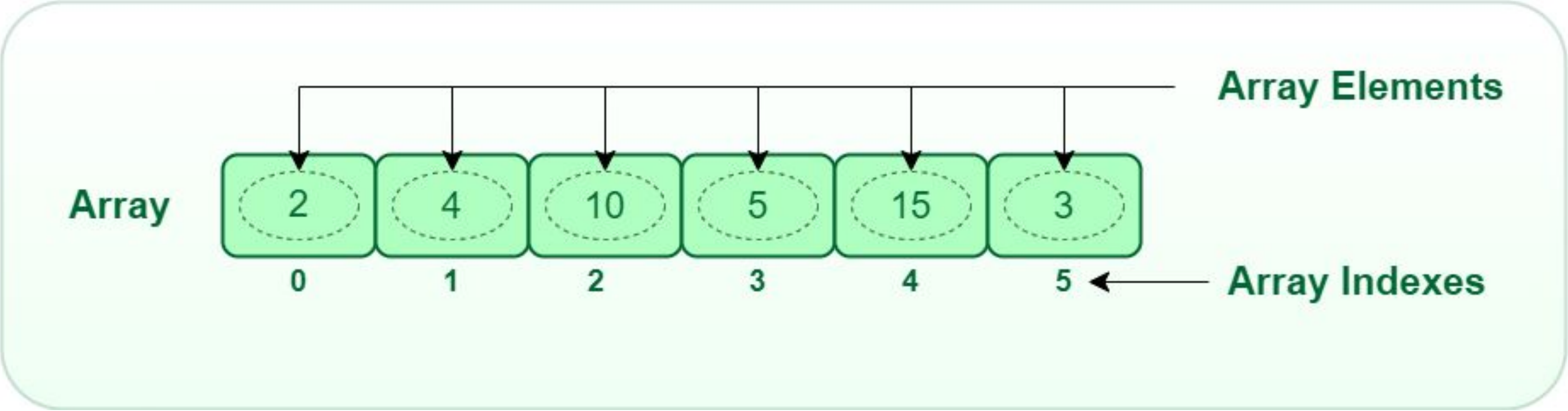
Data Structures

Lecture 1

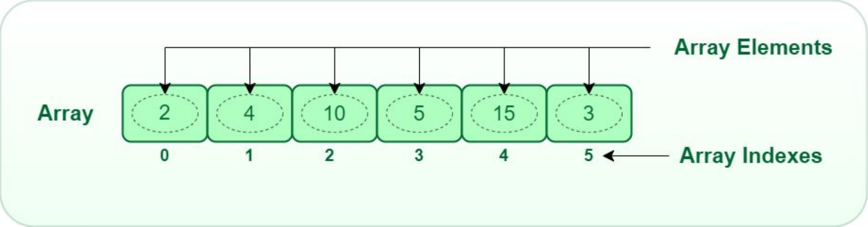
Linear Array Basics and Operations

Credit: Prantik Paul, Lecturer, BRAC University

Array - Basics



Array - Basics



Dimension - fixed (1, 2, 3, ..., 100, ..)

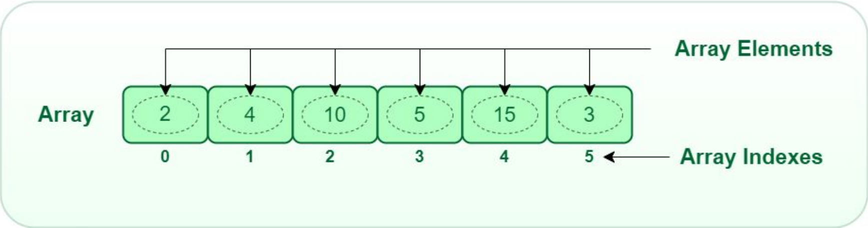
Index

→ 0 - indexing

Size / Length

Type - Fixed

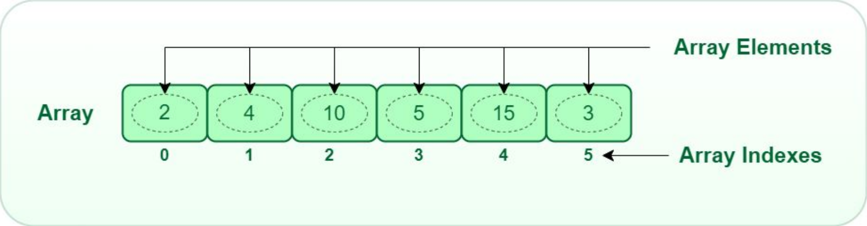
Array - Basics



Dimension - fixed (1, 2, 3, ..., 100, ..)

Linear Arrays - 1D
Matrix - 2D
Tensor - 3D

Array - Basics



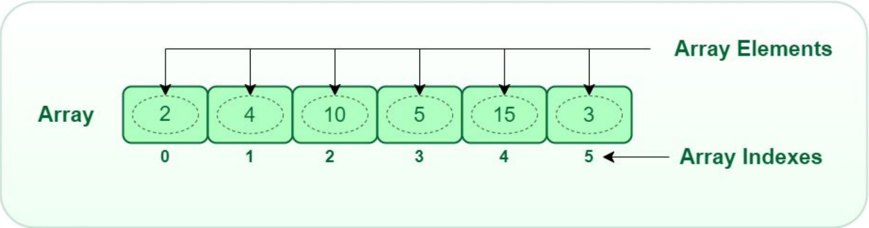
Index

→ 0 - indexing

Why 0-indexing?

