Problema 1
Procesamiento de Imágenes y Detección de Formas

IWG-101

Universidad Técnica Federico Santa María

Departamento de Informática

2018-2

Procesamiento de imágenes

- Son métodos que realizan ciertas operaciones sobre una imagen con el objetivo de extraer información útil.
- Una imagen se representa como una matriz donde cada punto x, y describe el brillo o el color del pixel.
- En el modelo RGB cada pixel se representa como un vector de 3 componentes (Red, Green, Blue) con valores entre 0 y 255.

Procesamiento de imágenes

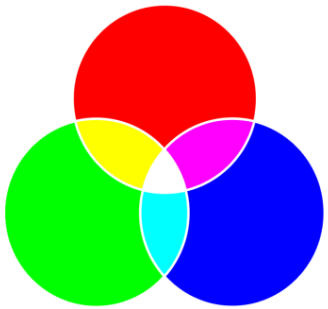
Problema

Detectar formas y colores en una imagen.

OpenCV

Librería Open Source para Visión Computacional y Machine Learning.
`https://docs.opencv.org/master/d6/d00/tutorial_py_root.html`

Detección de colores con OpenCV



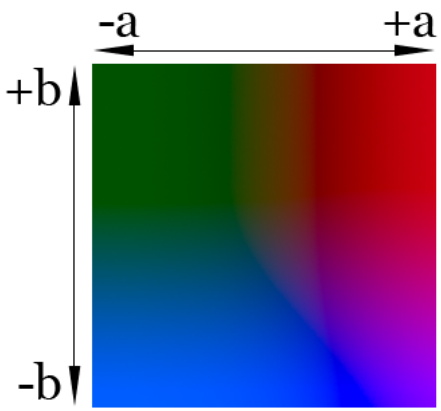
Detección de colores con OpenCV

```python
# diccionario de colores en formato RGB
colorDictionary = {
    "rojo": (255, 0, 0),
    "verde": (0, 255, 0),
    "azul": (0, 0, 255), ... }

# crear un arreglo (matriz) con NumPy para guardar los colores
labColors = np.zeros((len(colorDictionary), 1, 3), dtype="uint8")
labColorNames = []

# iterar sobre los colores del diccionario
for (i, (name, rgb)) in enumerate(colorDictionary.items()):
    # actualizar el arreglo con el color en RGB
    labColors[i] = rgb
    labColorNames.append(name)

# convertir el arreglo al espacio de color Lab desde RGB
labColors = cv2.cvtColor(labColors, cv2.COLOR_RGB2LAB)
```

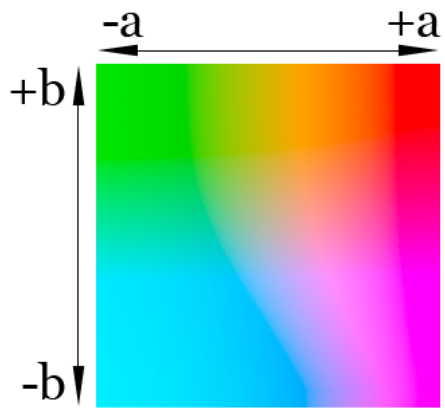
Detección de colores con OpenCV

¿Por qué se utiliza el espacio L*a*b* en lugar de RGB o HSV?

R: Para detectar colores calcularemos la distancia Euclidiana entre nuestro arreglo de colores y una región en particular de la imagen. El color que minimice la distancia será escogido. A diferencia de RGB o HSV, Lab es un espacio *perceptivamente lineal*.



Luminosidad al 25%.



Luminosidad al 75%.

Detección de colores con OpenCV

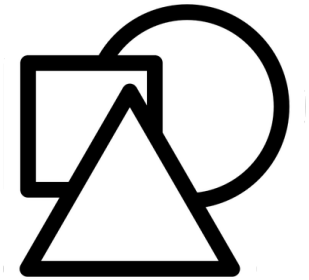
Para detectar colores debe implementar la función **detectColor(image, c)** que recibe la imagen en espacio de colores Lab y un contorno. Debe retornar un string con el nombre del color detectado.

```
def detectColor(image, c):
        # ...
        return "rojo"
```

También debe implementar la función **detectOppositeColor(image, c)** que recibe la imagen en espacio de colores Lab y un contorno. Debe retornar un string con el nombre del color, distinto de negro, más alejado al detectado.

```
def detectOppositeColor(image, c):
        # ...
        return "rojo"
```

Detección de formas con OpenCV



Pre-procesamiento de la imagen con OpenCV I

Funciones que permiten mejorar las condiciones de la imagen antes de aplicar métodos de detección.

