

## Pythonサンプルコード

Pythonによる数学入門  $\pi$ の計算、エラトステネスの篩で素数を拾う

### ##2のn乗の計算とハローワールド

```
import sys
import math
sys.stdout.write("Hello, World!\n")
n = int(input())

for i in range(n+1):
    sys.stdout.write("%0.f " % math.exp(math.log(2)*i))
```

計算結果:

```
Hello, World!
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072 262144
524288 1048576 2097152 4194304 8388608 16777216 33554432 67108864 134217728
268435456 536870912 1073741824
```

### ## 円周率 $\pi$ とネイピア数 $e$ をmath.数学ライブラリから求める

```
import sys
import math

pi = math.pi
sys.stdout.write("%f \n" % pi)
print(math.pi)
sys.stdout.write("%.15f \n" % math.pi)

napier = math.e
sys.stdout.write("%f \n" % napier)
print (math.e)
sys.stdout.write("%.15f \n" % math.e)
```

計算結果:

```
3.141593
3.141592653589793
3.141592653589793
2.718282
2.718281828459045
2.718281828459045
```

### ## 円周率 $\pi$ をアルゴリズムで求める

```
## arctan(1/n)をTaylor展開で求める
```

```
def arctan(p, n):
```

```
    x = p/n
    nn = n*n
    c = 1
    s = x
    k = 1
    while x > 0:
        x /= nn
        k += 2
        c = -c
        s += c*(x/k)
    return s
```

```
## 整数部分「3」も含むmantissa仮数表現の長整数で返します
```

```
def pi_mantissa(digit, redund=10):
```

```
    p = 10**digit
    q = 10**redund
    p *= q
    pi = 4*(12*arctan(p,18)+8*arctan(p,57)-5*arctan(p,239))
    pi /= q
    return pi
```

```
print (pi_mantissa(100))
print (pi_mantissa(200))
```

計算結果:

```
3.141592653589794e+100
3.141592653589793e+200
```

```
#素数を求める
```

```
#エラトステネスの篩
```

```
import math
```

```
def sieve(n):
```

```
    nums = [i+1 for i in range(2, n, 2) if (i+1) % 3 != 0 and (i+1) % 5 != 0]
    ans = [2,3,5]
    while nums[0] <= math.sqrt(n):
        for i in range(nums[0]**2, nums[-1]+1, nums[0]):
            if i in nums: nums.remove(i)
        ans.append(nums.pop(0))
    ans += nums
    return ans
```

```
print (sieve(10000))
```

計算結果:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833, 3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253, 4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549, 4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861, 4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011, 5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233, 5237, 5261, 5273, 5279, 5281, 5297, 5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443, 5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527,

5531, 5557, 5563, 5569, 5573, 5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717, 5737, 5741, 5743, 5749, 5779, 5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073, 6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173, 6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269, 6271, 6277, 6287, 6299, 6301, 6311, 6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581, 6599, 6607, 6619, 6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779, 6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911, 6917, 6947, 6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019, 7027, 7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219, 7229, 7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411, 7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561, 7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669, 7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741, 7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853, 7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919, 7927, 7933, 7937, 7949, 7951, 7963, 7993, 8009, 8011, 8017, 8039, 8053, 8059, 8069, 8081, 8087, 8089, 8093, 8101, 8111, 8117, 8123, 8147, 8161, 8167, 8171, 8179, 8191, 8209, 8219, 8221, 8231, 8233, 8237, 8243, 8263, 8269, 8273, 8287, 8291, 8293, 8297, 8311, 8317, 8329, 8353, 8363, 8369, 8377, 8387, 8389, 8419, 8423, 8429, 8431, 8443, 8447, 8461, 8467, 8501, 8513, 8521, 8527, 8537, 8539, 8543, 8563, 8573, 8581, 8597, 8599, 8609, 8623, 8627, 8629, 8641, 8647, 8663, 8669, 8677, 8681, 8689, 8693, 8699, 8707, 8713, 8719, 8731, 8737, 8741, 8747, 8753, 8761, 8779, 8783, 8803, 8807, 8819, 8821, 8831, 8837, 8839, 8849, 8861, 8863, 8867, 8887, 8893, 8923, 8929, 8933, 8941, 8951, 8963, 8969, 8971, 8999, 9001, 9007, 9011, 9013, 9029, 9041, 9043, 9049, 9059, 9067, 9091, 9103, 9109, 9127, 9133, 9137, 9151, 9157, 9161, 9173, 9181, 9187, 9199, 9203, 9209, 9221, 9227, 9239, 9241, 9257, 9277, 9281, 9283, 9293, 9311, 9319, 9323, 9337, 9341, 9343, 9349, 9371, 9377, 9391, 9397, 9403, 9413, 9419, 9421, 9431, 9433, 9437, 9439, 9461, 9463, 9467, 9473, 9479, 9491, 9497, 9511, 9521, 9533, 9539, 9547, 9551, 9587, 9601, 9613, 9619, 9623, 9629, 9631, 9643, 9649, 9661, 9677, 9679, 9689, 9697, 9719, 9721, 9733, 9739, 9743, 9749, 9767, 9769, 9781, 9787, 9791, 9803, 9811, 9817, 9829, 9833, 9839, 9851, 9857, 9859, 9871, 9883, 9887, 9901, 9907, 9923, 9929, 9931, 9941, 9949, 9967, 9973]

