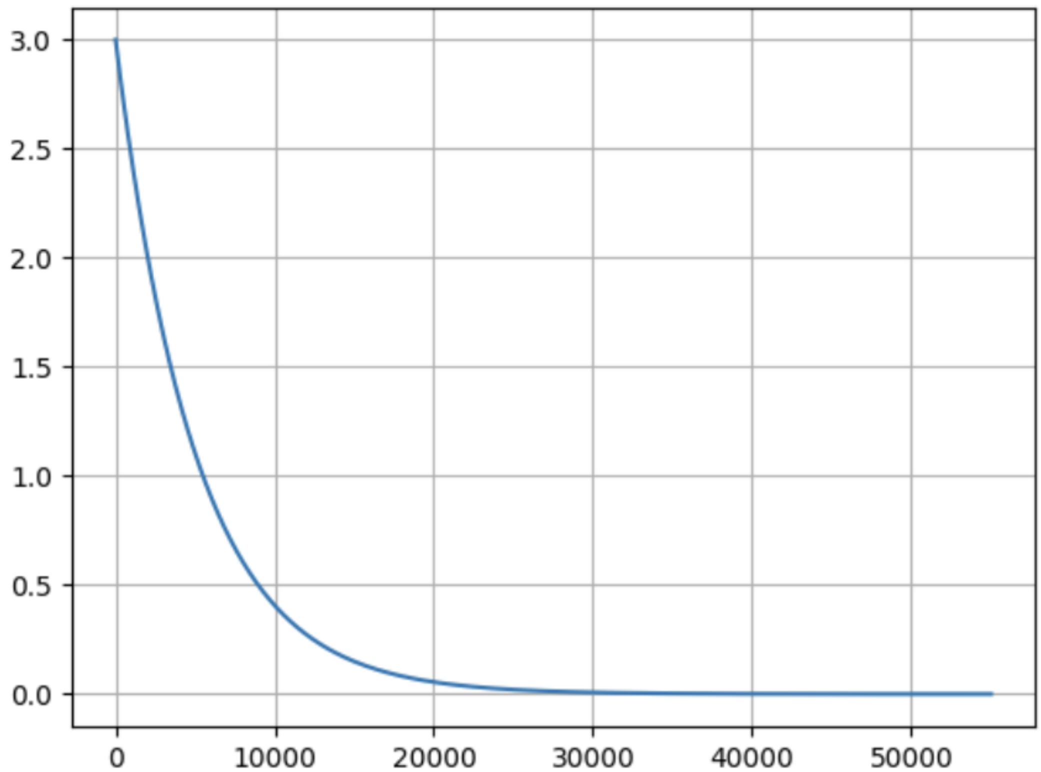
# Assignment 5 - Python

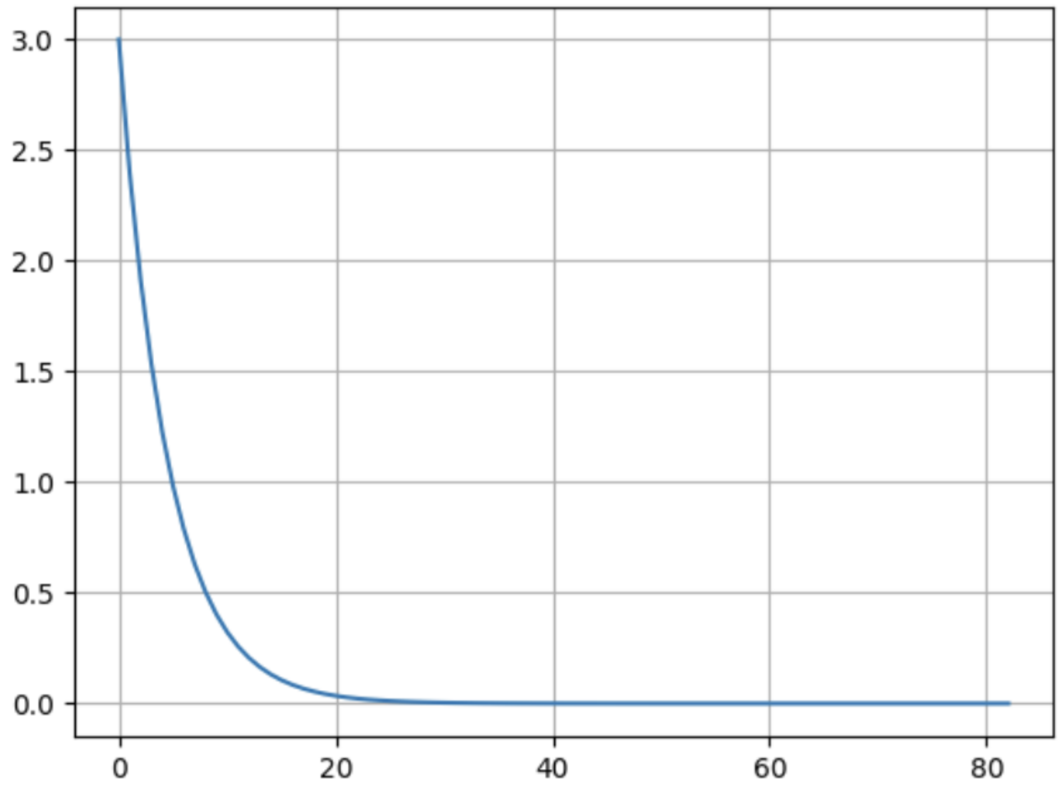
The following code represents my solution for a) b)

```
"""
Let f:R→R be differentiable. To minimize f, consider the gradient descent method
 x_n+1=x_n−eta f'(x_n),
where x_1∈R and eta >0 (learning rate).
(a) Take a convex f and show that for small eta the method converges to the minimum of f.
(b) Show that by increasing eta the method can converge faster (in fewer steps).


Let f:R->R , f(x) = x^2 be the convex function
"""
import math

import numpy as np
import matplotlib.pyplot as plt

def f1_prime(x):
    return 2*x  # 2x

def find_new_element(x, learning_rate):
    return x - learning_rate * f1_prime(x)

series = []
max_number_of_iterations = 10000
precision = 0.000001
x_1 = 3
series.append(x_1)
learning_rate = 0.0001
previous_precision = 1
iteration_counter = 0
while previous_precision > precision and iteration_counter < max_number_of_iterations:
    iteration_counter += 1
    previous_precision = abs(x_1 - find_new_element(x_1, learning_rate))
    x_1 = find_new_element(x_1, learning_rate)
    series.append(x_1)

print("The local minimum occurs at", x_1)

plt.plot(series)
plt.grid()
plt.show()
```