Redimensionar la imagen

```
resized = imutils.resize(image, width=300)
```

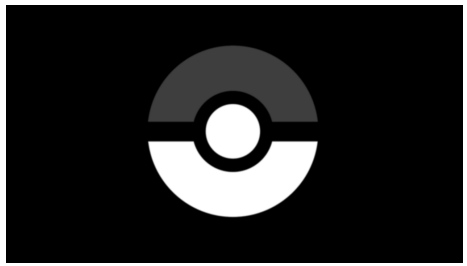
Suavizar para reducir el ruido con difuminado

```
blurred = cv2.GaussianBlur(image, (5, 5), 0)
```

Pre-procesamiento de la imagen con OpenCV II

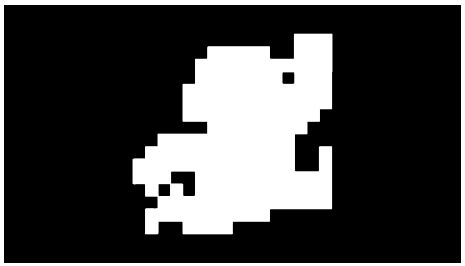
Convertir a escala de grises

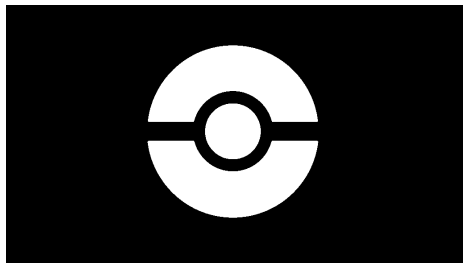
```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```



Simplificar detalles para mejorar análisis visual

```
thresh = cv2.threshold(gray, 20, 255, cv2.THRESH_BINARY)[1]
```

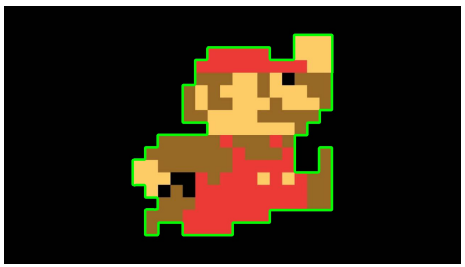


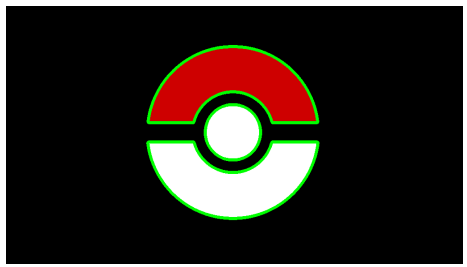


Pre-procesamiento de la imagen con OpenCV III

Buscar contornos en la imagen

```
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)
```





Manipular contornos con OpenCV

OpenCV provee una serie de funciones para manipular contornos:

Perímetro del contorno

```
perimeter = cv2.arcLength(cnt,True)
```

Delimitador de rectángulos

```
x,y,w,h = cv2.boundingRect(cnt)
```

`(x,y)`: coordenada superior, izquierda.

`(w,h)`: ancho, alto.

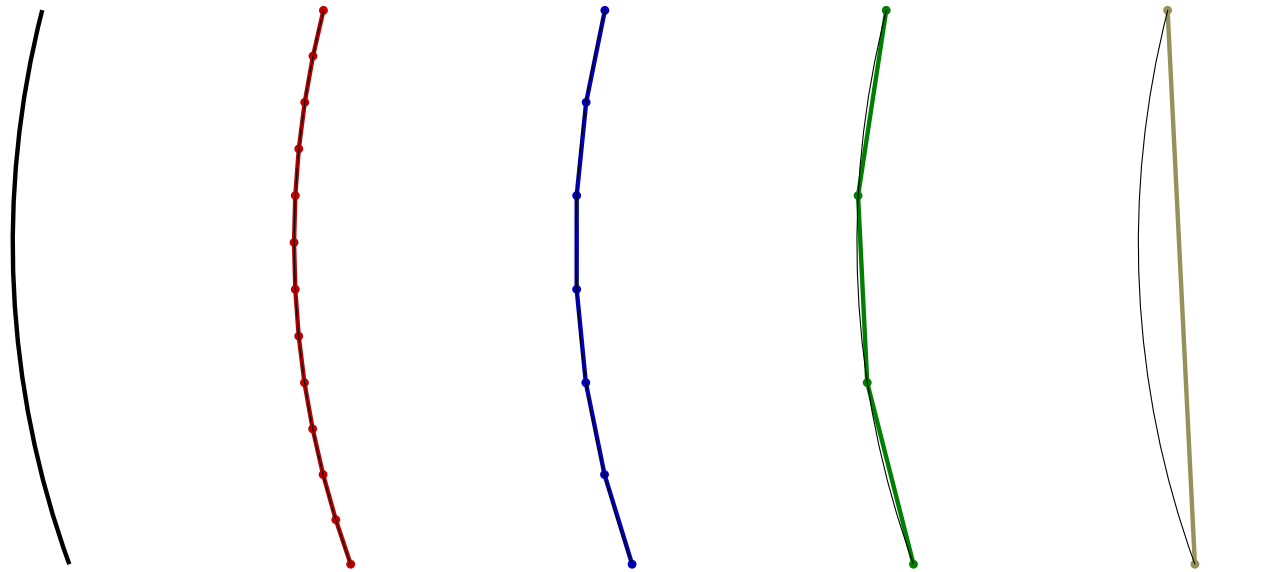
Manipular contornos con OpenCV

Aproximar forma