For efficient memory access

Array - Basics



Size / Length

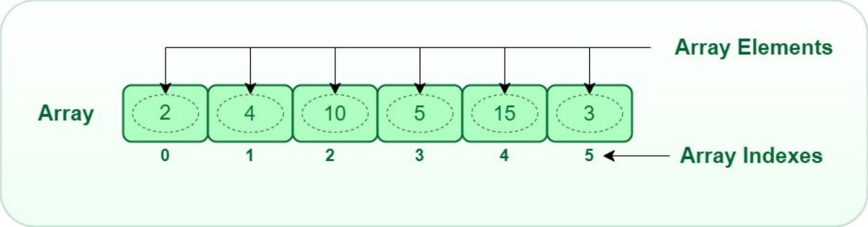
Size?

Number of Valid Elements

Length?

Fixed - As Declared

Array - Basics

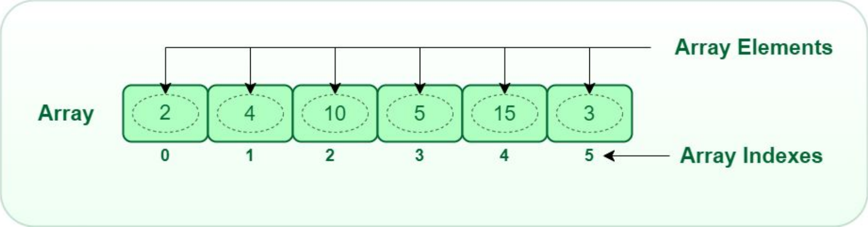


Type - Fixed

Why Not Dynamic?

Efficient Memory Access

Array - Basics



Integer : 4bytes

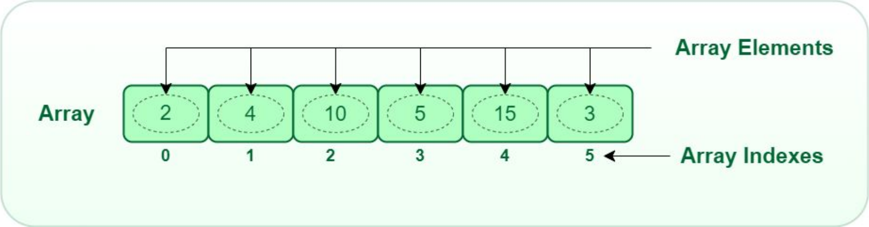
Floats : 8bytes

Character : 1byte

Formula-

Starting address of array + index * bytes required for one data

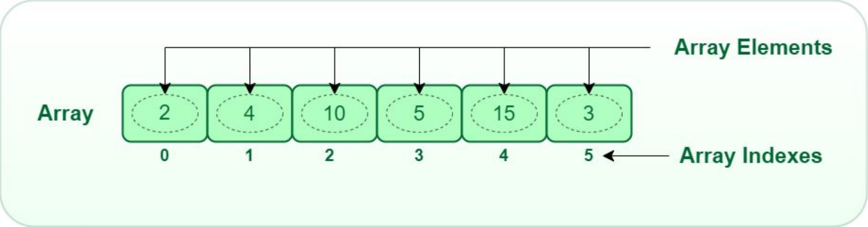
Array - Determine Length



Efficient-

- Doesn’t store size
- May give invalid data if index out of bounds
- can access faster
- C Fortran

Array - Determine Length



Inefficient-

- Store size in first byte
- Check if index out of bound
- Slow access
- Java C#

Refresh your memory

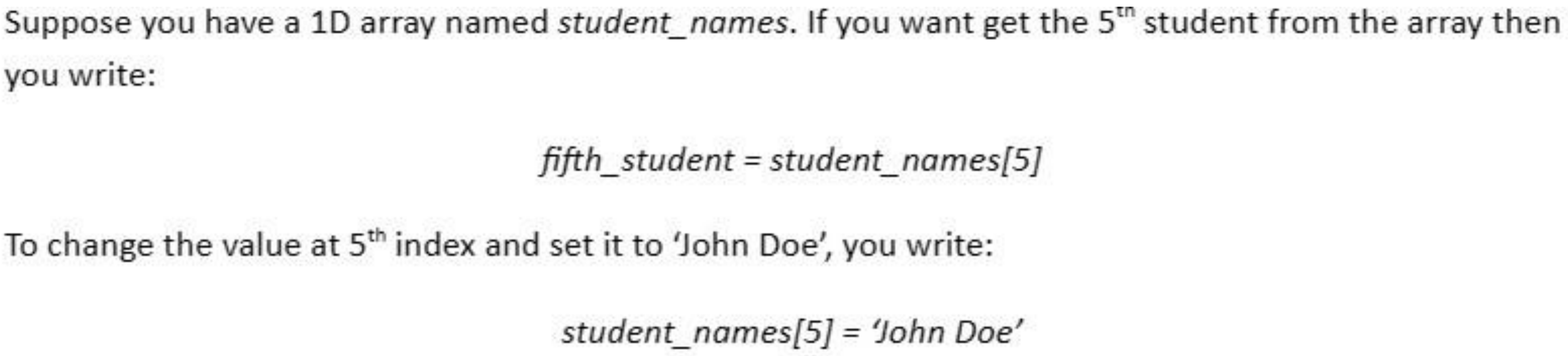
Python:

1. for i in range (stop)
2. for i in range (start, stop)
3. for i in range (start, stop, step)
4. Negative step?
5. Negative stop?

Java:

1. for(int i = 0; i < stop; i++)
2. for(int i = start; i < stop; i++)
3. for(int i = start; i < stop; i += step)
4. Negative step?
5. Negative stop?

