Exercise Lesson 1-1

Lucía Martín Fernández

In [1]:
```python
import matplotlib.pyplot as plt
import numpy as np
```

In the previous programming exercise, you modeled a discrete-time version of the logistic equation. The discrete-time logistic equation is an example of a discrete map, that is, a function that recursively maps input values from a specific interval to output values from the same interval. This definition implies that if we start from a value x0 located inside the interval, the subsequent values will never leave the interval. The interval in which the discrete map "works" is called the map's domain. In the case of the logistic map, its domain is the interval [0,1].

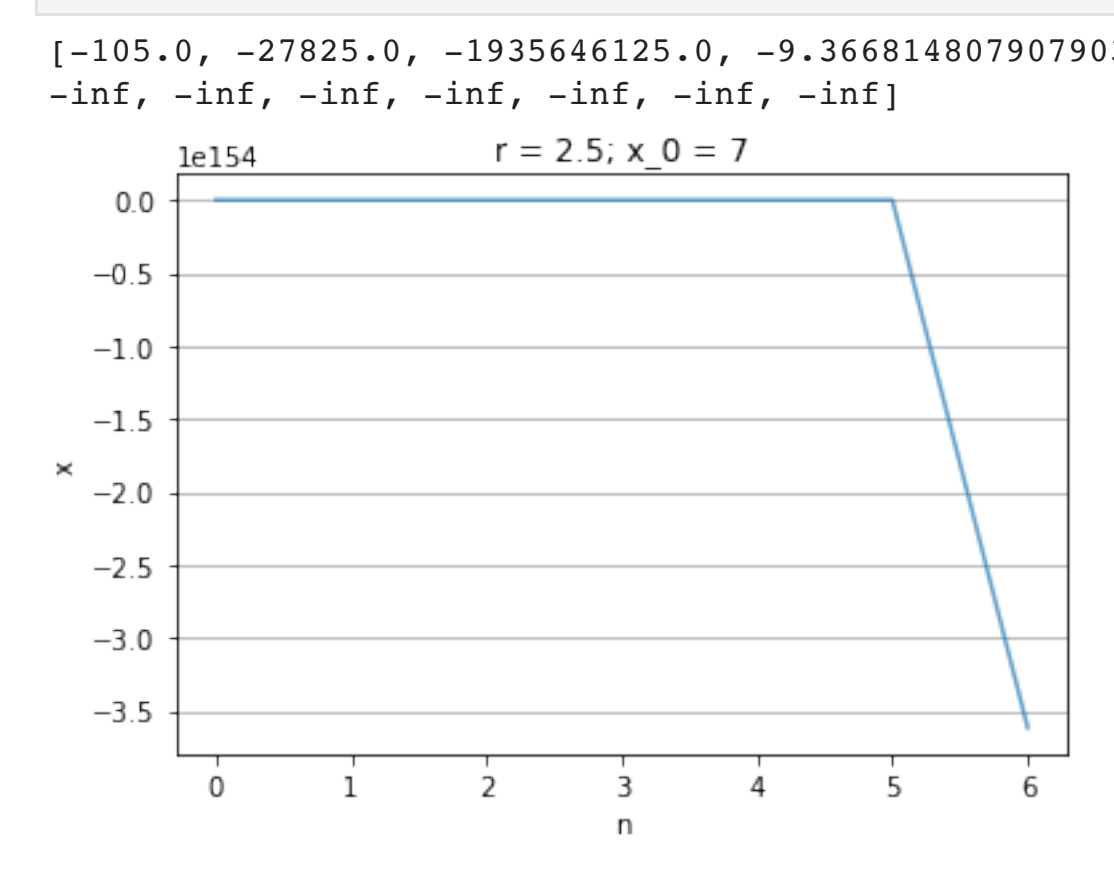
In [2]:
```python
def logistic_function(r,x):

    """
    Definition of the logistic function
    """

    y = r*x*(1-x)


    return y



def logistic_map(r, x_0, n_iter, a=0.7):

    """
    Recording and plotting of x and y values of the logistic map
    """

    x_1 = x_0 #saving x_0 value

    # Initializing x and y lists
    x=[]
    y=[]

    # Recording x and y values for each iteration
    for i in range(n_iter) :
        x_0 = logistic_function(r,x_0)
        y.append(x_0)
        x.append(i)

    # Plotting x and y values
    plt.plot(x,y, alpha = a)
    plt.xlabel("n")
    plt.ylabel("x")
    plt.grid(visible=True, axis="y")
    plt.title("r = "+str(r)+"; x_0 = "+str(x_1))


    return x,y
```

QUESTION 1: What happens if you run the logistic map starting from a value that does not belong to its domain (that is, if x0 > 1 or x0 < 0)?.

