1. Write a program to check if a given string is valid or not, the string contains the characters '(', ')', '{', '}', '[' and ']'. The string is valid if the open brackets must be closed by the same type of brackets and in correct order.
   Solution:

```
class py_solution:
  def is_valid_parenthese(self, str1):
      stack, pchar = [], {"(": ")", "{": "}", "[": "]"}
      for parenthese in str1:
         if parenthese in pchar:
            stack.append(parenthese)
         elif len(stack) == 0 or pchar[stack.pop()] != parenthese:
            return False
      return len(stack) == 0
print(py_solution().is_valid_parenthese("(){}[]"))
print(py_solution().is_valid_parenthese("()[{)}"))
print(py_solution().is_valid_parenthese("()"))
```

2. Write a program to generate all combinations of well-formed parentheses from n given pairs of parentheses.
   Solution:

```
def generateParenthesis(n, Open, close, s, ans):
        if(Open == n and close == n):
                ans.append(s)
                return
        if(Open < n):
                generateParenthesis(n, Open+1, close, s+"(", ans)
        if(close < Open):
                generateParenthesis(n, Open, close + 1, s+")", ans)
n = 10
ans = []
generateParenthesis(n, 0, 0, "", ans)
for s in ans:
        print(s)
```

3. Write a program to remove the duplicates from a given array of integers.
   Solution:

```
def Remove(duplicate):
        final_list = []
        for num in duplicate:
                if num not in final_list:
                        final_list.append(num)
        return final_list
duplicate = [2, 4, 10, 20, 5, 2, 20, 4]
print(Remove(duplicate))
```

4. Write a program to remove all instances of a given value in a given array of integers and return the length of the new array.
   Solution:

```
def remove_element(array_nums, val):
   i = 0
   while i < len(array_nums):
      if array_nums[i] == val:
         array_nums.remove(array_nums[i])
      else:
         i += 1
   return len(array_nums)
print(remove_element([1, 2, 3, 4, 5, 6, 7, 5], 5))
print(remove_element([10,10,10,10,10], 10))
print(remove_element([10,10,10,10,10], 20))
print(remove_element([], 1))
```

5. Write a program to find the index of the first occurrence of a given string within another given string. If not found return -1.
   Solution:

```
def find_Index(str1, pos):
   if len(pos) > len(str1):
      return 'Not found'
   for i in range(len(str1)):
      for j in range(len(pos)):
         if str1[i + j] == pos[j] and j == len(pos) - 1:
            return i
         elif str1[i + j] != pos[j]:
            break
   return 'Not found'
```

```
print(find_Index("Python Exercises", "Ex"))
print(find_Index("Python Exercises", "yt"))
print(find_Index("Python Exercises", "PY"))
```

6. Write a program to divide two given integers without using multiplication, division and mod operator. Return the quotient after dividing.
   Solution:

```
def Divide(a, b):
    dividend = a;
    divisor = b;
    sign = -1 if ((dividend < 0) ^(divisor < 0)) else 1;
    dividend = abs(dividend);
   divisor = abs(divisor);
   if (divisor == 0):
       print("Cannot Divide by 0");
   if (dividend == 0):
        print(a, "/", b, "is equal to :", 0);
   if (divisor == 1):
         print(a, "/", b, "is equal to :",(sign * dividend));
         print(a, "/", b, "is equal to :",math.floor(sign * math.exp
(math.log(dividend) -math.log(divisor))));
a = 10;
b = 5;
Divide(a, b);
a = 49;
b = -7;
Divide(a, b);
```

7. Write a program to find the length of the longest valid (correct-formed) parentheses substring of a given string.
   Solution:

```
def findMaxLen(string):
        n = len(string)
        stk = []
        stk.append(-1)
```

```
        result = 0
        for i in range(n):
                if string[i] == '(':
                        stk.append(i)
                                else:
                        if len(stk) != 0:
                        stk.pop()
                        if len(stk) != 0:
                                result = max(result,i - stk[len(stk)-1])
                        else:
                                stk.append(i)
        return result
string = "((()()"
print (findMaxLen(string))
string = "()(()))))"
```

8. Write a programming to find the sum of all the multiples of 3 or 7 below 100.
   Solution:

```
n = 0
for i in range(1,500):
    if not i % 5 or not i % 3:
       n = n + i
print(n)
```

