



**IPN-CICATA**  
**Querétaro**

# Python 101

Dr. Angel Moisés Hernández Ponce

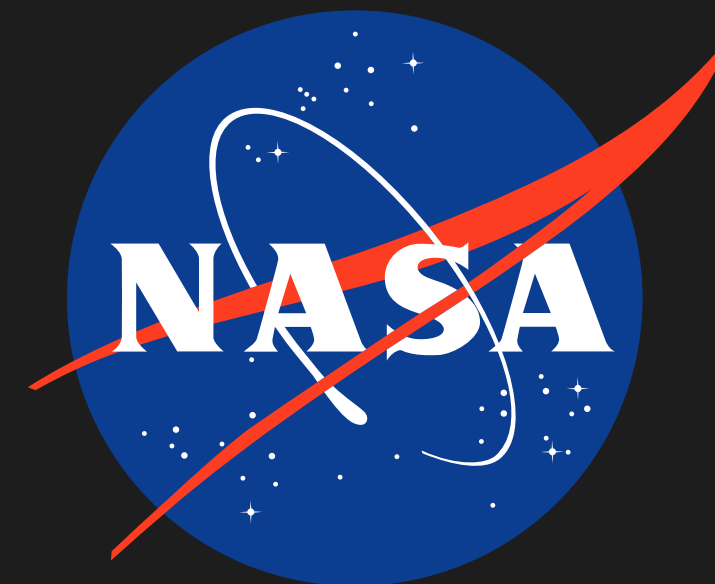
[www.icasatconference.com](http://www.icasatconference.com)

# Qué es Python?

- Python es un lenguaje de programación interpretado, de alto nivel, de tipado dinámico y orientado a objetos con una filosofía amigable.
- Python es un lenguaje de código abierto.
- Corre en cualquier sistema operativo.



# Quién usa Python?





# Para qué se usa Python?

Python se puede utilizar para una variedad de aplicaciones, las más importantes actualmente son:

- Inteligencia Artificial
- Computo científico
- Ciencia y Análisis de datos
- Desarrollo de interfaces gráficas de usuario
- Frameworks para páginas web
- Procesamiento de imágenes
- Desarrollo de videojuegos



# La filosofía de Python

*Bello es mejor que feo*

*Explicito es mejor que implícito*

*Simple es mejor que complejo*

*Complejo es mejor que complicado[...]*

*Si la idea es fácil de explicar es una buena idea*

*Si la idea es difícil de explicar es una mala idea*



Guido Van Rossum  
Creador de Python



# Cómo usar Python?

IDE	Entorno Virtual	Entorno Virtual Online
<ul style="list-style-type: none"><li>• Tiene una interfaz integrada para escribir, editar, compilar y depurar código</li><li>• Tiene herramientas adicionales para analizar errores, realizar pruebas y administra proyectos.</li><li>• Interfaz gráfica cómoda para programar.</li></ul>	<ul style="list-style-type: none"><li>• Entornos aislados para proyectos independientes.</li><li>• Permite organizar y mantener limpios los proyectos de desarrollo.</li><li>• Facilita la colaboración entre varios usuarios.</li></ul>	<ul style="list-style-type: none"><li>• Combina un IDE y un entorno virtual pero con la ventaja de que corre en la nube, es decir, no necesitamos instalar software en nuestra computadora.</li><li>• Viene con muchos paquetes y librerías precargados.</li><li>• Integración con otros servicios en la nube</li></ul>

# Google Colaboratory



<https://colab.research.google.com>

Configuración del proyecto

Login

Nombre del proyecto

Botón de compartir

Barra de herramientas

Gestor de celdas

índice del proyecto

Herramientas  
adicionales

Celdas de texto

Celda de código



# Fundamentos de programación en Python

# Tipos de variables

# Qué es una variable?

- Una variable se utiliza para almacenar información o datos para que podamos llamarlas cuando lo necesitemos.
- Una variable siempre debe tener un nombre, de preferencia que haga referencia a lo que contiene.
- A diferencia de otros lenguajes de programación, en Python no necesitamos declarar el tipo de variable.
- También podemos cambiar el contenido de la variable sobre la marcha, incluso cambiar el tipo de variable



# Algunas restricciones

- No pueden iniciar con números
  - 2x, 20kilos, 345i
- No puede haber espacios en el nombre
- No puede contener estos simbolos ' " < > / \ ? ! ( ) @ \$ % & \* + -
- No pueden utilizar palabras clave como: for, while, if, else, help, or, and



# Palabras reservadas

and	as	assert	break	class
continue	def	del	elif	else
except	False	finally	for	from
global	if	import	in	is
lambda	none	nonlocal	not	or
pass	raise	return	True	try
while	with	yield		

# Tipos de datos

# Tipos de datos



Pro tip: Si no están seguros del tipo de dato de una variable, pueden escribir `type(nombre_variable)` y Python les dirá el tipo.

# Numéricos

- **Puede ser cualquier número.**
- **Los enteros se conocen como int**
- **Los números con decimales se llaman flotantes**
- **Se puede realizar cualquier operación aritmética con ellos, comparaciones lógicas.**
- **Los números complejos deben tener su parte real y su parte imaginaria.**



# Operadores numéricos

Nombre	Simbolo	Ejemplo
Suma	+	1 + 1
Resta	-	5 - 1
Multiplicación	*	10 * 4
División	/	20 / 4
Modulo	%	24 % 3
Potencia	**	2 ** 3
División por suelo (floor division)	//	15 // 2

# Jerarquía de operaciones



**Exponenciación/Potenciación**

**Negación**

**Multiplicación / División  
División por Suelo/ Módulo**

**Suma y Resta**



**Los paréntesis también influyen en la jerarquía de las operaciones!**

# Practica 1 — Python como calculadora



# Cadenas de caracteres (strings)

- Pueden ser cualquier carácter siempre y cuando estén rodeados por comillas simples (") o comillas dobles (""").
  - Letras (mayúsculas y minúsculas) y dígitos numéricos (0-9)
  - Espacios en blanco
  - Caracteres especiales
  - Caracteres de escape
    - \n --> Salto de línea
    - \t --> Tabulación
    - \a --> Produce un beep (solo funciona en la terminal)
  - Unicode (casi cualquier lenguaje humano es soportado)



# Strings pt2

- En la mayoría de los casos, las strings son de linea simple, es decir, no hay un 'enter' entre ellas.
- Las strings multilineas se ponen entre tres comillas dobles (""" """)
- Si nuestra string tiene un apostrofe (') es mejor usar comillas dobles (" ")
- Con los acentos no hay problema.

# Practica 2 — Strings



# Manipulación de strings

- Las strings se pueden manipular ya que es una cadena de caracteres.
- La operación de indexación (indexing) permite extraer un carácter específico.
  - Se utilizan los corchetes []
- La operación de slicing extrae una parte del string
  - Se utilizan los corchetes [] pero indicando el rango deseado.



**En Python la cuenta inicia desde 0**  
**Los espacios también cuentan**

# Ejemplo

```
# Operación de indexación y slicing de strings
```

```
texto = "Python es genial"
```

```
# Indexación
```

```
print(texto[0])      # Acceder al primer carácter, imprime: "P"
```

```
print(texto[7])      # Acceder al octavo carácter, imprime: "e"
```

```
print(texto[-1])     # Acceder al último carácter, imprime: "l"
```

```
# Slicing
```

```
print(texto[0:6])     # Obtener los caracteres desde el índice 0 hasta el 5, imprime: "Python"
```

```
print(texto[7:])      # Obtener los caracteres desde el índice 7 hasta el final, imprime: "es genial"
```

```
print(texto[:6])      # Obtener los caracteres desde el inicio hasta el índice 5, imprime: "Python"
```

```
print(texto[-6:])     # Obtener los últimos 6 caracteres, imprime: "genial"
```

```
print(texto[::3])     # Obtener cada segundo carácter, imprime: "Ph nl"
```



# Ejemplo 2

```
# Operación de indexación y slicing de strings
texto = "Hola, bienvenidos al curso de Python 101"

# Indexación
print(texto[0])    # Acceder al primer carácter, imprime: "H"
print(texto[5])    # Acceder al sexto carácter, imprime: ","
print(texto[-1])   # Acceder al último carácter, imprime: "1"

# Slicing
print(texto[0:4])  # Obtener los caracteres desde el índice 0 hasta el 3, imprime: "Hola"
print(texto[7:])   # Obtener los caracteres desde el índice 7 hasta el final, imprime: "bienvenidos al curso de Python 101"
print(texto[:4])   # Obtener los caracteres desde el inicio hasta el índice 3, imprime: "Hola"
print(texto[-3:])  # Obtener los últimos 3 caracteres, imprime: "101"
print(texto[::2])  # Obtener cada segundo carácter, imprime: "Hl,benvnslcus ePto 1"
```

# Practica 3 — Manipulación de strings



# Arreglos



# Listas

- Arreglos no ordenados
- Pueden contener cualquier tipo de dato, incluso otra lista.
- Son mutables
- Se pueden añadir o eliminar valores según se necesite.
- Se declaran con corchetes []
- Cada valor se separa con una coma (,)

# Ejemplo

```
numeros = [1, 2, 3, 4, 5] # Lista de números
nombres = ["Juan", "María", "Carlos"] # Lista de strings
mezclado = [1, "dos", True, [1, 2, 3]] # Lista con elementos de diferentes tipos

print(numeros) # Imprime: [1, 2, 3, 4, 5]
print(nombres) # Imprime: ["Juan", "María", "Carlos"]
print(mezclado) # Imprime: [1, "dos", True, [1, 2, 3]]
```



# Manipulación de listas

Nombre	Función	Sintaxis
append()	Añade un elemento al final de la lista	append(elemento)
insert()	Inserta un elemento en un índice específico de la lista	insert(índice, elemento)
remove()	Elimina la primera aparición de un elemento	remove(elemento)
pop()	Elimina y devuelve el elemento en el índice especificado. Si no se proporciona el índice, se elimina y devuelve el último elemento	pop([índice])
index()	Devuelve el índice de la primera aparición de un elemento en la lista	index(elemento)
count()	Cuenta el número de veces que un elemento aparece en la lista	count(elemento)
sort()	Ordena los elementos de la lista de forma ascendente	nombre_lista.sort()
reverse()	Invierte el orden de los elementos en la lista	nombre_lista.reverse()
clear()	Elimina todos los elementos de la lista	nombre_lista.clear()

# Ejemplo métodos

```
# append(elemento)
frutas = ['manzana', 'pera', 'banana']
frutas.append('naranja')
print(frutas) # Imprime: ['manzana', 'pera', 'banana', 'naranja']
```

```
# insert(indice, elemento)
colores = ['rojo', 'verde', 'azul']
colores.insert(1, 'amarillo')
print(colores) # Imprime: ['rojo', 'amarillo', 'verde', 'azul']
```

```
# remove(elemento)
animales = ['perro', 'gato', 'perro', 'conejo']
animales.remove('perro')
print(animales) # Imprime: ['gato', 'perro', 'conejo']
```

```
# pop([indice])
numeros = [1, 2, 3, 4, 5]
elemento = numeros.pop(2)
print(numeros) # Imprime: [1, 2, 4, 5]
print(elemento) # Imprime: 3
```

```
# index(elemento)
frutas = ['manzana', 'pera', 'banana']
indice = frutas.index('pera')
print(indice) # Imprime: 1
```

```
# count(elemento)
numeros = [1, 2, 2, 3, 2, 4]
contador = numeros.count(2)
print(contador) # Imprime: 3
```

```
# sort()
numeros = [4, 2, 1, 3]
numeros.sort()
print(numeros) # Imprime: [1, 2, 3, 4]
```

```
# reverse()
letras = ['a', 'b', 'c', 'd']
letras.reverse()
print(letras) # Imprime: ['d', 'c', 'b', 'a']
```

```
# clear()
elementos = [1, 2, 3, 4, 5]
elementos.clear()
print(elementos) # Imprime: []
```

# Practica 4 — Listas y manipulación



# Tuplas

- Las tuplas son un tipo de arreglos inmutable, es decir, no se pueden modificar una vez son declaradas.
- Es una colección ordenada.
- Se declaran con paréntesis ().
- Son útiles para guardar valores constantes o que nunca van a cambiar.
- Se pueden aplicar la indexación y slicing a las tuplas.
- Algunos métodos están restringidos por la naturaleza de las tuplas.

