`4a.` จงเขียนฟังก์ชัน cube_root(n) ที่รับพารามิเตอร์เป็นจำนวนจริง n และคืนค่ากลับเป็น**รากที่สาม**ของ n โดย**ใช้วิธีนิวตัน ห้าม**คำนวณโดยใช้ฟังก์ชัน `math.pow(n,1/3)` หรือ ใช้ `n**(1/3)` . กำหนดให้ ความคลาดเคลื่อน (error) ต้องต่ำกว่า `+/- 0.0001`

โดย**เขียนโค้ดลงในช่องว่างที่กำหนด**ระหว่าง `### BEGIN SOLUTION` และ `### END SOLUTION` และหลังจากเขียนโค้ดเสร็จ ให้**ทำการรันเซลล์ถัดไปเพื่อแคปหน้าจอส่ง** (ถ้าได้ผลลัพธ์ `ผ่าน` ถือว่าผ่าน) https://forms.office.com/x

---

In [79]:
```
##### เขียนคำตอบลงในเซลล์นี้ระหว่าง ### BEGIN SOLUTION และ ### END SOLUTION เท่านั้น
def cube_root(n):
    ### BEGIN SOLUTION
    ans = n + 100
    i = 0
    while(ans != n and i < 100):
        ans = ans - ((ans**3)-n)/(3*ans**2)
        i += 1
    return ans
    ### END SOLUTION

# ลองทดสอบฟังก์ชัน
print(cube_root(-27)) # ผลลัพธ์ที่คาดหวังใกล้เคียง 3
```

```
-3.0
```

In [80]:
```
# ห้ามแก้ไขเซลล์นี้โดดเด็ดขาด (ไม่ตรวจ if แก้ไข)
# หลังจากเขียนโค้ดเสร็จ ให้กดรันเซลล์นี้ ถ้าแสดงผลลัพธ์ ```ผ่าน``` ถือว่าผ่าน (เกณฑ์ระดับ 2)
import numpy as np
error = 0.0001
print("ผลการตรวจสอบคำตอบ:")
assert type(cube_root(27)) is float, f"Should return cube root of n"
assert abs(cube_root(27) - np.cbrt(27)) < error, f"Should be {np.cbrt(27)}"
assert abs(cube_root(8) - np.cbrt(8)) < error, f"Should be {np.cbrt(8)}"
assert abs(cube_root(125) - np.cbrt(125)) < error, f"Should be {np.cbrt(125)}"
assert abs(cube_root(1000) - np.cbrt(1000)) < error, f"Should be {np.cbrt(1000)}
assert abs(cube_root(0.1) - np.cbrt(0.1)) < error, f"Should be {np.cbrt(0.1)}"
assert abs(cube_root(1000000) - np.cbrt(1000000)) < error, f"Should be {np.cbrt(
assert abs(cube_root(0) - np.cbrt(0)) < error, f"Should be {np.cbrt(0)}"
assert abs(cube_root(-99) - np.cbrt(-99)) < error, f"Should be {np.cbrt(-99)}"
print("ผ่าน")
```

```
ผลการตรวจสอบคำตอบ:
ผ่าน
```
