Write a function f1(list) that prints the number of odd elements in a given list.

```
>>> f1([1,2,3,4])
2

>>> f1([1,2,3,4,5])
3
```

Write a function f2(list) that prints out each odd element in a given list.

```
>>> f2([1,2,3,4])
1
3

>>> f2([1,2,3,4,5])
1
3
5
```

Write a function f3(list) that prints the sum of all odd elements in a list.

```
>>> f3([1,2,3,4])
4
>>> f3([1,2,3,4,5])
9
```

Write a function f4(list) that returns the sum of all indices whose elements are odd in a given list.

```
>>> f4([1,2,3,4])
2

>>> f4([1,2,3,4,5])
6
```

Write a function f5(list) that returns the same list with each element squared.

```
>>> f5([1,2,3,4])
[1, 4, 9, 16]

>>> f4([1,2,3,4,5])
[1, 4, 9, 16, 25]
```

Write a function f6(list) that returns the largest number in a specified list.

```
>>> f6([1,2,3,4])
```

4

>>> f6([1,2,3,4,5])
5

Write a function f7(list) that returns the average of all numbers in a specified list.

>>> f7([1,2,3,4])
2.5

>>> f7([1,2,3,4,5])
3.0

Write a function f8(a, b, n) that prints all the numbers that can be divided by n within a and b inclusive range. Suppose n is a positive number.

>>> f8(1,10,2)
2
4
6
8
10

>>> f8(1,10,11)

>>> f8(1,10,7)
7

Write a function f9(width,height) that prints an ASCII rectangle with a given width and height.

>>> f9(0,1)

>>> f9(10,0)

>>> f9(1,1)
*****

>>> f9(1,2)
*****
*****

>>> f9(5,5)
****

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

Write a function f10(n) that prints a triangle with a given height n. Suppose n is not negative.

```
>>> f10(1)
*

>>> f10(2)
*
**

>>> f10(3)
*
**
***
```

Write a function f11(list) that returns True if the list is sorted in descending order, and False otherwise. An empty list returns True.

```
>>> f11([])
True

>>> f11([5,4,3,2,1])
True

>>> f11([5,4,3,2,0])
True

>>> f11([5,4,5,2])
False
```

Write a function f12(list) that returns True if the list consists of all negative numbers, False otherwise. An empty list returns True.

```
>>> f12([])
True

>>> f12([-1,-2,-3,-4,5])
False
```

```
>>> f12([1,2,3,4,5])
False

>>> f12([-1,-2,-3])
True
```

Write a function f13(list, target) that returns the index of the last target in the list. Assume the list is not empty and always contains a target.

```
>>> f13([1,2,3], 3)
2

>>> f13([1,2,3,1,2,3], 3)
5

>>> f13([1,1,1,1], 1)
3
```

Write a function f14(list) that returns the last negative index of a list. Assume that the list is not empty and always contains negative numbers.

```
>>> f14([1,2,-3])
2

>>> f14([1,-2,-3,1,-2,-3])
5

>>> f14([-1,1,1,1])
0
```

Write a function f15(list) that returns the sum of all elements at even index.

```
>>> f15([1,2,-3])
-2

>>> f15([1,-2,-3,1,-2,-3])
-4

>>> f15([-1,1,1,1])
0
```

Write a function f16(n) that will print an inverted triangle.

```
>>> f16(3)
***
```

```
**
*

>>> f16(2)
**
*

>>> f16(1)
*
```

Write a function f17(list) that prints out every other element in the list in reverse order.

```
>>> f17([1,2,3,4,5,6])
6
4
2

>>> f17([1,2,3,4])
4
2

>>> f17([1])
1
```

Write a function f18(n) that returns n!

```
>>> f18(0)
1

>>> f18(2)
2

>>> f18(3)
6
```

Write a function f19(list) that will print the factorial of each element of a given list.

```
>>> f19([ ])
>>> f19([1,2,3])
```

```
1
2
6

>>> f19([1,2,3,4])
1
2
6
24
```

Write a function f20(list) that prints a zero-terminated countdown for each element for a given list.

```
>>> f20([ ])

>>> f20([1,3,5])
1 0
3 2 1 0
5 4 3 2 1 0

>>> f20([5,3,6,2])
5 4 3 2 1 0
3 2 1 0
6 5 4 3 2 1 0
2 1 0
```

If you have two lists of the same length, list1 and list2, write a function f21(list1, list2) that returns a new list created by adding the elements at the same index of each list.

```
>>> f21([ ], [ ])
[]

>>> f21([1,2,3], [1,2,3])
[2, 4, 6]

>>> f21([0,0,0], [1,2,3])
[1, 2, 3]
```

Write a function f22(n) that prints all numbers from 1 to n that are a multiple of 2 or 3.

```
>>> f22(10)
```

2
3
4
6
8
9
10

```
>>> f22(1)

>>> f22(3)
2
3
```

Write a function f23(list) that returns the largest value in a nested list.

```
>>> f23( [ [1,2,3], [4,5,6], [7,8,9] ] )
9

>>> f23( [ [3,2,1] , [0,-1,-2] ] )
3

>>> f23( [ [1,2,3,4], [ ], [34], [ ], [ ], [56], [67] ] )
67
```

Write a function f24(list) that returns the second largest value in a list. Assume that all elements of the list are unique and contain more than one element.

```
>>> f24([1,4,3,2,5])
4

>>> f24([3,2])
2

>>> f24([3,4])
3
```

The function f25(n) returns the leftmost digit of n. n is a positive number. Implement f25(n).

```
>>> f25(1234)
```

1

>>> f25(4321)
4

>>> f25(3)
3

Write a function f26(list) that will print the largest value of each nested list in a given list. Assume that the nested list is not empty.

