Lab3

Exercise 1

```
>>> a = 0.3
>>> b = 0.4
>>> c = (a**2 + b**2)**0.5
>>> c
0.5
```

Exercise 2

```
>>> 7 / 4
1.75
>>> 7 // 4
1
>>>
```

Exercise 3

```
>>> a = 3e-200
>>> b = 4e-200
>>> c1 = (a**2 + b**2)**0.5
>>> c2 = a * (1 + (b/a)**2)**0.5
>>> c1
0.0
>>> c2
5e-200
>>>
```

Exercise 4

```
00011110b = 2^1 + 2^0 + 2^-1 + 2^-2 = 3.75
01101111b = 2^3 + 2^2 + 2^0 + 2^-1 + 2^-2 + 2^-3 = 13.875
00010100b = 2^1 + 2^-1 = 2.5
01001010b = 2^3 + 2^0 + 2^-2 = 9.25

0x54 = 01010100b = 2^3 + 2^1 + 2^-1 = 10.5
0x60 = 01100000b = 2^3 + 2^2 = 12
0x3E = 00111110b = 2^2 + 2^1 + 2^0 + 2^-1 + 2^-2 = 7.75
0x0D = 00001101b = 2^0 + 2^-1 + 2^-3 = 1.625
```

Exercise 5

```
01011111b = (-1)^0 * 2^(11-7) * (1 + 2^-1 + 2^-2 + 2^-3) = 30
00100001b = (-1)^0 * 2^(4-7) * (1 + 2^-3) = 0.140625
01111011b = (-1)^0 * 2^(15-7) * (1 + 2^-2 + 2^-3) = 352
01101010b = (-1)^0 * 2^(13-7) * (1 + 2^-2) = 80

0x93 = 10010011b = (-1)^1 * 2^(2-7) * (1 + 2^-2 + 2^-3) = -0.04
0x56 = 01010110b = (-1)^0 * 2^(10-7) * (1 + 2^-1 + 2^-2) = 14
0x8D = 10001101b = (-1)^1 * 2^(1-7) * (1 + 2^-1 + 2^-3) = -0.02
0xE4 = 11100100b = (-1)^1 * 2^(12-7) * (1 + 2^-1) = -48
```

Exercise 6

```
import numpy as np
def factorial_py(n):
    if n == 0:
        return 1
    else:
        return n * factorial_py(n-1)
```

```
def factorial_np(n):
    if n == 0:
        return np.int64(1)
    else:
        return np.int64(n * factorial_np(n-1))

def binomial_py(n,k):
    return factorial_py(n) / (factorial_py(k) * factorial_py(n-

def binomial_np(n,k):
    return factorial_np(n) / (factorial_np(k) * factorial_np(n-
```

Exercise 7

```
def binomial_recursive(n,k):
    if k == 0:
        return 1
    elif k == n:
        return 1
    else:
        return binomial_recursive(n-1,k) + binomial_recursive(n
```

Exercise 8

```
def cosine(alpha, terms):
    acc = 0
    for i in range(terms):
        acc += ((-1)**i) * (alpha**(2*i)) / factorial_py(2*i)
    return acc
```

Exercise 9

```python
def newtons_method(terms):
    x = 10
    for _ in range(terms):
        x = x - (x**2 - 10*x + 25) / (2*x - 10)
    return x
```

Exercise 10

```python
def newtons_method2(x, terms):
    for _ in range(terms):
        x = x - (x**2 - 11*x + 30) / (2*x - 11)
    return x

def newtons_method3(x, terms):
    for _ in range(terms):
        x = x - (x**2 - 10*x + 31) / (2*x - 10)
    return x
```

```
>>> newtons_method2(10, 100)
6.0000000000000036
>>> newtons_method2(-10, 100)
4.999999999999998
>>> newtons_method2(0, 100)
4.999999999999997
>>> newtons_method2(-100, 100)
5.0000000000000036
>>> newtons_method3(10, 100)
8.151226767966968
>>> newtons_method3(0, 100)
-3.15027368285062
```

```
>>> newtons_method3(-1, 100)
2.611785866468352
```

As we can see, my implementation of the Newton’s method can find multiple roots of polynomials, if we use different starting guesses we will get different roots. However for polynomials with no real roots this algorithm behaves chaotically and doesn’t give good answers. I can think of two solutions to this problem, one is to implement some kind of algorithm to detect if we are getting closer and closer to some solution, if not then it will return `no_roots`, or a different solution which is to use complex numbers as a starting guess, and this way we will find all roots of every polynomial, and we can select only the real ones.

Below is my implementation of the second method:

```
>>> newtons_method3(1 + 1j, 100)
(5+2.449489742783178j)
>>> newtons_method3(0 + 1j, 100)
(5+2.449489742783178j)
>>> newtons_method3(0 + -1j, 100)
(5-2.449489742783178j)
>>> newtons_method3(0 + -10j, 100)
(5-2.4494897427831783j)
>>> newtons_method3(0 + 10j, 100)
(5+2.4494897427831783j)
```

As we can see, we found two complex roots of the above polynomial.