# Ejemplos de tuplas

```
# Tupla de nombres de colores  
colores = ('rojo', 'verde', 'azul')
```

```
# Tupla de coordenadas en un plano  
coordenadas = (3.5, -2.8)
```

```
# Tupla de información personal  
informacion_personal = ('Juan', 'Pérez', 25, 'Calle Principal')
```

```
# Tupla de días de la semana  
dias_semana = ('Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado', 'Domingo')
```

```
# Tupla de coordenadas geográficas  
coordenadas_geograficas = (40.7128, -74.0060)
```



# Practica 5 — Tuplas



# Diccionarios

- Son arreglos no ordenados.
- Relacionan los datos la forma llave-valor.
- Las llaves pueden deben ser valores inmutables como strings, números o tuplas.
- Los valores asociados a cada llave puede ser cualquier tipo de dato, incluyendo otros diccionarios.
- Se declaran con las llaves {}.
- Después de escribir la llave se escriben dos puntos(:) para indicar los valores asociados a esa llave, cada par llave-valor se separa por una coma.

# Ejemplos diccionarios

```
persona = {  
    'nombre': 'Juan',  
    'edad': 25,  
    'ciudad': 'Madrid'  
}  
  
print(persona['nombre']) # Imprime: 'Juan'  
print(persona['edad'])   # Imprime: 25  
print(persona['ciudad']) # Imprime: 'Madrid'  
  
persona['edad'] = 26      # Modificar el valor de 'edad'  
persona['genero'] = 'Masculino' # Agregar una nueva pareja clave-valor  
  
print(persona) # Imprime: {'nombre': 'Juan', 'edad': 26, 'ciudad': 'Madrid', 'genero': 'Masculino'}
```

# Ejemplo Diccionarios

```
generos_music = {  
    'rock': {  
        'origen': 'Estados Unidos',  
        'decada': 'década de 1950',  
        'artistas_destacados': ['The Beatles', 'Led Zeppelin', 'Queen'],  
        'canciones_famosas': ['Stairway to Heaven', 'Bohemian Rhapsody']  
    },  
    'jazz': {  
        'origen': 'Estados Unidos',  
        'decada': 'finales del siglo XIX',  
        'artistas_destacados': ['Louis Armstrong', 'Miles Davis', 'Ella Fitzgerald'],  
        'canciones_famosas': ['Take Five', 'Summertime']  
    },  
    'pop': {  
        'origen': 'Estados Unidos',  
        'decada': 'década de 1950',  
        'artistas_destacados': ['Michael Jackson', 'Madonna', 'Taylor Swift'],  
        'canciones_famosas': ['Thriller', 'Like a Prayer']  
    }  
}  
  
print(generos_music['rock']['origen']) # Imprime: 'Estados Unidos'  
print(generos_music['jazz']['artistas_destacados']) # Imprime: ['Louis Armstrong', 'Miles Davis', 'Ella Fitzgerald']  
print(generos_music['pop']['canciones_famosas'][0]) # Imprime: 'Thriller'
```

# Practica Diccionarios



# Sets

- Los sets es un arreglo no ordenado que almacena datos únicos, es decir, no admite duplicados.
- Son útiles cuando necesitamos almacenar datos sin orden específico y únicos.
- Se declaran usando las llaves {} y los datos se separan con coma (,).
- Los datos no están indexados por lo que no podemos hacer uso de la indexación o slicing en los datos, sin embargo, sí podemos interactuar con los sets de otras formas.
- Los sets son mutables



# Ejemplo Sets

```
# Sets
numeros = {1, 2, 3, 4, 4, 5}
print(numeros) # Imprime: {1, 2, 3, 4, 5}

# Un set con varios tipos de datos
datos = {1, "Hola", True, 3.14}
print(datos) # Imprime: {1, 'Hola', True, 3.14}

# También pueden declarar un conjunto vacío
conjunto_vacio = set()
print(conjunto_vacio) # Imprime: set()

# Modificando valores de los sets
frutas = {"manzana", "plátano", "naranja"}

print("manzana" in frutas) # Verifica si "manzana" está en el conjunto

frutas.add("mango") # Agrega un elemento al conjunto

frutas.remove("plátano") # Elimina un elemento del conjunto

print(len(frutas)) # Devuelve la cantidad de elementos en el conjunto
```

# Ejemplo de métodos con Sets

```
# Aplicando varios métodos a los sets
# Creación de un conjunto
frutas = {"manzana", "plátano", "naranja"}

# Método add()
frutas.add("mango")
print(frutas) # Imprime: {'manzana', 'plátano', 'naranja', 'mango'}

# Método remove()
frutas.remove("plátano")
print(frutas) # Imprime: {'manzana', 'naranja', 'mango'}

# Método pop()
elemento = frutas.pop()
print(elemento) # Imprime un elemento aleatorio del conjunto
print(frutas) # Imprime el conjunto sin el elemento removido

# Método clear()
frutas.clear()
print(frutas) # Imprime: set()
```

# Mas ejemplos de métodos

```
# Creación de dos conjuntos para ilustrar otros métodos
set1 = {1, 2, 3, 4, 5}
set2 = {4, 5, 6, 7, 8}

# Método union()
union = set1.union(set2)
print(union) # Imprime: {1, 2, 3, 4, 5, 6, 7, 8}

# Método intersection()
interseccion = set1.intersection(set2)
print(interseccion) # Imprime: {4, 5}

# Método difference()
diferencia = set1.difference(set2)
print(diferencia) # Imprime: {1, 2, 3}
```

# Operadores booleanos & de comparación

# Operadores booleanos

- Sirven para establecer comparaciones lógicas entre dos valores booleanos, True (Verdadero) y False (False).
- Están basados en el algebra booleano desarrollada por George Boole
- El resultado siempre sera de tipo booleano (bool).
- Solo hay dos resultados posibles, True o False.
- Las tablas de verdad nos indican los resultados posibles de las combinaciones.

# Tablas de verdad

**Compuerta AND**

Entrada		Salida
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

**Compuerta OR**

Entrada		Salida
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1



Pro tip: En la compuerta AND la salida, siempre es 1 si ambas entradas son 1. En la compuerta OR, la salida siempre es 1 a menos que ambas entradas sean 0.



# Operadores de comparación

- Los operadores de comparación evalúan si una afirmación es verdadera.
- La comparación debe ser entre un mismo tipo de valores.
- El resultado siempre es de tipo bool.
- Nos sirven para controlar el flujo de un programa o determinar cierto tipo de acciones según sea el caso.

Operador	Afirmacion
<code>==</code>	Es igual a
<code>!=</code>	Es distinto a
<code>&lt;</code>	Menor que
<code>&gt;</code>	Mayor que
<code>&lt;=</code>	Menor o igual que
<code>&gt;=</code>	Mayor o igual que

# Ejemplos Operadores

```
# Operadores booleanos
```

```
a = True
```

```
b = False
```

```
print(a and b) # Imprime: False
```

```
print(a or b) # Imprime: True
```

```
print(not a) # Imprime: False
```

```
# Otra forma de usar los operadores booleanos
```

```
print(a & b) # Imprime False
```

```
print(a | b) # Imprime True
```

```
# Operadores de comparación
```

```
x = 5
```

```
y = 10
```

```
print(x == y) # Imprime: False
```

```
print(x != y) # Imprime: True
```

```
print(x < y) # Imprime: True
```

```
print(x >= y) # Imprime: False
```

# Practica Operadores



# Examen sorpresa

# Estructuras de control

## Condicionales

If-Then-Else   Switch case

## Repetitivas

Ciclo For   Ciclo While

# Sintaxis

Palabra clave

Condición a evaluar

Estos : (dos puntos) son obligatorios

```
>>> if var < 200:  
    print('Afirmativo')
```

Este pequeño espacio  
se llama indentación

Instrucciones a ejecutar dentro de la condición

Afirmativo ← Resultado



# if-else-elif

- En las condicionales, la condición que está hasta arriba se evalúa primero y así sucesivamente hacia abajo.
- Pueden hacer múltiples evaluaciones con la palabra "elif" y siempre deben escribir la condición a evaluar.
- La palabra 'else' siempre indica la acción a realizar cuando no se cumpla la condición.

```
num = int(input("Ingresa un número: "))  
  
if num > 0:  
    print("El número es positivo")  
elif num < 0:  
    print("El número es negativo")  
else:  
    print("El número es cero")
```

# Ejemplos

```
if calificacion >= 90:  
    print("Aprobado con A")  
elif calificacion >= 80:  
    print("Aprobado con B")  
elif calificacion >= 70:  
    print("Aprobado con C")  
elif calificacion >= 60:  
    print("Aprobado con D")  
else:  
    print("Reprobado")
```

# Ejemplos

```
if num % 2 == 0:  
    print("El número es par")  
else:  
    print("El número es impar")
```

# Practica If-Else



# Estructuras repetitivas

## Ciclo for

- Se utiliza para iterar sobre una secuencia o arreglo de datos.
- A diferencia del ciclo while, este ciclo itera sobre todos los elementos del arreglo.
- Se pueden ejecutar acciones mientras el ciclo esté activo.
- Si van a iterar sobre un rango de valores se usa la palabra clave range().

## Ciclo while

- Este ciclo se ejecuta siempre y cuando se cumpla una condición.
- Si la condición no se cumple o está mal planteada, tendremos un bucle infinito y NO queremos eso.
- Se usa una variable contadora para evitar los ciclos infinitos.
- Se puede interrumpir de manera arbitraria con la palabra clave break.

# Sintaxis

## Ciclo for

```
for elemento in secuencia:  
    # Código a ejecutar para cada elemento
```

## Ciclo while

```
while condicion:  
    # Código a ejecutar mientras se cumpla la condición
```



# Ejemplo ciclo for

```
for fruta in frutas: # Itera sobre cada elemento de la lista 'frutas'  
    print(fruta) # Imprime el valor de cada elemento
```

```
# Ahora usamos animales en lugar de frutas
```

```
animales = ["león", "tigre", "jirafa", "elefante"]
```

```
for animal in animales:
```

```
    print(animal) # Imprime cada animal de la lista
```

```
# Usando la función range()
```

```
for num in range(1, 6):
```

```
    print(num) # Imprime una lista de números del 0 al 5
```

# Ejemplo ciclo while

```
# Ejemplo de ciclos while

contador = 0 # Variable contadora

while contador < 5:
    print(contador) # Imprime el valor de la variable contador
    contador += 1 # Incrementa en 1 el valor de contador

# Usando ciclo while y la estructura if
respuesta = "" # String vacía

while respuesta != "si":
    respuesta = input("¿Quieres continuar? (si/no): ")

    if respuesta == 'si':
        print('Cotinuemos')
    else:
        print('Ciclo cancelado')

# Ejemplo 3
suma = 0
numero = 1

while numero != 0:
    numero = int(input("Ingresa un número (0 para salir): "))
    suma += numero

print("La suma total es:", suma)
```

# Practica Ciclos



# Funciones



# Funciones personalizadas

- Las funciones personalizadas nos permite encapsular código estructurado y reutilizarlo cuando sea necesario.
- Podemos definir tantas funciones como queramos y utilizarlas para tareas específicas.
- Pueden tener cualquier tipo de variable como entrada, incluso puede no tener entradas.
- Para definir una función se usa la palabra clave **def**

# Sintaxis

Palabra clave **def**

No olviden los dos puntos

```
def nombre_de_la_funcion(parametros):  
    # Código de la función  
    # ...  
    return resultado
```

Parámetros de entrada

Palabra clave **return**

# Ejemplo 1 Funciones

```
def sumar(a, b):  
    resultado = a + b  
    return resultado
```



Todas las variables que declaren dentro de la función son temporales, es decir, existen solamente mientras la función se ejecute. Si quiere llamar a la variable fuera de la función les arrojará un error



# Ejemplo 2 Funciones

```
def es_positivo(numero):  
    if numero > 0:  
        return True  
    else:  
        return False
```

# Más sobre los parámetros

- Pueden asignar un valor por default, en caso de ser necesario.
- El número de parámetros de la función siempre tiene que ser igual los valores de entrada. De lo contrario les marcara un error.
- Cada parámetro lo deben usar en la función.
- Recuerden que cada parámetro debe estar separado por un coma (,)

# Más sobre los parámetros

```
def saludar(nombre, mensaje="¡Hola!"):  
    print(mensaje, nombre)
```

# Funciones propias de Python

- Son funciones que tiene Python de forma nativa.
- Cada una tiene su función específica.
- Pueden llamarlas en cualquier parte del código.
- Incluso pueden usar estas funciones dentro de las funciones personalizadas.

