Day 30

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [6]:
```
fig0,ax0 = plt.subplots()
x = np.linspace(0,1,100)
ax0.plot(x, np.exp(-x))
ax0.plot(x,x)
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x7f0dfb85a630>]
```

In [8]:
```
x = 1
for i in range(20):
    x = np.exp(-x)
    print(x)
```

```
0.36787944117144233
0.6922006275553464
0.5004735005636368
0.6062435350855974
0.545395785975027
0.5796123355033789
0.5601154613610891
0.571143115080177
0.5648793473910495
0.5684287250290607
0.5664147331468833
0.5675566373282834
0.5669089119214953
0.5672762321755696
0.5670678983907884
0.567186050099357
0.5671190400572149
0.5671570440012975
0.5671354902062784
0.5671477142601192
```

In [11]:
```
x = 100
for i in range(20):
    x = np.exp(-x)
    print(x)
```

```
3.720075976020836e-44
1.0
0.36787944117144233
0.6922006275553464
0.5004735005636368
0.6062435350855974
0.545395785975027
0.5796123355033789
0.5601154613610891
0.571143115080177
0.5648793473910495
0.5684287250290607
0.5664147331468833
0.5675566373282834
0.5669089119214953
0.5672762321755696
0.5670678983907884
0.567186050099357
0.5671190400572149
0.5671570440012975
```

Let's try with a new function: $$x = e^{1-x^2}$$

In [15]:
```
fig2, ax2 = plt.subplots()
x = np.linspace(0,2,100)
ax2.plot(x,np.exp(1-x**2))
```

```
ax2.plot(x,x)
```

Out[15]: [<matplotlib.lines.Line2D at 0x7f0dfb9eb198>]

In [14]:

```
x = 0.5
for i in range(20):
    x = np.exp(1-x**2)
    print(x)
```

```
2.117000016612675
0.030755419069985038
2.715711832754083
0.0017034651847384463
2.71827394057758
0.001679913095081425
2.7182741571849562
0.0016799111168229455
2.7182741572030236
0.0016799111166579386
2.7182741572030253
0.0016799111166579221
2.7182741572030253
0.0016799111166579221
2.7182741572030253
0.0016799111166579221
2.7182741572030253
0.0016799111166579221
2.7182741572030253
0.0016799111166579221
```

Let's try the inverted form of the previous equation: $$x=\sqrt{1-\ln x}$$

In [17]:

```
fig1, ax1 = plt.subplots()
x = np.linspace(0,2,100)
ax1.plot(x, np.sqrt(1-np.log(x)))
ax1.plot(x,x)
```

```
/home/eric/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3: RuntimeWarning: divide by zero encount
ered in log
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[17]: [<matplotlib.lines.Line2D at 0x7f0dfb8a7fd0>]

In [18]:

```
x = 0.5
for i in range(20):
    x = np.sqrt(1-np.log(x))
    print(x)
```

```
1.3012098910475378
0.8583154914892762
1.0736775779454883
0.9637999044091371
1.0182689104343374
0.990906635925747
1.004557096969838
0.997724037576543
1.0011386299421705
0.9994308469350205
1.000284617043603
0.9998577016016549
1.000071151730577
0.9999644247674951
1.0000177877744567
0.9999911061523217
1.0000044469337268
0.9999977765356085
1.000011117328136
0.9999994441337476
```

In [21]:

```
In [21]:
x = 2 # guess
diff = 1 # arbitrary
while abs(diff) > 2**-32:
    x1 = np.sqrt(1-np.log(x))
    diff = x - x1
    x = x1

print(x)
```

```
1.0000000000589966
```

```
In [2]:
def bisection(function, lower_guess, upper_guess, tolerance=2**-32):
    midpoint = (lower_guess + upper_guess)/2
    while upper_guess - lower_guess > tolerance:
        if function(lower_guess)*function(midpoint)<0:
            upper_guess = midpoint
            midpoint = (lower_guess + upper_guess)/2
        elif function(midpoint)*function(upper_guess)<0:
            lower_guess = midpoint
            midpoint = (lower_guess + upper_guess)/2
        elif function(lower_guess)*function(midpoint)>0 and function(midpoint)*function(upper_guess)>0:
            print('no unique root in that bracket')
            break
    return(midpoint)
```

```
In [3]:
def func(x):
    return(x-np.exp(1-x**2))

bisection(func, 0, 1.7)
```

```
Out[3]: 1.0000000000640283
```

```
In [33]:
func(2)
```

```
Out[33]: 1.9502129316321362
```

```
In [ ]:
```