Question 01

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [2]:
```
np.linspace(0, np.pi, num = 30)
```

Out[2]:
```
array([0.        , 0.10833078, 0.21666156, 0.32499234, 0.43332312,
       0.54165391, 0.64998469, 0.75831547, 0.86664625, 0.97497703,
       1.08330781, 1.19163859, 1.29996937, 1.40830016, 1.51663094,
       1.62496172, 1.7332925 , 1.84162328, 1.94995406, 2.05828484,
       2.16661562, 2.2749464 , 2.38327719, 2.49160797, 2.59993875,
       2.70826953, 2.81660031, 2.92493109, 3.03326187, 3.14159265])
```

In [3]:
```
x = (np.linspace(0, np.pi, num = 30))
```

In [4]:
```
## Define the functions for the overall formulas

def f(y):
    return (y-0.9*np.sin(y))

def fp(y):
    return (1-(0.9*np.cos(y)))

## Define the function to solve for X.
## Each input needs its own loop, find out which input is being entered and play with iterations
## No need to add a spot in the function for tolerance. It is already set in class as 10**(-6)

def Prob1(n, Iterations):
    y = 0
    i = 0
    while i <= Iterations:
        x = y - ((f(y)-n)/fp(y))
        if np.absolute(x - y) < 10**(-6):
            print(x)
            break
        i = i + 1
        y = x
    if i > Iterations:
        print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

In [5]:
```
## Run the function to solve for x in each iteration
## Self note, had to raise the number of iterations to something high

x = np.linspace(0, np.pi, num = 30)
for n in x:
    Prob1(n, 100)
```

```
0.0
0.6604701856042979
0.9473407987171497
1.1444131303639302
1.300691844877185
1.433140220934481
1.5497860499774534
1.655117812587781
1.7519234076803416
1.8420654702561892
1.9268572778959292
2.007264078579107
2.0840193067347905
2.1576957200296207
2.2287510506335493
2.297558467587193
2.3644275836436326
2.4296193565068216
2.4933569231885917
2.5558336524672014
2.617219250484121
2.677664476989677
2.7373048536363815
2.7962636311392703
2.854654205967585
2.912582125684288
2.970146786655176
3.027442903313447
3.0845618111153983
3.141592653589793
```

Question 2

Step 1: Find the max time and amount by getting the derivate and solving it for zero. Plus the max time into the regular equal to solve for the max amount.

Graphed the equation to get an idea of when the equation will be around .25 amount. Looks like it happens around 11.

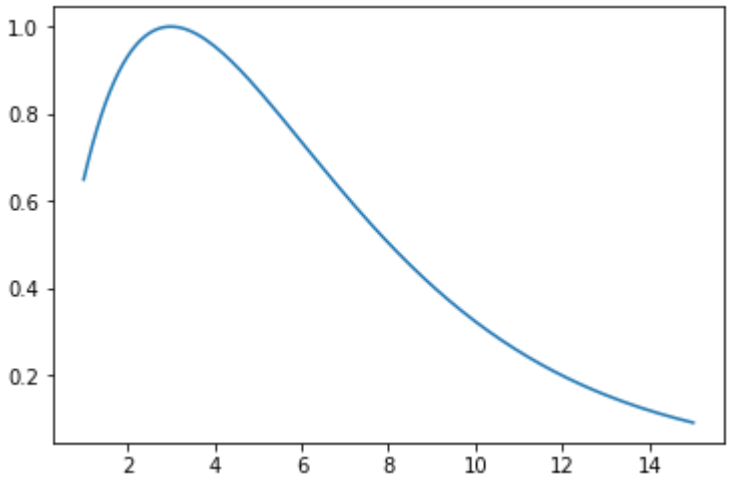
In [6]:
```
def f(x):
    return (math.exp(1)/3) * x * (math.exp(-x/3))

f2 = np.vectorize(f)

x = np.arange(1, 15.1, 0.1)

plt.plot(x, f2(x))

