

APUNTES PYTHON

Manuel Vergara – 2023

Índice

TEMA 1 - Programa un creador de nombres.....	6
1.1. - print.....	6
1.2. - strings.....	6
1.3. - input.....	7
1.4. - Proyecto del Día 1.....	7
TEMA 2 - Programa un calculador de comisiones.....	8
2.1. - Tipos de datos.....	8
2.2. - Variables.....	8
2.3. - Nombres de variables.....	9
2.4. - integers & floats.....	9
2.5. - Conversiones.....	10
2.6. - Formatear cadenas.....	10
2.7. - Operadores matemáticos.....	10
2.8. - Redondeo.....	11
2.9. - Proyecto del Día 2.....	11
TEMA 3 - Programa un analizador de texto.....	12
3.1. - Index().....	12
3.2. - Substrings.....	13
3.3. - Métodos para Strings.....	13
3.4. - Strings: propiedades.....	17
3.5. - Listas.....	18
3.6. - Diccionarios.....	19
3.7. - Tuples.....	20
3.8. - sets.....	20
3.9. - booleanos.....	24
3.10. - Proyecto del Día 3.....	25
TEMA 4 - Programa el juego "adivina el número".....	27
4.1. - Operadores de comparación.....	27
4.2. - Operadores lógicos.....	28
4.3. - Control de flujo.....	28
4.4. - loops while.....	29
4.5. - range().....	29
4.6. - enumerate().....	30
4.7. - zip().....	30

4.8. - min() & max().....	31
4.9. - random.....	31
4.10. - Comprensión de listas.....	32
4.11. - match.....	32
4.12. - Proyecto del Día 4.....	33
TEMA 5 - Programa el juego "El ahorcado"	36
5.1. - Documentación.....	36
5.2. - Funciones.....	36
5.3. - return.....	37
5.4. - Funciones dinámicas.....	37
5.5. - Interacción entre funciones.....	38
5.6. - *args.....	38
5.7. - **kwargs.....	39
5.8. - Ejercicios.....	39
5.9. - Proyecto del Día 5.....	45
TEMA 6 - Programa un recetario.....	47
6.1. - Abrir y leer archivos.....	47
6.2. - Crear y escribir archivos.....	47
6.3. - Directorios.....	48
6.4. - pathlib.....	48
6.5. - Path.....	49
6.6. - Limpiar la consola.....	50
6.7. - Archivos + funciones.....	50
6.8. - Proyecto del Día 6.....	50
TEMA 7 - Programa una cuenta bancaria.....	53
7.1. - Clases.....	53
7.2. - Atributos.....	54
7.3. - Métodos.....	54
7.4. - Tipos de métodos.....	55
7.5. - Herencia.....	56
7.6. - Herencia extendida.....	56
7.7. - Polimorfismo.....	57
7.8. - Pilares de la Programación Orientada a Objetos.....	57
7.9. - Métodos especiales.....	58
7.10. - Proyecto del Día 7.....	58
TEMA 8 - Programa una consola de turnos.....	60
8.1. - Instalar paquetes.....	60
8.2. - Módulos y paquetes.....	60
8.3. - Manejo de errores.....	61
8.4. - pylint.....	62
8.5. - unittest.....	62
8.6. - Decoradores.....	63
8.7. - Generadores.....	63
8.8. - Proyecto del Día 8.....	64
TEMA 9 - Programa un buscador de números de serie.....	65
9.1. - Módulo collections.....	65
9.2. - Módulos shutil & os.....	66
9.3. - Módulo datetime.....	66

9.4. - Módulo para medir el tiempo.....	67
9.5. - Módulo math.....	68
9.6. - Expresiones regulares.....	69
9.7. - Comprimir y descomprimir archivos.....	70
9.8. - Proyecto del Día 9.....	70
TEMA 10 - Programa el juego "Invasión espacial"	71
10.1. - Distancia entre dos puntos.....	71
10.2. - Convertir el Juego en un Archivo Ejecutable (.exe).....	72
TEMA 11 - Programa un extracto de datos web.....	76
11.1. - Extraer elementos de una clase.....	76
TEMA 12 - Programa un gestor de restaurantes.....	77
TEMA 13 - Programa un asistente de voz.....	78
13.1. - Librerías y módulos.....	78
13.2. - Algunos problemas con las bibliotecas.....	79
13.3. - Enlaces.....	81
TEMA 14 - Programa un controlador de asistencia.....	82
14.1. - Bibliotecas.....	82
TEMA 15 - Programa un modelo de machine learning.....	84
15.1. - Bibliotecas.....	84
15.2. - Definiciones.....	85
15.3. - Cuadernos de trabajo en Colab de google drive.....	86
TEMA 16 - Programa una aplicación web de tareas pendientes.....	87
16.1. - Entornos Virtuales.....	87
16.2. - Módulos.....	89
16.3. - Preparación de estructura de trabajo.....	90
16.4. - Configurar url.....	93
16.5. - Crear tabla de tareas.....	94
16.6. - Configurar la vista.....	99
16.7. - Configurar la vista de Detalle.....	102
16.8. - Crear Links a Detalle.....	103
16.9. - Agregar nueva tarea.....	104
16.10. - Formulario para nueva tarea.....	106
16.11. - Editar tarea.....	107
16.12. - Eliminar tarea.....	108
16.13. - Crear la lógica de Logueo / Deslogueo.....	109
16.14. - Formulario de Logueo / Deslogueo.....	110
16.15. - Restringir acceso.....	112
16.16. - Información específica de usuario.....	112
16.17. - Registrar nuevo usuario.....	114
16.18. - Barra de búsquedas.....	117
16.19. - Un estilo para todas las vistas.....	119
16.20. - Estilo general.....	121
16.21. - Estilo de barra superiores.....	122
16.22. - Estilo de la lista.....	124
16.23. - Estilo de la barra de búsqueda.....	128
16.24. - Terminar el sitio.....	131
TEMA 17 - Extra. Bibliotecas para hacking ético.....	132
17.1. - Bibliotecas.....	132

Este documento contiene los apuntes tomados en el curso «[Python total](#)» impartido por «Escuela Directa» en enero y febrero de 2023. El curso udemy consta de 30 horas aproximadamente de vídeo-tutoriales. Las prácticas aquí contenidas tuvieron una duración de alrededor de unas 150 horas.

Los apuntes no fueron pensados para compartirlos, por ello pueden tener lagunas de información o contenido adicional respecto al curso, ya que se redactaron para recordar procedimientos y conceptos que el autor creyó relevantes. Teniendo un documento, a mi parecer, tan completo y entendiendo que el conocimiento debe ser libre se decidió compartirlo.

Si te parece útil este documento puedes agradecerlo a través de las vías de contacto de la web <https://vergaracarmona.es>

Recuerda,

"Quien se corta su propia leña se calienta dos veces"



Esta obra está bajo una [Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/legalcode.es). Para ver una copia de esta licencia, visite <https://creativecommons.org/licenses/by-sa/4.0/legalcode.es>.

Usted es libre de:

- **Compartir** — copiar y redistribuir el material en cualquier medio o formato
 - **Adaptar** — remezclar, transformar y crear a partir del material para cualquier finalidad, incluso comercial.
- Bajo las condiciones siguientes:

- **Reconocimiento** — Debe reconocer adecuadamente la autoría, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puede hacerlo de cualquier manera razonable, pero no de una manera que sugiera que tiene el apoyo del licenciador o lo recibe por el uso que hace.
- **Compartir Igual** — Si remezcla, transforma o crea a partir del material, deberá difundir sus contribuciones bajo la misma licencia que el original.



- **No hay restricciones adicionales** — No puede aplicar términos legales o medidas tecnológicas que legalmente restrinjan realizar aquello que la licencia permite.



Esta licencia está aceptada para Obras Culturales Libres.
El licenciador no puede revocar estas libertades mientras cumpla con los términos de la licencia.

TEMA 1 - Programa un creador de nombres

Para instalar python: <https://wiki.python.org/moin/BeginnersGuide/Download>

El IDE que se usará en el curso es pyCharm: <https://www.jetbrains.com/pycharm/download/>

1.1. - print

print - Declaración que al ejecutarse muestra (o imprime) en pantalla el argumento que se introduce dentro de los paréntesis.

1.1.1. - Mostrar texto

Ingresamos entre comillas simples o dobles los caracteres de texto que deben mostrarse en pantalla.

```
print("Hola mundo")
>> Hola mundo
```

1.1.2. - Mostrar números

Podemos entregarle a print() el número que debe mostrar, o una operación matemática a resolver. No empleamos comillas en estos casos.

```
Print(150 + 50)
>> 200
```

1.2. - strings

Los strings en Python son un tipo de dato formado por cadenas (o secuencias) de caracteres de cualquier tipo, formando un texto.

1.2.1. - Concatenación

Unificación de cadenas de texto:

```
print("Hola" + " " + "mundo")
>> Hola mundo
```

1.2.2. - Caracteres especiales

Indicamos a la consola que el caracter a continuación del símbolo \ debe ser tratado como un caracter especial.

- \ " > Imprime comillas
- \ n > Separa texto en una nueva línea
- \ t > Imprime un tabulador
- \\ > Imprime la barra invertida textualmente

1.3. - input

Función que permite al usuario introducir información por medio del teclado al ejecutarse, otorgándole una instrucción acerca del ingreso solicitado. El código continuará ejecutándose luego de que el usuario realice la acción.

```
input("Dime tu nombre: ")
>> Dime tu nombre: |
print("Tu nombre es " + input("Dime tu
nombre: "))
>> Dime tu nombre: Federico
>> Tu nombre es Federico
```

1.4. - Proyecto del Día 1

Imagina esta situación: tu mejor amigo ha puesto una fábrica de cerveza y tiene todo listo. Su producto es fantástico, tiene cuerpo, buen sabor, buen color, el nivel justo de espuma... pero le falta una identidad. No se le ocurre qué nombre ponerle su cerveza para que tenga una identidad única y original. Entonces vienes tú y le dices "No te preocupes, yo voy a crear un programa que te va a hacer dos preguntas y luego te va a decir cuál es el nombre de tu cerveza". Así de simple.

Ya sé que en el mundo real no necesitaríamos desarrollar un software solo para hacer dos preguntas, pero hasta que aprendamos más funcionalidades los programas van a tener que mantenerse en el terreno de lo simple. Igualmente, si está recién comenzando, este va a ser todo un desafío.

Vas a crear un código en Python que le pida a tu amigo que responda dos preguntas que requieran una sola palabra cada una y que luego le muestre en pantalla esas palabras combinadas, para formar una marca creativa.

Puedes usar las preguntas que quieras. La idea es que el resultado sea original, creativo, y hasta cómico, y si quieres agregar dificultad al desafío, te sugiero que intentes que el nombre de la cerveza se imprima entre comillas. Recuerda que hay diferentes formas de que la función print muestre las comillas sin cortar el string, y que ingrese la impresión final en al menos dos líneas utilizando saltos de línea dentro del código.

Intenta hacerlo por tu cuenta y si se complica, no te preocupes, en la próxima elección lo vamos a resolver juntos.

Solución:

```
print("El nombre de tu cerveza\nes '" + input("Que ciudad te gustaria visitar?: ") + "' " +
input("Cual es tu color favorito?: ") + "'\nFelicitaciones!")
```

TEMA 2 - Programa un calculador de comisiones

2.1. - Tipos de datos

En Python tenemos varios tipos o estructuras de datos, que son fundamentales en programación ya que almacenan información, y nos permiten manipularla.

texto (str)	números (int y float)	booleanos
"Python"	Int 250	True
"750"	float 12.50	False

estructuras	mutable	ordenado (Tiene índice)	duplicados
listas [] <i>list</i>			
tuplas () <i>Entre paréntesis</i>			
sets { } <i>Entre llaves</i>			
diccionarios { } <i>dic</i> <i>key:valor</i>		*	: **

*: En Python 3.7+, existen consideraciones

** : *key* es única; *value* puede repetirse

2.2. - Variables

Las variables son espacios de memoria que almacenan valores o datos de distintos tipos, y (como su nombre indica) pueden variar. Se crean en el momento que se les asigna un valor, por lo cual en Python no requerimos declararlas previamente.

2.2.1. - Algunos ejemplos de uso

```
pais = "México"

nombre = input("Escribe tu nombre: ")
print("Tu nombre es " + nombre)
```



```
num1 = 55
num2 = 45
print(num1 + num2)
>> 100
```

2.3. - Nombres de variables

Existen convenciones y buenas prácticas asociadas al nombre de las variables creadas en Python. Las mismas tienen la intención de facilitar la interpretabilidad y mantenimiento del código creado.

2.3.1. - Reglas

1. Legible: nombre de la variable es relevante según su contenido
2. Unidad: no existen espacios (La práctica en Python es incorporar guiones bajos *ejemplo_variable*)
3. Hispanismos: omitir signos específicos del idioma español, como tildes o la letra ñ
4. Números: los nombres de las variables no deben empezar por números (aunque pueden contenerlos al final)
5. Signos/símbolos: no se deben incluir : " ' , < > / ? | \ () ! @ # \$ % ^ & * ~ - +
6. Palabras clave: no utilizamos palabras reservadas por Python.

2.4. - integers & floats

Existen dos tipos de datos numéricos básicos en Python: int y float. Como toda variable en Python, su tipo queda definido al asignarle un valor a una variable. La función `type()` nos permite obtener el tipo de valor almacenado en una variable.

2.4.1. - int

Int, o integer, es un número entero, positivo o negativo, sin decimales, de un largo indeterminado.

```
Num1 = 7
print(type(num1))
>> <class 'int'>
```

2.4.2. - float

Float, o "número de punto flotante" es un número que puede ser positivo o negativo, que a su vez contiene una o más posiciones decimales.

```
Num2 = 7.525587
print(type(num2))
>> <class 'float'>
```

2.5. - Conversiones

Python realiza conversiones **implícitas** de tipos de datos automáticamente para operar con valores numéricos. En otros casos, necesitaremos generar una conversión de manera **explícita**.

```

int(var)
>> <class 'int'>   —————>  """Convierte el dato en integer"""

float(var)
>> <class 'float'> —————>  """Convierte el dato en float"""

```

2.6. - Formatear cadenas

Para facilitar la concatenación de variables y texto en Python, contamos con dos herramientas que nos evitan manipular las variables, para incorporarlas directamente al texto:

- Función format: se encierra las posiciones de las variables entre corchetes { }, y a continuación del string llamamos a las variables con la función format

```
print("Mi auto es {} y de matrícula
{}".format(color_auto,matricula))
```

- Cadenas literales (f-strings): a partir de Python 3.8, podemos anticipar la concatenación de variables anteponiendo f al string

```
print(f"Mi auto es {color_auto} y de
matrícula {matricula}")
```

2.7. - Operadores matemáticos

Veamos cuáles son los operadores matemáticos básicos de Python, que utilizaremos para realizar cálculos:

Suma:	+	
Resta:	-	
Multiplicación:	*	
División:	/	
Cociente (división "al piso". Redondeo):	//	
Resto (módulo):	%	# Útil para detectar valores pares.
Potencia:	**	
Raíz cuadrada:	**0.5	# ¡es un caso especial de potencia!

2.8. - Redondeo

El redondeo facilita la interpretación de los valores calculados al limitar la cantidad de decimales que se muestran en pantalla. También, nos permite aproximar valores decimales al entero más próximo.

```
round(number,ndigits)
```

└_ valor a redondear cantidad de

└_ Cantidad de decimales (si se omite, el resultado es entero)

2.8.1. - algunos ejemplos de uso

```
print(round(100/3))  
>> 33
```

```
print(round(12/7,2))  
>> 1.71
```

2.9. - Proyecto del Día 2

La situación es esta: tú trabajas en una empresa donde los vendedores reciben comisiones del 13% por sus ventas totales, y tu jefe quiere que ayudes a los vendedores a calcular sus comisiones creando un programa que les pregunte su nombre y cuánto han vendido en este mes. Tu programa le va a responder con una frase que incluya su nombre y el monto que le corresponde por las comisiones.

Esto no es un programa complejo, pero es entendible que pueda complicarse cuando estás aprendiendo. Por más que lo que has aprendido hasta ahora es muy simple, ponerlo todo junto en un solo programa puede ser complejo, por lo que te doy un par de ayudas:

- Este programa debería comenzar preguntando cosas al usuario, por lo tanto, vas a necesitar input para poder recibir los ingresos del usuario y deberías usar variables para almacenar esos ingresos. Recuerda que los ingresos de usuarios se almacenan como strings. Por lo tanto, deberías convertir uno de esos ingresos en un float para poder hacer operaciones con él.
- ¿Y qué operaciones necesitas hacer? Bueno, calcular el 13% del número que haya ingresado el usuario. Es decir, que debes multiplicar ese número por 13 y luego dividirlo por 100. Recuerda almacenar ese resultado en una variable.
- Sería bueno que para imprimir en pantalla el resultado te asegures de que esa información no tenga más de dos decimales, para que sea fácil de leer, y luego organiza todo eso en un string al que debes dar formato. Recuerda que conocimos dos maneras de hacerlo y cualquiera de ellas es válida.

Solución:

```

nombre = input("Por favor, dime tu nombre: ")
ventas = int(input("Diga sus ventas totales del mes: "))

comision = round(ventas * 13 / 100,2)

print(f"Hola {nombre}, tus comisiones de este mes son de ${comision}")

```

TEMA 3 - Programa un analizador de texto

Los strings en python tienen incorporados más de 30 métodos que nos permiten manipularlos y analizarlos. Aquí veremos algunos de los más importantes.

3.1. - Index()

Cada carácter de los strings tienen una posición en el índice:



También podemos utilizar el índice reverso, que también empieza por cero pero luego todos los números están al revés con el signo menor.



Utilizamos el método `index()` para explorar strings, ya que permite hallar el índice de aparición de un carácter o cadena de caracteres dentro de un texto dado. Sintaxis:

`string.index(value, start, end)`

string: variable que almacena un string
value: carácter(es) | buscado(s)
start: las apariciones antes del índice **start** se ignoran
end: las apariciones luego del índice **end** se ignoran

Búsqueda en sentido inverso. Sintaxis:

`string.rindex(value, start, end)`

Devuelve el caracter en el índice i^*

`string[i]`

*: En Python, el índice en primera posición es el 0.

3.2. - Substrings

Podemos extraer porciones de texto utilizando las herramientas de manipulación de strings conocidas como slicing (rebanar). Sintaxis:

`string[start:stop:step]`

start: índice de inicio del sub-string (incluido)
 stop: índice del final del sub-string (no incluido)
 step: paso

Ejemplos:

```
H o l a _ M u n d o
0 1 2 3 4 5 6 7 8 9
```

```
saludo = H o l a _ M u n d o
print(saludo[2:6])
```

>> la_M

```
H o l a _ M u n d o
0 1 2 3 4 5 6 7 8 9
```

```
print(saludo[3::3])
```

>> auo

```
H o l a _ M u n d o
-9 -8 -7 -6 -5 -4 -3 -2 -1 0
```

```
print(saludo[::-1])
```

>> odnuM_aloH

3.3. - Métodos para Strings

3.3.1. - Métodos de análisis

`count()` : retorna el número de veces que se repite un conjunto de caracteres especificado.

```
"Hola mundo".count("Hola")  
>> 1
```

find() e **index()** retornan la ubicación (comenzando desde el cero) en la que se encuentra el argumento indicado. Difieren en que **index** lanza *ValueError* cuando el argumento no es encontrado, mientras **find** retorna

```
"Hola mundo".find("world")  
>> -1.
```

rfind() y **rindex()**. Para buscar un conjunto de caracteres pero desde el final.

```
"C:/python36/python.exe".rfind("/")  
>> 11
```

startswith() y **endswith()** indican si la cadena en cuestión comienza o termina con el conjunto de caracteres pasados como argumento, y retornan True o False en función de ello.

```
"Hola mundo".startswith("Hola")  
>> True
```

isdigit(): determina si todos los caracteres de la cadena son dígitos, o pueden formar números, incluidos aquellos correspondientes a lenguas orientales.

```
"abc123".isdigit()  
>> False
```

isnumeric(): determina si todos los caracteres de la cadena son números, incluye también caracteres de connotación numérica que no necesariamente son dígitos (por ejemplo, una fracción).

```
"1234".isnumeric()  
>> True
```

isdecimal(): determina si todos los caracteres de la cadena son decimales; esto es, formados por dígitos del 0 al 9.

```
"1234".isdecimal()  
>> True
```

isalnum(): determina si todos los caracteres de la cadena son alfanuméricos.

```
"abc123".isalnum()
>> True
```

isalpha(): determina si todos los caracteres de la cadena son alfabéticos.

```
"abc123".isalpha()
>> False
```

islower(): determina si todos los caracteres de la cadena son minúsculas.

```
"abcdef".islower()
>> True
```

isupper(): determina si todos los caracteres de la cadena son mayúsculas.

```
"ABCDEF".isupper()
>> True
```

isprintable(): determina si todos los caracteres de la cadena son imprimibles (es decir, no son caracteres especiales indicados por \...).

```
"Hola \t mundo!".isprintable()
>> False
```

isspace(): determina si todos los caracteres de la cadena son espacios.

```
"Hola mundo".isspace()
>> False
```

3.3.2. - Métodos de transformación

En realidad los strings son inmutables; por ende, todos los métodos a continuación no actúan sobre el objeto original sino que retornan uno nuevo.

capitalize() retorna la cadena con su primera letra en mayúscula.

```
"hola mundo".capitalize()
>> 'Hola mundo'
```

encode() codifica la cadena con el mapa de caracteres especificado y retorna una instancia del tipo bytes.

```
"Hola mundo".encode("utf-8")
>> b'Hola mundo'
```

replace() reemplaza una cadena por otra.

```
"Hola mundo".replace("mundo", "world")
>> 'Hola world'
```

lower() retorna una copia de la cadena con todas sus letras en minúsculas.

```
"Hola Mundo!".lower()
>> 'hola mundo!'
```

upper() retorna una copia de la cadena con todas sus letras en mayúsculas.

```
"Hola Mundo!".upper()
>> 'HOLA MUNDO!'
```

swapcase() cambia las mayúsculas por minúsculas y viceversa.

```
"Hola Mundo!".swapcase()
>> 'hOLA mUNDO!'
```

strip(), **lstrip()** y **rstrip()** remueven los espacios en blanco que preceden y/o suceden a la cadena.

```
" Hola mundo! ".strip()
>> 'Hola mundo!'
```

Los métodos **center()**, **ljust()** y **rjust()** alinean una cadena en el centro, la izquierda o la derecha. Un segundo argumento indica con qué carácter se deben llenar los espacios vacíos (por defecto un espacio en blanco).


```
"Hola".center(10, "*")
>> '***Hola***'
```

3.3.3. - Métodos de separación y unión

split() divide una cadena según un caracter separador (por defecto son espacios en blanco). Un segundo argumento en **split()** indica cuál es el máximo de divisiones que puede tener lugar (-1 por defecto para representar una cantidad ilimitada).

```
"Hola mundo!\nHello world!".split()
>> ['Hola', 'mundo!', 'Hello', 'world!']
```

splitlines() divide una cadena con cada aparición de un salto de línea.

```
"Hola mundo!\nHello world!".splitlines()
>> ['Hola mundo!', 'Hello world!']
```

partition() retorna una tupla de tres elementos: el bloque de caracteres anterior a la primera ocurrencia del separador, el separador mismo, y el bloque posterior.

```
"Hola mundo. Hello world!".partition(" ")
>> ('Hola', ' ', 'mundo. Hello world!')
```

rpartition() opera del mismo modo que el anterior, pero partiendo de derecha a izquierda.

```
"Hola mundo. Hello world!".rpartition(" ")
>> ('Hola mundo. Hello', ' ', 'world!')
```

join() debe ser llamado desde una cadena que actúa como separador para unir dentro de una misma cadena resultante los elementos de una lista.

```
", ".join(["C", "C++", "Python", "Java"])
>> 'C, C++, Python, Java'
```

3.4. - Strings: propiedades

Esto es lo que debes tener presente al trabajar con strings en Python:

- **Son inmutables:** una vez creados, no pueden modificarse sus partes, pero sí pueden reasignarse los valores de las variables a través de métodos de strings.

Las variables varían, pero los strings son inmutables.

```
mi_texto = "hola mundo"
mi_texto [0:3]
mi_texto [::-1]
mi_texto.replace("hola", "adios")
```

- **Concatenable:** es posible unir strings con el símbolo +
- **Multiplicable:** es posible concatenar repeticiones de un string con el símbolo *

```
mi_texto = "hola" + " mundo"
mi_texto = "hola" * 5
>>> "holaholaholaholahola"
```

- **Multilineales:** pueden escribirse en varias líneas al encerrarse entre triples comillas simples (""" """) o dobles (""" """)

```
mi_texto = "hola mundo"
mi_texto = """hola
mundo"""
```

- **Determinar su longitud:** a través de la función len(mi_string)

```
mi_texto = "hola mundo"
len(mi_texto)
```

- **Verificar su contenido:** a través de las palabras clave in y not in. El resultado de esta verificación es un booleano (True / False).

```
mi_texto = "hola mundo"
print("hola" in mi_texto)
>>> True
```

3.5. - Listas

Se pueden anidar listas. Se puede analizar y manipularlas, son mutables.

Una lista es de tipo <class 'list'>. Se pueden añadir tanto elementos str, int... todos juntos sin conflictos.

