Lab1: Introduction to coding using PYTHON

Student: Jorge González Cardelús

GroupID: 1 B

Date: 09/09/2019

In [1]:
```
import math

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

def title(text):
    print()
    print(text)
    print()

value = 12
```

Calculus of

$$number^3$$

using python's built in functions

In [2]:
```
cube_a = value ** 3
print("Cube of %s is %s" % (value, cube_a))

cube_b = value * value * value
print("Cube of %s is %s" % (value, cube_b))
```

```
if cube_a == cube_b:
    print("Same operation!")
```

```
Cube of 12 is 1728
Cube of 12 is 1728
Same operation!
```

Calculus of powers using `math` and `numpy`

In [3]:

```
cube_c = math.pow(value, 3)
print("Cube of %s is %s" % (value, cube_c))

cube_d = np.power(value, 3)
print("Cube of %s is %s" % (value, cube_d))

if cube_c == cube_d:
    print("Same operation!")
```

```
Cube of 12 is 1728.0
Cube of 12 is 1728
Same operation!
```

First contact with loops

In [4]:

```
import time

seconds = 10
delay = 1
print("Starting timer of %s seconds with step of %s second/s" % (second
s, delay)) #Prints info about timer
for i in reversed(range(seconds)): #For loop, it iterates second number
 of times in reverse order
    print("Timer: %s" % (str(i))) #Prins time left in timer
    time.sleep(delay) #Stops the program for delay seconds

print("Moving on!")
```

```
Starting timer of 10 seconds with step of 1 second/s
Timer: 9
Timer: 8
Timer: 7
Timer: 6
Timer: 5
Timer: 4
Timer: 3
Timer: 2
Timer: 1
Timer: 0
Moving on!
```