# Funciones propias de Python

Nombre	Funcionamiento	Ejemplo
<code>abs()</code>	Devuelve el valor absoluto de cualquier número	<code>abs(x)</code>
<code>bin()</code>	Convierte un número entero a binario. Esta función añade el prefijo '0b' al número binario	<code>bin(x)</code>
<code>enumerate()</code>	Itera sobre un arreglo de datos y al mismo tiempo registra la posición el índice de cada objeto dentro del arreglo	<code>enumerate(x)</code>
<code>len()</code>	Devuelve la longitud de un arreglo o secuencia de datos	<code>len(x)</code>
<code>max(), min()</code>	Devuelve el valor máximo o mínimo de un arreglo de datos	<code>max(x), min(x)</code>
<code>print()</code>	Despliega la información que esté dentro de los paréntesis	<code>print(x)</code>

# Tips para las funciones

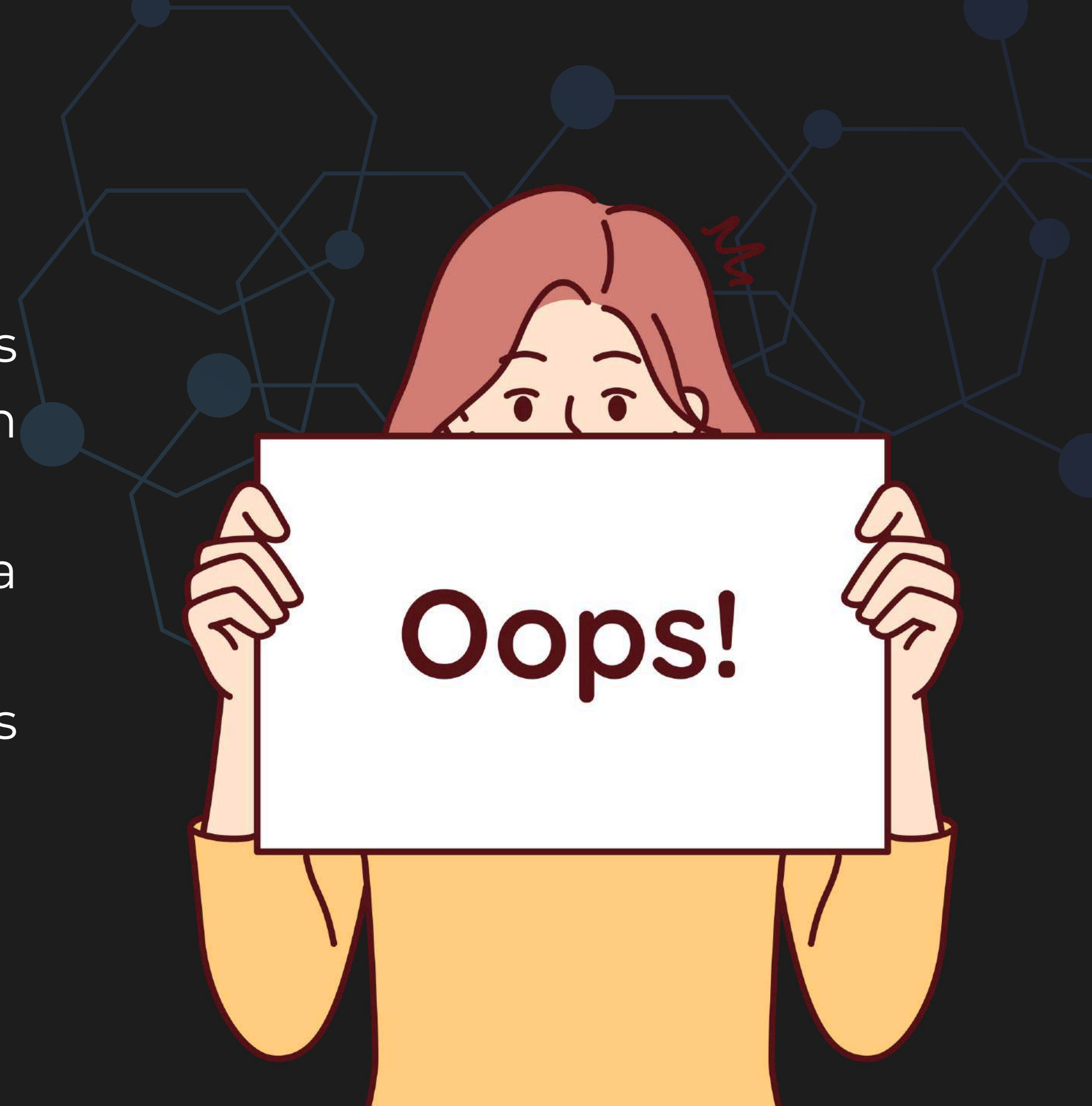
- Definan muy bien y antes de comenzar qué hará su función exactamente.
- Procuren documentar bien su código, nunca saben quién leerá su código en el futuro,
- Elijan un nombre acorde al funcionamiento de la función.
- Prueben con muchos ejemplos, traten de que su función se infalible.
- Mantengan su código limpio y ordenado.
- Si su función hará dos o más cosas distintas, mejor divídanlas. Divide y vencerás.

# Manejo de errores



# ¿Qué son los errores?

- En Python nunca estaremos exentos de errores, es más si no encuentran errores es que algo anda mal.
- Un error en nuestro código significa la interrupción de nuestro programa.
- Los errores se pueden clasificar en dos grupos:
  - Errores de sintaxis
  - Excepciones



# Tipos de errores

## Sintaxis

- Cualquier cosa que escribamos mal o de manera incorrecta en el código.
- Se detectan antes de la ejecución del programa

```
print("Hola mundo!" # Faltó cerrar el paréntesis
```

## Excepciones

- Errores de ejecución del programa
- Operaciones invalidas o atributos inválidos.

```
10 / 0 # División entre cero  
"10" + 5 # Suma de una string y un entero
```

# Tipos de excepciones

Excepción	Causa común	Ejemplo
<code>ZeroDivisionError</code>	División entre cero	<code>10 / 0</code>
<code>TypeError</code>	Operación con tipo incorrecto	<code>"10"+5</code>
<code>ValueError</code>	Valor inapropiado	<code>int("abc")</code>
<code>IndexError</code>	Índice fuera de rango	<code>lista = [1, 2]; lista[3]</code>
<code>KeyError</code>	Clave inexistente en diccionario	<code>d = {"a":1}; d["b"]</code>
<code>FileNotFoundError</code>	Archivo no encontrado	<code>open("inexistente.txt")</code>

# Es el fin?







# Manejando las excepciones

## Try-except

- Con try-except podemos insertar código de 'respaldo' si creemos que haya una posibilidad de que nuestro código falle.
- La parte de try es donde pondremos nuestro código dudoso, mientras que en el bloque except contendrá aquellas acciones que haremos solo si se presenta una de las excepciones.
- Podemos añadir cuantos bloques except queramos.

```
try:
    # Código que puede fallar
    resultado = 10 / int(input("Divisor: "))
except ZeroDivisionError:
    # Maneja división entre cero
    print("¡No puedes dividir entre cero!")
except ValueError:
    # Maneja entrada no numérica
    print("Debes ingresar un número")
except Exception as e:
    # Captura cualquier otra excepción
    print(f"Error inesperado: {type(e).__name__}")
```

# Manejando las excepciones

## Else-finally

- El bloque else nos ayuda a ejecutar código solo si no hubo excepciones.
- El bloque finally se ejecuta siempre, haya o no errores o excepciones.
- El bloque finally es ideal para realizar cierre de API, limpieza de variables, borrar buffers, entre otras.

```
try:
    archivo = open("datos.txt", "r")
    contenido = archivo.read()
except FileNotFoundError:
    print("Archivo no encontrado")
else:
    print(f"Contenido: {contenido}")
finally:
    archivo.close() # Siempre se cierra el archivo
    print("Proceso finalizado")
```



# Práctica Funciones



# Librerías



# Aumentando el poder de Python

- Python puede aumentar sus funciones a través de librerías externas.
- Estas librerías son gratuitas y hay para muchos propósitos.
- Las librerías no vienen instaladas, hay que instalarlas para poder utilizarlas.
- En nuestro caso, Google Colab tiene una gran cantidad de librerías ya instaladas que podemos usar
- Para poder utilizar las librerías debemos llamarlas con la palabra clave **import**
- Si quieren conocer cuáles librerías tienen instaladas puede usar el comando **!pip list**.

# Numpy

- Es un librería especializada para el cómputo numérico científico y avanzado.
- Es ideal para trabajar con matrices y arreglos de datos.
- Tiene integrada operaciones de álgebra lineal, matricial, transformadas de Fouries, logaritmos, exponenciales, entre otras.
- También pueden utilizar constantes matemáticas como el valor de Pi o el valor de Euler
- Su sintaxis y varias funciones son muy parecidas a Matlab

# Matplotlib

- Es una librería que les permite realizar gráficas elegantes, entendibles y personalizadas.
- Pueden hacer gráficas de barras, histogramas, gráficas de pastel, gráficas de dispersión, entre otras.
- Pueden editar vía comandos los aspectos como el título, los ejes, las leyendas, escala de la gráfica, color de líneas, etc.
- En pocas palabras, programan su gráfica para que se vea como ustedes quieran.

# Pandas

- Permite manejar bases de datos y arreglos de datos mas facil y mas ordenado
- Pandas usa dos tipos de datos principales, DataFrame (la base de datos total) y Series (columnas, filas o intervalos).
- Sirve para realizar limpieza de datos, filtrado, ordenamiento, etc.
- Permite leer y escribir bases de datos en formatos de Excel, SQL, csv, JSON, entre otras.
- Puede ejecutar cálculos estadísticos como media, moda, mediana, desviación estándar, cuartiles, etc.

# Seaborn

- Librería enfocada a la visualización de datos.
- Permite hacer gráficas mas avanzadas para facilitar interpretabilidad.
- Puede trabajar con Pandas fácilmente.
- Permite un nivel más amplio de personalización



# OpenCV

- Es la librería más utilizada dentro del campo de la visión por computadora. Es de acceso libre y de código abierto.
- Es multiplataforma, actualmente está disponible para Python, C++, Java, además de correr en la mayoría de sistemas operativos.
- A través de esta plataforma se pueden crear aplicaciones de reconocimiento facial, detección de objetos, corrección de color, entre muchas otras más.

<https://opencv.org/>



# OpenCV

# Visión por computadora

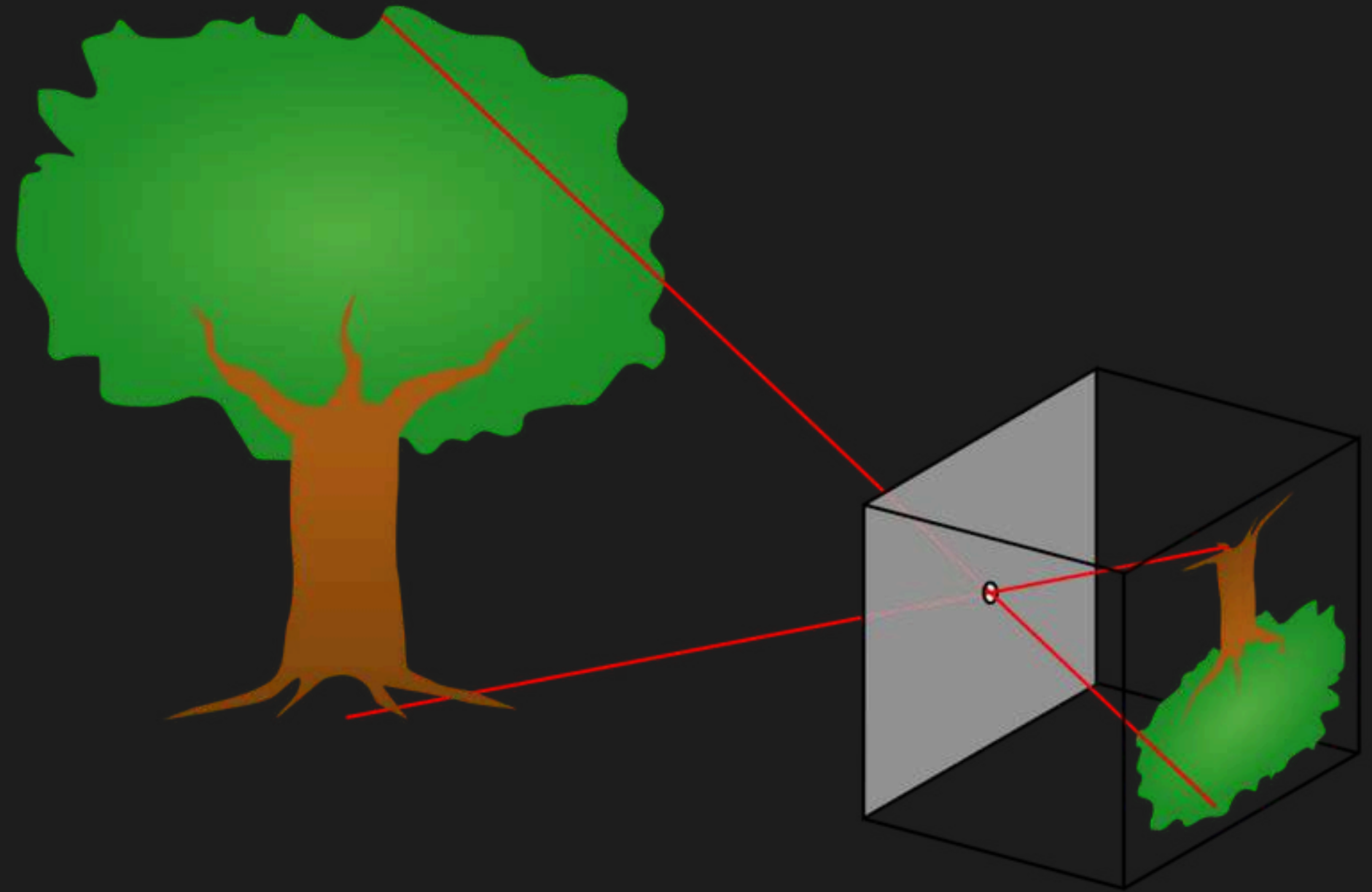
# Qué es una imagen?