Array - Accessing/Changing an Element



Suppose you have a 1D array named *student_names*. If you want get the 5th student from the array then you write:

fifth_student = student_names[5]

To change the value at 5th index and set it to ‘John Doe’, you write:

student_names[5] = ‘John Doe’

Array - Initializing

```
array = [None] * length
```

```
int[] array = new int[length];
```

Array - Initializing

```
Import numpy as np
```

```
my_array = np.array([8, 3, 13, 1])
```

Array - Initializing

```
Import numpy as np
```

```
my_list = [2, 45, 3, 2, 56]
my_array = np.asarray(my_list)
```

Array - Initializing

```
Import numpy as np
```

```
size = 5
my_array = np.zeros(size)
```

Array - Initializing

```
Import numpy as np
```

```
size = 5
my_array = np.ones(size)
```

Array - Initializing

```
Import numpy as np
```

```
size = 5
my_array = np.empty(size)
```

Array - Initializing

```
Import numpy as np
```

```
size = 5
my_array = np.full(size, 4)
```

Array - Initializing

```
Import numpy as np
```

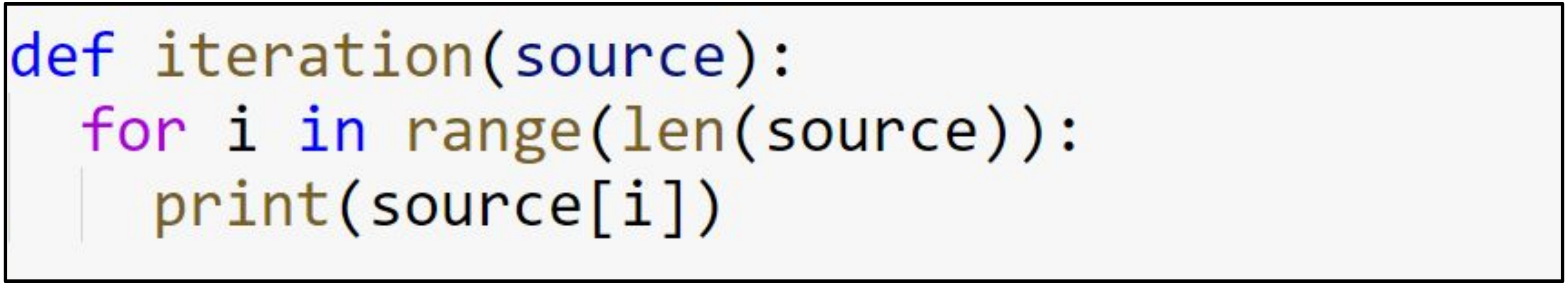
```
size = 5
my_array = np.ones(size)
print(my_array.size)
```

Array - Initializing

```
Import numpy as np
```

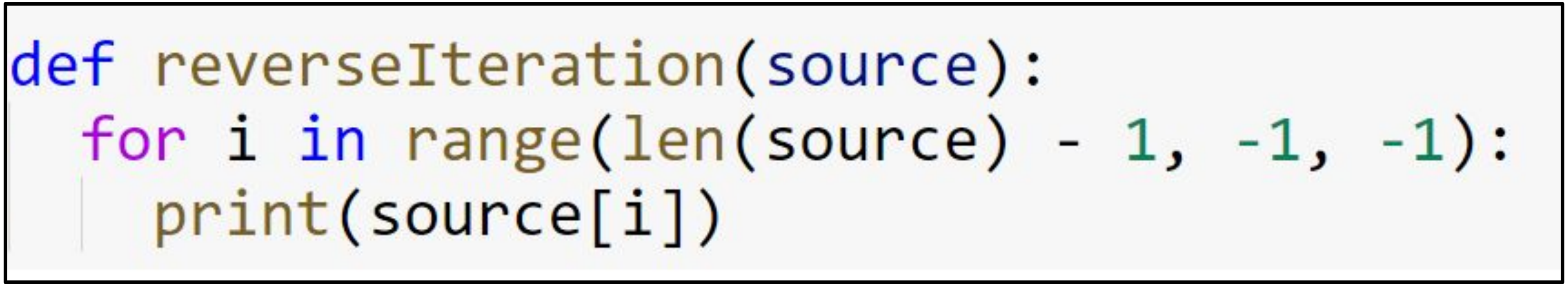
```
size = 5
my_array = np.ones(size)
print(len(my_array))
```

Array - Functions (Iteration)



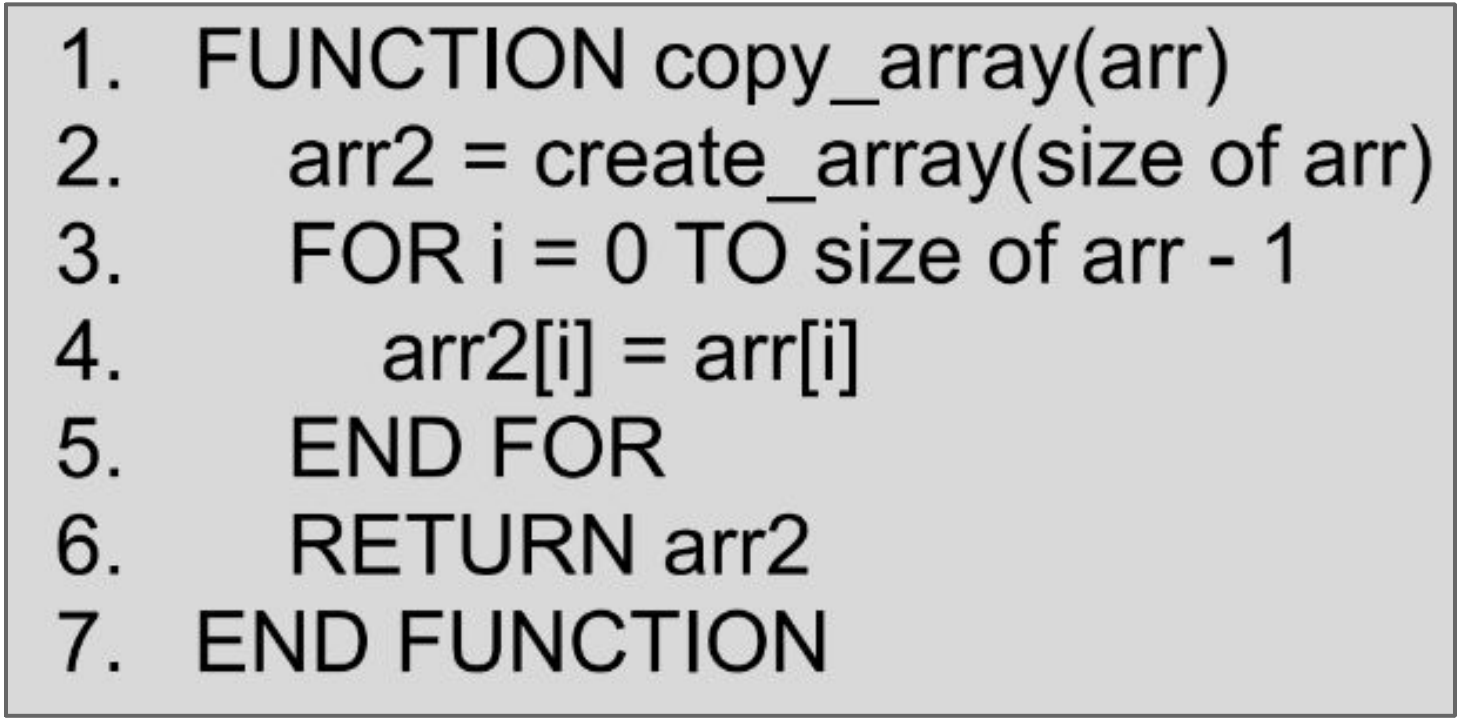
```
def iteration(source):
  for i in range(len(source)):
    print(source[i])
```

