```python
def is_anagram(str1, str2):
    if len(str1)!=len(str2):
        return False

    char_count = [0] * 26
    # count characters in both strings
    for i in range(len(str1)):
        char_count[ord(str1[i]) - ord('a')]+=1
        char_count[ord(str2[i]) - ord('a')]-=1

    for count in char_count:
        if count!=0:
            return False, str1 + " and " + str2 + " are not anagram."

    return True, str1 + " and " + str2 + " are anagram."

# Example usage
print(is_anagram("listen", "silent"))
print(is_anagram("hello", "world"))
```

```
(True, 'listen and silent are anagram.')
(False, 'hello and world are not anagram.')
```

```python
def anagram(str1, str2):
    #convert both the strings into lowercase
    str1 = str1.lower()
    str2 = str2.lower()
    # check if length is same
    if len(str1) == len(str2):
        # sort the string
        sorted_str1 = sorted(str1)
        sorted_str2 = sorted(str2)

        # if sorted char arrays are same
        if (sorted_str1 == sorted_str2):
            print(str1 + " and " + str2 + " are anagram.")
        else:
            print(str1 + " and " + str2 + " are not anagram.")
    else:
        print(str1 + " and " + str2 + "are not anagram.")

anagram(str1 = "Race", str2 = "Care")
```

```
race and care are anagram.
```

```python
def ascending_order(list1):
    print("Given list:", list1)
    for i in range(0, len(list1)):
        for j in range(0, len(list1)-i-1):
            if list1[j] > list1[j+1]:
                list1[j],list1[j+1]=list1[j+1],list1[j]
    return list1
```

```
print(ascending_order(list1=[3,2,1,6,5,8,7]))
```

```
Given list: [3, 2, 1, 6, 5, 8, 7]
[1, 2, 3, 5, 6, 7, 8]
```

```
def decending_order(list1):
    print("Given list:", list1)
    for i in range(0, len(list1)):
        for j in range(0, len(list1)-i-1):
            if list1[j]<list1[j+1]:
                list1[j],list1[j+1]=list1[j+1],list1[j]
    return list1

print(decending_order(list1=[2,1,4,3,6,5,7,9,8]))
```

```
Given list: [2, 1, 4, 3, 6, 5, 7, 9, 8]
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
def even_odd(numbers):
    even = []
    odd = []
    if numbers == 0:
        print("Please enter a valid numbers:")
    elif numbers:
        for i in numbers:
            if i%2 == 0:
                even.append(i)
            else:
                odd.append(i)
        return even, odd

numbers = [1,2,3,4,5,6,7,8,9,10]
even, odd = even_odd(numbers)
print("Even numbers:", even)
print("Odd numbers:", odd)
```

```
Even numbers: [2, 4, 6, 8, 10]
Odd numbers: [1, 3, 5, 7, 9]
```

```
def factorial(n):
    if n < 0:
        print("Enter a valid number:", n)
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        fact = 1
        for i in range(1, n+1):
            fact = fact * i
            print(fact)

factorial(-1)
print(factorial(0))
print(factorial(1))
```

```
factorial(6)
```

```
Enter a valid number: -1
0
1
1
2
6
24
120
720
```

```
def fibonnic_series(n):
    """
    Fibonnic series will return the sum of two values
    (second value + first value) in the series of numbers
    Paramaeters: a, b are variables
    n is actual number to calculate the fibbonic series
    return: 0, 1, 1, 2, 3, 5, 8
    """
    a, b = 0, 1
    if n < 0:
        print("You have provided negative number:", n)
        print("Please enter the valid number:")
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        for i in range(2, n+1):
            c = a + b
            a = b
            b = c
            print(c)

user_input = int(input("Enter a number:"))
n=user_input
fibonnic_series(n)
```

```
Enter a number: 10
```

```
1
2
3
5
8
13
21
34
55
```

```
# How to reverse the string
def reverse_string(str1):
    return str1[::-1]
```

```python
print(reverse_string(str1="helloworld"))
```

```
dlrowolleh
```

```python
def replace_string(str1):
    print("Given string:", str1)
    old_string = "hi"
    replace_string = "Hello"
    replaced_string = str1.replace(old_string, replace_string)
    return replaced_string

print(replace_string(str1="hi world"))
```

```
Given string: hi world
Hello world
```

```python
import copy

def shallow_copy(list1, list2):
    print("list1", list1)
    print("list2", list2)
    print("id of list1", id(list1))
    print("id of list2", id(list2))
    list2 =  list1.copy()
    print("id of list2 after copy()", id(list2))

def deep_copy(list1, list2, list3):
    print("list1", list1)
    print("list2", list2)
    print("id of list1", id(list1))
    print("id of list2", id(list2))
    list2 =  list1.copy()
    print("id of list2", id(list2))
    print("list3", list3)
    print("id of list3", list3)
    list3 = copy.deepcopy(list1)
    print("id of list3 after deepcopy()", id(list3))

shallow_copy(list1=[1,2], list2=[])
deep_copy(list1=[1,2], list2=[], list3=[])
```

```
list1 [1, 2]
list2 []
id of list1 70677496
id of list2 70677600
id of list2 after copy() 66627144
list1 [1, 2]
list2 []
id of list1 68856224
id of list2 69359944
id of list2 66627272
list3 []
id of list3 []
id of list3 after deepcopy() 69359944
```

```
def sum_of_two_dictionary(dict1, dict2):
    print("dict1", dict1)
    print("dict2", dict2)
    dict3={}
    for k1, v1 in dict1.items():
        for k2, v2 in dict2.items():
            if k1==k2:
                dict3[k1]=dict1[k1]+dict2[k2]

            if k1 not in dict3.keys():
                dict3[k1]=v1

            if k2 not in dict3.keys():
                dict3[k2]=v2



    return dict3

dict1 = {'a':100, 'b':200, 'c':300}
dict2 = {'a':100, 'b':200, 'd':400, 'e':500}
print(sum_of_two_dictionary(dict1, dict2))
```

```
dict1 {'a': 100, 'b': 200, 'c': 300}
dict2 {'a': 100, 'b': 200, 'd': 400, 'e': 500}
{'a': 200, 'b': 400, 'd': 400, 'e': 500, 'c': 300}
```

```
def try_except(n):
    try:
        res = n / 0 # This will raise a ZeroDivisionError
    except ZeroDivisionError:
        print("can't be divided by zero")
    finally:
        print("Program ends here")


try_except(10)
```

```
can't be divided by zero
Program ends here
```

```
def find_duplicates(list1):

    print("Given list:", list1)

    duplicates = []
    unique = []

    for i in list1:
        if i not in unique:
            unique.append(i)
        else:
            duplicates.append(i)
    return unique, duplicates


unique, duplicates = find_duplicates(list1=[1,2,1,2,4,3,5,3])
print("unique numbers in given list", unique)
print("Duplicate numbers in given list", duplicates)
```

```
Given list: [1, 2, 1, 2, 4, 3, 5, 3]
unique numbers in given list [1, 2, 4, 3, 5]
Duplicate numbers in given list [1, 2, 3]
```

```
def find_palindrome(str1):
    print("Given string:", str1)
    if str1.lower()==str1.lower()[::-1]:
        print("Given string is Palindrome", str1)
    else:
        print("Given string is not palindrome", str1)

find_palindrome(str1="madam")
find_palindrome(str1="car")
```

```
Given string: madam
Given string is Palindrome madam
Given string: car
Given string is not palindrome car
```

```
def prime_number(num):

    flag = False

    # Program to check if a number is prime or not

    #num = 29

    # To take input from the user
    #num = int(input("Enter a number: "))

    # define a flag variable
    flag = False

    if num == 0 or num == 1:
        print(num, "is not a prime number")
    elif num > 1:
        # check for factors
        for i in range(2, num):
            if (num % i) == 0:
                # if factor is found, set flag to True
                flag = True
                # break out of loop
                break

        # check if flag is True
        if flag:
            print(num, "is not a prime number")
        else:
            print(num, "is a prime number")


prime_number(149)
prime_number(113)
prime_number(127)
prime_number(144)
prime_number(1)
prime_number(0)
```

```
149 is a prime number
113 is a prime number
127 is a prime number
144 is not a prime number
1 is not a prime number
0 is not a prime number
```

```
# Vowel Program in Python
def check_vowel(char):
    vowels = "aeiouAEIOU"
    if char in vowels:
        return char, "Vowel"
    else:
        return char, "Consonant"

print(check_vowel("E"))
print(check_vowel("e"))
print(check_vowel("A"))
print(check_vowel("e"))
print(check_vowel("i"))
print(check_vowel("I"))
print(check_vowel("o"))
print(check_vowel("O"))
print(check_vowel("u"))
print(check_vowel("U"))
print(check_vowel("V"))
```

```
('E', 'Vowel')
('e', 'Vowel')
('A', 'Vowel')
('e', 'Vowel')
('i', 'Vowel')
('I', 'Vowel')
('o', 'Vowel')
('O', 'Vowel')
('u', 'Vowel')
('U', 'Vowel')
('V', 'Consonant')
```

```
def check_vowelinstring(str1):
    vowels = "aeiouAEIOU"
    vowels_list = []
    Not_vowel = []
    for char in str1:
        if char in vowels:
            vowels_list.append(char)
        else:
            Not_vowel.append(char)

    return vowels_list, Not_vowel

vowels_list,Not_vowel = check_vowelinstring(str1="helloworld")
print("vowels_list",vowels_list)
print("Not_vowel",Not_vowel)
```

```
vowels_list ['e', 'o', 'o']
Not_vowel ['h', 'l', 'l', 'w', 'r', 'l', 'd']
```

```
def count_vowels(string):
    vowels = "aeiouAEIOU"
    count = 0
    for char in string:
        if char in vowels:
            count+=1
    return count

input_string = "Python Programming"
vowel_count = count_vowels(input_string)
print(f"Number of vowels in '{input_string}: {vowel_count}")
```

```
Number of vowels in 'Python Programming: 4
```

```
def list_comprehension(input_string):
    vowel_count = len([char for char in input_string if char in "aeiouAEIOU"])
    print(f"Number of vowels in '{input_string}': {vowel_count}")

input_string="helloworld"
list_comprehension(input_string)
```

```
Number of vowels in 'helloworld': 3
```

```
def generator_values():
    for i in range(10):
        yield i
    print(" ")


for value in generator_values():
    print(value)
```

```
0
1
2
3
4
5
6
7
8
```

```
9
```

```python
def iterator_values():
    iter_obj = iter(range(10))
    for i in iter_obj:
        print(i, iter_obj.__next__(), end="\n")

iterator_values()
```

```
0 1
2 3
4 5
6 7
8 9
```

```python
# Move all zero to end of the list
def move_all_zerostoendoflist(list1):
    for i in list1:
        if i == 0:
            p = list1.pop(list1.index(i))
            list1.append(p)
    return list1

print(move_all_zerostoendoflist(list1=[0,1,20,0,3,4]))
```

```
[1, 20, 3, 4, 0, 0]
```

```python