```
epsilon = 0.05 * perimeter
approx = cv2.approxPolyDP(cnt,epsilon,True)
```

`epsilon`: precisión recomendada entre 1-5% del perímetro.

Ejemplo de aproximación de un contorno detectado usando diferentes valores para `epsilon` (alto `epsilon` implica menor resolución):



Funciones Adicionales (Módulo `Utilities.py`)

Se proveerá de una serie de funciones útiles para detección de formas:

```
distance(p1,p2)
```

```
edgeLength(cntr,i1,i2)
```

```
equal(v1, v2, margin)
```

```
internalAngle(cntr,i)
```

```
isConvex(cntr)
```

```
...
```

Detección de formas con OpenCV

Para detectar formas debe implementar la función **detectShape(cntr, margin)** que recibe un contorno y un margen de error. Debe retornar la tupla **(shape,triangle,quad)** que contenga el nombre de la figura geométrica detectada según su número de lados (triángulo: 3, cuadrilátero: 4, pentágono: 5, ...), el tipo de triángulo (equilátero, escaleno, isósceles) y cuadrilátero (cuadrado, rectángulo, rombo, romboide, trapecio, trapezoide).

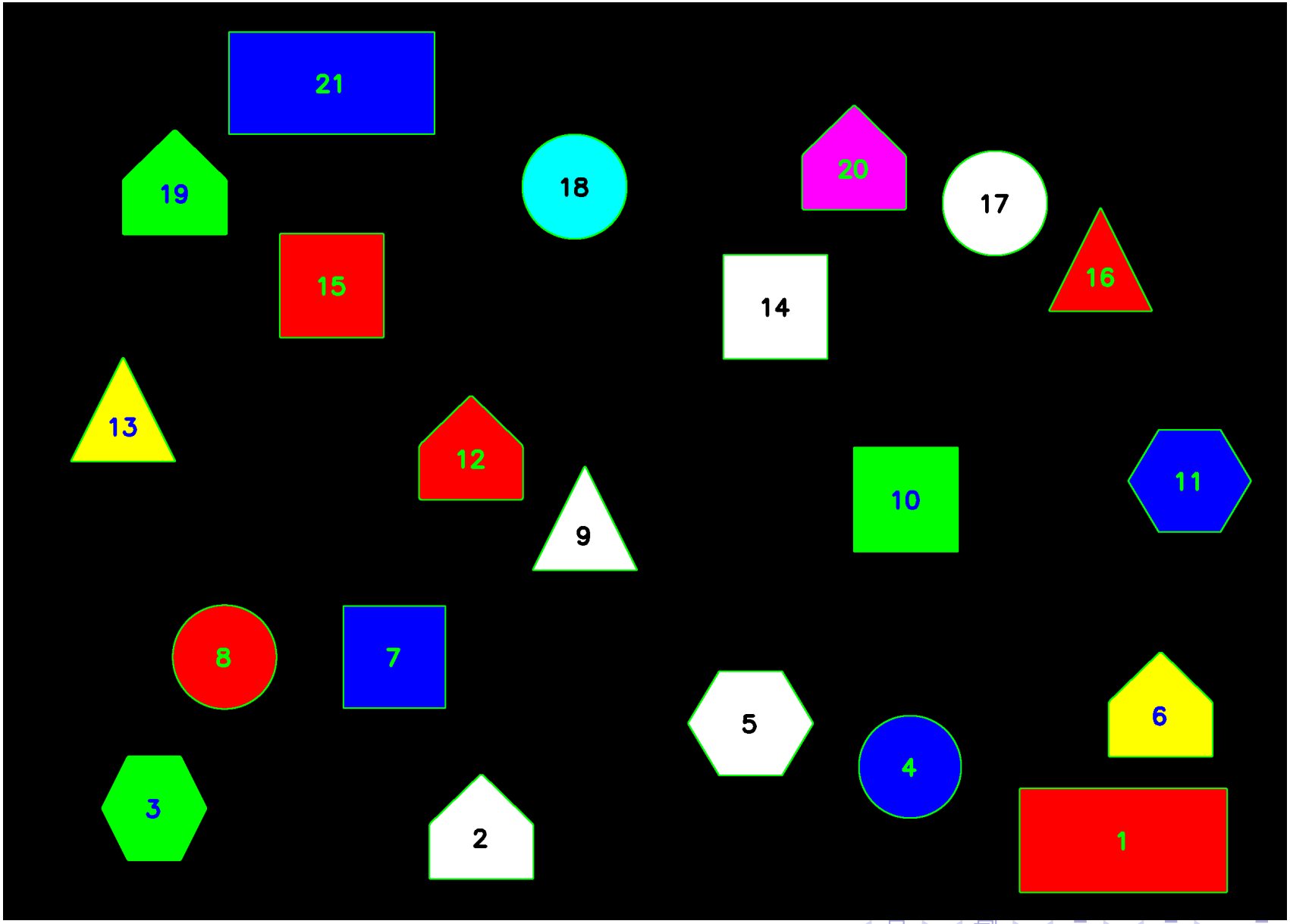
Ver Anexos para mayor información sobre clasificación de figuras geométricas.

