1. Write a Python function is_multiple(n, m) that takes two integer values and returns True if n is a multiple of m, i.e., n=mi, for some integer i, and False otherwise.

2. Write a Python function is_even(n) that takes an integer value and returns True if n is even, and False otherwise. Your function can't use *, //, or % operators.

3. Write a Python function sum_to(n) that generates and returns the sum of all positive integer numbers up to and including n. So sum_to(10) would be 1+2+3...+10 which would return the value 55. Validate your input as it neither negative number nor a real number, i.e., sum_to(-10) and sum_to(10.5) are invalid input. Use the standard formula to finds the sum $1 + \cdots + n = n(n+1)/2$ and compare running time of your program. Write a recursive function for to generate and find the sum of all positive integer numbers up to and including n. What do you conclude?

4. Write a Python function square_sum_to(n) that generates and returns the sum of all positive integer numbers up to and including n.
$1^2 + 2^2 + 3^2 + \cdots + n^2 = n(n+1)(2n+1)/6$ .

5. Write a Python function that reads an integer and returns the sum of its digits.

6. Write a Python function largest_fibo(n) that returns the largest Fibonacci number less than a positive integer number 'n' passed to it.

7. Write a Python function prime_factors(n) that returns only those factors of n that are prime.

8. Write a Python function prime_count(n) that returns the number of primes less than a positive integer number 'n' passed to it.

9. Write a Python function smallest_prime_larger_than(x) that returns the smallest primes larger than a positive composite integer number passed to it. E.g., if x=100, then output is 101. If you input x as any prime number then it should returns x itself.

10. Write recursive functions in Python for the followings

(a) compute the product of two positive integers, *m* and *n*
(b) Compute $x^y$
(c) Check whether your input integer is palindrome
(d) Compute GCD of two positive integers x and y
(e) Find sum of digits of a given positive integer
(f) Convert a decimal number into binary number

11. Let d(*n*) be defined as the sum of proper divisors of *n* (numbers less than *n* which divide evenly into *n*). If d(*a*) = *b* and d(*b*) = *a*, where $a \neq b$, then *a* and *b* are an amicable pair and each of *a* and *b* are called amicable numbers.
For example, the proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110; therefore d(220) = 284. The proper divisors of 284 are 1, 2, 4, 71 and 142; so d(284) = 220.
Write a Python function is_amicable(a, b) that return 1 a and b are amicable pair, and write a complete program to find all the amicable numbers in the range 1-10000.

12. A polite(or staircase) number is a positive integer that can be written as the sum of two or more consecutive positive integers. Rest of the integers are said to be impolite or rude.
The first few polite numbers are
3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, ...
where as 4, 8, 16, 32, 64,... are impolite numbers. The impolite numbers are exactly the powers of two.
Say, 7, 9 and 18 are a polite numbers as 7 = 3+4, 9 = 2+3+4(as well as 4+5) and 18=5+6+7(as well as 3+4+5+6). Write a Python function polite_num(n), which takes a positive integer(n>2) and prints various ways in which it can be represented as a sum of consecutive positive integers. If the number is not polite, output the message "Impolite number".
Each possible representation should be printed in a new line.
For e.g.

i. output for polite_num(10) should be
1 2 3 4
ii. output for polite_num(9) should be
2 3 4
4 5
iii. output for polite_num(15) should be
4 + 5 + 6
1 + 2 + 3 + 4 + 5
7 + 8
iv. output for polite_num(42) should be

3 4 5 6 7 8 9
9 10 11 12
13 14 15

v. output for polite_num(32) should be
32 is Impolite number

13. Twin primes (consecutive odd primes differ by 2) ,say, (3, 5), (5, 7), (11, 13) (17, 19), ….. Write a Python function is_prime(n) that returns 1 if n is prime, returns 0 otherwise. Use this function
(a) Write a program to print Twin primes in the range 100-200.
(b) Write a program to print first 50 Twin primes.
Check your output with
(101, 103), (107, 109), (137, 139), (149, 151), (179, 181), (191, 193), (197, 199)

14. Goldback's conjucture - It was proposed by German mathematician Christian Goldbach that every even integer greater than 4 can be expressed as the sum of two odd primes
6 = 3 + 3
8 = 3 + 5
10 = 3 + 7 = 5 + 5
12 = 7 + 5
...
30=7+23, 11+19, 13+17
100 = 3 + 97 = 11 + 89 = 17 + 83 = 29 + 71 = 41 + 59 = 47 + 53
…
200 = 3 + 197 = 7 + 193 = 19 + 181 = 37 + 163 = 43 + 157 = 61 + 139 = 73 + 127 = 97 + 103
Write a Python function is_prime(n) that returns 1 if n is prime, returns 0 otherwise. Use this function input an integer from the user and express it as sum of two odd primes.

15. Create a function to reverse a number. Use this to create a function named differev that will return the difference of the reversed number from the original number (d=N-R). For example, for N=514 reversed number is 415 and the difference is 99. Verify that differev(N) + rev(N) = N

16. The **Catalan numbers** form a sequence of natural numbers that occur in various central counting problems.
Catalan numbers can be defined directly

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} = \prod_{k=2}^{n} \frac{n+k}{k} \qquad \text{for } n \geq 0.$$

Or recursively:

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \sum_{i=0}^{n} C_i\, C_{n-i} \quad \text{for } n \geq 0;$$

Or alternatively (also recursive):

$$C_0 = 1 \quad \text{and} \quad C_n = \frac{2(2n-1)}{n+1} C_{n-1},$$

The first few Catalan numbers for $n = 0, 1, 2, 3, \ldots$ are

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440 , 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452,

Write a Python function that takes a non-negative integer n and find $n^{th}$ Catalan Number. Call your program by
catalan(20)
ctalan(30)
catalan(50)
Write a Python function that takes a non-negative integer n and generate print out the first 20 Catalan numbers. Observe that your program takes long time to generate Catalan numbers because the value of $n^{th}$ Catalan number is exponential that makes the time complexity exponential. Compare the running time of above 3 formula.