When running the logistic map starting from a point that does not belong to its domain, the y values drop drastically showing a decrease towards -∞

In [3]:
```python
x,y = logistic_map(2.5, 7, 20)
print(y)
```

```
[-105.0, -27825.0, -1935646125.0, -9.366814807907903e+18, -2.1934304911410694e+38, -1.202784329866838e+77, -3.6167253604330473e+154, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf]
```

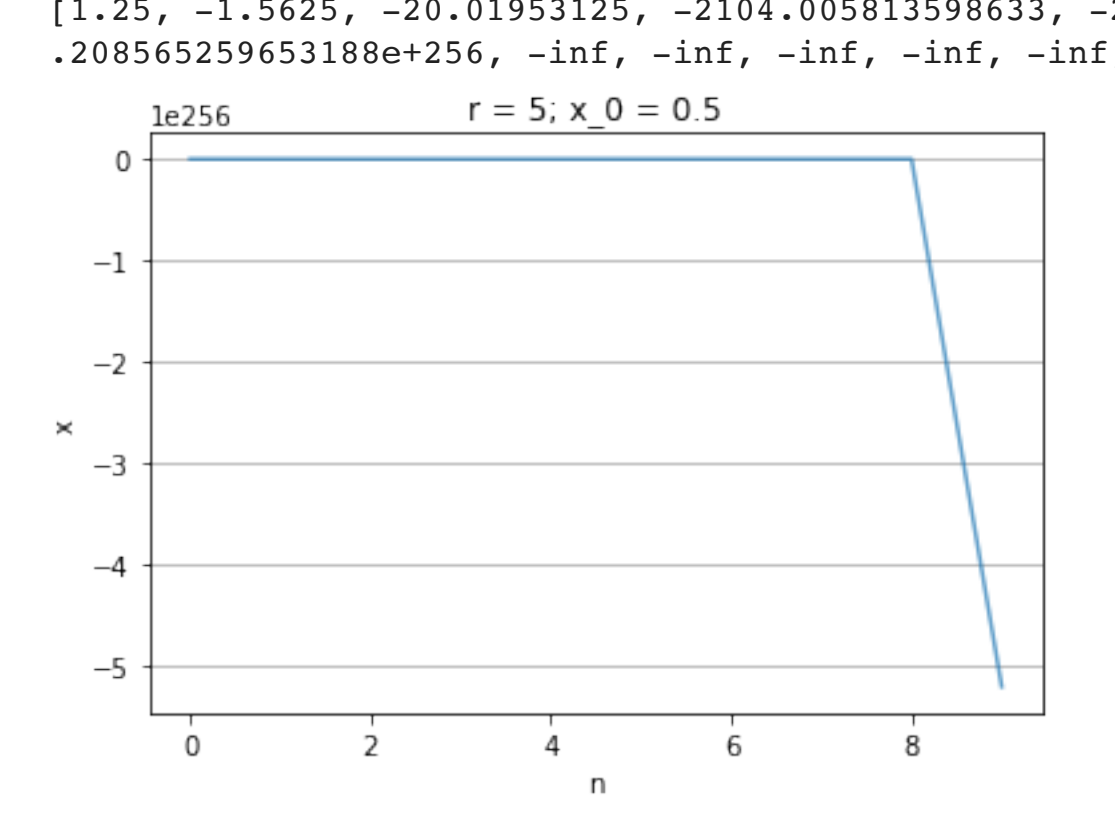


QUESTION 2: What happens if you set r > 4? Does the discrete logistic equation with r > 4 fulfil the properties of a discrete map with domain [0,1]?

When setting r > 4, the logistic map shows a similar behaviour to what happens when running the function starting from a point outside the domain [0,1]. In other words, the values decrease drastically towards -∞

In [4]:
```python
x2,y2 = logistic_map(5, 0.5, 20)
print(y2)
```

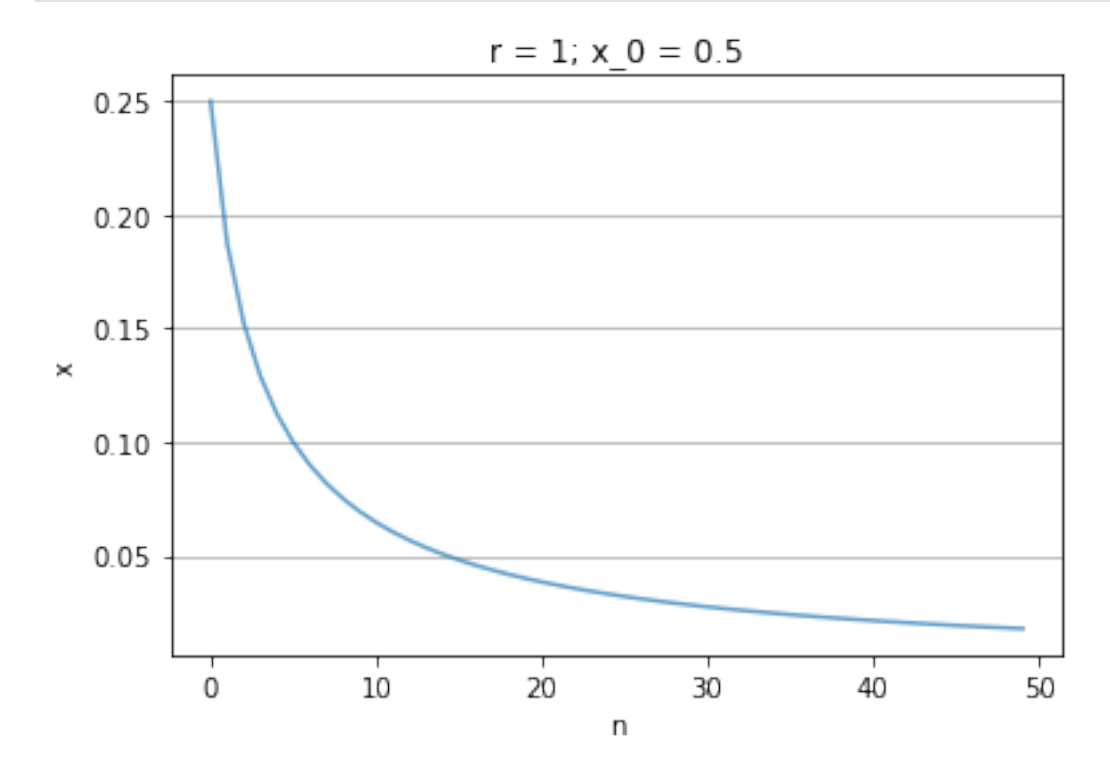
```
[1.25, -1.5625, -20.01953125, -2104.005813598633, -22144722.347352218, -2451943749930215.5, -3.006014076410925e+31, -4.518060313790313e+63, -1.0206434499523511e+128, -5.208565259653188e+256, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf, -inf]
```

