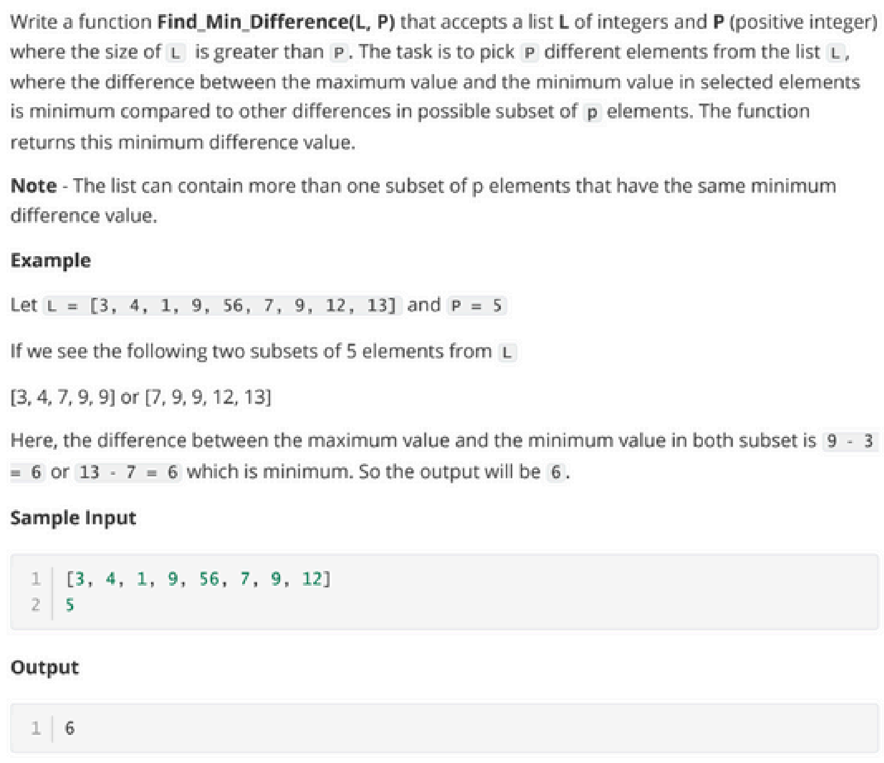
GRPA-1



Write a function **Find_Min_Difference(L, P)** that accepts a list **L** of integers and **P** (positive integer) where the size of `L` is greater than `P`. The task is to pick `P` different elements from the list `L`, where the difference between the maximum value and the minimum value in selected elements is minimum compared to other differences in possible subset of `p` elements. The function returns this minimum difference value.

Note - The list can contain more than one subset of p elements that have the same minimum difference value.

Example

Let `L = [3, 4, 1, 9, 56, 7, 9, 12, 13]` and `P = 5`

If we see the following two subsets of 5 elements from `L`

[3, 4, 7, 9, 9] or [7, 9, 9, 12, 13]

Here, the difference between the maximum value and the minimum value in both subset is `9 - 3 = 6` or `13 - 7 = 6` which is minimum. So the output will be `6`.

