al final de la lista (las fusiona sobre lista)

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

insert(i,x)

- lista.insert(i,x), agrega el valor x en la posición i

clear()

- lista.clear(), quita todos los elementos de la lista

index(x)

- lista.index(x), devuelve la posición del primer valor encontrado de x.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

LISTAS

index(x,inicio,fin)

- lista.index(x,inicio,fin), devuelve la posición del primer valor encontrado de x entre el rango de inicio y fin.

copy(*x*)

- lista.copy(), copia la lista la puede ser almacenada en otra variable

  lista2 = lista.copy()

  Así la lista2 tendrá los mismos valores que lista.

max(lista)

min(lista)

UNIVERSIDAD
SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

Fichero (o archivo) es una colección de datos que se almacenan en una ubicación específica en el sistema de archivos de la computadora. Los ficheros pueden contener texto, datos binarios, imágenes, audio, u otros tipos de información, y se pueden leer y escribir utilizando las funciones y métodos que Python proporciona para la manipulación de archivos.

Ficheros

En Python, trabajar con ficheros (archivos) es una tarea común y se puede realizar utilizando las funciones integradas que proporciona el lenguaje.

El trabajo con ficheros permite realizar diversas tareas como por ejemplo:

1. Almacenamiento Permanente de Datos: Los ficheros permiten almacenar datos de forma permanente, a diferencia de la memoria RAM, que es volátil. Esto es útil para guardar la información que debe persistir entre ejecuciones del programa.

2. Lectura de Datos de Entrada: Los ficheros permiten leer grandes cantidades de datos de entrada sin tener que ingresarlos manualmente cada vez que se ejecuta el programa.

3. Generación de Informes: Los programas a menudo necesitan generar informes o resultados que se deben guardar para revisión o análisis posteriores. Estos informes pueden ser en formato de texto, CSV, PDF, etc.

4. Manejo de Logs: El registro de eventos y errores en ficheros de log es crucial para el monitoreo y depuración de aplicaciones. Los logs permiten rastrear el comportamiento del software y detectar problemas.

7. Procesamiento de Datos Masivos: Para aplicaciones que necesitan procesar grandes cantidades de datos, como análisis de datos o big data, los ficheros son una herramienta esencial para almacenar y gestionar estos datos.

Manipulación de Ficheros

1. Abrir un fichero

Para abrir un fichero en Python, se usa la función `**open()**`. Esta función toma al menos dos argumentos: el nombre del fichero y el modo en el que se va a abrir.

Modos comunes para abrir un fichero:

- `**'r'**`: Lectura (modo predeterminado).
- `**'w'**`: Escritura (crea un fichero nuevo o sobrescribe uno existente).
- `**'a'**`: Añadir (agrega al final del fichero).
- `**'b'**`: Modo binario.
- `**'+'**`: Actualización (lectura y escritura).

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

Ejemplo:

```
# Abrir un fichero en modo de lectura
fichero = open('archivo.txt', 'r')
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

¿Cómo leo un fichero?

2. **Leer de un fichero**.

Existen varias maneras de leer de un fichero en Python: read(), readline(), y readlines().

```
contenido = fichero.read()
print(contenido)
```

read(): Lee todo el contenido del fichero.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

¿Cómo leo un fichero?

2. Leer de un fichero.

Existen varias maneras de leer de un fichero en Python: read(), readline(), y readlines().

```
linea = fichero.readline()
print(linea)
```

readline(): Lee una línea del fichero.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

¿Cómo leo un fichero?

2. Leer de un fichero.

Existen varias maneras de leer de un fichero en Python: read(), readline(), y readlines().

```
lineas = fichero.readlines()
for linea in lineas:
    print(linea)
```

readlines(): Lee todas las líneas del fichero y las devuelve como una lista.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

¿Cómo leo un fichero línea por línea?

Para esto primero debemos saber si existen elementos en nuestro fichero, si esto es así mostramos la línea leída y luego leemos la siguiente hasta que encontremos una línea vacía que indica que llegamos al final del archivo o fichero.

```
f = open('prueba.txt','r')
linea = f.readline()
while linea != "":
    print(linea)
     linea = f.readline()
f.close()
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

En Python, existen dos opciones para abrir los archivos:

La opción 1 es la que vimos anteriormente usando la función open(). Esta opción requiere que, para cada archivo abierto, luego se cierre usando la siguiente sintaxis:

```
# Abriendo un archivo, leyendo su contenido y luego cerrándolo
archivo = open('ejemplo.txt', 'r')  # Abre el archivo en modo lectura
contenido = archivo.read()  # Lee el contenido del archivo
print(contenido)  # Imprime el contenido del archivo
archivo.close()  # Cierra el archivo
```

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

FICHEROS

En Python, existen dos opciones para abrir los archivos:

La opción 2 es usando la palabra reservada *with*

Ambas opciones se utilizan para abrir archivos, pero tienen diferencias importantes en cómo manejan el cierre de archivos.

```
# Abriendo y leyendo un archivo con 'with open'
with open('ejemplo.txt', 'r') as archivo:
    contenido = archivo.read()  # Lee el contenido del archivo
    print(contenido)  # Imprime el contenido del archivo
```

FICHEROS

En el caso del uso de with:

Ventajas:
1)Automáticamente cierra el archivo. El bloque with garantiza que el archivo se cierre automáticamente al finalizar el bloque, incluso si ocurre una excepción. Esto es importante para evitar fugas de recursos y posibles problemas de acceso concurrente.
2) Código más limpio y seguro: Al usar with, no necesitas preocuparte por cerrar el archivo manualmente con archivo.close(), lo que hace que el código sea más limpio y menos propenso a errores.

