

01 Python Interview Questions - KirkYagami - Nikhil Sharma

1. Explain the difference between `__init__` and `__new__` methods.

Answer:

In Python, `__new__` and `__init__` are two methods that are involved in object creation and initialization, but they serve different purposes.

- `__new__` : This is a static method that is responsible for creating a new instance of a class. It is called before `__init__` and returns a new instance of the class. The `__new__` method takes the class as its first argument (`cls`) and returns an instance of that class.
- `__init__` : This method is called after `__new__` has created a new instance. It initializes the instance with the parameters passed to it. The `__init__` method takes the instance as its first argument (`self`) and does not return anything.

Example:

```
class MyClass:
    def __new__(cls, *args, **kwargs):
        print("Creating instance...")
        instance = super(MyClass, cls).__new__(cls) # Create a new instance
        return instance

    def __init__(self, value):
        print("Initializing instance...")
        self.value = value

# Creating an instance of MyClass
obj = MyClass(10)
```

Output:

```
Creating instance...
Initializing instance...
```

In this example, `__new__` is responsible for the actual creation of the instance, while `__init__` initializes the instance with a value.

2. How do you implement multithreading in Python?

Answer:

Multithreading in Python can be implemented using the `threading` module, which allows you to run multiple threads (smaller units of a process) concurrently. However, it's important to note that Python's Global Interpreter Lock (GIL) can prevent true parallel execution of threads; therefore, multithreading is generally more effective for I/O-bound tasks rather than CPU-bound tasks.

Example:

```
import threading
import time

def print_numbers():
    for i in range(5):
        print(f"Number: {i}")
        time.sleep(1)

def print_letters():
    for letter in ['A', 'B', 'C', 'D', 'E']:
        print(f"Letter: {letter}")
        time.sleep(1)

# Creating threads
thread1 = threading.Thread(target=print_numbers)
thread2 = threading.Thread(target=print_letters)

# Starting threads
thread1.start()
thread2.start()

# Wait for threads to complete
thread1.join()
thread2.join()

print("Finished!")
```

Output:

```
Number: 0
Letter: A
Number: 1
Letter: B
Number: 2
Letter: C
Number: 3
```

```
Letter: D
Number: 4
Letter: E
Finished!
```

In this example, `print_numbers` and `print_letters` run concurrently in separate threads, demonstrating how multithreading can be useful for I/O-bound tasks like printing.

3. What are the built-in data types in Python?

Answer:

Python provides several built-in data types that can be categorized into different classes:

- **Numeric Types:**
 - `int`: Integer type, e.g., `x = 5`
 - `float`: Floating-point type, e.g., `y = 3.14`
 - `complex`: Complex number type, e.g., `z = 1 + 2j`
- **Sequence Types:**
 - `str`: String type, e.g., `name = "Alice"`
 - `list`: Mutable sequence, e.g., `my_list = [1, 2, 3]`
 - `tuple`: Immutable sequence, e.g., `my_tuple = (1, 2, 3)`
 - `range`: Represents a sequence of numbers, e.g., `my_range = range(5)`
- **Mapping Type:**
 - `dict`: Dictionary type for key-value pairs, e.g., `my_dict = {'a': 1, 'b': 2}`
- **Set Types:**
 - `set`: Unordered collection of unique elements, e.g., `my_set = {1, 2, 3}`
 - `frozenset`: Immutable version of a set, e.g., `my_frozenset = frozenset([1, 2, 3])`
- **Boolean Type:**
 - `bool`: Boolean type, can be `True` or `False`, e.g., `flag = True`

Example:

```
# Numeric types
x = 5          # int
y = 3.14       # float
z = 1 + 2j     # complex

# Sequence types
name = "Alice" # str
my_list = [1, 2, 3] # list
my_tuple = (1, 2, 3) # tuple
my_range = range(5) # range
```

```
# Mapping type
my_dict = {'a': 1, 'b': 2} # dict

# Set types
my_set = {1, 2, 3} # set
my_frozenset = frozenset([1, 2, 3]) # frozenset

# Boolean type
flag = True # bool
```

4. What is the difference between `==` and `is` operators?

****Answer:****

In Python, `==` and `is` are used to compare values, but they do so in different ways:

- **`==` Operator:** This operator checks for value equality. It evaluates to `True` if the values of two objects are the same, regardless of whether they are the same object in memory.
- **`is` Operator:** This operator checks for identity. It evaluates to `True` if both operands point to the same object in memory.

Example:

```
a = [1, 2, 3]
b = a # b points to the same list as a
c = a[:] # c is a new list that is a copy of a

print(a == b) # True, values are the same
print(a is b) # True, both point to the same object

print(a == c) # True, values are the same
print(a is c) # False, c is a different object
```

In this example, `a` and `b` refer to the same list, while `c` is a separate copy of `a`.