- f(x^2) is the convex function ⇒ the minimum of the function is 0. When we take a small leaning rate (for instance leaning_rate = `0.0001`) the method will converge to the minimum of the function, which is 0. This can be observed in the following graph:




- By increasing the leaning rate (leaning_rate = `0.1`) the method will converge faster. In this example, il only takes around 80 iterations to reach the precision `0.00000001`

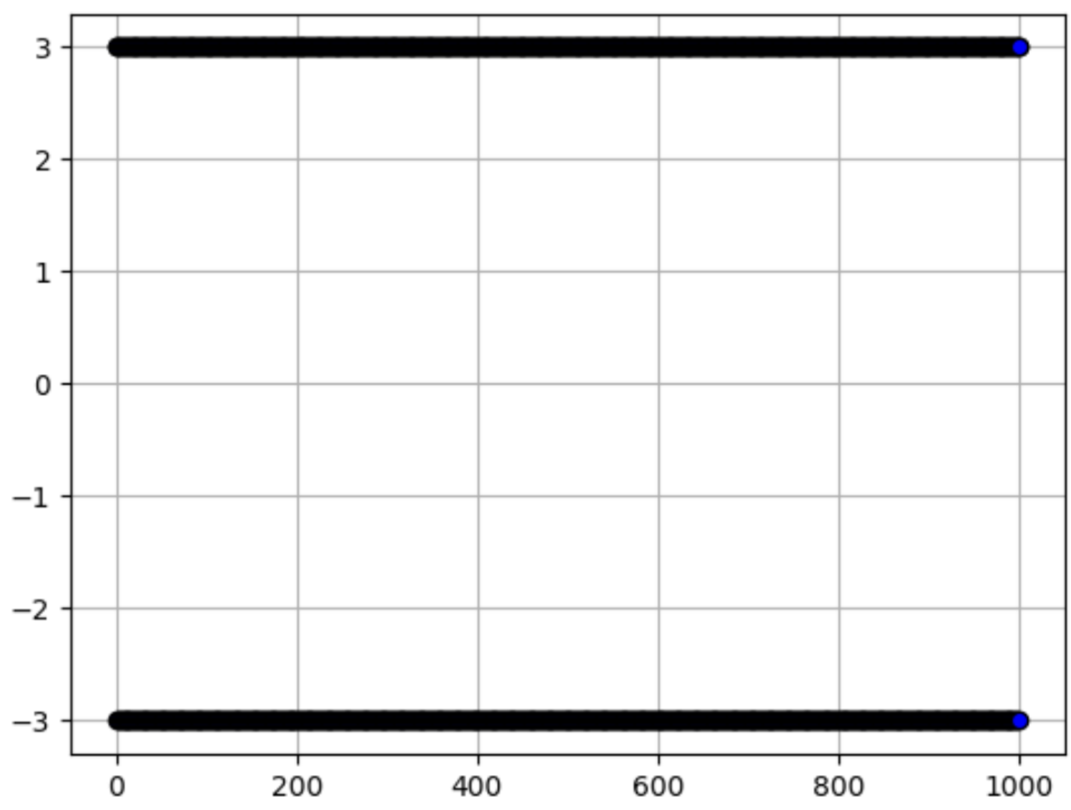



To solve c) “Show that taking eta too large might lead to the divergence of the method”, I wrote another ploting algorithm that can print the graph in a more pleasant way

```
"""
Let f:R→R be differentiable. To minimize f, consider the gradient descent method x_n+1=x_n−eta f'(x_n),
where x_1∈R and eta >0 (learning rate).
(c) Show that taking eta too large might lead to the divergence of the method.

"""
import matplotlib.pyplot as plt
def f1_prime(x):
    return 2*x  # 2x

def find_new_element(x, learning_rate):
    return x - learning_rate * f1_prime(x)

x=[]
y=[]

max_number_of_iterations = 1000
precision = 0.00000001
x_1 = 3
x.append(x_1)
```

```
learning_rate = 1
previous_precision = 1
iteration_counter = 0
y.append(iteration_counter)
while previous_precision > precision and iteration_counter < max_number_of_iterations:
    iteration_counter += 1
    previous_precision = abs(x_1 - find_new_element(x_1, learning_rate))
    x_1 = find_new_element(x_1, learning_rate)
    x.append(x_1)
