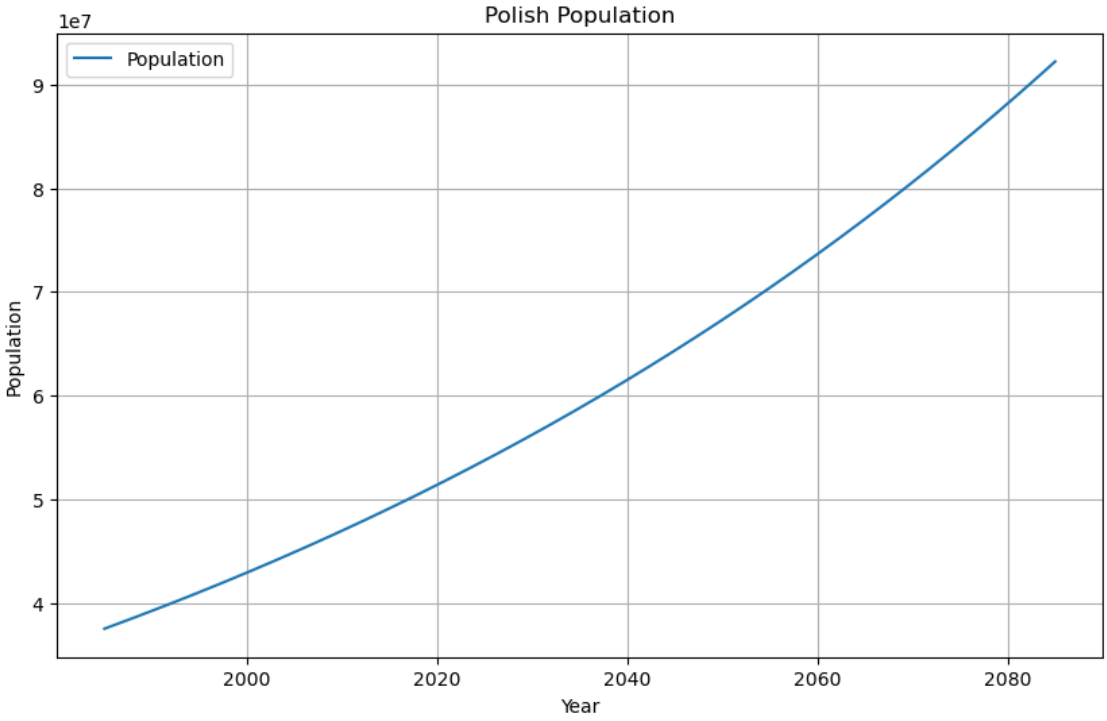
Population and Carrying Capacity

November 21, 2024

[1]:
```
#SIRPLOT Poland
import matplotlib.pyplot as plt

t_data, P_data = [], []
tinitial = 1985
tfinal = 2085
P = 37500000

numsteps = 100000
deltat = (tfinal - tinitial)/numsteps
t = tinitial

while (t<=tfinal):

    #Append current values
    t_data.append(t)
    P_data.append(P)

    #Calculate Pprime
    Pprime = .009*P

    deltaP = Pprime * deltat

    #Update P and t for next iteration
    P = P + deltaP
    t = t + deltat


    #print ("t = " + str(t), "S = " + str(S), "I = " + str(I), "R = " + str(R))

plt.figure(figsize=(10, 6))
plt.plot(t_data, P_data, label='Population')
plt.xlabel('Year')
plt.ylabel('Population')
plt.title('Polish Population')
plt.legend()
plt.grid(True)
```

```
plt.show()

print ('Final population in 2085: ', P_data[-1])
```




```
Final population in 2085:  92233913.0139442
```

[2]:
```
# Logistics Equation 2.2 -2


for j in range (20):
    y = 100
    tinitial = 0
    tfinal = 37
    numsteps = 2**(j-1)
    deltat = (tfinal - tinitial)/numsteps
    t = tinitial
    while (t<= tfinal):
        yprime = .1 * y * (1 - y/1000)
        deltay = yprime * deltat
        y = y + deltay
        t = t + deltat
    print(numsteps, y)
```

