

```
In [3]: #1 Write a function that inputs a number and prints the multiplication table of that number
def mult(num):
    #Function implementation
    for multiplier in range(1,11):
        print('{0} * {1} = {2}'.format(num,multiplier,num*multiplier))

mult(int(input('Enter the Number to fetch its Table :')))) #Function Call

Enter the Number to fetch its Table :7
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

In [11]: #2. Write a program to print twin primes less than 1000.

prime_num=[]
twin_prime=[]
for num in range(3,1001):
    # As 2 is neither prime nor composite, so started from 3
    #pdb.set_trace()
    for i in range(2,num):
        if(num%i==0):
            break
        else:
            prime_num.append(num)
            if len(prime_num)>1:
                twin_prime.append((prime_num[-2],prime_num[-1]))
print('List of Twin Primes:')
print(twin_prime)

List of Twin Primes:
[(3, 5), (5, 7), (7, 11), (11, 13), (13, 17), (17, 19), (19, 23), (23, 29), (29, 31), (31, 37),
(37, 41), (41, 43), (43, 47), (47, 53), (53, 59), (59, 61), (61, 67), (67, 71), (71, 73), (73, 7
9), (79, 83), (83, 89), (89, 97), (97, 101), (101, 103), (103, 107), (107, 109), (109, 113), (11
3, 127), (127, 131), (131, 137), (137, 139), (139, 149), (149, 151), (151, 157), (157, 163), (16
3, 167), (167, 173), (173, 179), (179, 181), (181, 191), (191, 193), (193, 197), (197, 199), (19
9, 211), (211, 223), (223, 227), (227, 229), (229, 233), (233, 239), (239, 241), (241, 251), (25
1, 257), (257, 263), (263, 269), (269, 271), (271, 277), (277, 281), (281, 283), (283, 293), (29
3, 307), (307, 311), (311, 313), (313, 317), (317, 331), (331, 337), (337, 347), (347, 349), (34
9, 353), (353, 359), (359, 367), (367, 373), (373, 379), (379, 383), (383, 389), (389, 397), (39
7, 401), (401, 409), (409, 419), (419, 421), (421, 431), (431, 433), (433, 439), (439, 443), (44
3, 449), (449, 457), (457, 461), (461, 463), (463, 467), (467, 479), (479, 487), (487, 491), (49
1, 499), (499, 503), (503, 509), (509, 521), (521, 523), (523, 541), (541, 547), (547, 557), (55
7, 563), (563, 569), (569, 571), (571, 577), (577, 587), (587, 593), (593, 599), (599, 601), (60
1, 607), (607, 613), (613, 617), (617, 619), (619, 631), (631, 641), (641, 643), (643, 647), (64
7, 653), (653, 659), (659, 661), (661, 673), (673, 677), (677, 683), (683, 691), (691, 701), (70
1, 709), (709, 719), (719, 727), (727, 733), (733, 739), (739, 743), (743, 751), (751, 757), (75
7, 761), (761, 769), (769, 773), (773, 787), (787, 797), (797, 809), (809, 811), (811, 821), (82
1, 823), (823, 827), (827, 829), (829, 839), (839, 853), (853, 857), (857, 859), (859, 863), (86
3, 877), (877, 881), (881, 883), (883, 887), (887, 907), (907, 911), (911, 919), (919, 929), (92
9, 937), (937, 941), (941, 947), (947, 953), (953, 967), (967, 971), (971, 977), (977, 983), (98
3, 991), (991, 997)]

In [9]: #3. Write a program to find out the prime factors of a number
num = int(input('Enter the number to find its factor : '))
print_num=num;
list1=[]

while num!=1:
    #Restart Logic implemented to restart for loop
    restart=False
    for i in range(2,num+1):
        if(num%i==0):
            num = num//i
            list1.append(i)
            restart=True #once the the 1st factor is found, force the code to find the second fact
or from 2 onwards
            break
print("Factors of {0} = {1}".format(print_num,list1))

Enter the number to find its factor : 162
Factors of 162 = [2, 3, 3, 3, 3]

In [ ]: #4. Write a program to implement these formulae of permutations and combinations.
#Number of permutations of n objects taken r at a time: p(n, r) = n! / (n-r)!