5. What is the purpose of the `if __name__ == "__main__":` statement?

Answer:

The statement `if __name__ == "__main__":` is used to determine whether a Python script is being run as the main program or if it is being imported as a module in another script. When a Python file is executed, Python sets the special variable `__name__` to `"__main__"` in that file. If the file is imported as a module, `__name__` will be set to the module's name.

This construct allows you to include code in a module that only runs when the module is executed directly, making it possible to provide module-level test code without executing it when the module is imported.

Example:

```
def main():  
    print("This is the main function.")  
  
if __name__ == "__main__":  
    main()
```

If you run this script directly, it will print "This is the main function." However, if you import this script in another module, the main function won't execute automatically.

6. What is the difference between a shallow copy and a deep copy?

Answer:

In Python, copying objects can be done in two ways: shallow copy and deep copy.

- **Shallow Copy:** A shallow copy creates a new object but does not create copies of nested objects within the original object. It only copies references to those nested objects. Changes made to nested objects in the copied object will reflect in the original object.
- **Deep Copy:** A deep copy creates a new object and recursively copies all objects found within the original object. Changes made to nested objects in the copied object do not affect the original object.

You can use the `copy` module to create shallow and deep copies.

Example:

```
import copy  
  
original_list = [[1, 2, 3], [4, 5, 6]]  
  
# Creating a shallow copy  
shallow_copied_list = copy.copy(original_list)
```

```
# Creating a deep copy
deep_copied_list = copy.deepcopy(original_list)

# Modifying the original list
original_list[0][0] = 'Changed'

print("Original List:", original_list)          # [['Changed', 2, 3], [4, 5, 6]]
print("Shallow Copied List:", shallow_copied_list) # [['Changed', 2, 3], [4, 5, 6]]
print("Deep Copied List:", deep_copied_list)    # [[1, 2, 3], [4, 5, 6]]
```

In this example, modifying `original_list` affects `shallow_copied_list` but not `deep_copied_list`.

7. Explain the use of decorators in Python.

Answer:

Decorators in Python are

a way to modify or enhance functions or methods without changing their code. They are often used for tasks like logging, enforcing access control, instrumentation, or modifying function behavior. A decorator is a function that takes another function as an argument, extends its behavior, and returns a new function.

Example:

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

Output:

```
Something is happening before the function is called.
Hello!
Something is happening after the function is called.
```

In this example, `my_decorator` adds behavior before and after the `say_hello` function is called. The `@my_decorator` syntax is a shorthand for `say_hello = my_decorator(say_hello)`.

8. What is the purpose of `self` in class methods?

Answer:

In Python classes, `self` refers to the instance of the class itself. It is a convention to name the first parameter of instance methods `self`, allowing access to the attributes and methods of the class. Using `self`, you can refer to instance variables and call other methods from within class methods.

Example:

```
class Dog:
    def __init__(self, name):
        self.name = name # Assigning the instance variable

    def bark(self):
        print(f"{self.name} says woof!")

# Creating an instance of Dog
my_dog = Dog("Buddy")
my_dog.bark() # Output: Buddy says woof!
```

In this example, `self.name` refers to the `name` attribute of the `Dog` instance, allowing us to access it in the `bark` method.

9. How do you handle exceptions in Python?

Answer:

In Python, exceptions can be handled using the `try`, `except`, `else`, and `finally` blocks. This allows for graceful error handling and resource cleanup.

- `try` : The block of code that may raise an exception.
- `except` : This block catches and handles the exception.
- `else` : This block runs if the `try` block does not raise an exception.
- `finally` : This block runs regardless of whether an exception occurred or not, typically used for cleanup actions.

Example:

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
```

```

except ValueError:
    print("That's not a valid number!")
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print(f"Result: {result}")
finally:
    print("Execution complete.")

```

In this example, the program handles `ValueError` for invalid input and `ZeroDivisionError` for division by zero, ensuring that it gracefully exits without crashing.

10. What is polymorphism in Python?

Answer:

Polymorphism in Python refers to the ability of different classes to be treated as instances of the same class through a common interface. It allows methods to be defined in different ways based on the object it is acting upon. Polymorphism is often implemented via method overriding or duck typing.

Example:

```

class Cat:
    def sound(self):
        return "Meow"

class Dog:
    def sound(self):
        return "Woof"

def make_sound(animal):
    print(animal.sound())

# Creating instances
cat = Cat()
dog = Dog()

# Using polymorphism
make_sound(cat) # Output: Meow
make_sound(dog) # Output: Woof

```

In this example, both `Cat` and `Dog` classes implement the `sound` method, and the `make_sound` function can take any object that has a `sound` method, demonstrating polymorphism in action.

These explanations and examples should provide a comprehensive understanding of the interview questions related to Python.

1. Write a python code to accept a string and count the number of vowels and consonants.