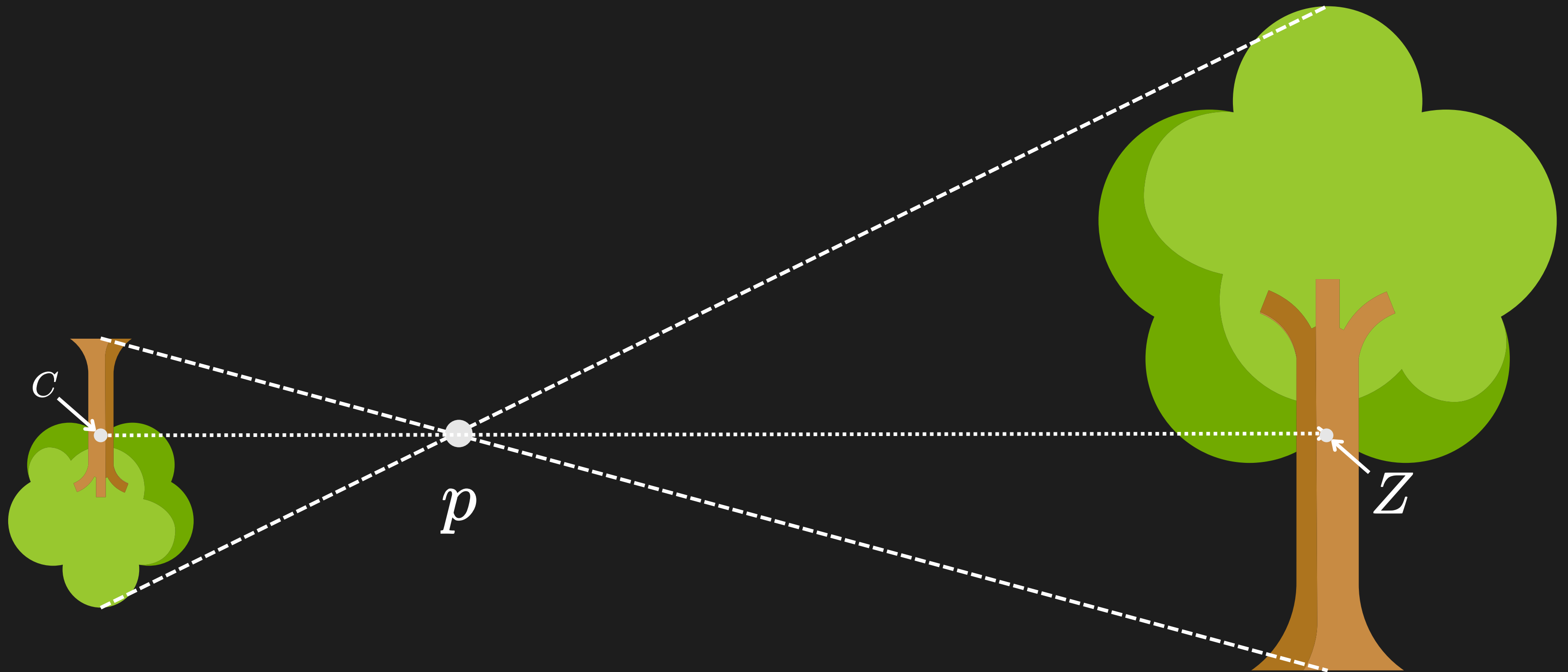
- Una imagen es una representación visual de un objeto, sea real o imaginario.
- Desde el punto de vista óptico una imagen no es un objeto físico sino el resultado de la interacción entre los fotones emitidos por un entorno y un sistema visual.
- Desde el punto de vista matemático, una imagen es una función bidimensional de la intensidad de la luz que incide sobre un espacio determinado.
- Desde el punto de vista digital, una imagen es una función  $f(x,y)$  que ha sido discretizada en coordenadas espaciales y en el brillo.
- Nostros vamos a considerar a una imagen como una matriz.



# Formación de imágenes

- Una imagen se forma al proyectar un objeto sobre un plano.
- Las cámaras se usaban para capturar imágenes en un inicio.
- La cámara oscura o pinhole es el modelo más rudimentario, datando su conocimiento desde el 500 a.C.
- El modelo matemático de una cámara se basa en el pinhole.







Origen



•  $f(x, y)$

Eje Y

Eje X

Origen

Eje X

Eje Y

•  $f(x, y)$

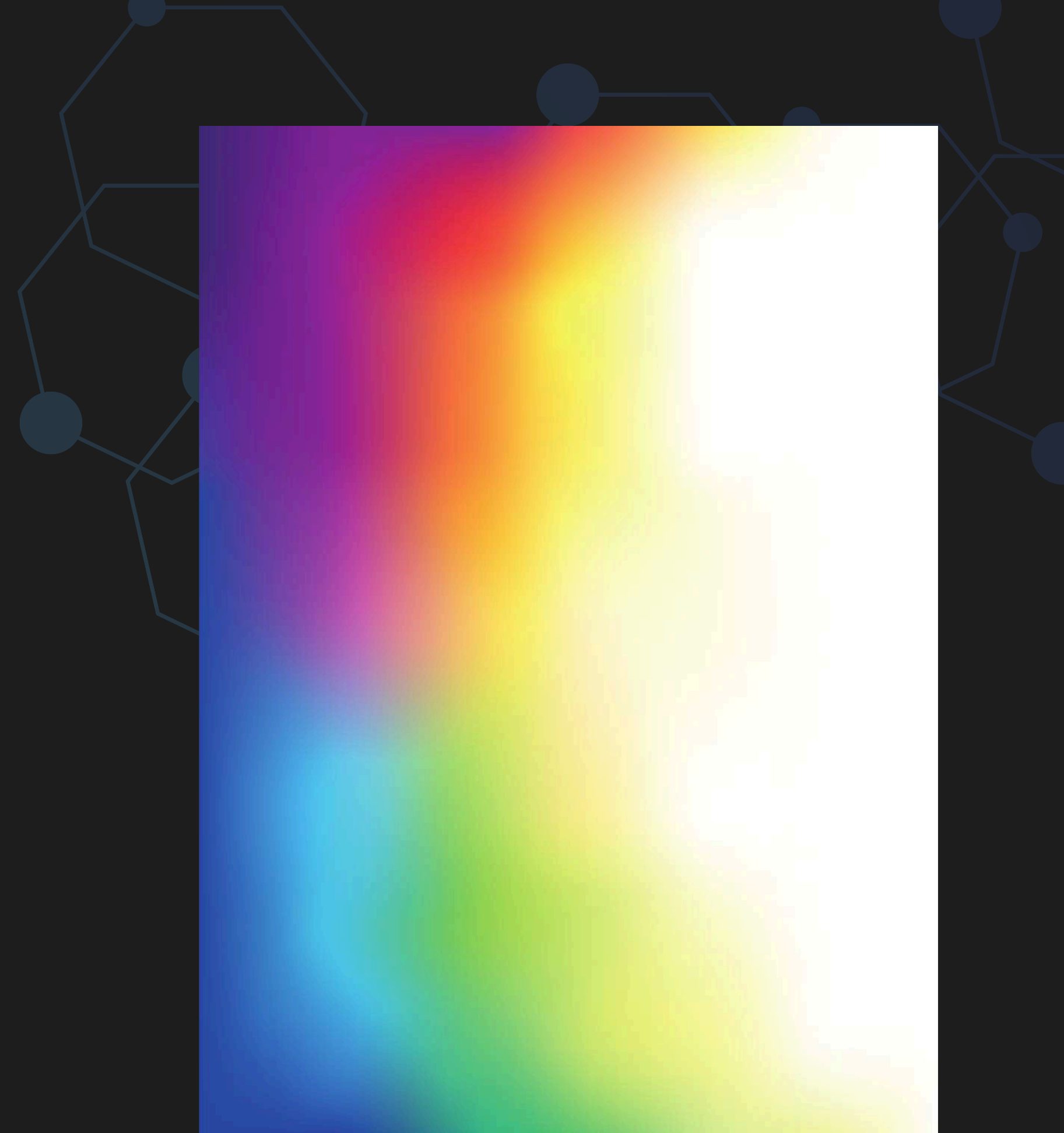
# Que es el color?





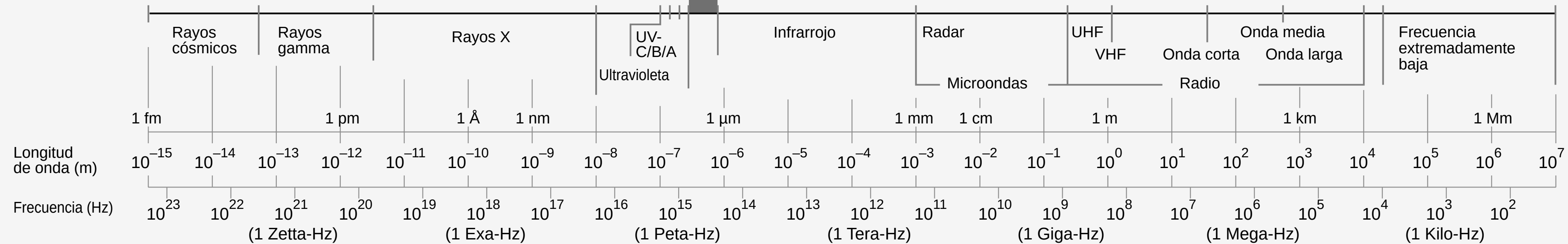
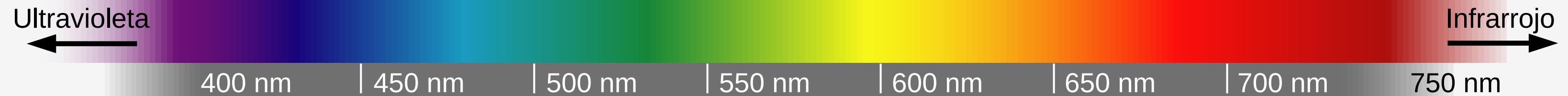
# Qué es el color?

- El color es la impresión que produce un tono de luz en los órganos visuales
- Es la percepción visual que nuestro cerebro genera al interpretar señales nerviosas enviadas por los fotorreceptores del ojo.
- El color depende de tres factores fundamentales: el observador, la fuente de luz y del objeto.