Igual que los strings, se puede seleccionar los elementos por su índice como un array:

```
mi_lista[0:]
```

Se pueden concatenar las listas.

```
lista_1 = ["C", "C++", "Python", "Java"]
lista_2 = ["PHP", "SQL", "Visual Basic"]
lista_3 = ["d", "a", "c", "b", "e"]
lista_4 = [5, 4, 7, 1, 9]
```

función append(): agrega un elemento a una lista en el lugar

```
lista_1.append("R")
print(lista_1)
>> ["C", "C++", "Python", "Java", "R"]
```

función pop(): elimina un elemento de la lista dado el índice, y devuelve el valor quitado

```
print(lista_1.pop(4))
>> "R"
```

función sort(): ordena los elementos de la lista en el lugar

```
lista_3.sort()
print(lista_3)
>> ['a', 'b', 'c', 'd', 'e']
```

función reverse(): invierte el orden de los elementos en el lugar. *

```
lista_4.reverse()
print(lista_4)
>> [9, 1, 7, 4, 5]
```

**reverse no es lo opuesto a sort*

3.6. - Diccionarios

Los diccionarios son estructuras de datos que almacenan información en pares **clave:valor**. Son especialmente útiles para guardar y recuperar información a partir de los nombres de sus claves (no utilizan índices).

mutable ordenado * admite duplicados :

```
mi_diccionario = {"curso": "Python TOTAL", "clase": "Diccionarios"}
```

Agregar nuevos datos, o modificarlos:

```
mi_diccionario["recursos"] = ["notas","ejercicios"]
```

Acceso a valores a través del nombre de las claves

```
print(mi_diccionario["recursos"][1])
>> "ejercicios"
```

Métodos para listar los nombres de:

- claves keys()
- valores values()
- pares clave:valor items()

**: A partir de Python 3.7+, los diccionarios son tipos de datos ordenados, en el sentido que dicho orden se mantiene según su orden de inserción para aumentar la eficiencia en el uso de la memoria.*

Cuidado, las claves no se pueden repetir.

3.7. - Tuples

Los tuples o tuplas, son estructuras de datos que almacenan múltiples elementos en una única variable. Se caracterizan por ser ordenadas e inmutables. Esta característica las hace más eficientes en memoria y a prueba de daños, se procesan más rápido. Si se intentan cambiar devolverá un error.

Mutable ordenado admite duplicados

```
mi_tuple = (1, "dos", [3.33, "cuatro"], (5.0, 6))
```

indexado (acceso a datos)

```
print(mi_tuple[3][0])
>> 5.0
```

casting (conversión de tipos de datos)

```
lista_1 = list(mi_tuple)
print(lista_1)
>> [1, "dos", [3.33, "cuatro"], (5.0, 6)]                    # ahora es una estructura mutable
```

unpacking (extracción de datos)

```
a, b, c, d = mi_tuple
print(c)
>> [3.33, "cuatro"]
```

3.8. - sets

Los sets son otro tipo de estructuras de datos. Se diferencian de listas, tuplas y diccionarios porque son una colección mutable de elementos inmutables, no ordenados y sin datos repetidos.

mutable	ordenado	admite duplicados
<code>mi_set_a = {1, 2, "tres"}</code>		<code>mi_set_b = {3, "tres"}</code>

3.8.1. - Métodos set

<code>mi_set_a = {1, 2, "tres"}</code>	<code>mi_set_b = {3, "tres"}</code>
--	-------------------------------------

add(item) agrega un elemento al set

```
mi_set_a.add(5)
print(mi_set_a)
>> {1, 2, "tres", 5}
```

clear() remueve todos los elementos de un set

```
mi_set_a.clear()
print(mi_set_a)
>> set()
```

copy() retorna una copia del set

```
mi_set_c = mi_set_a.copy()
print(mi_set_c)
>> {1, 2, "tres"}
```

difference(set) retorna el set formado por todos los elementos que únicamente existen en el set A

```
mi_set_c = mi_set_a.difference(mi_set_b)
print(mi_set_c)
>> {1, 2}
```



difference_update(set) remueve de A todos los elementos comunes a AyB

```
mi_set_a.difference_update(mi_set_b)
print(mi_set_a)
>> {1, 2}
```



discard(item) remueve un elemento del set

```
mi_set_a.discard("tres")
print(mi_set_a)
>> {1, 2}
```

intersection(set) retorna el set formado por todos los elementos que existen en A y B simultáneamente.

```
mi_set_c = mi_set_a.intersection(mi_set_b)
print(mi_set_c)
>> {'tres'}
```



intersection_update(set) mantiene únicamente los elementos comunes a A y B

```
mi_set_b.intersection_update(mi_set_a)
print(mi_set_b)
>> {"tres"}
```



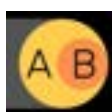
isdisjoint(set) devuelve True si A y B no tienen elementos en común

```
conjunto_disjunto = mi_set_a.isdisjoint(mi_set_b)
print(conjunto_disjunto)
>> False
```



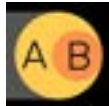
issubset(set) devuelve True si todos los elementos de B están presentes en A

```
es_subset = mi_set_b.issubset(mi_set_a)
print(es_subset)
>> False
```



issuperset(set) devuelve True si A contiene todos los elementos de B

```
es_superset = mi_set_a.issuperset(mi_set_b)
print(es_superset)
>> False
```



pop() elimina y retorna un elemento al azar del set

```
aleatorio = mi_set_a.pop()
print(aleatorio)
>> {2}
```

remove(item) elimina un item del set, y arroja error si no existe

```
mi_set_a.remove("tres")
print(mi_set_a)
>> {1, 2}
```

symmetric_difference(set) retorna todos los elementos de A y B, excepto aquellos que son comunes a los dos

```
mi_set_c = mi_set_b.symmetric_difference(mi_set_a)
print(mi_set_c)
>> {1, 2, 3}
```



symmetric_difference_update(set) elimina los elementos comunes a A y B, agregando los que no están presentes en ambos a la vez

```
mi_set_b.symmetric_difference_update(mi_set_a)
print(mi_set_b)
>> {1, 2, 3}
```



union(set) retorna un set resultado de combinar A y B (los datos duplicados se eliminan)

```
mi_set_c = mi_set_a.union(mi_set_b)
print(mi_set_c)
>> {1, 2, 3, 'tres'}
```



update(set) inserta en A los elementos de B

```
mi_set_a.update(mi_set_b)
print(mi_set_a)
>> {1, 2, 3, 'tres'}
```



3.9. - booleanos

Los booleanos son tipos de datos binarios (True/False), que surgen de operaciones lógicas, o pueden declararse explícitamente.

3.9.1. - operadores lógicos

==	igual a
!=	no igual a
>	mayor que
<	menos que
>=	mayor o igual a
<=	menor o igual a
and	y (True si dos declaraciones son True)
or	o (True si al menos una declaración es True)
not	no (invierte el valor del booleano)

3.10. - Proyecto del Día 3

Analizador de texto.

La consigna es la siguiente: vas a crear un programa que primero le pida al usuario que ingrese un texto. Puede ser un texto cualquiera: un artículo entero, un párrafo, una frase, un poema, lo que quiera. Luego, el programa le va a pedir al usuario que también ingrese tres letras a su elección y a partir de ese momento nuestro código va a procesar esa información para hacer cinco tipos de análisis y devolverle al usuario la siguiente información:

1. Primero: ¿cuántas veces aparece cada una de las letras que eligió? Para lograr esto, te recomiendo almacenar esas letras en una lista y luego usar algún método propio de string que nos permita contar cuantas veces aparece un sub string dentro del string. Algo que debes tener en cuenta es que al buscar las letras pueden haber mayúsculas y minúsculas y esto va a afectar el resultado. Lo que deberías hacer para asegurarte de que se encuentren absolutamente todas las letras es pasar, tanto el texto original como las letras a buscar, a minúsculas.
2. Segundo: le vas a decir al usuario cuántas palabras hay a lo largo de todo el texto. Y para lograr esta parte, recuerda que hay un método de string que permite transformarlo en una lista y que luego hay una función que permite averiguar el largo de una lista.
3. Tercero: nos va a informar cuál es la primera letra del texto y cuál es la última. Aquí claramente echaremos mano de la indexación.
4. Cuarto: el sistema nos va a mostrar cómo quedaría el texto si invirtiéramos el orden de las palabras. ¿Acaso hay algún método que permita invertir el orden de una lista, y otro que permita unir esos elementos con espacios intermedios? Piénsalo.
5. Y por último: el sistema nos va a decir si la palabra “Python” se encuentra dentro del texto. Esta parte puede ser algo complicada de imaginársela, pero te voy a dar una pista: puedes usar booleanos para hacer tu averiguación y un diccionario para encontrar la manera de expresarle al usuario tu respuesta.

Solución:

```
texto = input("Ingresa un texto a elección: ")
letras = []

texto = texto.lower()

letras.append(input("Ingresa la primera letra: ").lower())
letras.append(input("Ingresa la segunda letra: ").lower())
letras.append(input("Ingresa la tercera letra: ").lower())

print("\n")
print("CANTIDAD DE LETRAS")
cantidad_letras1 = texto.count(letras[0])
```

```
cantidad_letras2 = texto.count(letras[1])
cantidad_letras3 = texto.count(letras[2])

print(f"Hemos encontrado la letra '{letras[0]}' repetida {cantidad_letras1} veces")
print(f"Hemos encontrado la letra '{letras[1]}' repetida {cantidad_letras2} veces")
print(f"Hemos encontrado la letra '{letras[2]}' repetida {cantidad_letras3} veces")

print("\n")
print("CANTIDAD DE PALABRAS")
palabras = texto.split()
print(f"Hemos encontrado {len(palabras)} palabras en tu texto")

print("\n")
print("LETRAS DE INICIO Y DE FIN")
letra_inicio = texto[0]
letra_final = texto[-1]
print(f"La letra inicial es '{letra_inicio}' y la letra final es '{letra_final}'")

print("\n")
print("TEXTO INVERTIDO")
palabras.reverse()
texto_invertido = ' '.join(palabras)

print(f"Si ordenamos tu texto al revés va a decir: '{texto_invertido}'")

print("\n")
print("BUSCANDO LA PALABRA PYTHON")
buscar_python = 'python' in texto
dic = {True:"sí", False:"no"}
print(f"La palabra 'Python' {dic[buscar_python]} se encuentra en el texto")
```

TEMA 4 - Programa el juego "adivina el número"

4.1. - Operadores de comparación

Como su nombre lo indica, sirven para comparar dos o más valores. El resultado de esta comparación es un booleano (True/False)

==	igual a
!=	diferente a
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que

Si la comparación resulta verdadera, devuelve el resultado *True*

Si dicha comparación es falsa, el resultado es *False*

```
mi_bool = 5 >= 6
print(mi_bool)
>> False

mi_bool = 5 != 6
print(mi_bool)
>> True

mi_bool = 10 == 2*5
print(mi_bool)
>> True

mi_bool = 5 < 6
print(mi_bool)
>> True
```

4.2. - Operadores lógicos

Estos operadores permiten tomar decisiones basadas en múltiples condiciones.

```
a = 6 > 5          b = 30 == 15*3
```

and devuelve True si todas las condiciones son verdaderas

```
mi_bool = a and b
print(mi_bool)
>> False
```

or devuelve True si al menos una condición es verdadera

```
mi_bool = a or b
print(mi_bool)
>> True
```

not devuelve True si el valor del booleano es False, y False si es True

```
mi_bool = not a
print(mi_bool)
>> False
```

4.3. - Control de flujo

El control de flujo determina el orden en que el código de un programa se va ejecutando. En Python, el flujo está controlado por estructuras condicionales, loops y funciones.

4.3.1. - Estructuras condicionales (if)

Expresión de resultado booleano (True/False)

la indentación es obligatoria en Python

Los dos puntos (:) dan paso al código que se ejecuta si expresión = True

```
if expresión:
    código a ejecutarse
elif expresión:
    código a ejecutarse
elif expresión:
    código a ejecutarse
...
else:
    código a ejecutarse
```

else & elif son opcionales y pueden incluirse varias cláusulas elif

4.4. - loops while

Si bien los loops while son otro tipo de bucles, resultan más parecidos a los condicionales if que a los loops for. Podemos pensar a los loops while como una estructura condicional que se ejecuta en repetición, hasta que se convierte en falsa.

```
while condición:
    expresión
else:
    expresión
```

Se compone de la *Estructura condicional* seguida de dos puntos (:)

La indentación es obligatoria en Python.

Este código se ejecutará cuando la condición se convierta en False

4.4.1. - instrucciones especiales

Si el código llega a una instrucción break, se produce la salida del bucle.

La instrucción continue interrumpe la iteración actual dentro del bucle, llevando al programa a la parte superior del bucle.

La instrucción pass no altera el programa: ocupa un lugar donde se espera una declaración, pero no se desea realizar una acción.

4.5. - range()

La función range() devuelve una secuencia de números dados 3 parámetros. Se utiliza fundamentalmente para controlar el número de ejecuciones de un loop o para crear rápidamente una serie de valores.

valor_inicio	número a partir del cual inicia el rango (incluido)
valor_final	número antes del cual el rango finaliza (no incluido)
paso	diferencia entre cada valor consecutivo de la secuencia

```
range(valor_inicio, valor_final, paso)
print(list(range(1,13,3)))
>> [1,4,7,10]
```

El único parámetro obligatorio es valor_final. Los valores predeterminados para valor_inicio y paso son 0 y 1 respectivamente.

4.6. - enumerate()

La función `enumerate()` nos facilita llevar la cuenta de las iteraciones, a través de un contador de índices de un iterable, que se puede utilizar de manera directa en un loop, o convertirse en una lista de tuplas con el método `list()`.

iterable Cualquier objeto que pueda ser iterado

inicio Valor [int] de inicio del índice (por defecto iniciado en 0)

```
enumerate(iterable, inicio)
```

```
print(list(enumerate("Hola")))
>> [(0, 'H'), (1, 'o'), (2, 'l'), (3, 'a')]
```

```
for indice, numero in enumerate([5.55, 6, 7.50]):
    print(indice, numero)
>> 0 5.55
>> 1 6
>> 2 7.5
```

4.7. - zip()

La función `zip()` crea un iterador formado por los elementos agrupados del mismo índice provenientes de dos o más iterables. Zip deriva de zipper (cremallera o cierre), de modo que es una analogía muy útil para recordar.

La función se detiene al cuando se agota el iterable con menor cantidad de elementos.

```
letras = ['w', 'x', 'c']
numeros = [50, 65, 90, 110, 135]
for letra, num in zip(letras, numeros):
    print(f'Letra: {letra}, y Número: {num}')
>> Letra: w, y Número: 50
>> Letra: x, y Número: 65
>> Letra: c, y Número: 90
```

4.8. - min() & max()

La función min() retorna el item con el valor más bajo dentro de un iterable. La función max() funciona del mismo modo, devolviendo el valor más alto del iterable. Si el iterable contiene strings, la comparación se realiza alfabéticamente.

```
ciudades_habitantes = {"Tijuana":1810645, "León":1579803}
lista_valores = [5**5, 12**2, 3050, 475*2]
print(min(ciudades_habitantes.keys()))
>> León

print(max(ciudades_habitantes.values()))
>> 1810645

print(max(lista_valores))
>> 3125
```

4.9. - random

Python nos facilita un módulo (un conjunto de funciones disponibles para su uso) que nos permite generar selecciones pseudo-aleatorias* entre valores o secuencias.

Nombre del módulo

```
from random import *
```

* = Todos los métodos

También pueden importarse de manera independiente aquellos a utilizar.

randint(min, max): devuelve un integer entre dos valores dados (ambos límites incluidos)

uniform(min, max): devuelve un float entre un valor mínimo y uno máximo

random(sin parámetros): devuelve un float entre 0 y 1

choice(secuencia): devuelve un elemento al azar de una secuencia de valores (listas, tuples, rangos, etc.)

shuffle(secuencia): toma una secuencia de valores mutable (como una lista), y la retorna cambiando el orden de sus elementos aleatoriamente.

* La mecánica en cómo se generan dichos valores aleatorios viene en realidad predefinida en "semillas". Si bien sirve para todos los usos habituales, no debe emplearse con fines de seguridad o criptográficos, ya que son vulnerables.

4.10. - Comprensión de listas

La comprensión de listas es una manera dinámica de construir una lista. Ofrece una sintaxis más breve en la creación de una nueva lista basada en valores disponibles en otra secuencia. Vale la pena mencionar que la brevedad se logra a costo de una menor interpretabilidad.

```
nueva_lista= [expresion for item in iterable if condicion == True]
```

expresión	fórmula matemática
Item	cada elemento del iterable
iterable	tuplas, sets, otras listas...
condicion == True	operación lógica

Caso especial con else:

```
nueva_lista= [expresion if condicion == True else otra_expresion for item in iterable]
```

Ejemplo:

```
nueva_lista = [num**2 for num in range(10) if num < 5]
print(nueva_lista)
>> [0, 1, 4, 9, 16]
```

4.11. - match

En Python 3.10, se incorpora la coincidencia de patrones estructurales mediante las declaraciones `match` y `case`. Esto permite asociar acciones específicas basadas en las formas o patrones de tipos de datos complejos.

match objeto:

```
case <patron_1>:
    <accion_1>
case <patron_2>:
    <accion_2>
case <patron_3>:
    <accion_3>
case _:
    <accion_comodin>
```

El caracter `_` es un comodín que actúa como coincidencia si la misma no se produce en los casos anteriores.

Es posible detectar y deconstruir diferentes estructuras de datos: esto quiere decir que los patrones no son únicamente valores literales (strings o números), sino también estructuras de datos, sobre los cuales se buscan coincidencias de construcción.

4.12. - Proyecto del Día 4

La consigna es esta: el programa le va a preguntar al usuario su nombre, y luego le va a decir algo así como “Bueno, Juan, he pensado un número entre 1 y 100, y tienes solo ocho intentos para adivinar cuál crees que es el número”. Entonces, en cada intento el jugador dirá un número y el programa puede responder cuatro cosas distintas:

- Si el número que dijo el usuario es menor a 1 o superior a 100, le va a decir que ha elegido un número que no está permitido.
- Si el número que ha elegido el usuario es menor al que ha pensado el programa, le va a decir que su respuesta es incorrecta y que ha elegido un número menor al número secreto.
- Si el usuario eligió un número mayor al número secreto, también se lo hará saber de la misma manera.
- Y si el usuario acertó el número secreto, se le va a informar que ha ganado y cuántos intentos le ha tomado.

Si el usuario no ha acertado en este primer intento, se le va a volver a pedir que elija otro número. Y así hasta que gane o hasta que se agoten los ocho intentos.

Solución:

```
"""
```

```
Programa día 3 - Juego "Adivina el número"
```

```
"""
```

```
# Importamos módulos
```

```
from random import randint
```

```
import itertools
```

```
import threading
```

```
import time
```

```
import sys
```

```
import os
```

```
# Funciones
```

```
## Función limpiar consola
```

```
def clearConsole():
```

```
    command = 'clear'
```

```
    if os.name in ('nt', 'dos'): # If Machine is running on Windows, use cls
```

```
        command = 'cls'
```

```
    os.system(command)
```

```
## Función de la animación para 'pensar'
```

```
def animate():
```

```
    for c in itertools.cycle(['|', '/', '-', '\\']):
```

```

    if DONE:
        break
    sys.stdout.write("\rPensando un número ' + c)
    sys.stdout.flush()
    time.sleep(0.1)
    sys.stdout.write("\rYa tengo el número!          ')
    sys.stdout.write("\nA ver si aciertas, dime un número: ')

## Función validar input es un número
def validar(dato, tipos):
    for tipo in tipos:
        try:
            return tipo(dato)
        except ValueError:
            pass
    return None

# Limpiar la terminal
clearConsole()

# Mensaje bienvenida
print("\n#####")
print("JUEGO - Adivina el número")
print("#####\n")

# Pregunta nombre, saluda y explica el juego (Con opción a salir)
nombre = input("¿Cuál es tu nombre? ")
print(f"\nOk {nombre}, voy a pensar un número del 1 al 100.")
print("Tienes 8 intentos para adivinarlo\n")

# Filigrana como que piensa la máquina
DONE = False
t = threading.Thread(target=animate)
t.start()

## El tiempo de ejecución
time.sleep(3)
DONE = True

# Guardando el número secreto
n_secreto = randint(1, 100)

# Bucle de intentos y control de flujo
for intentos in range(8):

    # Calcular intentos restantes
    lista_intentos = list(range(8))

```

```
lista_intentos.reverse()
intentos_restantes = lista_intentos.pop(intentos)

# Pregunta posible número y valida que es de type número
while True:
    n_posible = input("Dime un número: ")
    x = validar(n_posible, (int, float, complex))
    if x is None:
        print("Error: El dato introducido no es un número\n")
    else:
        break

# Para comparar el número se necesita que sea integer
n_posible = int(n_posible)

# Posibles casuísticas
## Si el número no está en el rango especificado
if (n_posible < 1) or (n_posible > 100):
    print("\nEse número no está entre el 1 y el 100")
    print(f"Desperdiciaste un turno, te quedan {intentos_restantes} intentos")

## Si el número es menor
elif n_posible > n_secreto:
    print(f"\nLo siento, has fallado. Te quedan {intentos_restantes} intentos.")
    print("\n\tPista --> El número es menor.\n")

## Si el número es mayor
elif n_posible < n_secreto:
    print(f"\nLo siento, has fallado. Te quedan {intentos_restantes} intentos.")
    print("\n\tPista --> El número es mayor.\n")

## Si se acierta el número
elif n_posible == n_secreto:
    print("\n¡Ole! ¡¡¡¡HAS ACERTADO!!!!")
    intentos += 1
    print(f"Lo conseguiste en {intentos} intentos")
    break

## Si se agotan los intentos
if intentos == 7:
    print("\n¡Vaya! Has gastado los 8 intentos\nOtra vez será...")

print("\n¡Espero que te hayas divertido!")
```

TEMA 5 - Programa el juego "El ahorcado"

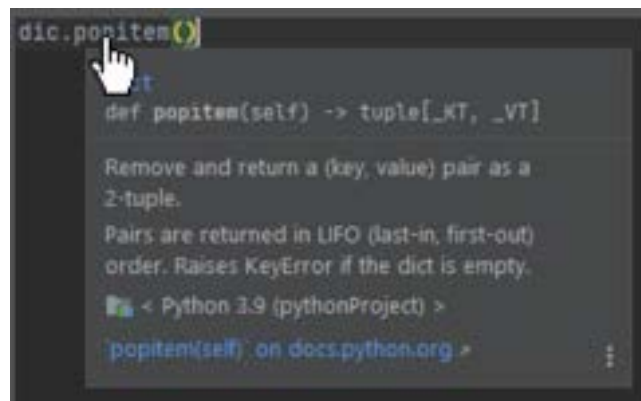
5.1. - Documentación

La documentación es nuestra biblioteca de consulta en Python. Al escribir código, es de uso permanente para solucionar dudas relacionadas al funcionamiento de métodos y los argumentos que reciben.

Si utilizas PyCharm:

Sostén el cursor sobre el nombre del método del que deseas obtener información. Se desplegará una ventana flotante.

Esto funcionará en otros IDEs del mismo modo.



También, debes mantener cerca la documentación oficial de Python (o Biblioteca Estándar de Python), que contiene toda la información necesaria:

<https://docs.python.org/es/3.9/library/index.html>

No dejes de buscar en Google tus dudas, para hallar una explicación que se ajuste a ti.

5.2. - Funciones

Una función es un bloque de código que solamente se ejecuta cuando es llamada. Puede recibir información (en forma de parámetros), y devolver datos una vez procesados como resultado.

Una función es definida mediante la palabra clave **def**

```
def mi_funcion(argumento):
```

Los argumentos contienen información que la función utiliza o transforma para devolver un resultado

```
mi_funcion(mi_argumento)
```

Para llamar a una función, basta utilizar su nombre, entregando los argumentos que la misma requiere entre paréntesis.

5.3. - return

Para que una función pueda devolver un valor (y que pueda ser almacenado en una variable, por ejemplo), utilizamos la declaración return.

```
def mi_funcion():  
    return [algo]
```

La declaración return provoca la salida de la función. Cualquier código que se encuentre después en el bloque de la función, no se ejecutará

```
resultado = mi_funcion()
```

La variable resultado almacenará el valor devuelto por la función mi_funcion()

5.4. - Funciones dinámicas

La integración de diferentes herramientas de control de flujo, nos permite crear funciones dinámicas y flexibles. Si debemos utilizarlas varias veces, lograremos un programa más limpio y sencillo de mantener, evitando repeticiones de código.

- Funciones
- Loops (for/while)
- Estructuras condicionales
- Palabras clave (return, break, continue, pass)

```
def mi_funcion(argumento):  
    for item in ...  
        if a == b ...  
            ...  
        else:  
            return ...  
    return ...
```

5.5. - Interacción entre funciones

Las salidas de una determinada función pueden convertirse en entradas de otras funciones. De esa manera, cada función realiza una tarea definida, y el programa se construye a partir de la interacción entre funciones.

```
def funcion_1():
    |...
    |return a

def funcion_2(a):
    |...
    |return b

def funcion_3(b):
    |...
    |return c

def funcion_4(a,c):
    |...
    |return d
```

5.6. - *args

En aquellos casos en los que no se conoce de antemano el número exacto de argumentos que se deben pasar a una función, se debe utilizar la sintaxis `*args` para referirse a todos los argumentos adicionales luego de los obligatorios.

El nombre `*args` no es mandatorio pero si es sugerido como buena práctica.

Cualquier nombre iniciado en `*` referirá a estos argumentos de cantidad variable.

La función recibirá los argumentos indefinidos `*args` en forma de tupla, a los que se podrá acceder o iterar de las formas habituales dentro del bloque de código de la función.

```
def mi_funcion(arg_1, arg_2, *args):
```

```
    mi_funcion("ejemplo", 25, 40, 75, 10):
```

```
arg_1 → "ejemplo"
```

```
arg_2 → 25
```

```
args = (40, 75, 10)
```

5.7. - **kwargs

Los argumentos con palabras clave (keyword arguments, abreviado kwargs), sirven para identificar el argumento por su nombre, independientemente de la posición en la que se pasen a su función. Si no se conoce de antemano su cantidad, se utiliza el parámetro **kwargs que los agrupa en un diccionario.

Al igual que para *args, el nombre **kwargs no es mandatorio pero si es sugerido como buena práctica. Cualquier nombre iniciado en ** referirá a estos argumentos de cantidad variable.

```
def atributos_persona(**kwargs):
```

```
    atributos_persona(ojos="azules", pelo="corto")
```

```
kwargs = {'ojos': 'azules', 'pelo': 'rubio'}
```

5.8. - Ejercicios

5.8.1. - Ejercicio 1

Crea una función llamada devolver_distintos() que reciba 3 integers como parámetros.