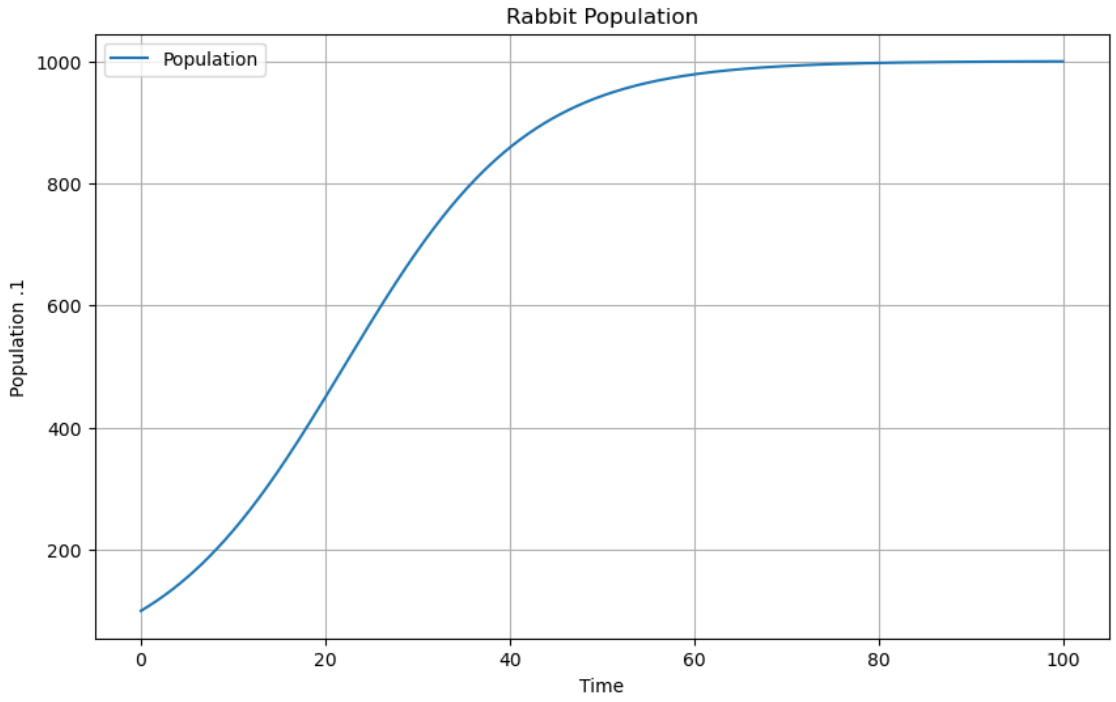
```
0.5 766.0
```

```
1 1341.3907
2 1060.2600189219188
4 925.7175423217548
8 870.185685171989
16 843.8713442060501
32 830.8942303207776
64 824.4344205058576
128 821.2098232551336
256 819.5986191145495
512 818.7932615437581
1024 818.390640231662
2048 818.1893434865547
4096 818.0886985344986
8192 818.0383769064324
16384 818.0132163034949
32768 818.0006360546901
65536 817.9943459434477
131072 817.9912008911032
262144 817.9896283657584
```

2.2 3 a) As t gets larger the value of y stabilizes near 1000 b) At y(0) of 1000, the population wouldn't change because it the (1 - y/1000) would become 0 therefore the whole yprime would be 0. c) At y(0) of 1500 the population would initially decrease because (1-y/1000) would be a negative number. d) Other than the obvious requirement for more than 1 rabbit to make more rabits. .1 * y = 0 if y(0)=0 therefore yprime would be 0 signifying no change. e) In the case of rabbits the obvious carrying capacity is the amount of rabbits the food available in the area can support. More than that and eventually rabbits will die off until there is enough for the survivors. This could also apply to water, shelter, or any other limiting factor for survival.

