optional 1

November 20, 2020

1 1. Write a function that inputs a number and prints the multiplication table of that number

[8]:
```
#function for printing multiplication table of input number
def multiplicationTable():
    # for taking input from user in integer format
    number = int(input())
    #for printing multiplication table for input number
    print("--Multiplication table of",number, "is--")
    for i in range(1, 11):
       print(number, 'x', i, '=', number * i )
multiplicationTable()
```

```
4
--Multiplication table of 4 is--
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
```

2 2. Write a program to print twin primes less than 1000. If two consecutive odd numbers are both prime then they are known as twin primes

[9]:
```
#for checking if a number is prime number
def isPrime(n):
   for i in range(2, n):
    if n % i == 0:
        return False
```

```
    else:
        return True
#for finding twin primes less than 1000
print("Twin Primes less than 1000 ")
for i in range(2, 1000):
     j = i + 2
     if(isPrime(i) and isPrime(j)):
        print("{:d} and {:d}".format(i, j))
```

```
Twin Primes less than 1000
3 and 5
5 and 7
7 and 9
9 and 11
11 and 13
13 and 15
15 and 17
17 and 19
19 and 21
21 and 23
23 and 25
25 and 27
27 and 29
29 and 31
31 and 33
33 and 35
35 and 37
37 and 39
39 and 41
41 and 43
43 and 45
45 and 47
47 and 49
49 and 51
51 and 53
53 and 55
55 and 57
57 and 59
59 and 61
61 and 63
63 and 65
65 and 67
67 and 69
69 and 71
71 and 73
73 and 75
75 and 77
```

77 and 79
79 and 81
81 and 83
83 and 85
85 and 87
87 and 89
89 and 91
91 and 93
93 and 95
95 and 97
97 and 99
99 and 101
101 and 103
103 and 105
105 and 107
107 and 109
109 and 111
111 and 113
113 and 115
115 and 117
117 and 119
119 and 121
121 and 123
123 and 125
125 and 127
127 and 129
129 and 131
131 and 133
133 and 135
135 and 137
137 and 139
139 and 141
141 and 143
143 and 145
145 and 147
147 and 149
149 and 151
151 and 153
153 and 155
155 and 157
157 and 159
159 and 161
161 and 163
163 and 165
165 and 167
167 and 169
169 and 171
171 and 173

173 and 175
175 and 177
177 and 179
179 and 181
181 and 183
183 and 185
185 and 187
187 and 189
189 and 191
191 and 193
193 and 195
195 and 197
197 and 199
199 and 201
201 and 203
203 and 205
205 and 207
207 and 209
209 and 211
211 and 213
213 and 215
215 and 217
217 and 219
219 and 221
221 and 223
223 and 225
225 and 227
227 and 229
229 and 231
231 and 233
233 and 235
235 and 237
237 and 239
239 and 241
241 and 243
243 and 245
245 and 247
247 and 249
249 and 251
251 and 253
253 and 255
255 and 257
257 and 259
259 and 261
261 and 263
263 and 265
265 and 267
267 and 269

269 and 271
271 and 273
273 and 275
275 and 277
277 and 279
279 and 281
281 and 283
283 and 285
285 and 287
287 and 289
289 and 291
291 and 293
293 and 295
295 and 297
297 and 299
299 and 301
301 and 303
303 and 305
305 and 307
307 and 309
309 and 311
311 and 313
313 and 315
315 and 317
317 and 319
319 and 321
321 and 323
323 and 325
325 and 327
327 and 329
329 and 331
331 and 333
333 and 335
335 and 337
337 and 339
339 and 341
341 and 343
343 and 345
345 and 347
347 and 349
349 and 351
351 and 353
353 and 355
355 and 357
357 and 359
359 and 361
361 and 363
363 and 365

365 and 367
367 and 369
369 and 371
371 and 373
373 and 375
375 and 377
377 and 379
379 and 381
381 and 383
383 and 385
385 and 387
387 and 389
389 and 391
391 and 393
393 and 395
395 and 397
397 and 399
399 and 401
401 and 403
403 and 405
405 and 407
407 and 409
409 and 411
411 and 413
413 and 415
415 and 417
417 and 419
419 and 421
421 and 423
423 and 425
425 and 427
427 and 429
429 and 431
431 and 433
433 and 435
435 and 437
437 and 439
439 and 441
441 and 443
443 and 445
445 and 447
447 and 449
449 and 451
451 and 453
453 and 455
455 and 457
457 and 459
459 and 461

461 and 463
463 and 465
465 and 467
467 and 469
469 and 471
471 and 473
473 and 475
475 and 477
477 and 479
479 and 481
481 and 483
483 and 485
485 and 487
487 and 489
489 and 491
491 and 493
493 and 495
495 and 497
497 and 499
499 and 501
501 and 503
503 and 505
505 and 507
507 and 509
509 and 511
511 and 513
513 and 515
515 and 517
517 and 519
519 and 521
521 and 523
523 and 525
525 and 527
527 and 529
529 and 531
531 and 533
533 and 535
535 and 537
537 and 539
539 and 541
541 and 543
543 and 545
545 and 547
547 and 549
549 and 551
551 and 553
553 and 555
555 and 557