Si la suma de los 3 numeros es mayor a 15, va a devolver el número mayor.

Si la suma de los 3 numeros es menor a 10, va a devolver el número menor.

Si la suma de los 3 números es un valor entre 10 y 15 (incluidos) va a devolver el número de valor intermedio.

Solución:

```
def devolver_distintos(num1, num2, num3):
```

```
    suma = num1 + num2 + num3
```

```
    lista = [num1, num2, num3]
```

```
    if suma > 15:
```

```
        return max(lista)
```

```
    elif suma < 10:
```

```
        return min(lista)
```

```
    else:
```

```
        lista.sort()
```

```
        return lista[1]
```

```
print(devolver_distintos(6, 5, 3))
```

5.8.2. - Ejercicio 2

Escribe una función (puedes ponerle cualquier nombre que quieras) que reciba cualquier palabra como parámetro, y que devuelva todas sus letras únicas (sin repetir) pero en orden alfabético.

Por ejemplo si al invocar esta función pasamos la palabra "entretenido", debería devolver ['d', 'e', 'i', 'n', 'o', 'r', 't']

Solución:

```
def cualquier(palabra):
    mi_set = set()
    for letra in palabra:
        mi_set.add(letra)
    lista = list(mi_set)
    lista.sort()
    return lista
print("\n")
print(cualquier('entretenido'))
```


5.8.3. - Ejercicio 3

Escribe una función que requiera una cantidad indefinida de argumentos. Lo que hará esta función es devolver True si en algún momento se ha ingresado al número cero repetido dos veces consecutivas.

Por ejemplo:

```
(5,6,1,0,0,9,3,5) >>> True
```

```
(6,0,5,1,0,3,0,1) >>> False
```

Solución:

```
def me_gustan_los_ceros(*noceros):
    contador = 0
    for num in noceros:
        if num == 0:
            contador += 1
            if contador == 2:
                contador = 0
                return True
        elif num != 0:
            contador = 0
        else:
            pass
    return False

print("\n")
print(me_gustan_los_ceros(5, 6, 1, 0, 0, 9, 3, 5))
print(me_gustan_los_ceros(6, 0, 5, 1, 0, 3, 0, 1))
```

Solución curso:

```
def ceros_vecinos(*args):
    contador = 0
    for num in args:
        if contador + 1 == len(args):
```

```
        return False
    elif args[contador] == 0 and args[contador + 1] == 0:
        return True
    else:
        contador += 1
    return False
print("\n")
print(ceros_vecinos(5, 6, 1, 0, 0, 9, 3, 5))
print(ceros_vecinos(6, 0, 5, 1, 0, 3, 0, 1))
```

5.8.4. - Ejercicio 4

Escribe una función llamada contar_primos() que requiera un solo argumento numérico.

Esta función va a mostrar en pantalla todos los números primos existentes en el rango que va desde cero hasta ese número incluido, y va a devolver la cantidad de números primos que encontró.

Aclaración, por convención el 0 y el 1 no se consideran primos.

Solución:

```
def primo(numero):
    if numero == 0 or numero == 1 or numero == 4:
        return False
    for x in range(2, int(numero/2)):
        if numero % x == 0:
            return False
    return True

def primos(rango_max):
    rango_max += 1
    lista_num = [n for n in range(1, rango_max)]
    primos = []
    for numero in lista_num:
        es_primo = primo(numero)
        if es_primo:
            primos.append(numero)
    contar = len(primos)
    rango_max -= 1
    print(f"\nEntre 0 y el {rango_max} hay {contar} números primos.\n")
    print(*primos, sep=' ')

primos(1000)
```

Solución curso:

```
def contar_primos(numero):
    primos = [2]
```

```
iteracion = 3

if numero < 2:
    return 0

while iteracion <= numero:
    for n in range(3, iteracion, 2):
        if iteracion % n == 0:
            iteracion += 2
            break
    else:
        primos.append(iteracion)
        iteracion += 2

print(primos)
return len(primos)

print("\n")
print(contar_primos(1000))
```

5.9. - Proyecto del Día 5

Hoy vas a programar El Ahorcado. El programa va a elegir una palabra secreta y le va a mostrar al jugador solamente una serie de guiones que representa la cantidad de letras que tiene la palabra. El jugador en cada turno deberá elegir una letra y si la letra se encuentra en la palabra oculta, el sistema le va a mostrar en qué lugares se encuentra. Pero si el jugador dice una letra que no se encuentra en la palabra oculta, pierde una vida.

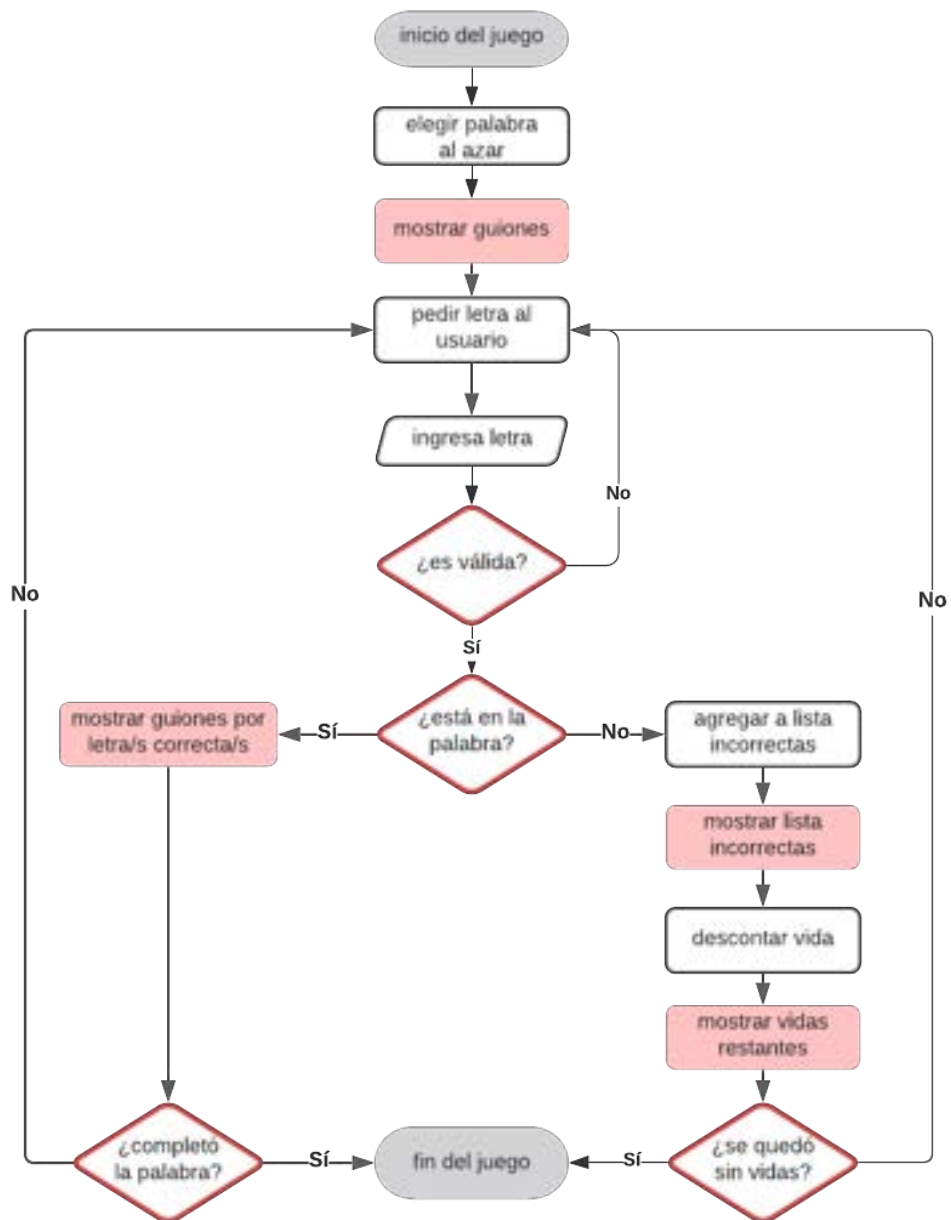
En el juego real del ahorcado, cada vez que perdemos una vida, se va completando el dibujo del ahorcado miembro por miembro. Pero en nuestro caso, como aún no tenemos elementos gráficos, simplemente le vamos a decir que tiene seis vidas y se las iremos descontando de una en una, cada vez que el jugador elija una letra incorrecta.

Si se agotan las vidas antes de adivinar la palabra, el jugador pierde. Pero si adivina la palabra completa antes de perder todas las vidas, el jugador gana.

Parece sencillo, pero ¿cómo diseñamos todo este código? Bueno, aquí van algunas ayudas:

- Primero vas a crear un código que importe el método choice, ya que lo vas a necesitar para que el sistema pueda elegir una palabra al azar dentro de una lista de palabras que también vas a crear al comienzo.
- Luego de eso, vas a crear tantas funciones como creas necesarias para que el programa haga cosas como pedirle al usuario que elija una letra, para corroborar si lo que el usuario ha ingresado es una letra válida, para chequear si la letra ingresada se encuentra en la palabra o no, para verificar si ha ganado o no, etc.
- Recuerda escribir primero las funciones y luego el código que las implementa ordenadamente.

Creo que este es un proyecto especial y de verdad quiero que sepas que no espero que lo puedas resolver sin ayuda. Las funciones parecen simples, pero cuando empezamos a poner funciones todas juntas en un código real, es difícil seguir mentalmente la sucesión del código porque se vuelve mucho menos lineal que antes. Lo importante es que lo intentes, que tu cabeza se zambulla en el problema y luego veremos cómo llegamos a la solución.



TEMA 6 - Programa un recetario

6.1. - Abrir y leer archivos

La manipulación de archivos desde Python se engloba bajo las funciones de E/S (entrada/salida) o I/O (en inglés: input/output). Comenzaremos explorando las funciones utilizadas para abrir, leer y cerrar archivos:

open(archivo, modo): abre un archivo, y devuelve un objeto de tipo archivo sobre el que pueden aplicarse métodos

read(bytes): devuelve un número especificado de bytes del archivo. De manera predeterminada (sin indicar un valor en el argumento bytes), devolverá el archivo completo (equivalente a escribir -1).

readline(bytes): devuelve una línea del archivo, limitada por el número indicado en el parámetro tamaño (en bytes).

readlines(bytes): devuelve una lista que contiene cada una de las líneas del archivo como item de dicha lista. Si el tamaño excede lo indicado en el parámetro bytes, no se devolverán líneas adicionales.

close(): cierra el archivo abierto, tal que no puede ser leído o escrito luego de cerrado. Es una buena práctica utilizar este método si ya no será necesario realizar acciones sobre un archivo.

Por lo general, un byte equivale a un caracter.

6.2. - Crear y escribir archivos

Para escribir en un archivo desde Python, deberemos elegir con cuidado el parámetro "modo de apertura".

```
open(arhivo, modo)
```

Parámetros de modo de apertura:

- **"r"** - Read (Lectura) - Predeterminado. Permite leer pero no escribir, y arroja un error si el archivo no existe.
- **"a"** - Append (Añadir) - Abre el archivo para añadir líneas a continuación de la última que ya exista en el mismo. Crea un archivo en caso de que el mismo no exista.
- **"w"** - Write (Escritura) - Abre o crea un archivo (si no existe previamente) en modo de escritura, lo que significa que cualquier contenido previo se sobrescribirá.
- **"x"** - Create (Creación) - Crea un archivo, y arroja un error si el mismo ya existe en el directorio.

El método **write**() escribe un texto especificado en el argumento sobre el archivo.

writelines(lista) recibe el texto a ser escrito en forma de lista.

6.3. - Directorios

Trabajar sobre archivos que se encuentran en directorios diferentes al de nuestro código requiere del soporte del módulo OS, que contiene una serie de funciones para interactuar con el sistema operativo.

```
import os
```

os.getcwd(): obtiene y devuelve el directorio de trabajo actual. Será el mismo en el que corre el programa si no se ha modificado.

os.chdir(ruta): cambia el directorio de trabajo a la ruta especificada

os.makedirs(ruta): crea una carpeta, así como todas las carpetas intermedias necesarias de acuerdo a la ruta especificada.

os.path.basename(ruta): dada una ruta, obtiene el nombre del archivo (nombre de base)

os.path.dirname(ruta): dada una ruta, obtiene el directorio (carpeta) que almacena el archivo

os.path.split(ruta): devuelve una tupla que contiene dos elementos: el directorio, y el nombre de base del archivo.

rmdir(ruta): elimina el directorio indicado en la ruta.

En Windows, es necesario indicar las rutas con dobles barras invertidas

(\\) para que sean correctamente interpretadas por Python.

6.4. - pathlib

El módulo pathlib, disponible desde Python 3.4, permite crear objetos Path, generando rutas que pueden ser interpretadas por diferentes sistemas operativos y cuentan con una serie de propiedades útiles.

```
from pathlib import Path
```

```
ruta = Path("C:/Users/Usuario/Desktop")
```

A partir de una semántica sencilla, devuelve una ruta que el sistema puede comprender. Por ejemplo, en Windows, devolverá: C:\Users\Usuario\Desktop y en Mac: C:/Users/Usuario/Desktop

6.4.1. - Navegación

```
ruta = Path("C:/Users/Usuario/Desktop") / "archivo.txt"
```

Es posible concatenar objetos Path y strings con el delimitador "/" para construir rutas completas.

6.4.2. - Algunos métodos y propiedades sobre objetos Path

read_text(): lee el contenido del archivo sin necesidad de abrirlo y cerrarlo

name: devuelve el nombre y extensión del archivo

suffix: devuelve la extensión del archivo (sufijo)

stem: devuelve el nombre del archivo sin su extensión (sufijo)

exists(): verifica si el directorio o archivo al que referencia el objeto Path existe y devuelve un booleano de acuerdo al resultado (True/False)

6.5. - Path

La clase Path permite representar rutas de archivos en el sistema de archivos de nuestro sistema operativo. Se destaca por su legibilidad frente a alternativas semejantes.

`base = Path.home()` → Devuelve un objeto Path representando el directorio base del usuario

`ruta = Path(base, "Europa", "Barcelona", "SagradaFamilia.txt")`

Se aceptan strings y otros objetos Path

`ruta2 = ruta.with_name("LaPedrera.txt")`

Devuelve un nuevo objeto Path cambiando únicamente el nombre de archivo

Cada invocación de la propiedad `parent` devuelve la ruta de jerarquía inmediata superior

`C:\Users\...`

|_ `Europa`

|_ `Barcelona`

|_ `SagradaFamilia.txt`

|_ `LaPedrera.txt`

```
    continente = ruta.parent.parent
```

```
    print(continente)
```

```
>> C:\Users\...\Europa
```

Devuelve el conjunto de archivos que coinciden con el "patrón"

```
Path(ruta).glob("*.txt")
```

```
Path(ruta).glob("**/*.txt")
```

Búsqueda recursiva en subdirectorios

6.6. - Limpiar la consola

Para controlar la información mostrada al usuario en consola podemos limpiarla, eliminando los diferentes mensajes que han aparecido conforme se va ejecutando el programa.

```
from os import system
```

En Unix/Linux/macOS:

```
system("clear")
```

En DOS/Windows:

```
system("cls")
```

6.7. - Archivos + funciones

Recordatorio: puedes crear funciones para que ejecuten código cada vez que sean invocadas, evitando repeticiones y facilitando su lectura. Esto aplica para todo Python, y desde luego también cuando manipulamos archivos.

¿Qué hace la función? ¿Qué información necesita recibir?

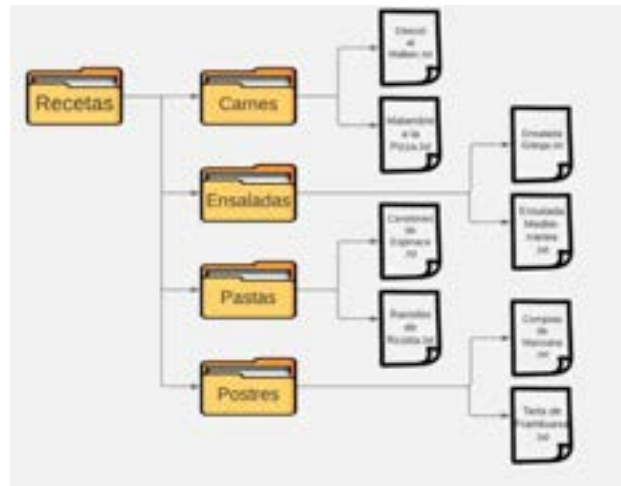
```
def manipular_archivo(ruta, nombre_archivo):
    # Código que convierte las entradas en salidas
    return
```

¿Cuál debe ser su salida?

6.8. - Proyecto del Día 6

Vas a crear un administrador de recetas. Básicamente esto es un programa a través del cual un usuario puede leer, crear y eliminar recetas que se encuentren en una base de datos.

Entonces, antes de comenzar, es necesario que crees en tu ordenador un directorio en la carpeta base de tu ordenador, con una carpeta llamada Recetas, que contiene cuatro carpetas y cada una de ellas contiene dos archivos de texto. Dentro de los archivos puedes escribir lo que quieras, puede ser la receta en sí misma o no, pero eso no es importante para este ejercicio. Lo importante es que escribas algo para poder leerlas cuando haga falta o, si prefieres, también puedes directamente descargar y descomprimir el archivo adjunto a esta elección y ubicarlo en tu directorio raíz si no tienes ganas de crearlo tú mismo.



Aquí viene la consigna: tu código le va a dar primero la bienvenida al usuario, le va a informar la ruta de acceso al directorio donde se encuentra nuestra carpeta de recetas, le va a informar cuántas recetas hay en total dentro de esa carpeta, y luego le va a pedir que elija una de estas opciones que tenemos aquí:

1. La opción 1 le va a preguntar qué categoría elige (carnes, ensaladas, etc.), y una vez que el usuario elija una, le va a preguntar qué **receta** quiere **leer**, y mostrar su contenido.
2. En la opción 2 también se le va a hacer elegir una categoría, pero luego le va a pedir que escriba el nombre y el contenido de la nueva **receta** que quiere **crear**, y el programa va a crear ese archivo en el lugar correcto.
3. La opción 3 le va a preguntar el nombre de la **categoría** que quiere **crear** y va a generar una carpeta nueva con ese nombre.
4. La opción 4, hará todo lo mismo que la opción uno, pero en vez de leer la **receta**, la va a **eliminar**.
5. La opción 5, le va a preguntar qué **categoría** quiere **eliminar**
6. Finalmente, la opción 6 simplemente va a **finalizar** la ejecución del código.

Este programa tiene algunas cuestiones importantes a considerar:

- Cada vez que el usuario realice exitosamente cualquiera de sus opciones, el programa le va a pedir que presione alguna letra para poder volver al inicio, por lo que el código se va a repetir una y otra vez, hasta que el usuario elija la opción 6. Esto implica que todo el menú debe estar dentro de un loop while que se repita una y otra vez hasta que no se cumpla la condición de que la elección del usuario sea 6
- Sería genial que cada vez que el usuario vuelva al menú inicial, la consola limpie la pantalla para que no se acumulen las ejecuciones anteriores. Recuerda que cuentas con system para poder reiniciar la pantalla y comenzar a mostrar todo desde cero.

- Si bien te he enseñado muchos métodos para administrar archivos, para este ejercicio vas a necesitar algunos que aún no has visto, pero que están incluidos en los objetos con los que hemos estado trabajando, por lo que en ocasiones deberás buscar entre los métodos que trae Path, por ejemplo, leer la documentación y aprender a implementarlo. Yo sé que sería mucho más fácil que yo te enseñe todo acerca de cada uno de los métodos, pero recuerda que también es importante que a medida que avanzamos vayamos aprendiendo a gestionar tu propio aprendizaje. Es parte de la vida real y cotidiana del programador en el mundo en que vivimos.
- Utiliza muchas funciones, todas las que creas necesario. Las funciones ayudan a compartir, modular el código y hacerlo mucho más dinámico, ordenado, repetible y más fácil de mantener.
- Recuerda comenzar con un diagrama de flujos o un gráfico hecho a mano que te permita visualizar con más facilidad el árbol de decisiones que necesitas ejecutar en tu código. Sin eso te vas a enredar más rápido de lo que crees y se te va a complicar bastante.
- Y, por último, no te frustres si no logras hacerlo o completarlo. Si logras hacer una parte, un par de funciones, algunas cosas sí y otras no, está muy bien. Siempre estamos aprendiendo y parte de aprender es no saber.

Mis desafíos siempre te van a estar ubicando en el borde de tus capacidades, sacándote del lugar de confort para que tu cerebro tenga que desconcertarse y descubrir cómo hacer algo nuevo. Tu avanza hasta donde puedas.

TEMA 7 - Programa una cuenta bancaria

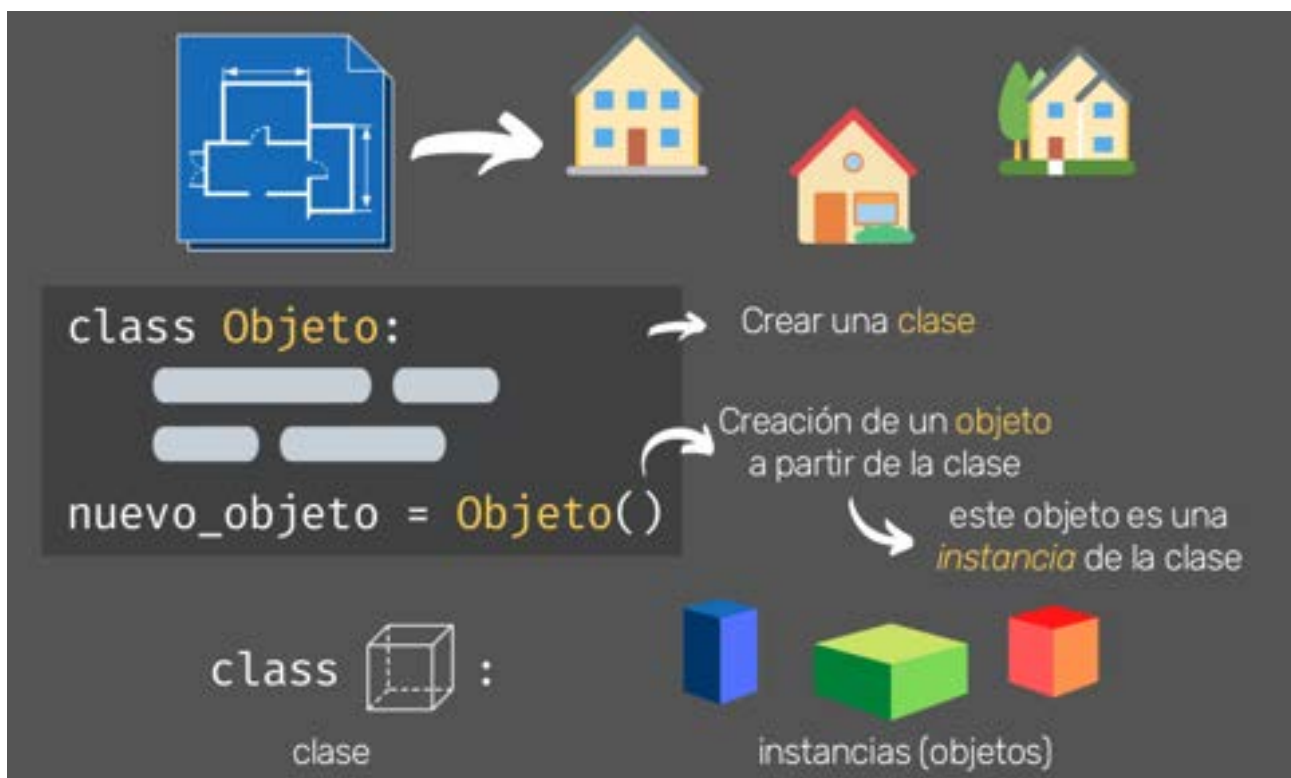
Principios de la programación orientada a objetos (POO):

- Herencia
- Polimorfismo
- Cohesión
- Abstracción
- Acoplamiento
- Encapsulamiento

7.1. - Clases

Python es un lenguaje de Programación Orientado a Objetos (POO). Como tal, utiliza y manipula objetos, a través de sus métodos y propiedades. Las clases son las herramientas que nos permiten crear objetos, que "empaquetan" datos y funcionalidad juntos.

Podemos pensar a las clases como el "plano" o "plantilla" a partir del cual podemos crear objetos individuales, con propiedades y métodos asociados:



7.2. - Atributos

Los atributos son variables que pertenecen a la clase. Existen **atributos de clase** (compartidos por todas las instancias de la clase), y **de instancia** (que son distintos en cada instancia de la clase).



Todas las clases tienen una función que se ejecuta al instanciarla, llamada `__init__()`, y que se utiliza para asignar valores a las propiedades del objeto que está siendo creado. `self`: representa a la instancia del objeto que se va a crear.

7.3. - Métodos

Los objetos creados a partir de clases también contienen métodos. Dicho de otra manera, los métodos son funciones que pertenecen al objeto.

```
class Persona:
    especie = "humano"
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def saludar(self):
        print(f'Hola, mi nombre es {self.nombre}')
    def cumplir_anios(self, estado_humor):
        print(f'Cumplir {self.edad + 1} años me pone {estado_humor}')
juan = Persona("Juan", 37)
juan.saludar()
juan.cumplir_anios("feliz")
```

```
>> Hola, mi nombre es Juan
```

```
>> Cumplir 38 años me pone feliz
```

Cada vez que un atributo del objeto sea invocado (por ejemplo, en una función), debe incluirse `self`, que refiere a la instancia en cuestión, indicando la pertenencia de este atributo.

