(1) Counter Loop Pattern

In [ ]:
```
'''
Q1: Write a function checkSorted() that:

(1) Takes a list of comparable items as input; and
(2) Returns True if the sequence is decreasing and False, otherwise.

>>> checkSorted([10, 8, 4, 2, 1])
True

>>> checkSorted([10, 8, 2, 4, 1])
False
'''
```

In [1]:
```
def checkSorted(lst):
    for i in range(len(lst) - 1):
        if lst[i] > lst[i + 1]:
            return False
        return True
```

In [2]:
```
def checkSorted(lst):
    for i in range(len(lst) - 1):
        if lst[i] > lst[i + 1]:
            return False
    return True
```

In [3]:
```
checkSorted([10, 8, 4, 2, 1])
```

Out[3]: False

In [4]:
```
checkSorted([10, 8, 2, 4, 1])
```

Out[4]: False

(2) Accumulator Loop Pattern

In [ ]:
```
'''
Q2: Write a function sum() that:

(1) Takes a starting number and a ending number as two inputs; and
(2) Returns a sum of numbers between the starting and ending numbers.

>>> sum(1, 10)
55

>>> sum(1, 100)
```

```
5050
'''
```

In [5]:
```
def sum(x, y):
    res = 0
    for i in range(x, y + 1):
        res += i
    return res
```

In [6]:
```
sum(1, 10)
```

Out[6]: 55

In [7]:
```
sum(1, 100)
```

Out[7]: 5050

(3) Nested For Loop

In [ ]:
```
'''
Q3: Write a function inAmongst() that takes:

(1) Three lists of numbers as inputs; and
(2) Returns True if there is an item that is common to three lists and False, otherwise

>>> inAmongst([3, 2, 5, 4, 7], [9, 0, 1, 3],[3, 5, 4, 7])
True
>>> inAmongst([2, 5, 4, 7], [9, 0, 1, 3], [6, 8, 10, 11])
False
'''
```

In [8]:
```
def inAmongst(lst_1, lst_2, lst_3):
    for i in lst_1:
        for j in lst_2:
            for k in lst_3:
                if i == j == k:
                    return True
        return False
```

In [9]:
```
inAmongst([3, 2, 5, 4, 7], [9, 0, 1, 3], [3, 5, 4, 7])
```

Out[9]: True

In [10]:
```
inAmongst([2, 5, 4, 7], [9, 0, 1, 3], [6, 8, 10, 11])
```

Out[10]: False

In [ ]:
```
'''
Q4: Write a function pairSum() that takes:

(1) Two lists of numbers;
(2) A target value as inputs; and
(3) Prints the indexes of all pairs of values in the first and second lists that add up

>>> pairSum([2, 5, 4, 7], [9, 0, 6, 7], 11)
0 0
1 2
2 3
'''
```

In [11]:
```
def pairSum(lst_1, lst_2, target):
    for i in lst_1:
        for j in lst_2:
            if i + j == target:
                print(lst_1.index(i), lst_2.index(j))
```

In [12]:
```
pairSum([2, 5, 4, 7], [9, 0, 6, 7], 11)
```

```
0 0
1 2
2 3
```

In [ ]:
```
'''
Q5: Write a fuction square_graph() that takes:

(1) An integer number that represents a number of rows; and
(2) Prints a sequence of numbers on each row, which ranges from zero to the square of r

>>> squre_graph(5)
0 1
0 1 2 3 4
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
'''
```

In [13]:
```
def square_graph(n):
    for i in range(n):
        for j in range(((i + 1)**2) + 1):
            print(j, end = ' ')
        print()
```

In [14]:
```
square_graph(5)
```

```
0 1
0 1 2 3 4
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

(4) While Loop

In [ ]:
```
'''
Q6: Write a function n_halved() that takes:

(1) A positive integer n as an input; and
(2) Returns answers of the following question:
    How many times can the number n be halved (using integer division) before reaching 1

>>> n_halved(4)
2
>>> n_halved(11)
3
>>> n_halved(25)
4
'''
```

In [15]:
```
def n_halved(n):
    count = 0
    while n > 1:
        count += 1
        n //= 2
    return count
```

In [16]:
```
n_halved(4)
```

Out[16]: 2

In [17]:
```
n_halved(11)
```

Out[17]: 3

In [18]:
```
n_halved(25)
```

Out[18]: 4

In [ ]:
```
'''
Q7. Write a function fibonnaci() that takes:

(1) A bound as an input; and
(2) Returns the Fibnonaci sequence that its biggest Fibonaci number is smaller than the

>>> fibonnaci(12)
1 1 2 3 5 8

>>> fibonnaci(25)
1 1 2 3 5 8 13 21
'''
```

In [23]:
```
def fibonnaci(n):
    lst = [1, 1]
    a = 1
    b = 1
    c = a + b
    while c <= n:
        a = b
        b = c
        c = a + b
        lst.append(b)
    print(* lst, sep = ' ')
```

In [24]:
```
fibonnaci(12)
```

```
1 1 2 3 5 8
```

In [25]:
```
fibonnaci(25)
```

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

(5) Break and Continue Statements

In [ ]:
```
'''
Q8: Write a function r_pixels() that takes:

(1) A two-dimensional list of nonnegative integer entries (representing the values of p
(2) Prints a two-dimensional list of numbers. But, if there is 0 in the list, numbers a

>>> r_pixels([[1, 0, 5, 7, 10], [2, 3, 5, 7], [11, 3, 0, 7]])
1,
2, 3, 5, 7,
11, 3,
'''
```

In [26]:
```
def r_pixels(lst):
    for n in lst:
        for i in n:
            if i == 0:
                break
            print(i, end = ', ')
        print()
```

In [27]:
```
r_pixels([[1, 0, 5, 7, 10], [2, 3, 5, 7], [11, 3, 0, 7]])
```

```
1,
2, 3, 5, 7,
11, 3,
```

(6) Others

In [ ]:
```
'''
Q9: Write a function evensum() that takes:

(1) A two-dimensional list of integers; and
(2) Returns True if every row of the table sums up to an even number
    and False, otherwise (i.e., if any row sums up to an odd number).

>>> evensum([[2, 4], [3, 5, 2], [2, 9, 1]])
True
>>> evensum([[1, 5, 4], [3, 5, 1], [2, 9, 1]])
False
'''
```

In [30]:
```
def evensum(lst):
    for i in lst:
        if sum(i) % 2 != 0:
            return False
    return True
```

In [33]:
```
evensum([[2, 4], [3, 5, 2], [2, 9, 1]])
```

```
True
```

In [34]:
```
evensum([[1, 5, 4], [3, 5, 1], [2, 9, 1]])
```

```
False
```

In [ ]:
```
'''
Q10: Write function lst_prime() that:

(1) Takes a bound as an input, which is positive integer; and
(2) Returns a list of prime numbers smaller than or equal to the bound.

>>> lst_prime(0)
[]
>>> lst_prime(6)
[2, 3, 5]
>>> lst_prime(11)
[2, 3, 5, 7, 11]
'''
```

In [35]:
```
def lst_prime(n):

    def is_prime(n):
        if n < 2:
            return False
        for i in range(2, n):
            if n % i == 0:
                return False
        return True

    lst = []
    if n < 2:
        return lst
```

```
    lst.append(2)
    for n in range(3, n + 1):
        if is_prime(n):
            lst.append(n)
    return lst
```

In [36]:
```
lst_prime(0)
```

Out[36]:
```
[]
```

In [37]:
```
lst_prime(6)
```

Out[37]:
```
[2, 3, 5]
```

In [38]:
```
lst_prime(11)
```

Out[38]:
```
[2, 3, 5, 7, 11]
```