plt.show()
```



Subtracted the max amount from the base equation. The .25 amount should now be at the 'x' axis. Plugged it into the Newton method and solved.

In [7]:
```
def f(t):
    return (math.exp(1)/3) * t * (math.exp(-t/3)) - 0.25

def fp(t):
    return - (1/9) * math.exp(1-(t/3)) * (t-3)


p_0 = 11
TOL = 1 * 10**-6
Iterations = 20

results = []   ## Need to save the results to something
results.append(p_0)
## Step 1 - Set i = 1

i = 1

## Step 2 - While (i <= N_0) do steps 3-6

while i <= Iterations:

## Step 3 Set p = p_0 - f(p_0)/f'(p_0) (Compute p_i)

    p = p_0 - (f(p_0)/fp(p_0))

## Step 4 if |p - p_0| < TOL then OUTPUT(p); (The procedure was successful.) STOP

    if np.absolute(p - p_0) < TOL:
        print(f'The procedure was successful. p_0: {p_0} n: {i}')
        break
## Step 5 Set i= i + 1.
    i = i + 1

## Step 6 Set p_0 = p. (Update p_0)
    p_0 = p
    results.append(p)

## Step 7 OUTPUT ('The method failed after N-0 iterations, N-0 =', N_0)
## (The procedure was unsuccessful. STOP)
if i > Iterations:
    print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

```
The procedure was successful. p_0: 11.07790354509074 n: 3
```

The second injection should be given at 11.0078 hours or

```
In [8]: p_0 * 60
Out[8]: 664.6742127054443
```

^ Number of minutes after the first injection.

Part 3: 75% of the original injection. Still subtracting .25 from the formula:

In [9]:
```
def f(t):
    return (.75)*(math.exp(1)/3) * t * (math.exp(-t/3)) - (.25)

def fp(t):
    return - (1/9) * math.exp(1-(t/3)) * (t-3)


p_0 = 11
TOL = 1 * 10**-6
Iterations = 20

results = []   ## Need to save the results to something
results.append(p_0)
## Step 1 - Set i = 1

i = 1

## Step 2 - While (i <= N_0) do steps 3-6

while i <= Iterations:

## Step 3 Set p = p_0 - f(p_0)/f'(p_0) (Compute p_i)

    p = p_0 - (f(p_0)/fp(p_0))

## Step 4 if |p - p_0| < TOL then OUTPUT(p); (The procedure was successful.) STOP

    if np.absolute(p - p_0) < TOL:
        print(f'The procedure was successful. p_0: {p_0} n: {i}')
        break
## Step 5 Set i= i + 1.
    i = i + 1

## Step 6 Set p_0 = p. (Update p_0)
    p_0 = p
    results.append(p)

## Step 7 OUTPUT ('The method failed after N-0 iterations, N-0 =', N_0)
## (The procedure was unsuccessful. STOP)
if i > Iterations:
    print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

```
The procedure was successful. p_0: 9.867844878232496 n: 11
```

In [10]:
```
11.0779035450907 + 9.86784487823250
```

Out[10]:
```
20.945748423323202
```

^ hr

Question 3

Newton's Method

In [11]:
```
## Newton's method
## Testing for pg.68
## Chapter 2.3 Example 1

def f(x):
    return x**2 - 2

def fp(x):
    return 2*x

## INPUT initial approximation p_0; tolerance TOL; maximum number of iterations N_0.

p_0 = 1
TOL = 1 * 10**-6
Iterations = 10

results = []   ## Need to save the results to something
## Step 1 - Set i = 1

i = 1

## Step 2 - While (i <= N_0) do steps 3-6

while i <= Iterations:

## Step 3 Set p = p_0 - f(p_0)/f'(p_0) (Compute p_i)

    p = p_0 - (f(p_0)/fp(p_0))

## Step 4 if |p - p_0| < TOL then OUTPUT(p); (The procedure was successful.) STOP

    if np.absolute(p - p_0) < TOL:
        print(f'The procedure was successful. p_0: {p_0} n: {i-1}')
        break
## Step 5 Set i= i + 1.
    i = i + 1