Array - Functions (Reverse Iteration)



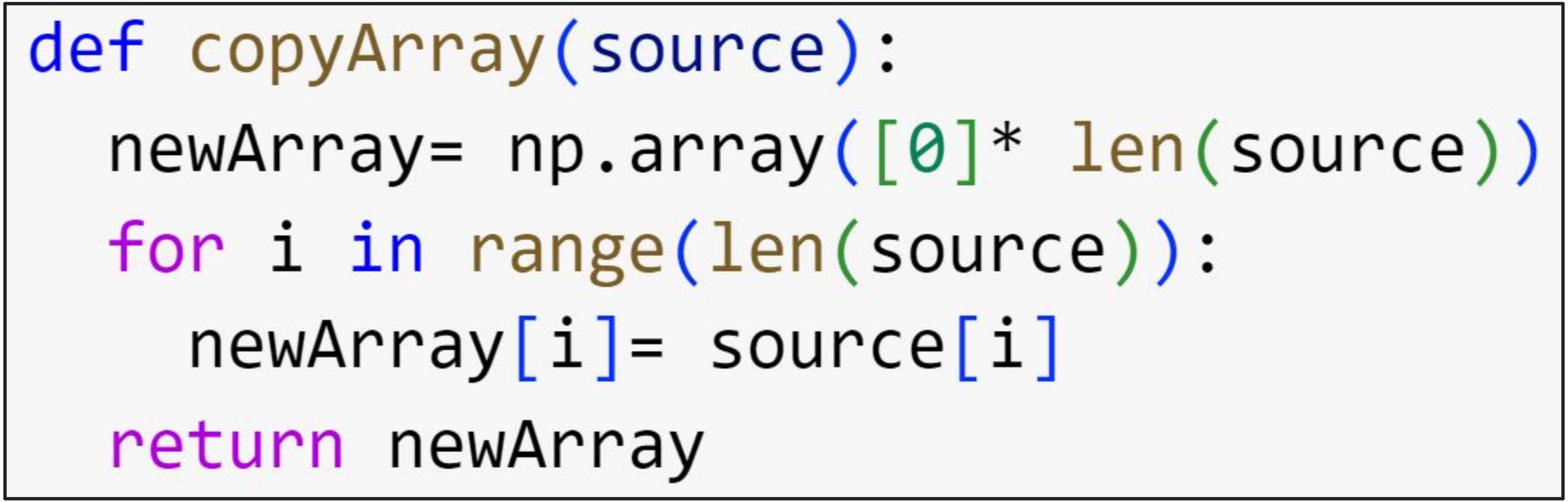
```
def reverseIteration(source):
  for i in range(len(source) - 1, -1, -1):
    print(source[i])
```

Array - Functions (Copy Array)



```
FUNCTION copy_array(arr)
    arr2 = create_array(size of arr)
    FOR i = 0 TO size of arr - 1
        arr2[i] = arr[i]
    END FOR
    RETURN arr2
END FUNCTION
```

Array - Functions (Copy Array)