7.4. - Tipos de métodos

Los métodos estáticos y de clase anteponen un decorador específico, que indica a Python el tipo de método que se estará definiendo

		<i>@classmethod</i>	<i>@staticmethod</i>
	Métodos de instancia	Métodos de clase	Métodos estáticos
Acceso a métodos y atributos de la clase	sí	sí	no
Requiere una instancia	sí	no	no
Acceso a métodos y atributos de la instancia	sí	no	no

Así como los métodos de instancia requieren del parámetro **self** para acceder a dicha instancia, los métodos de clase requieren del parámetro **cls** para acceder a los atributos de clase. Los métodos estáticos, dado que no pueden acceder a la instancia ni a la clase, no indican un parámetro semejante.

Resumen sobre Decoradores:

- **Métodos de instancia:**

```
def mi_metodo(self):
    print('algo')
mi_metodo
```

- Acceden y modifican atributos del objeto.
- Acceden a otro métodos.
- Pueden modificar el estado de la clase.

- **Métodos de clase @classmethod:**

```
def mi_metodo(cls):
    print('algo')
```

- No están asociados a las instancias de la clase, sino a la clase en si misma
- Pueden ser llamados desde la instancia y desde la clase
- No pueden acceder a los atributos de instancia pero si a los de la clase

- **Métodos estáticos** `@staticmethod`:

```
def mi_metodo():
    print('algo')
```

- No acepta como parámetro ni `self` ni `cls`
- No pueden modificar ni el estado de la clase ni de la instancia
- Pueden aceptar parámetros de entrada

7.5. - Herencia

La herencia es el proceso mediante el cual una clase puede tomar métodos y atributos de una clase superior, evitando repetición del código cuando varias clases tienen atributos o métodos en común.

Es posible crear una clase "hija" con tan solo pasar como parámetro la clase de la que queremos heredar:

```
class Personaje:
    def __init__(self, nombre, herramienta):
        self.nombre = nombre
        self.arma = arma
class Mago(Personaje):
    pass
hechicero = Mago("Merlín", "caldero")
```

Una clase "hija" puede sobrescribir los métodos o atributos, así como definir nuevos, que sean específicos para esta clase.

7.6. - Herencia extendida

Las clases "hijas" que heredan de las clases superiores, pueden crear nuevos métodos o sobrescribir los de la clase "padre". Asimismo, una clase "hija" puede heredar de una o más clases, y a su vez transmitir herencia a clases "nietas".

Si varias superclases tienen los mismos atributos o métodos, la subclase sólo podrá heredar de una de ellas. En estos casos Python dará prioridad a la clase que se encuentre más a la izquierda.

Del mismo modo, si un mismo método se hereda por parte de la clase "padre", e "hija", la clase "nieta" tendrá preferencia por aquella más próxima ascendente (siguiendo nuestro esquema, la tomará de la clase "hija").



Clase.__mro__

devuelve el orden de resolución de métodos

super().__init__(arg1, arg2,...)

hereda atributos de las superclases de manera compacta

7.7. - Polimorfismo

El polimorfismo es el pilar de la POO mediante el cual un mismo método puede comportarse de diferentes maneras según el objeto sobre el cual esté actuando, en función de cómo dicho método ha sido creado para la clase en particular.

El método len() funciona en distintos tipos de objetos: listas, tuplas, strings, entre otros. Esto se debe a que para Python, lo importante no son los tipos de objetos, sino lo que pueden hacer: sus métodos.

```
class Perro:
    def hablar(self):
        print("Guau!")
class Gato:
    def hablar(self):
        print("Miau!")
hachiko = Perro()
garfield = Gato()
for animal in [hachiko, garfield]:
    animal.hablar()
>> Guau!
>> Miau!
```

7.8. - Pilares de la Programación Orientada a Objetos

Se ha visto Herencia y Polimorfismo en las prácticas anteriores. Información conceptual del resto:

Cohesión: <https://escueladirecta-blog.blogspot.com/2021/09/cohesion-pilares-de-la-programacion.html>

Acoplamiento: <https://escueladirecta-blog.blogspot.com/2021/10/acoplamiento-pilares-de-la-programacion.html>

Abstracción: <https://escueladirecta-blog.blogspot.com/2021/10/abstraccion-pilares-de-la-programacion.html>

Encapsulamiento: <https://escueladirecta-blog.blogspot.com/2021/10/encapsulamiento-pilares-de-la.html>

7.9. - Métodos especiales

Puedes encontrarlos con el nombre de métodos mágicos o dunder methods (del inglés: dunder = double underscore, o doble guión bajo). Pueden ayudarnos a sobrescribir métodos incorporados de Python sobre nuestras clases para controlar el resultado devuelto.

```
class Libro:
    def __init__(self, autor, titulo, cant_paginas):
        self.autor = autor
        self.titulo = titulo
        self.cant_paginas = cant_paginas
    def __str__(self):
        return f'Título: "{self.titulo}", escrito por {self.autor}'
    def __len__(self):
        return self.cant_paginas
libro1 = Libro("Stephen King", "It", 1032)
print(str(libro1))
print(len(libro1))
>> Título: "It", escrito por Stephen King
>> 1032
```

7.10. - Proyecto del Día 7

Crear un código que le permita a una persona realizar operaciones en su cuenta bancaria. No te asustes que la consigna va a estar bien definida para que puedas hacerlo en poco tiempo.

Primero vas a crear una clase llamada Persona, y Persona va a tener solo dos atributos: nombre y apellido. Luego, vas a crear una segunda clase llamada Cliente, y Cliente va a heredar de Persona, porque los clientes son personas, por lo que el Cliente va a heredar entonces los atributos de Persona, pero también va a tener atributos propios, como número de cuenta y balance, es decir, el saldo que tiene en su cuenta bancaria.

Pero eso no es todo: Cliente también va a tener tres métodos. El primero va a ser uno de los métodos especiales y es el que permite que podamos imprimir a nuestro cliente. Este método va a permitir que cuando el código pida imprimir Cliente, se muestren todos sus datos, incluyendo el balance de su cuenta. Luego, un método llamado Depositar, que le va a permitir decidir cuánto dinero quiere agregar a su cuenta. Y finalmente, un tercer método llamado Retirar, que le permita decidir cuánto dinero quiere sacar de su cuenta.

Una vez que hayas creado estas dos clases, tienes que crear el código para que tu programa se desarrolle, pidiéndole al usuario que elija si quiere hacer depósitos o retiros. El usuario puede hacer tantas operaciones como quiera hasta que decida salir del programa. Por lo tanto, nuestro código tiene que ir llevando la cuenta de cuánto dinero hay en el balance, y debes procurar, por supuesto, que el cliente nunca pueda retirar más dinero del que posee. Esto no está permitido.

Recuerda que ahora que sabes crear clases y objetos que son estables y que retienen información, no necesitas crear funciones que devuelvan el balance, ya que la instancia de cliente puede saber constantemente cuál es su saldo debido a que puede hacer sus operaciones llamando directamente a este atributo y no a una variable separada.

Para que tu programa funcione, puedes organizar tu código como quieras, hay muchas formas de hacerlo, pero mi recomendación es que básicamente, luego de crear las dos clases que te he mencionado, crees dos funciones una que se encarguen de crear al cliente pidiéndole al usuario toda la información necesaria y devolviendo, a través del return, un objeto cliente ya creado.

La otra función (que puede llamarse inicio, o algo por el estilo), es la función que organiza la ejecución de todo el código: primero llama a la función “crear cliente” y luego se encarga de mantener al usuario en un loop que le pregunte todo el tiempo si quiere depositar, retirar o salir del programa y demostrarle el balance, cada vez que haga una modificación.

Para que este programa no se te haga súper largo o complejo, te propongo que esta vez no nos fijemos tanto en los controles, para ver si el usuario ha puesto opciones permitidas o no, si ha puesto números o no, si ha puesto mayúsculas o minúsculas, y creemos el código confiando en que el usuario va a ingresar siempre información apropiada. Por supuesto que si tú prefieres incluir todos esos controles, está genial.

TEMA 8 - Programa una consola de turnos

8.1. - Instalar paquetes

Una de las grandes ventajas de Python como lenguaje de programación, es que cuenta con un amplia comunidad activa, que desarrolla paquetes que añaden muchas más funcionalidades.

PyPi (Python Package Index) es el repositorio de referencia para hallar paquetes desarrollados por la comunidad. <https://pypi.org/>

Si ya conoces el nombre del módulo que quieres instalar, puedes obtenerlo de PyPi directamente desde la consola:

```
pip install <módulo>
```

También podrás actualizar los módulos que ya tengas instalados añadiendo --upgrade luego del nombre del paquete:

```
pip install <módulo> --upgrade
```

No dejes de utilizar Google para investigar nuevos paquetes y sus funcionalidades.

8.2. - Módulos y paquetes

Los módulos no son más que archivos .py, que almacenan funciones, variables y clases, y pueden ser importado por otros. Los paquetes agrupan estos módulos en carpetas, de los cuales uno debe ser `__init__.py`

Importar un módulo:

```
import modulo1
```

Importar una función del módulo:

```
from modulo1 import funcion
```

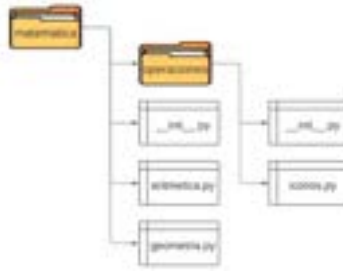
Ejecutar desde la consola

```
C:\... ruta> python modulo1.py
```

Importar un módulo de un (sub) paquete

```
from paquete.subpaquete import modulo3
```

Todos los paquetes, para ser considerados como tales, deben contar con un archivo `__init__.py` (constructor)



8.3. - Manejo de errores

Existen estrategias para capturar y gestionar los errores que pueden presentarse al ejecutar un programa, a fines de evitar una falla mayor y controlar la información que es mostrada al usuario.

try:

El código que se encuentra dentro de try **se ejecuta hasta finalizar o hasta que se presenta un error** (excepción)

except:

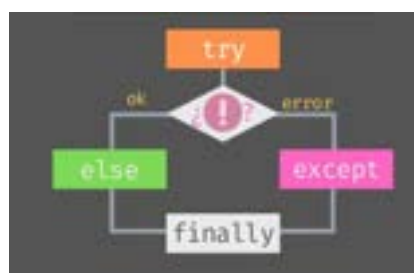
Contiene el **manejador de errores** (respuesta del programa ante un error), atrapando las excepciones que se presentan durante la ejecución de try*

else:

Engloba el código que se ejecutará únicamente **cuando ninguna excepción haya sido detectada** en la ejecución de try (sin errores)

finally:

Contiene código que **se ejecuta siempre**, se hayan presentado o no errores. *



Es buena práctica capturar y manejar las excepciones posibles individualmente y brindar información acerca del error y su posible solución.

```
except ValueError:
except TypeError:
except FileNotFoundError:
...
```

Documentación python de errores: <https://docs.python.org/es/3/library/exceptions.html>

8.4. - pylint

Pylint es un verificador de código, errores y calidad para Python, siguiendo el estilo recomendado por PEP 8, la guía de estilo de Python. Es de gran utilidad en el trabajo en equipo.

```
pip install pylint
```

Ejecutar desde la consola:

```
ruta> pylint modulo1.py -r y
```

↑
archivo de Python a evaluar

Al ejecutarse, Pylint devuelve un reporte con las características que fueron evaluadas, errores y puntuaciones parciales

```
Your code has been rated at 6.67/10
```

A mayor puntaje, mayor será la calidad de tu código. Un umbral aceptable será $\geq 7.00/10$

8.5. - unittest

Unit Testing es un método o herramienta utilizado en programación para determinar si un módulo o un conjunto de módulos de código funciona correctamente. Dicha evaluación se realiza en un archivo independiente. En Python, se implementa desde el módulo incorporado unittest.

```
import unittest
import mimodulo
class NombrePrueba(unittest.TestCase):
    def test_prueba(self):
        primer_valor = {algo}
        segundo_valor = {salida de mimodulo.funcion}
        self.assertEqual(primer_valor, segundo_valor, mensaje)

if __name__ == '__main__':
    unittest.main()
```

Los primeros dos argumentos de `assertEqual` son dos valores que se comparan para establecer si hay igualdad entre ellos. Por eso, uno debe obtenerse a partir de una función del módulo evaluado, y otro ser la salida esperada para una misma entrada de información que en el primer caso. El tercer parámetro (mensaje), contendrá un string con información que se mostrará al usuario en caso de que el test falle.

Antes incluso de ejecutar el código, Python lee el archivo para definir algunas variables globales. Una de ellas es `__name__`, que toma el nombre "`__main__`" en caso que Python esté corriendo en dicho módulo de manera individual. Si por el contrario, el módulo fuera importado, la variable `__name__` toma el nombre del módulo. Este bloque de código evalúa que la prueba se esté ejecutando directamente.

8.6. - Decoradores

Los decoradores son patrones de diseño en Python utilizados para dar nueva funcionalidad a objetos (funciones), modificando su comportamiento sin alterar su estructura: **son funciones que modifican funciones.**

Las funciones en Python soportan operaciones tales como ser asignadas a una variable, pasadas como argumento, y ser devueltas por otra función como resultado.

También, es posible definir funciones dentro de funciones, sin que estén disponibles fuera de la función dentro de la cual fueron definidas.

Los decoradores permiten que una función se modifique ante determinados escenarios, sin duplicar código.

```
def mostrar_informacion(funcion):          > Función como parámetro de una función
    def interior():                       > Definición de una función dentro de otra
        print(f'Ejecutando la función {funcion.__name__}')
        funcion()
        print('Ejecución finalizada')
    return interior                       > Función que devuelve otra función como resultado

def impresion():
    print("Hola Mundo")
funcion_decorada = mostrar_informacion(impresion) > Se asigna una función a una variable
funcion_decorada()                          > Ejecución de función decorada
```

```
>>Ejecutando la función impresion
>>Hola Mundo
>>Ejecución finalizada> Funcionalidad extendida
```

8.7. - Generadores

Los generadores son tipos especiales de funciones que devuelven un iterador que no almacena su contenido completo en memoria, sino que "demora" la ejecución de una expresión hasta que su valor se solicita.

```
def secuencia_infinita():
    num = 0
    while True:
        yield num
        num += 1
```

Dado que un ordenador no cuenta con memoria infinita, no podría generarse una secuencia de números sin límite sin la ayuda de un generador.

Lo mismo ocurre con datos que, sin ser infinitos, ocuparían demasiado espacio en memoria de almacenarse repentinamente.

```
generador = secuencia_infinita()
print(next(generador))
print(next(generador))
print(next(generador))

>>0
>>1
>>2
```

8.8. - Proyecto del Día 8

El desafío de hoy, es que crees un software que funcione como el turnero de una farmacia.

En nuestro caso, vas a crear el turnero para una farmacia que tiene tres áreas de atención: perfumería, farmacia (que es donde venden los medicamentos), y cosméticos. Tu programa le tiene que preguntar al cliente a cuál de las áreas desea dirigirse, y le va a dar un número de turno según a qué área se dirija. Por ejemplo, si elige cosmética le va a dar el número C-54 (“C” de cosmética). Luego de eso, nos va a preguntar si queremos sacar otro turno. Esto, en realidad, es para simular si viene un nuevo cliente. Y repetirá todo el proceso.

Algunas cosas a tener en cuenta:

Los diferentes clientes van a ir sacando turnos para diferentes áreas (perfumería, farmacia, cosmética), en diferentes órdenes, por lo que el sistema debe llevar la cuenta de cuántos turnos ha dado para cada una de esas áreas, y producir el siguiente número de cada área a medida que se lo pida. ¿No te parece genial aprovechar la eficiencia de los generadores para poder hacer esto?

Por otro lado, el mensaje donde le comunicamos el número de espera al cliente, debería tener algo de texto adicional antes y después del número. Por ejemplo, “su turno es (-el número de turno con el del comienzo-)”, y luego algo así como “aguarde y será atendido”. Para que nuestro código no se repita, en vez de poner ese texto en cada una de las funciones que calculen los números, podemos aprovechar la flexibilidad de los decoradores para crear ese texto adicional una sola vez, y luego envolver a cualquiera de nuestras funciones con ese texto único.

Finalmente, deberías aprovechar que ahora ya sabes dividir tu programa en diferentes módulos, y entonces separar el código en dos partes: por un lado, un módulo que se puede llamar **números.py**, en el que vas a escribir todos los generadores y el decorador, y un segundo módulo que podemos llamar **principal.py**, donde vas a escribir las funciones que administran el funcionamiento del programa (como las instrucciones para elegir un área y para decidir si seguirá tomando nuevos turnos o si va a finalizar el programa). Recuerda que vas a necesitar importar el contenido de **numeros.py** dentro de **principal.py** para poder disponer de sus funciones.

TEMA 9 - Programa un buscador de números de serie

9.1. - Módulo collections

El módulo Collections amplía los tipos de contenedores disponibles en Python. Un contenedor almacena diferentes objetos y proporciona una nueva forma de acceder e iterar sobre los mismos.

Counters (Contadores): Es una subclase del **diccionario**, usado para contar las repeticiones de cada elemento en un iterable, en forma de diccionario:

```
from collections import Counter

Counter(iterable)

>> Counter({'valor': repeticiones, ...})
```

DefaultDict: Es una subclase del **diccionario**, usado para proporcionar valores por defecto para las claves que no existan, sin generar un mensaje de error. El valor por defecto puede ser un tipo de dato (**int**, **list**, etc.) o una función lambda que proporcione dicho valor directamente (**lambda: "mi valor"**).

```
from collections import defaultdict

mi_dic = defaultdict(lambda: "No encontrado")
```

NamedTuple: devuelve una tupla donde las posiciones de sus elementos tienen un nombre, además de un número de índice como las tuplas tradicionales.

```
from collections import namedtuple

mi_tupla= namedtuple('Persona',['nombre','edad','altura'])
persona1 = Persona("Marcos", 39, 1.88)
```

nombres de los elementos
▲

9.2. - Módulos shutil & os

El módulo Shutil ofrece funcionalidades de alto nivel sobre archivos, tales como copiar, crear, eliminar y mover entre directorios. También mencionaremos métodos del módulo os que cumplen objetivos similares.

```
import shutil
```

- **shutil.move(archivo, directorio)** : mueve un archivo desde el directorio actual hacia aquel que se especifica en el segundo parámetro.
- **os.unlink(directorio)** : elimina un archivo del directorio indicado
- **os.rmdir(directorio)** : elimina una carpeta vacía
- **shutil.rmtree(directorio)** : elimina una carpeta indicada en el directorio, incluyendo todas sus ramificaciones (subcarpetas y archivos), de manera definitiva y sin pasar por la papelera de reciclaje.
- **send2trash.send2trash(archivo)** : envía un archivo a la papelera de reciclaje (es necesario instalar el módulo desde "*pip install Send2Trash*" y luego importarlo)
- **os.walk(directorio)** : recorre el directorio indicado, y devuelve los nombres de carpetas, subcarpetas y archivos dentro de ellas en forma de tupla, a través de un generador.

9.3. - Módulo datetime

El módulo datetime (incorporado en Python) puede importarse en proyectos para trabajar con fechas y horas, así como intervalos y duraciones.

```
import datetime
```

fecha:

```
mi_fecha = datetime.date(año,mes,día)
```

año,mes,día son integers comprendidos en los rangos de fechas "reales" (12 meses y 31 días como máximo). También podemos extraer el año, mes y día individualmente:

```
anio = mi_fecha.year
```

```
mes = mi_fecha.month
```

```
día = mi_fecha.day
```

```
hoy = datetime.date.today() # obtener el día actual
```

hora:

```
mi_hora = datetime.time(hora, minuto, segundo, microsegundo)
```

Todos los argumentos son opcionales (se asumen 0 si no se indican), y deben estar comprendidos entre 0 y 24 para las horas, 0 y 60 para minutos y segundos, y 0 y 1000000 para los microsegundos.

fecha y hora:

combina fechas y horas

```

fecha_hora = datetime.datetime(año, mes, día, hora, minuto, segundo, microseg)
ahora = datetime.datetime.now()          # obtener fecha y hora actual

hora = ahora.hour
minuto = ahora.minute
segundo = ahora.second

# obtener horas, minutos y segundos

```

9.4. - Módulo para medir el tiempo

Estudiar el tiempo transcurrido durante la ejecución de un código nos permite conocerlo mejor y tomar decisiones acerca de la vía más eficiente para resolver un problema. Tenemos dos módulos que nos ayudarán: time y timeit.

utilizando time:

```

inicio = time.time()
[código]
final = time.time()
duracion = final - inicio      # para funciones que toman varios segundos para ejecutarse

```

utilizando timeit:

```

duracion = timeit.timeit(declaracion, setup, number = numero)

```

declaracion: recibe el código que cuya duración de ejecución queremos medir

→ es la invocación mi función

setup: recibe las instrucciones que el parámetro declaracion requiere para funcionar

→ es la definición (def...) de mi función

number: cantidad de veces que se evaluará el código para obtener su tiempo de ejecución mínimo.

→ pueden ser varios miles o cientos de veces (dependiendo de la complejidad)

9.5. - Módulo math

El módulo math contiene un conjunto de métodos y constantes que se pueden utilizar para resolver tareas matemáticas de mayor complejidad. Es el equivalente a la calculadora científica dentro de Python.

Algunas de las situaciones que pueden ayudarnos a resolver son:

- **Relaciones trigonométricas** (seno, coseno, tangente, sus inversas e hiperbólicas)
- **Funciones logarítmicas**
- **Potencias y raíces**
- **Combinatoria y permutaciones**
- **Redondeos**
- **Factoriales**

... entre muchas otras (¡te recomendamos leer su documentación de acuerdo con tus necesidades!)

Las constantes que encontrarás:

- **Pi** (3.1415...)
- **e o Constante de Euler** (2.7182...)
- **Tau** (6.2831...)
- **Infinito** (el concepto matemático de infinito positivo)
- **Nulo** (NaN: not-a-number)

9.6. - Expresiones regulares

Una expresión regular es una secuencia de caracteres que forman un patrón de búsqueda determinado. Pueden ser utilizadas para verificar strings en búsqueda de un contenido (patrón) específico. Utilizamos el módulo re en Python.

```
import re
```

Funciones:

- **search()**: devuelve un objeto "match" que contiene información acerca del hallazgo si se encuentra en algún punto del string
- **findall()**: devuelve una lista que contiene todos los hallazgos del patrón

Para formar patrones, utilizamos los siguientes cuantificadores y caracteres especiales.

Operadores especiales:

[]	un set de caracteres
.	un carácter cualquiera
^	inicia con
\$	finaliza con
	operador lógico "O"

Cuantificadores:

*	cero o más ocurrencias: 0 – n
+	una o más ocurrencias: 1 – n
?	cero o una ocurrencia: 0 – 1
{ }	un número especificado de ocurrencias
{n}	se repite n veces
{n,m}	se repite de n a m veces
{n, }	se repite de n veces hacia arriba

Caracteres especiales:

\d	dígito numérico
\D	NO numérico
\w	caracter alfanumérico
\W	NO alfanumérico
\s	espacio en blanco
\S	NO espacio en blanco

```
patron = r'\w{4}\d{4}'
verificar = re.search(patron,"cont1234")
```

Expresiones regulares python online: <https://pythex.org/>

9.7. - Comprimir y descomprimir archivos

El formato zip permite comprimir archivos sin pérdida de información, ahorrando espacio de almacenamiento y manteniendo documentos relacionados en un mismo archivo .zip.

Utilizando el módulo zipfile:

```
import zipfile
```

Comprimir archivos:

```
mi_zip = zipfile.ZipFile("archivo_comprimido.zip", "w")      # modo escritura
mi_zip.write("mi_archivo.txt")                             # comprimir archivo en mi_zip
mi_zip.close()
```

Descomprimir archivos:

```
mi_zip = zipfile.ZipFile("archivo_comprimido.zip", "r")
mi_zip.extractall()                                       # extraer todos los archivos de mi_zip
mi_zip.extract("mi_archivo.txt")                         # extraer un archivo específico
```

Utilizando el módulo shutil:

```
import shutil
```

Comprimir archivos:

```
shutil.make_archive(archivo_destino, "zip", carpeta_origen)
```

Descomprimir archivos:

```
shutil.unpack_archive(archivo_zip, nombre_carpeta_extraccion, "zip")
```

9.8. - Proyecto del Día 9

A esta altura del día ya estoy un poco cansado, así que vamos a hacer algo distinto en esta ocasión. Esta vez, en vez de explicarte los detalles del proyecto del día de hoy, los vas a tener que encontrar tú mismo.

Junto a esta lección, vas a encontrar un archivo zip descargable. Descárgalo, guárdalo en la misma carpeta donde vas a guardar tus archivos de Python para este proyecto, y luego quiero que descomprimas ese archivo para encontrar las consignas del proyecto de hoy. Pero no hagas trampa: la idea es que descomprimas ese archivo usando el código que has aprendido el día de hoy, y lo que vas a encontrar ahí, es un archivo llamado Instrucciones. Ese archivo tiene todo lo que necesitas para ponerte a trabajar.

Así que manos a la obra, y nos vemos en un rato para mostrarte mi solución al proyecto del día de hoy.