```
#taking the input string
inp= input('Enter the string : ')
#initializing counters for vowels and consonants
v=0
c=0
#running a for loop to check all the characters of the input string
for i in inp:
    #checking if the character is a vowel, then increasing v by 1
    if i in ['a','e','i','o','u']:
        v=v+1
    #checking if the character is a space, then pass and do nothing
    elif i == " ":
        pass
    #checking if the character is a consonant, then increasing c by 1
    else :
        c=c+1
#print the values of v and c
print( 'The number of vowels is ', v)
print( 'The number of components is ', c)
```

The output for the above code will look like this:

```
Enter the string : Sweta Shaw
The number of vowels is  3
The number of components is  6
```

2. Write a program to find the number of occurrences of elements from list1 to list2.

```
#initializing list1 and list2, instead of this input can also be taken from user
list1=[1,2,3,4,5]
list2= [2,8,9,4,5]
#initializing a counter for occurrence
occ=0
#running a for loop on list1 to compare with list2
for i in list1:
```

```

#checking if the element of list1 is in list2, then increasing the occ counter by
1
if i in list2:
    occ+=1
#printing the value of occ counter
print( occ, 'elements of list1 are in list2')

```

The output of the above code will look like this:

```

3 elements of list1 are in list2

```

3. Write a function that returns the lesser of two given numbers if both numbers are even, but returns the greater if one or both numbers are odd.

```

#defining the function which takes two numbers
def lesorgrt(a,b):
#checking if a and b are even, then returning the minimum of a and b.
    if (a%2==0 and b%2==0):
        return min(a,b)
#if the sum is not even then returning the maximum of a and b
    else:
        return max(a,b)
#calling the function with input 2 and 4
lesorgrt(2,4)

```

The output of the above code will be 2:

4. Write a python function that accepts a string and calculates the number of upper case letters and lower case letters.

```

#taking the input string
inp = input('Enter the string : ')
#initializng the counters for upper and lower letters
u=0
l=0
#running a for loop for all characters of the input
for i in inp:
    #checking if the character is space, then do nothing and pass

```

```

if i == " ":
    pass
#checking if the character is in upper case, then increasing u by 1
elif i.isupper():
    u+=1
#checking if the character is in lower case, then increasing l by 1
else:
    l+=1
#printing the values of u and l
print('The number of upper characters is ', u)
print('The number of lower characters is ', l)

```

The output of the other code will look like this:

```

Enter the string : Sweta Shaw
The number of upper characters is  2
The number of lower characters is  7

```

5. Write a program that accepts input in this form: s3t1z5. Here any character is followed by a number. The program should return a string where the character is repeated for the corresponding number of times.

```

#initializing a blank result string
res=""
#taking the input string
inp= input("Enter the string : ")
#running a for loop to access the characters
for i in range(0,len(inp),2):
    #running a for loop to access the numbers
    for j in range(int(inp[i+1])):
        #appending the ith string to res for j number of times
        res=res+inp[i]
#printing the result
print(res)

```

The output of the above code will look like this:

```

Enter the string : s3t1z5
ssstzzzzz

```

6. Take a sentence as input and print only the words that start with "s" in the sentence.

```
#taking the input string
inp=input('Enter the string: ')
#splitting the sentence by a space, result will be a list of words in the sentence.
inp=inp.split(" ")
#running a for loop over all words in the sentence
for i in inp:
    #checking if the word starts with s, then printing the word
    if i.lower().startswith('s'):
        print(i, end=" ")
```

The output of the above code will look like this:

```
Enter the string: Sweta Shaw loves to stay in Spain
Sweta Shaw stay Spain
```

7. From a dictionary, print only the items with an odd value of less than 45.

```
#initializing the dictionary with some key names and values
dic={'Sweta': 45, 'Mausam': 31, 'Pratik': 312, 'Pari': 400}
#running a for loop over all the key-value pairs of dic
for i,j in dic.items():
    #checking if the value is odd and less than 45, then printing the key and value
    if (j%2!=0) and (j<45):
        print(i,':', j)
```

The output of the above code will look like this:

```
Mausam : 31
```

8. Write a program to count pairs of elements of a list with a given sum

```
#initializing the list
l=[1,2,3,4,5,4,3,2,1]
#initializing the sum values, this can be taken as input from user as well
sum=6
```

```

#initializing the counter for pairs as 0
c=0
#running a for loop for index of list l
for i in range(0,len(l)):
    #running a for loop from i+1 to end of list
    for j in range(i+1, len(l)):
        #checking if the sum of ith and jth element is equal to sum, then increase c
        by 1
        if l[i]+l[j]==sum:
            c=c+1
#printing the value of c
print('there are total ', c, ' number of pairs with given sum')

```

The output of the above code will look like this:

```
there are total 7 number of pairs with given sum
```

9. Write a python program to find the common characters from two input strings.