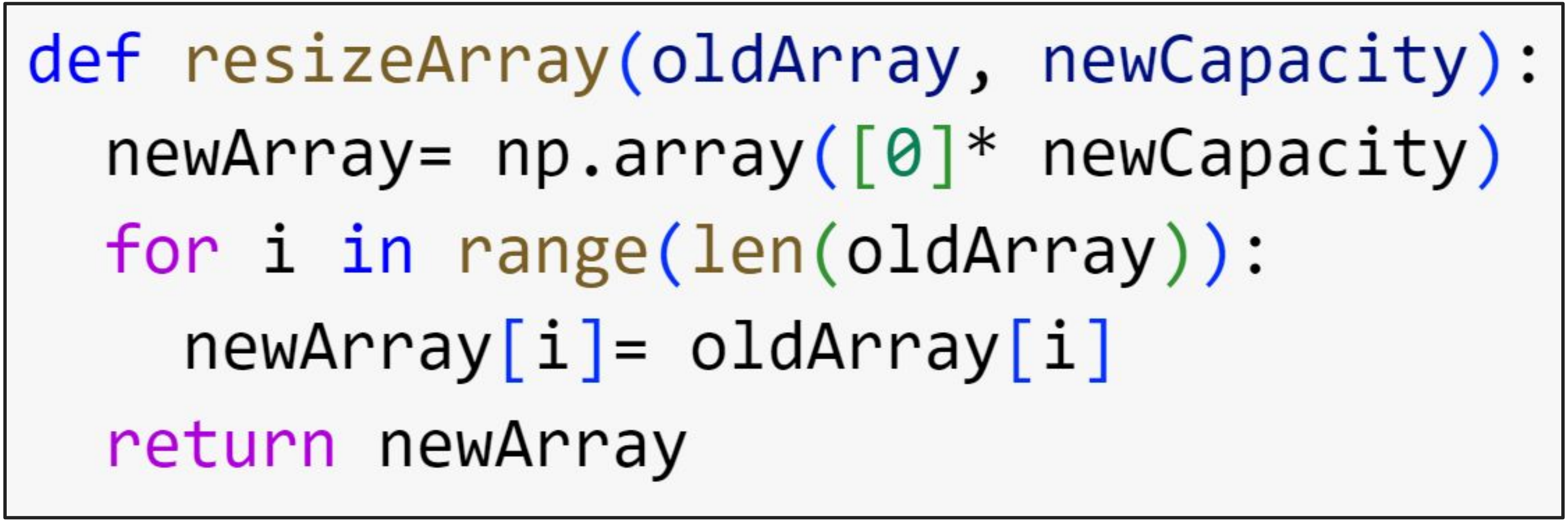


```
def copyArray(source):
  newArray= np.array([0]* len(source))
  for i in range(len(source)):
    newArray[i]= source[i]
  return newArray
```

Array - Functions (Resize Array)

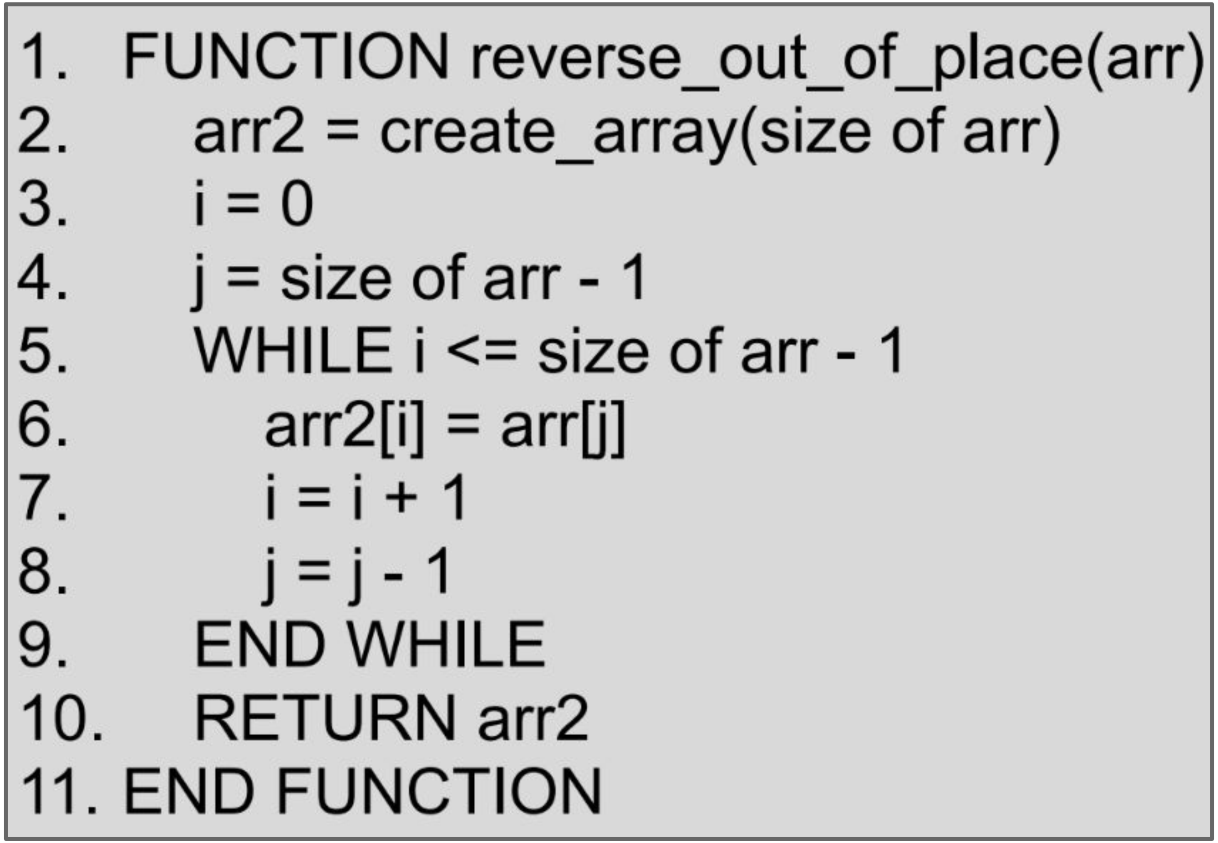
```
function resize(arr, new_size)
  set arr2 to an array of size new_size, initialized with 0
  set i to 0
  while i is less than length of arr
    set arr2[i] to arr[i]
    increment i by 1
  end while
  return arr2
end function
```

Array - Functions (Resize Array)



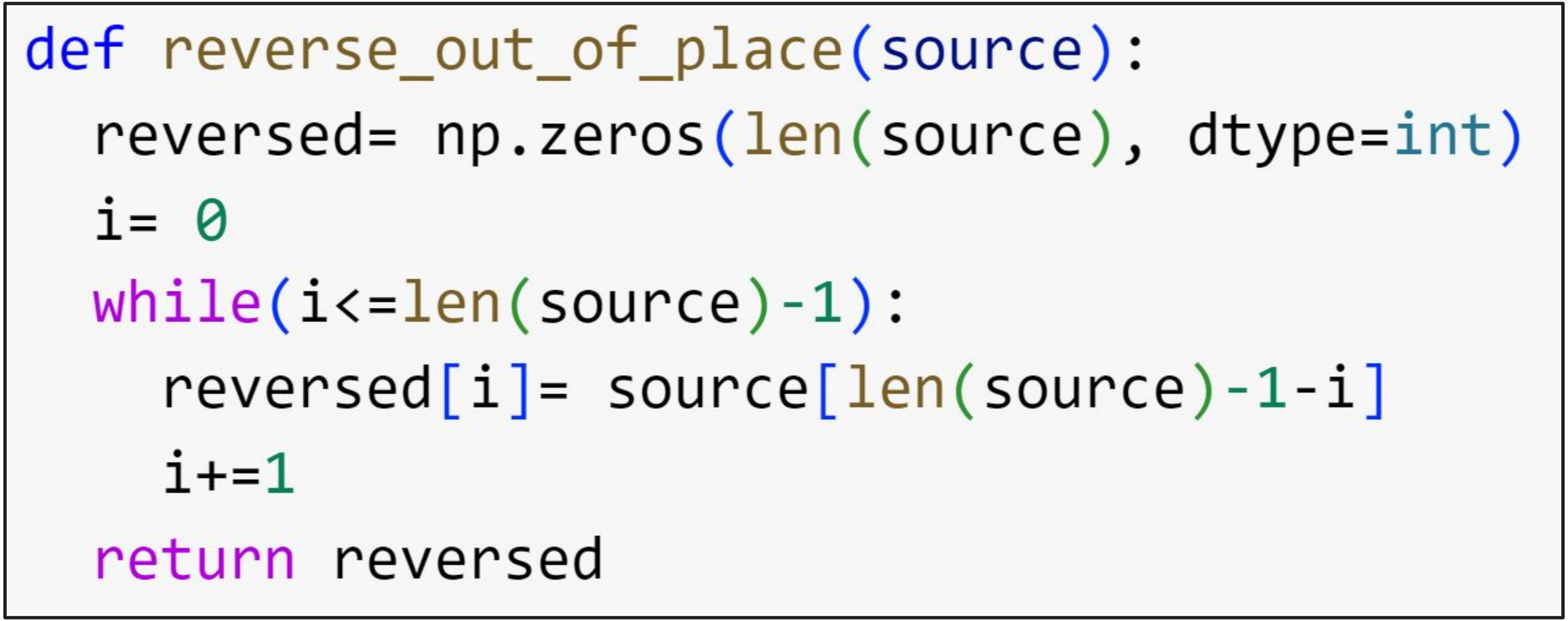
```
def resizeArray(oldArray, newCapacity):
  newArray= np.array([0]* newCapacity)
  for i in range(len(oldArray)):
    newArray[i]= oldArray[i]
  return newArray
```

Array - Functions (Reverse Array)



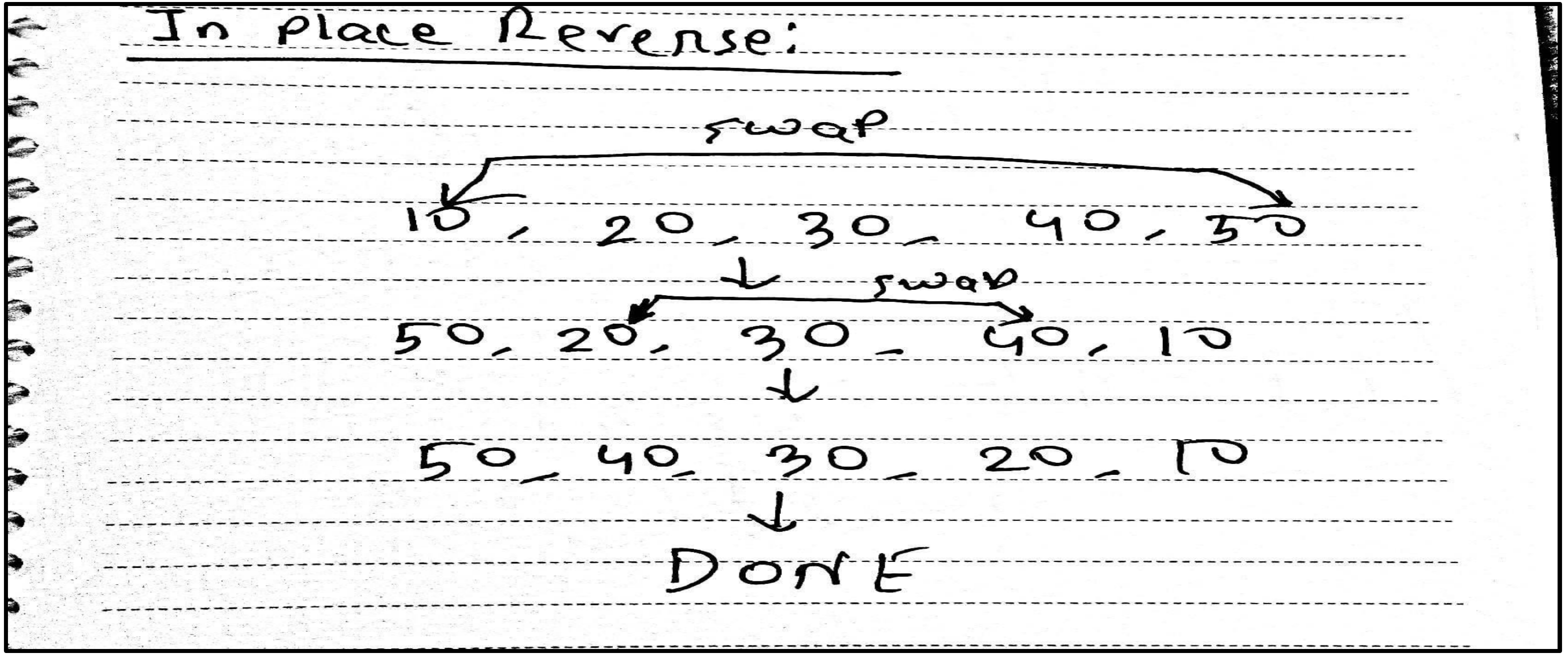
```
FUNCTION reverse_out_of_place(arr)
    arr2 = create_array(size of arr)
    i = 0
    j = size of arr - 1
    WHILE i <= size of arr - 1
        arr2[i] = arr[j]
        i = i + 1
        j = j - 1
    END WHILE
    RETURN arr2
END FUNCTION
```

Array - Functions (Reverse Array)



```python
def reverse_out_of_place(source):
  reversed= np.zeros(len(source), dtype=int)
  i= 0
  while(i<=len(source)-1):
    reversed[i]= source[len(source)-1-i]
    i+=1
  return reversed
```

Array - Functions (Reverse Array) (Efficient)



In place Reverse:

swap

10, 20, 30, 40, 50

↓ swap

50, 20, 30, 40, 10

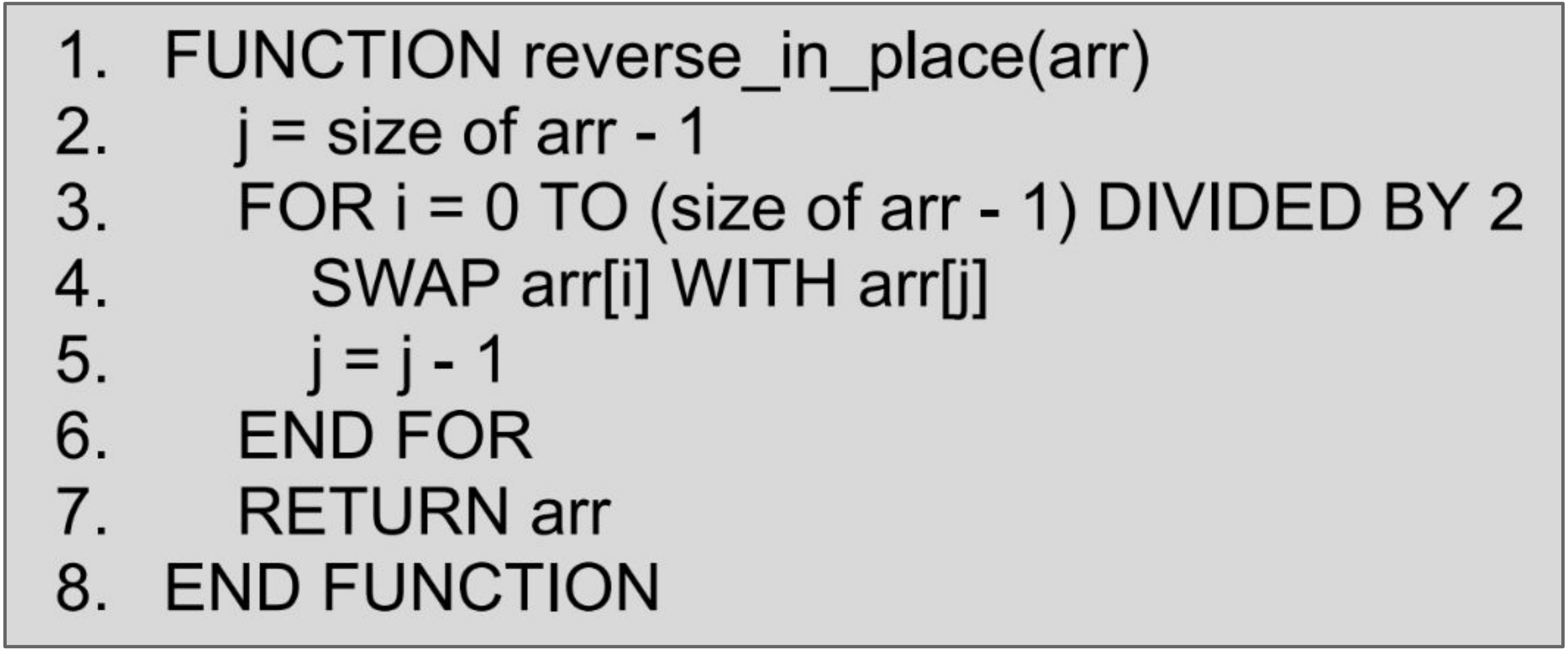
↓