TEMA 10 - Programa el juego "Invasión espacial"

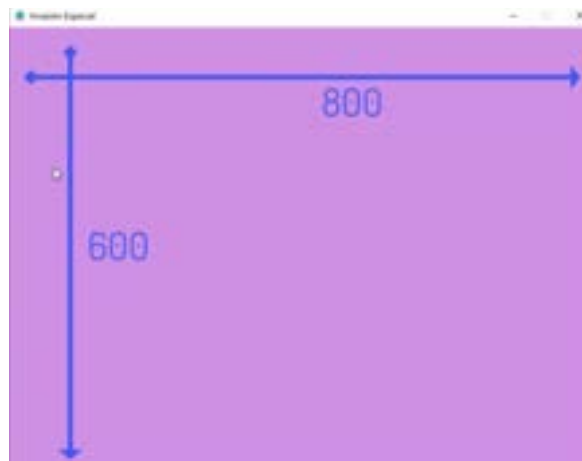
Biblioteca pygame <https://pypi.org/project/pygame/>

pip install pygame

Para descargar iconos: <https://www.flaticon.com/>

Rueda RGB: <https://www.colorspire.com/rgb-color-wheel/>

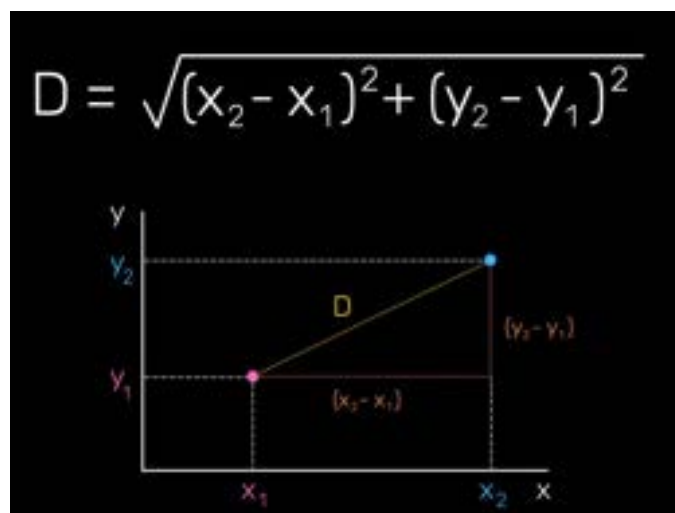
Debemos tener en cuenta el ancho y alto de la pantalla creada, respecto el eje x e y, para ubicar objetos en ella.



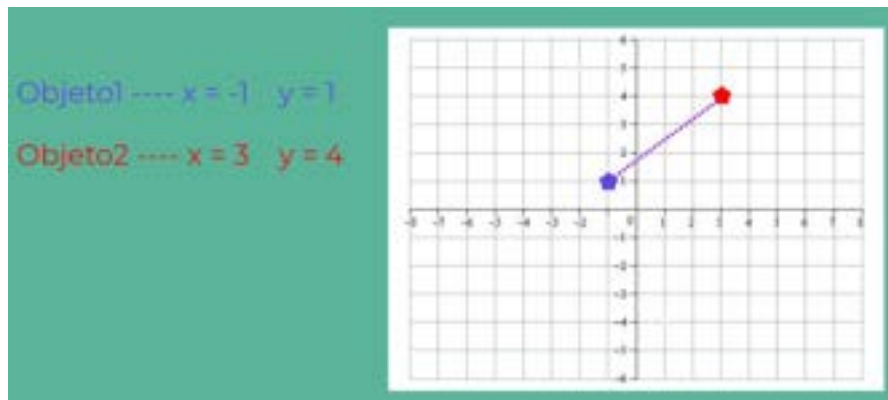
Descargar imágenes: <https://www.freepik.com/>

10.1. - Distancia entre dos puntos

La fórmula de la distancia es una expresión algebraica utilizada para determinar la distancia entre dos puntos de coordenadas (x_1, y_1) y (x_2, y_2) .



Ejemplo de la fórmula:



$$D = \text{sqrt}[(3 - -1)^2 + (4 - 1)^2]$$

$$D = \text{sqrt}[(4)^2 + (3)^2]$$

$$D = \text{sqrt}[16 + 9]$$

$$D = \text{sqrt}[25]$$

$$D = 5$$

Esto nos sirve para ajustar las colisiones en el juego, de la bala con el enemigo o del enemigo con el jugador.

10.2. - Convertir el Juego en un Archivo Ejecutable (.exe)

Este es un manual paso a paso para transformar cualquier programa de Python en un programa independiente, para poderlo ejecutar fuera del IDE.

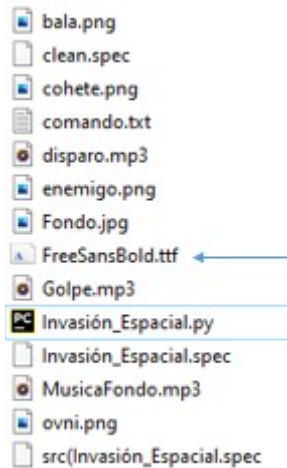
Básicamente el proceso consta de 2 grandes pasos:

1. Convertir las fuentes de tipo Sting a objetos Bytes
2. Utilizar pyinstaller

Vamos por partes:

1. Convertir las fuentes de tipo Sting a objetos Bytes

1. Descarga la o las fuentes empleadas en el juego, en este caso FreeSansBold.ttf (<https://www.download-free-fonts.com/details/2045/free-sans-bold>). Luego guárdala en la carpeta donde se encuentra Invasión_Espacial.py como se muestra en la imagen siguiente:



2. Crea una función que transforme el nombre de la fuente (“FreeSansBold.ttf”) de string a objeto Bytes. Para eso importamos la librería io, y pasamos como parámetro el nombre de la fuente al almacenar la función en una variable.

```
import io

def fuente_bytes(fuente):
    # Abre el archivo TTF en modo lectura binaria
    with open(fuente, 'rb') as f:
        # Lee todos los bytes del archivo y los almacena en una
        variable
        ttf_bytes = f.read()
        # Crea un objeto BytesIO a partir de los bytes del archivo TTF
        return io.BytesIO(ttf_bytes)
```

3. Almacena la función en una variable que luego se pasará como objeto Bytes a pygame.font.Font.

```
# Puntaje
puntaje = 0
fuente_como_bytes = fuente_bytes("FreeSansBold.ttf")
fuente = pygame.font.Font(fuente_como_bytes, 32)
texto_x = 10
texto_y = 10
```

```
# Texto Final de Juego
fuente_final = pygame.font.Font(fuente_como_bytes, 40)

def texto_final():
    mi_fuente_final = fuente_final.render("GAME OVER", True, (255, 255,
255))
    pantalla.blit(mi_fuente_final, (60, 200))
```

2. Utilizar pyinstaller
 1. Instala pyinstaller usando:

pip installer pyinstaller

2. Abre CMD en la carpeta donde se encuentra el archivo `Invasión_Espacial.py`

```
DATA (D:) > PYTHON_CURSO > DIA 10  
C:\Windows\System32\cmd.exe  
Microsoft Windows [Versión 10.0.19044.2364]  
(c) Microsoft Corporation. Todos los derechos reservados.  
D:\PYTHON_CURSO\DIA 10>
```

3. Escribe el siguiente comando:

```
pyinstaller --clean --onefile --windowed Invasión_Espacial.py
```

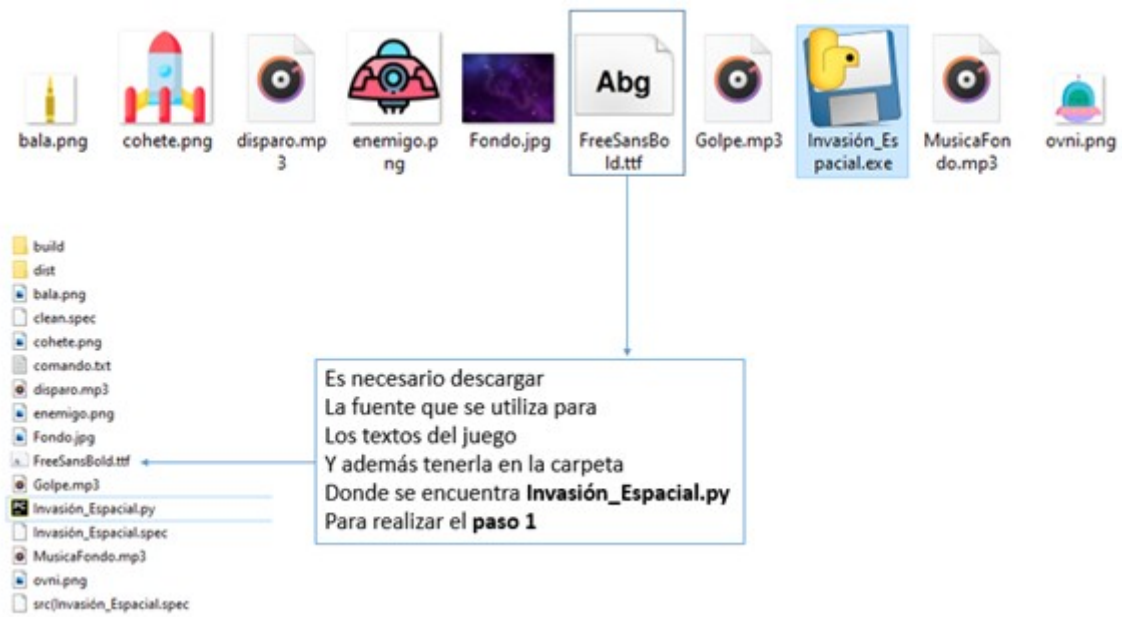
Donde cada expresión significa lo siguiente:

- **--clean**: elimina todos los archivos temporales y directorios creados por pyinstaller durante la construcción del archivo ejecutable.
- **--onefile**: crea un archivo ejecutable que contiene todos los archivos necesarios para ejecutar el script, incluyendo los módulos y bibliotecas utilizadas por el script.
- **--windowed**: crea un archivo ejecutable que se ejecuta en una ventana en lugar de en pantalla completa.
- **Invasión_Espacial.py**: es el nombre del script Python que se va a convertir en un archivo ejecutable.

4. Luego de unos segundos se terminará de correr el comando y la consola mostrará el siguiente mensaje:

```
28808 INFO: Building EXE from EXE-00.toc completed successfully.  
D:\PYTHON_CURSO\DIA 10>
```

Se van a generar dos carpetas, una llamada `built` y otra llamada `dist`. En esta última se deberán copiar todos los archivos que son referencias para que el juego funcione:



¡Y eso es todo! ¡Espero que te sea de gran ayuda para que puedas compartir tus programas con el mundo!

A partir de este tema son ejercicios prácticos. La mayoría de los apuntes están documentados en comentarios en el mismo código.

TEMA 11 - Programa un extracto de datos web

Web scraping = raspar internet

- Reglas del web scraping
- Limitaciones del web scraping

Se utilizarán tres bibliotecas: beautifulsoup4, lxml y requests. Se deben instalar:

```
pip install beautifulsoup4
```

```
pip install requests
```

Enlace: <https://escueladirecta-blog.blogspot.com/>

11.1. - Extraer elementos de una clase

Carácter	Sintaxis	Resultados
"	soup.select('div')	Todos los elementos con la etiqueta 'div'
#	soup.select('#estilo_4')	Elementos que contengan id='estilo4'
.	soup.select('.columna_der')	Elementos que contengan class='columna der'
(ESPACIO)	soup.select('div span')	Cualquier elemento llamado 'span' dentro de un elemento 'div'
>	soup.select('div>span')	Cualquier elemento llamado 'span' directamente dentro de un elemento 'div', sin nada en el medio

Enlace: <https://www.escueladirecta.com/courses>

Enlace: <https://toscraper.com/>

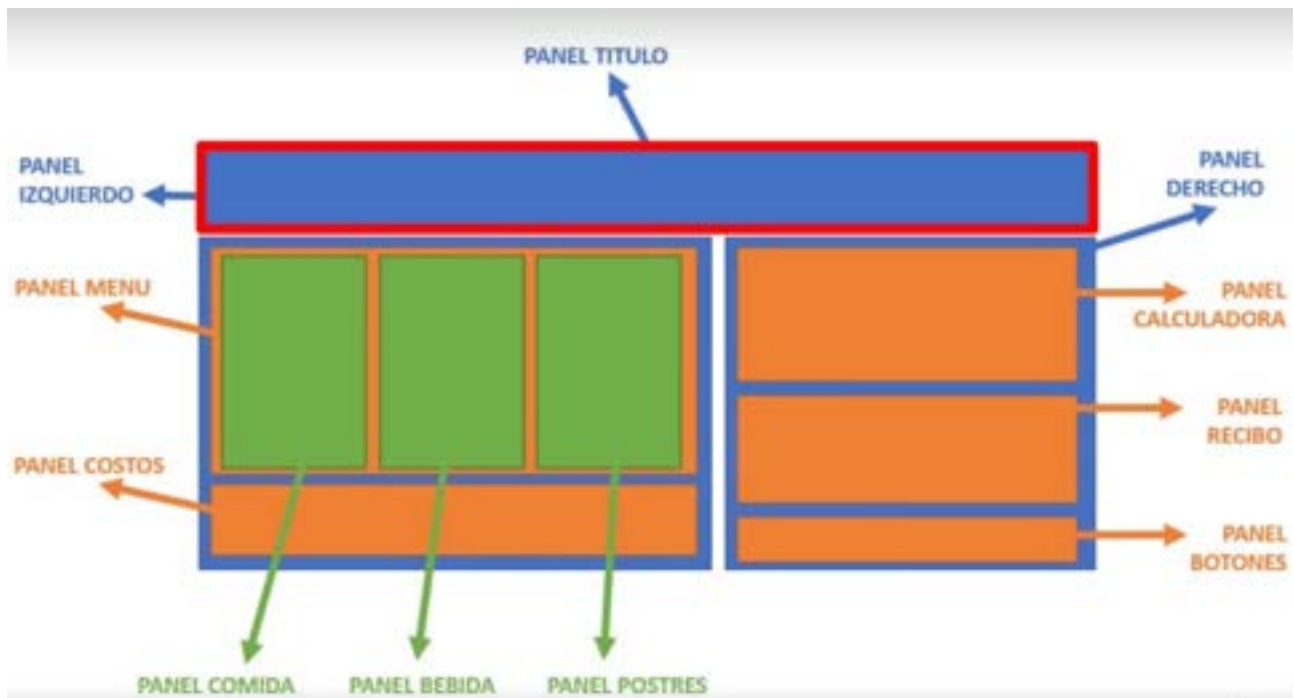
TEMA 12 - Programa un gestor de restaurantes

Tkinter sirve para programar interfaces gráficas de usuario.

```
from tkinter import *
```

```
from tkinter import filedialog, messagebox
```

Enlace. Lista de colores de Tkinter: https://es.wikibooks.org/wiki/Python/Interfaz_gr%C3%A1fica_con_Tkinter/Los_nombres_de_los_colores



Para el Frame (marco) podemos indicar el relief con estas opciones:

- Flat
- Raised
- Sunked
- Groove
- Ridge

Para darle una especie de tridimensionalidad

7	8	9	+
4	5	6	-
1	2	3	x
CE	Borrar	0	/

TEMA 13 - Programa un asistente de voz

13.1. - Librerías y módulos

pyttsx3: Librería de síntesis de voz en Python. Permite convertir texto en voz humana de forma programática utilizando diferentes voces y ajustes.

speech_recognition: Librería para reconocimiento de voz en Python. Permite grabar voz desde un micrófono o cargar un archivo de audio, y convertirlo en texto utilizando tecnologías de reconocimiento de voz. *pip install SpeechRecognition*

webbrowser: Módulo de Python que permite controlar el navegador web del SO. Permite abrir páginas web, buscar en Internet y realizar otras acciones relacionadas con la navegación web.

pywhatkit: Librería de Python que proporciona una interfaz de alto nivel para realizar tareas comunes en la web, como búsqueda en Google, envío de mensajes de WhatsApp, reproducción de vídeos de YouTube y otras tareas similares.

yfinance: Librería de Python para acceder a datos financieros de Yahoo Finance. Permite descargar y analizar datos históricos de precios de acciones, índices y otros recursos financieros.

pyjokes: Librería de Python que proporciona una colección de chistes en inglés. Los chistes se pueden utilizar para hacer programas más divertidos o para fines de aprendizaje de idiomas.

wikipedia: es una librería de Python que permite acceder a la enciclopedia en línea Wikipedia. Permite buscar y recuperar información detallada sobre temas específicos, así como interactuar con la API de Wikipedia.

13.2. - Algunos problemas con las bibliotecas

13.2.1. - Problemas con Flask

Si al ejecutar el código de la lección "**Transformar Voz en Texto**" recibes el siguiente mensaje de error:

```
ModuleNotFoundError: No module named 'flask'
```

Esto simplemente se soluciona instalando la biblioteca Flask con el siguiente comando:

```
pip install flask
```

13.2.2. - Problemas con PyAudio

Aunque no vamos a instalar manualmente la biblioteca PyAudio, sí vamos a instalar otra biblioteca (**SpeechRecognition**) que al ejecutarse se encarga de instalar PyAudio sin que lo veamos en pantalla. Curiosamente PyAudio no siempre se instala como corresponde y esto le está trayendo dolores de cabeza a más de un estudiante.

Si al ejecutar el código de la lección "**Transformar Voz en Texto**" recibes el siguiente mensaje de error:

```
AttributeError("Could not find PyAudio; check installation")
```

en ese caso, la forma de resolverlo es instalar PyAudio manualmente, pero esto no siempre resulta como debería, por lo que a continuación voy a compartir contigo las diferentes soluciones que están dando resultado. Intenta con cada una de ellas en orden hasta que alguna te de resultado.

13.2.2.1. - Solución 1

Abre CMD en Windows (o la terminal en Mac) y escribe lo siguiente:

```
pip install PyAudio
```

13.2.2.2. - Solución 2

1 - En PyCharm ve a **File / Settings / Project: Python / Python Interpreter**

2 - Click en +

3 - En la barra de búsquedas escribe "**PyAudio**"

4 - Selecciona el resultado correcto y haz click en **Install Package**

13.2.2.3. - Solución 3

Abre CMD en Windows (o la terminal en Mac) y escribe lo siguiente:

```
pip install pipwin
```

```
pipwin install PyAudio
```

13.2.2.4. - Solución 4

1 - En PyCharm ve a **View / Tool Windows / Terminal**

2 - Escribe lo siguiente:

```
pip install pipwin
pipwin install PyAudio
```

13.2.2.5. - Solución 5

1 - Asegúrate de conocer la versión de python que tienes instalada. Puedes verificarlo escribiendo lo siguiente en la terminal:

```
python --version
```

Obtendrás, por ejemplo:

```
Python 3.10.0
```

2 - Comprueba qué versión de Python tienes instalada (32 o 64 bits). La manera más sencilla es escribir en la terminal lo siguiente:

```
python
```

De esta manera obtendrás, entre otra información, algo parecido a lo siguiente, que se muestra en mi caso

```
MSC v.1927 64 bit (AMD64)] on win32.
```

Lo importante es lo resaltado entre paréntesis. Toma nota de lo que obtienes al hacerlo tú.

3 - De acuerdo con lo anterior, en mi ordenador tendría que instalar una versión de PyAudio para Python 3.10 (310) de 64 bits (win_amd64). Nuevamente, puede no ser la misma versión que necesites tú.

4 - Ingresa en este enlace <https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyaudio> , y dirígete al encabezado que dice PyAudio: bindings for the PortAudio library.

5 - Ahí encontrarás listados varios archivos que se usan para reemplazar la versión de PyAudio que ofrece PyPi. Debes elegir la que corresponde a tu ordenador y a tu versión de Python (según los datos que te enseñé a recoger en los puntos 1 y 2 de esta solución). En mi caso tendría que descargar la siguiente versión:

```
PyAudio-0.2.11-cp310-cp310-win_amd64.whl
```

Descarga la que necesites tú de acuerdo a lo verificado en los puntos 1 y 2.

6. Localiza la carpeta de descargas en donde se haya bajado el archivo. Por ejemplo en mi caso:

```
C:\Users\Win10\Downloads
```

7. Abre la terminal y escribe:


```
cd C:\Users\Usuario\Downloads
```

8. A continuación escribe en la terminal lo siguiente:

```
pip install PyAudio-0.2.11-cp39-cp39-win_amd64.whl
```

(deberás reemplazar el nombre del archivo descargado por el que hayas bajado tú).

9. Esto habrá instalado PyAudio y solucionará tu problema.

13.3. - Enlaces

Instrucciones para descargar idiomas en Windows:

<https://support.microsoft.com/es-es/topic/descargar-idiomas-y-voces-para-lector-inmersivo-el-modo-lectura-y-lectura-en-voz-alta-4c83a8d8-7486-42f7-8e46-2b0fdf753130>

Instrucciones para cambiar el idioma en Mac:

<https://support.apple.com/es-es/guide/mac-help/mh26684/mac#:~:text=En%20el%20Mac%2C%20selecciona%20el,en%20%E2%80%9CIdioma%20y%20regi%C3%B3n%E2%80%9D%20.&text=Haz%20clic%20en%20General.,y%20haz%20clic%20en%20A%C3%B1adir.>

TEMA 14 - Programa un controlador de asistencia

Paso 1 – Reconocimiento facial

Paso 2 – Análisis facial

Paso 3 – Convertir la imagen en datos

hog → Histograms of oriented gradients for human detection

https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

Vscode: <https://visualstudio.microsoft.com/es/downloads/>

14.1. - Bibliotecas

cmake: es una biblioteca de software que se utiliza para construir y compilar aplicaciones de software. Es muy útil para proyectos que tienen dependencias y para proyectos que deben compilarse en diferentes sistemas operativos.

dlib: es una biblioteca de procesamiento de imágenes y aprendizaje automático en C++ que se puede utilizar en Python. Es muy utilizada en aplicaciones de reconocimiento facial, detección de objetos y seguimiento de objetos.

face-recognition: es una biblioteca de Python basada en dlib, que permite detectar y reconocer caras en imágenes y vídeos. Utiliza técnicas de aprendizaje automático para identificar características faciales únicas, y es muy útil en aplicaciones de seguridad, reconocimiento de personas y análisis de imágenes.

numpy: es una biblioteca de Python para computación científica que se utiliza para realizar operaciones matemáticas en matrices y vectores. Es muy útil en aplicaciones de análisis de datos, aprendizaje automático, procesamiento de señales y otras áreas de la ciencia y la ingeniería.

opencv-python: es una biblioteca de procesamiento de imágenes y vídeo en Python. Proporciona funciones para leer, escribir y procesar imágenes y vídeos, así como para realizar operaciones avanzadas como reconocimiento facial, detección de objetos y seguimiento de objetos.

Para instalar estas bibliotecas en Ubuntu, hay varias opciones:

- Con el comando "sudo apt-get install" seguido del nombre del paquete.
- Con el comando "pip install" seguido del nombre de la biblioteca.
- Con el comando "conda install" seguido del nombre de la biblioteca.

Cuando realizamos la comparación de las imágenes, con las caras codificadas y mediante el código:

```
fr.compare_faces(lista_imagenes, imagen_a_comparar)
```

Tienen un valor True cuando tiene un valor de 0.6 (Valor por defecto) en el punto de comparación (La distancia entre ambas caras)

Los ficheros .csv se llaman así por "Comma-Separated Values" (Valores separados por comas), lo que indica que se trata de un formato de archivo en el que los datos están separados por comas.

TEMA 15 - Programa un modelo de machine learning

15.1. - Bibliotecas

numpy: es una biblioteca de Python para computación científica que se utiliza para realizar operaciones matemáticas en matrices y vectores. Es muy útil en aplicaciones de análisis de datos, aprendizaje automático, procesamiento de señales y otras áreas de la ciencia y la ingeniería.

Documentación: <https://numpy.org/devdocs/user/index.html>

pandas: es una biblioteca de Python para análisis de datos que se utiliza para manipular y analizar datos estructurados. Proporciona estructuras de datos y funciones para la limpieza, manipulación y análisis de datos en tablas y series de tiempo. Se le conoce como el excel de python.

Documentación: <https://pandas.pydata.org/pandas-docs/stable/>

matplotlib: es una biblioteca de Python para visualización de datos que se utiliza para crear gráficos y visualizaciones en 2D y 3D. Proporciona una amplia gama de tipos de gráficos, como gráficos de línea, gráficos de barras, gráficos de dispersión y gráficos de torta, y es muy útil en aplicaciones de análisis de datos y ciencia de datos.

Documentación: <https://matplotlib.org/stable/index.html>

Seaborn: es una biblioteca de visualización de datos en Python que se basa en matplotlib. Proporciona una interfaz de alto nivel para crear gráficos estadísticos atractivos y informativos. Seaborn simplifica la creación de visualizaciones complejas al proporcionar estilos preestablecidos y funciones optimizadas para representar relaciones estadísticas. Es especialmente útil en el análisis exploratorio de datos y la presentación visual de resultados.

Documentación: <https://seaborn.pydata.org/>

scikit-learn (sklearn): es una biblioteca de aprendizaje automático de código abierto para Python que proporciona una amplia gama de algoritmos y herramientas para el análisis de datos. Es ampliamente utilizado en ciencia de datos debido a su facilidad de uso y eficiencia computacional.

Documentación: <https://scikit-learn.org/stable/>

DecisionTreeClassifier es un módulo de scikit-learn es una implementación de un clasificador de árbol de decisiones en Python. Permite construir modelos de clasificación basados en árboles de decisión, que son diagramas de flujo donde cada nodo interno representa una característica, cada rama representa una regla de decisión y cada hoja representa una clase o una predicción.

Documentación:

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

accuracy_score: es un módulo de scikit-learn que proporciona funciones para evaluar y visualizar el rendimiento de los modelos de clasificación. En concreto, calcula la precisión del modelo.

Documentación:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

confusion_matrix: es un módulo de scikit-learn que proporciona funciones para evaluar y visualizar el rendimiento de los modelos de clasificación. En concreto, genera una matriz de confusión que muestra la cantidad de aciertos y errores de cada clase. traza una representación gráfica de la matriz de confusión

Documentación:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

plot_confusion_matrix: es un módulo de scikit-learn que proporciona funciones para evaluar y visualizar el rendimiento de los modelos de clasificación. En concreto, traza una representación gráfica de la matriz de confusión. DEPRECATED. En su lugar utiliza *ConfusionMatrixDisplay*.

ConfusionMatrixDisplay: es una clase en scikit-learn que permite visualizar de forma gráfica una matriz de confusión. Proporciona una representación visual clara y legible de los resultados de la clasificación. La matriz de confusión es una tabla que muestra las predicciones de un modelo de clasificación en comparación con las etiquetas verdaderas. La visualización de la matriz de confusión proporcionada por *ConfusionMatrixDisplay* puede ayudar a evaluar el rendimiento del modelo al mostrar de manera intuitiva los aciertos y los errores de clasificación para cada clase.