## Step 6 Set p_0 = p. (Update p_0)
    p_0 = p
    results.append(p)

## Step 7 OUTPUT ('The method failed after N-0 iterations, N-0 =', N_0)
```

```
## (The procedure was unsuccessful. STOP)
if i > Iterations:
    print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

```
The procedure was successful. p_0: 1.4142135623746899 n: 4
```

Compare Bisection Method Results To Newton's Method

In [12]:
```
## Bisection method (pg. 49)
## Testing for pg.50
## Chapter 2.1 Example 1

## INPUT endpoints a,b; tolerance TOL; maximum number of iterations N_0

## Define the function
## Maybe find an easy way to find the derivative?
def f(x):
    return x**2 - 2

## INPUT endpoints a,b; tolerance TOL; maximum number of iterations N_0

a = 1
b = 2

TOL = 1 * 10**-6
Iterations = 30

results = []   ## Need to save the results to something

## OUTPUT approximate solution p or message of failure
## Step 1 Set i - 1; FA = f(a)

i = 1
FA = f(a)

## Step 2 While i <= N_0 do steps 3-6

while i <= Iterations:

## Step 3 Set p = 1 + (b - a)/2; (Compute P-i)

    p = a + (b - a)/2
    FP = f(p)

## Step 4 If FP = 0 or (b - a)/2 < TOL then OUTPUT(p); (Procedure completed successfully) STOP
    if FP == 0 or (b - a)/2 < TOL:
        print(f'The procedure was successful. p: {p} n: {i}')
        break
## Step 5 Set i = i + 1
    i = i + 1
```

```
## Step 6 If FA * FP > 0 then set a = p; (Compute a_i, b_i) FA = FP; else set b = p. (FA is unchanged)
    if FA * FP > 0:
        a = p
        FA = FP
    else:
        b = p

    results.append(p)
## Step 7 OUTPUT ('Method failed after N_0 iterations, N_0 =', N_0); STOP
if i > Iterations:
    print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

```
The procedure was successful. p: 1.4142141342163086 n: 20
```

Compare Secant Method Results To Newton's Method

In [13]:
```
## Secant method
## Testing for pg.71
## Chapter 2.3 Example 2

# Define the function

def f(x):
    return x**2 - 2

## Define bounderies

a = 1
b = 2

## INPUT initial approximations p_0, p; tolerance TOL; maximum number of iterations N_0.

p_0 = 1
p_1 = 2
TOL = 1 * 10**-6
Iterations = 20

results = []   ## Need to save the results to something

## Step 1 set i = 2; q_0 = f(p_0); q_1 = f(p_1)

i = 2
q_0 = f(p_0)
q_1 = f(p_1)

## Step 2 While i <= N_0 do Steps 3-6

while i <= Iterations:

## Step 3 Set p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)    (Compute p_i)
    p = p_1 - (q_1 * (p_1 - p_0)/(q_1 - q_0))
## Step 4 If |p - p_1| < TOL then OUTPUT(p); (The procedure was successful.) STOP
    if np.absolute(p - p_1) < TOL:
        print(f'The procedure was successful. p: {p} n: {i - 2}')
        break
## Step 5 Set i = i + 1
    i = i + 1
## Step 6 Set p_0 = p_1; q_0 = q_1; p_1 = p; q_1 = f(p) (Update P_0, q_0, p_1, q_1)
```

```
    p_0 = p_1
    q_0 = q_1
    p_1 = p
    q_1 = f(p)

    results.append(p)
## Step 7 OUTPUT ('The method failed after N_0 iterations, N_0 =', N_0);
    ## (The procedure was unsuccessful.) STOP
if i > Iterations:
    print(f'The method failed after N_0 iterations, N_0 = {Iterations}')
```

```
The procedure was successful. p: 1.4142135623730954 n: 5
```