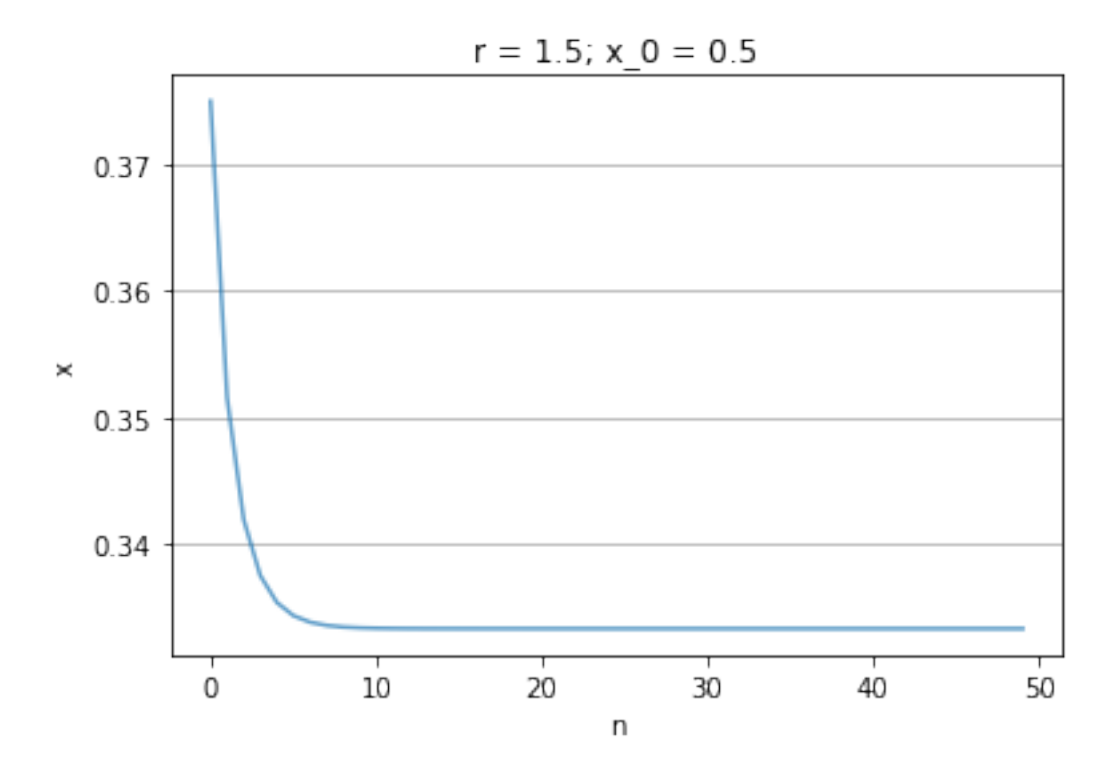


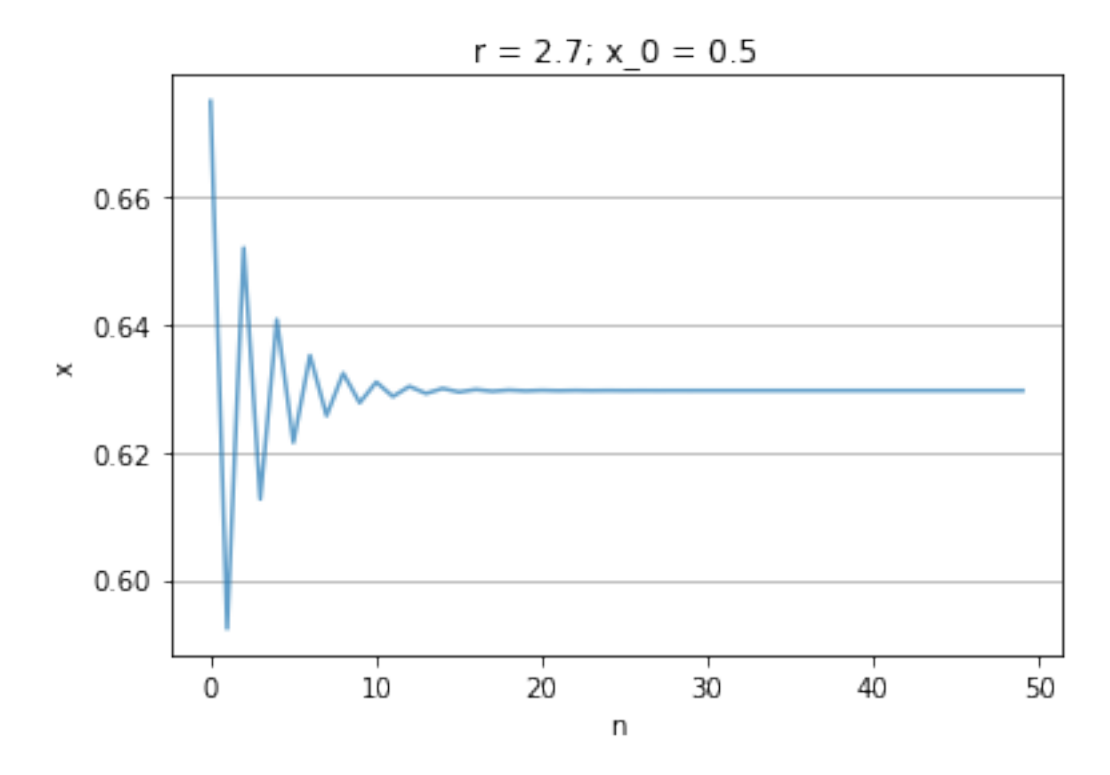
TASK 3: Describe with words the long-term qualitative behavior of the logistic map for the values of r that you explored in the previous exercise.

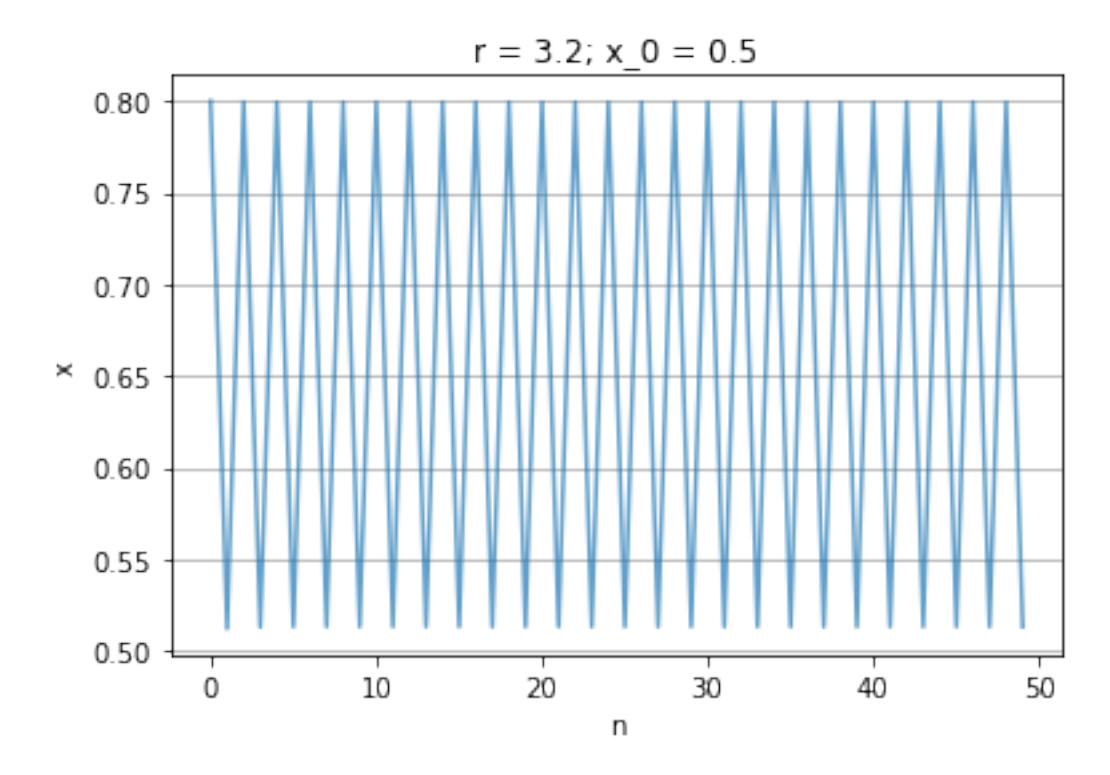
- r = 1, the system decreases towards 0 where it remains stable
- r = 1.5, the system decreases towards a value different than 0, in this case 0.33
- r = 2.7, the system oscilates until it arrives to an equilibrium point, in this case 0.63
- r = 3.2, the system arrives to a period-2-cycle, alternating values between 0.51 and 0.80
- r = 3.5, the system arrives to a period-4-cycle, alternating values between 0.38, 0.87, 0.50 and 0.83
- r = 3.8, the system arrives to an aperiodic / chaotic situation
- r = 5, the system decreases until it arrives to -∞