# El espectro visible

## Espectro visible por el ojo humano (Luz)

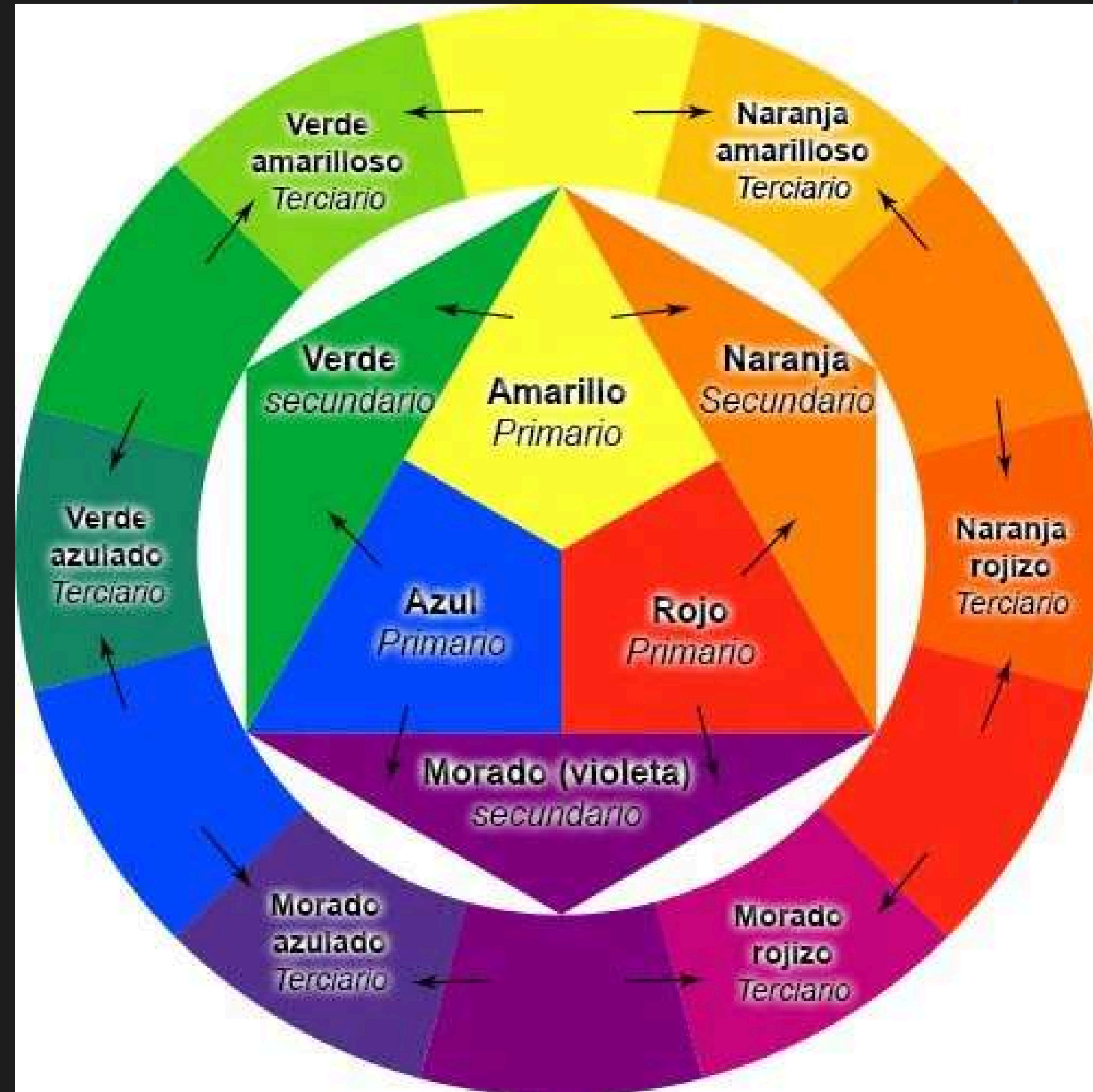


Color	Longitud de onda
Violeta	~ 380-427 nm
Azul	~ 427-476 nm
Cian	~ 476-497 nm
Verde	~ 497-570 nm
Amarillo	~ 570-581 nm
Naranja	~ 581-618 nm
Rojo	~ 618-780 nm

# Espacios de Color

- Son sistemas de interpretación de color que permiten organizar de manera específica los colores.
- Un espacio de color puede basarse en dispositivos físicos, señales analógicas, representaciones digitales, modelos matemáticos, entre otros.







# Blanco y negro vs Escala de grises vs Color



Imagen a color



Blanco y negro



Escala de grises



# Blanco y negro vs Escala de grises vs Color



27 89 12 90  
 104 56 99 1  
 179 101 206 49  
 45 37 69 102 73 250 108 127  
 45 67 159 213  
 147 25 93 15  
 52 87 109 113 57 197 33 254  
 128 100 199 56  
 43 215 178 96  
 87 63 45 10

0 0 1 0  
 0 0 1 1  
 1 0 1 0  
 1 1 0 1

100 52 96 10  
 5 485 78 63  
 12 85 107 65  
 205 8 183 249

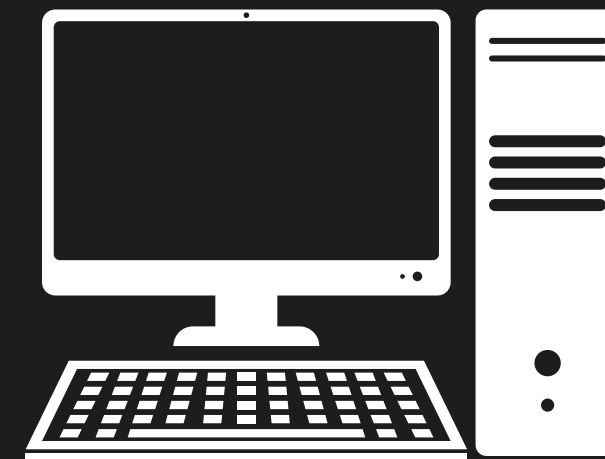
# Practica

## Tomar imágenes con OpenCV



## Tomar imágenes con OpenCV

- Hay varias formas de leer y procesar imágenes con OpenCV
  - Leer la imagen desde disco
  - Tomar captura con la cámara
  - Procesar videos en cuadros
- Si quieren trabajar con video, deben hacer la captura cuadro por cuadro, es decir, toma continua de imágenes.
- Funciones
  - `cv2.VideoCapture(0)`
  - `cv2.isOpened()`
  - `cap.read()`
  - `cv2.imshow()`



Código en Python

Llamar OpenCV →



Controlar la cámara

## Workflow



# Practica

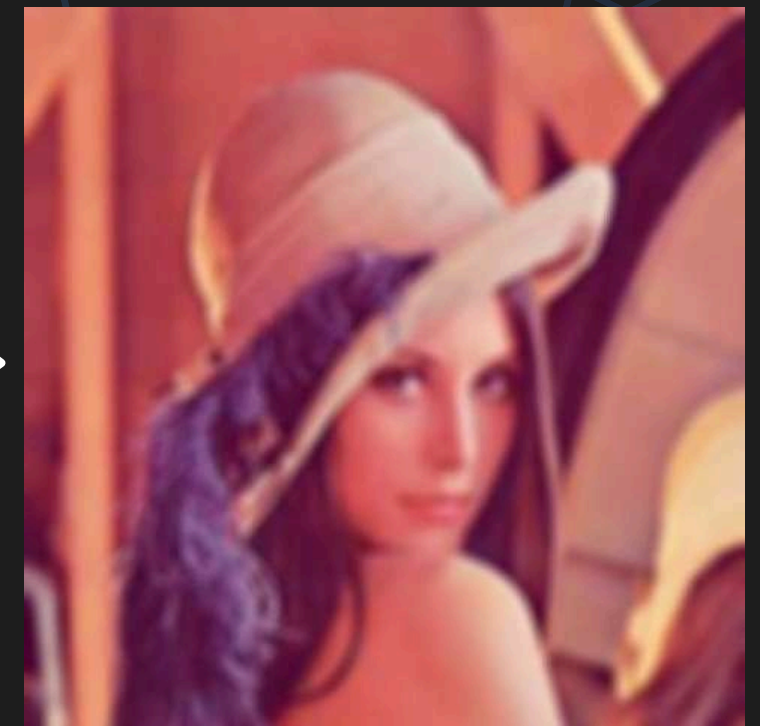
## Filtrado de imágenes





Imagen original

# Operaciones de filtrado



Resultado



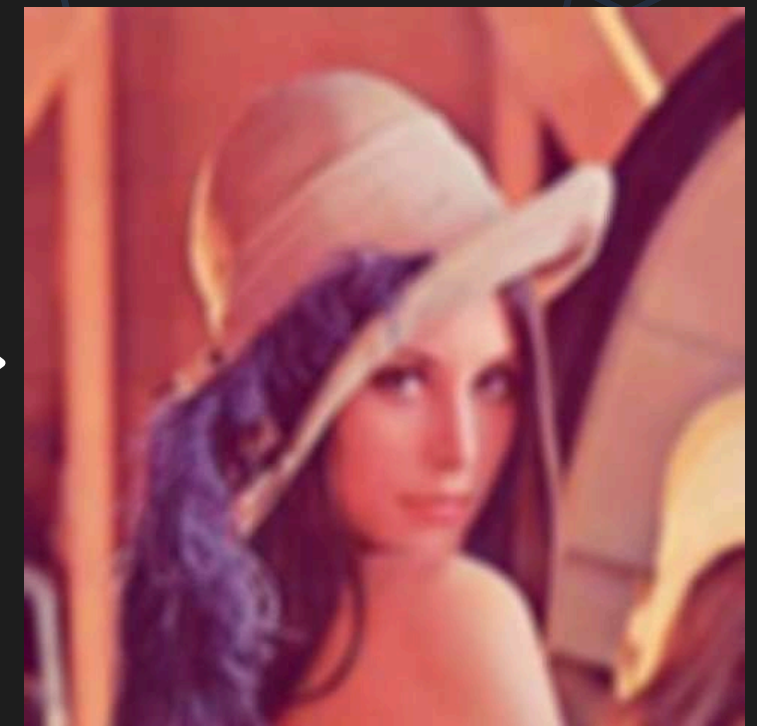
Imagen original



$$\begin{bmatrix} 1 & 2 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Matriz (kernel)