```
def detectShape(cntr,margin):
    # ...
    return shape,triangle,quad
```

Código base para detectar formas y colores

```
for c in cnts:
    # calcular el centroide del contorno
    M = cv2.moments(c)
    cX = int((M["m10"] / M["m00"]) * ratio)
    cY = int((M["m01"] / M["m00"]) * ratio)
    approx = cv2.approxPolyDP(c, 0.05 * peri, True)

    # detectar la forma y color usando el contorno (IMPLEMENTAR!)
    shape,triangle,quad = detectShape(approx,equalityMargin)
    color = detectColor(lab, c)

    # multiplicar las coordenadas del contorno segun la redimension
    c = c.astype("int")

    # dibujar el contorno y la forma detectada en la imagen
    cv2.drawContours(image, [approx], -1, (0, 255, 0), 2)

# guardar la imagen
cv2.imwrite("NewImagen",image)
```

Resultado Esperado



Resultado Esperado (Ejemplo con errores)

detectado.txt

```
1,cuadrilatero,rojo,no triangulo,cuadrado
2,pentagono,blanco,no triangulo,no cuadrilatero
3,hexagono,verde,no triangulo,no cuadrilatero
4,octogono,azul,no triangulo,no cuadrilatero
...
```

real.txt

```
1,cuadrilatero,rojo,no triangulo,rectangulo
2,pentagono,blanco,no triangulo,no cuadrilatero
3,hexagono,verde,no triangulo,no cuadrilatero