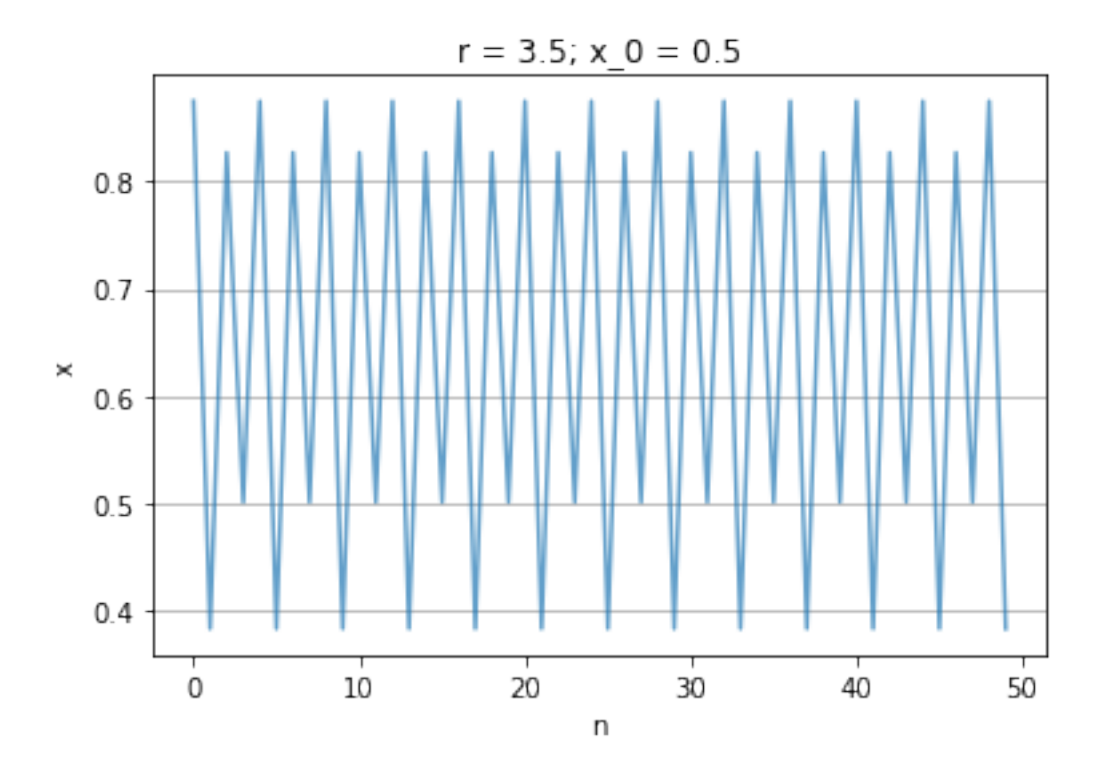
In [5]:
```python
for r in [1, 1.5, 2.7, 3.2, 3.5, 3.8, 5]:
    fig = plt.figure()
    x, y = logistic_map(r, 0.5, 50)
```