$\omega$



Resultado

## Kernels

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

**Identidad**

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

**Sharpen**

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

**Ridge detection**

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

**Box blur**

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

**Edge detection**

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

**Gaussian blur**

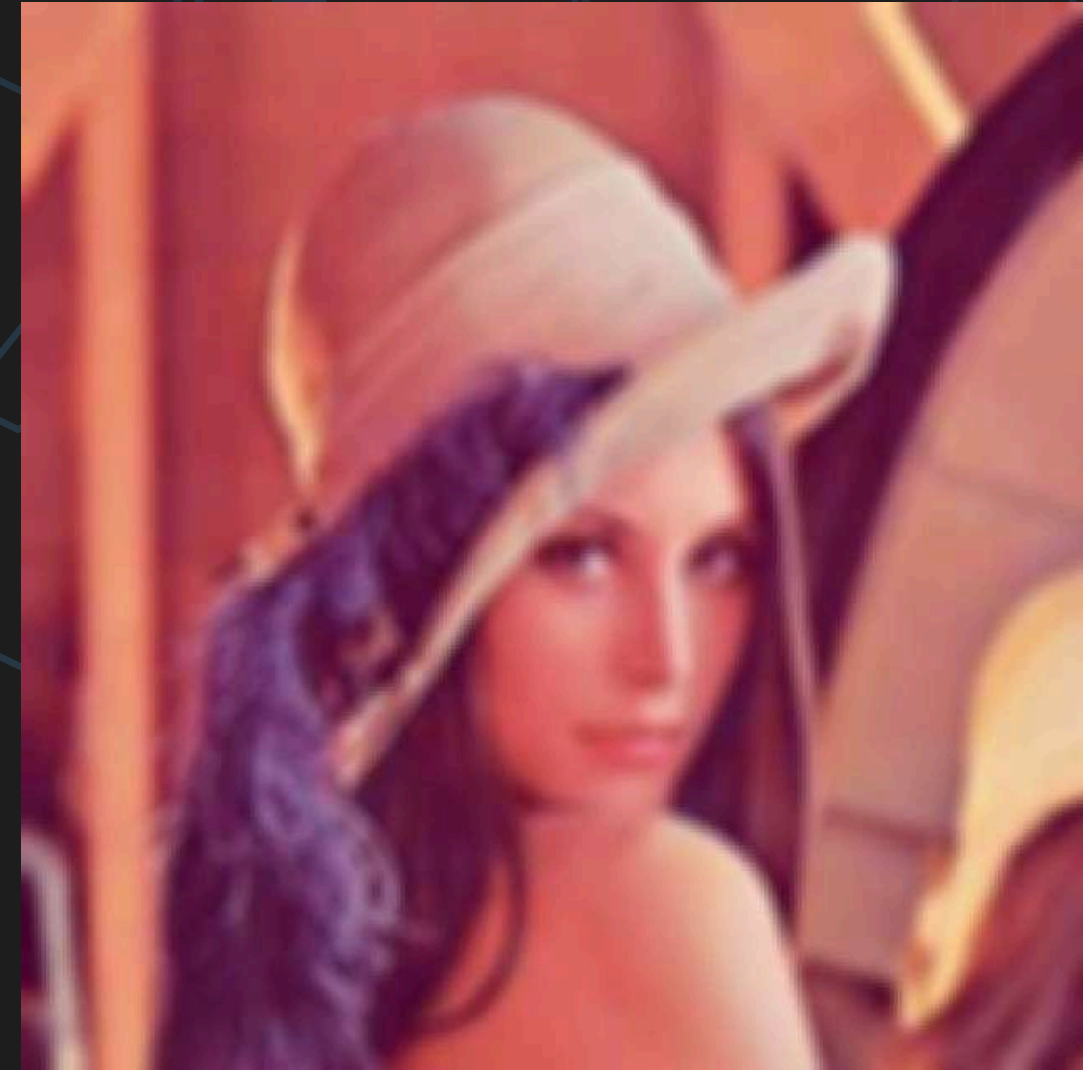


Original



1	234	100	89
4	54	12	0
16	35	47	201
189	127	134	150

Filtrada



0	117	50	44
2	27	6	0
8	17	23	100
145	65	71	75





# Practica Quitafondos



# Practica

## Detección de bordes



## Detección de bordes

- La detección de bordes es una técnica esencial que extrae los contornos de los objetos en una imagen, localizando transiciones bruscas de intensidad.
- Se basa en detectar los gradientes de las imágenes, es decir, el cambio de “dirección” de los píxeles en la imagen.
- Las derivadas o los gradientes nos indicarán la “dirección” que sigue el borde y la “fuerza” de ese cambio.
- Algunas aplicaciones de la detección de bordes:
  - Segmentación de objetos
  - Reconocimiento de formas
  - Detección de contornos
  - OCR

Original Image



Edge Detected



## Detección de bordes

- Al igual que los filtros, tenemos varios tipos de kernels que nos permiten identificar ciertos tipos de ejes y contornos.

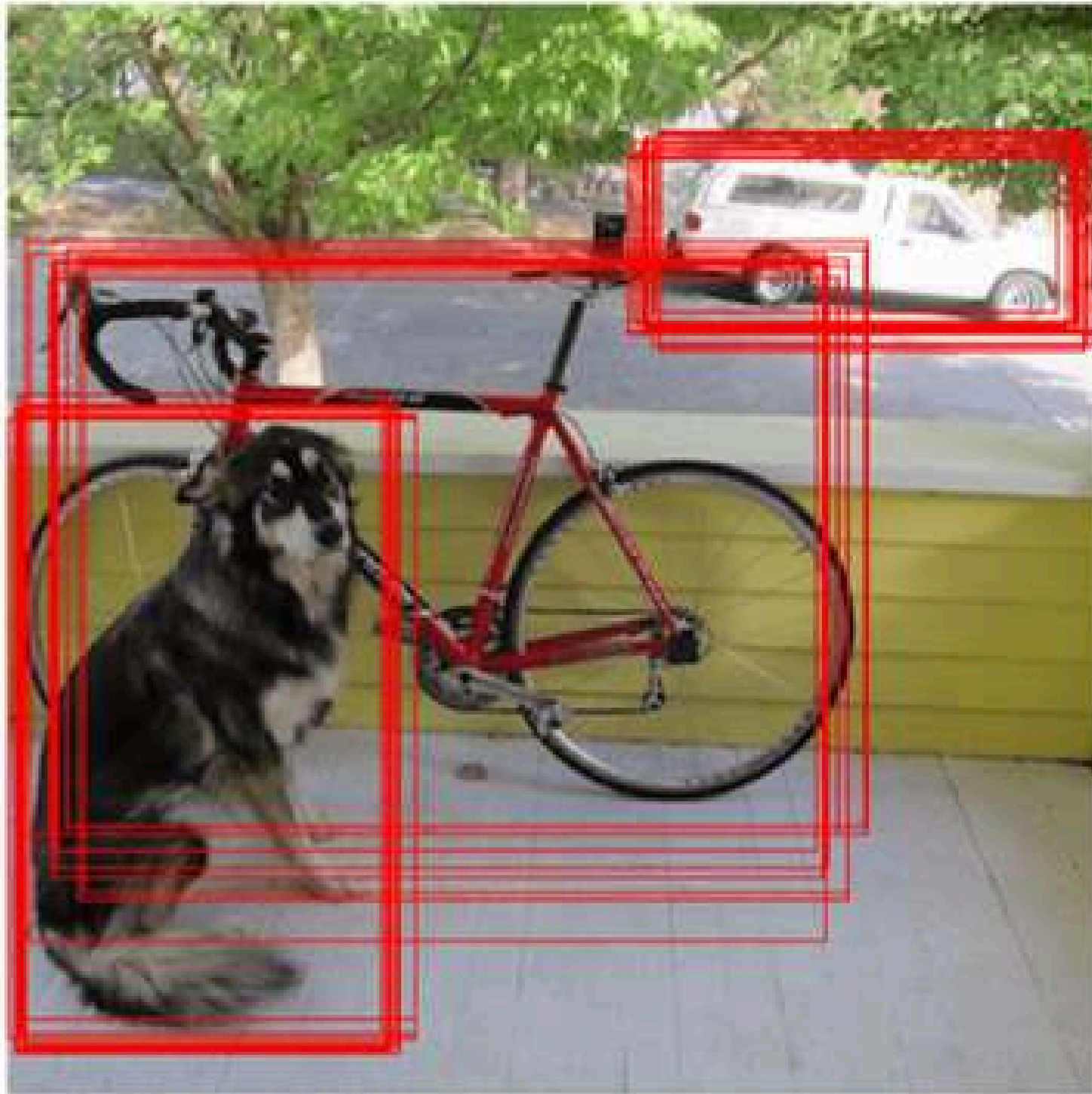
Método	Principio	Utilidad
Sobel	Convolución 3×3 (derivada de primer orden)	Detección robusta Suavizado ligero
Prewitt	Convolución 3×3 (derivada de primer orden)	Rapido y eficiente
Roberts	Convolución 3×3 (derivada de primer orden)	Mejor para figuras complejas o con bordes muy curvados
Canny	Algoritmo multi-etapa	Bordes finos y conectados OCR

# Practica

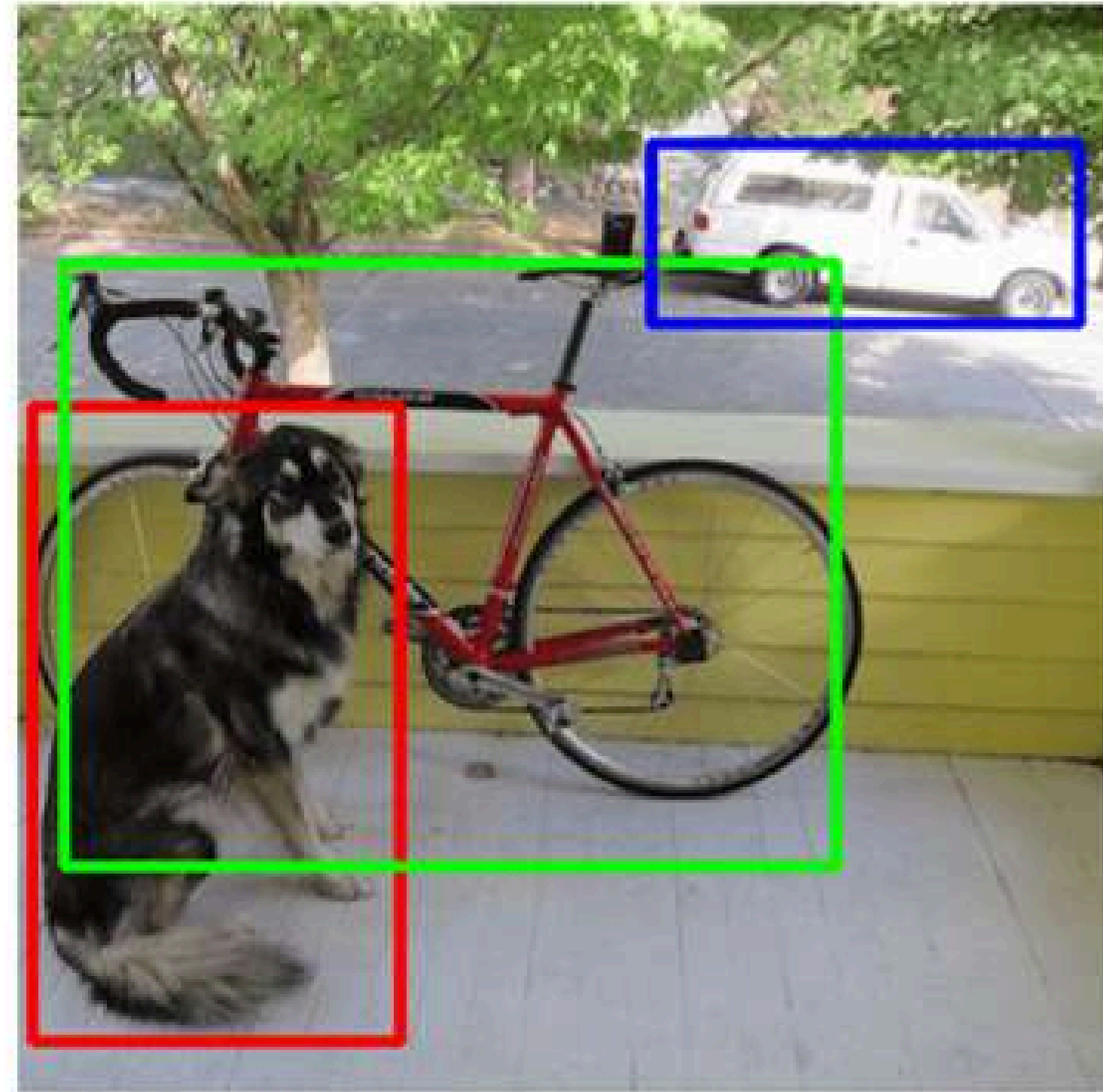
## Contador de objetos



Output Predictions



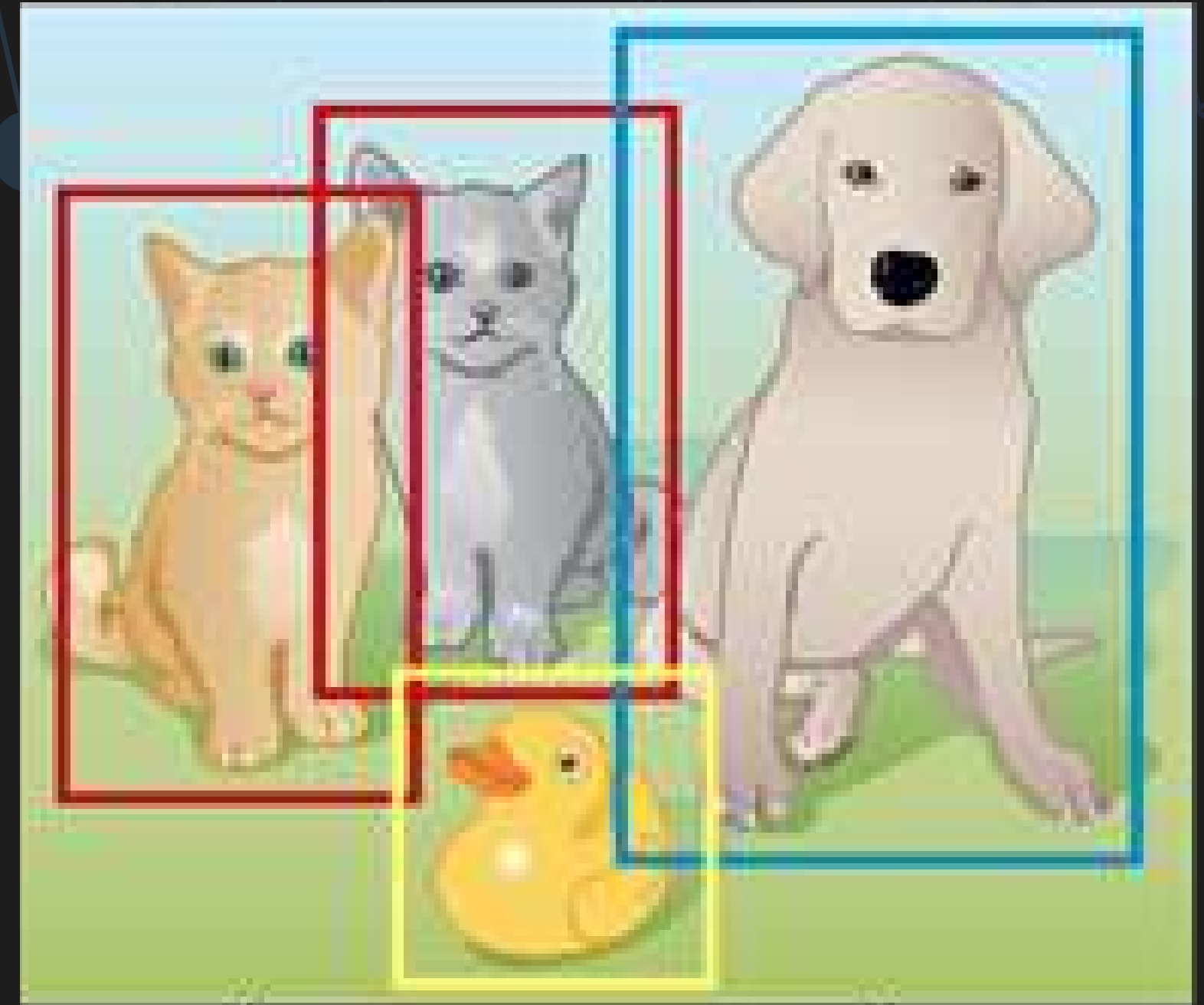
Non-Maximum Suppression (NMS)