```

#taking both the strings as input
s1=input("Enter first string:")
s2=input("Enter second string:")
#finding the common characters using the & connector
a=list(set(s1)&set(s2))
#printing the characters by running a for loop
print("The common letters are:")
for i in a:
    print(i)

```

The output of the above code looks like this:

```

☞ Enter first string:Sweta
Enter second string:Shaw
The common letters are:
S
W
a

```

10. Swap two numbers by using one line of code.

```

#taking the two numbers as input
a=input("Enter value of a: ")
b=input("Enter value of b: ")
#storing value of a in value of b and value of b in value of a
a,b=b,a
#printing new values of a and b
print('New value of a is', a)
print('New value of b is', b)

```

The output of the above code looks like this:

```

❏ Enter value of a: 5
   Enter value of b: 4
   New value of a is 4
   New value of b is 5

```

11. Write a program to take two lists and check if the product of even position elements from the first list is completely divisible by the sum of odd position elements from the second list.

```

#initializing two lists
l1=[1,4,5,7,9,0]
l2=[3,4,6,8,19]
#initializing the product variable as 1 and sum variable as 0
pr=1
sm=0
#running a for loop over the index and elements of l1
for i,j in enumerate(l1):
    #checking if the index is even, then updating the pr variable with product of pr
    and element at ith location
    if i%2==0:
        pr*=j
#running a for loop over the index and elements of l2
for i,j in enumerate(l2):
    #checking if the index is odd, then updating the sm variable with sum of sm and
    element at ith location
    if i%2!=0:
        sm+=j
#checking if pr is completely divisible by sm, then print so

```

```

if(pr%sm==0):
    print('Completely Divisible')
#else print not completely divisible
else:
    print('Not completely divisible')

```

★★

The output of the above code will be 'Not completely divisible'

12. Write a function that will take a string and a character as arguments and check if the character is present in the string then it will be replaced with a “*” .

```

#defining the function with two parameters
def rep(string, chr):
    #replacing the character in the string with a '*'
    return string.replace(chr, '*')
#calling the function and printing the result
res=rep('Sweta Shaw', 'a')
print(res)

```

☞ Swet* Sh*w

13. Write a program to delete vowels from a string.

```

#creating a list of vowels
vowels=['a','e','i','o','u']
#initializing a new string variable with none
new_str=''
#taking the input string
old_str=input('Enter the input: ')
#running the for loop over the input string
for i in old_str:
    #checking if element is a not a vowel, then appending in new_str
    if i not in vowels:
        new_str+=i

```

```
#printing the result
print('The result without the vowels is: ', new_str)
```

The output of the above code looks like this:

```
Enter the input: My name is Sweta Shaw.
The result without the vowels is:  My nm s Swt Shw.
```

14. Write a program to count the highest occurring/maximum frequency character in a string.

```
#taking the string as input
inp=input('Enter the input: ')
#initializing a blank dictionary which will be used to hold the elements of input as
key and their frequency as corresponding value
dic={}
#running a for loop over input string
for i in inp:
    #checking if the element is already a key in dic, then increacing corresponding
value by 1
    if i in dic:
        dic[i]+=1
    #else assigning 1 to the new key
    else:
        dic[i]=1
#extracting the key with maximum value and printing the result
res= max(dic, key=dic.get)
print('The maximum occuring character is', res)
```

The output of the above code will look like this:

```
Enter the input: Sweta S
The maximum occuring character is S
```

15. Write a program to find the second-highest number in a list.

```
#initializing the list
l=[12,56,89,93,25]
```



```
#printing the second last element of the sorted list(in ascending order)
print('The second highest is: ', sorted(l)[-2])
```

The output of the above code looks like this:

The second highest is: 89

16. Write a program to take a list as input and print another list with only the unique values of the input list.