Calculate a polinomic function

In [5]:

```
instances = 100
x = np.linspace(-5, 5, instances) #Create 10 elements with the same dis
tance appart from one another between the range [-5, 5]
print(x)

y_math = []

title("Calculation of f(x) with math")
for i in range(len(x)):
    n = x[i]
    inside_sqrt = 1
    for j in reversed(range(1, 5)):
        inside_sqrt = inside_sqrt + math.pow(n, j)

    result = math.sqrt(inside_sqrt)
    y_math.append(result)
    print("f(%s) = %s" % (n, result))

y_math = np.array(y_math) #Conver to numpy array

title("Calculation of f(x) with numpy")
inside_sqrt = np.full(x.shape, 1) #Create np array of x.shape (100, 1)
```

```
 and fill it with 1s
for j in reversed(range(1, 5)):
    inside_sqrt = np.add(inside_sqrt, np.power(x, j)) #Calculate x^j fr
om 4, 1 and added to inside_sqrt array

result = np.sqrt(inside_sqrt) #Sqrt of array inside_sqrt
y_numpy = result
print(result)

#Check that they are equal
if y_numpy.all() == y_math.all():
    print("Both funcitons have the same results.")
```

```
[-5.         -4.8989899  -4.7979798  -4.6969697  -4.5959596  -4.4949494
9
 -4.39393939 -4.29292929 -4.19191919 -4.09090909 -3.98989899 -3.8888888
9
 -3.78787879 -3.68686869 -3.58585859 -3.48484848 -3.38383838 -3.2828282
8
 -3.18181818 -3.08080808 -2.97979798 -2.87878788 -2.77777778 -2.6767676
8
 -2.57575758 -2.47474747 -2.37373737 -2.27272727 -2.17171717 -2.0707070
7
 -1.96969697 -1.86868687 -1.76767677 -1.66666667 -1.56565657 -1.4646464
6
 -1.36363636 -1.26262626 -1.16161616 -1.06060606 -0.95959596 -0.8585858
6
 -0.75757576 -0.65656566 -0.55555556 -0.45454545 -0.35353535 -0.2525252
5
 -0.15151515 -0.05050505  0.05050505  0.15151515  0.25252525  0.3535353
5
  0.45454545  0.55555556  0.65656566  0.75757576  0.85858586  0.9595959
6
  1.06060606  1.16161616  1.26262626  1.36363636  1.46464646  1.5656565
7
  1.66666667  1.76767677  1.86868687  1.96969697  2.07070707  2.1717171
7
  2.27272727  2.37373737  2.47474747  2.57575758  2.67676768  2.7777777
8
  2.87878788  2.97979798  3.08080808  3.18181818  3.28282828  3.3838383
```

```
8
  3.48484848  3.58585859  3.68686869  3.78787879  3.88888889  3.9898989
9
  4.09090909  4.19191919  4.29292929  4.39393939  4.49494949  4.5959596
  4.6969697   4.7979798   4.8989899   5.        ]

Calculation of f(x) with math

f(-5.0) = 22.825424421026653
f(-4.898989898989899) = 21.87532290457873
f(-4.797979797979798) = 20.945612887857454
f(-4.696969696969697) = 20.036294550667755
f(-4.595959595959596) = 19.147368199410387
f(-4.494949494949495) = 18.278834295152603
f(-4.393939393939394) = 17.4306934876723
f(-4.292929292929293) = 16.602946656836206
f(-4.191919191919192) = 15.795594963012745
f(-4.090909090909091) = 15.0086399086545
f(-3.9898989898989896) = 14.242083413741504
f(-3.888888888888889) = 13.495927908493861
f(-3.787878787878788) = 12.770176447690732
f(-3.686868686868687) = 12.06483285214179
f(-3.5858585858585856) = 11.3799018844402
f(-3.484848484848485) = 10.715389468210393
f(-3.383838383838384) = 10.071302962824657
f(-3.282828282828283) = 9.447651509241538
f(-3.1818181818181817) = 8.844446467552185
f(-3.080808080808081) = 8.261701973477635
f(-2.9797979797979797) = 7.699435650099772
f(-2.878787878787879) = 7.157669523461461
f(-2.7777777777777777) = 6.63643120765268
f(-2.676767676767677) = 6.13575544847497
f(-2.5757575757575757) = 5.655686147383436
f(-2.474747474747475) = 5.196279032853064
f(-2.3737373737373737) = 4.757605209740746
f(-2.272727272727273) = 4.339755905529941
f(-2.1717171717171717) = 3.9428488543230285
f(-2.070707070707071) = 3.5670369248506737
f(-1.9696969696969697) = 3.2125198147565444
```

```
f(-1.868686868686869) = 2.8795598939320786
f(-1.7676767676767677) = 2.5685035377740415
f(-1.6666666666666665) = 2.2798093920759097
f(-1.5656565656565657) = 2.0140845401974135
f(-1.4646464646464645) = 1.772127511046062
f(-1.3636363636363638) = 1.5549713609381646
f(-1.2626262626262625) = 1.36390687578687