# Move all zero to end of the list
list1 = [0,12,3,0,3,2,30,0,0,4,5,6]
list1[:] =  [i for i in list1 if i!=0]+[0]*list1.count(0) # [0]*4 # list1.count(0)=4
print(list1[:])
```

```
[12, 3, 3, 2, 30, 4, 5, 6, 0, 0, 0, 0]
```

```python
from collections import Counter

# Input text
text = "apple banana apple orange banana apple"
word_counts = Counter(text.split())

# Get the most common word
most_common_word = word_counts.most_common(1)[0][0]
print(most_common_word)
```

```
apple
```

```python
def most_common_word(text):
    word_counts = {}
    for word in text.split():
        word_counts[word] = word_counts.get(word,0) + 1

    return word_counts

word_counts = most_common_word(text)
most_common_word = max(word_counts, key=word_counts.get)
print(most_common_word)
```

```
apple
```

```python
# Using regular expression(Handle Punctuation)
import re
from collections import Counter

# Input text with punctuation
text = "apple, banana, apple? orange; banana:apple."
words = re.findall(r'\w+', text.lower())
print("words", words)
most_common_word = Counter(words).most_common(1)[0][0]
print(most_common_word)

# only punctuation
words = re.findall(r'\b\W\b', text.lower())
print("words", words)
most_common_word = Counter(words).most_common(1)[0][0]
print(most_common_word)
```

```
words ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']
apple
words [':']
:
```

```python
def palindrome_number(num):
    if num_str == num_str[::-1]:
        print("Given number is Palindrome number:", num_str)
    else:
        print("Given number is not a Palindrome number:", num_str)

num = 121
num_str = str(num)
palindrome_number(121)

num = -121
num_str = str(num)
palindrome_number(-121)

num = 10
num_str = str(num)
palindrome_number(10)
```

```
Given number is Palindrome number: 121
Given number is not a Palindrome number: -121
Given number is not a Palindrome number: 10
```

```python
def call_by_value(x):
    x = x * 2
    return x

def call_by_reference(b):
    b.append("D")
    return b
```

```python
a = ["E"]
num = 6

# call functions
updated_num = call_by_value(num)
updated_list = call_by_reference(a)

# Print after function calls
print("Updated value after call by value:", updated_num)
print("Updated list after call by reference:", updated_list)
```

```
Updated value after call by value: 12
Updated list after call by reference: ['E', 'D']
```

```python
"""
Valid Parenthesis - Leet Code Problem - Easy
--------------------------------------------

Given a string s containing just the characters
'(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets.
Open brackets must be closed in the correct order.
Every close bracket has a corresponding open bracket of the same type.


Example 1:
Input: s = "()"
Output: true

Example 2:
Input: s = "()[]{}"
Output: true

Example 3:
Input: s = "(]"
Output: false

Example 4:
Input: s = "([])"
Output: true

Example 5:
Input: s = "([)]"
Output: false

Constraints:
1 <= s.length <= 104
s consists of parentheses only '()[]{}'.
"""
def is_balanced(s):
    stack = []
    mapping = {')':'(','}':'{',']':'['}
    for char in s:
        if char in mapping.values():
            stack.append(char) # stack = ['(','}','[']
        elif char in mapping.keys(): # mapping.keys() = ')','{','['
            if not stack or stack[-1]!=mapping[char]:
                # '['!=mapping[']'], '['!='[','('!='}','{'!='}'
                return False
            stack.pop()
    return not stack

print(is_balanced("{[()]}"))
print(is_balanced("{[(])}"))
print(is_balanced('()'))
print(is_balanced('()[]{}'))
print(is_balanced('(]'))
print(is_balanced('([])'))
print(is_balanced('([)]'))
```

```
True
False
True
True
False
True
False
```

```
# Functions
#define functions with def
#Always use () to call a function
#Add return to send values back
#Create anonymous functions with the lambda keyword

# defining functions
def greet():
    return "Hello!"

def greet_person(name):
    return f"Hello, {name}!"

def add(x, y=10):    # Default parameter
    return x + y

# Function call
print(greet())
print(greet_person("Ram"))
print(add(10))
print(add(7,10))
```

```
Hello!
Hello, Ram!
20
```

```
17
```

```python
def positional_arguments(name, age):
    return f"my name is {name}, my age is {age}"

print(positional_arguments('Anamika', 8)) #Positional argument
```

```
my name is Anamika, my age is 8
```

```python
def positional_default_arguments(name, age=7):
    return f"my name is {name}, my age is {age}"

print(positional_default_arguments('Anamika')) # Default argument
```

```
my name is Anamika, my age is 7
```

```python
def variable_no_of_arguments(*args):
    args = args
    for i in args:
        print(i, type(i))

args = ["hello", 10, 5.5, [1,2,3,4,5], (10,20,30), {1,2,3,4,5}, {'a':1,'b': 2}]
variable_no_of_arguments(args) # variable no of arguments
```

```
['hello', 10, 5.5, [1, 2, 3, 4, 5], (10, 20, 30), {1, 2, 3, 4, 5}, {'a': 1, 'b': 2}]
```

```python
def variable_no_of_keyword_arguments(**kwargs):
    kwargs = kwargs
    for k, value in kwargs.items():
        print(k, value, type(k))

# variable no of keyword arguments
variable_no_of_keyword_arguments(s1='Hi', s2='Bro', s3='Where are you')
```

```
s1 Hi <class 'str'>
s2 Bro <class 'str'>
s3 Where are you <class 'str'>
```

```python
# Return values
def get_min_max(numbers):
    return min(numbers), max(numbers)

minimum, maximum = get_min_max([1, 5, 3])
print("minimum", minimum)
print("maximum", maximum)
```

```
minimum 1
maximum 5
```

```python
# To remove a duplicate from the list
def remove_duplicate(list1):
    output = set(list1)
    return output

print(remove_duplicate(list1=[1,2,3,1,2,4,5]))
```

```
{1, 2, 3, 4, 5}
```

```
def func(daynumber):

        d = {1:"Monday", 2:"Tuesday", 3:"Wednesday", 4:"Thursday", 5:"Friday",6:"Sat

        daynumber = d.get(daynumber,0)

        if daynumber == "Monday":
            print("Monday")
        elif daynumber == "Tuesday":
            print("Tuesday")
        elif daynumber == "Wednesday":
            print("Wednesday")
        elif daynumber == "Thursday":
            print("Thursday")
        elif daynumber == "Friday":
            print("Friday")
        elif daynumber == "Saturday":
            print("Saturday")
        elif daynumber == "Sunday":
            print("Sunday")
        else:
            print("Invalid key")


func(daynumber=1)
func(daynumber=2)
func(daynumber=3)
func(daynumber=4)
func(daynumber=5)
func(daynumber=6)
func(daynumber=7)
func(daynumber=8)
```

```
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Invalid key
```

```
# write a program that handles discussion by zero using try-except-finally ?
def number_divison(number1, number2):
    try:
        op = number1 / number2
        print("Divisible by number", op)
    except ZeroDivisionError:
        print("Enter the number other than zero")
    except ValueError:
        print("Enter the number and not aplhabets")
    except NameError:
        print("Enter the number and not name")
    finally:
        print("Program ended")

number_divison(10, 2)
```

```
Divisible by number 5.0
Program ended
```

```
# reverse the list using list comphrehension
def reverse_list_comphrenshion(list1):
    reversedlist =  [list1[i] for i in range(len(list1) -1,-1,-1)]
    return reversedlist

my_list = [1,2,3,4,5]
print(reverse_list_comphrenshion(my_list))
```

```
[5, 4, 3, 2, 1]
```

```
# How to reverse the list using python
def reverse_list(my_list):
    reversed_list = []
    for item in my_list:
        if item not in reversed_list:
            reversed_list.insert(0, item)

    print("Reversed list", reversed_list)

my_list = [1,2,3,4,5]
reverse_list(my_list)
```

```
Reversed list [5, 4, 3, 2, 1]
```

```
# How to reverse the list using python
def reverselist(my_list):
    my_list = [1,2,3,4,5]
    i = len(my_list) - 1
    reversedlist = []
    while i>=0:
        reversedlist.append(my_list[i])
        i-=1

    print("Reversed list", reversedlist)

reverselist(my_list=[1,2,3,4,5])
```

```
Reversed list [5, 4, 3, 2, 1]
```

```
def repeating_nonrepeatingchar(str1=""):
        char_dict = {}
        for char in str1:
            if char in char_dict:
                char_dict[char]=char_dict.get(char, 0) + 1
            else:
                char_dict[char]=char_dict.get(char, 0) + 1

        print("char_dict", char_dict)
        repeating_char = {}
        nonrepeating_char = {}
        for k, cnt in char_dict.items():
            if cnt > 1:
                repeating_char[k] = cnt
            else:
                nonrepeating_char[k] = cnt

        return repeating_char, nonrepeating_char


repeating_char, nonrepeating_char = repeating_nonrepeatingchar(str1="helloworld")
print("repeating_char", repeating_char)
print("nonrepeating_char", nonrepeating_char)
```

```
char_dict {'h': 1, 'e': 1, 'l': 3, 'o': 2, 'w': 1, 'r': 1, 'd': 1}
repeating_char {'l': 3, 'o': 2}
nonrepeating_char {'h': 1, 'e': 1, 'w': 1, 'r': 1, 'd': 1}
```

```
def repeating_nonrepeatingword(words):
        word_dict = {}
        words = words.split()
        for word in words:
            if word not in word_dict:
                word_dict[word]=word_dict.get(word, 0) + 1
            else:
                word_dict[word]=word_dict.get(word, 0) + 1

        repeating_word = {}
        nonrepeating_word = {}

        for k, cnt in word_dict.items():
            if cnt > 1:
                repeating_word[k] = cnt
            else:
                nonrepeating_word[k] = cnt

        return repeating_word, nonrepeating_word


repeating_word, nonrepeating_word = repeating_nonrepeatingword(words="Hi are you the
print("repeating_words", repeating_word)
print("nonrepeating_word", nonrepeating_word)
```

```
repeating_words {'are': 2, 'you': 2, 'there': 2}
nonrepeating_word {'Hi': 1}
```

```
def duplicate_items(list1):
        print("Original list:", list1)
        unique = []
        duplicate = []
        for item in list1:
            if item not in unique:
                unique.append(item)
            else:
                duplicate.append(item)

        return unique, duplicate

print(duplicate_items(list1=[1,2,3,1,2,4,5]))
```

```
Original list: [1, 2, 3, 1, 2, 4, 5]
([1, 2, 3, 4, 5], [1, 2])
```

```
# Right angle triangle
def rightangle_triangle(num):
    for i in range(num):
        for j in range(i):
            print("*", end=' ')
        print()

rightangle_triangle(num=10)
```