En el caso del uso del open() sin with:

Ventajas:
1) Control manual: Puedes controlar explícitamente cuándo cerrar el archivo, pero esto requiere recordar llamar a archivo.close() para liberar los recursos.

Desventajas:
1) Posible olvido de cierre: Si olvidas llamar a archivo.close(), el archivo puede permanecer abierto, lo que puede llevar a problemas como la falta de liberación de recursos o la corrupción de datos.

En resumen, usar with open() es generalmente preferido por su simplicidad y seguridad, ya que gestiona automáticamente el cierre del archivo.

UNIDAD 1: Gestión de archivos

Uso del with:

```
with open("lista_compras.txt") as archivo:
    print(archivo.read())
```

```
archivo = open("lista_compras.txt")
print(archivo.read())
archivo.close()
```

FICHEROS

¿Cómo escribo sobre un fichero?

Al momento de abrir el archivo primero debemos definir el modo que más nos acomoda, estos pueden ser:

"x": únicamente crear el fichero (da error si ya existe el fichero)

"w": escribir (crea el fichero si no existe y borra el contenido anterior del fichero)

"a": añadir (crea el fichero si no existe, no borra el contenido existente y escribe al final del fichero)

ESCRITURA DE FICHEROS

1. Escribir en un archivo (modo `'w'`)

El modo `'w'` abre el archivo para escritura y crea el archivo si no existe. Si el archivo ya existe, su contenido se sobrescribe.

```
# Abre el archivo en modo escritura
with open('archivo.txt', 'w') as archivo:
    archivo.write('Este es el nuevo contenido del archivo.\n')  # Escribe una línea
```

ESCRITURA DE FICHEROS

2. **Agregar contenido a un archivo existente (modo `'a'`)**

El modo `'a'` abre el archivo para agregar contenido al final del archivo sin sobrescribir el contenido existente.

```
# Abre el archivo en modo agregar
with open('archivo.txt', 'a') as archivo:
    archivo.write('Este es un contenido agregado al final del archivo.\n')  # Añade
```

ESCRITURA DE FICHEROS

3. **Escribir múltiples líneas en un archivo**

Puedes usar el modo `'w'` o `'a'` para escribir varias líneas al mismo tiempo. Aquí se muestra cómo hacerlo con una lista de líneas.

```
# Abre el archivo en modo escritura
with open('archivo.txt', 'w') as archivo:
    lineas = ['Primera línea.\n', 'Segunda línea.\n', 'Tercera línea.\n']
    archivo.writelines(lineas)  # Escribe múltiples líneas en el archivo
```

ESCRITURA DE FICHEROS

5. Escribir en binario (modo `'wb'`)

Si necesitas escribir datos binarios, usa el modo `'wb'`.

```
# Abre el archivo en modo escritura binaria
with open('archivo.bin', 'wb') as archivo:
    datos = bytes([120, 3, 255, 0])  # Datos binarios a escribir
    archivo.write(datos)  # Escribe los datos binarios en el archivo
```

UNIDAD 1: Gestión de archivos

Ejemplo 1: Leer un archivo de texto:
Supongamos que tenemos un archivo de texto llamado "ejemplo.txt" con el siguiente contenido:

Hola, este es un ejemplo de archivo de texto.
Estamos aprendiendo a trabajar con archivos en Python.
Espero que encuentres esto útil.

Lea el contenido del archivo e imprímalo en pantalla

```
# Abre el archivo en modo lectura
with open('ejemplo.txt', 'r') as archivo:
    contenido = archivo.read()  # Lee todo el contenido del archivo
    print(contenido)  # Imprime el contenido del archivo en pantalla
```

UNIDAD 1: Gestión de archivos

Tarea 1: Escribir hacia un archivo de texto:
Supongamos que tenemos un archivo de texto llamado "ejemplo.txt" con el siguiente contenido:

Hola, este es un ejemplo de archivo de texto.
Estamos aprendiendo a trabajar con archivos en Python.
Espero que encuentres esto útil.

Lea el contenido del archivo y copie las últimas dos líneas en otro archivo llamado doslineas_ejemplo.txt e imprima en pantalla las líneas copiadas

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

Tarea 2

Descripción del Problema

Crea un programa que permita a un usuario registrar sus datos personales en un archivo de texto. El programa debe permitir al usuario ingresar su nombre, edad y correo electrónico. Luego, el programa debe guardar esta información en un archivo llamado `**registros.txt**` en el siguiente formato:

```
Nombre: [nombre]
Edad: [edad]
Correo: [correo]
```

Cada nuevo registro debe ser añadido al final del archivo.

Requisitos

1. El programa debe permitir al usuario ingresar múltiples registros.
2. Después de cada registro, el programa debe preguntar si el usuario desea agregar otro registro.
3. Cuando el usuario decida no agregar más registros, el programa debe finalizar.

UNIVERSIDAD SAN SEBASTIAN
VOCACIÓN POR LA EXCELENCIA

EJEMPLO DE EJECUCIÓN

```
Ingrese su nombre: Ana
Ingrese su edad: 28
Ingrese su correo electrónico: ana@example.com
¿Desea agregar otro registro? (sí/no): sí

Ingrese su nombre: Luis
Ingrese su edad: 35
Ingrese su correo electrónico: luis@example.com
¿Desea agregar otro registro? (sí/no): no
```

EJEMPLO DE ARCHIVO FINAL

```
Nombre: Ana
Edad: 28
Correo: ana@example.com

Nombre: Luis
Edad: 35
Correo: luis@example.com
```