In [1]:
```
# Our Setup, Import Libaries, Create our Imshow Function and Download our Images
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Define our imshow function
def imshow(title = "Image", image = None, size = 10):
    w, h = image.shape[0], image.shape[1]
    aspect_ratio = w/h
    plt.figure(figsize=(size * aspect_ratio,size))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.show()
```

In [2]:
```
# Adding comma zero in cv2.imread loads our image in as a grayscaled image
image = cv2.imread('download.jpg',0)
imshow("Grayscaled Image",image)
print(image)

# Create a matrix of ones, then multiply it by a scaler of 100
# This gives a matrix with same dimensions of our image with all values being 100
M = np.ones(image.shape, dtype = "uint8") * 100
```

Grayscaled Image

```
[[ 46  97 113 ...  63  79  51]
 [ 89 138 101 ...  80  87  75]
 [ 84  90  26 ...  69  62  73]
 ...
 [ 26  23  51 ...  71  72  73]
 [ 42  35  73 ...  56  57  58]
 [ 48  17  18 ...  68  68  68]]
```

In [3]:
```
print(M)
```

```
[[100 100 100 ... 100 100 100]
 [100 100 100 ... 100 100 100]
 [100 100 100 ... 100 100 100]
 ...
 [100 100 100 ... 100 100 100]
 [100 100 100 ... 100 100 100]
 [100 100 100 ... 100 100 100]]
```

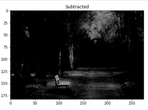
In [4]:
```
# We use this to add this matrix M, to our image
# Notice the increase in brightness
added = cv2.add(image, M)
imshow("Increasing Brightness", added)

# Now if we just added it, look what happens
added = image + M
imshow("Simple Numpy Adding Results in Clipping", added2)
```

Increasing Brightness

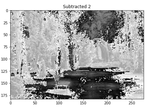
Simple Numpy Adding Results in Clipping

In [5]:
```
# Likewise we can also subtract
# Notice the decrease in brightness
subtracted = cv2.subtract(image, M)
imshow("Subtracted", subtracted)

subtracted = image - M
imshow("Subtracted 2", subtracted)
```

Subtracted

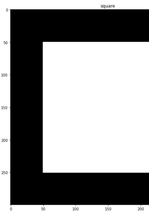


Subtracted 2

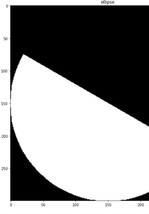


In [6]:
```
# If you're wondering why only two dimensions, well this is a grayscale image,

# Making a square
square = np.zeros((300, 300), np.uint8)
cv2.rectangle(square, (50, 50), (250, 250), 255, -2)
imshow("square", square)

# Making a ellipse
ellipse = np.zeros((300, 300), np.uint8)
cv2.ellipse(ellipse, (150, 150), (150, 150), 30, 0, 180, 255, -1)
imshow("ellipse", ellipse)
```

square

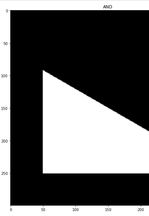


ellipse

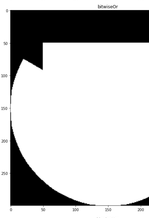


In [7]:
```
# Shows only where they intersect
And = cv2.bitwise_and(square, ellipse)
imshow("AND", And)

# Shows where either square or ellipse is
bitwiseOr = cv2.bitwise_or(square, ellipse)
imshow("bitwiseOr", bitwiseOr)

# Shows where either exist by itself
bitwiseXor = cv2.bitwise_xor(square, ellipse)
imshow("bitwiseXor", bitwiseXor)

# Shows everything that isn't part of the square
bitwiseNot_sq = cv2.bitwise_not(square)
imshow("bitwiseNot_sq", bitwiseNot_sq)
```

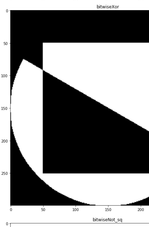
AND



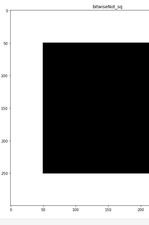
bitwiseOr



bitwiseXor



bitwiseNot_sq



In [ ]:
```
```