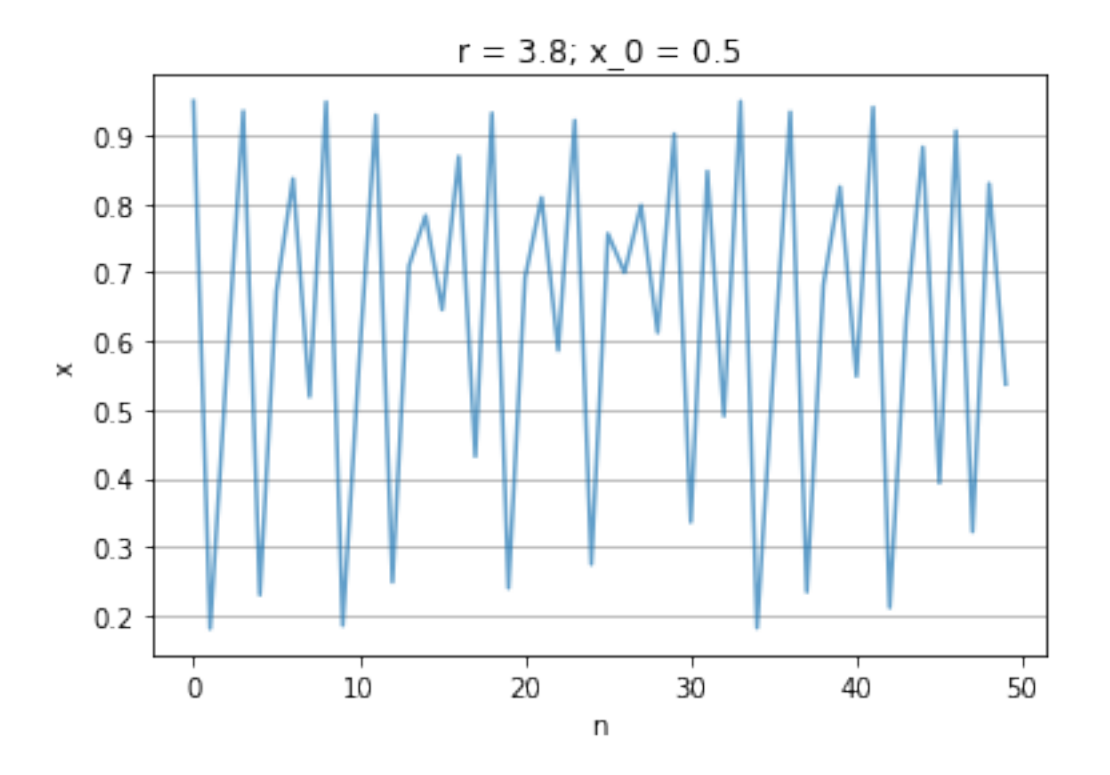


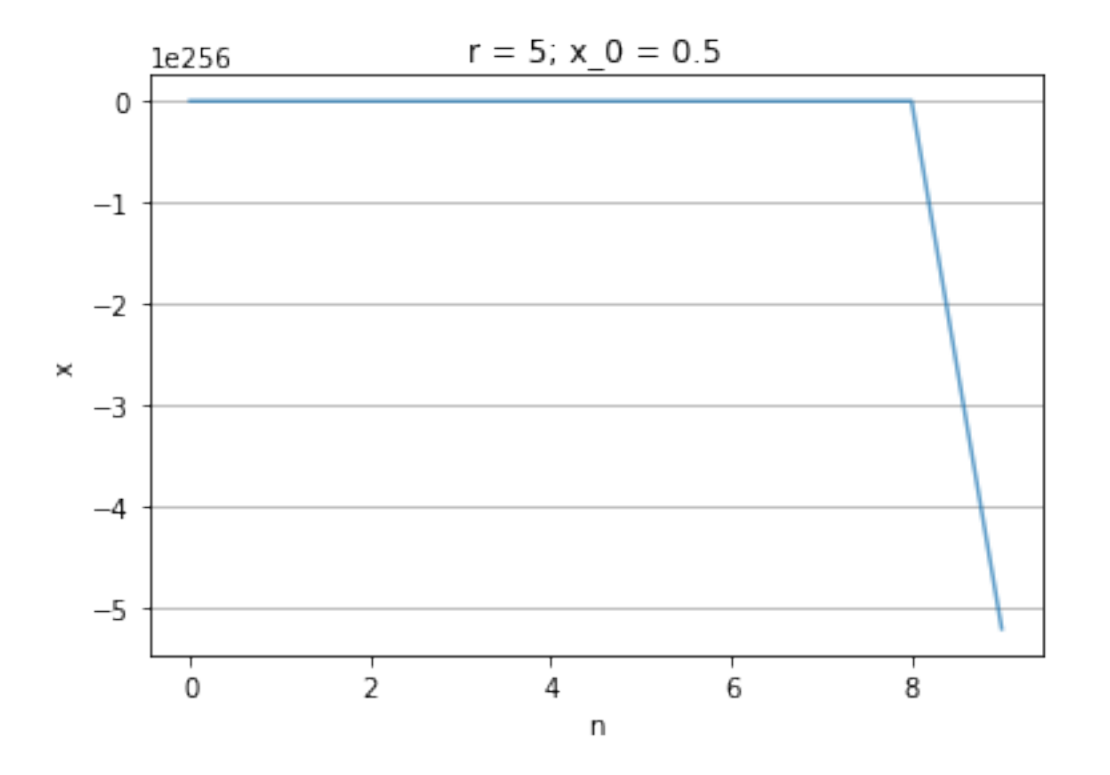












TASK 4: Systematize the characterization of long-term behaviors and design a function that automatically identifies the long-term regime associated to a given value of r. The function should do the following:

1. Given a pair of values of r and x0 (inputs), iterate the logistic function n = 1000 times (there is no need to store the values of these iterations). If this takes too long, try with n = 500.
2. Record values from n = 1000 to n = 1100
3. Process the values obtained in step (2), returning a qualitative description of the long-term behavior. To implement this step, you may need to think about how to describe the long-term behavior in a systematic way. I suggest that you classify the behavior of the system based on the period of its oscillations (period 1: fixed point; period 2: the system oscillates between 2 values; period 3: the system oscillates between 3 values; and so on). With this classification, the output of the function will be a number. Optionally, the function can return 0 if the system collapses to x = 0.

Hint: there are several ways to calculate the period in a relatively simple way. For example, you can look for values that repeat over time. Or you can count how many unique values are observed througout the time series. In any case, allow for some (small) numerical inaccuracy when comparing values (for example, diff < 0.001).