>>> f26([[1,2,3], [4,5,6], [7,8,9]])
3
6
9

>>> f26([[3,2,1], [0,-1,-2]])
3
0

>>> f26([[1,2,3,4], [1], [34], [2], [3], [56], [67]])
4
1
34
2
3
56
67

The f27(n) function uses several (1, …, n) sequences and outputs a triangular shape as shown below. Implement f27(n).

>>> f27(5)
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
>>> f27(0)

>>> f27(1)
1

The f28(n) function uses the number of times the number has been printed as an element and outputs a triangle shape as shown below. Implement f28(n). >>> f28(3)

1
2 3
4 5 6

>>> f28(0)

>>> f28(1)
1

>>> f28(5)
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15

The f29(n) function uses the number of times the number is exposed up to the nth row as an element, and after the nth row, the number of times decreasing by 1 is used as an element, and the following pyramid shape is output. Implement f29(n).
>>> f29(3)
1
2 3
4 5 6
2 3
1

>>> f29(4)
1
2 3
4 5 6
7 8 9 10
4 5 6
2 3
1

>>> f29(0)
>>> f29(1)
1

The f30(n) function takes the number of times a number is printed as an element to print a pyramid like the one below. Implement f30(n).

```
>>> f30(3)
1
2 3
4 5 6
7 8
9

>>> f30(0)

>>> f30(1)
1
```

The f31(matrix) function outputs the sum of each row of the received matrix. Implement f31(matrix).

```
>>> f31([[1,0],[0,1]])
1
1

>>> f31([[1,2,3],[4,5,6]])
6
15

>>> f31([[1],[2],[3],[4]])
1
2
3
4
```

The f32(matrix) function outputs the diagonal elements of the matrix. However, it is assumed that the matrix is a square matrix. Implement f32(matrix). >>> f32([[1,0],[0,1]])

```
1
1

>>> f32([[1,2,3],[4,5,6],[7,8,9]])
1
5
9
```

>>> f32([[1]])
1

The f33(matrix) function returns the sum of the members of all rows of the received matrix. Implement f33(matrix).

```
>>> f33( [[1,0],[0,1]] )
1
1

>>> f33( [[1,2,3],[4,5,6]] )
6
15

>>> f33( [[1],[2],[3],[4]] )
1
2
3
4
```

The f34(matrix) function returns the sum of all elements of the received matrix. Implement f34(matrix).

```
>>> f34([[1,0],[0,1]])
2

>>> f34([[1,2,3],[4,5,6]])
21

>>> f34([[1],[2],[3],[4]])
10
```

The f35(matrix) function outputs the odd number that exists in each row of the received matrix in "one line". Implement f35(matrix).

```
>>> f35([[1,0],[0,1]])
1
1

>>> f35([[1,2,3],[4,5,6]])
1 3
5

 >>> f35([[1],[2],[3],[4]])
1
```

3

The f36(matrix1, matrix2) function outputs the sum of the matrices for matrix1 and matrix2. However, it is assumed that the two matrices have the same size. Implement f36(matrix1, marix2).

```
>>> f36([[1,0],[0,1]],[[1,0],[0,1]])
[[2, 0], [0, 2]]

>>> f36([[1,2,3],[4,5,6]],[[-1,-1,-1],[-1,-1,-1]])
[[0, 1, 2], [3, 4, 5]]

>>> f36([[1],[2],[3],[4]],[[4],[3],[2],[1]])
[[5], [5], [5], [5]]
```

The f37(matrix1, matrix2) function returns the multiplication of the matrix for matrix1 and matrix2. However, it is assumed that the size of the columns of matrix1 and the rows of matrix2 for matrix multiplication are the same. Implement f37(matrix1, matrix2).

```
>>> f37([[1,0],[0,1]],[[1,0],[0,1]],)
[[1, 0], [0, 1]]

>>> f37([[1,2,3],[4,5,6]],[[-1,-1],[-1,-1],[-1,-1]])
[[-6, -6], [-15, -15]]

>>> f37([[4,3,2,1]],[[1],[2],[3],[4]])
[[20]]
```

38. f38(matrix) function returns True if it is the identity matrix. If it is not a unit matrix, it returns False. However, it is assumed that the rows and columns of the input matrix always have the same size. Implement f38(matrix).

```
>>> f38([[1]])
True

>>> f38([[1,0,0],[0,1,0],[0,0,1]])
True

>>> f38([[1,0,0],[0,1,5],[0,0,1]])
False
```

39. The f39(rows, cols) function returns a two-dimensional list containing the number of neighboring elements. The meaning of the neighboring element is when the element exists in “top, bottom, left, right”. Implement f39(rows, cols).

```
>>> f39(3,3)
[[2, 3, 2], [3, 4, 3], [2, 3, 2]]

>>> f39(5,1)
[[1], [2], [2], [2], [1]]

>>> f39(5,0)
[[], [], [], [], []]

>>> f39(0,5)
[]

>>> f39(2,2)
[[2, 2], [2, 2]]
```