#グラフへプロットする

```
import sys
import math
x_numbers = [1, 2, 3]
y_numbers = [2, 4, 6]
from pylab import plot, show
plot(x_numbers, y_numbers)
```

>>show()

グラフ省略

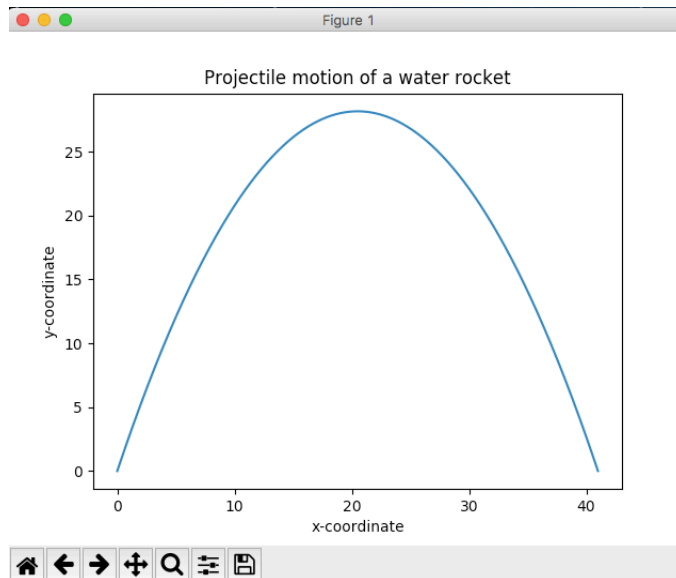
## 水ロケットを発射する

```
import sys
import math
from matplotlib import pyplot as plt
def draw_graph(x, y):
    plt.plot(x, y)
    plt.xlabel('x-coordinate')
    plt.ylabel('y-coordinate')
    plt.title('Projectile motion of a water rocket')

def frange(start, final, interval):
    numbers = []
    while start < final:
        numbers.append(start)
        start = start + interval
    return numbers

def draw_trajectory(u, theta):
    theta = math.radians(theta)
    g = 9.8
    # Time of flight
    t_flight = 2*u*math.sin(theta)/g
    #find time interval
    intervals = frange(0, t_flight, 0.001)
    # list of x and y coordinates
    x = []
    y = []
    for t in intervals:
        x.append(u*math.cos(theta)*t)
        y.append(u*math.sin(theta)*t - 0.5*g*t*t)
    draw_graph(x, y)

if __name__=="__main__":
    try:
        u =25.0      # 'the initial velocity m/s:'
        theta = 70.0 # 'the angle of projection (degree)'
    except ValueError:
        print ("error")
    else:
        draw_trajectory(u, theta)
        plt.show()
```