4,circulo,azul,no triangulo,no cuadrilatero
...
```

Nota: Cada imagen viene con su archivo de etiquetas reales.

Validar resultado

Matriz de Confusión: Herramienta que permite visualizar el desempeño del algoritmo de clasificación.

		Detectado	
		+	-
Real	+	TP	FN
	-	FP	TN

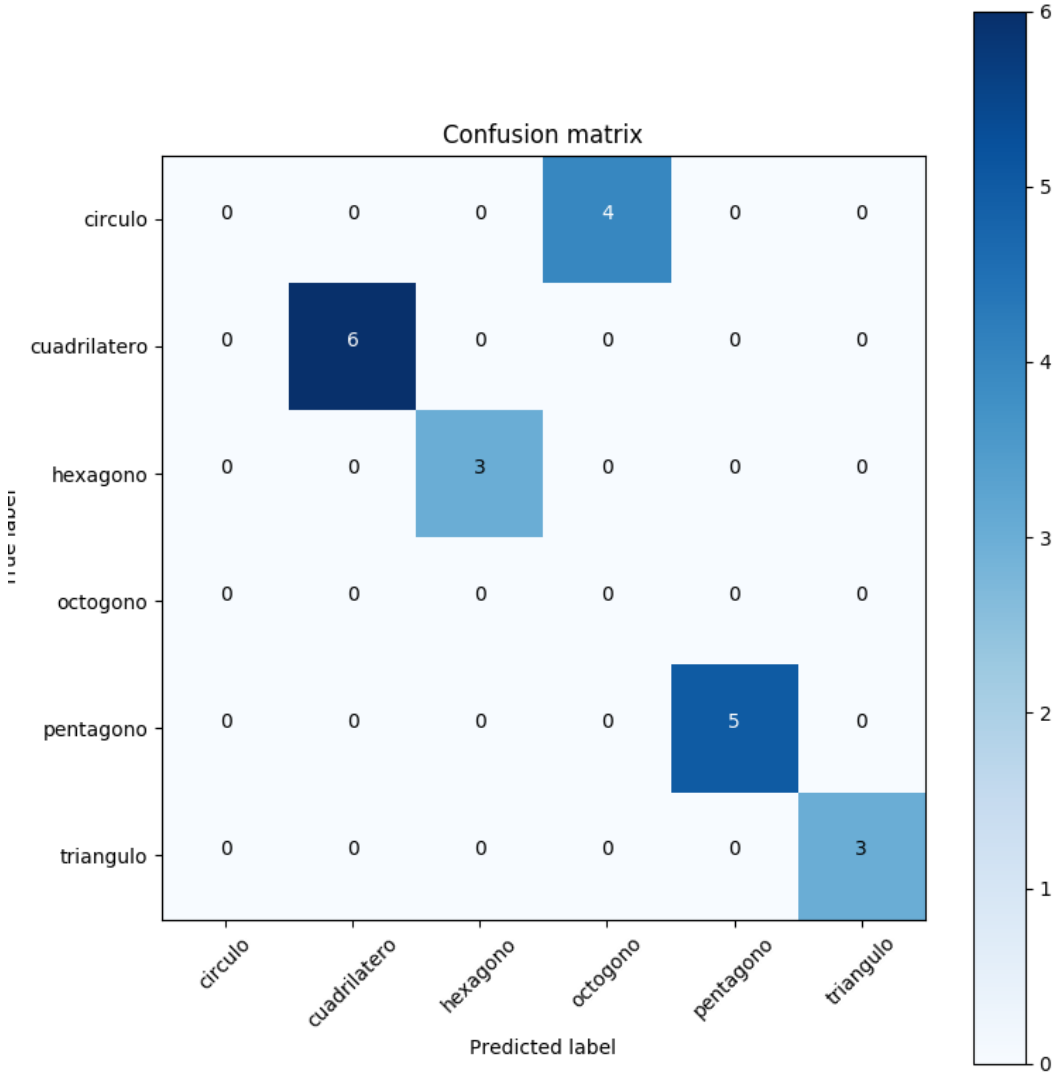
Precisión: que proporción de los objetos detectados como positivos es correcta.

$$\frac{TP}{TP + FP}$$

Recall: que proporción de los objetos positivos se detectó correctamente.

$$\frac{TP}{TP + FN}$$

Matriz de Confusión para detección de forma



En resumen...

1. Instalar Jupyter Notebook y todas las librerías necesarias para el problema (OpenCV, SciPy, etc.).
2. Analizar el código entregado y pensar como resolver el problema.
3. Implementar las funciones `detectColor`, `detectOppositeColor` y `detectShape`:
   - `detectColor` retorna un string con el nombre del color detectado.
   - `detectOppositeColor` retorna un string con el nombre del color, distinto de negro, más alejado al detectado.
   - `detectShape` retorna una tupla con el nombre de la figura geométrica detectada según su número de lados (triángulo, cuadrilátero, pentágono, ...) y el tipo de triángulo y cuadrilátero. Ver Anexos.