```
#initializing a blank list
l=[]
#taking the list as input
print('Enter the elements of a list one by one and click on "e" to end the process')
#running an infinite loop
while True:
    #taking the input
    i=input('Enter input: ')
    #checking if the input is 'e', then breaking the loop
    if i.lower() == 'e':
        break
    #else appending the input to the list l
    else:
        l.append(i)
#printing the input list
print('The input list is', l)
#printing the resultant list which is a sorted list of set of l. A set only redeems
the unique values in a list
print('The new list is', sorted(list(set(l))))
```

The output of the above code looks like this:

```
➞ Enter the elements of a list one by one and click on "e" to end the process
Enter input: 1
Enter input: 1
Enter input: 4
Enter input: 6
Enter input: 8
Enter input: 2
Enter input: 4
Enter input: 3
Enter input: 3
Enter input: 2
Enter input: e
The input list is ['1', '1', '4', '6', '8', '2', '4', '3', '3', '2']
The new list is ['1', '2', '3', '4', '6', '8']
```

17. Implement the `unique_names` method. When passed two lists of names, it will return a list containing the names that appear in either or both lists. The returned list should have no duplicates.

```
#defining the function with two parameters
def unique_names(l1,l2):
    #concatenating elements of list1 and list2
    l3=l1+l2
    #finding the set of list3 which will only have unique elements and converting it
    into list
    l3=list(set(l3))
    #returning the list
    return l3

#calling the function with two lists and printing the result
l4=unique_names(['Sweta', 'Shubham', 'Pari'], ['Anish', 'Sweta', 'Amrita'])
print('The resultant list is', l4)
```

The output of the above code will look like this:

```
➞ The resultant list is ['Pari', 'Amrita', 'Shubham', 'Anish', 'Sweta']
```

18. Implement a function that accepts a dictionary containing the file owner name as the value for each file name as key and returns

a dictionary containing a list of file names as value for each owner name as the key, in any order.

```
#defining the function with one parameter
def group_by_owners(files):
    #initializing a blank dictionary
    inv = {}
    #running a for loop over keys and values of input dictionary
    for k, v in files.items():
        #setting the value as a key and initializing the value as a blank list
        keys = inv.setdefault(v, [])
        #print(inv)
        #appending the keys as values in the above key_value pair
        keys.append(k)
    #returning the created dictionary
    return inv
#initializing the input dictionary
dictionary= {'file1.jpeg': 'Sweta', 'file2.py': 'Shaw', 'file3.txt': 'Sweta'}
#calling the function with input dictionary
dic1=group_by_owners(dictionary)
#printing the results
print('The input dictionary is', dictionary)
print('The resultant dictionary is', dic1)
```

The output of the above code looks like this:

```
➤ The input dictionary is {'file1.jpeg': 'Sweta', 'file2.py': 'Shaw', 'file3.txt': 'Sweta'}
  The resultant dictionary is {'Sweta': ['file1.jpeg', 'file3.txt'], 'Shaw': ['file2.py']}
```

19. Implement a function to take a list of numbers as an argument and return a list that will have a 0 for every corresponding number of the input list if the number is not perfect, otherwise 1.

A perfect number is a positive integer that is equal to the sum of its proper divisors.

```
#defining the function with one parameter
def lper(lst):
    #initializing a blank list which will hold 0s and 1s
    newList=[]
    #running a loop over input list
```

```

for i in lst:
    #initializing the sum variable as 0
    sum=0
    #running a for loop over integers from 1 to i-1
    for j in range(1,(i-1)):
        #checking if the ith element is divisible by any j value, then adding in sum
        if(i%j==0):
            sum+=j
    #checking if sum is equal to the ith element, then appending 1 to the resultant
list
    if(sum==i):
        newList.append(1)
    #otherwise appending 0 to the resultant list
    else:
        newList.append(0)
#returning the resultant list
return newList
#initializing the input list and calling the function over it
l=[143,563,9,365,496,8128]
k=lper(l)
#printing the result
print('The input list is', l)
print('The resultant list is', k)

```

The output of the above code looks like this:

```

The input list is [143, 563, 9, 365, 496, 8128]
The resultant list is [0, 0, 0, 0, 1, 1]

```

20. Implement a function that accepts 2 lists. One with the values and the other with corresponding integer values and returns a list with the elements of list1 arranged in increasing order of corresponding list2 elements.

```

#defining the function with two parameters
def sort_list(list1, list2):
    #zipping the two lists together keeping the second list before the first one
    zipped_pairs = zip(list2, list1)
    #for elements in sorted zipped_pairs, taking only the element from second list and
storing in a new list
    z = [x for _, x in sorted(zipped_pairs)]
    #returning the new list
    return z
#initializing the two input lists lists
x = ["red", "blue", "cyan", "green", "purple", "black", "blue", "orange", "brown"]
y = [ 0,   1,   1,   0,   1,   2,   2,   0,   1]
#calling the function with x & y and printing the result

```

```
res=sort_list(x, y)
print('The sorted list is\n', res)
```

The output of the above function looks like this:

```
☞ The sorted list is
   ['green', 'orange', 'red', 'blue', 'brown', 'cyan', 'purple', 'black', 'blue']
```

21. Implement a function that accepts a list and a string. For every element of the list having one or more characters similar to the string, the function returns a list of their index.