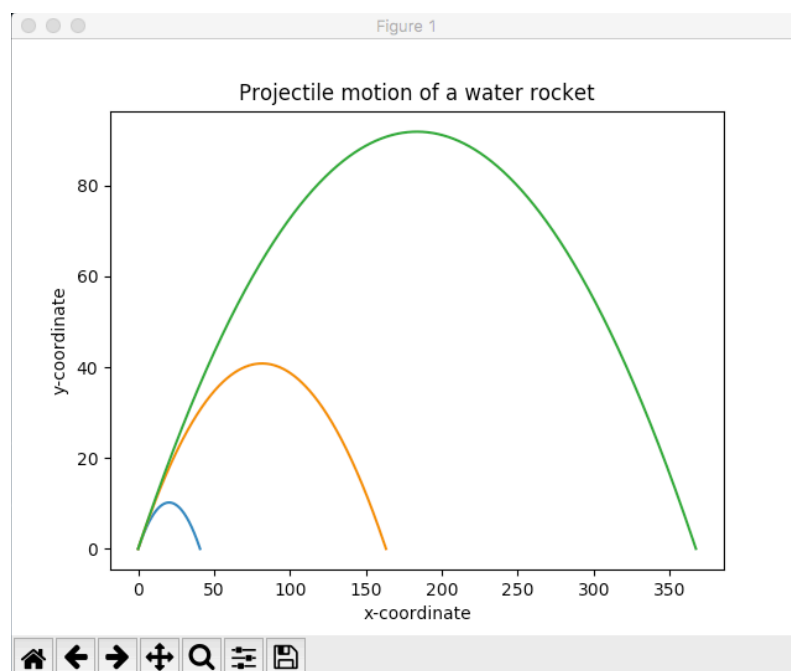


# 速度条件を3条件に変えてプロット

```
if __name__=="__main__":
    u_list = [20, 40, 60]
    theta = 45
    for u in u_list:
        draw_trajectory(u, theta)

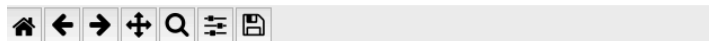
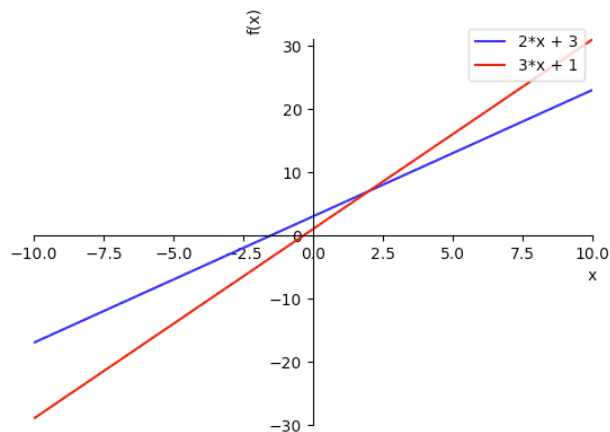
    plt.legend(['20', '40', '60'])
    plt.show()
```

計算結果：



```
Last login: Thu Feb 16 10:05:53 on ttys000
KUSAFUSA-no-Mac-mini:~ kusafusasejiro$ python
Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016,
13:19:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy.plotting import plot
>>> from sympy import Symbol
>>> x = Symbol("x")
>>> p = plot(2*x=3, 3*x+1, legend=True, show=False)
File "<stdin>", line 1
SyntaxError: keyword can't be an expression
>>> p = plot(2*x+3, 3*x+1, legend=True, show=False)
>>> p[0].line_color = 'b'
>>> p[1].line_color = 'r'
>>> p.show()
```

```
kusafusasejiro — python — 80x24
Last login: Thu Feb 16 10:05:53 on ttys000
KUSAFUSA-no-Mac-mini:~ kusafusasejiro$ python
Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13:19:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy.plotting import plot
>>> from sympy import Symbol
>>> x = Symbol("x")
>>> p = plot(2*x=3, 3*x+1, legend=True, show=False)
File "<stdin>", line 1
SyntaxError: keyword can't be an expression
>>> p = plot(2*x+3, 3*x+1, legend=True, show=False)
>>> p[0].line_color = 'b'
>>> p[1].line_color = 'r'
>>> p.show()
```



```

# これらの航跡をアニメーションで表示する

#...
# Animate the trajectory of an object in projectile motion
# - C. Kyaдpyca, Refer to Doing math with Python Amit Saha
# ...
from matplotlib import pyplot as plt
from matplotlib import animation
import math
g = 9.8
def get_intervals(u, theta):
    t_flight = 2*u*math.sin(theta)/g
    intervals = []
    start = 0
    interval = 0.005
    while start < t_flight:
        intervals.append(start)
        start = start + interval
    return intervals

def update_position(i, circle, intervals, u, theta):
    t = intervals[i]
    x = u*math.cos(theta)*t
    y = u*math.sin(theta)*t - 0.5*g*t*t
    circle.center = x, y
    return circle,

def create_animation(u, theta):
    intervals = get_intervals(u, theta)
    xmin = 0
    xmax = u*math.cos(theta)*intervals[-1]
    ymin = 0
    t_max = u*math.sin(theta)/g
    ymax = u*math.sin(theta)*t_max - 0.5*g*t_max**2
    fig = plt.gcf()
    ax = plt.axes(xlim=(xmin, xmax), ylim=(ymin, ymax))
    circle = plt.Circle((xmin, ymin), 1.0)
    ax.add_patch(circle)
    anim = animation.FuncAnimation(fig, update_position,
                                  fargs=(circle, intervals, u, theta),
                                  frames=len(intervals), interval=1,
                                  repeat=False)

    plt.title('Projectile Motion')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.show()