    y.append(iteration_counter)
plt.plot(y,x, color='none', linestyle='dashed', linewidth = 3,marker='o', markerfacecolor='blue')
plt.grid()
plt.show()
```

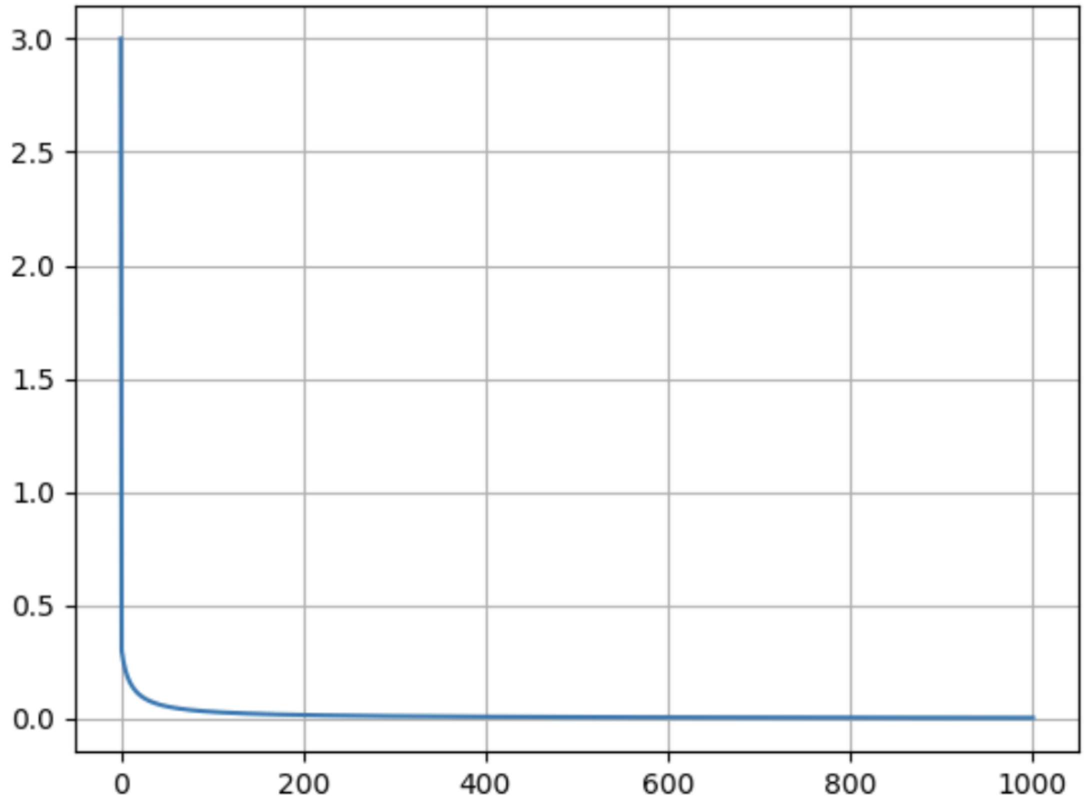
- For the learning_rate = 1, the graph is:




it is obvious that this method diverges

The following code plots the graph for a nonconvex function f=x^3

```
"""
Let f:R→R be differentiable. To minimize f, consider the gradient descent method x_n+1=x_n−eta f'(x_n),
where x_1∈R and eta >0 (learning rate).
(d) Take a nonconvex f and show that the method can get stuck in a local minimum.


Let f:R->R , f(x) = x^3
"""
import math
import numpy as np
import matplotlib.pyplot as plt

def f2_nonconvex_prime(x):
    return 3*x**2   # 3x^2
def find_new_element(x, learning_rate):
    return x - learning_rate * f2_nonconvex_prime(x)

series = []
series.append(3)
max_number_of_iterations = 1000
precision = 0.00000001
x_1 = 3
learning_rate = 0.1
previous_precision = 1
iteration_counter = 0
while previous_precision > precision and iteration_counter < max_number_of_iterations:
    iteration_counter += 1
    previous_precision = abs(x_1 - find_new_element(x_1, learning_rate))
    x_1 = find_new_element(x_1, learning_rate)
    series.append(x_1)
    print(x_1)

print("The local minimum occurs at", x_1)
plt.plot(series)
plt.grid()
plt.show()
```




- We observe that for a learning_rate = 0.1 and a maximum of 1000 iterations, the method gets stuck at a local minimum equal to 0.0032851306356031645