#Number of combinations of n objects taken r at a time is: c(n, r) = n! / (r!*(n-r)!) = p(n,r) / r!

n=int(input('Enter the number of elements : '))
r=int(input('Enter the samples : '))

def factorial(num):
    return 1 if num==1 else num*factorial(num-1)

permutation=factorial(n)/factorial(n-r)
combination=permutation/factorial(r)
print('Probability and Combination of P{0},{1} is {2} and {3}'.format(n,r,permutation,combination
))

In [14]: #5. Write a function that converts a decimal number to binary number
list1=[]
def dec2bin(num):
    list1.append(num%2)
    num=num//2
    if num==1:
        list1.append(1)
    return list1 if num==1 else dec2bin(num)

a=dec2bin(int(input('enter the Decimal number : ')))
a.reverse()
print(''.join(map(str,a)))

enter the Decimal number : 20
10100

In [17]: #6. Write a function cubesum() that accepts an integer and returns the sum of the cubes of
#individual digits of that number. Use this function to make functions PrintArmstrong() and
#isArmstrong() to print Armstrong numbers and to find whether is an Armstrong number.
def cubesum(num,b=0):
    a=divmod(num,10)
    num=a[0]
    b=b+a[1]**3
    return b if num==0 else cubesum(num,b)

def isArmstrong(calculated_cube,num):
    return True if calculated_cube==num else False

def PrintArmstrong(dec,num):
    print('Number {0} is an Armstrong Number'.format(num)) if dec else print('Number {0} is Not an A
rmstrong Number'.format(num))

num=int(input('Enter no to verify Armstong : '))
PrintArmstrong(isArmstrong(cubesum(num),num),num)

Enter no to verify Armstong : 160
Number 160 is Not an Armstrong Number

In [16]: #7. Write a function prodDigits() that inputs a number and returns the product of digits of that num
ber.
from functools import reduce
def prodDigits(str_num):
    return reduce(lambda x,y:x*y,[int(ele) for ele in str_num]) #Convert string into list of integ
er and then applied the Lambda function

a=input('Enter the number : ')
print('Product of {0} is {1}'.format(int(a),prodDigits(a)))

Enter the number : 4567
Product of 4567 is 840

In [18]: #8. Using the function prodDigits() of previous exercise write functions MDR() and
#MPersistence() that input a number and return its multiplicative digital root and
#multiplicative persistence respectively
#Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3)

from functools import reduce
def prodDigits(str_num):
    return reduce(lambda x,y:x*y,[int(ele) for ele in str_num])

def MDR(a,count=0):
    count=MPersistence(count)
    a=prodDigits(str(a))
    return (a,count) if len(str(a))==1 else MDR(a,count)
#Return a,count if len(str(a))==1 else MDR(a,count) #Doubt p If i try to runt his code , them t
he op is tuples of tuples.

def MPersistence(count):
    count=count+1
    return count

a=b=input('Enter the number : ')
op=MDR(a)
print('MDR and MPersistence of {0} is {1} and {2} respectively'.format(b,op[0],op[1]))

Enter the number : 123456
MDR and MPersistence of 123456 is 0 and 2 respectively

In [9]: #9. Write a function sumPdivisors() that finds the sum of proper divisors of a number. Proper
#divisors of a number are those numbers by which the number is divisible, except the
#number itself. For example proper divisors of 36 are 1, 2, 3, 4, 6, 9,12,18
from functools import reduce
def sumPdivisors(num):
    return reduce(lambda x,y:x+y,[ele for ele in range(1,(num//2)+1) if num%ele==0])

a=int(input('Enter the Number :'))
print('Sum of proper divisor of number {0} is {1}'.format(a,sumPdivisors(a)))

Enter the Number :36
Sum of proper divisor of number 36 is 55

In [1]: #10. A number is called perfect if the sum of proper divisors of that number is equal to the