557 and 559
559 and 561
561 and 563
563 and 565
565 and 567
567 and 569
569 and 571
571 and 573
573 and 575
575 and 577
577 and 579
579 and 581
581 and 583
583 and 585
585 and 587
587 and 589
589 and 591
591 and 593
593 and 595
595 and 597
597 and 599
599 and 601
601 and 603
603 and 605
605 and 607
607 and 609
609 and 611
611 and 613
613 and 615
615 and 617
617 and 619
619 and 621
621 and 623
623 and 625
625 and 627
627 and 629
629 and 631
631 and 633
633 and 635
635 and 637
637 and 639
639 and 641
641 and 643
643 and 645
645 and 647
647 and 649
649 and 651
651 and 653

653 and 655
655 and 657
657 and 659
659 and 661
661 and 663
663 and 665
665 and 667
667 and 669
669 and 671
671 and 673
673 and 675
675 and 677
677 and 679
679 and 681
681 and 683
683 and 685
685 and 687
687 and 689
689 and 691
691 and 693
693 and 695
695 and 697
697 and 699
699 and 701
701 and 703
703 and 705
705 and 707
707 and 709
709 and 711
711 and 713
713 and 715
715 and 717
717 and 719
719 and 721
721 and 723
723 and 725
725 and 727
727 and 729
729 and 731
731 and 733
733 and 735
735 and 737
737 and 739
739 and 741
741 and 743
743 and 745
745 and 747
747 and 749

749 and 751
751 and 753
753 and 755
755 and 757
757 and 759
759 and 761
761 and 763
763 and 765
765 and 767
767 and 769
769 and 771
771 and 773
773 and 775
775 and 777
777 and 779
779 and 781
781 and 783
783 and 785
785 and 787
787 and 789
789 and 791
791 and 793
793 and 795
795 and 797
797 and 799
799 and 801
801 and 803
803 and 805
805 and 807
807 and 809
809 and 811
811 and 813
813 and 815
815 and 817
817 and 819
819 and 821
821 and 823
823 and 825
825 and 827
827 and 829
829 and 831
831 and 833
833 and 835
835 and 837
837 and 839
839 and 841
841 and 843
843 and 845

845 and 847
847 and 849
849 and 851
851 and 853
853 and 855
855 and 857
857 and 859
859 and 861
861 and 863
863 and 865
865 and 867
867 and 869
869 and 871
871 and 873
873 and 875
875 and 877
877 and 879
879 and 881
881 and 883
883 and 885
885 and 887
887 and 889
889 and 891
891 and 893
893 and 895
895 and 897
897 and 899
899 and 901
901 and 903
903 and 905
905 and 907
907 and 909
909 and 911
911 and 913
913 and 915
915 and 917
917 and 919
919 and 921
921 and 923
923 and 925
925 and 927
927 and 929
929 and 931
931 and 933
933 and 935
935 and 937
937 and 939
939 and 941

```
941 and 943
943 and 945
945 and 947
947 and 949
949 and 951
951 and 953
953 and 955
955 and 957
957 and 959
959 and 961
961 and 963
963 and 965
965 and 967
967 and 969
969 and 971
971 and 973
973 and 975
975 and 977
977 and 979
979 and 981
981 and 983
983 and 985
985 and 987
987 and 989
989 and 991
991 and 993
993 and 995
995 and 997
997 and 999
999 and 1001
```

3 3. Write a program to find out the prime factors of a number. Example: prime factors of 56 :- 2, 2, 2, 7

[10]:
```
#function for printing prime factors of a number
def prime_factors(n):
    while n % 2 == 0:
        print (2),
        n = n / 2
    for i in range(3,int(n ** 0.5)+1,2):
        while n % i== 0:
            print (i),
            n = n / i
    if n > 2:
        print (n)
```

```
m = int(input()) #For taking input from user
print("Prime Factors of",m,"are - ")
prime_factors(m)
```

```
56
Prime Factors of 56 are -
2
2
2
7.0
```

4 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!

[11]:
```
#function for returning factorial of a number
def factorial(n):
    fact = 1;
    for i in range(2,n+1):
        fact = fact * i;
    return fact;
#function for returning Number of permutations of n objects taken r at a time
def nPr(n, r):
    pnr = factorial(n) / factorial(n - r);
    return pnr;
#function for returning Number of combinations of n objects taken r at a time
def nCr(n, r):
    return (factorial(n) / (factorial(r) * factorial(n - r)))

print("Number of permutations of 8 objects, taken 5 at a time is ",nPr(8, 5))
print("Number of combinations of 8 objects, taken 5 at a time is ",nCr(8, 5))
```

```
Number of permutations of 8 objects, taken 5 at a time is  6720.0
Number of combinations of 8 objects, taken 5 at a time is  56.0
```