In [6]:
```python
def first_iterations(r, x_0, n_iter=1000):

    """
    Iteration of the logistic function n_iter times (by defect n_iter=1000)
    """

    # Running the first iteractions n_iter of the logistic map
    for _ in range(n_iter):
        x_0 = logistic_function(r, x_0)


    return x_0



def periodic_behaviour(r, x_0, n_last=100, n_iter=1000):

    """
    Calculation of periodic cycle depending on r value taking the last n_last iterations
    """

    # getting x_0 as the last y value of the first n_iter iterations
    x_0 = first_iterations(r, x_0, n_iter)

    # initializing dif list and period variable
    dif = []
    period = 0

    # running n_last iterations and  calculating periodic cycle counting unique values
    for _ in range(n_last):

        x_0 = logistic_function(r, x_0)

        a = round(x_0, 3) #error +/- 0.001

        if a == 0:
            period = 0

            return period

        elif a not in dif:
            dif.append(a)
            period += 1

            # under this condition, period is automatically the max period
            if period > 98 :
                period = n_last

                return period

        else:

            # making sure the periodic prediction is accurate
            if a == dif[0]:

                return period

            else:
                period +=1
                continue
```

In [7]:
```python
period = periodic_behaviour(3.5, 0.5)
print(period)
```

```
4
```

TASK 5: Write a script that loops over a range of values of r (from r = 0 to r = 4 in intervals of 0.01), calls the function from task 4, and records, for each r, the long-term period of the system (let's call it P). You can start from any value of x0 such that 0 < x0 < 1.

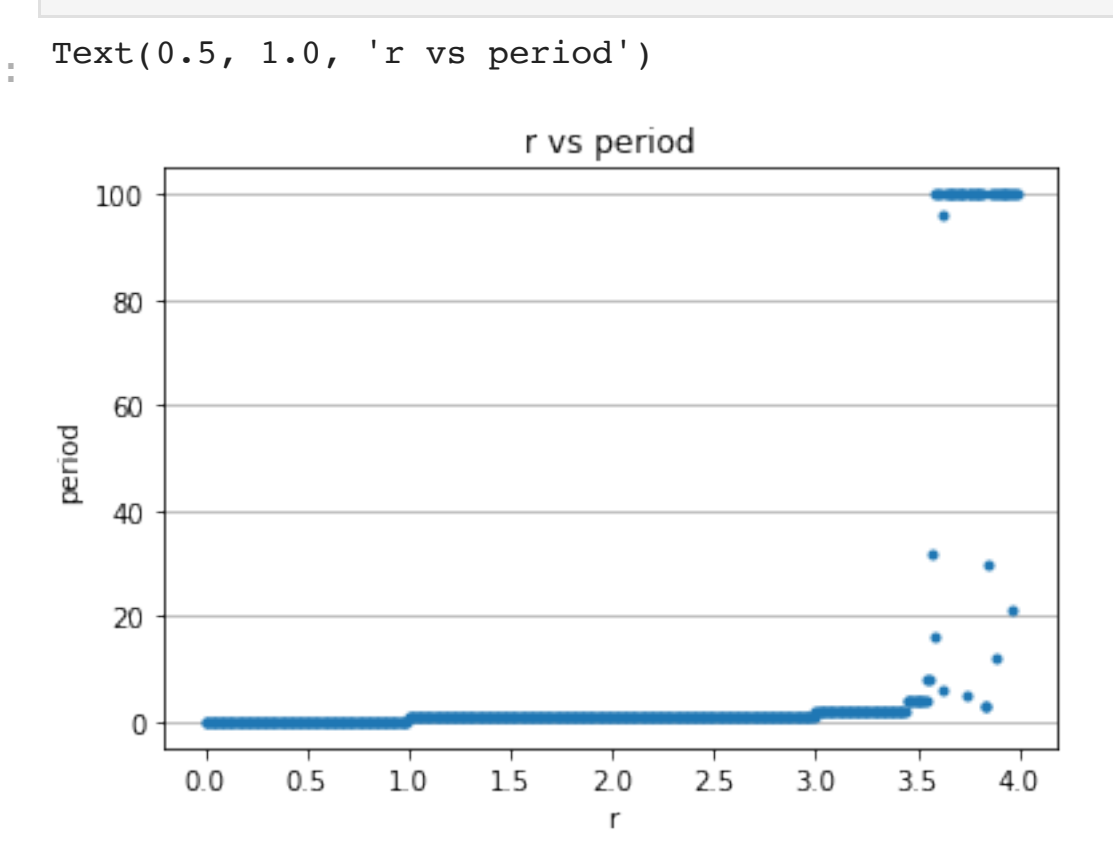
In [8]:
```python
def all_period(max_r, x_0 = 0.5):

    """
    Recording of period value for each r from 0 to a r_max in intervals of 0.01
    """

    # initializing r_list and period_list
    r_list = []
    period_list = []


    for r in np.arange(0.0, max_r, 0.01):

        r = round(r, 2)
        period = periodic_behaviour(r, x_0)

        # appending r values and periods to its lists
        r_list.append(r)
        period_list.append(period)


        #print("r = "+str(r)+"; period = "+str(period))


    return r_list, period_list
```

In [9]:
```python
r_list, period_list = all_period(4, 0.5)
```

Plot r vs P and discuss your results.

