#1 List

```python
if __name__ == '__main__':
    N = int(input())
    Output = [];
    for i in range(0,N):
        ip = input().split();
        if ip[0] == "print":
            print(Output)
        elif ip[0] == "insert":
            Output.insert(int(ip[1]),int(ip[2]))
        elif ip[0] == "remove":
            Output.remove(int(ip[1]))
        elif ip[0] == "pop":
            Output.pop();
        elif ip[0] == "append":
            Output.append(int(ip[1]))
        elif ip[0] == "sort":
            Output.sort();
        else:
            Output.reverse();
```

#2. If else

```python
import sys
N = int(input().strip())
if N % 2 != 0:
    print ("Weird")
else:
    if N >= 2 and N <= 5:
        print ("Not Weird")
    elif N >= 6 and N <= 20:
        print ("Weird")
    elif N > 20:
        print ("Not Weird")
```

#3 Arthematic opp

```
a = int(raw_input())
b = int(raw_input())
print a + b
print a - b
print a * b
```

#4 loops

```
if __name__ == '__main__':
   n = int(input())
   for i in range(0, n):
       print(i ** 2)
```

# 5 list comp

```
X = int(input())
Y = int(input())
Z = int(input())
N = int(input())
ans = [[i, j, k] for i in range(X + 1) for j in range(Y + 1) for k in
range(Z + 1) if i + j + k != N]
print (ans)
```

# 6 Complex no.

```
import math
class Complex(object):
    def __init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
    def __add__(self, no):
        real = self.real + no.real
        imaginary = self.imaginary + no.imaginary
        return Complex(real,imaginary)
    def __sub__(self, no):
        real = self.real - no.real
        imaginary = self.imaginary - no.imaginary
        return Complex(real,imaginary)
    def __mul__(self, no):
        real = self.real * no.real - self.imaginary * no.imaginary
        imaginary = self.real * no.imaginary + self.imaginary * no.real
        return Complex(real,imaginary)
    def __truediv__(self, no):
        x = float(no.real ** 2 + no.imaginary ** 2)
        y = self * Complex(no.real, -no.imaginary)
        real = y.real / x
        imaginary = y.imaginary / x
        return Complex(real, imaginary)
    def mod(self):
        real = math.sqrt(self.real ** 2 + self.imaginary ** 2)
        return Complex(real, 0)
# Classes Dealing with Complex Numbers in python - Hacker Rank Solution END
    def __str__(self):
        if self.imaginary == 0:
            result = "%.2f+0.00i" % (self.real)
        elif self.real == 0:
            if self.imaginary >= 0:
                result = "0.00+%.2fi" % (self.imaginary)
            else:
                result = "0.00-%.2fi" % (abs(self.imaginary))
        elif self.imaginary > 0:
            result = "%.2f+%.2fi" % (self.real, self.imaginary)
        else:
            result = "%.2f-%.2fi" % (self.real, abs(self.imaginary))
        return result
if __name__ == '__main__':
    c = map(float, input().split())
    d = map(float, input().split())
    x = Complex(*c)
    y = Complex(*d)
    print(*map(str, [x+y, x-y, x*y, x/y, x.mod(), y.mod()]), sep='\n')
```

#7 numpy matrix equal

```
import numpy as np
nums1 = np.array([0.5, 1.5, 0.2])
nums2 = np.array([0.4999999999, 1.500000000, 0.2])
np.set_printoptions(precision=15)
print("Original arrays:")
print(nums1)
print(nums2)
print("\nTest said two arrays are equal (element wise) or not:?")
print(nums1 == nums2)
nums1 = np.array([0.5, 1.5, 0.23])
nums2 = np.array([0.4999999999, 1.5000000001, 0.23])
print("\nOriginal arrays:")
np.set_printoptions(precision=15)
print(nums1)
print(nums2)
print("\nTest said two arrays are equal (element wise) or not:?")
print(np.equal(nums1, nums2))
```

#8 numpy sort

```
import numpy as np
nums = np.array([[5.54, 3.38, 7.99],
[3.54, 4.38, 6.99],
[1.54, 2.39, 9.29]])
print("Original array:")
print(nums)
print("\nSort the said array by row in ascending order:")
print(np.sort(nums))
print("\nSort the said array by column in ascending order:")
print(np.sort(nums, axis=0))
```

#9 numpy reverse

```python
import numpy as np
nums = np.array([[[1, 2, 3, 4],
[0, 1, 3, 4],
[90, 91, 93, 94],
[5, 0, 3, 2]]])
print("Original array:")
print(nums)
print("\nSwap rows and columns of the said array in reverse order:")
new_nums = print(nums[::-1, ::-1])
print(new_nums)
```

#10 SQLite database and connect with the database and print the version of the SQLite database

```python
import sqlite3
try:
   sqlite_Connection = sqlite3.connect('temp.db')
   conn = sqlite_Connection.cursor()
   print("\nDatabase created and connected to SQLite.")
   sqlite_select_Query = "select sqlite_version();"
   conn.execute(sqlite_select_Query)
   record = conn.fetchall()
   print("\nSQLite Database Version is: ", record)
   conn.close()
except sqlite3.Error as error:
   print("\nError while connecting to sqlite", error)
finally:
   if (sqlite_Connection):
       sqlite_Connection.close()
       print("\nThe SQLite connection is closed.")
```

#11 connect a database and create a SQLite table within the database.

```
import sqlite3
from sqlite3 import Error
def sql_connection():
   try:
     conn = sqlite3.connect('mydatabase.db')
     return conn
   except Error:
     print(Error)
def sql_table(conn):
   cursorObj = conn.cursor()
   cursorObj.execute("CREATE TABLE agent_master(agent_code
char(6),agent_name char(40),working_area char(35),commission
decimal(10,2),phone_no char(15) NULL);")
   print("\nagent_master file has created.")
   conn.commit()
sqllite_conn = sql_connection()
sql_table(sqllite_conn)
if (sqllite_conn):
 sqllite_conn.close()
 print("\nThe SQLite connection is closed.")
```

#12 Write a Python program to create a table and insert some records in that table. Finally selects all rows from the table and display the records.

```
import sqlite3
from sqlite3 import Error
def sql_connection():
   try:
     conn = sqlite3.connect('mydatabase.db')
     return conn
   except Error:
     print(Error)
def sql_table(conn):
   cursorObj = conn.cursor()
# Create the table
   cursorObj.execute("CREATE TABLE salesman(salesman_id n(5), name
char(30), city char(35), commission decimal(7,2));")
# Insert records
   cursorObj.executescript("""
   INSERT INTO salesman VALUES(5001,'James Hoog', 'New York', 0.15);
   INSERT INTO salesman VALUES(5002,'Nail Knite', 'Paris', 0.25);
   INSERT INTO salesman VALUES(5003,'Pit Alex', 'London', 0.15);
   INSERT INTO salesman VALUES(5004,'Mc Lyon', 'Paris', 0.35);
   INSERT INTO salesman VALUES(5005,'Paul Adam', 'Rome', 0.45);
   """)
   conn.commit()
   cursorObj.execute("SELECT * FROM salesman")
   rows = cursorObj.fetchall()
   print("Agent details:")
   for row in rows:
       print(row)
sqllite_conn = sql_connection()
sql_table(sqllite_conn)
if (sqllite_conn):
 sqllite_conn.close()
 print("\nThe SQLite connection is closed.")
```