#number. For example 28 is perfect number, since 1+2+4+7+14=28. Write a program to
#print all the perfect numbers in a given range
from functools import reduce
low=int(input('Enter the Low Number :'))
high=int(input('Enter the High Number :'))

lst=[]
for range_ele in range(low,high+1):
    a=0
    for ele in range(1,(range_ele//2) +1):
        #pdb.set_trace()
        if range_ele%ele==0:
            a=a+ele
    if a==range_ele:
        lst.append(a)
print('Perfect number between {0} and {1} are {2}'.format(low,high,lst))

Enter the Low Number :1
Enter the High Number :10000
Perfect number between 1 and 10000 are [6, 28, 496, 8128]

In [3]: #10 solution 2. A number is called perfect if the sum of proper divisors of that number is equal to
the
#number. For example 28 is perfect number, since 1+2+4+7+14=28. Write a program to
#print all the perfect numbers in a given range
def sumPdivisors(num):
    return reduce(lambda x,y:x+y,[ele for ele in range(1,(num//2)+1) if num%ele==0])

def perfect_number(num):
    #This Returns the a number if its perfect
    return num if (sumPdivisors(num)==num) else None

from functools import reduce
low=int(input('Enter the Low Number :'))
high=int(input('Enter the High Number :'))

'''If i>1 is used because function 'reduce()' of empty sequence with no initial value'
is thrown -> in the lambda function only x is present and not y for the initial iteration'''
number=[i for i in range(low,high+1) if i>1]
a=list(filter(perfect_number,number))
print('Perfect number between {0} and {1} are {2}'.format(low,high,a))

Enter the Low Number :1
Enter the High Number :1000
Perfect number between 1 and 1000 are [6, 28, 496]

In [1]: #11. Write a function to print pairs of amicable numbers in a range-
#Sum of proper divisors of 220 = 1+2+4+5+10+11+20+22+44+55+110 = 284 Sum of proper divisors of 284 =
1+2+4+7+14+28 = 220
def sumOfDiv(x):
    sum = 1
    for i in range(2, x):
        if x % i == 0:
            sum += i
    return sum

def isAmicable(a, b):
    if sumOfDiv(a) == b and sumOfDiv(b) == a:
        return True
    else:
        return False

def countPairs(arr, n):
    count = 0
    for i in range(0, n):
        for j in range(i + 1, n):
            if isAmicable(arr[i], arr[j]):
                count = count + 1
            amicable_list.append((arr[i], arr[j]))
    return count

a = int(input('Enter Lower range : '))
b = int(input('Enter upper range : '))
amicable_list=[]
# Driver Code
arr = [num for num in range(a,b+1)]
arr_len = len(arr)
print('Count of Amicable number in range {0} and {1} is {2} and the pairs of amicable numbers is {3}
'.format(a,b,countPairs(arr, arr_len),amicable_list))

Enter Lower range : 10
Enter upper range : 3000
Count of Amicable number in range 10 and 3000 is 3 and the pairs of amicable numbers is [(220, 2
84), (1184, 1210), (2620, 2924)]

In [5]: #12. Write a program which can filter odd numbers in a list by using filter function
num_list=[i for i in range(0,100)]
odd_list=list(filter(lambda x:(x%2!=0),num_list))
print('Odd List:\n',odd_list)

Odd List:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49,
51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97,
99]

In [6]: #13. Write a program which can map() to make a list whose elements are cube of elements in a given li
st
num_list=[i for i in range(0,100)]
cube_list=list(map(lambda x:x**3,num_list))
print(cube_list)

[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 4096, 4913, 5832,
6859, 8000, 9261, 10648, 12167, 13824, 15625, 17576, 19683, 21952, 24389, 27000, 29791, 32768, 3
5937, 39304, 42875, 46656, 50653, 54872, 59319, 64000, 68921, 74088, 79507, 85184, 91125, 97336,