Sample Input

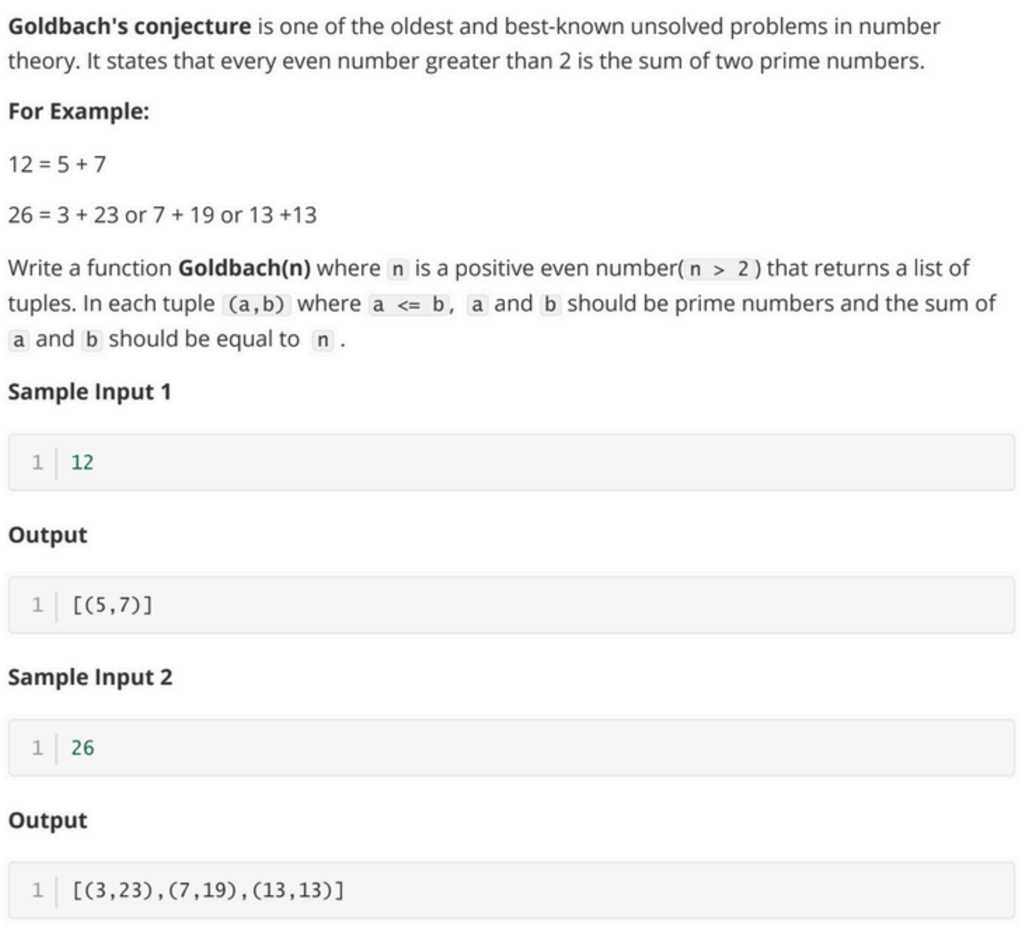
```
[3, 4, 1, 9, 56, 7, 9, 12]
5
```

Output

```
6
```

```
def find_Min_Difference(L, P):
    #Sort the list in ascending order
    L.sort()

    #Initialize the minimum difference to positive infinity
    min_difference = float('inf')

    #Iterate through all possible starting indices of subsets
    for i in range(len(L) - P + 1):
        #Calculate the difference between the maximum and minimum
values in the current subset
        current_difference = L[i + P - 1] - L[i]

        # Step 5: Update the minimum difference if the current difference
is smaller
        min_difference = min(min_difference, current_difference)

    # Step 6: Return the minimum difference found
    return min_difference


L=eval(input().strip())
P=int(input())
print(find_Min_Difference(L,P))
```

GRPA-2



Goldbach's conjecture is one of the oldest and best-known unsolved problems in number theory. It states that every even number greater than 2 is the sum of two prime numbers.

For Example:

12 = 5 + 7

26 = 3 + 23 or 7 + 19 or 13 +13

Write a function **Goldbach(n)** where `n` is a positive even number(`n > 2`) that returns a list of tuples. In each tuple `(a,b)` where `a <= b`, `a` and `b` should be prime numbers and the sum of `a` and `b` should be equal to `n`.