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
```

```
# Reversed Right angle triangle
def reversedrightangle_triangle(num):
    for i in range(num, 0, -1):
        for j in range(i):
            print(i, end=' ')
        print()

reversedrightangle_triangle(num=10)
```

```
10 10 10 10 10 10 10 10 10 10
9 9 9 9 9 9 9 9 9
8 8 8 8 8 8 8 8
7 7 7 7 7 7 7
6 6 6 6 6 6
5 5 5 5 5
4 4 4 4
3 3 3
2 2
1
```

```
def rhombus(rows):
    for i in range(1,rows+1):
        print(' '*(rows-i)+"* "*i)
    for i in range(rows-1,0,-1):
        print(' '*(rows-i)+" *"*i)
```

```
rhombus(rows=10)
```

```
         *
        * *
       * * *
      * * * *
     * * * * *
    * * * * * *
   * * * * * * *
  * * * * * * * *
 * * * * * * * * *
* * * * * * * * * *
  * * * * * * * * *
   * * * * * * * *
    * * * * * * *
     * * * * * *
      * * * * *
       * * * *
        * * *
         * *
          *
```

```
# Pyramid
def pyramid(rows):
    for i in range(1, rows+1):
        print(' '*(rows-1)+"* "*i)

pyramid(rows=5)
```

```
    *
    * *
    * * *
    * * * *
    * * * * *
```

```
def find_character(string):
    int_value = {}
    for char in string:
        print(f"Numeric Value of {char}: {ord(char)}")
        int_value.update({char: ord(char)})

    for key, value in int_value.items():
        print(f"ASCII Value of {value}: {chr(value)}")

string = "abcdefghijklmnopqrstuvwxyz"
find_character(string)
```

```
Numeric Value of a: 97
Numeric Value of b: 98
Numeric Value of c: 99
Numeric Value of d: 100
Numeric Value of e: 101
Numeric Value of f: 102
Numeric Value of g: 103
Numeric Value of h: 104
Numeric Value of i: 105
Numeric Value of j: 106
Numeric Value of k: 107
Numeric Value of l: 108
Numeric Value of m: 109
Numeric Value of n: 110
Numeric Value of o: 111
Numeric Value of p: 112
Numeric Value of q: 113
Numeric Value of r: 114
Numeric Value of s: 115
Numeric Value of t: 116
Numeric Value of u: 117
Numeric Value of v: 118
Numeric Value of w: 119
Numeric Value of x: 120
Numeric Value of y: 121
Numeric Value of z: 122
ASCII Value of 97: a
ASCII Value of 98: b
ASCII Value of 99: c
ASCII Value of 100: d
ASCII Value of 101: e
ASCII Value of 102: f
ASCII Value of 103: g
ASCII Value of 104: h
ASCII Value of 105: i
ASCII Value of 106: j
ASCII Value of 107: k
ASCII Value of 108: l
ASCII Value of 109: m
ASCII Value of 110: n
ASCII Value of 111: o
ASCII Value of 112: p
ASCII Value of 113: q
ASCII Value of 114: r
ASCII Value of 115: s
ASCII Value of 116: t
ASCII Value of 117: u
ASCII Value of 118: v
ASCII Value of 119: w
ASCII Value of 120: x
ASCII Value of 121: y
ASCII Value of 122: z
```

```python
def list_comprehension(a):
    res = [val ** 2 for val in a]
    print(res)

a = [1,2,3,4,5]
list_comprehension(a)
```

```
[1, 4, 9, 16, 25]
```

```python
def swapping_two_values():
    # swapping numbers without temporary variable in Python