103823, 110592, 117649, 125000, 132651, 140608, 148877, 157464, 166375, 175616, 185193, 195112,
205379, 216000, 226981, 238328, 250047, 262144, 274625, 287496, 300763, 314432, 328509, 343000,
357911, 373248, 389017, 405224, 421875, 438976, 456533, 474552, 493039, 512000, 531441, 551368,
571875, 593500, 616251, 640128, 665136, 691297, 718624, 747217, 777084, 808225, 840928, 875191,
911072, 948691, 988064, 1029297, 1072400, 1117391, 1164280, 1214077, 1266792, 1322435, 1381028,
1441591, 1504234, 1568977, 1636820, 1707873, 1782146, 1859649, 1940400, 2023547, 2109090, 2197039,
2287492, 2380359, 2476640, 2576345, 2679484, 2786067, 2896104, 3009605, 3126580, 3247041, 3371088,
3498731, 3629976, 3764821, 3903376, 4045641, 4191616, 4341301, 4494706, 4651841, 4812726, 4977371,
5145686, 5317671, 5493336, 5672581, 5855416, 6046841, 6246856, 6455461, 6672656, 6898441, 7132816,
7375781, 7627336, 7887481, 8156216, 8433641, 8719756, 9014551, 9318026, 9630171, 9951006, 10280631,
10618946, 10975951, 11351656, 11746061, 12159176, 12590901, 13041236, 13510281, 14000036, 14511501,
15045476, 15602951, 16184926, 16791401, 17422376, 18077851, 18757926, 19462601, 20192876, 20948751,
21730326, 22537601, 23370576, 24229251, 25113626, 26023701, 26959476, 27920951, 28908226, 29921301,
30960176, 32024851, 33116126, 34233901, 35378176, 36550951, 37752226, 38981451, 40238626, 41522751,
42833876, 44161001, 45520126, 46911251, 48335476, 49792701, 51282926, 52806151, 54362376, 55951601,
57573826, 59229051, 60927276, 62668501, 64452726, 66280951, 68153176, 70069401, 72020626, 74006851,
76028076, 78084301, 80175526, 82301751, 84462976, 86659201, 88890426, 91156651, 93457876, 95794101,
98165326, 100671551, 103212776, 105789001, 108400226, 111046451, 113727676, 116543901, 119396126,
122284351, 125208576, 128168801, 131165026, 134197251, 137355476, 140539701, 143849926, 147186151,
150648376, 154138601, 157659826, 161251951, 164915076, 168649201, 172454326, 176330451, 180277576,
184295701, 188384826, 192544951, 196776076, 201088201, 205470326, 209923451, 214447576, 219042701,
223708826, 228445951, 233254076, 238134201, 243086326, 248110451, 253206576, 258484701, 263844826,
269286951, 274811076, 280418201, 286107326, 291888451, 297750576, 303703701, 309647826, 315682951,
321809076, 327916201, 334104326, 340394451, 346776576, 353250701, 359816826, 366473951, 373223076,
380067201, 386997326, 394013451, 401115576, 408293701, 415547826, 422877951, 430283076, 437859201,
445506326, 453236551, 461043776, 468927901, 476888926, 484926851, 493041576, 501233001, 509511126,
517865851, 526296976, 534804501, 543388526, 552048951, 560785776, 569598901, 578488326, 587454351,
596495876, 605612901, 614806426, 624077451, 633414976, 642818901, 652299226, 661855851, 671487876,
681195301, 690977626, 700834751, 710766576, 720772901, 730853726, 740999051, 751208876, 761483101,
771821726, 782224651, 792691876, 803223301, 813819926, 824486551, 835213176, 845999701, 856846126,
867752351, 878718276, 889743801, 900828926, 911973551, 923177676, 934441301, 945764426, 957146951,
968588876, 979993101, 991457726, 1002932851, 1014418476, 1025914601, 1037431126, 1048958051, 1060495276,
1072042801, 1083600726, 1095169051, 1106747776, 1118336901, 1129926426, 1141526351, 1153136676,
1164757401, 1176388526, 1188029951, 1199781776, 1211543901, 1223316426, 1235099351, 1246892676,
1258696401, 1270510526, 1282335051, 1294169976, 1306025301, 1317890926, 1329767851, 1341655176,
1353553301, 1365461926, 1377381751, 1389312876, 1401254501, 1413206626, 1425169151, 1437142176,
1449125701, 1461119726, 1473114351, 1485129076, 1497153901, 1509188826, 1521233851, 1533288876,
1545353901, 1557428926, 1569513951, 1581608976, 1593714001, 1605829026, 1617954051, 1630089076,