Sample Input 1

```
12
```

Output

```
[(5,7)]
```

Sample Input 2

```
26
```

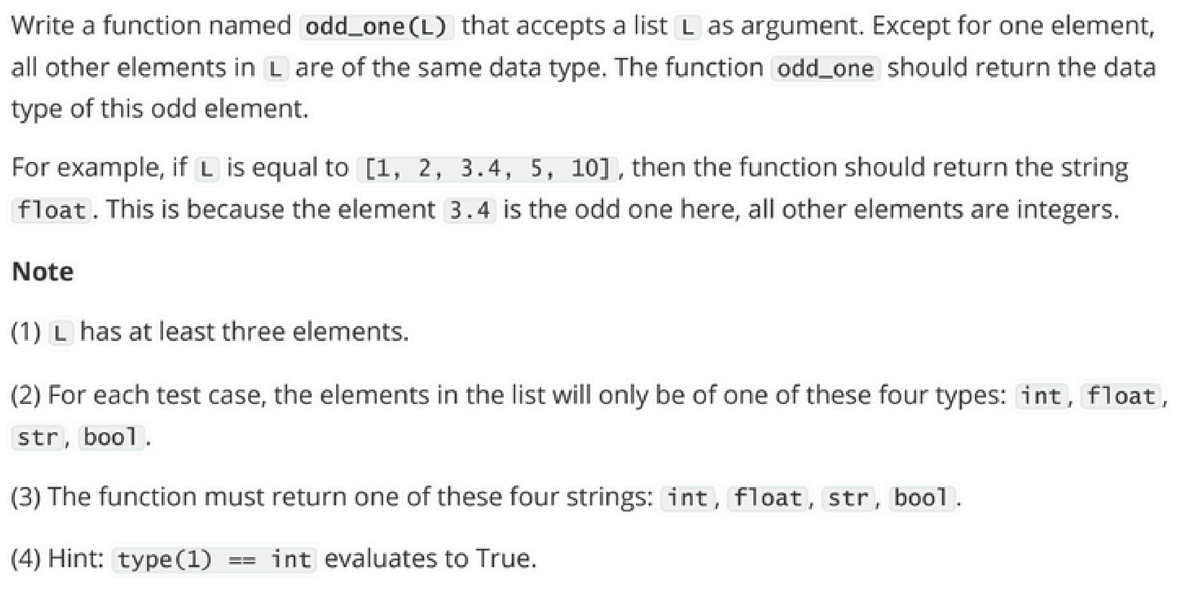
Output

```
[(3,23),(7,19),(13,13)]
```

```
#every number greater than 2 is the sum of two prime numbers

def factors(n):
   factor_list = []
   for i in range(1,(n+1)):
      if n % i == 0:
         factor_list.append(i)
   return factor_list

def prime1(temp1, n):
   if temp1 == [1, n]:
      return True
   else:
      return False
```

```
def Goldbach(n):
    list1 = []
    for i in range(3,n+1):
        if prime1(factors(i), i):
            list1.append(i)
    k =[]
    for i in list1:
        for j in list1:
            if j + i == n:
                if i <= j:
                    k.append((i, j))
                # if j <= i:
                #    k.append((j, i))
    return k

n=int(input())
print(sorted(Goldbach(n)))
```

GRPA-3



Write a function named `odd_one(L)` that accepts a list `L` as argument. Except for one element, all other elements in `L` are of the same data type. The function `odd_one` should return the data type of this odd element.

For example, if `L` is equal to `[1, 2, 3.4, 5, 10]`, then the function should return the string `float`. This is because the element `3.4` is the odd one here, all other elements are integers.

Note

(1) `L` has at least three elements.

(2) For each test case, the elements in the list will only be of one of these four types: `int`, `float`, `str`, `bool`.

(3) The function must return one of these four strings: `int`, `float`, `str`, `bool`.

(4) Hint: `type(1) == int` evaluates to True.

```python
def odd_one(L):
    k = type(L[0])
    m = type(L[1])
    n = type(L[2])

    if k != m and m == n:
        if k == int:
            return 'int'
        if k == float:
            return 'float'
        if k == str:
            return 'str'
        if k == bool:
            return 'bool'

    for i in L:
        if k == type(i):
            continue
        else:
            if type(i) == int:
                return 'int'
            if type(i) == float:
                return 'float'
            if type(i) == str:
                return 'str'
            if type(i) == bool:
                return 'bool'


print(odd_one(eval(input().strip())))
```