In [10]:
```python
"""
Plotting of each r vs its calculated periodic cycle
"""

plt.plot(r_list, period_list, ".")
plt.xlabel("r")
plt.ylabel("period")
plt.grid(visible=True, axis="y")
plt.title("r vs period")
```

Out[10]:
```
Text(0.5, 1.0, 'r vs period')
```



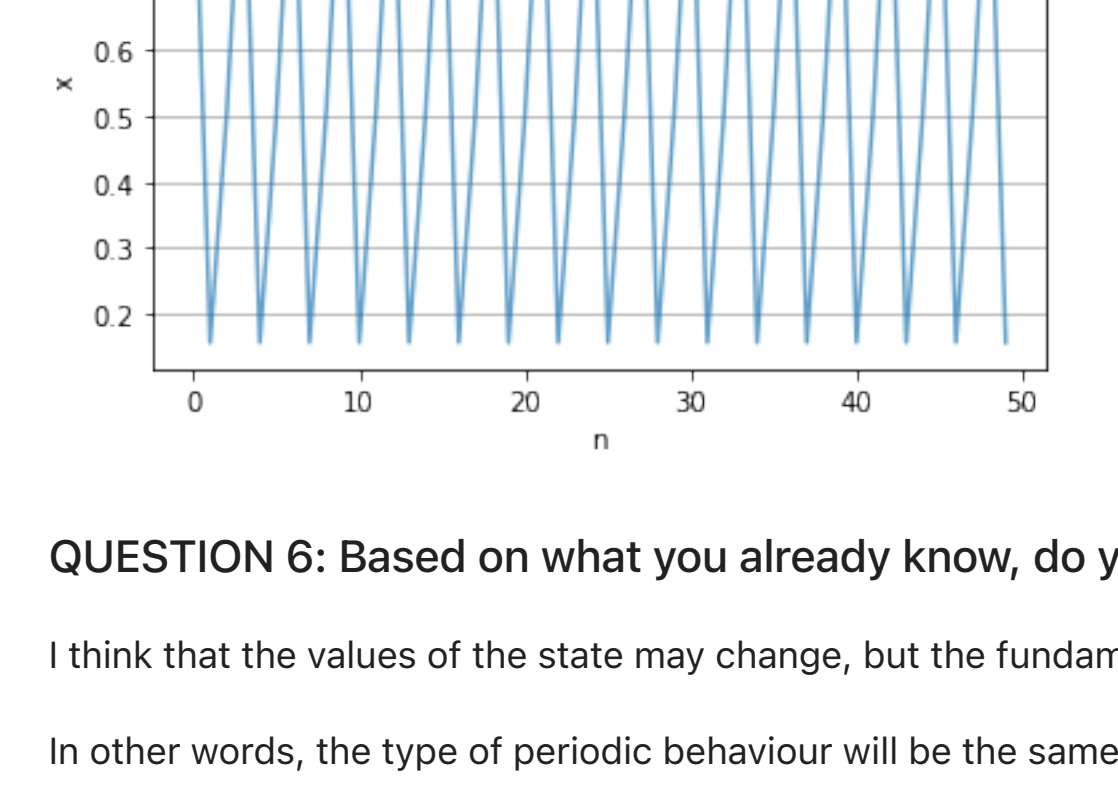
There are some critical r values where the system changes its periodic behaviour. The intervals of each period behaviour are:

- **r < 1** : period = 0
- **1 <= r < 3** : period = 1
- **3 <= r < 3.44** : period = 2
- **3.44 <= r < 3.55** : period = 4
- **3.55 <= r <= 3.56** : period = 8
- (Period = 16 should be in between)
- **r = 3.57** : period = 32
- **r >= 3.58** : period = max_period, in this case we set this value to 100 which is the max number of iterations, but this max_period is equivalent to ∞, and it's called period of infinity. This means that the system is aperiodic and shows a chaotic behaviour

However, when **r = 3.83** and **r = 3.84**, a period-3-cycle is observed. Li and Yorke, describe the appearence of this period-3 cycle as the true beggining of chaotic behaviour. They said: "*Period Three Implies Chaos*"

In 1976, May explained that until r = 3.6786, cycles show very long even period (2^n, $n \to \infty$) that for the human eye seem like very noisy cycles of period 2, they seem aperiodic. It is at r = 3.6786 when the first odd period appears. The odd periods go from longer to shorter, until at r = 3.8284 the cycle of period 3 appears. From this point and until r = 4, the true chaotic behaviour appears.

In [11]:
```python
x,y = logistic_map(3.83, 0.5, 50)
```



QUESTION 6: Based on what you already know, do you think that the results would change if you use a different x0?

I think that the values of the state may change, but the fundamental behaviour will remain the same.

In other words, the type of periodic behaviour will be the same but the fluctuating values may change when using a different x0.

For aperiodic behaviours, the chaotic regimes will be very different when changing x0