Homework for Lecture 6

1. Find the output of these codes without running them (in other words: what would get printed on screen if you were to run these codes?). If there is a bug, explain what the problem is.

```
x = 1

def f():
    x=2
    print(x)

def g():
    print(x)
    f()

g()
print(x)
```

```
x = 1
y = 10

def f():
    x=2
    print(x)
    print(y)

def g():
    print(x)
    f()

print(x)
g()
```

```
y = 10

def f():
    x=2
    print(x)
    print(y)

def g():
    x = 3
    f()
    print(x)

print(x)
g()
```

2. The trapezoidal rule is often the go to way to approximate an integral:

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}\Big(f(x_0) + 2f(x_1) + 2f(x_2) + \ldots + 2f(x_{N-2}) + 2f(x_{N-1}) + f(x_N)\Big)$$

where $\Delta x = (b-a)/N$ and $x_i = i\Delta x$. Complete the code below, and then use it to compute the following 3 integrals:

- $\int_0^1 x dx$
- $\int_0^1 x^2 dx$
- $\int_0^1 e^{-x^2} dx$

(Make sure to double check your code by computing some of these integrals with pen and paper.)

```
def trapezoidal_rule(f,a,b,N):
    """
    Approximate the integral of f(x) on the interval [a,b]
    using the trapezoidal rule with step size dx = (b-a)/N:

    INPUT: f (function): function to be integrate
           a,b (floats): left and right boundaries of the interval [a,b]
           N (int): number of cells in which [a,b] is divided

    OUTPUT: integral (float): approximation of the integral of f(x) on [a,b]

    """
```

3. No computer needed for this question. Consider the function

$$f(x) = \begin{cases} x+1 & \text{if } x > 0 \\ x-1 & \text{if } x < 0 \end{cases}$$

   (a) What would be the most reasonable solution of $f(x) = 0$?

   (b) Which algorithm converges toward this solution? Newton's method or the Bisection method? Explain.

   (c) What would be the successive iterates generated by Newton's method?

4. Write a function called approx_derivative that takes as input a function f, a float x, and a float h, and returns

$$\frac{f(x+h) - f(x)}{h}$$

.

5. When using Newton's method, one need to provide a formula for both the function AND its derivative, which can be a pain for lazy people. Write a variant of Newton's method that only requires a formula for the function $f(x)$. The derivative $f'(x)$ will be approximated using the function you wrote in the previous problem. Complete the code below. Then use both Newton's method and Lazy Newton's method to solve $3x - e^x = 0$ with initial guess $x_{init} = 2$. Do they both find the same solution?

```
def approx_derivative(f,x,h):
    """
    Approximate the derivative of f at x using the formula
    f'(x) ~ (f(x+h)-f(x) )/h
    """
    # COMPLETE


def newton(f, df, x_init, maxiter):
    """
    Newton's method to solve f(x)= 0 with initial guess x_init

    INPUTS
    f (function):  function for which we want to solve f(x)=0
    df (function): derivative of f
    x_init (floats): initial guess
    maxiter (int): number of iterations

    OUTPUTS
    x (float): approximate solution of f(x)=0
    """
    # COMPLETE


def lazy_newton(f, x_init, maxiter):
    """
    Modified Newton's method to solve f(x)= 0 with initial guess x_init
    The derivative of f(x) is approximated using the function approx_derivative

    INPUTS
    f (function):  function for which we want to solve f(x)=0
    x_init (floats): initial guess
    maxiter (int): number of iterations

    OUTPUTS
    x (float): approximate solution of f(x)=0
    """
    # COMPLETE
```

6. A better way to write a Newton solver that do not require the derivative is to use the secant method that we discussed in class. Implement the sequent method and use it to solve $3x - e^x = 0$ with initial iterate $x_{init1} = 2$ and $x_{init1} = 1.9$.

   Remark: Try with max_iter = 5 then max_iter = 10. What's happening? Try to understand what is the problem.

```
def secant(f, x_init_1, x_init_2, maxiter):
    """
    Secant method to solve f(x)= 0 with initial iterates x_init_1 and x_init_2.

    INPUTS
    f (function): function for which we want to solve f(x)=0
    x_init_1, x_init_2 (floats):first two iterates of the sequence
    maxiter (int): number of iterations

    OUTPUTS
    x (float): approximate solution of f(x)=0
    """
    # COMPLETE
```

7. Run your Newton solver (with initial guess $x = 2$) and your bisection solver (with initial interval $[-1, 5]$) on the function

$$f(x) = \begin{cases} x+1 & \text{if } x > 0 \\ x-1 & \text{if } x < 0 \end{cases}$$

   from problem 3. Check that the codes behave as expected (to do this print each successive iterate)

8. The equation $3x - e^x = 0$ has 2 solutions:

$$x_1 = 0.61906 \qquad \text{and} \qquad x_2 = 1.51213$$

   (a) How many steps are needed to arrive within 10^{-3} of the solution if you use Newton's method with initial guess $x = 0$.

   (b) How many steps are needed to arrive within 10^{-3} of the solution if you use Newton's method with initial guess $x = 2$.

   (c) How many steps are needed to arrive within 10^{-3} of the solution if you use the secant method with initial iterates $x = 2$ and $x = 1.9$.

   (d) How many steps are needed to arrive within 10^{-3} of the solution if you use the bisection method with initial interval [1,2].

   To clarify: for all of the questions above you need to find the smallest max_iter value that lead to an approximate solution which is within 10^{-3} of the exact solution.

   Which of the three algorithm converges the fastest to the solution?

   Hint: in your code, you should add a line that print the distance between the current iterate and the true solution. For example, in my Newton code I have added:

```
print('iteration', i, '\t |x-x1| =', abs(x-x1), '\t |x-x2| =', abs(x-x2) )
```

   The function abs() is a built in python function that compute the absolute value of a float.

9. (a) Write a function that takes as input an integer, and returns true if this integer contains the digit 7 (and false otherwise). To do this you will need to convert the integer into a string and then loop through this string.

   (b) Compute the sum of all the integers which are less than 1000 and that contains the digit 7.