5 5. Write a function that converts a decimal number to binary number

[12]:
```
def decimalToBinary(n):
    binaryNum = [0] * n
    i = 0
    while (n > 0):
        binaryNum[i] = n % 2
        n = int(n / 2)
```

```
        i += 1
    for j in range(i - 1, -1, -1):
        print(binaryNum[j], end = "")
print("Binary number of 45 is - ",end="")
decimalToBinary(45)
```

```
Binary number of 45 is - 101101
```

6 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

[13]:
```
#function to add cube of digits
def cubesum(n):
    temp = n
    sum = 0
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
    return sum

def isArmstrong(x):
    if (x == cubesum(x)):
        return True
    else:
        return False

def PrintArmstrong(min,max):
    for i in range(min,max):
        if(isArmstrong(i)):
            print(i)
print("Armstrong number in range 1-1000 are :-")
PrintArmstrong(1,1000)
```

```
Armstrong number in range 1-1000 are :-
1
153
370
371
407
```

7 7. Write a function prodDigits() that inputs a number and returns the product of digits of that number.

[14]:
```
# Function to get product of digits
def prodDigits(n):
    product = 1
    while (n != 0):
        product = product * (n % 10)
        n = n // 10
    return product
print("Product of digits of number 4671 is :-",prodDigits(4671))
```

```
Product of digits of number 4671 is :- 168
```

8 8. If all digits of a number n are multiplied by each other repeating with the product, the one digit number obtained at last is called the multiplicative digital root of n. The number of times digits need to be multiplied to reach one digit is called the multiplicative persistance of n. Example: 86 -> 48 -> 32 -> 6 (MDR 6, MPersistence 3) 341 -> 12->2 (MDR 2, MPersistence 2) 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

[15]:
```
def MDR(n):
    temp = str(n)
    while len(temp) > 1:
        temp = str(prodDigits(int(temp)))
    return int(temp)
print("MDR of 86 : ",MDR(86))
print("MDR of 341 : ",MDR(341))
def MPersistence(n):
    temp = str(n)
    pers = 0
    while len(temp) > 1:
        temp = str(prodDigits(int(temp)))
        pers += 1
    return pers
print("MPersistence of 86 : ",MPersistence(86))
print("MPersistence of 341 : ",MPersistence(341))
```

```
MDR of 86 :  6
MDR of 341 :  2
MPersistence of 86 :  3
```

```
MPersistence of 341 :  2
```

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, 18

[16]:
```
def sumPdivisors(n) :
    sum = 0
    for i in range(2, int(n**0.5)+ 1) :
        if (n % i == 0) :
            if (i == int(n / i)) :
                sum = sum + i
            else :
                sum = sum + (i + int(n / i))
    return (sum + 1)
print("sum of proper divisors of 36 is :",sumPdivisors(36))
```

```
sum of proper divisors of 36 is : 55
```

10 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

[17]:
```
def perfectNumbers(min,max):
    for i in range(min,max):
        if(i == sumPdivisors(i)):
            print(i)
print("Perfect numbers in range 1-1000 are :")
perfectNumbers(1,1000)
```

```
Perfect numbers in range 1-1000 are :
1
6
28
496
```

11 11. Two different numbers are called amicable numbers if the sum of the proper divisors of each is equal to the other number. For example 220 and 284 are amicable numbers.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 + 71 + 142 = 220 . Write a function to print pairs of amicable numbers in a range

[18]:
```
def amicablePairs(x, y) :

    if (sumPdivisors(x) != y) :
        return False

    return (sumPdivisors(y) == x)

def amicableNumbers(min,max):
    for i in range(min,max):
        for j in range(i+1,max):
            if(amicablePairs(i,j)):
                print(i,j)

print("Amicable number in range 1-2000 :" )
amicableNumbers(1,2000)
```

```
Amicable number in range 1-2000 :
220 284
1184 1210
```

12 12. Write a program which can filter odd numbers in a list by using filter function

[19]:
```
list1 = [101,10,1,24,2,3,4,5,9,13,22,17,99]
def Odd(x):
    if x % 2 != 0:
        return True
    else:
        return False

odd_list = filter(Odd , list1)
for x in odd_list:
    print(x)
```

```
101
1
3
5
```

```
9
13
17
99
```

13 13. Write a program which can map() to make a list whose elements are cube of elements in a given list

[20]:
```
list2 = [1,2,3,4,5,6]
def cube(x):
    return x*x*x
cube_list = map(cube, list2)
for x in cube_list:
    print(x)
```

```
1
8
27
64
125
216
```

14 14. Write a program which can map() and filter() to make a list whose elements are cube of even number in a given list

[21]:
```
list3 = [101,10,1,24,2,3,4,5,9,13,22,17,99]

def Even(x):
    if x % 2 == 0:
        return True
    else:
        return False

even_list = filter(Even, list3)
even_cube_list = map(cube, even_list) #using cube function made in problem 13

for x in even_cube_list:
    print(x)
```

```
1000
13824
8
64
10648
```

[ ]: