```
#exercise 7.1
#15.12.9
#chuanlu

import numpy as np

def fixed_point_iter(func, x0, tol = 1e-8):
    count = 0
    while True:
        count += 1
        x1 = func(x0)
        print(x1)
        if abs(x1 - x0) < tol:
            break
        x0 = x1
    return x1, count

def steffensen_accelerated_iter(func, x0, tol = 1e-8):
    count = 0
    while True:
        count += 1
        y = func(x0)
        z = func(y)
        x1 = x0 - ((y - x0) **2) / (z - 2 * y + x0)
        print(x1)
        if abs(x1 - x0) < tol:
            break
        x0 = x1
    return x1, count

def newton_iter(func1, func2, x0, tol = 1e-8):
    count = 0
    f1 = func1(x0)
    f2 = func2(x0)
    while True:
        count += 1
        x1 = x0 - f1 / f2
        print(x1)
        f1 = func1(x1)
        f2 = func2(x1)
        if abs(x1 - x0) < tol:
            break
        x0 = x1
    return x1, count
```

```
def main():
    print("exercise7.1.2")
    #question7.1.1
    x0 = 0.5
    func = lambda x: (x**2 + 2 - np.exp(x))/3
    func1 = lambda x: x ** 2 - 3 * x + 2 - np.exp(x)
    func2 = lambda x: 2 * x - 3 - np.exp(x)
    print("fixed_point_iter")
    result, count = fixed_point_iter(func, x0)
    print("result:", result)
    print("count:", count)
    print("steffensen_accelerated_iter")
    result, count = steffensen_accelerated_iter(func, x0)
    print("result:", result)
    print("count:", count)
    print("newton_iter")
    result, count = newton_iter(func1, func2, x0)
    print("result:", result)
    print("count:", count)

    #question7.1.2
    print("exercise7.1.2")
    x0 = 1
    func = lambda x: (28 - 7*x) ** (1/3)
    func1 = lambda x: x**3 + 2*(x**2) + 10*x - 20
    func2 = lambda x: 3*(x**2) + 4*x + 10
    print("fixed_point_iter")
    result, count = fixed_point_iter(func, x0)
    print("result:", result)
    print("count:", count)
    print("steffensen_accelerated_iter")
    result, count = steffensen_accelerated_iter(func, x0)
    print("result:", result)
    print("count:", count)
    print("newton_iter")
    result, count = newton_iter(func1, func2, x0)
    print("result:", result)
    print("count:", count)



if __name__ == '__main__':
    main()
```

第一小题中，构造出来的不动点迭代函数为(x**2 + 2 - np.exp(x))/3

运行结果如下：

```
exercise7.1.1
fixed_point_iter
0.2004262431
0.272749065098
0.253607156584
0.258550376265
0.257265636335
0.257598985162
0.257512454515
0.257534913615
0.257529084168
0.257530597238
0.25753020451
0.257530306446
0.257530279988
0.257530286855
result: 0.257530286855
count: 14
steffensen_accelerated_iter
0.258684427566
0.25753031772
0.25753028544
0.25753028544
result: 0.25753028544
count: 4
newton_iter
0.253688702418
0.257528900795
0.25753028544
0.25753028544
result: 0.25753028544
count: 4
```

结果分析：newton 迭代法和斯特芬森迭代法的收敛速度是直接迭代的平方。

第二小题：

令 x+1 = y，构造出来的迭代函数为(28-7*x)**(1/3)

运算结果如下：

```
exercise7.1.2
fixed_point_iter
2.7589241763811203
2.0557270563211336
2.387546053923847
```

```
2.243167925600752
2.308214140457189
2.279368157276044
2.2922500909269226
2.2865152571745133
2.289071863777502
2.2879328264438263
2.288440438413343
2.2882142489662534
2.2883150434170063
2.2882701285310865
2.288290143213584
2.288281224442784
2.288285198757364
2.2882834277553363
2.288284216935323
2.288283865267215
2.288284021974773
2.2882839521439946
2.2882839832614326
2.2882839693951262
2.28828397557412
result: 2.28828397557412
count: 25
steffensen_accelerated_iter
2.2565645172437203
2.288257348336069
2.2882839736504734
2.288283973669436
result: 2.288283973669436
count: 4
newton_iter
1.4117647058823528
1.3693364705882352
1.3688081886175318
1.3688081078213745
1.3688081078213727
result: 1.3688081078213727
count: 5
[Finished in 0.3s]
```

当然，在结果中，需要将前两种迭代法的值-1

分析：

和第一小题有着相同的结论