1642234101, 1654389126, 1666554151, 1678729201, 1690914226, 1703109251, 1715314276, 1727529301,
1739754326, 1751989351, 1764234376, 1776489401, 1788754426, 1801029451, 1813314476, 1825609501,
1837914526, 1850229551, 1862554576, 1874889601, 1887234626, 1899579651, 1911934676, 1924299701,
1936674726, 1949059751, 1961454801, 1973859826, 1986274851, 1998699876, 2011134901, 2023579926,
2036034951, 2048499976, 2060974901, 2073469826, 2085984851, 2098509876, 2111044901, 2123589926,
2136154951, 2148729976, 2161324901, 2173909826, 2186504851, 2199109876, 2211734901, 2224379926,
2237044951, 2249719976, 2262414901, 2275099826, 2287794851, 2300409876, 2313044901, 2325699926,
2338374951, 2351069976, 2363784901, 2376419826, 2389074851, 2401739876, 2414384901, 2427059826,
2439754851, 2452379876, 2465024901, 2477699826, 2490394851, 2503079876, 2515784901, 2528409826,
2541054851, 2553699876, 2566374901, 2579079826, 2591784851, 2604459876, 2617154901, 2629809826,
2642444851, 2655059876, 2667694901, 2680359826, 2693044851, 2705699876, 2718374901, 2731049826,
2743744851, 2756399876, 2769074901, 2781729826, 2794404851, 2807099876, 2819774901, 2832449826,
2845144851, 2857809876, 2870474901, 2883149826, 2895874851, 2908549876, 2921224901, 2933899826,
2946544851, 2959199876, 2971874901, 2984549826, 2997224851, 3009899876, 3022574901, 3035249826,
3047924851, 3060599876, 3073274901, 3085949826, 3098624851, 3111299876, 3123974901, 3136649826,
3149324851, 3161999876, 3174674901, 3187349826, 3200024851, 3212699876, 3225374901, 3238049826,
3250724851, 3263399876, 3276074901, 3288749826, 3301424851, 3314099876, 3326774901, 3339449826,
3352124851, 3364799876, 3377474901, 3390149826, 3402824851, 3415499876, 3428174901, 3440849826,
3453524851, 3466199876, 3478874901, 3491549826, 3504224851, 3516899876, 3529574901, 3542249826,
3554924851, 3567599876, 3580274901, 3592949826, 3605624851, 3618299876, 3630974901, 3643649826,
3656324851, 3668999876, 3681674901, 3694349826, 3707024851, 3719699876, 3732374901, 3745049826,
3757724851, 3770399876, 3783074901, 3795749826, 3808424851, 3821099876, 3833774901, 3846449826,
3859124851, 3871799876, 3884474901, 3897149826, 3909824851, 3922499876, 3935174901, 3947849826,
3960524851, 3973199876, 3985874901, 3998549826, 4011224851, 4023899876, 4036574901, 4049249826,
4061924851, 4074599876, 4087274901, 4100049826, 4112724851, 4125399876, 4138074901, 4150749826,
4163424851, 4176099876, 4188774901, 4201449826, 4214124851, 4226799876, 4239474901, 4252149826,
4264824851, 4277499876, 4290174901, 4302849826, 4315524851, 4328199876, 4340874901, 4353549826,
4366224851, 4378899876, 4391574901, 4404249826, 4416924851, 4429599876, 4442274901, 4454949826,
4467624851, 4480299876, 4492974901, 4505649826, 4518324851, 4531099876, 4543774901, 4556449826,
4569124851, 4581799876, 4594474901, 4607149826, 4619824851, 4632499876, 4645174901, 4657849826,
4670524851, 4683199876, 4695874901, 4708549826, 4721224851, 4733899876, 4746574901, 4759249826,
4771924851, 4784599876, 4797274901, 4810049826, 4822724851, 4835399876, 4848074901, 4860749826,
4873424851, 4886099876, 4898774901, 4911449826, 4924124851, 4936799876, 4949474901, 4962149826,
4974824851, 4987499876, 5000174901, 5012849826, 5025524851, 5038199876, 5050874901, 5063549826,
5076224851, 5088899876, 5101574901, 5114249826, 5126924851, 5139599876, 5152274901, 5164949826,
5177624851, 5190299876, 5202974901, 5215649826, 5228324851, 5240999876, 5253674901, 5266349826,
5279024851, 5291699876, 5304374901, 5317049826, 5329724851, 5342399876, 5355074901, 5367749826,
5380424851, 5393099876, 5405774901, 5418449826, 5431124851, 5443799876, 5456474901, 5469149826,
5481824851, 5494499876, 5507174
```