[3]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 100
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .1 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population')
plt.xlabel('Time')
```

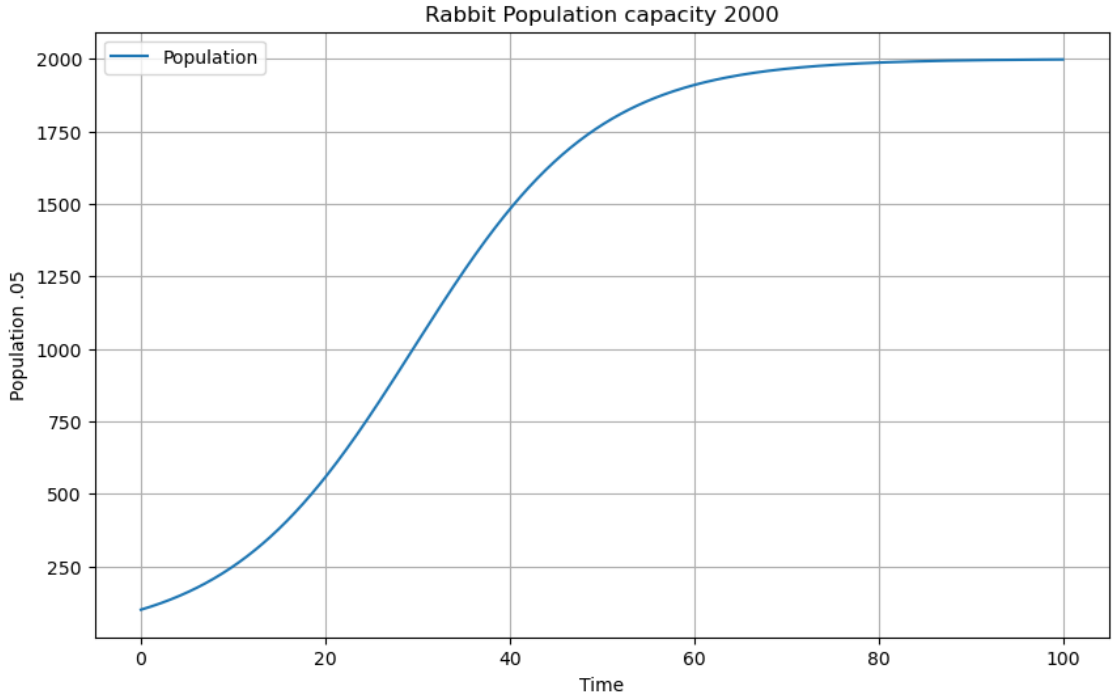
```
plt.ylabel('Population .1')
plt.title('Rabbit Population')
plt.legend()
plt.grid(True)
plt.show()
```



2.2 4 I haven't figured out plotting multiple different runs on the same graph so here are a couple graph examples. As carrying capacity is increased the population limits out at the new carrying capacity but interestingly takes about the same about of time to do so (in this case approximately 80 days.)

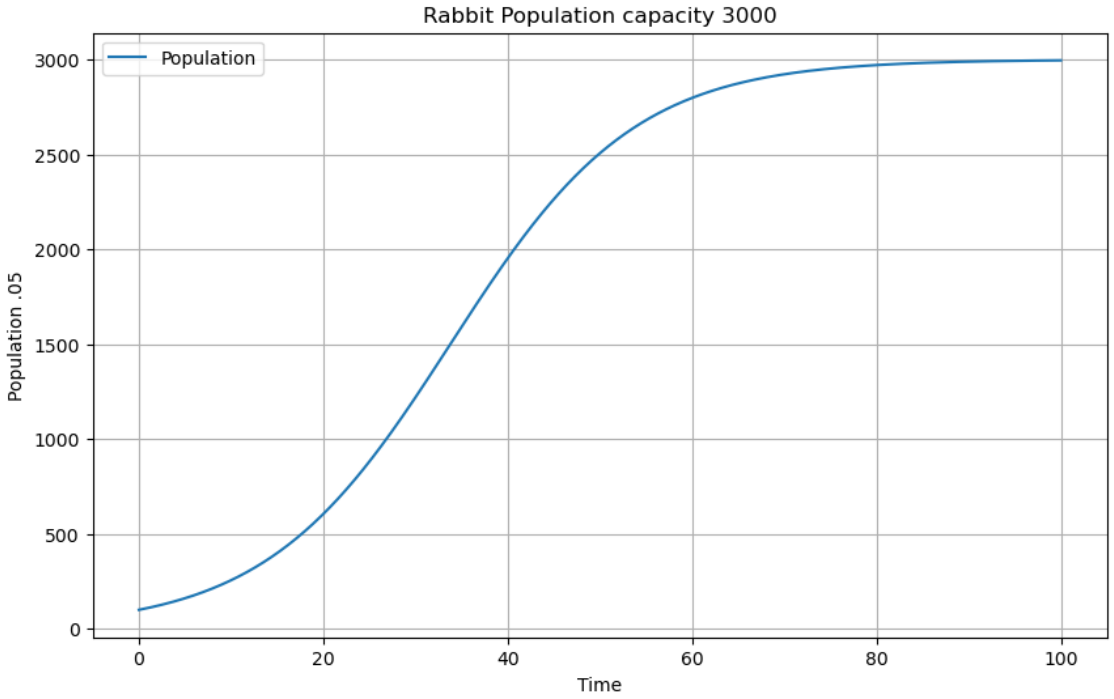
[4]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 100
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
```

```
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .1 * y * (1 - y/2000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population')
plt.xlabel('Time')
plt.ylabel('Population .05')
plt.title('Rabbit Population capacity 2000')
plt.legend()
plt.grid(True)
plt.show()
```