# Non maximum supression

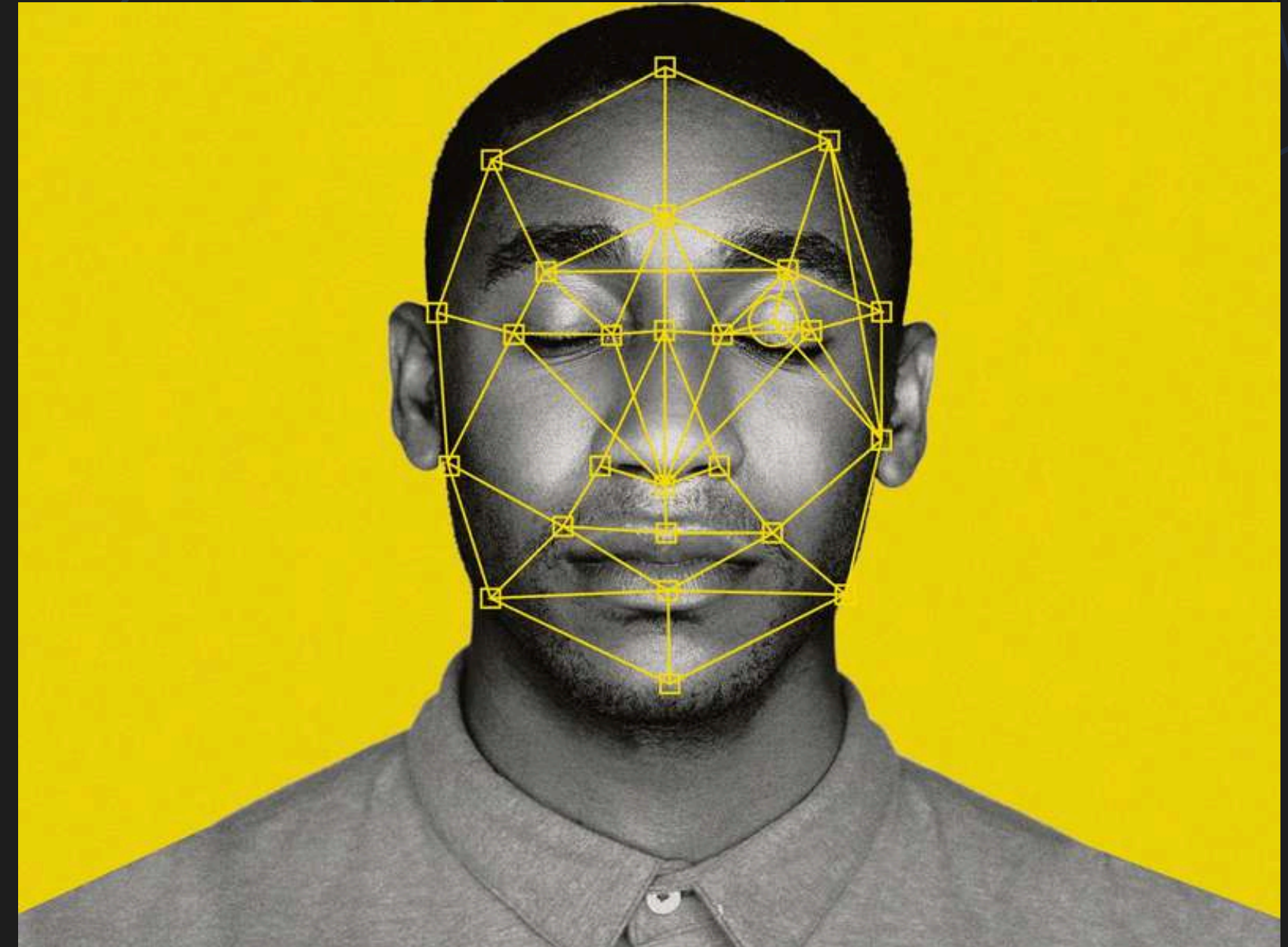
- Con NMS eliminamos el solapamiento de las bounding boxes y de paso las ordena según la confianza o score de nuestro algoritmo.
- NMS tiene la ventaja de que se adapta fácilmente a otros algoritmos de detección, además de ser bastante eficiente a pesar de todas las etapas que conlleva.



# Proyecto final

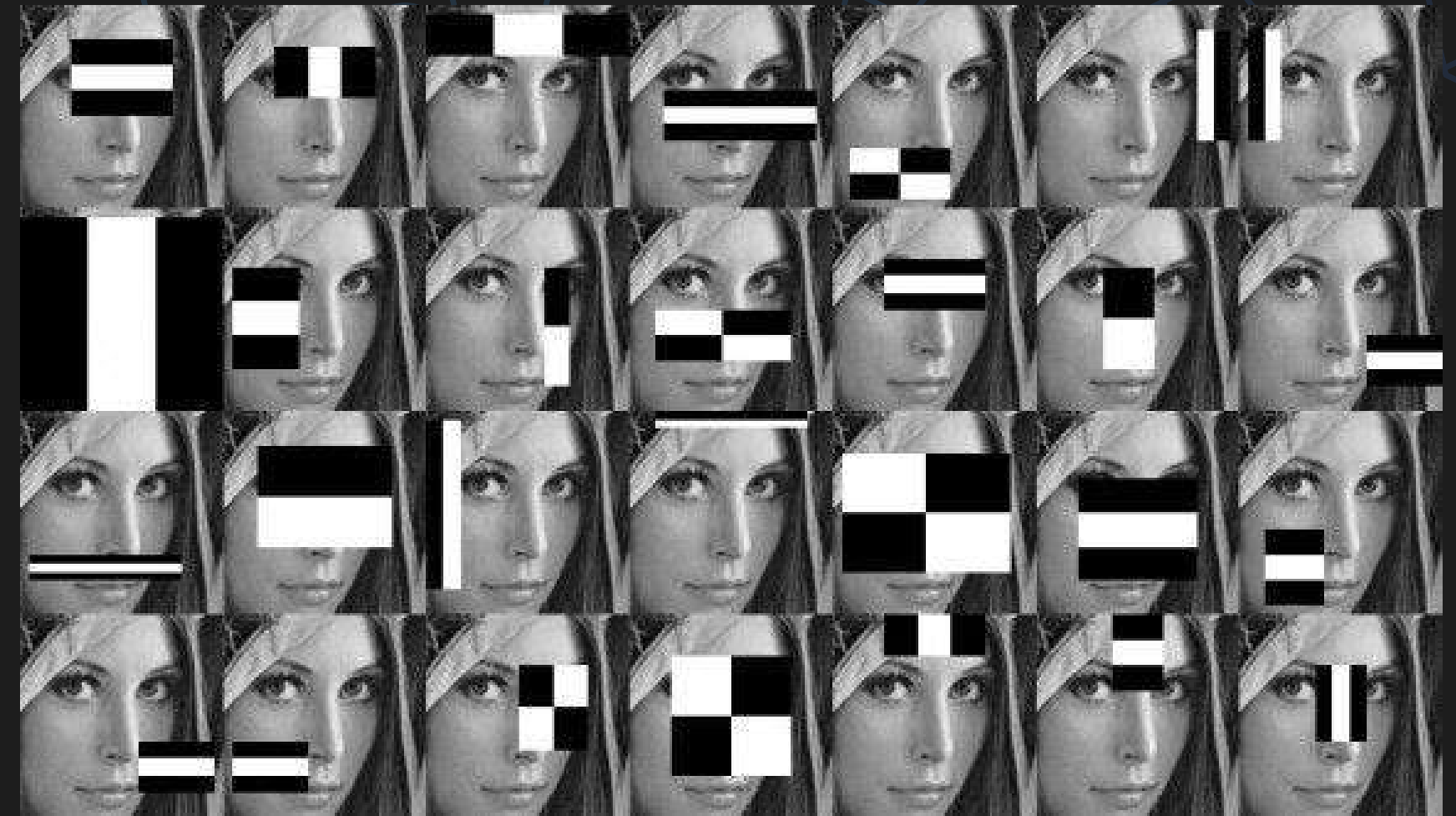
# Detección de rostros

- Un programa biométrico que mapea las características de un rostro con una identidad.
- Parte de la idea de que cada rostro es único e irrepetible.
- Se pueden utilizar fotografías o videos.
- Variedad de aplicaciones:
  - Seguridad y vigilancia
  - Entretenimiento
  - Detección de enfermedades
  - Reconocimiento de emociones

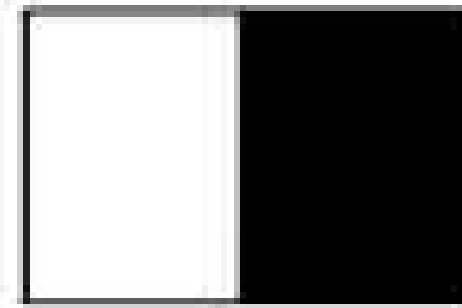


# Detección de rostros – Haar Cascade

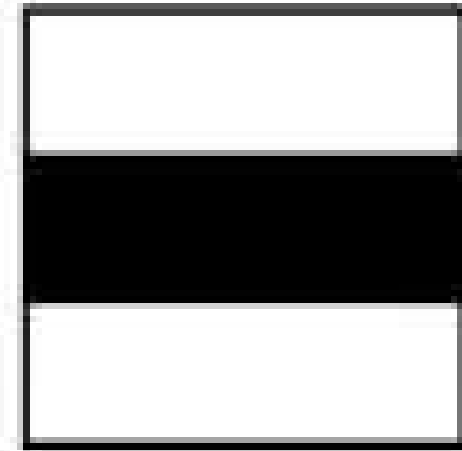
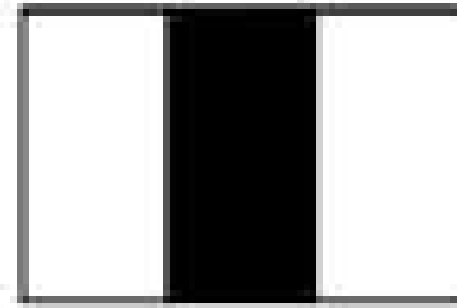
- Inspirados en la forma que ve el ser humano (bordes y contrastes).
- Detectan patrones claros y oscuros en las imágenes.
- Cada “feature” representa una sección del rostro o área de interés.
- Hay cuatro tipos básicos de *features*: bordes (verticales y horizontales), líneas y centro.
- Puede haber combinaciones para hacer la detección más robusta.