    # 1. Using tuple unpacking
    a,b = 5,10
    print("Before unpacking", "a", a, "b", b)

    a,b = b,a
    print("After unpacking", "a", a, "b", b)

    # 2. Using arithmetic operations
    a,b = 5,10
    a = a + b # a becomes 15
    b = a - b # b becomes 5 (original value of a) 15-10=5
    a = a - b # a becomes 10(original value of b) 15-5=10
    # Now b is 5, a is 10

    # Using XOR Bitwise Operator
    a, b = 5, 10
    a = a ^ b
    b = a ^ b
    a = a ^ b
    print("a:", a, "b:", b)

swapping_two_values()
```

```
Before unpacking a 5 b 10
After unpacking a 10 b 5
a: 10 b: 5
```

```python
# Searching Algorithms
# Searching algorithms are used to find a specific element in a data structure.
# Linear search and Binary search are two common searching algorithms.
# Binary search is more efficient for sorted data.
# Example of using the built-in sorted() function in Python
arr = [3,1,4,1,5,9,2,6,5,3,5]
sorted_arr = sorted(arr)
print(sorted_arr)

def binary_search(arr, target):
    left, right = 0, len(arr) - 1 # len(arr) == 10, (0, 10-1), (0,9), 0 + (9-0)//2
    while left - right:
        mid = left + (right - left) // 2 # mid = 4,
        if arr[mid] == target: #arr[4] = 5, 5 == 5
            return mid # mid is 4
        elif arr[mid] < target: # arr[4] < 5; left = 4+1 = 5
            left = mid + 1
        else:
            right = mid - 1 # 5 - 1 = 3

    return -1
```

```
arr = [1,2,3,4,5,6,7,8,9,10]
target = 5
result = binary_search(arr, target)
print("binary sort result", result)
```

```
[1, 1, 2, 3, 3, 4, 5, 5, 5, 6, 9]
binary sort result 4
```

```
def lambda_function(s1):
    """
    A lambda function is an anonymous function.
    This function can have any number of parameters
    but, can have just one statement
    In this example, we defined a lambda function(upper)
    to convert a string to its upper case using upper()
    """
    print("Given string:", s1)
    s2 = lambda func: func.upper()
    print(s2(s1), end="\n")

lambda_function(s1="hello")
```

```
Given string: hello
HELLO
```

```
def variable_arguments(*argv):
    for arg in argv:
        print(arg, end="\n")
        print("type:", type(arg), end="\n")

variable_arguments("Hello",[1,2,3],(10,20,30),50, 4.5, {1,2,3}, {'a':1,'b':2})
```

```
Hello
type: <class 'str'>
[1, 2, 3]
type: <class 'list'>
(10, 20, 30)
type: <class 'tuple'>
50
type: <class 'int'>
4.5
type: <class 'float'>
{1, 2, 3}
type: <class 'set'>
{'a': 1, 'b': 2}
type: <class 'dict'>
```

```
def variable_keywordarguments(**kwargs):
    for k, value in kwargs.items():
        print(k, value, end="\n")

variable_keywordarguments(kwargs={'a':10,'b':20,'c':30})
```

```
kwargs {'a': 10, 'b': 20, 'c': 30}
```

```python
# Function to check if a number is an Armstrong number
def is_armstrong_number(number):
    num_str = str(number)
    print("num_str", num_str)
    num_digits = len(num_str)
    print("num_digits", num_digits)
    armstrong_sun = sum(int(digit) ** num_digits for digit in num_str)
    print("armstrong_sun",armstrong_sun)
    print(int(1) ** 3)
    print(int(5) ** 3)
    print(int(3) ** 3)
    return armstrong_sun == number