Documentación:

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html>

tree: es un módulo de scikit-learn que proporciona herramientas para visualizar los árboles de decisión creados por los modelos entrenados, lo que ayuda a comprender mejor cómo se toman las decisiones dentro del modelo.

Documentación: <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.tree>

15.2. - Definiciones

La **ciencia de datos** o data Science es un campo especial de la ciencia que aplica métodos científicos, procesos y sistemas para poder extraer conocimiento o un mejor entendimiento de los datos en sus diferentes formas.

Machine Learning (ML) o Aprendizaje Automático es una rama de la inteligencia artificial que permite a las máquinas aprender a partir de datos y realizar tareas sin ser específicamente programadas para hacerlo. El objetivo es que la máquina pueda generalizar el conocimiento aprendido a partir de un conjunto de datos y aplicarlo en nuevas situaciones. Se utiliza en una amplia variedad de aplicaciones, incluyendo la visión por computadora, el procesamiento del lenguaje natural, la detección de fraude, el reconocimiento de voz, la predicción y la toma de decisiones.

Datasets - Conjunto de datos

Un **documento Colab** es un archivo de Jupyter Notebook que se ejecuta en la plataforma de Google Colaboratory, lo que permite acceder a recursos de hardware y herramientas de análisis de datos y bibliotecas de Python desde cualquier lugar con acceso a internet.

15.3. - Cuadernos de trabajo en Colab de google drive

Cuaderno de prácticas con Numpy:

https://colab.research.google.com/drive/1vp7zrchG_pJF3uzEgbCjfb_piu43mJXS?usp=sharing

Cuaderno de prácticas con Panda: [https://colab.research.google.com/drive/1-](https://colab.research.google.com/drive/1-E33EMCehgPnmqgwm13-SZSnVYOHMpuQ?usp=sharing)

[E33EMCehgPnmqgwm13-SZSnVYOHMpuQ?usp=sharing](https://colab.research.google.com/drive/1-E33EMCehgPnmqgwm13-SZSnVYOHMpuQ?usp=sharing)

Cuaderno de prácticas con Matplotlib:

https://colab.research.google.com/drive/1MX9ee5h_TTEc0HqZA6f6Ns6J4rwReHOH?usp=sharing

Cuaderno de prácticas con Machine Learning:

https://colab.research.google.com/drive/1zRVpPLmhMkmhX_kB9qrWUppHzRTJPFM?usp=sharing

Podemos conectar nuestro cuaderno con:

```
from google.colab import drive
drive.mount('/content/drive')
```

Página original con el reto «supervivientes del Titanic»: <https://www.kaggle.com/c/titanic>

TEMA 16 - Programa una aplicación web de tareas pendientes

16.1. - Entornos Virtuales

Un **entorno virtual de Python** es una herramienta que permite aislar y gestionar de forma independiente los paquetes y dependencias de un proyecto Python específico. Permite crear un entorno de desarrollo aislado donde se pueden instalar versiones específicas de bibliotecas y paquetes, sin interferir con otros proyectos de Python en el mismo sistema.



Cuando se trabaja en proyectos de Python, es común tener diferentes versiones de paquetes y dependencias para cada proyecto. Esto puede generar conflictos si se instalan globalmente en el sistema, ya que un proyecto podría depender de una versión específica de una biblioteca, mientras que otro proyecto puede requerir una versión diferente.

Al utilizar un entorno virtual, se crea un directorio separado con su propia instalación de Python y su propio espacio de trabajo aislado. Dentro de este entorno, se pueden instalar los paquetes y dependencias necesarios para el proyecto sin afectar el sistema global.

Existen varias herramientas populares para crear y gestionar entornos virtuales en Python, como `virtualenv`, `venv` (incorporado en Python 3.3 y versiones posteriores) y `conda` (utilizado con el gestor de paquetes Anaconda).

Al activar un entorno virtual, se configuran las variables de entorno y se modifica el `PATH` para que el sistema utilice la instalación y las bibliotecas específicas del entorno virtual. Esto asegura que el proyecto utilice las versiones correctas de las bibliotecas y evita conflictos con otras instalaciones globales.

En resumen, un entorno virtual de Python es una herramienta que permite crear un espacio de trabajo aislado con su propia instalación de Python y bibliotecas. Se utiliza para gestionar y mantener las dependencias de un proyecto específico, evitando conflictos con otras instalaciones y versiones globales de paquetes. Esto facilita el desarrollo y la colaboración en proyectos de Python al garantizar la consistencia en las dependencias utilizadas.

Instalación de `virtualenv`:

```
pip install virtualenv
```

pip freeze es un comando utilizado en Python para generar una lista de todas las bibliotecas instaladas y sus versiones en un entorno virtual.

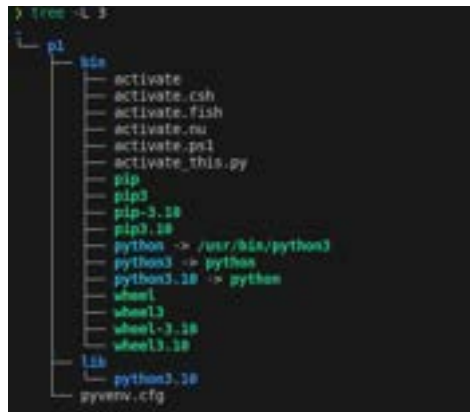
Ahora se hace una estructura de carpetas para diferenciar entornos



En el proyecto1 ejecutamos el comando:

```
virtualenv p1
```

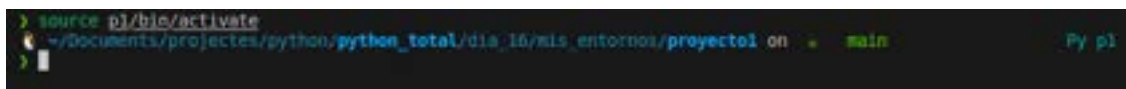
Crearé la estructura de un entorno



Para activar el entorno se hace con el comando:

```
source p1/bin/activate
```

Aparecerá el nombre del entorno en el prompt



Para desactivar:

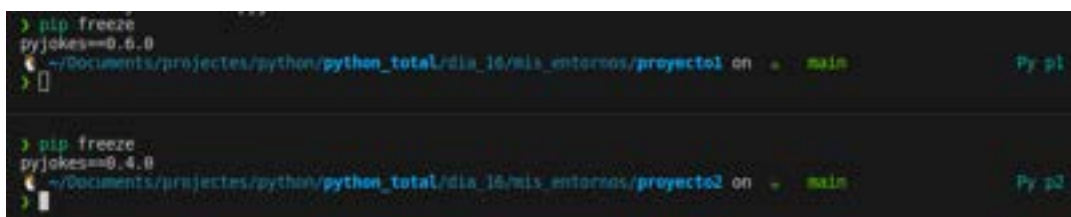
```
deactivate
```

Cuando este el entorno virtual activado, si probamos el comando «pip freeze» veremos que no tenemos ningún módulo instalado en el entorno virtual.

Para instalar una versión distinta, que no sea la última, podemos escogerla con doble igual. Por ejemplo:

```
pip install pyjokes==0.4.0
```

Ahora si lo instalamos en el proyecto2 tendremos dos versiones distintas en cada uno de los entornos.



16.2. - Módulos

asgiref es un paquete de referencia para la especificación ASGI (Asynchronous Server Gateway Interface), que es una interfaz estándar para servidores web y aplicaciones web en Python. Proporciona una serie de utilidades y adaptadores para facilitar el desarrollo de aplicaciones web asíncronas con soporte para ASGI.

Documentación oficial: <https://asgiref.readthedocs.io/>

Django es un framework de desarrollo web de alto nivel y de código abierto, escrito en Python. Proporciona una estructura sólida y un conjunto de herramientas para simplificar el desarrollo de aplicaciones web complejas y escalables. Django se basa en el patrón de diseño MVC (Modelo-Vista-Controlador) y ofrece características como ORM (Object-Relational Mapping), enrutamiento de URLs, autenticación de usuarios, generación de formularios, administración de bases de datos y mucho más.

Documentación oficial: <https://docs.djangoproject.com/>

Tutorial de mozilla: <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction>

sqlparse es un analizador y formateador de SQL para Python. Permite analizar consultas SQL y dividirlos en componentes lógicos como palabras clave, identificadores, literales, etc. Además, puede formatear consultas SQL para mejorar su legibilidad al agregar sangría y espacios en blanco adecuados. sqlparse es útil para tareas como resaltar la sintaxis SQL en aplicaciones, depurar consultas SQL y generar consultas SQL legibles.

Documentación oficial: <https://sqlparse.readthedocs.io/>

tzdata es un módulo de Python que proporciona información sobre zonas horarias. Contiene una base de datos actualizada con información sobre zonas horarias de todo el mundo, como nombres de zonas horarias, desplazamientos de tiempo, reglas de horario de verano, entre otros. Este módulo es útil para trabajar con conversiones de tiempo y fechas en diferentes zonas horarias. Sin embargo, ten en cuenta que tzdata se utiliza principalmente como dependencia interna y es posible que no encuentres documentación específica para este módulo en sí.

16.3. - Preparación de estructura de trabajo

Creamos una carpeta que se llame mi_web y dentro de ella creamos el entorno virtual de la web:
virtualenv web

Debemos instalar django en el entorno virtual.

Pip install django

```
> pip install django
Collecting django
  Downloading Django-4.2.1-py3-none-any.whl (8.0 MB)
    0.0/8.0 MB 13.8 MB/s eta 0:00:00
Collecting asgiref<4,=>3.6.0 (from django)
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting sqlparse<0.3.1 (from django)
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
    41.2/41.2 kB 3.4 MB/s eta 0:00:00
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.6.0 django-4.2.1 sqlparse-0.4.4
> pip freeze
asgiref==3.6.0
django==4.2.1
sqlparse==0.4.4
~/Documents/proyectos/python/python_total/dia_16/mi_web on - main Py web
```

Se instalan 3 módulos, pero también instalamos tzdata.

pip install tzdata

Ahora creamos una carpeta que hará de fuente, que tradicionalmente es src

```
> tree -L 3
.
├── src
├── web
│   ├── bin
│   │   ├── activate
│   │   ├── activate.csh
│   │   ├── activate.fish
│   │   ├── activate.nu
│   │   ├── activate.ps1
│   │   ├── activate_this.py
│   │   ├── django-admin
│   │   ├── pip
│   │   ├── pip3
│   │   ├── pip-3.10
│   │   ├── pip3.10
│   │   ├── python -> /usr/bin/python3
│   │   ├── python3 -> python
│   │   ├── python3.10 -> python
│   │   ├── sqlformat
│   │   ├── wheel
│   │   ├── wheel3
│   │   ├── wheel-3.10
│   │   └── wheel3.10
│   ├── lib
│   │   └── python3.10
│   └── pyvenv.cfg
```

Dentro de la carpeta src iniciamos el proyecto django con:

django-admin startproject proyecto1

```
> django-admin startproject proyecto
> ls
proyecto
> tree
.
├── proyecto
│   ├── manage.py
│   └── proyecto
│       ├── asgi.py
│       ├── init_.py
│       ├── settings.py
│       ├── urls.py
│       └── wsgi.py
```

En «manage.py» se administra todo del proyecto.

Ahora iniciamos desde la carpeta proyecto el servidor para correr la web dentro:

```
python manage.py runserver
```

Indica que tenemos 18 migración sin aplicar

```

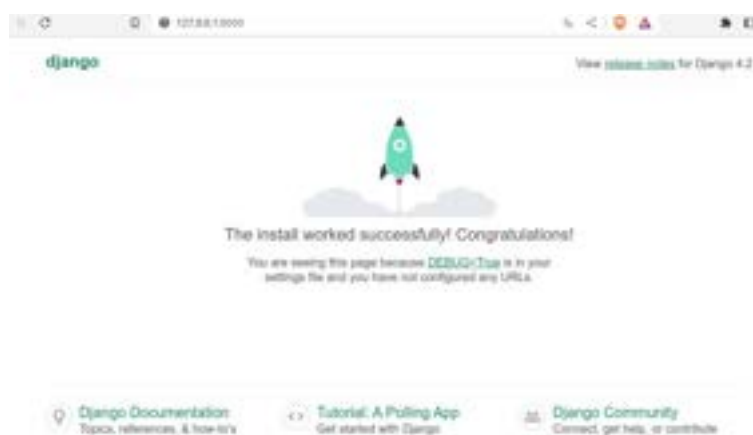
> python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migrations. Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
May 18, 2023 - 18:54:12
Django version 4.2.1, using settings 'proyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

```

También vemos el enlace donde podemos ver la web de django:



Antes de seguir, vamos a migrar los fichero pendientes con el comando:

```
python manage.py migrate
```

Y volvemos a ejecutar el servidor

```

> python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
> python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 18, 2023 - 21:24:23
Django version 4.2.1, using settings 'proyecto.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

```

Vemos que ya no tiene esos problemas de migración por aplicar.

En /admin tenemos la entrada del administrador.



Pero Django no tiene un nombre de usuario y contraseña por defecto establecidos. Cuando se crea un proyecto Django, se configura un archivo de configuración llamado "settings.py" donde se definen varias opciones, incluyendo la configuración de la base de datos.

En la configuración de la base de datos, se especifican las credenciales de acceso, como el nombre de usuario y la contraseña para acceder a la base de datos. Estas credenciales deben ser configuradas por el desarrollador según los requisitos del proyecto y la base de datos que se esté utilizando.

Por defecto, Django utiliza una base de datos SQLite, y en la configuración inicial de un proyecto, se establece un archivo de base de datos local. En este caso, no se requiere un nombre de usuario ni contraseña para acceder a la base de datos SQLite.

Sin embargo, es importante tener en cuenta que en un entorno de producción, es recomendable utilizar una base de datos más robusta como MySQL o PostgreSQL, y en esos casos, se deben configurar las credenciales de acceso correspondientes en la configuración de la base de datos de Django.

Para crear el superusuario lo hacemos con el comando:

```
python manage.py createsuperuser
```

```
> python3 manage.py createsuperuser
Username (leave blank to use 'v'):
Email address:
Password:
Password (again):
Superuser created successfully.
```

Voy a dejar que el usuario sea el de mi pc, el correo electrónico lo dejo vacío y es password, que tiene que ser mínimo de 8 caracteres, pongo LaDeSiempre.

Después de ingresar la clave ya veo un escritorio estilo CMS.

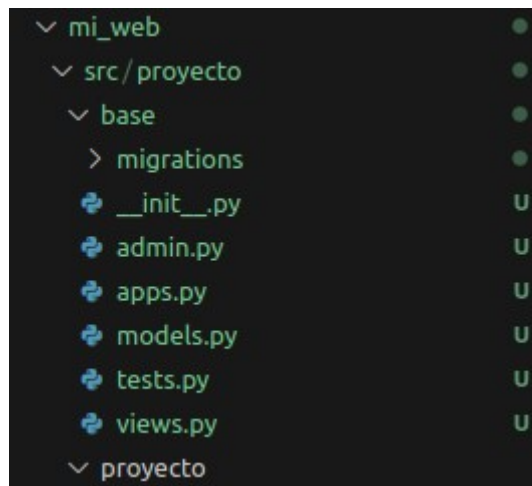


16.4. - Configurar url

Creamos el fichero de la app donde estará el núcleo de nuestro código, su estructura, los ajustes principales, su lógica principal, etc. Iniciamos la app con el nombre base:

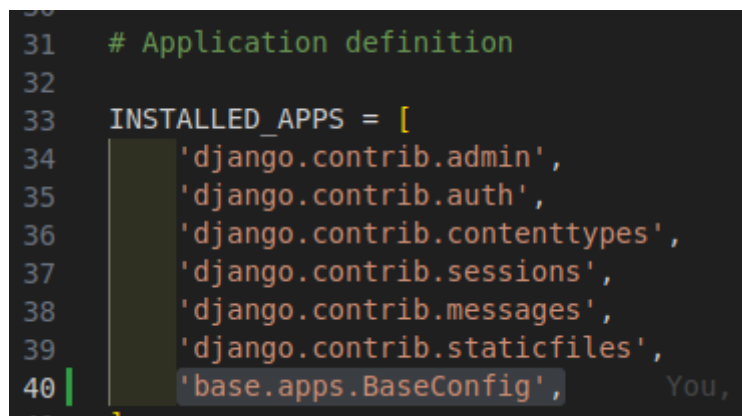
```
python manage.py startapp base
```

Esto creará otra estructura de carpetas



Para conectar "base" con "proyecto" vamos a ir al fichero proyecto/settings.py y en INSTALLED_APPS añadimos la línea para que conecte con la clase dentro de apps.py:

```
'base.apps.BaseConfig',
```



Ahora creamos el fichero urls.py en base e importamos las librerías necesarias y creamos una lista de urls:

```
from django.urls import path
from . import views

urlpatterns = []
```

Y en base/views.py añadimos esta línea:

```
from django.http import HttpResponse
```

Además creamos nuestra primera vista:

```
def lista_pendientes(pedido):
    return HttpResponse('Lista de pendientes')
```

Con lo que en la lista de urls de base/urls.py debemos añadirla:

```
urlpatterns = [
    path("", views.lista_pendientes, name='pendientes')
]
```

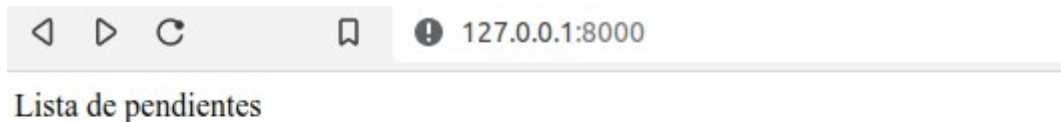
Pero para que el proyecto conozca esta url debemos añadir en proyecto/urls.py la función include:

```
from django.urls import path, include
```

Además de incluir el path en urlpatterns.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('base.urls')),
]
```

Ahora, en la misma url de django veremos el texto que incluimos en la vista:



16.5. - Crear tabla de tareas

Necesitamos que la base de datos almacena las tareas creadas por el usuario.

En Django, la base de datos por defecto que viene integrada se llama SQLite. SQLite es una base de datos ligera y de fácil configuración que se almacena en un archivo local en lugar de ejecutarse en un servidor separado. Esto hace que sea conveniente para el desarrollo y pruebas, ya que no requiere una configuración adicional del servidor de base de datos.

Al trabajar con Django, la configuración de la base de datos se especifica en el archivo settings.py de tu proyecto. Dentro de este archivo, encontrarás una sección llamada DATABASES que contiene la configuración de la base de datos por defecto. Para SQLite, la configuración típica se ve así:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

En este ejemplo, 'ENGINE': 'django.db.backends.sqlite3' indica que se utilizará el motor de base de datos SQLite, y 'NAME': BASE_DIR / 'db.sqlite3' especifica la ruta del archivo de base de datos, que se ubicará en el directorio del proyecto.

Es importante tener en cuenta que SQLite es adecuada para proyectos más pequeños o de desarrollo, pero para aplicaciones en producción con requisitos de alto rendimiento o concurrencia, es posible que desees considerar otras bases de datos como PostgreSQL, MySQL o Oracle, entre otras. En esos casos, deberás actualizar la configuración de la base de datos en el archivo settings.py para utilizar el motor y los detalles de conexión correspondientes a la base de datos que elijas.

Documentación bbdd django: <https://docs.djangoproject.com/en/3.2/topics/db/>

Para crear las tablas debemos hacerlo en base/models.py creando una clase que represente cada tabla. Sus atributos serán las columnas o los campos.

- En **usuario** especificamos el usuario en concreto mediante un módulo de django que tenemos que importar. Con una relación de 1:n. La función **ForeignKey** es una clave externa con la que podremos asociar usuarios que se repitan. La función responderá a:
 - **User** que es la biblioteca anterior
 - Definimos un atributo **on_delete** para cuando se elimine un usuario se elimine en cascada sus tareas.
 - **null y blank** en True para poder dejar este campo en blanco
- En **título** haremos que responda a **CharField** que es el campo de caracteres donde ajustamos el valor máximo de caracteres con max_length
- En **descripción** haremos que responda a **TextField** que es parecido al campo de caracteres anterior pero tiene algunos atributos extra que no tiene el anterior. Lo único que indicamos es que puede quedar vacío el campo.
- En **completo** vamos a añadir la función **BooleanField** de campo booleano que por defecto este False.

- En creado le vamos a indicar el momento en el que se creó la tarea con la función `DateTimeField` y con los atributos le indicamos que se autoconfigure con la fecha `now`

Ahora vamos a definir un valor `STR` que es el que nos va a reflejar como valor string si pedimos que imprima una tarea. Será el contenido de título.

Además le indicamos en **Meta** como se va a ordenar las tareas dentro de la tabla, que será por completo.

El documento queda así:

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.
class Tarea(models.Model):
    usuario = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        null=True,
        blank=True
    )
    titulo = models.CharField(max_length=200)
    descripcion = models.TextField(
        null=True,
        blank=True
    )
    completo = models.BooleanField(default=False)
    creado = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.titulo

class Meta:
```



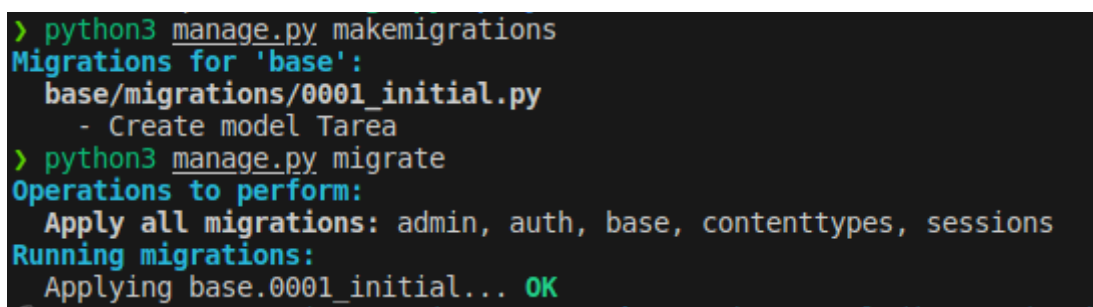
```
ordering = ['completo']
```

Pero ahora tenemos que migrar la tabla en la terminal, primero creando el fichero con:

```
python manage.py makemigrations
```

Esto creará una carpeta migrations en base. Dentro tendrá un fichero 0001_initial.py donde indica que esta preparado para migrar. Con el siguiente comando hacemos efectiva la migración:

```
python manage.py migrate
```

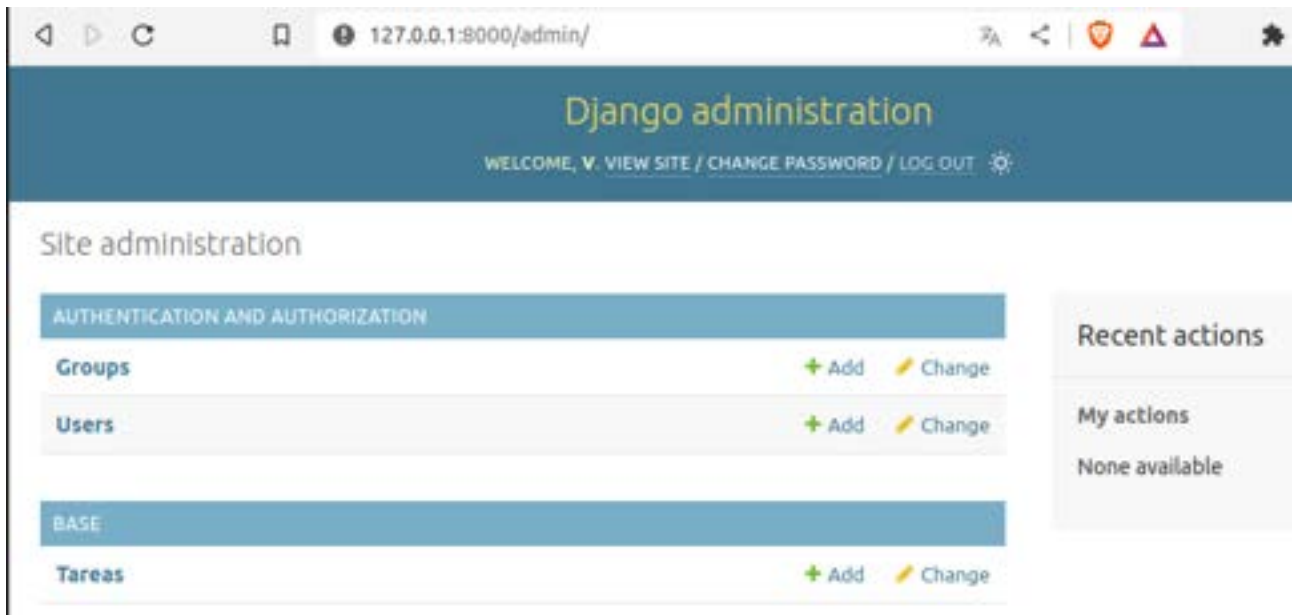


```
> python3 manage.py makemigrations
Migrations for 'base':
  base/migrations/0001_initial.py
    - Create model Tarea
> python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, base, contenttypes, sessions
Running migrations:
  Applying base.0001_initial... OK
```

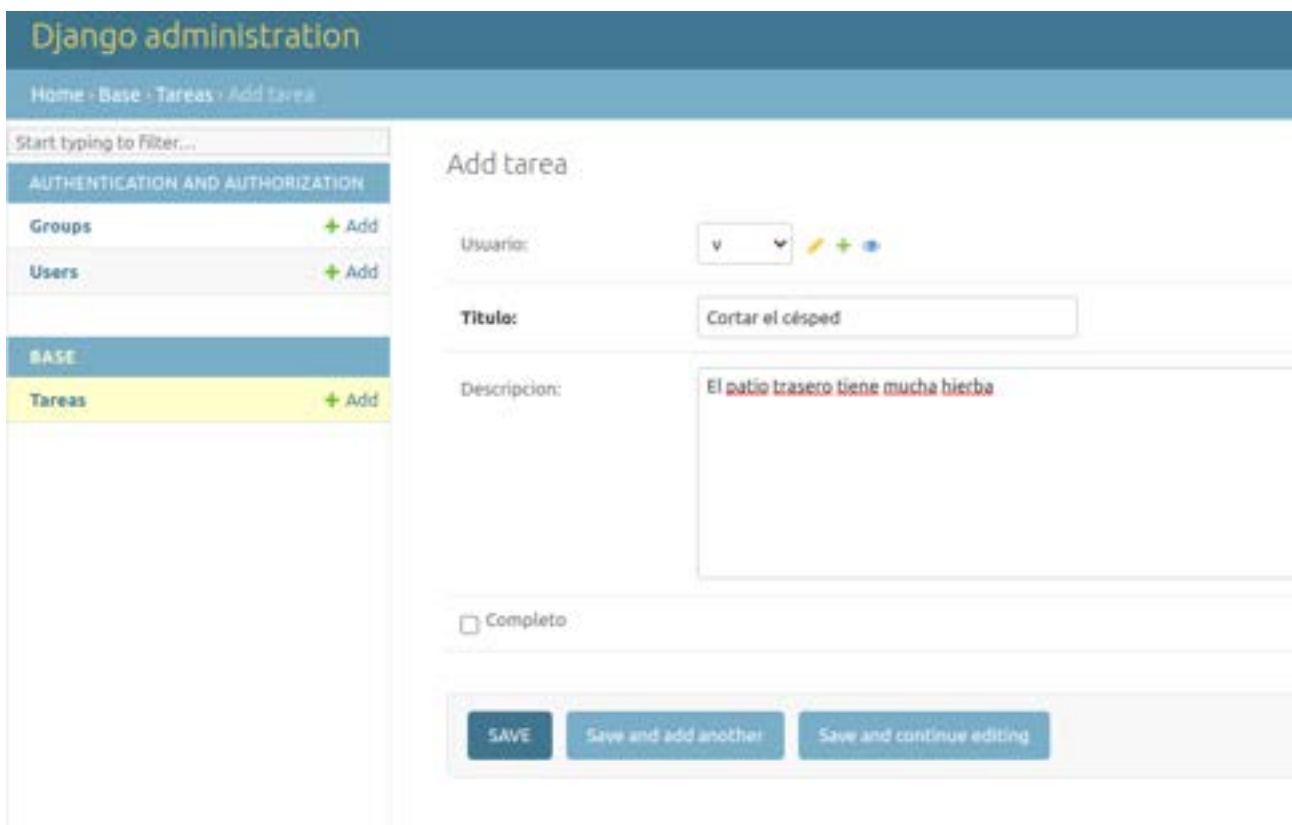
Ahora tenemos que registrar el modelo en base/admin.py . Tenemos que importar la función Tarea y añadir la tabla:

```
from django.contrib import admin
from .models import Tarea
# Register your models here.
admin.site.register(Tarea)
```

Con lo cual, si vamos a la url de nuestro sitio a /admin veremos que tenemos un nuevo campo de Tareas:



Si añadimos una nueva tarea veremos que nos permite añadir los campos que indicamos a nuestra tabla:



Hemos añadido tres tareas para las pruebas.