4. Calcular la matriz de confusión para cada detección.
5. Preparar una presentación en Jupyter Notebook que incluya todo el código fuente y las conclusiones del trabajo (la presentación debe permitir modificar código y parámetros en vivo).
6. Exponer su solución en clases.

Conclusiones

Debe intentar responder estas (u otras) preguntas:

- ¿Qué limitaciones presenta su método de detección?
- ¿Es posible detectar todo el espectro de colores?
- ¿Se detectan todas las figuras geométricas?
- ¿Se detectan y/o clasifican correctamente los tipos de figuras geométricas?

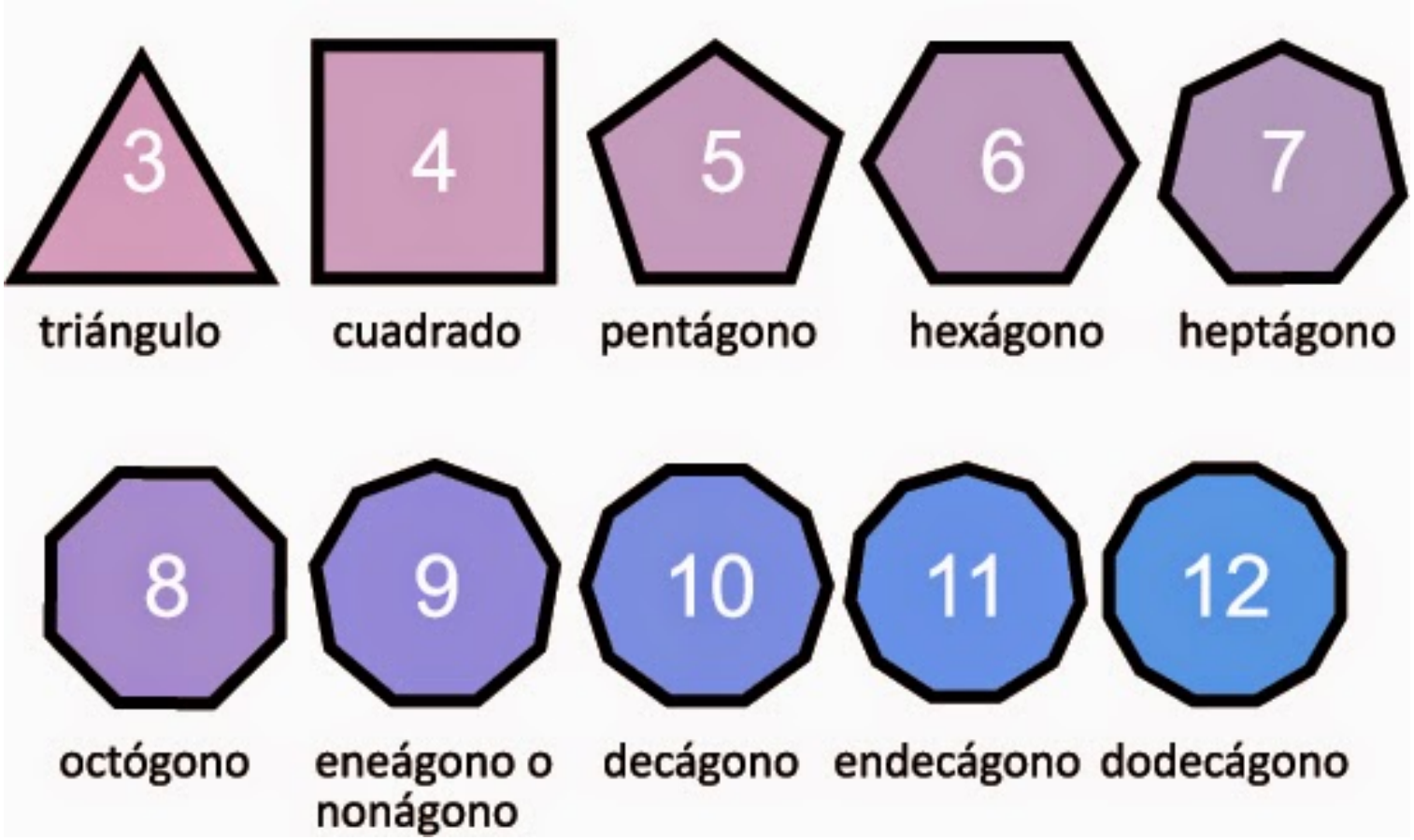
Condiciones de Entrega

- El trabajo completo debe ser desarrollado en **Jupyter Notebook**, incluyendo el código y la presentación.
- Puede incluir más imágenes pero **debe funcionar** para las cinco (5) que vienen junto al código.
- La entrega es **máximo una semana** a partir de conocer el problema en clases vía `moodle.inf.utfsm.cl`.
- Todos los integrantes del equipo deben presentar la solución.

Anexos

Clasificación de figuras geométricas I

Según su número de lados:



Nota: Se asume que el círculo tiene más de doce lados.

Clasificación de figuras geométricas II

CUADRILÁTEROS