# Input number
number =  153

# check and print result
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

```
num_str 153
num_digits 3
armstrong_sun 153
1
125
27
153 is an Armstrong number.
```

```python
def test_variables():
    # Rules for naming variables or identifier
    # Data types: Int, float, bool, str, list, tuple, dict, set, frozenset
    # variable - to store the data (word, number, decimal) - (str, int, float)
    # Python is supported dynamic data type -
    # Mutable(list, dict) or immutable(str, tup) - changeable or unchangeable
    # data type determined during compile time

    # Variables & Data type
    var = "Hello"
    Var = 10
    _var = 3.5
    data = {'a':1,'b':2}
    tup = (1,2,3,4)
    boo = bool(True)
    VAR = [1,2,3,4]
    __value = set({1,2,3,4})
    # class methods - str
    print(f"var: {var}, type: {type(var)}")
    #print(f"var: {var}, type: {type(var)}, dir: {dir(var)}",end = "\n")
    print(" ")
    # class methods - int
    print(f"Var: {Var}, type: {type(Var)}")
    print(" ")
    # class methods - float
    print(f"_var: {_var}, type: {type(_var)}")
    print(" ")
    # class methods - dict
    print(f"data: {data}, type: {type(data)}")
    print(" ")
    # class methods - tuple
    print(f"tup: {tup}, type: {type(tup)}")
    print(" ")
    # class methods - bool
    print(f"boo: {boo}, type: {type(boo)}")
    print(" ")
    # class methods - list
    print(f"VAR: {VAR}, type: {type(VAR)}")
    print(" ")
    # class methods - set
    print(f"__value: {__value}, type: {type(__value)}")
    print(" ")

test_variables()
```

```
var: Hello, type: <class 'str'>

Var: 10, type: <class 'int'>

_var: 3.5, type: <class 'float'>

data: {'a': 1, 'b': 2}, type: <class 'dict'>

tup: (1, 2, 3, 4), type: <class 'tuple'>

boo: True, type: <class 'bool'>

VAR: [1, 2, 3, 4], type: <class 'list'>

__value: {1, 2, 3, 4}, type: <class 'set'>
```

```python
def iterate_items_usingloop(list1):
    for item in list1:
        print(item)

iterate_items_usingloop(list1=[1,2,3,4,5,6])
```

```
1
2
3
4
5
6
```

```python
def iterate_items_usingrangelen(list1):
    for index in range(len(list1)):
        for item in list1:
            if item == 2:
                pass

            if item == 5:
                break

            if item == 4:
                continue
    print(index, list1[index])

iterate_items_usingrangelen(list1=[1,2,3,4,5,6])
```

```
5 6
```

```python
def iterate_items_usingenumerate(list1):
    for indx, item in enumerate(list1):
        print(indx, item)

iterate_items_usingenumerate(list1=[1,2,3,4,5,6])
```

```
0 1
1 2
2 3
3 4
4 5
5 6
```

```python
# Different types of arguments
    # 1. default argument
    # 2. Keyword argument
    # 3. Positional argument
    # 4. Variable length argument
    # 5. Variable length keyword argument


 # Arguments vs Parameters
def greet(name, message="hello"):
    """Greets, someone with a message"""
    print(message, name + "!")

greet("Python")
```

```
hello Python!
```

```python
#  ------------------------------------
# keyword argument or named argument
#  ------------------------------------

def area_of_rectangle(length, width): # paramters
    #Document string
    '''
    This function calculates the area of the rectangle
    '''
    area = length * width
    return area

#rectangle_Area = area_of_rectangle(length, width)
#Arguments - length=5, width=3

rectangle_Area = area_of_rectangle(5, 3)
print(rectangle_Area)
```

```
15
```

```
#Arguments vs Parameters

#  ------------------------------------
#  Postional argument first and second
#  ------------------------------------

def area_of_rectangle(length, width): # paramters
    #Document string
    '''
    This function calculates the area of the rectangle
    '''
    area = length * width
    return area

#rectangle_Area = area_of_rectangle(length, width) #Arguments

#rectangle_Area = area_of_rectangle(5)
# TypeError: area_of_rectangle() missing 1 required positional argument: 'width'
rectangle_Area = area_of_rectangle(5, 3)
# a, b length, width is not ordered - it maps the  argument value to variable name
# keyword argument or named argument
print(rectangle_Area)
```

```
15
```

```
# Arguments vs Parameters

#  ------------------------------------
#  first one is keyword argument or named argument, last one is default argument
#  ------------------------------------

def area_of_rectangle(length, width=10): # paramters
    #Document string
    '''
    This function calculates the area of the rectangle
    '''
    area = length * width
    return area

#rectangle_Area = area_of_rectangle(length, width) #Arguments - length=5, width=3

rectangle_Area = area_of_rectangle(5)
print(rectangle_Area)
```

```
50
```

```python
# void function -> Not return any value, without arguments
def add_func():
    # user input for integers for add operation,
    # we can't ask every time user to provide input
    # we can make use of function parameters a, b
    #a = input("Enter a number")
    #b = input("Enter a number")
    # user has to provide the same input to proceed the add operation
    # User has to provide different input to proceed the add operation
    # input: a, b
    # values: 10, 20
    # expression: a +b
    # output: 30
    # print the output
    # print(a + b)
    output = add(10,20)
    print(output)

add_func()
```

```
30
```

```python
# void function -> Not return any value
def add(a,b):
    print(a + b)

add(10,20)
```

```
30
```

```python
#Non Void function
# void function -> Not return any value.,without arguments
# Non void function -> It return value
def add(a,b):
    return a + b

output = add(10,20)
print(output)
```

```
30
```

```python
# Non void function with or without arguments
# user input for integers for add operation, we can't ask every time user to provide
# we can make use of function parameters a, b
# a = input("Enter a number")
# b = input("Enter a number")
# user has to provide the same input or different input to proceed the add operation
# input: a, b
# No of arguments in function definition and function call should match
# values: 10, 20
# expression: a +b
# output: 30
# return the output
```

```python
def add(a, b): # Function Parameters
    return a + b

def subract(a, b): # Function Parameters
    return a - b

def multiply(a, b): # Function Parameters
    return a * b

def division(a, b): # Function Parameters
    return a / b

# 10, 20 are Function Arguments, add() is function call
a = add(10,20)

# 10, 20 are Function Arguments, subtract() is function call.
b = subract(20,10)

# 10, 20 are Function Arguments, multiply() is function call.
c = multiply(10,20)

# 10, 20 are Function Arguments, division() is function call.
d = division(20,10)

print("add:", a)
print("subract:", b)
print("multiplication:", c)
print("Division:", d)
```

```
add: 30
subract: 10
multiplication: 200
Division: 2.0
```

```python
# Functions
def hello():
    print("blah blab")
    for i in range(5):
        print("hello world")
    print("blah blab")


hello() # Function code for re-use, Function is a block of code,
# No need to repeat the code multiple times, re-use using function call,

# Function definition using def keyword
# Function definition def followed by function name
# Function Parameter ( a, b are variables
# arguments is nothing values for variables a=10, b=20)
# Function logic code
    # input from user, for loop, while loop,
    # if-elif-else statements
    # list comphrehension,
    # lambda, expression, map, reduce, filter
# function call, recursion
# After validation, print the output, None, Return value
 # Function return keyword to return the values

def add(a,b):
    return a + b

print(add(10,20))
```

```
blah blab
hello world
hello world
hello world
hello world
hello world
blah blab
30
```

```
def test_remove_vowels_in_string(str1, results="", vowels=[]):
    #str1 = str(input("Enter the string: "))
    print(f"Given string: {str1}")
    result= ""
    vowels = ['a','e','i','o','u']
    for letter in str1:
        if letter.lower() not in vowels:
            result += letter

    print(f"Remove vowels in string, {result}")

test_remove_vowels_in_string(str1="helloworld", results="", vowels=[])
```

```
Given string: helloworld
Remove vowels in string, hllwrld
```

```
def test_reverse_a_string(str1=" "):
        #str1 = str(input("Enter a string:"))
        print(f"Given string: {str1}")
        rev = ""
        for char in str1:
            rev = char + rev
        print(rev)

test_reverse_a_string(str1="hello")
```

```
Given string: hello
olleh
```

```
# Binary search algorithm
def binary_search(list1, num):
    left = 0
    right = len(list1) - 1
    mid = (left + right) // 2
    if mid == num:
        return mid
    if mid <= num:
        return left + 1
    else:
        return right - 1

    return -1

print(binary_search(list1=[1,2,3,4,5], num=3))
```

```
1
```

```
# Linear or sequential search or quick search algorithm
def quick_search(arr, x):
    n = len(arr)
    for i in range(0, n):
        if (arr[i] == x):
            return i
    return -1

if __name__ == "__main__":
    arr = [2,3,4,10,40]
    x = 10
    result = quick_search(arr, x)
    if result == -1:
        print("Element is not present in array")
    else:
        print("Element is present at index", result)
```

```
Element is present at index 3
```

```
def find_largest_number(listvalue):
    print("Given list:", listvalue)
    largest = listvalue[0]
    for value in listvalue:
        if value > largest:
            largest = value
    print(largest)


find_largest_number(listvalue=[4,5,6,10,7,8,9])
```

```
Given list: [4, 5, 6, 10, 7, 8, 9]
10
```

```
def find_smallest_number(listvalue):
    print("Given list:", listvalue)
    smallest = listvalue[0]
    for value in listvalue:
        if value < smallest:
            smallest = value
    print(smallest)

find_smallest_number(listvalue=[4,5,6,10,7,8,9])
```

```
Given list: [4, 5, 6, 10, 7, 8, 9]
4
```

```
def flattened_list(nested_list):
    flattend_list = []
    for sublist in nested_list:
        for item in sublist:
            flattend_list.append(item)
    return flattend_list

flattened_list = flattened_list(nested_list=[[1,2,3],[4,5,6],[7,8,9]])
print("flattened_list", flattened_list)
```

```
flattened_list [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
from itertools import combinations
combination_pairs = list(combinations(numbers, 2))
print(combination_pairs)
```

```
[(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)]
```

```
numbers = [1, 2, 3, 4, 5]
combination_pairs_manual = []

for i in range(len(numbers)):
    for j in range(i + 1, len(numbers)):
        # Start 'j' from 'i + 1' to avoid duplicate pairs and self-pairing
        combination_pairs_manual.append((numbers[i], numbers[j]))

print(combination_pairs_manual)
```

```
[(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)]
```

```
def remove_vowels_in_string(str1, results="", vowels=[]):
        #str1 = str(input("Enter the string: "))
        print(f"Given string: {str1}")
        result= ""
        vowels = ['a','e','i','o','u']
        for letter in str1:
            if letter.lower() not in vowels:
                result += letter

        print(f"Remove vowels in string, {result}")

remove_vowels_in_string(str1="helloworld", results="", vowels=[])
```

```
Given string: helloworld
Remove vowels in string, hllwrld
```

```
def reverse_string_consonants_first(str1):
        vowels = []
        consonants = []
        for val in str1:
            if val in 'AEIOUaeiou': # check val not in consonants or vowels
                vowels.append(val)
            else:
                consonants.append(val)

        reverse_string = ''.join(consonants + vowels)
        # Convert a list of characters into a string
        print("reverse of string with vowels first", reverse_string)

reverse_string_consonants_first(str1="helloworld")
```

```
reverse of string with vowels first hllwrldeoo
```

```
# Define a function named word_count that takes one argument, 'str'.
def test_word_count(user_str):
    # Create an empty dictionary named 'counts' to store word frequencies.
    # dict method - using keys to check the word frequency is incremented or not
    # https://blog.finxter.com/how-to-count-the-number-of-words-in-a-string-in-pytho
    counts = dict()

    # Split the input string 'str' into a list of words
    # using spaces as separators and store it in the 'words' list.
    words = user_str.split()

    # Iterate through each word in the 'words' list.
    for word in words:
        # Check if the word is already in the 'counts' dictionary.
        if word in counts:
            # If the word is already in the dictionary, increment its frequency by 1
            counts[word] += 1
        else:
            # If the word is not in the dictionary,
            # add it to the dictionary with a frequency of 1.
                counts[word] = 1
    print(f'word count: {word}, counts: {counts}',end="\n")

    # Return the 'counts' dictionary, which contains word frequencies.
    return counts

test_word_count(user_str="Hi are you there are you")
```

```
word count: you, counts: {'Hi': 1, 'are': 2, 'you': 2, 'there': 1}
```

```
{'Hi': 1, 'are': 2, 'you': 2, 'there': 1}
```

```
import re

def test_word_count_using_re_method(user_str):
    # count the number of words in given string using re module and len() func
    op = re.findall(r"\w+", user_str)
    op = list(op)
    print(op)
    print("Length of word", len(re.findall(r"\w+", user_str)),end="\n")

test_word_count_using_re_method(user_str="Hi are you there are you")
```

```
['Hi', 'are', 'you', 'there', 'are', 'you']
Length of word 6
```

```
# File handling
# Write all content of a file into a new file by skipping line number 5
# create a test.txt file and add the below content to it.
# Read all lines from the 'test.txt' file using the readlines() method
# This method returns all lines from a file as a list.
# Open a new text file in write mode ('w')
# Set counter = 0. Iterate through each line from the list
# If the counter is 4. Skip that line,
# otherwise, write that line to the new text file using the write() method
# Increment counter by 1 in each iteration

# File handling is an important part of any web application.
# Python has several functions for creating, reading, updating and deleting files.
# The open() function takes two paramters: filename and mode
# There are four different methods for opeining a file
# "r" - Read - Default value. Opens a file for reading error if the file does not ex
# "a" - Append - Opens a file for appending, creates the file if it does not exist
# "w" - Write - Opens a file for writing, creates the file if it does not exist
# "x" - create - Creates the specified file, returns an error if the file exists.
# "t" - Text - Default value. Text mode
# "b" - Binary - Binary mode (e.g images)
# Syntax - To open a file for reading it is enough to specify the name of the file.
'''
with open("test.txt", "r") as fp:
    lines = fp.readlines()

Traceback (most recent call last):
File "<pyshell#53>", line 1, in <module>
with open("test.txt", "r") as fp:
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
f = open("demo.txt", "r")
f.read()

#'Hello! Welcome to demofile.txt\nThis file is for testing purposes.\nGood Luck!\n'

with open("demo.txt", "r") as fp:
    count  = 0
    for line in fp.read().lower().split(" "):
        print(line)


with open("demo.txt", "r") as fp:
    count  = 0
    for line in fp.readlines():
        print(line)

with open("demo.txt", "a") as f:
    f.write("RDT is a new technology which brings new level of visibility and contro
    over shared platform resources such as L3, DIMM and IO memory like UPI,
    CXL, PCIe IO cards, Accelerator used by applications
    or VM or containers on platform concurrently.")

with open("demo.txt", "r") as f:
    print(f.read())
```

```
if os.path.exists('demo.txt'):
    print("demo.txt file exists")
else:
    print("The file does not exist")


with open("demo.txt", "w") as f:
    f.write("Deleted all the lines")

with open("demo.txt", "r") as f:
        print(f.read())

import os
os.remove('demo.txt')

if os.path.exists('demo.txt'):
    print("demo.txt file exists")
else:
    print("The file does not exist")
'''

# How is exceptional handling done in Python ?
'''
There are 3 main keywords 1.e try, except and finally which are used to catch except


1. try: A block of code that is monitored for errors

2. except: Execute when an error occurs in the try block

3. finally: Executes after the try and except blocks, regardless of whether an error
It is used for cleanup tasks.

Example: Trying to divide a number by zero will cause an exception
In this example, dividing number by 0 raises a ZeroDvisionError.
The try block contains the code that might cause an exception and the except block h
printing an error message instead of stopping the program.

What is exception ?
An exception is an unwanted or unexpected event that occurs during the execution of
(i.e at runtime) and disrupts the normal flow of the program's instructions.
It occurs when something unexpected things happens, like accessing an invalid index,
or trying to open a file that does not exist.

Without Exception Handling
- If months is 0, the program will throw an error(.eg .. Division by Zero error) and

with Exception Handling
- This pseudocode includes exception handling to manage division by zero.

try - The program attempts to execute the code
catch block - if a division by zero occus, it handles the error by displaying a mess
```

```
Finally Block - The program logs the calculation attempt or ensures necessary cleanu
Custom Exception - We can create a custom exception like InvalidMonthException for i
'''

# Regular Expressions - Special characters and Patterns in regular expressions
# Regular expression
# re module is a powerful tool for searching, matching, and manipulating text based
'''
# Special characters   what is matches
.                     Any single character except a newline
*                     Zero or more repetiions of the preceding character or pattern
+                     One or more repetitions of the preceding character or pattern
?                     Zero or one repetition of the proceding character or pattern
{m,n}                 The preceding character/pattern at least m times and at most n
[abc]                 Any single character in the set (a,b,c)
\d                    Any digit (0-9)
\D                    Non digit
\s                    Any whitespace character in the set(a,b,c)
\S                    Any other character apart from whitespace
[A-Z]                 Uppercase letters
[a-z]                 lowercase letters
[0-9]                 digits
\.                    a literal dot
^                     start of the string
$                     end of the string

'''
# re.aearch() - searches the entire input string for a match to the given pattern.
# It returns a match object if a match is found, and None
# re.match()
# - is pretty similar to re.search(),
# but only checks if the pattern matches at the beginning of the input string

# re.findall() - returns all non-overlapping matches of the pattern in the input str
# re.compile()
# - It compiles a regular expression pattern into a pattern object
# Basic pattern matching using search()

# Excercise 1
import re

# Checking if the string starts with "Hello"
s = "hello World"
match = re.match(r"\d+", s)

if match:
    print("Pattern found!")
else:
    print("Pattern not found.")
```

```
<>:4: SyntaxWarning: invalid escape sequence '\d'
/tmp/xpython_42/556466855.py:4: SyntaxWarning: invalid escape sequence '\d'
  '''
```

```
Pattern not found.
```

```
# Excercise 2
# Using findall() to find all occurrences of a pattern
text = "I love to code in Python and it's amazing!"
pattern = "a"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))

# Output:Matches found: 3
```

```
Matches found: 3
```

```
# Excercise 3
import re
pattern = r"\s"
text = "I like Python and it is amazing!"
split_text = re.split(pattern, text)
print("Split text:", split_text)

# Output: Split text: ['I', 'like', 'Python', 'and', 'it', 'is', 'amazing!']
```

```
Split text: ['I', 'like', 'Python', 'and', 'it', 'is', 'amazing!']
```

```
# Excericse 4
# Using sub() to replace a pattern with a string
import re
pattern = "Python"
replacement = "Java"
text = "Python is fun"
substituted_text = re.sub(pattern, replacement, text)
print("Substituted text:", substituted_text)
```

```
Substituted text: Java is fun
```

```
# Excericse 5
import re
pattern = "hi"
replacement = "hello"
text="hiworld"
op = re.sub(pattern, replacement, text)
print(op)
```

```
helloworld
```

```
#Using ^ to match the start of a string
import re
pattern = "^I"
text = "I love Python!"
match = re.search(pattern, text)
print("Match found:", bool(match))
```

```
Match found: True
```

```
#Using $ to match the end of a string
pattern = "Python!$"
text = "Python programming is great!"
match = re.search(pattern, text)
```

```
print("Match found:", bool(match))
```

```
Match found: False
```

```
# Using [] to create a character set
pattern = "[Pp]ython"
text = "I write code in python and Python!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
```

```
Matches found: 2
```

```
#using . to match any character
pattern = "Py...n"
text = "I love Python, Pyt3on, Py45n, and Py@#n!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
print(matches) # ['Python', 'Pyt3on']
```

```
Matches found: 2
['Python', 'Pyt3on']
```

```
# Using + to match one or more occurrences of a character
import re
pattern = "Py.+n"
count = 0
text = ["Python coding", "Pyt3on","Java", "Py45n", "Py@#n","Pyn"]
for i in text:
    op = re.findall(pattern, i)
    print(op)
for i in text:
    if(re.findall(pattern, i)):
        count+=1
print("Matches found:", count)
```

```
['Python codin']
['Pyt3on']
[]
['Py45n']
['Py@#n']
[]
Matches found: 4
```

```
# To find my mobile number
import re
mobile = "9138673223"
op = re.findall(r"9\d{9}", mobile)
pattern =r"(9)?[0-9]{9}"
print(op)
```

```
['9138673223']
```

```
op = re.findall(r"^\d{10}$", mobile)
op
```

```
['9138673223']
```

```
# With country code (e.g +91 for India)
op = re.findall(r"^\+?\d{1,3}?[-\s]?\(?\d{2,4}\)?[-\s]?\d{3,4}[-\s]?\d{4}$", mobile)
```

```
op
```

```
['9138673223']
```

```
import re

def is_valid_phone(phone):
    # Allows optional country code, spaces, dashes, parentheses
    pattern = re.compile(r'^\+?\d{1,3}?[-\s]?\(?\d{2,4}\)?[-\s]?\d{3,4}[-\s]?\d{4}$'
    return pattern.match(phone) is not None

# Test cases
print(is_valid_phone("+91 98765 43210"))      # True
print(is_valid_phone("(022) 123-4567"))       # True
print(is_valid_phone("9876543210"))           # True
print(is_valid_phone("12345"))                # False
```

```
False
False
True
False
```

```
# To find email ID
email_regex = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[a-z]{2,7}\b'
email = "my.ownsite@our-earth.org"
valid = re.search(email_regex, email)
valid.span()
valid.group(0)
```

```
'my.ownsite@our-earth.org'
```

```
email = "prakashec1995@gmail.com"
valid = re.search(email_regex, email)
valid
valid.group(0)
'prakashec1995@gmail.com'
```

```
'prakashec1995@gmail.com'
```

```
# 250-255 25[0-5]: Match numbers from 250 to 255
# 200-249 2[0-4][0-9] Match numbers from 2oo to 249
# 100-199 1[0-1][0-9][0-9]? Match numbers from 0 to 199
# \. Matches the literal dot (.) seperating octets
# {3} Ensures there are exactly three dots in the IP address
# The pattern ensures all four octets are valid (0-255)
# re. match - check if the input string matches the pattern
# Returns True if valid, False otherwise

def is_valid_ip(ip):
    # Regular expression for validating an IPV4 Address
    patt = r'^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-
    return re.match(patt, ip) is not None

ip_address = "192.168.1.1"
if is_valid_ip(ip_address):
    print(f"{ip_address} is a valid IP address.")
```

```
else:
    print(f"{ip_address} is not a vaild Ip address")
```

```
192.168.1.1 is a valid IP address.
```

```
# what is the process of compilation and linking in Python ?

# Compilation and Linking in Python

'''
Python is an interpreted language,
but it does involve a form of compilation and linking during execution.
Here's a breakdown of the process:

1. Compilation
When you write Python code, it is saved as a .py file (source code).
During execution, Python compiles the source code into bytecode,
a lower-level, platform-independent representation.
This bytecode is stored in .pyc files
(or in the __pycache__ directory) to optimize future executions.
If the source code changes, the bytecode is regenerated.
Bytecode is not machine code
but an intermediate step that the Python interpreter can execute.

2. Linking
Unlike languages like C or C++, Python does not perform explicit linking at compile
Instead, it dynamically links components during runtime.
When a Python program imports modules or libraries,
the interpreter resolves and links these dependencies dynamically.
This is often referred to as dynamic linking.
The interpreter ensures that all functions, classes,
and variables are properly resolved and accessible during execution.
'''

# Exception handling

# unhandled exception can lead to system crashes or security vulnerabilities
# exception handled properly in code
# program can gracefully handle unexpected errors
#exception handling
        # try:
        # except
        # else
        # finally

try:
    # try block contains the code that might raise an exception,
    # NameError, IndexError, SyntaxError, FileNotFound Error,
    # ValueError, ModuleNotFound Error
    # importError, AttributeError, IndendationError, KeyError,
    # RunTimeError, StopIteration, TabError, SystemError, TypeError, ZerDivsionError
    a = 5
    print(a+1)
    print(lesson)
except NameError:
    print("lesson variable not declared")
except Exception as e: # except block allows me to catch and respond to specific typ
    print(e)
finally:
    print("Execution completed.")
```

```
6
lesson variable not declared
Execution completed.
```

```
# List comprehension


#list_comprehension = [expression sequence condition]
# To find the even numbers
@staticmethod
def list_comprehension1():
    return [i for i in range(10) if i%2==0]

# To find the odd numbers
@staticmethod
def list_comprehension2():
    return [i for i in range(10) if i%2!=0]

def tuple_comprehension1():
    return tuple(i for i in range(10) if i%2==0)

# To find the odd numbers
@staticmethod
def tuple_comprehension2():
    return tuple(i for i in range(10) if i%2!=0)

@staticmethod
def dictionary_comprehension():
    return {i:i**2 for i in range(10) if i%2==0}

@staticmethod
def set_comprehension():
    return {i for i in range(10) if i%2==0}

# Function call
print(list_comprehension1())
print(list_comprehension2())
print(tuple_comprehension1())
print(tuple_comprehension2())
print(dictionary_comprehension())
print(set_comprehension())
```

```
[0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
(0, 2, 4, 6, 8)
```

```
(1, 3, 5, 7, 9)
{0: 0, 2: 4, 4: 16, 6: 36, 8: 64}
{0, 2, 4, 6, 8}
```

```
def min_number(sequence):
    min_num = min(sequence,key=lambda x: x)
    return min_num

def max_number(sequence):
    max_num = max(sequence, key=lambda x: x)
    return max_num

def sorted_output(sequence):
    sorted_output = sorted(sequence,reverse=True)
    return sorted_output

sequence = range(20)
minimum_number = min_number(sequence)
print("min_num", minimum_number)

maximum_number = max_number(sequence)
print("max_num", maximum_number)

sorted_output = sorted_output(sequence)
print("sorted output", sorted_output)
```

```
min_num 0
max_num 19
sorted output [19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

```
# The fibonacci series is a sequence of numbers
# where each number is the sum of the two preceding ones, usally starting with 0 and
# The program to produce the fibonnic series
def generate_fibonacci(n):
    # intialize the first two Fibonacci Numbers
    fib_series = [0,1]
    # Generate Fibonacci sequence
    for i in range(2, n):
        next_fib = fib_series[-1] + fib_series[-2]
        fib_series.append(next_fib)
    return fib_series

print(generate_fibonacci(n=10))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
def test_check_leap_year(year):
    if year % 4 == 0:
        return str(year) , "is Leap year"
    else:
        return str(year) , "is Not a leap year"

test_check_leap_year(year=2000)
```

```
('2000', 'is Leap year')
```

```
# Write a program in Python to execute the buble sort algorithm ?
# Bubble sort is a straightforward sorting algorithm that repeatedly steps through t
# compares adjacent elements, and swaps them if they are in the wrong order
# 11, 7, 9, 15, 26, 23, 21
# Iteration 1 - First-second, second-third, third-fourth, fourth-fifth, fifth-sizth
# 9 is smaller than 15
# there will be no change
# 15 is smaller than 26, there will be no change
# 26 is smaller than 23, 26 is bigger, move it
# 11, 7, 9, 15, 23, 21, 26
#Iteration 2:
# The iterations will take place until we have a fully sorted list.
# sorted array
def bubble_sort(list1):
    n = len(list1)
    # Traverse through all elements in the list
    for i in range(n):
    # last elements are already in place, so the inner loop can avoid looking at the
        for j in range(0, n-i-1):
            # swap if the element found is greater than the next element
            if list1[j] > list1[j+1]:
                list1[j], list1[j+1]=list1[j+1],list1[j]
    return list1

sorted_op = bubble_sort(list1=[11, 7, 9, 15, 26, 23, 21])
print("soirted output", sorted_op)
```

```
soirted output [7, 9, 11, 15, 21, 23, 26]
```

```
#1. Find the fist , second , third and min value of the given list :
#maxx=a[0] , slarge = a[1] , tlarge = a[2] we can also take like this rather than in
def fun():
    a = [1000,2,3,4,100,500]
    maxx=a[0]
    slarge = float('-inf')
    tlarge = float('-inf')
    minn = a[0]
    for i in a[1:]:
        if i > maxx:
            tlarge = slarge
            slarge = maxx
            maxx=i
        elif i > slarge:
            tlarge = slarge
            slarge = i
        elif i > tlarge:
            tlarge = i
        if i < minn:
            minn = i
    print(maxx,slarge,tlarge,minn)
```

```
fun()
```

```
1000 500 100 2
```

```
#2. #print unique if it repeat in it's place :
inp = "abbbbbcccccdddccdd"
csub =''
out = []
for i in range(1,len(inp)):
    if inp[i-1] == inp[i]:
        csub = inp[i]
    else:
        out.append(inp[i-1])
else:
    out.append(inp[i])
print(out)
```

```
['a', 'b', 'c', 'd', 'c', 'd']
```

```
#3. convert the time to the 12 hours format with PM / AM. 24 hours converted to 12 h
from datetime import datetime

# Input time in 24-hour format
time_24 = "18:30"

# Convert to 12-hour format
time_12 = datetime.strptime(time_24, "%H:%M").strftime("%I:%M %p")
print(time_12)   # Output: 06:30 PM

#2. Using time Module
#Another approach using the time module
import time

# Input time in 24-hour format
time_24 = "23:45"

# Convert to 12-hour format
time_12 = time.strftime("%I:%M %p", time.strptime(time_24, "%H:%M"))
print(time_12)   # Output: 11:45 PM