9. Write a programming to find the sum of the even-valued terms from the terms in the Fibonacci sequence whose values do not exceed one million.
   Solution:

```
cache = {}
def fiba(n):
    cache[n] = cache.get(n, 0) or (n <= 1 and 1 or fiba(n-1) + fiba(n-2))
    return cache[n]
n = 0
x = 0
while fiba(x) <= 1000000:
     if not fiba(x) % 2: n = n + fiba(x)
     x=x+1
print(n)
```

10. Write a programming to find the largest prime factor of the number 438927456?
    Solution:

```
def Largest_Prime_Factor(n):
    return next(n // i for i in range(1, n) if n % i == 0 and is_prime(n // i))
def is_prime(m):
    return all(m % i for i in range(2, m - 1))
print(Largest_Prime_Factor(200))
print(Largest_Prime_Factor(330))
print(Largest_Prime_Factor(243423423330))
```

11. Write a programming to find the largest palindrome made from the product of two 3-digit numbers.
    Solution:

```
n = 0
for a in range(9999, 100, -1):
    for b in range(a, 100, -1):
        x = a * b
        if x > n:
            s = str(a * b)
            if s == s[::-1]:
                n = a * b
print(n)
```

12. Write a programming to find the smallest positive number that is evenly divisible by all of the numbers from 1 to 20?
    Solution:

```
def gcd(x,y): return y and gcd(y, x % y) or x
def lcm(x,y): return x * y / gcd(x,y)
n = 1
for i in range(1, 31):
    n = lcm(n, i)
print(n)
```

13. Write a programming to find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.

Solution:

```
r = range(1, 201)
a = sum(r)
print (a * a - sum(i*i for i in r))
```

14. Write a python program to get the 1001st prime number?

Solution:

```
def isPrime(n):
        if n <= 1:
                return False
        for i in range(2, n):
                if n % i == 0:
                        return False;
        return True
print("true") if isPrime(11) else print("false")
print("true") if isPrime(14) else print("false")
```

15. Write a programming to find the product xyz.

Solution:

```
for a in range(1, 1000):
    for b in range(a, 1000):
        c = 1000 - a - b
        if c > 0:
            if c*c == a*a + b*b:
                print (a*b*c)
                break
```

16. Write a programming to find the sum of all the primes below ten thousand.

Solution:

```
MAX = 105000
print("Input a number (n≤10000) to compute the sum:(0 to exit)")
is_prime = [True for _ in range(MAX)]
is_prime[0] = is_prime[1] = False
for i in range(2, int(MAX ** (1 / 2)) + 1):
  if is_prime[i]:
```

```
    for j in range(i ** 2, MAX, i):
      is_prime[j] = False
primes = [i for i in range(MAX) if is_prime[i]]
while True:
  n = int(input())
  if not n:
    break
  print("Sum of first",n,"prime numbers:")
  print(sum(primes[:n]))
```

17. Write a programming to find the first missing positive integer from a given unsorted integer array.
Solution:

```
class Solution(object):
    def firstMissingPositive(self, nums):
        i = 0
        nums = [0] + nums
        for i in range(len(nums)):
            while nums[i]>=0 and nums[i]<len(nums) and
nums[nums[i]]!=nums[i]:
                nums[nums[i]],nums[i] = nums[i],nums[nums[i]]
        num = 1
        for i in range(1,len(nums)):
            if num == nums[i]:
                num+=1
        return num
ob = Solution()
print(ob.firstMissingPositive([4,-3,1,-1]))
```

18. Write a program to find the minimum number of characters that must be inserted into a given string with no whitespace characters to make it a palindrome.
Solution:

```
import sys
def findMinInsertions(str, l, h):
        if (l > h):
                return sys.maxsize
```

```
        if (l == h):
                return 0
        if (l == h - 1):
                return 0 if(str[l] == str[h]) else 1
        if(str[l] == str[h]):
                return findMinInsertions(str, l + 1, h - 1)
        else:
                return (min(findMinInsertions(str, l, h - 1),
                                        findMinInsertions(str, l + 1, h)) + 1)
if __name__ == "__main__":
        str = "geeks"
        print(findMinInsertions(str, 0, len(str) - 1))
```

19. Write a program to create and display all prime numbers in strictly ascending decimal digit order.
    Solution:

```
lower = 900
upper = 1000
print("Prime numbers between", lower, "and", upper, "are:")
for num in range(lower, upper + 1):
  if num > 1:
     for i in range(2, num):
        if (num % i) == 0:
           break
     else:
        print(num)
```

20. Write a program (subroutine, function, procedure, whatever it may be called in your language) to test if a number is a colorful number or not.
    Solution:

```
def my_func():
        x = 10
        print("Value inside function:",x)
x = 20
my_func()
print("Value outside function:",x)
```