f(-1.1616161616161618) = 1.2004406851919607
f(-1.0606060606060606) = 1.066107793116274
f(-0.9595959595959593) = 0.9620429579695005
f(-0.858585858585859) = 0.888301463663565
f(-0.7575757575757578) = 0.8431737424028319
f(-0.6565656565656566) = 0.8229885367698734
f(-0.5555555555555554) = 0.8227262756319079
f(-0.45454545454545503) = 0.8371619356791724
f(-0.3535353535353538) = 0.8619085184078478
f(-0.2525252525252526) = 0.893983754200869
f(-0.15151515151515138) = 0.9319283217689762
f(-0.050505050505050164) = 0.9756656136862026
f(0.050505050505050164) = 1.0262510137767498
f(0.15151515151515138) = 1.0855769518888607
f(0.2525252525252526) = 1.1560553728431853
f(0.3535353535353538) = 1.2402950542994704
f(0.45454545454545414) = 1.3408056219550997
f(0.5555555555555554) = 1.45976887343407
f(0.6565656565656566) = 1.598907148912167
f(0.7575757575757578) = 1.7594518438671236
f(0.8585858585858581) = 1.9421890039000347
f(0.9595959595959593) = 2.1475468168430543
f(1.0606060606060606) = 2.375693183572022
f(1.1616161616161618) = 2.626623039595313
f(1.262626262626262) = 2.9002267838606417
f(1.3636363636363633) = 3.1963390881907787
f(1.4646464646464645) = 3.5147712009125014
f(1.5656565656565657) = 3.855330910167036
f(1.666666666666667) = 4.217833976911625
f(1.7676767676767673) = 4.602110008141612
f(1.8686868686868685) = 5.0080048850870105
f(1.9696969696969697) = 5.43538116447439
```

```
f(2.070707070707071) = 5.8841173627292696
f(2.1717171717171713) = 6.3541066856630755
f(2.2727272727272725) = 6.845255538927004
f(2.3737373737373737) = 7.3574820105340955
f(2.474747474747475) = 7.890714427965367
f(2.5757575757575752) = 8.444890039054087
f(2.6767676767676765) = 9.019953834783703
f(2.7777777777777777) = 9.615857514780005
f(2.878787878787879) = 10.232558587198112
f(2.9797979797979792) = 10.870019590512063
f(3.0808080808080813) = 11.5282074233266
f(3.1818181818181817) = 12.207092768482715
f(3.282828282828282) = 12.906649598667864
f(3.383838383838384) = 13.626854752026357
f(3.4848484848484844) = 14.3676875676469
f(3.5858585858585865) = 15.129129572146798
f(3.686868686868687) = 15.911164209808382
f(3.787878787878787) = 16.713776609827022
f(3.8888888888888893) = 17.536953385193396
f(3.9898989898989896) = 18.38068245856429
f(4.09090909090909) = 19.24495291118653
f(4.191919191919192) = 20.12975485154122
f(4.292929292929292) = 21.03507930088605
f(4.3939393939393945) = 21.960918093303995
f(4.494949494949495) = 22.907263788228644
f(4.595959595959595) = 23.874109593722903
f(4.696969696969697) = 24.86149299044025
f(4.797979797979798) = 25.86927721524519
f(4.8989898989899) = 26.8975881227468
f(5.0) = 27.94637722496424

Calculation of f(x) with numpy

[22.82542442 21.8753229  20.94561289 20.03629455 19.1473682  18.2788343
 17.43069349 16.60294666 15.79559496 15.00863991 14.24208341 13.4959279
1
 12.77017645 12.06483285 11.37990188 10.71538947 10.07130296  9.4476515
1
  8.84444647  8.26170197  7.69943565  7.15766952  6.63643121  6.1357554
```

```
5
  5.65568615  5.19627903  4.75760521  4.33975591  3.94284885  3.5670369
2
  3.21251981  2.87955989  2.56850354  2.27980939  2.01408454  1.7721275
1
  1.55497136  1.36390688  1.20044069  1.06610779  0.96204296  0.8883014
6
  0.84317374  0.82298854  0.82272628  0.83716194  0.86190852  0.8939837
5
  0.93192832  0.97566561  1.02625101  1.08557695  1.15605537  1.2402950
5
  1.34080562  1.45976887  1.59890715  1.75945184  1.942189    2.1475468
2
  2.37569318  2.62662304  2.90022678  3.19633909  3.5147712   3.8553309
1
  4.21783398  4.60211001  5.00800489  5.43538116  5.88411736  6.3541066
9
  6.84525554  7.35748201  7.89071443  8.44489004  9.01995383  9.6158575
1
 10.23255859 10.87001959 11.52820742 12.20709277 12.9066496  13.6268547
5
 14.36768757 15.12912957 15.91116421 16.71377661 17.53695339 18.3806824
6
 19.24495291 20.12975485 21.0350793  21.96091809 22.90726379 23.8741095
9
 24.8614493  25.86927722 26.89758812 27.94637722]
Both funcitons have the same results.
```