```



```

if __name__ == '__main__':
    try:
        u = float(input('Enter the initial velocity(m/s): '))
        theta = float(input('Enter the angle of projection (degree): '))
    except ValueError:
        print('You entered an invalid input')
    else:
        theta = math.radians(theta)
        create_animation(u, theta)

```

# 勾配上昇法を用いて最大値を求めるジェネリックなプログラム

# Use gradient ascent to find the maximum value of a single variable function

```

from sympy import Derivative, Symbol, sympify

```

```

def grad_ascent(x0, f1x, x):
    epsilon = 1e-6
    step_size = 1e-4
    x_old = x0
    x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
    while abs(x_old - x_new) > epsilon:
        x_old = x_new
        x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
    return x_new

```

```

if __name__ == '__main__':
    f = input('Enter a function one variable: ')
    var = input('Enter the variable to differentiate with respect to: ')
    var0 = float(input('Enter the initial value of the variable: '))
    try:
        f = sympify(f)
    except SympifyError:
        print('You entered an invalid input')
    else:
        var = Symbol(var)
        d = Derivative(f, var).doit()
        var_max = grad_ascent(var0, d, var)
        print('{0}: {1}'.format(var.name, var_max))
        print('Maximun vale: {0}'.format(f.subs({var:var_max})))

```

計算結果：

```
Last login: Sun Feb 19 11:44:09 on ttys000
KUSAFUSA-no-Mac-mini:~ kusafusaseijiro$ python
Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016,
13:19:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import Derivative, Symbol, sympify
>>>
>>> def grad_ascent(x0, f1x, x):
...     epsilon = 1e-6
...     step_size = 1e-4
...     x_old = x0
...     x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
...     while abs(x_old - x_new) > epsilon:
...         x_old = x_new
...         x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
...     return x_new
...
>>> if __name__=='__main__':
...     f = input('Enter a function one variable: ')
...     var = input('Enter the variable to differentiate with respect
to: ')
...     var0 = float(input('Enter the initial value of the variable:
'))
...     try:
...         f = sympify(f)
...     except SympifyError:
...         print('You entered an invalid input')
...     else:
...         var = Symbol(var)
...         d = Derivative(f, var).doit()
...         var_max = grad_ascent(var0, d, var)
...         print('{0}: {1}'.format(var.name, var_max))
...         print('Maximun vale: {0}' .format(f.subs({var:var_max})))
...
Enter a function one variable: 25*25*sin(2*theta)/9.8
Enter the variable to differentiate with respect to: theta
Enter the initial value of the variable: 0.001
theta: 0.785360029379083
Maximun vale: 63.7755100185965
```

```

Last login: Sun Feb 19 11:44:09 on ttys000
[KUSAFUSA-no-Mac-mini:~ kusafusasejiro$ python
Python 3.6.0 |Anaconda 4.3.0 (x86_64)| (default, Dec 23 2016, 13:19:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import Derivative, Symbol, sympify
>>>
>>> def grad_ascent(x0, f1x, x):
...     epsilon = 1e-6
...     step_size = 1e-4
...     x_old = x0
...     x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
...     while abs(x_old - x_new) > epsilon:
...         x_old = x_new
...         x_new = x_old + step_size*f1x.subs({x:x_old}).evalf()
...     return x_new
...
>>> if __name__=='__main__':
...     f = input('Enter a function one variable: ')
...     var = input('Enter the variable to differentiate with respect to: ')
...     var0 = float(input('Enter the initial value of the variable: '))
...     try:
...         f = sympify(f)
...     except SympifyError:
...         print('You entered an invalid input')
...     else:
...         var = Symbol(var)
...         d = Derivative(f, var).doit()
...         var_max = grad_ascent(var0, d, var)
...         print('{0}: {1}'.format(var.name, var_max))
...         print('Maximun vale: {0}' .format(f.subs({var:var_max})))
...
[Enter a function one variable: 25*25*sin(2*theta)/9.8
[Enter the variable to differentiate with respect to: theta
[Enter the initial value of the variable: 0.001
theta: 0.785360029379083
Maximun vale: 63.7755100185965
>>> █
```

refer Doing math with Python Amit Saha, O'Reilly