```
#defining the function with two arguments
def lst_ind(l1, str1):
    #initializing a blank list
    l2=[]
    #running a for loop over index and element of input list
    for n,i in enumerate(l1):
        #running a for loop over characters of i element
        for j in i:
            #checking if the character is present in input string, then appending the
            index of i element to l2
            if j in str1:
                l2.append(n)
                break
    #returning the resultant list
    return l2
#calling the function with input list and string
res=lst_ind(['Sweta', 'Joy', 'Shaw', 'Spain', 'Fly', 'India'], 'Sweta')
#printing the result
print('The result is', res)
```

The output of the above code looks like this:

```
☞ The result is [0, 2, 3, 5]
```

22. Implement a function that accepts a list and returns a corresponding list where the

element is 1 if the element in list1 is a palindrome, otherwise 0.

```
#defining a function with one parameter
def lst_pal(list1):
    #initializing a blank list
    result=[]
    #running a for loop over input list
    for l in list1:
        #reversing the element
        k=l[::-1]
        #checking if element and reversed element are equal, then appending 1 to the
        resultant list
        if k==l:
            result.append(1)
        #else appending 0 to the resultant list
        else:
            result.append(0)
    #returning the resultant list
    return result
#initializing the input list
ll=['MADAM', 'SWETA', 'KAYAK']
#calling the function and printing the result
lk=lst_pal(ll)
print('The input list is', ll)
print('The output list is', lk)
```

The output of the above code looks like this:

```
The input list is ['MADAM', 'SWETA', 'KAYAK']
The output list is [1, 0, 1]
```

23. Implement a function that takes a list as input and returns the absolute difference between values in the even index and odd index.

```
#initializing the function to take one parameter
def sumdiff(lst):
    #initializing even sum, odd sum and difference variables as 0
    esum,osum,dif=0,0,0
    #running a for loop over index and elements of input list
    for n,i in enumerate(lst):
        #checking if index is even, then adding i to esum
```

```

if(n%2==0):
    esum+=i
#else adding i to osum
else:
    osum+=i
#finding the absolute difference between esum and osum and returning it
dif=abs(esum-osum)
return dif
#initializing the input list
kk=[3, 6, 34, 12, 6, 8, 8, 5, 6, 8]
#calling the function and printing the result
res=sumdiff(kk)
print('Entered list is\n', kk)
print('The result is', res)

```

The output of the above code looks like this:

```

❏ Entered list is
    [3, 6, 34, 12, 6, 8, 8, 5, 6, 8]
    The result is 18

```

24. Implement a function that prints a triangle with a given number of rows. `#rows` to be taken as argument.

```

#defining the function with one argument
def tri(n):
    #initializing the number of front spaces of a row as n-1
    k=n-1
    #running a for loop over number of rows
    for i in range(0,n):
        #running a for loop over number of spaces and printing a space
        for j in range(0,k):
            print(end=" ")
        #decreasing the number of space by 1 for next row
        k=k-1
        #running a for loop over number of stars in a row
        for j in range(0,i+1):
            #printing a star with a space
            print("* ", end="")
        #printing a carriage return to send the pointer to start of next line
        print("\r")
    #taking the number of rows as input and calling the function with it

```

```
n=int(input("Enter the number of layers: "))
tri(n)
```

The output of the above code looks like this:

➞ Enter the number of layers: 12

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

25. Write a program to reverse an integer in Python.

```
def reverse_int(i):
    new_i =0
    while i>0:
        ld=i%10
        new_i = new_i*10+ld
        i = i//10
    return new_i

print(f'The number after reversal is {reverse_int(12345)}')
```

The output of the above code looks like this:

➞ The number after reversal is 54321

26. Write a function which checks a list of numbers. If the number is an armstrong number, it returns 1 else returns 0 in a new list.

```
def armstrong(n):
    p = len(str(n))
    prev_n = n
    sum = 0
    while n>0:
        d = n%10
        sum+=(d**p)
        n=n//10
    if sum == prev_n:
        return 1
    else: return 0

l = [1,153, 89, 370, 1634]
new_l = [armstrong(i) for i in l]
print(new_l)
```

The output of the above code looks like this:

```
[1, 1, 0, 1, 1]
```

27. Write a function to check if a number is a prime or not.