Plot a polynomic function

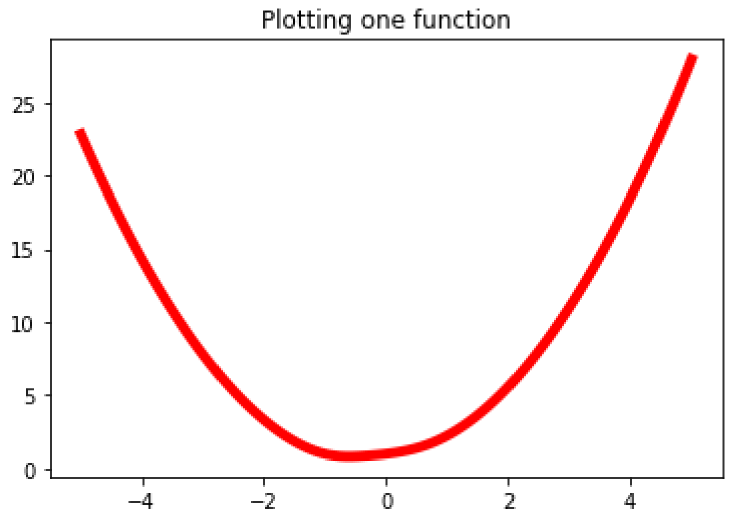
The function above is

$$f(x) = \sqrt{x^4 + x^3 + x^2 + x + 1}$$

.

In [6]:
```
plt.plot(x, y_numpy, 'r', lw=5) #Print x, and f(x) in red with a linewi
```

```
dth of 5
plt.title("Plotting one function")
plt.show()
```



Update existing values

By chaging `x = np.linspace(-5, 5, 10)` to `x = np.linspace(-5, 5, 100)`, instead of having 10 instances in x, we will have 100 which will cause the loops to iterate over more instances and for the plotting of the function above to be more defined and precise.

Plot two functions

Functions to be plotted

$$f_1(x) = \sqrt{x^4 + x^3 + x^2 + x + 1}$$

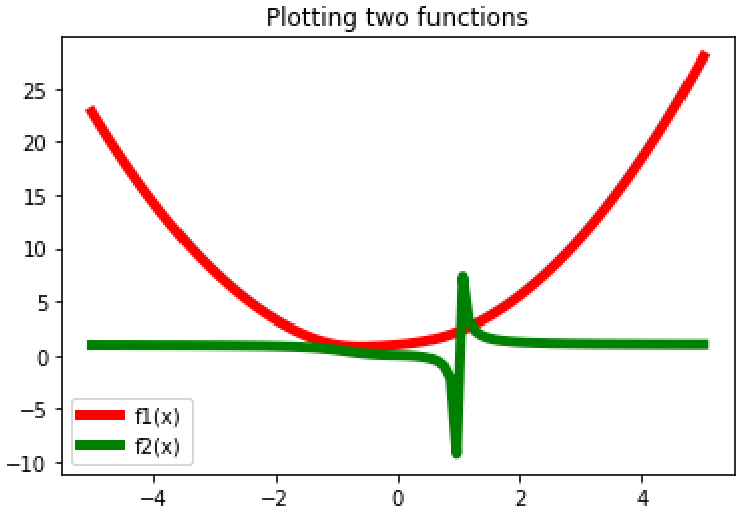
$$f_2(x) = \frac{\log\left(x^4 + x^3 + x^2 + x + 1\right)}{\log\left(x^4\right)}$$

In [7]:

```
'''
For this block will be reusing x which is a np.linspace: [-5, 5] with 1
00 instances and inside_sqrt since the functions
both include x^4 + x^3... and y_numpy which has been already calculated
'''

y_func_a = y_numpy
inside_upper_log = inside_sqrt
y_func_b = np.log10(inside_upper_log) / np.log10(np.power(x, 4))

plt.plot(x, y_func_a, 'r', lw=5, label="f1(x)") #Print x, and f(x) in r
ed with a linewidth of 5
plt.plot(x, y_func_b, 'g', lw=5, label="f2(x)") #Print x, and f(x) in g
reen with a lw of 5
plt.title("Plotting two functions")
plt.legend()
plt.show()
```



In [ ]: