Circular Array Structure: Check the following Linear Array:

```
  0    1    2    3    4     5    6   7
----------------------------------------   capacity = 8
| 5 |  7 |  9 |  15 | -4 |  17 |    |   |   size     = 6
----------------------------------------
                                  ^-- nextAvailPos = size
```

Here capacity represents the fixed array length and size represents the number of elements present in the array at that time. For example, if 17 was not present then the capacity would still be the same which is 8 but the size would be 5.

Now check the following image where this linear array is converted to a circular array with different start indexes.

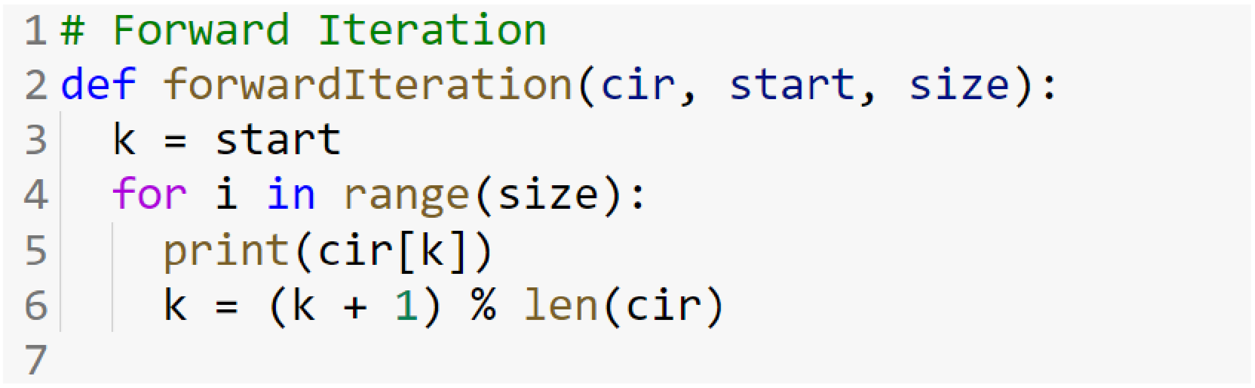
```
  0    1    2    3    4     5    6   7
----------------------------------------   capacity = 8
| 5 |  7 |  9 |  15 | -4 |  17 |    |   |   size     = 6
----------------------------------------
  ^-- start                      ^-- nextAvailPos

  0    1    2    3    4     5    6   7
----------------------------------------   capacity = 8
|   | 5  |  7 |  9  | 15 | -4  | 17|   |   size     = 6
----------------------------------------
      ^-- start                      ^-- nextAvailPos

  0    1    2    3    4     5    6   7
----------------------------------------   capacity = 8
| 15| -4 | 17 |     |    | 5   | 7 | 9 |   size     = 6
----------------------------------------
 nextAvailPos --^          ^-- start
```

Note that the capacity is 8 every time and the size is 6 because we are not adding or deleting any item from the given linear array.

The important thing to remember is that we do not allow any empty slots in between consecutive items in an array.

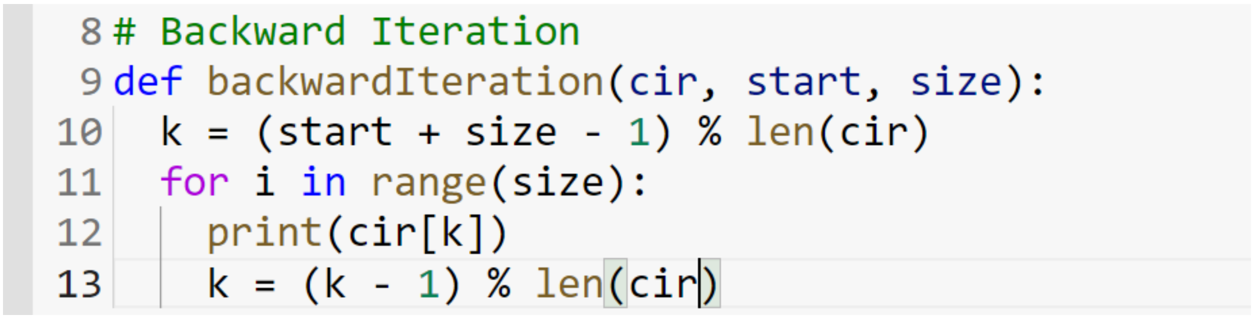
Iteration over a circular array: The iteration is similar to linear array iteration. However, the change is that we need to know the start index and how many values we need to iterate through. To solve this, we can use two variables. One for accessing the value (e.g: k in the below code) and one for checking the iteration number (e.g: i in the below code).

Forward:



```python
# Forward Iteration
def forwardIteration(cir, start, size):
  k = start
  for i in range(size):
    print(cir[k])
    k = (k + 1) % len(cir)

```

Backward:
Last item's index of circular array = (start + size - 1) % len(cir). You will understand the formula when you read the offset part.



```python
# Backward Iteration
def backwardIteration(cir, start, size):
  k = (start + size - 1) % len(cir)
  for i in range(size):
    print(cir[k])
    k = (k - 1) % len(cir)
```

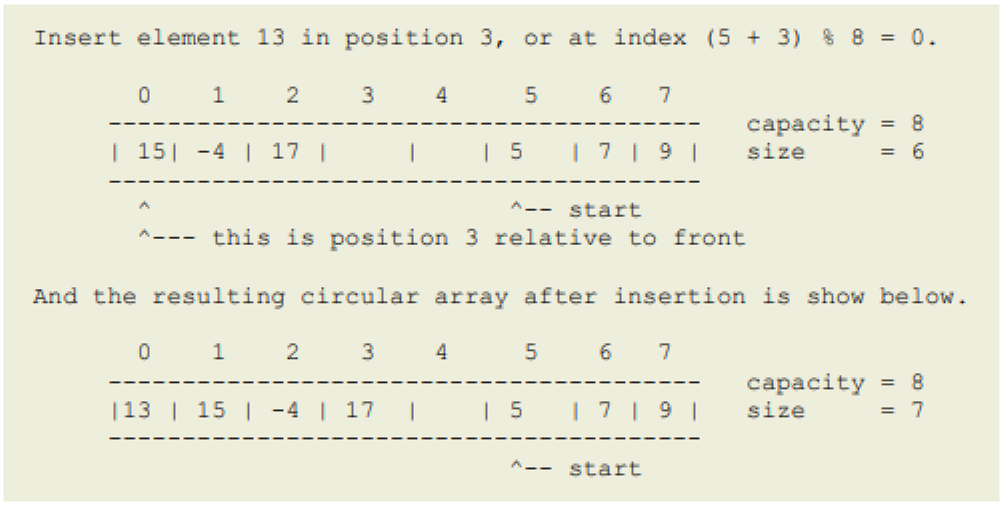
Insertion: Insertion in a circular array is similar to the linear array where we only need an extra offset. The initial offset 0 starts from the start index and then it linearly increases. Offset is also referred as position sometimes. For offset i, the actual index formula is:

Offset i index = (start + i) % len(cir)

From this formula, we can get the formula of the last index and next available index.

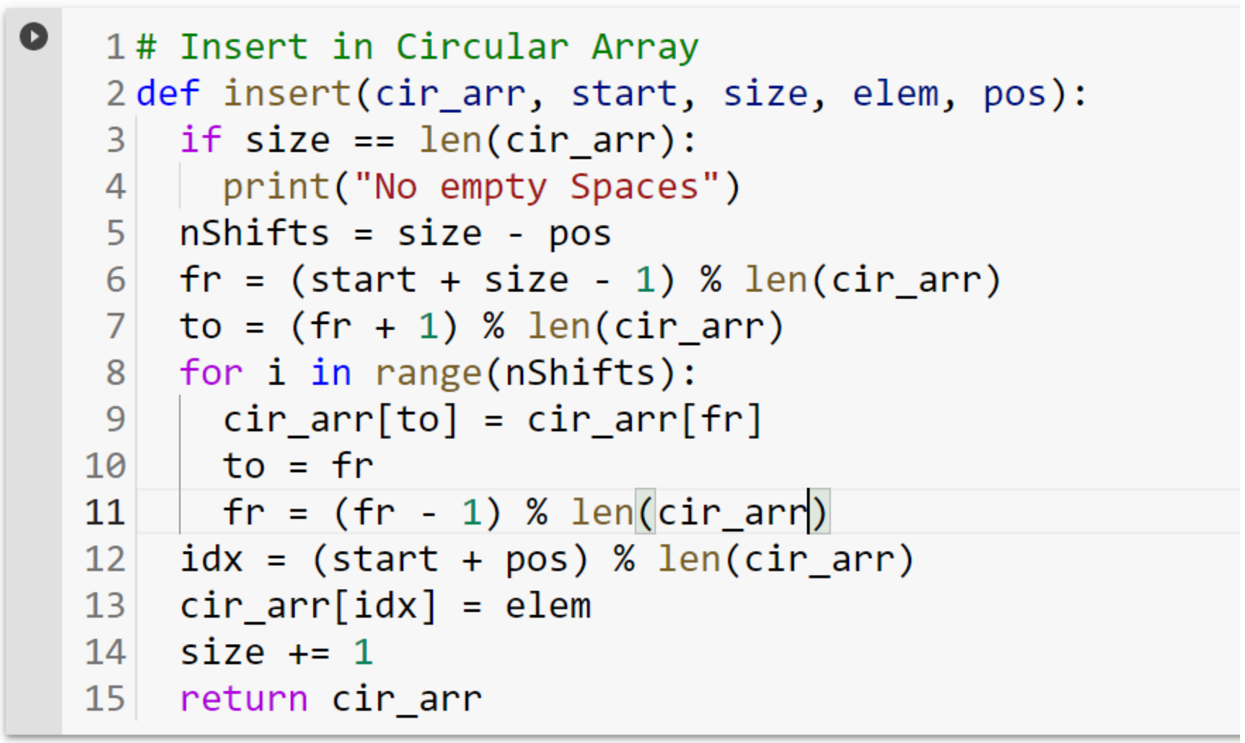
Last index = (start + size - 1) % len(cir)
Next available index = (start + size) % len(cir)

Below there is a simulation of circular array insertion is given.



```
Insert element 13 in position 3, or at index (5 + 3) % 8 = 0.

      0    1    2    3    4     5    6   7
     ----------------------------------------   capacity = 8
     | 15| -4 | 17 |     |    | 5    | 7 | 9 |   size     = 6
     ----------------------------------------
      ^                         ^-- start
      ^--- this is position 3 relative to front

And the resulting circular array after insertion is show below.

      0    1    2    3    4     5    6   7
     ----------------------------------------   capacity = 8
     |13 | 15 | -4 | 17  |    | 5    | 7 | 9 |   size     = 7
     ----------------------------------------
                                ^-- start
```

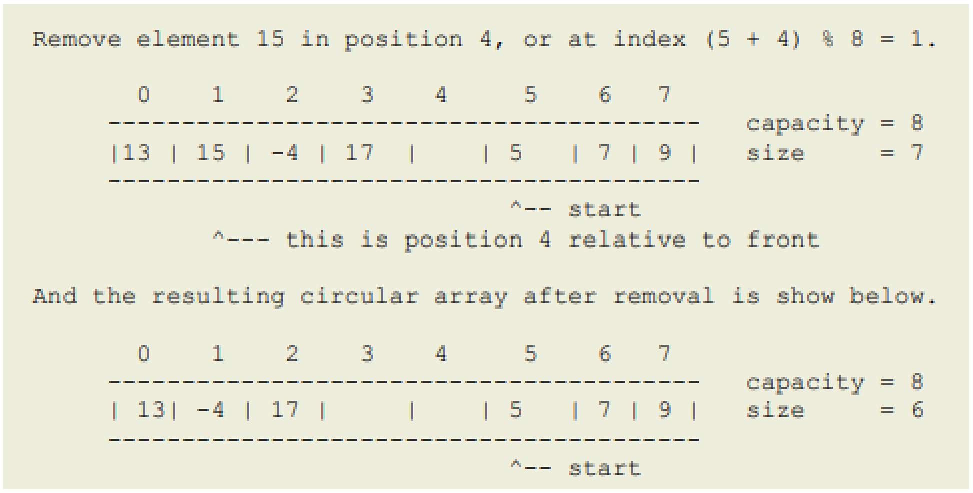
In this method, the start parameter represents the starting index, the size represents the number of elements present in the circular array, elem represents the element that will be inserted into the array and pos represents the position in which the value should be inserted.



```python
# Insert in Circular Array
def insert(cir_arr, start, size, elem, pos):
  if size == len(cir_arr):
    print("No empty Spaces")
  nShifts = size - pos
  fr = (start + size - 1) % len(cir_arr)
  to = (fr + 1) % len(cir_arr)
  for i in range(nShifts):
    cir_arr[to] = cir_arr[fr]
    to = fr
    fr = (fr - 1) % len(cir_arr)
  idx = (start + pos) % len(cir_arr)
  cir_arr[idx] = elem
  size += 1
  return cir_arr
```

Remove by Left Shift:

With the help of the knowledge of offset mapping, we can remove items by left shit. Check the picture to see how an item is removed from a circular array.



```
Remove element 15 in position 4, or at index (5 + 4) % 8 = 1.

       0    1    2    3    4     5    6   7
     ----------------------------------------   capacity = 8
     |13 | 15 | -4 | 17  |     | 5    | 7 | 9 |   size     = 7
     ----------------------------------------
                                ^-- start
            ^--- this is position 4 relative to front

And the resulting circular array after removal is show below.

       0    1    2    3    4     5    6   7
     ----------------------------------------   capacity = 8
     | 13| -4 | 17 |      |     | 5    | 7 | 9 |   size     = 6
     ----------------------------------------
                                ^-- start
```

The code snippet is given below: Here, the start represents the start index, the size represents the number of items present in the array and pos represents the position / offset to remove.

```python
# Remove value from circular array by left shift
def removeByLeftShift(cir_arr, start, size, pos):
  nShift = size - pos - 1
  idx = (start + pos) % len(cir_arr)
  removed = cir_arr[idx]
  to = idx
  fr = (to + 1) % len(cir_arr)
  for i in range(nShifts):
    cir_arr[to] = cir_arr[fr]
    to = fr
    fr = (fr + 1) % len(cir_arr)
  cir_arr[fr] = None
  size -= 1
  return removed
```