#3. Manual Conversion
#If you want to avoid libraries, you can manually convert:
def convert_to_12_hour_format(time_24):
    hours, minutes = map(int, time_24.split(":"))
    period = "AM" if hours < 12 else "PM"
    hours = hours % 12 or 12   # Convert 0 to 12 for midnight
    return f"{hours:02}:{minutes:02} {period}"

# Input time in 24-hour format
time_24 = "00:15"
time_12 = convert_to_12_hour_format(time_24)
print(time_12)   # Output: 12:15 AM
```

```
06:30 PM
11:45 PM
12:15 AM
```

```
#4. prime numbers in list comprehension  :
prime = [i for i in range(1,11) if sum(1 for j in range(1,i+1) if i%j==0)==2]
print(prime)
```

```
[2, 3, 5, 7]
```

```
#5. Convert back aabbbccbbbbd => a2b3c2b4d1
#add space to the given string at the end for comparision.
inp = 'aabbbccbbbbd '
c=''
ans = ''

for i in range(len(inp)-1):
    if(inp[i]==inp[i+1]):
        c=c+inp[i]
    else:
        c=c+inp[i]
        ans = ans + inp[i]+str(c.count(inp[i]))
        c=''
print(ans)
```

```
a2b3c2b4d1
```

```
#6. #in this if the digit of char is not found then it only prints the result for th
#one digit nums only .

inp = 'a3b3c2b4d1'
out =''
temp =''
for i in range(1,len(inp),2):
    temp = inp[i-1]*int(inp[i])
    out = out+temp
    temp=''

print(f'The convertion of {inp} is : {out}')
```

```
The convertion of a3b3c2b4d1 is : aaabbbccbbbbd
```

```
# Is Python a compiled language or an interpreted language?
'''
Please remember one thing, whether a language is compiled
or interpreted or both is not defined in the language standard.
In other words, it is not a properly of a programming language.
Different Python distributions (or implementations)
choose to do different things (compile or interpret or both).
However the most common implementations like CPython do both compile and interpret,
but in different stages of its execution process.

Compilation: When you write Python code and run it,
the source code (.py files) is first compiled
into an intermediate form called bytecode (.pyc files).
This bytecode is a lower-level representation of your code,
but it is still not directly machine code.
It’s something that the Python Virtual Machine (PVM) can understand and execute.
Interpretation: After Python code is compiled into bytecode,
it is executed by the Python Virtual Machine (PVM), which is an interpreter.
The PVM reads the bytecode and executes it line-by-line at runtime,
which is why Python is considered an interpreted language in practice.

'''
#Difference between for loop and while loop in Python
'''
For loop: Used when we know how many times to repeat, often with lists, tuples, sets

'''
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
# While loop: Used when we only have an end condition and don’t know exactly how man
c = 0
while c < 5:
    print(c)
    c += 1
```

```
0
1
2
3
4
```

```
# What are Modules and Packages in Python?
'''
A module is a single file that contains Python code
(functions, variables, classes) which can be reused in other programs.
You can think of it as a code library.
For example: math is a built-in module that provides math functions like sqrt(), pi,

package is a collection of related modules stored in a directory.
It helps in organizing and grouping modules together for easier management.
For example: The numpy package contains multiple modules for numerical operations.
To create a package, the directory must contain a special file named __init__.py.
'''
import math
print(math.sqrt(16))
```

```
4.0
```

```
# What is a namespace in Python ?
'''
A namespace in Python refers to a container where names (variables, functions, objec
In simple terms, a namespace is a space where names are defined and stored
```

```
and it helps avoid naming conflicts by ensuring that names are unique within a given

Types of Namespaces:

Built-in Namespace: Contains all the built-in functions and
exceptions, like print(), int(), etc.
These are available in every Python program.
Global Namespace:
Contains names from all the objects, functions and variables in the program at the t
Local Namespace:
Refers to names inside a function or method. Each function call creates a new local

'''

# What are Pickling and Unpickling?
'''
Pickling: The pickle module converts any Python object into a byte stream (not a str
This byte stream can then be stored in a file, sent over a network,
or saved for later use. The function used for pickling is pickle.dump().

Unpickling: The process of retrieving the original Python object
from the byte stream (saved during pickling) is called unpickling.
The function used for unpickling is pickle.load().

'''

# What is Walrus Operator?
'''
Walrus Operator allows you to assign a value to a variable within an expression.
This can be useful when you need to use a value multiple times in a loop,
but don't want to repeat the calculation.
Walrus Operator is represented by the `:=` syntax
and can be used in a variety of contexts including while loops and if statements.
'''

# Memory Management in Python
'''
Memory management refers to process of allocating and deallocating memory to a progr
Python handles memory management automatically using mechanisms like reference count
and garbage collection, which means programmers do not have to manually manage memor

Let's explore how Python automatically manages memory using garbage collection and r

Garbage Collection
It is a process in which Python automatically frees memory occupied by objects that

If an object has no references pointing to it
(i.e., nothing is using it), garbage collector removes it from memory.
This ensures that unused memory can be reused for new objects.

Reference Counting
It is one of the primary memory management techniques used in Python, where:
```

```
Every object keeps a reference counter,
which tells how many variables (or references) are currently pointing to that object
When a new reference to the object is created, counter increases.
When a reference is deleted or goes out of scope, counter decreases.
If the counter reaches zero, it means no variable is using the object anymore,
so Python automatically deallocates (frees) that memory.

Explanation:

a and b both refer to same list in memory.
Changing b also changes a, because both share same reference.
Now that we know how references work, let’s see how Python organizes memory.
'''

a = [1, 2, 3]
b = a

print(id(a), id(b))    # Same ID → both point to same list

b.append(4)
print(a)
```

```
81207216 81207216
[1, 2, 3, 4]
```

```
# Mmeory Allocation

#Memory Allocation in Python
'''
It is the process of reserving space in a computer’s memory
so that a program can store its data and variables while it runs.
In Python, this process is handled automatically by interpreter,
but the way objects are stored and reused can make a big difference in performance.

Let's see an example to understand it better.

Example: Memory Optimization with Small Integers
Python applies an internal optimization called object interning
for small immutable objects (like integers from -5 to 256 and some strings).
Instead of creating a new object every time, Python reuses same object to save memor
Here, Python does not create two separate objects for 10.
Instead, both x and y point to the same memory location. Let's verify if it's true:

'''

x = 10
y = x

if id(x) == id(y):
    print("x and y refer to same object")
```

```
x and y refer to same object
```

```python
#Now, if we change x to a different integer:

x = 10
y = x
x += 1

if id(x) != id(y):
    print("x and y do not refer to the same object")

# When x is changed, Python creates a new object (11) for it. The old link with 10 b
```

```
x and y do not refer to the same object
```

```python
# In Python, memory is divided mainly into two parts:
'''
Stack Memory
Heap Memory

Both play different roles in how variables and objects are stored and accessed.
Stack Memory
Stack memory is where method/function calls and reference variables are stored.

Whenever a function is called, Python adds it to the call stack.
Inside this function,
all variables declared (like numbers, strings or temporary references) are stored in
Once the function finishes executing, stack memory used by it is automatically freed
In simple terms: Stack memory is temporary
and is only alive until the function or method call is running.

How it Works:

Allocation happens in a contiguous (continuous) block of memory.
Python’s compiler handles this automatically, so developers don’t need to manage it.
It is fast, but it is limited in size and scope (only works within a function call).
'''

def func():
    # These variables are created in stack memory
    a = 20
    b = []
    c = ""
print(a, b, c)
# Here a, b and c are stored in stack memory when function func() is called.
# As soon as function ends, this memory is released automatically.
'''
Heap Memory
Heap memory is where actual objects and values are stored.

When a variable is created, Python allocates its object/value in heap memory.
Stack memory stores only the reference (pointer) to this object.
Objects in heap memory can be shared among multiple functions
```

```
or exist even after a function has finished executing.
In simple terms: Heap memory is like a storage area
where all values/objects live and stack memory just keeps directions (references) to

How it Works:

Heap memory allocation happens at runtime.
Unlike stack, it does not follow a strict order it’s more flexible.
This is where large data structures (lists, dictionaries, objects) are stored.
Garbage collection is responsible for cleaning up unused objects from heap memory.
'''
# This list of 10 integers is allocated in heap memory
a = [0] * 10
print(a)
# Explanation: Here, list [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] is stored in heap memory.
# The variable a in stack memory just holds a reference pointing to this list in the
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0] [1, 2, 3, 4] 5
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```