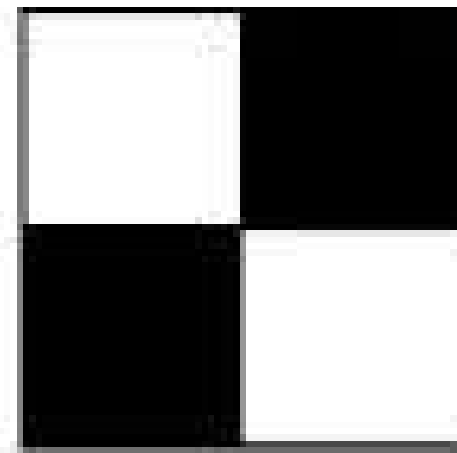
# Haar Cascade features



(a) Edge Features



(b) Line Features



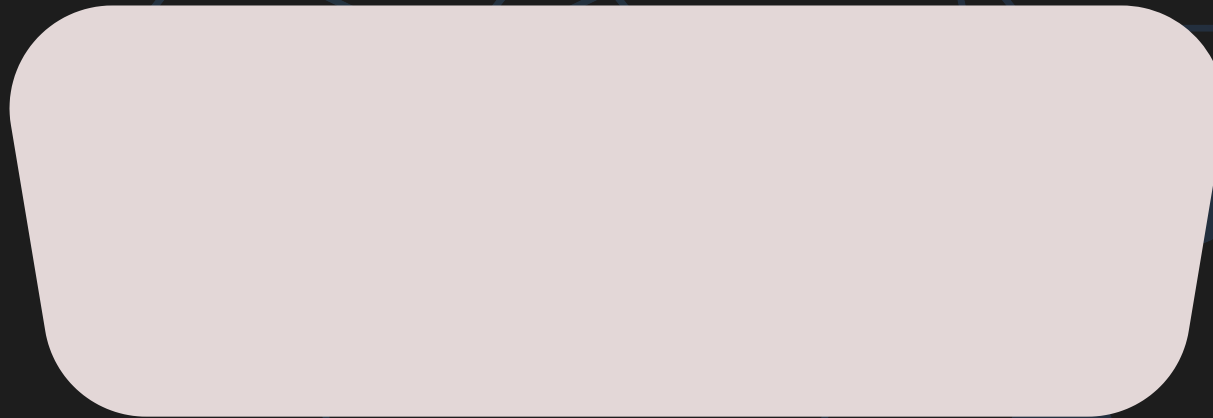
(c) Four-rectangle features

# Haar Cascade features

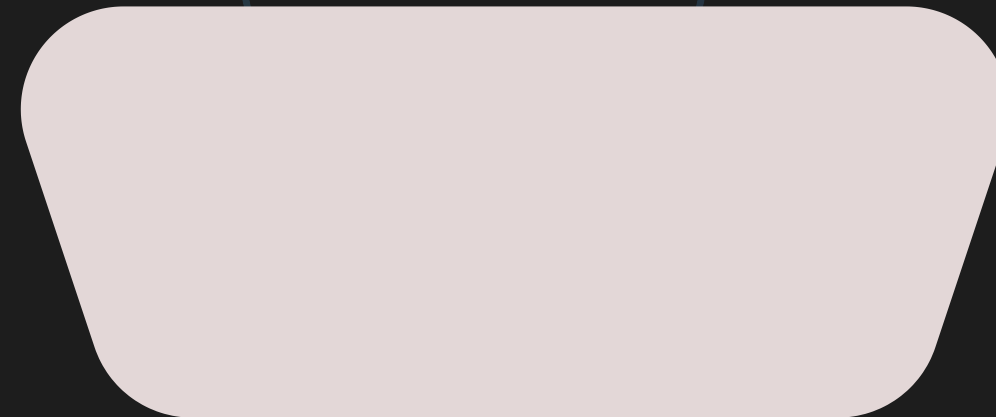


# Haar Cascade features

**Características simples**



**Características  
complejas**



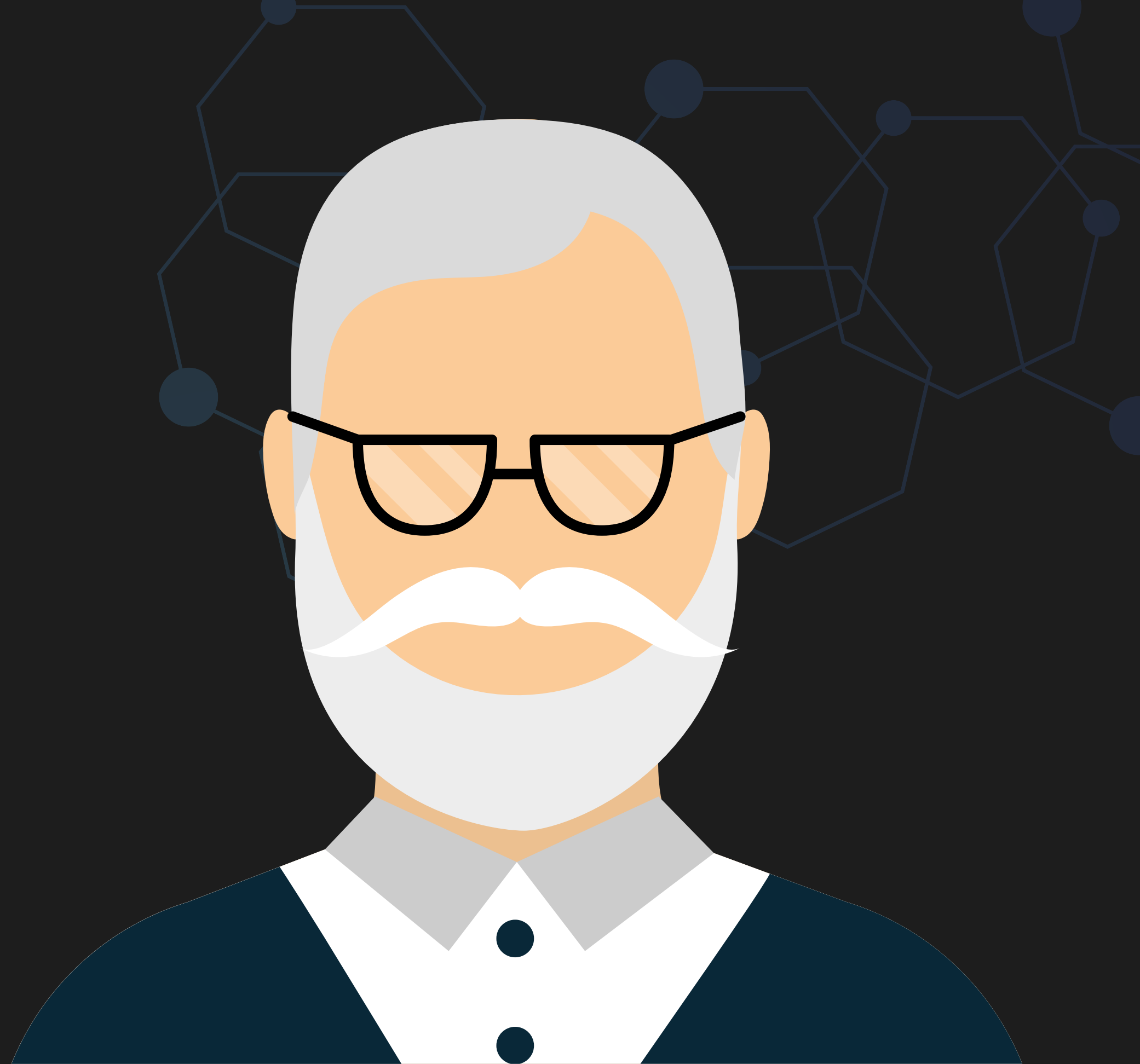
**Características  
mas complejas**





# Haar Cascade features

- A pesar de ser un algoritmo bastante rápido y eficiente con buenos resultados es susceptible a cambios en el rostro como el uso de lentes, barba.
- También presenta problemas cuando el rostro está de perfil.
- También si hay un cambio importante en la iluminación, puede haber falsos positivos.

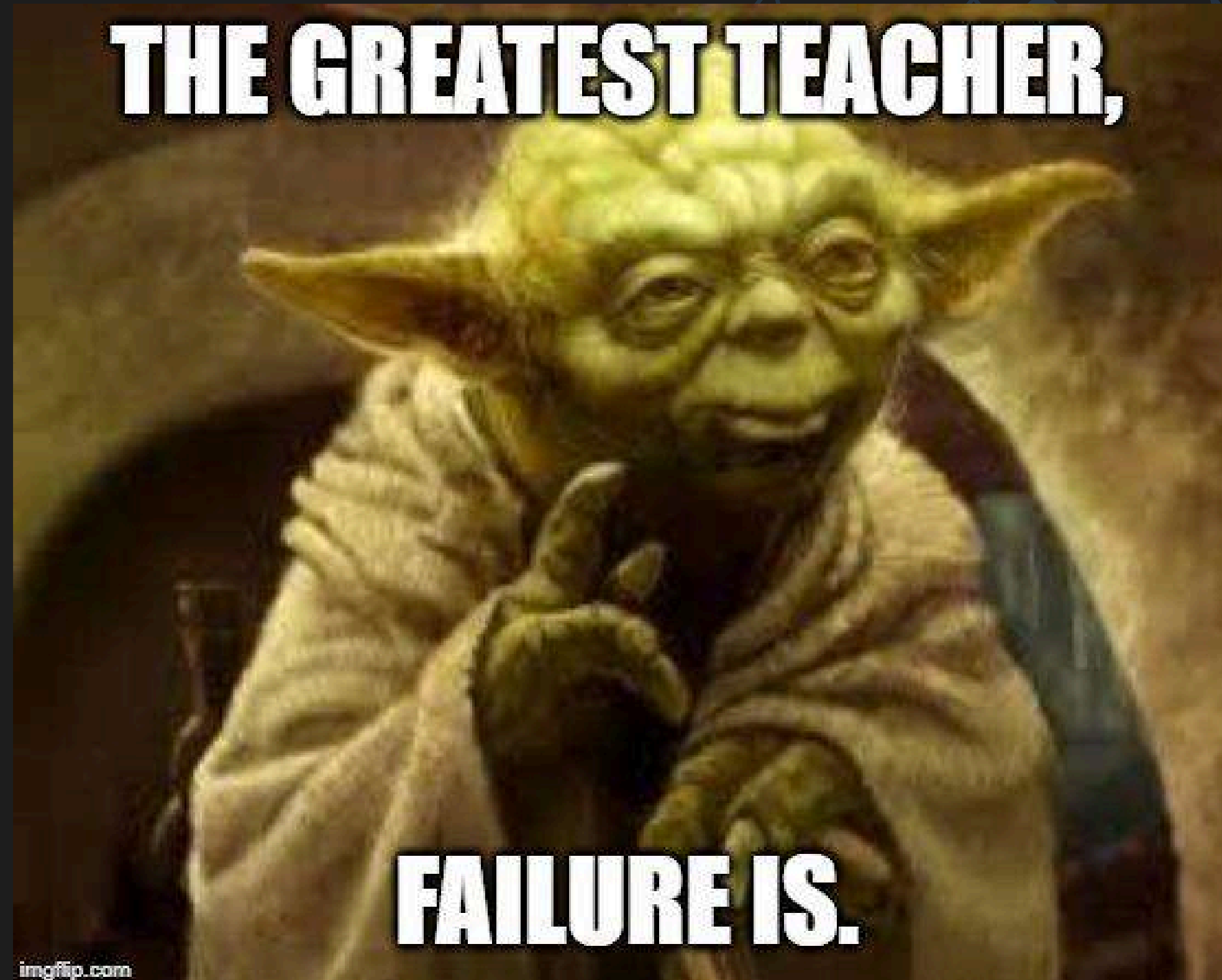


# Haar Cascade features

- Nuestro proyecto consiste en usar estas características Haar para escanear nuestro rostro y aplicarnos un filtro.
- Vamos a identificar la nariz y parte de la frente y sobreponer la nariz de perro y las orejas sobre esas zonas.
- Ajustar según nuestro rostro para que calce a la perfección.



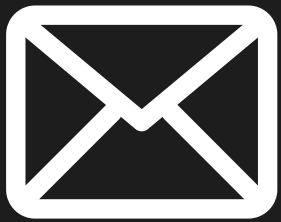
# Ya para terminar



# Bibliografía

- From zero to hero, José Portilla, Udemy, 2019
- Learning to program: the fundamentals, University of Toronto, Coursera, 2016.
- Python tutorials, w3schools.com
- Python Docs, python.org
- Computer Vision, Cristopher W. Tyler, O'Reilly.
- Practical Machine Learning for Computer Vision, Valliappa Lakshmanan, Martin Görner, Ryan Gillard, O'Reilly.

# Datos de Contacto



**amhrdz.1001@gmail.com**



**312 118 26 25**



**in/angelmoises-hernandezponce**



**DrKingKoma**



**amoiseshrdz**