[5]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 100
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
```

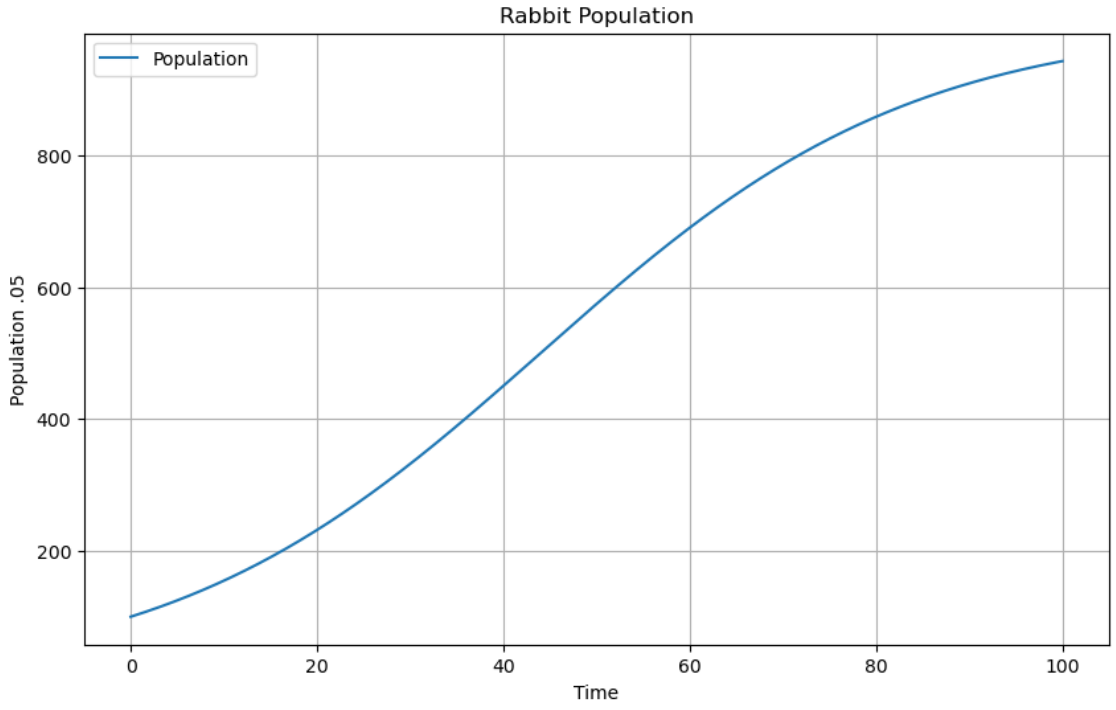
```
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .1 * y * (1 - y/3000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population')
plt.xlabel('Time')
plt.ylabel('Population .05')
plt.title('Rabbit Population capacity 3000')
plt.legend()
plt.grid(True)
plt.show()
```



2.2 5

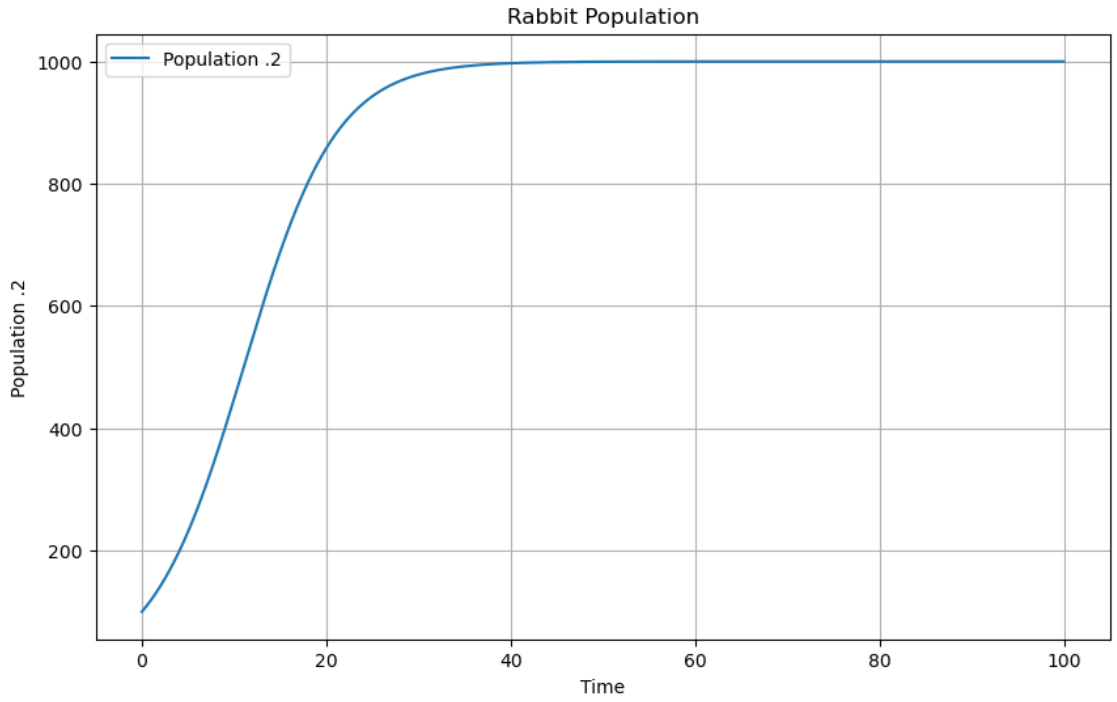
[6]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 100
numsteps = 1000
```

```
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .05 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population')
plt.xlabel('Time')
plt.ylabel('Population .05')
plt.title('Rabbit Population')
plt.legend()
plt.grid(True)
plt.show()
```