16.6. - Configurar la vista

Después de la prueba anterior vamos a traer una lista de objetos dinámica desde el fichero base/views.py.

Importamos ListView:

```
from django.views.generic.list import ListView
```

Creamos una clase que recoja la función. Para funcionar requiere de un modelo (lista de objetos completa) y un query set que haga la consulta filtrada de objetos. Pero tendremos que importar la Tarea de models:

```
from .models import Tarea
```

La clase queda así:

```
class ListaPendientes(ListView):  
    model = Tarea
```

Por ahora, el documento base/views.py lo tenemos así:

```
from django.shortcuts import render  
from django.views.generic.list import ListView  
from .models import Tarea  
# Create your views here.  
class ListaPendientes(ListView):  
    model = Tarea
```

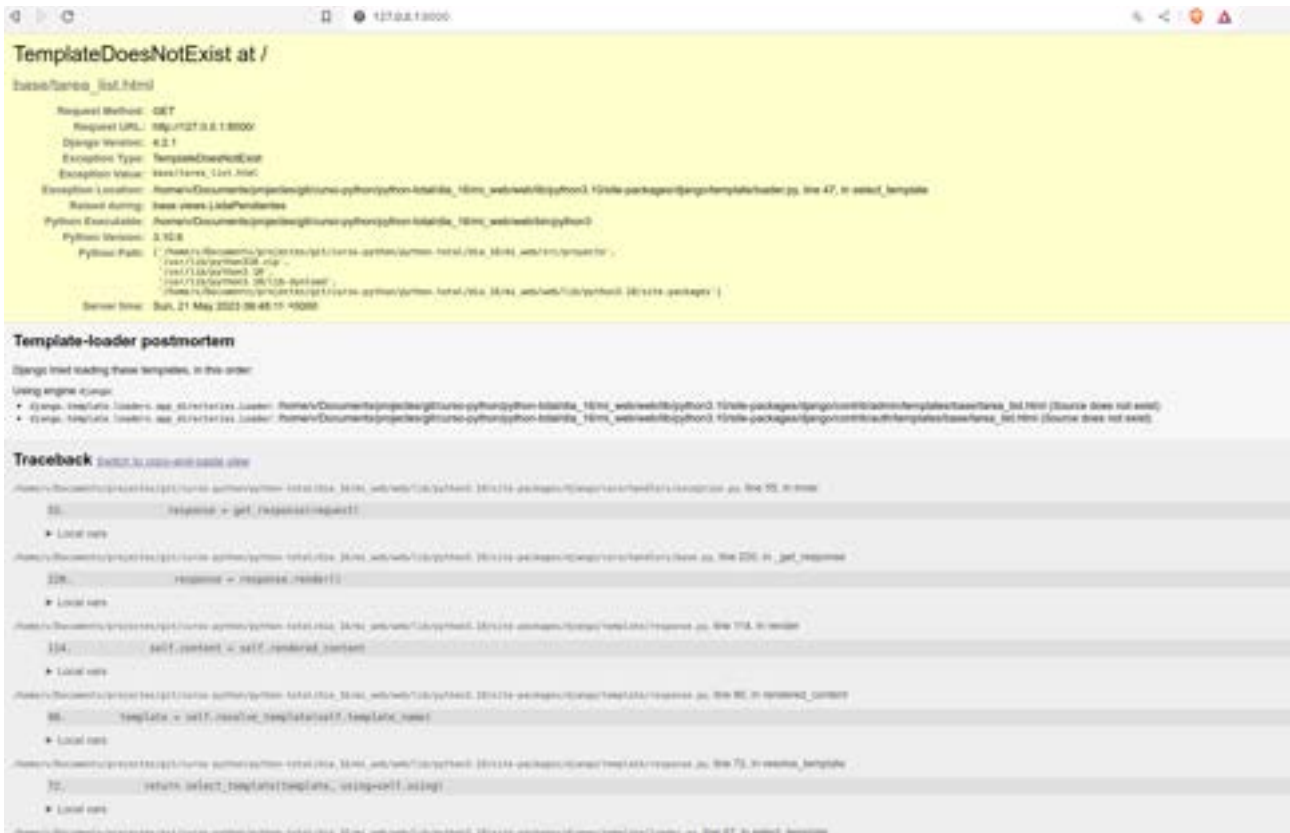
Ahora vamos a base/urls.py donde tendremos que indicar que importar la ListaPendientes:

```
from .views import ListaPendientes
```

También tenemos que cambiar el path añadiendo la ListaPendientes e indicando que lo lea como vista. El documento queda así:

```
from django.urls import path  
from .views import ListaPendientes  
urlpatterns = [  
    path("", ListaPendientes.as_view(), name='pendientes')  
]
```

Ahora ya hemos conectado la url a nuestra vista pero aun no hemos dicho como debe mostrarlo. Por eso, si entramos en la url nos da un error que nos indica que no existe el template, etc

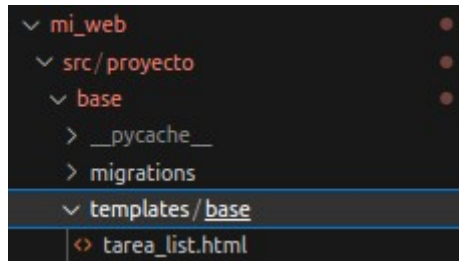


Entonces, para indicar las plantillas tenemos que ir a proyecto/settings.py e indicar el path de donde coger las plantillas en DIRS. El fragmento de código entero es el siguiente:

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['../base/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
    
```

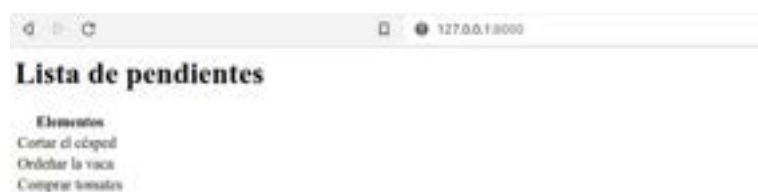
Vamos a crear la plantilla que nos dice, pero en una carpeta dentro de base que le llamaremos templates, y dentro con otro directorio que se llame base. Dentro crearemos el html que nos pide.



En este fichero vamos a añadir el texto html que queremos.

```
<h1>Lista de pendientes</h1>
<table>
  <tr>
    <th>Elementos</th>
  </tr>
  {% for tarea in object_list %}
  <tr>
    <td>{{ tarea.titulo }}</td>
  </tr>
  {% empty %}
  <h3>No hay elementos en esta lista</h3>
  {% endfor %}
</table>
```

Entonces, ya veremos nuestra lista en la url:



Para personalizar el `object_list` y que sea más legible, vamos a views y añadimos esta línea a la clase `ListaPendientes`:

```
context_object_name = 'areas'
```

Ahora ya lo podemos cambiar por `areas`.

16.7. - Configurar la vista de Detalle

En `base/views` vamos a importar la función para los detalles y crearemos una clase más. Quedando el fichero así:

```
from django.shortcuts import render
from django.views.generic.list import ListView
from django.views.generic.detail import DetailView
from .models import Tarea
# Create your views here.
class ListaPendientes(ListView):
    model = Tarea
    context_object_name = 'areas'
class DetalleTarea(DetailView):
    model = Tarea
```

Y necesitaremos crear otro fichero html como plantilla. Por defecto será `tarea_detail.html`, pero luego podemos ver como cambiar el nombre. Tendremos que incluir en `base/urls.py` el import y el path de la tarea, quedando el documento así:

```
from django.urls import path
from .views import ListaPendientes, DetalleTarea
urlpatterns = [
    path("", ListaPendientes.as_view(), name='pendientes'),
    path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea')
]
```

En el nombre del path se ha incluido `<int:pk>` que es el número entero de la Primary Key.

En el html creado añadimos lo siguiente:

```
<h1>Tarea: {{object}} </h1>
```

Esto hará que tenga el nombre de la tarea en cada url con un número de primary key válido:



Ahora podemos personalizar el object en base/views.py añadiendo a nuestra clase context_object_name con tarea y cambiando la palabra object por tarea del html.

Ahora vamos a personalizar el nombre del html de tarea_details.html por tarea.html y lo vamos a incluir en base/views.py, en la clase DetalleTarea, el template_name con la ubicación 'base/tarea.html'.

16.8. - Crear Links a Detalle

Simplemente, en la página de tarea_list.html incluimos el encabezado vacío de una nueva columna de la tabla:

```
<th></th>
```

E incluimos dentro del loop for una nueva columna con el texto Ver que contenga la lógica de la página, con el nombre tarea y el id de tarea:

```
<td><a href="{% url 'tarea' tarea.id %}">Ver</a></td>
```

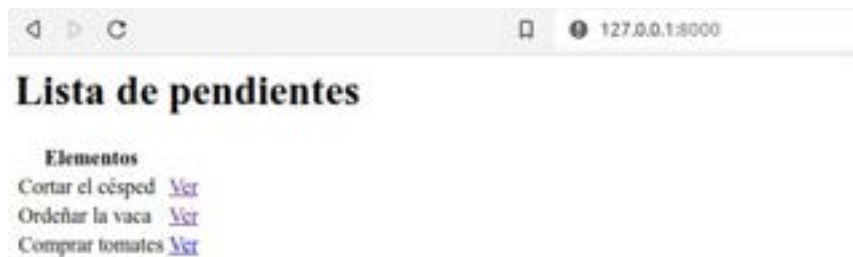
Quedando así el html:

```

<h1>Lista de pendientes</h1>
<table>
  <tr>
    <th>Elementos</th>
    <th></th>
  </tr>
  {% for tarea in tareas %}
  <tr>
    <td>{{ tarea.titulo }}</td>
    <td><a href="{% url 'tarea' tarea.id %}">Ver</a></td>
  </tr>
  {% empty %}
  <h3>No hay elementos en esta lista</h3>
  {% endfor %}
</table>

```

Y así la vista en el navegador:



16.9. - Agregar nueva tarea

Seguimos editando base/views.py y base/urls.py. En views.py importamos una clase más:

```
from django.views.generic.detail import DetailView
```

Creamos la clase CreaTarea que tendrá una lógica un poco más compleja porque el sistema que recoja un pedido que creará un nuevo elemento que se incluirá en la lista. Esta clase tomará un formulario por defecto de models.py basados en los campos que incluimos.

Se puede incluir en la clase una lista con todos los elementos que queremos en el formulario:


```
fields= ['titulo', 'descripcion', 'completo', 'creado']
```

Pero en este caso queremos todos los campos, así que utilizaremos

```
fields= '__all__'
```

Y para asegurarnos que cuando se envíe el formulario con éxito el usuario vaya a otra página distinta tenemos que importar otra herramienta:

```
from django.urls import reverse_lazy
```

`reverse_lazy` se ocupa de redirigir la página cuando encuentre el evento que se le indique. Cargamos la url donde se redirige con `success_url`

```
success_url = reverse_lazy('tareass')
```

Pero al poner `tareass` tenemos que asegurarnos que en el fichero `urls.py` tengamos lo mismo, que no es así. Tenemos «Pendientes», así que vamos a cambiar este nombre por `tareas` que tiene más lógica.

De nuevo, tendremos que añadir la nueva página en el fichero `urls.py`, importando la clase y añadiendo el `path`. Quedando el documento así:

```
from django.urls import path
```

```
from .views import ListaPendientes, DetalleTarea, CrearTarea
```

```
urlpatterns = [
```

```
    path("", ListaPendientes.as_view(), name='tareass'),
```

```
    path('tarea/<int:pk>', DetalleTarea.as_view(), name='tarea'),
```

```
    path('crear-tarea/', CrearTarea.as_view(), name='crear-tarea')
```

```
]
```

Ahora necesitaremos un nuevo fichero `html` en `templates` que le llamaremos `tarea_form.html` que es el nombre por defecto. Para las pruebas tan solo ponemos un título:

```
<h1>Formulario de tareas</h1>
```

Y por último, para que la página principal tenga un enlace que redirija a la página nueva, tendremos que editar `tareas_list.html` añadiendo este fragmento de código `html`:

```
<a href="{% url 'crear-tarea' %}">Crear nueva tarea</a>
```

Ahora ya podemos ver el enlace que nos lleva a la nueva página creada:



16.10. - Formulario para nueva tarea

Vamos a darle brillo al html del formulario "tarea_form.html". Los campos del formulario se añaden automáticamente con `{{form}}`, pero tenemos que darle un método para que se vea mejor. Por defecto es `as_table`:



Horrible, pero tenemos dos métodos más. Formato lista con `as_ul`:



O como párrafo, `as_p`, que es el que dejaremos:



Ya podemos probarlo, pero tendremos un error. Nos falta un fragmento de código:

```
{% csrf_token %}
```

El código `{% csrf_token %}` es una directiva utilizada en algunas plantillas o sistemas de generación estática de sitios web, como Hugo o Jekyll.

Esta directiva se utiliza para generar un token de seguridad [CSRF](#) (Cross-Site Request Forgery) y se inserta en un formulario HTML. El token CSRF ayuda a prevenir ataques de falsificación de solicitudes entre sitios, asegurando que las solicitudes enviadas al servidor sean legítimas y no provengan de fuentes maliciosas.

El token se genera y se incluye en el formulario para que cuando el usuario envíe el formulario, el servidor pueda verificar que el token es válido y corresponde a la sesión actual del usuario. Esto ayuda a proteger contra ataques en los que un tercero intenta enviar solicitudes maliciosas en nombre del usuario.

Ahora ya podemos crear tareas desde la página creada y que nos reenvíe a la página principal

Para rematar, vamos a poner un enlace para volver desde la página del formulario a la página principal:

```
<a href="{% url 'tareas' %}">Volver</a>
```



Formulario de tareas

[Volver](#)

Usuario:

Titulo:

Descripcion:

Completo:

16.11. - Editar tarea

En `base/views.py` vamos a importar una nueva clase:

```
from django.views.generic.edit import CreateView, UpdateView
```

Y añadimos una nueva clase:

```
class EditarTarea(UpdateView):
    model = Tarea
    fields = '__all__'
    success_url = reverse_lazy('tareass')
```

Y podemos ir a base/urls.py para añadir una nueva página. Importamos EditarTarea y añadimos el path:

```
path('editar-tarea/<int:pk>', EditarTarea.as_view(), name='editar-tarea')
```

Entonces, el html de editar tarea tiene que tener una nueva columna con el hipervínculo de modificar la tarea. Tenemos que añadir la celda de cabecera <th></th> y la del enlace <td>Editar</td>

Dará este resultado:

Lista de pendientes

```
Crear nueva tarea
Elementos
Cortar el césped Ver Editar
Ordeñar la vaca Ver Editar
Comprar tomates Ver Editar
```

16.12. - Eliminar tarea

Esto será muy parecido a lo anterior. Vayamos por documentos.

Añadimos en tarea_list.html:

```
<th></th>
```

y:

```
<td><a href="{% url 'eliminar-tarea' tarea.id %}">Eliminar</a></td>
```

En views.py importamos la clase DeleteView

```
from django.views.generic.edit import CreateView, UpdateView, DeleteView
```

y añadimos una nueva clase:

```
class EliminarTarea(DeleteView):
    model = Tarea
    context_object_name = 'tarea'
    success_url = reverse_lazy('tareas')
```

Ahora abrimos un nuevo html que le llamaremos como el sistema lo marca por defecto, tarea_confirm_delete.html y añadimos:

```
<a href="{% url 'tareas' %}">Volver</a>

<form method="post" action="">
    {% csrf_token %}
    <p>Vas a eliminar esta tarea: "{{ tarea }}"</p>
    <input type="submit" value="Eliminar">
</form>
```

Ahora nos queda importar EliminarTarea en urls.py y añadir el path:

```
path('eliminar-tarea/<int:pk>', EliminarTarea.as_view(), name='eliminar-tarea')
```

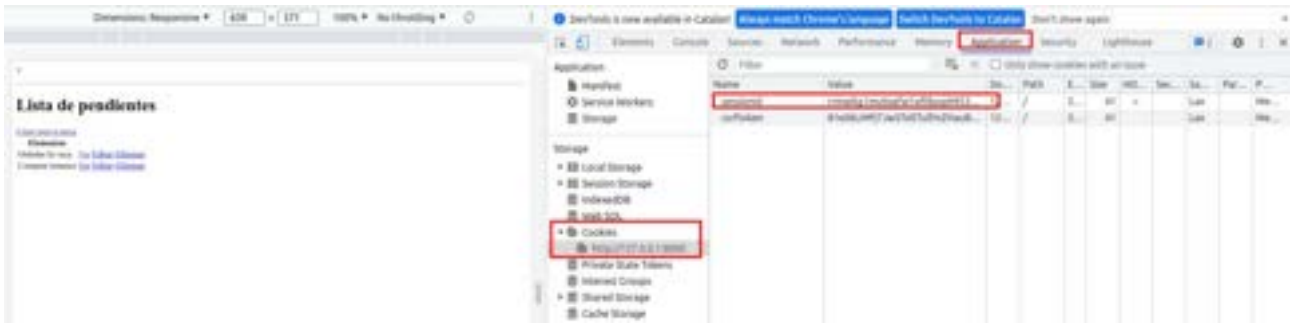
Ya podemos eliminar tareas

16.13. - Crear la lógica de Logueo / Deslogueo

Vamos a adaptar el fichero principal tarea_list.html para que tenga una condicional que detecte si el usuario está logueado o que enlace una página de logueo. También incluimos una barra horizontal para separarlo del contenido.

```
<p>
    {{ request.user }}
</p>
<hr>
```

Podemos ver el logueo con "Inspeccionar elemento" / Application /cookies / sessionid



Si lo borramos nos deslogueamos y aparecerá como usuario anonimo.



Vamos a poner el anterior fragmento de código dentro de un if, quedando así:

```
{% if request.user.is_authenticated %}
<p>
    {{request.user}}
</p>
<a href="{% url 'logout' %}">Salir</a>
{% else %}
<a href="{% url 'login' %}">Iniciar sesión</a>
{% endif %}
```

En el siguiente punto veremos como construir el formulario de logueo.

16.14. - Formulario de Logueo / Deslogueo

Nos vamos a views.py, importamos la siguiente vista:

```
from django.contrib.auth.views import LoginView
```

Y añadimos la siguiente clase, que tiene sentido que este arriba del todo, y dentro de la clase definimos el nombre del template, que queremos todos los campos que contiene LoginView, activar la redirección después de la autenticación y redefinimos la función `get_success_url` para que vaya a la página principal después del loguearse:

```
class Logueo(LoginView):
    template_name = "base/login.html"
    field = '__all__'
    redirect_authenticated_user = True
    def get_success_url(self):
        return reverse_lazy('tarear')
```

Ahora vamos a configurar las `urls.py` para que llegue al path. Importamos `Logueo` y añadimos el path:

```
path('login/', Logueo.as_view(), name='login'),
```

Creamos el fichero en `login.html` en `template/base` con el siguiente contenido:

```
<h1>Ingresar</h1>

<form method="post" action="">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Ingresar">
</form>
```

Para desloguear vamos a usar directamente el formulario que nos brinda django. En `urls.py` importamos la siguiente clase:

```
from django.contrib.auth.views import LogoutView
```

y añadimos el path:

```
path('logout/', LogoutView.as_view(next_page='login'), name='logout'),
```

16.15. - Restringir acceso

Nos vamos a views.py e importamos nueva clase:

```
from django.contrib.auth.mixins import LoginRequiredMixin
```

Con esta clase podríamos gestionar los atributos de un determinado usuario, si va a ser administrador, si va a tener alguna otra autorización especial, si va a ser un usuario común. Etc También la restricción de las vistas.

Tendremos que incluirlo en las clase que queramos que herede esta opción, como en ListaPendientes, en DetalleTarea y en definitiva, en todas las clases menos en la de logueo.

Pero si ahora intentamos entrar sin estar logueados nos saldrá un página de error de django. Para personalizarla tenemos que ir a settings.py y vamos a incluir esta línea de código antes de Static Files:

```
LOGIN_URL = 'login'
```

Así nos desviará a la página de logueo sin no estamos registrados.

16.16. - Información específica de usuario

Cada usuario debe ver tan solo sus tareas, para eso vamos a añadir un nuevo usuario en el panel de administración de django. Así que en views.py vamos añadir el método GetContentData y sobrescribirlo en la clase de ListaPendientes:

```
def get_context_data(self, **kwargs):
    context = super().get_context_data(**kwargs)
    context['tareas'] = context['tareas'].filter(usuario=self.request.user)
    context['count'] = context['tareas'].filter(completo=False).count()
    return context
```

Se puede ver dos filtros al contexto, el primero es para que solo se vea las tareas del usuario registrado y el segundo es para que solo se vean las tareas que no estén completas.

Ahora, con el nuevo usuario creado no veremos las tareas del anterior usuario:

Pedro

[Salir](#)

Lista de pendientes

[Crear nueva tarea](#)

No hay elementos en esta lista

Elementos

Ahora, si un usuario crea una tarea queremos que se le asigne a si mismo, así que vamos a views.py y en la clase de CrearTarea redefinimos la función de form_valid:

```
def form_valid(self, form):  
    form.instance.usuario = self.request.user  
    return super(CrearTarea, self).form_valid(form)
```

Y en la misma clase, donde tenemos que aparezcan todos los campos con `__all__` tenemos que dar la lista para que no de la opción de escoger el usuario. También lo tendremos que hacer en EditarTarea:

```
fields = ['titulo', 'descripcion', 'completo']
```

Formulario de tareas

[Volver](#)

Título:

Descripcion:

Completo:

16.17. - Registrar nuevo usuario

Siempre vamos cambiando lo mismo: Enlace, formulario, vista y urls.

Empezamos por el enlace en la login.html:

```
<p>No tienes cuenta<a href="{% url 'registro' %}">Registrate</a></p>
```

Ahora vamos a crear un nuevo fichero en templates que se llame registro.html:

```
<h1>Registro</h1>
```

```
<form method="post" action="">
```

```
    {% csrf_token %}
```

```
    {{ form.as_p }}
```

```
    <input type="submit" value="Registrar">
```

```
</form>
```

```
<p>Ya tienes cuenta?<a href="{% url 'login' %}">Registrate</a></p>
```

Ahora creamos una vista para esto en views.py. No hay una vista específica en Django para crear registros, pero podemos aprovechar una vista genérica. Tenemos que añadir 3 clases/métodos:

```
from django.views.generic.edit import CreateView, UpdateView, DeleteView, FormView
```

```
from django.contrib.auth.forms import UserCreationForm
```

```
from django.contrib.auth import login
```

Y añadimos la clase PaginaRegistro:

```
class PaginaRegistro(FormView):  
    template_name = 'base/registro.html'  
    form_class = UserCreationForm  
    redirect_authenticated_user = True  
    success_url = reverse_lazy('areas')
```

Ahora podemos crear el path en la url, antes importando la clase

