1.Python_Optinal_Questions

December 28, 2023

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

[3]:
```
def table(num):
    for i in range(1, 11):
        print(num * i)
```

[4]:
```
table(5)
```

```
5
10
15
20
25
30
35
40
45
50
```

0.2 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

[49]:
```
def twin_prime(num):
    prevPrime = 3
    for num in range(3, 1000):
        if num % 2 == 1:
            flag = True
            for i in range(3, int(num ** 0.5)+1):
                if num % i == 0:
                    flag = False
                    break
            if flag:
                if num - prevPrime == 2:
                    print('Twin Primes:', prevPrime, num)
                prevPrime = num
```

[50]:
```
twin_prime(1000)
```

```
Twin Primes: 3 5
Twin Primes: 5 7
Twin Primes: 11 13
Twin Primes: 17 19
Twin Primes: 29 31
Twin Primes: 41 43
Twin Primes: 59 61
Twin Primes: 71 73
Twin Primes: 101 103
Twin Primes: 107 109
Twin Primes: 137 139
Twin Primes: 149 151
Twin Primes: 179 181
Twin Primes: 191 193
Twin Primes: 197 199
Twin Primes: 227 229
Twin Primes: 239 241
Twin Primes: 269 271
Twin Primes: 281 283
Twin Primes: 311 313
Twin Primes: 347 349
Twin Primes: 419 421
Twin Primes: 431 433
Twin Primes: 461 463
Twin Primes: 521 523
Twin Primes: 569 571
Twin Primes: 599 601
Twin Primes: 617 619
Twin Primes: 641 643
Twin Primes: 659 661
Twin Primes: 809 811
Twin Primes: 821 823
Twin Primes: 827 829
Twin Primes: 857 859
Twin Primes: 881 883
```

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

[86]:
```
def primeFactors(num):
    i = 2
    while(i <= num):
        if num % i == 0:
            print(i)
            num = num / i
            i = 1
        i += 1
```

[91]:
```
primeFactors(60)
```

```
2
2
3
5
```

0.3.1 Q4. 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!

[27]:
```
def factorial(num):
    result = 1
    for i in range(1, num+1):
        result *= i
    return result

def permutation(num, r):
    return factorial(num)/factorial(num - r)

def combination(num, r):
    return permutation(num, r) / factorial(r)
```

[30]:
```
print(permutation(5, 2))
print(combination(5, 2))
```

```
20.0
10.0
```

0.3.2 Q5. Write a function that converts a decimal number to binary number

[59]:
```
def dectobinary(num):
    result = ''
    while(num):
        if num == 1 or num == 0:
            result += str(num)
            return int(result[::-1])
        result += str(num % 2)
        num //= 2
    return int(result[::-1])
```

[62]:
```
dectobinary(100)
```

[62]:
```
1100100
```

0.3.3 Q6. 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.

[71]:
```
def cubesum(num):
    result = 0
    while(num > 0):
        rem = num % 10
        result += rem ** 3
        num //= 10
    return result

def PrintArmstrong(num1, num2):
    for num in range(num1, num2+1):
        if isArmstrong(num):
            print(num)

def isArmstrong(num):
    result = cubesum(num)
    if result == num:
        return True
    return False
```

[75]:
```
PrintArmstrong(100, 400)
```

```
153
370
371
```

0.3.4 Q7. Write a function prodDigits() that inputs a number and returns the product of digits of that number.

[17]:
```
def prodDigits(num):
    result = 1
    n = len(str(num))
    while(n):
        result *= (num % 10)
        num //= 10
        n -= 1
    return result
```

```
[20]: print(prodDigits(123456789))
```

```
362880
```

0.3.5 Q8. 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

```
[81]: def MPersistence(num):
          count = 0
          while(True):
              num = prodDigits(num)
              count += 1
              if len(str(num)) == 1:
                  return num, count
```

```
[84]: MPersistence(340)
```

```
[84]: (0, 1)
```

0.3.6 Q9. 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

```
[106]: def sumPdivisors(num):
           result = 0
           for val in range(1, num//2 + 1):
               if num % val == 0:
                   result += val
           return result
```

```
[130]: sumPdivisors(92)
```

```
[130]: 76
```

0.3.7 Q10. 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

[108]:
```
def PerfectNumbers(num1, num2):
    for val in range(num1, num2+1):
        if val == sumPdivisors(val):
            print(val)
```

[111]:
```
PerfectNumbers(1, 1000)
```

```
6
28
496
```

0.3.8 Q11. 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
```

[113]:
```
def isAmicable(num1, num2):
    if num1 == sumPdivisors(num2):
        return True
    else:
        return False
```

[131]:
```
def printAmicable(num1, num2):
    for val1 in range(num1, num2+1):
        for val2 in range(val1+1, num2+1):
            if isAmicable(val1, val2):
                print(val2, val1)
```

[134]:
```
printAmicable(2, 10)
```

```
4 3
9 4
8 7
10 8
```

0.3.9 Q12. Write a program which can filter odd numbers in a list by using filter function

[152]:
```
def filterodd(num):
    if num % 2 != 0:
        return True
nums = list(range(11))
odds = filter(filterodd, nums)
print(*odds)
```

```
1 3 5 7 9
```

0.3.10 Q13. Write a program which can map() to make a list whose elements are cube of elements in a given list

[145]:
```
def cube(num):
    return num ** 3

nums = list(range(10))
output = map(cube, nums)
print(*output)
```

```
0 1 8 27 64 125 216 343 512 729
```

0.3.11 Q14. Write a program which can map() and filter() to make a list whose elements are cube of even number in a given list

[154]:
```
def filterEven(num):
    if num % 2 == 0:
        return True
```

[161]:
```
def cubeofevenNo(nums):
#     result = map(lambda x: x**3, filter(filterEven, nums)) # using lambda
    result = map(cube, filter(filterEven, nums)) # using function
    return result
nums = list(range(11))
print(*cubeofevenNo(nums))
```

```
0 8 64 216 512 1000
```

[ ]:
```

```