```
def primeornot(n):
    prime = True
    for i in range(2,n):
        if n%i==0:
            prime = False
            break
    return prime

primeornot(24)
```

The output of the above code looks like this:




False

28. FizzBuzz problem.

Given an integer n , for every integer $i \leq n$, print FizzBuzz if i is divisible by 3 and 5, Fizz if i is divisible by 3, Buzz if i is divisible by 5 and i if none of the conditions holds true.

```
def fizzbuzz(n):  
    for i in range(1, n+1):  
        op = ''  
        if i%3==0:  
            op+='Fizz'  
        if i%5==0:  
            op+='Buzz'  
        print(op or i)  
fizzbuzz(30)
```

The output of the above code looks like this:



```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
Fizz
22
23
Fizz
Buzz
26
Fizz
28
29
FizzBuzz
```

29. Write a recursive function to print the Fibonacci series till n^{th} term.

```
def fib(n):
    if n==1 or n==2 :
        return 1
    else:
        return fib(n-1)+fib(n-2)
for i in range(1, 11):
    print(fib(i))
```

The output of the above code looks like this:

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

I am also providing the code without recursion.

```
a, b = 1,1
print(a,b, end=' ')
for i in range(3, 31):
    a,b = b,a+b
    print(b, end=' ')
```

The output will be the same.

30. Write a recursive function which checks if a number is palindrome or not.

```
def pal(n):
    n=str(n)
    f=n[0]
    l=n[-1]
    m=n[1:-1]
    if f==l:
        if len(m)==0:
            return True
        else:
            return pal(m)
    else:
        return False

print(pal(1234321))
```

The output of the above code looks like this:

 True

31. Write a function to check if a number is binary or not.

```
def check_binary(n):  
    n=set(str(n))  
    b={'0', '1'}  
    bf=True  
    for i in n:  
        if i not in b:  
            return False  
    return True  
check_binary(1130011)
```

The output of the above code looks like this:

 False

32. Write a function to find the GCD of 2 numbers

```
def gcd(a,b):  
    if a==0:  
        return a  
    elif b ==0:  
        return b  
    elif a==b:  
        return a  
    elif a>b:  
        return gcd(a-b,b)  
    else:  
        return gcd(a,b-a)  
  
gcd(25,35)
```

The output of the above code looks like this:

 5

32. Write a function to find the LCM of 2 numbers.

```
def lcm(x,y):
    if x>y : g=x
    else: g=y
    while True:
        if(g%x)==0 and (g%y ==0):
            lcm = g
            break
        g=g+1
    return lcm

lcm(12,20)
```

The output of the above code looks like this:


60

33. Write a function to remove the repeated occurrence of characters from a string.

```
def removerepeated(st):
    s=set()
    r=set()
    o=''
    for i in st:
        if i not in s:
            s.add(i)
            o+=i
        else: r.add(i)
    return o

removerepeated('swetashaw')
```

The output of the above code looks like this:

 'swetah'

34. Write a function to remove duplicate elements from a list.

```
def remove_duplicate(l):
    s=set()
```

```

rl=[]
for i in l:
    if i not in s:
        s.add(i)
        rl.append(i)
    else:
        pass
return rl

remove_duplicate([-1,2,3,4,5,6,7,8,-9,-10, -10, 9, 8])

```

The output of the above code looks like this:

```

➞ [-1, 2, 3, 4, 5, 6, 7, 8, -9, -10, 9]

```

35. Write a function which takes a list l and a value t and returns a list of tuples with pairs of integers from l whose sum is equal to t.

```

def return_pair(l,t):
    result = []
    s={}
    for n,i in enumerate(l):
        if (t-i) not in s:
            s[i]=n
        else:
            result.append((t-i, i))
    return result
return_pair([1,2,3,4,5,6,7,8,9,10], 9)

```

The output of the above code looks like this:

```

➞ [(4, 5), (3, 6), (2, 7), (1, 8)]

```

36. Write a function to return the largest and the smallest number from a list.

```

def largest_smallest(l):
    mx, mn = max(l[0], l[1]), min(l[0], l[1])
    for i in l[2:]:
        if i > mx:

```

```

    mx=i
    elif i<mn:
        mn = i
    return mx, mn
largest_smallest([-1,2,3,4,5,6,7,8,9,-10])

```

The output of the above code looks like this:

```

➞ (9, -10)

```

37. Write a function to return the second-highest number from a list.

```

def second_highest(l):
    mx, smx = max(l[0], l[1]), min(l[0], l[1])
    for i in l[2:]:
        if i>mx:
            smx=mx
            mx=i
        elif i>smx and i!=mx:
            smx=i
        elif smx==mx and i!=smx:
            smx =i
    return smx
second_highest([-1,2,3,4,5,6,7,8,9,-9,-10])

```

The output of the above code looks like this:

```

➞ 8

```

38. Write a function to sort a list without using any built-in sorting function.