```
path('registro/', PaginaRegistro.as_view(), name='registro'),
```

Ahora mismo tenemos el formulario en inglés:

Registro

Username: Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation: Enter the same password as before, for verification.

Ya tienes cuenta? [Registrate](#)

Pero lo podemos cambiar al castellano o a muchos otros idiomas en settings.py:

```
LANGUAGE_CODE = 'es-es'
```

Cuando actualicemos la página de registro lo tendremos en español:

Registro

Nombre de usuario: Requerido, 150 caracteres como máximo. Únicamente letras, dígitos y @/./+/-/_

Contraseña:

- Su contraseña no puede asemejarse tanto a su otra información personal.
- Su contraseña debe contener al menos 8 caracteres.
- Su contraseña no puede ser una clave utilizada comúnmente.
- Su contraseña no puede ser completamente numérica.

Contraseña (confirmación): Para verificar, introduzca la misma contraseña anterior.

Ya tienes cuenta? [Registrate](#)

Para asegurarnos que se loguee el usuario una vez se registre, tenemos que ir a views para sobrescribir PaginaRegistro con una función:

```
def form_valid(self, form):
    usuario = form.save()
```

```

if usuario is not None:
    login(self.request, usuario)
return super(PaginaRegistro, self).form_valid(form)

```

Para asegurarnos de que un usuario registrado no pueda entrar en la página de registro de nuevo, vamos a modificar de nuevo la clase. Con esto último, la clase entera queda así:

```

class PaginaRegistro(FormView):
    template_name = 'base/registro.html'
    form_class = UserCreationForm
    redirect_authenticated_user = True
    success_url = reverse_lazy('areas')
    def form_valid(self, form):
        usuario = form.save()
        if usuario is not None:
            login(self.request, usuario)
        return super(PaginaRegistro, self).form_valid(form)
    def get(self, *args, **kwargs):
        if self.request.user.is_authenticated:
            return redirect('areas')
        return super(PaginaRegistro, self).get(*args, **kwargs)

```

16.18. - Barra de búsquedas

Para las pruebas necesitamos varias tareas que contengan una misma palabra.

Entonces, en `tarea_list.html` tenemos que añadir otro form. Como es para obtener información pondremos el método `get`:

```

<form method="get" action="">
    {% csrf_token %}
    <input type="text" name="area-buscar">
    <input type="submit" value="Buscar">
</form>

```

Ahora en views.py vamos a incorporar otro filtro en la clase ListaPendientes, dentro de get_context_data. Hacemos una variable para guardar la cerca que se ingrese en el formulario anterior y antes de devolver el contexto, cambiaremos el filtro del contexto para chequear el título contengan el valor, con un if para que se ejecute si la variable es verdadera:

```
valor_buscado = self.request.GET.get('area-buscar') or ""
if valor_buscado:
    context['tarefas'] = context['tarefas'].filter(
        titulo__icontains=valor_buscado)
```

Ahora ya funciona, he buscado vaca:

Lista de pendientes

[Crear nueva tarea](#)

Elementos

Ordeñar la vaca	Ver Editar Eliminar
Peinar la vaca	Ver Editar Eliminar
vaca a pelo completo	Ver Editar Eliminar

Para mantener la palabra buscada en el formulario debe hacer algún cambio más. Tenemos que añadir esta línea a la anterior función, antes del return:

```
context['valor_buscado'] = valor_buscado
```

Y al input del formulario, el de la caja de búsqueda, tenemos que añadir el valor de la variable creada:

```
value="{{ valor_buscado }}"
```

Ahora ya se queda la palabra:

Lista de pendientes

[Crear nueva tarea](#)

Elementos

Ordeñar la vaca [Ver](#) [Editar](#) [Eliminar](#)
 Peinar la vaca [Ver](#) [Editar](#) [Eliminar](#)
 vaca a pelo completo [Ver](#) [Editar](#) [Eliminar](#)

16.19. - Un estilo para todas las vistas

Vamos a crear un estilo para todas las vistas.

Para empezar, vamos a quitar la columna "ver" que ya no necesitamos. Tan solo de tarea_list.html eliminamos su línea:

```
<td><a href="{% url 'tarea' tarea.id %}">Ver</a></td>
```

Ahora necesitamos un html completo, con una estructura básica, que le vamos a llamar principal.html y es donde vamos a incluir los estilos. Por ahora le damos un color al fondo para probar y dentro de un div creamos la estructura:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>Lista de Pendientes</title>
    <style>
      body {
        background-color: aqua;
      }
    </style>
  </head>
```

```

<body>
  <div class="container">
    {% block content %}
    {% endblock content %}
  </div>
</body>
</html>

```

Ahora vamos a tarea_list.html para indicar que tenga de base el anterior html:

```

{% extends 'base/principal.html' %}
{% block content %}

{% endblock content %}

```

Dentro de "block content" se debe incluir el contenido de toda la página tarea_list.html y ahora ya veremos el color aqua de background:



Ahora tenemos que repetir la operación en cada una de las páginas para que principal.html tenga el estilo de todas.

16.20. - Estilo general

Vamos a coger una fuente de [google font](#) y añadir el link en el head de principal, justo antes de style:

```
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Abel&display=swap" rel="stylesheet">
```

Y añadimos al estilo de body la fuente:

```
font-family: 'Abel', sans-serif;
```

Vamos a añadir color con ayuda de una paleta online: <https://html-color.codes/> y cambiamos el color.

También le añadimos un padding top.

Y ahora creamos otro encapsulamiento css en .container, que es la clase que le dimos al div de principal donde contiene el resto de páginas. A este le damos un tamaño máximo, un margen automático, un color de background, una sombra,

Le damos estilo a los encabezados (h1,h2,...), con una fuente distinta y lo centramos.

Le damos un color distinto a los enlaces y quitamos el subrayado.

En definitiva, hemos incluido esto en style:

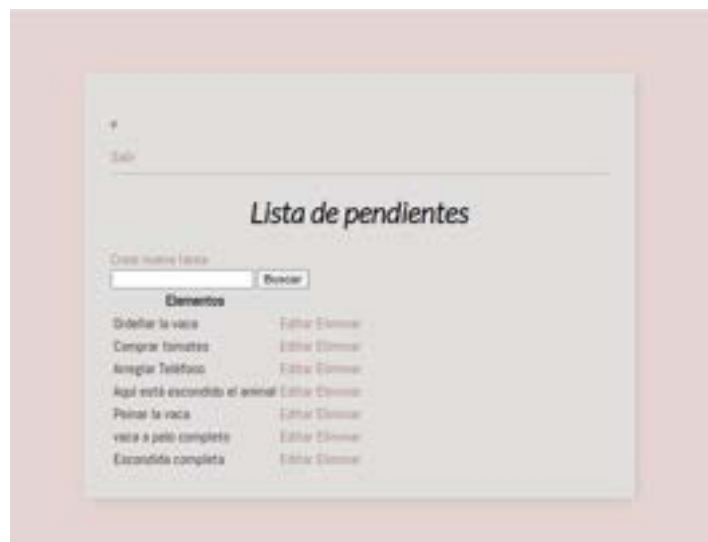
```
<style>
  body {
    background-color: #E5D4D1;
    font-family: 'Abel', sans-serif;
    padding-top: 60px;
  }
  .container {
    max-width: 550px;
    margin: auto;
    background-color: #E1DFDE;
    -webkit-box-shadow: 2px 2px 13px -4px rgba(0,0,0,0.21);
```

```

        box-shadow: 2px 2px 13px -4px rgba(0,0,0,0.21);
        padding: 30px;
    }
    h1,h2,h3,h4,h5,h6 {
        font-family: 'Lato', sans-serif;
        text-align: center;
    }
    a {
        text-decoration: none;
        color: #996E66;
    }
</style>

```

Y por ahora queda así:



16.21. - Estilo de barra superiores

En `tarea_list.html`, añadimos un `div` al principio del "block content" con un `h1` de saludo al usuario (El nombre se transforma con `title`, para que la primera letra sea mayúscula) y un `h3` que muestre las tareas incompletas (Lo muestra en singular y en plural con `{{count|pluralize}}`). Tendremos que añadir en el `div` el bloque de usuario que ya construimos. Queda así:

```
<div class="barra_superior">
```

```

<div>
  <h1>Hola {{request.user|title}}</h1>
  <h3 style="margin:0">Tienes <i>{{count}}</i> tarea{{count|pluralize}} incompleta{{count|pluralize}}</h3>
</div>
{% if request.user.is_authenticated %}
<p>
  {{request.user}}
</p>
<a href="{% url 'logout' %}">Salir</a>
{% else %}
<a href="{% url 'login' %}">Iniciar sesión</a>
{% endif %}
</div>

```

Ahora vamos a darle estilo en la página principal.html para que se aplique a la clase del div:

```

.barra_superior {
  display: flex;
  justify-content: space-between;
  color: #fff;
  padding: 10px;
  border-radius: 5px 5px 0 0;
  background: linear-gradient(90deg, #1e7dcb 0%, #547482 65%, #77c2e2 100%);
}
.barra_superior a {
  color: #fff;
}

```

Por ahora tenemos esto:



Ha cambiado alguna cosa del estilo anterior ;) CSS: Pinta y colorea

16.22. - Estilo de la lista

En tarea_list.html eliminamos el hr y el h1 de Lista de pendientes.

Por ahora comentamos el código del enlace de crear nueva tarea y del formulario de búsqueda.

Vamos así:



Ahora vamos a eliminar la tabla, pero antes vamos a ubicarlo en otro div con una nueva clase.

Vamos a meter el loop for de la tareas, tan solo el principio y el final, con el empty. Dentro ponemos otro div con una nueva clase. Dentro añadimos las tareas completas con un if y otro div.

Por ahora, este código descrito es esto:

```
<div class="envoltorio-items-tarea">
    {% for tarea in tareas %}
    <div class="envoltorio-tarea">
        {% if tarea.completo %}
```

```

    <div class="titulo-tarea">
    </div>
</div>
{% empty %}
<h3>No hay elementos en esta lista</h3>
{% endfor %}
</div>

```

Ahora, dentro del div con la clase titulo-tarea hacemos una estructura de divs en la que por ahora ponemos tareas completas con la url de la tarea en cursiva y tachado.

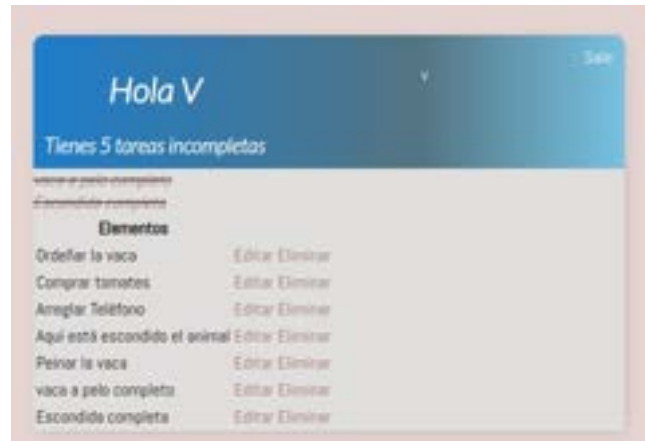
Tenemos este código:

```

<div class="envoltorio-items-tarea">
    {% for tarea in tareas %}
    <div class="envoltorio-tarea">
        {% if tarea.completo %}
        <div class="titulo-tarea">
            <div class="icono-tarea-completa">
            </div>
            <i><s><a href="{% url 'editar-tarea' tarea.id %}">{{ tarea }}</a></s></i>
        </div>
        {% endif %}
    </div>
    {% empty %}
    <h3>No hay elementos en esta lista</h3>
    {% endfor %}
</div>

```

Y se ve esto:



Completamos con un else la tarea no completa, que será igual que la anterior pero cambiando la clase a incompleta y quitando la cursiva y el tachado:

```
{% else %}
<div class="titulo-tarea">
  <div class="icono-tarea-incompleta">
  </div>
  <a href="{% url 'editar-tarea' tarea.id %}">{{tarea}}</a>
</div>
```

Ahora vamos a incluir el icono de un x para eliminar las tareas. Aprovechando el enlace que ya teníamos de eliminar tarea pero le vamos a dar clase al enlace y le añadimos el icono x de esta web: <https://www.htmlsymbols.xyz/>

Escogemos estas aspas: <https://www.htmlsymbols.xyz/unicode/U+2A2F>

Se tendrá que poner antes del endif y del else para que los dos tipos de tarea (completa|incompleta) tengan el aspa:

```
<a class="enlace--eliminar" href="{% url 'eliminar-tarea' tarea.id %}">&#x2A2F;</a>
```

Ahora ya podemos eliminar toda la tabla.

Vamos a darle estilo:

```
.envoltorio-tarea {
  display: flex;
```

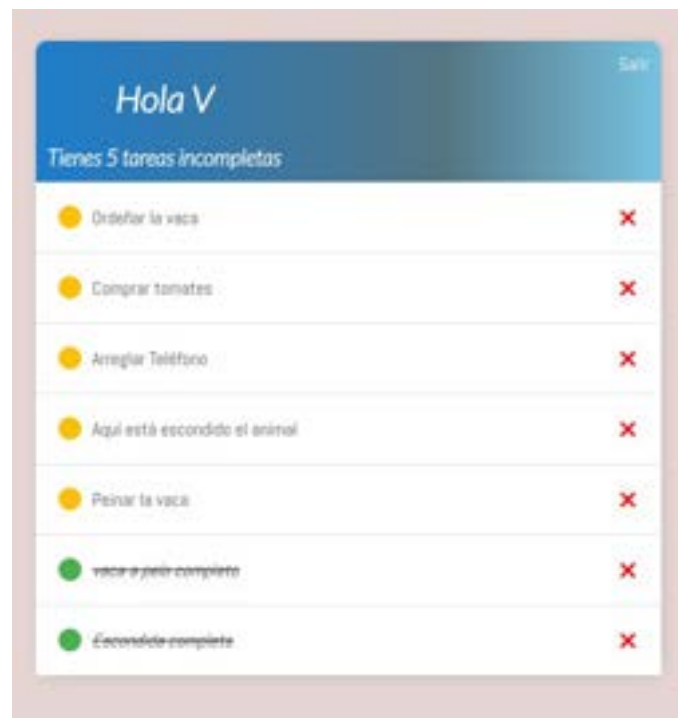
```
    align-items: center;
    justify-content: space-between;
    padding: 20px;
    background-color: #fff;
    border-top: 1px solid rgb(225,225,225);
}
.titulo-tarea {
    display: flex;
}
.titulo-tarea a {
    text-decoration: none;
    color: #4B5156;
    margin-left: 10px;
}
.icono-tarea-completa {
    width: 20px;
    height: 20px;
    background-color: #4CAF50;
    border-radius: 50%;
}
.icono-tarea-incompleta {
    width: 20px;
    height: 20px;
    background-color: #FFC107;
    border-radius: 50%;
}
.enlace-eliminar {
    text-decoration: none;
    font-weight: 900;
```

```

color: #FF0000;
font-size: 22px;
line-height: 0;
}

```

Queda así:



Verás que voy corrigiendo otros errores...

16.23. - Estilo de la barra de búsqueda

Vamos a descomentar el form de busquedas para añadirlo también.

Para darle estilo en vez de clase le damos un id ya que será solo una vez que se usará. Metemos el buscador y agregar nueva tarea en un div con id.

Le damos estilo al form con el parámetro:

```
style="margin-top: 20px;display: flex;
```

Al enlace de crear tarea le damos también un id. Y el texto también lo cambiamos por un signo:

<https://www.htmlsymbols.xyz/unicode/U+002B>

Ahora a darle estilo:


```
#envoltorio-agregar-buscar {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  padding: 10px;  
}
```

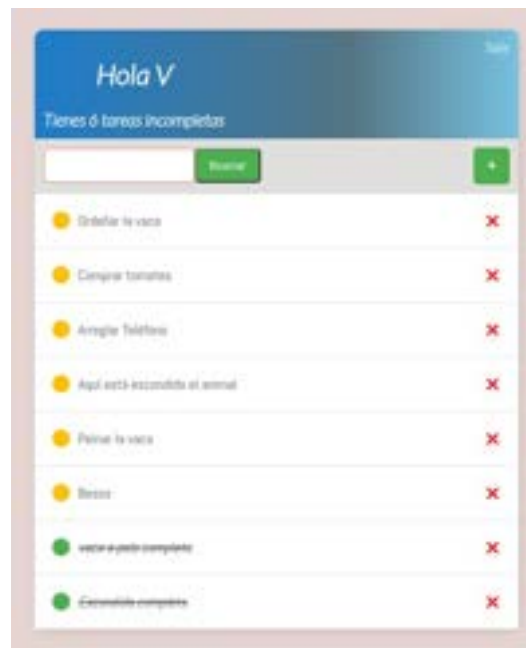
```
#enlace-agregar {  
  text-decoration: none;  
  color: #fff;  
  background-color: #4CAF50;  
  padding: 10px 15px;  
  border-radius: 5px;  
  font-size: 1.1em;  
  font-weight: 900;  
}
```

Ahora para darle estilo al botón de buscar tenemos que añadirle una clase al input. Y volvemos a principal a darle estilo:

```
input[type=text],  
input[type=password],  
textarea {  
  padding: 10px;  
  border-radius: 5px;  
  border: 1px solid #EB796F;  
  width: 90%;  
  font-size: 1.1em;  
  font-weight: 900;  
}
```

```
label {  
    padding-top: 10px !important;  
    display: block;  
    font-size: 1.1em;  
    font-weight: 900;  
}  
  
.boton {  
    text-decoration: none;  
    color: #fff;  
    background-color: #4CAF50;  
    padding: 10px 15px;  
    border-radius: 5px;  
    font-size: 1.1em;  
    font-weight: 900;  
}
```

Nos queda así:



16.24. - Terminar el sitio

Vamos a tareas_form.html para poner un enlace que permita volver al inicio:

```
<div class="barra_superior">
  <a href="{% url 'tareas' %}">⬅; Volver</a>
</div>
```

Le agregamos una flecha con un icono html.

En el input del boton incluimos la clase creada al botón de la página principal.

El formulario completo lo metemos en un div con otra clase para darle estilo:

```
{% extends 'base/principal.html' %}
{% block content %}
```

```
<div class="barra_superior">
  <a href="{% url 'tareas' %}">⬅; Volver</a>
</div>
```

```
<div class="cuerpo-tarjeta">
  <form method="post" action="">
    {% csrf_token %}
    {{ form.as_p }}
    <input class="" type="submit" value="Enviar">
  </form>
</div>
```

```
{% endblock content %}
```

A darle estilo:

```
.cuerpo-tarjeta {
  padding: 10px 20px;
}
```

Queda así:

The image shows a web form with a blue header bar containing a back arrow and the text 'Volver'. Below the header, there are three main sections: a 'Titulo:' label followed by a text input field; a 'Descripcion:' label followed by a larger text area; and a 'Completo:' label followed by a checkbox. At the bottom of the form is a button labeled 'Enviar'.

Vamos a `tarea_confirm_delete.html` y copiamos lo que hemos hecho anteriormente para volver. Y encapsulamos el form en un `div` con la misma clase que antes, también en el `input` de submit:

```
{% extends 'base/principal.html' %}
{% block content %}
<div class="barra_superior">
  <a href="{% url 'tarefas' %}">&#x2190; Volver</a>
</div>
<div class="cuerpo-tarjeta">
  <form method="post" action="">
    {% csrf_token %}
    <p>Vas a eliminar esta tarea: "{{ tarea }}"</p>
    <input class="boton" type="submit" value="Eliminar">
  </form>
</div>
{% endblock content %}
```

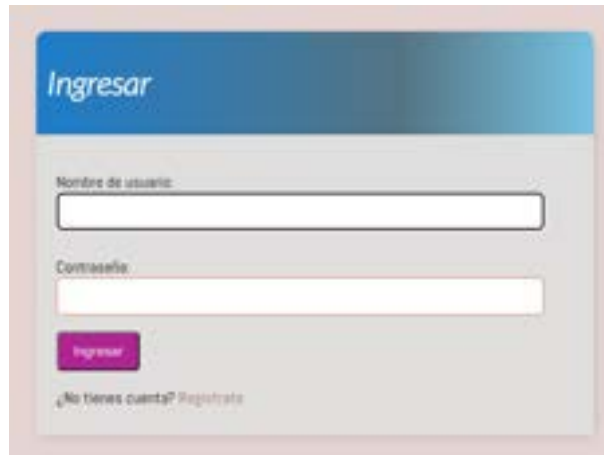
Ya tiene el estilo, lo único que le doy otro color al botón:



Ahora le toca a login.html. Creamos el div de la clase barra_superior pero sin volver, esta vez ponemos el h1 de la página. Metemos el form y la pregunta ¿Tienes cuenta? en un div con clase cuerpo-tarjeta y, por último, le damos la clase al input submit

```
{% extends 'base/principal.html' %}
{% block content %}
<div class="barra_superior">
  <h1>Ingresar</h1>
</div>
<div class="cuerpo-tarjeta">
  <form method="post" action="">
    {% csrf_token %}
    {{ form.as_p }}
    <input class="boton" type="submit" value="Ingresar">
  </form>
  <p>¿No tienes cuenta?<a href="{% url 'registro' %}">Registrate</a></p>
</div>
{% endblock content %}
```

Queda así:

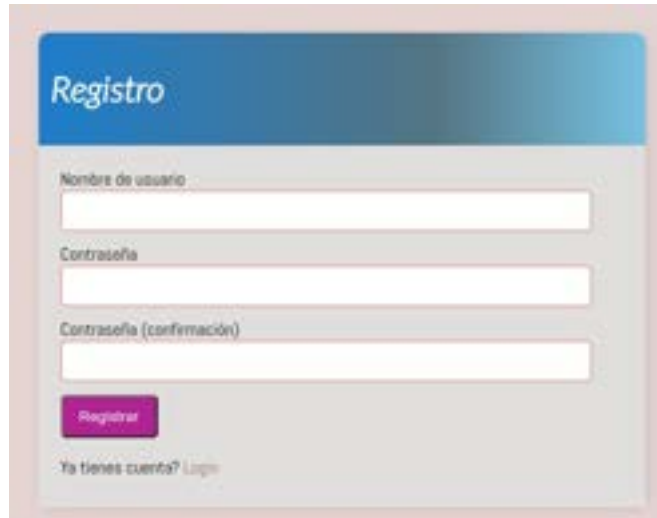


En registro.html es igual. Pero además, vamos a modificar el form por defecto poniendo uno a uno los inputs para que no se vea el texto aclaratorio y le damos una aclaración de estilo al input submit para separar el botón del último input:

```
{% extends 'base/principal.html' %}
{% block content %}
<div class="barra_superior">
  <h1>Registro</h1>
</div>
<div class="cuerpo-tarjeta">
  <form method="post" action="">
    {% csrf_token %}
    <label>{{ form.username.label }} </label>
    {{ form.username }}
    <label>{{ form.password1.label }} </label>
    {{ form.password1 }}
    <label>{{ form.password2.label }} </label>
    {{ form.password2 }}
    <input style="margin-top: 12px;" class="boton" type="submit" value="Registrar">
  </form>
  <p>Ya tienes cuenta? <a href="{% url 'login' %}">Login</a></p>
</div>
```

```
{% endblock content %}
```

Y queda así:



The image shows a registration form with a blue header containing the word "Registro". Below the header, there are three input fields: "Nombre de usuario", "Contraseña", and "Contraseña (confirmación)". A purple button labeled "Registrar" is positioned below the fields. At the bottom of the form, there is a link that says "Ya tienes cuenta? Login".

TEMA 17 - Extra. Bibliotecas para hacking ético

17.1. - Bibliotecas

Bibliotecas útiles para la manipulación de paquetes de red, la programación de red asincrónica, la generación de datos falsos, la criptografía, la creación de exploits y pruebas de penetración, etc

Python-nmap: es una biblioteca de Python que se utiliza para interactuar con el escáner de puertos nmap. Proporciona una interfaz de Python para nmap y permite a los usuarios realizar escaneos de red, descubrir hosts y puertos abiertos, y obtener información detallada sobre los sistemas y servicios de red.

IMPacket: es una biblioteca de Python para la manipulación de paquetes de red. Proporciona una amplia gama de herramientas para la creación, manipulación y envío de paquetes de red, y es muy útil en aplicaciones de pruebas de penetración y seguridad de red.

Requests: es una biblioteca de Python para hacer solicitudes HTTP. Proporciona una interfaz fácil de usar para realizar solicitudes HTTP, como GET y POST, y permite a los usuarios interactuar con servidores web y API.

Twisted: es una biblioteca de Python para programación de red asincrónica. Proporciona una amplia gama de herramientas para crear servidores y clientes de red asincrónicos, y es muy útil en aplicaciones de comunicación de red en tiempo real, como chat y videoconferencia.

Faker: es una biblioteca de Python para generar datos falsos. Proporciona una amplia gama de herramientas para generar nombres, direcciones, números de teléfono y otra información falsa, y es muy útil en aplicaciones de pruebas y simulaciones de datos.

Scapy: es una biblioteca de Python para la manipulación de paquetes de red. Proporciona una amplia gama de herramientas para la creación, manipulación y análisis de paquetes de red, y es muy útil en aplicaciones de pruebas de penetración y seguridad de red.

Cryptography: es una biblioteca de Python para la criptografía. Proporciona una amplia gama de herramientas para la encriptación y desencriptación de datos, y es muy útil en aplicaciones de seguridad y protección de datos.

Pwntools: es una biblioteca de Python para la creación de exploits y pruebas de penetración. Proporciona una amplia gama de herramientas para la creación y ejecución de exploits, así como para la manipulación de memoria y otros aspectos de la seguridad informática.

Paramiko: es una biblioteca de Python para la conexión y el manejo de servidores SSH. Proporciona una interfaz fácil de usar para la conexión y la gestión de servidores SSH, y es muy útil en aplicaciones de automatización y administración de servidores.

Pylibnet: es una biblioteca de Python para la creación y manipulación de paquetes de red. Proporciona una amplia gama de herramientas para la creación y manipulación de paquetes de red personalizados, y es muy útil en aplicaciones de pruebas de penetración y seguridad de red.