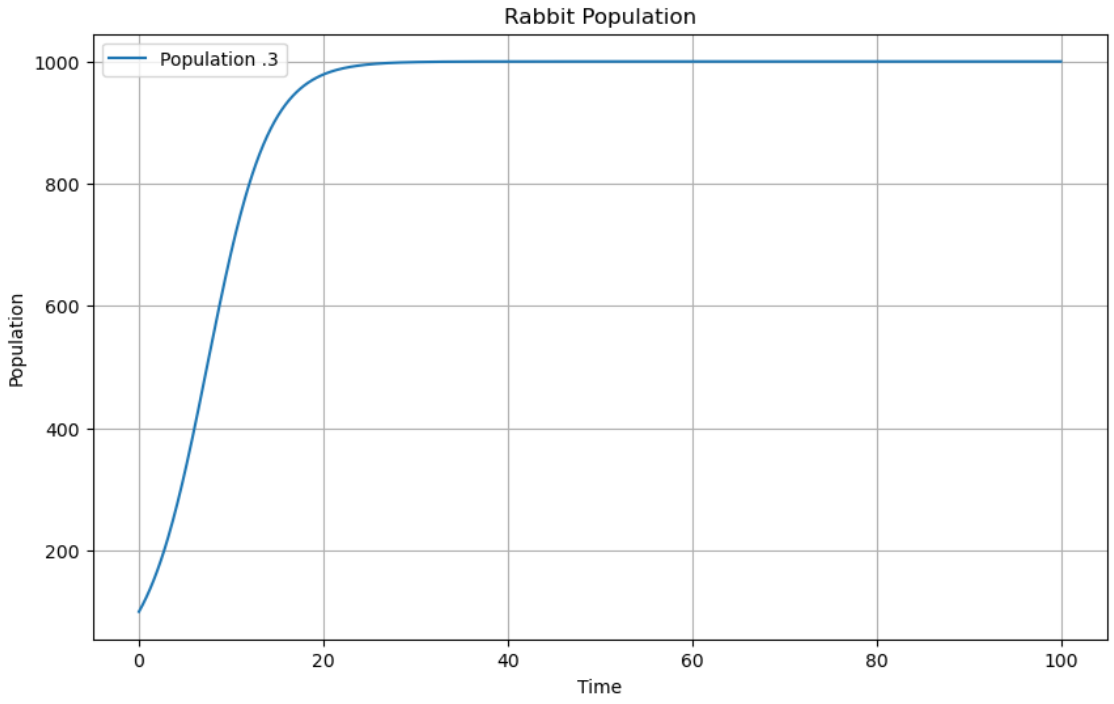
[7]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 100
```