```

def sort(l):
    for i in range(len(l)):
        for j in range(i+1, len(l)):
            if l[i]>l[j]: l[i], l[j] = l[j], l[i]
    return l
sort([1,2,3,1,2,7,4,5,3,1])

```

The output of the code looks like this:


```
➞ [1, 1, 1, 2, 2, 3, 3, 4, 5, 7]
```

39. Write a function to find the largest even and largest odd number from a list.

```
def largest_even_odd(l):  
    le, lo = -float('inf'), -float('inf')  
    for i in l:  
        if i%2==0 and i>le:le=i  
        elif i%2!=0 and i>lo: lo=i  
    return le, lo  
  
largest_even_odd([1,2,3,1,2,7,4,5,3,1])
```

The output of the code looks like this:

```
➞ (4, 7)
```

40. Write a function to find all pairs of consecutive odd positive integers smaller than 'a' with the sum greater than 'b'.

```
def find_pairs(a,b):  
    res=[]  
    for i in range(1,a-2,2):  
        if i+i+2>b:  
            res.append((i,i+2))  
    return res  
find_pairs(60,100)
```

The output of the code looks like this:

```
➞ [(51, 53), (53, 55), (55, 57), (57, 59)]
```

41. Write a function which finds the sum of all odd digits and the sum of all even digits from a list of numbers.

```
def sum_even_odd_digits(l):
    se,so=0,0
    for num in l:
        cnum=num
        while cnum>0:
            d=cnum%10
            cnum=cnum//10
            if d%2==0: se+=d
            else: so+=d
    return se, so

sum_even_odd_digits([345, 893, 1948, 34, 2346])
```

The output of the above code looks like this:

```
➞ (40, 36)
```

42. Write a function which finds the consecutive ranges of 'k' greater than size 'n' in a list.

```
def consecutive_range(l,k,s):
    res=[]
    st,en = 0,0
    for i,n in enumerate(l):
        if i < en :
            continue
        if n == k:
            st = i
            for j in range(i+1, len(l)):
                if l[j]!=k: break
            en =j-1
            if(en-st>=s-1):
                res.append((st,en))
    return res

consecutive_range(
l=[2, 6, 6, 6, 6, 5, 4, 6, 6, 8, 4, 6, 6, 6, 2, 6],
k=6,
s=3)
```

The output of the above code looks like this:

➞ [(1, 4), (11, 13)]

43. Write a function which extracts the index range of consecutive similar elements from a list.

```
def consecutive_sim(l):
    idx=0
    res=[]

    while idx<len(l):
        start = idx
        val = l[idx]
        while idx<len(l) and l[idx]==val:
            idx+=1
        end=idx-1
        res.append((val,start,end))
    return res

consecutive_sim([2, 3, 3, 3, 8, 8, 6, 7, 7])
```

The output of the above code looks like this:

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

44. Write a function which extracts pairs of the same integers from a list

```
def same_int(l):
    v=set()
    res=[]
    for t in test_list:
        if t in v:
            res.append((t,t))
            v.remove(t)
        else:
            v.add(t)
    return res

same_int([4, 4, 4, 4, 4, 4, 6, 7, 4, 2, 6, 2, 8, 8, 8])
```

The output of the above code looks like this:

```
➡ [(4, 4), (4, 4), (4, 4), (6, 6), (2, 2), (8, 8)]
```

45. Write a function which takes a list of numbers and returns a list of tuples with two values in it with the first value as a number from the list and the second value as the number of times the first value is repeated consecutively in the input list.

```
def consecutive_sim_frequency(tl):
    res=[]
    idx=0
    while idx<len(tl):
        st=idx
        val=tl[idx]
        while idx<len(tl) and tl[idx]==val:
            idx+=1
        count=idx-st
        res.append((val,count))
    return res
consecutive_sim_frequency([2, 2, 3, 3, 3, 3, 4,2])
```

The output of the above code looks like this:

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

46. Write a function to get the index of the first element greater than K in a list.

```
def first_element(l,k):
    for i,n in enumerate(l):
        if n>k:
            return i
    return None
first_element([0.4, 0.5, 11.2, 8.4, 10.4],0.6)
```

The output of the above code looks like this:

47. Write a function to implement binary search

```
def bsearch(li, n, start = 0):  
    l=0  
    r=len(li)-1  
    mid=(l+r)//2  
    if n>li[mid]:  
        return bsearch(li[mid+1:],n,start+mid+1)  
    elif n<li[mid]:  
        return bsearch(li[:mid],n,start=start)  
    else:  
        return start+mid  
bsearch([1,2,3,4,5,6,7,8,9],8)
```

The output of the above code looks like this:

 7