```
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .2 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population .2')
plt.xlabel('Time')
plt.ylabel('Population .2')
plt.title('Rabbit Population')
plt.legend()
plt.grid(True)
plt.show()
```



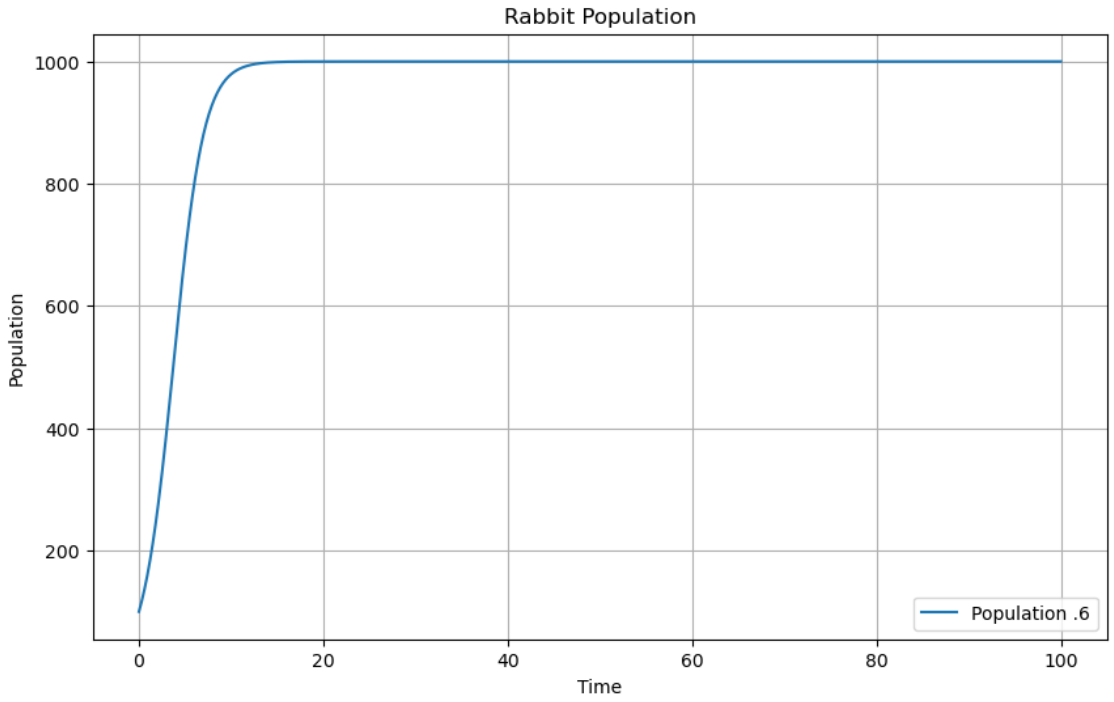
[8]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
```

```
tfinal = 100
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .3 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population .3')
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Rabbit Population')
plt.legend()
plt.grid(True)
plt.show()
```



[9]:
```
y = 100
t_datarabbit =  []
y_datarabbit = []
```

```
tinitial = 0
tfinal = 100
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .6 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat

plt.figure(figsize=(10, 6))
plt.plot(t_datarabbit, y_datarabbit, label='Population .6')
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('Rabbit Population')
plt.legend()
plt.grid(True)
plt.show()
```



As the constant increases the rapidness of increase towards the carrying capacity increases.

[10]:
```
#2.2 - 7

y = 100
t_datarabbit =  []
y_datarabbit = []
tinitial = 0
tfinal = 20
numsteps = 1000
deltat = (tfinal - tinitial)/numsteps
t = tinitial
while (t<= tfinal):
    t_datarabbit.append(t)
    y_datarabbit.append(y)
    yprime = .2196 * y * (1 - y/1000)
    deltay = yprime * deltat
    y = y + deltay
    t = t + deltat
print (y)

# I ran this multiple times ajusting the constant until I just barely was over␣
 ↪900 at 20 days at .2196 constant
```

```
900.174969004851
```