50, 40, 30, 20, 10

↓

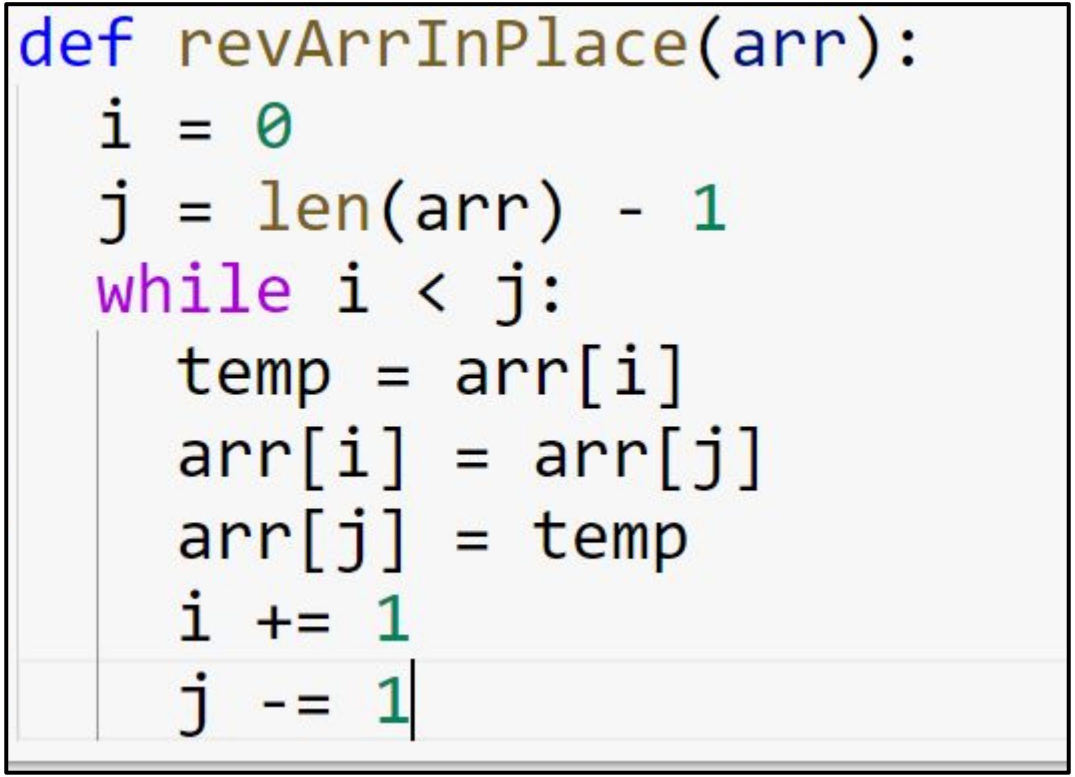
DONE

Array - Functions (Reverse Array) (Efficient)



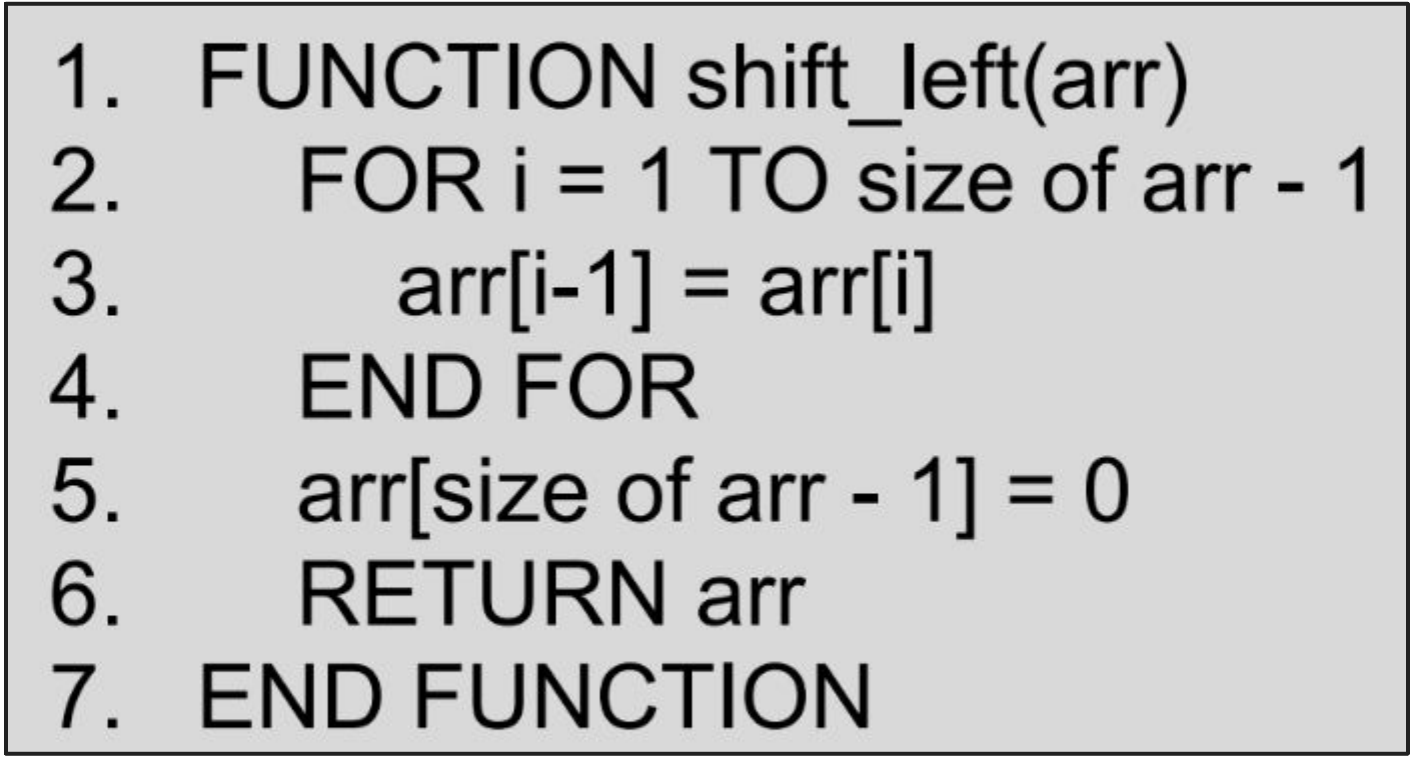
```
FUNCTION reverse_in_place(arr)
    j = size of arr - 1
    FOR i = 0 TO (size of arr - 1) DIVIDED BY 2
        SWAP arr[i] WITH arr[j]
        j = j - 1
    END FOR
    RETURN arr
END FUNCTION
```

Array - Functions (Reverse Array) (Efficient)



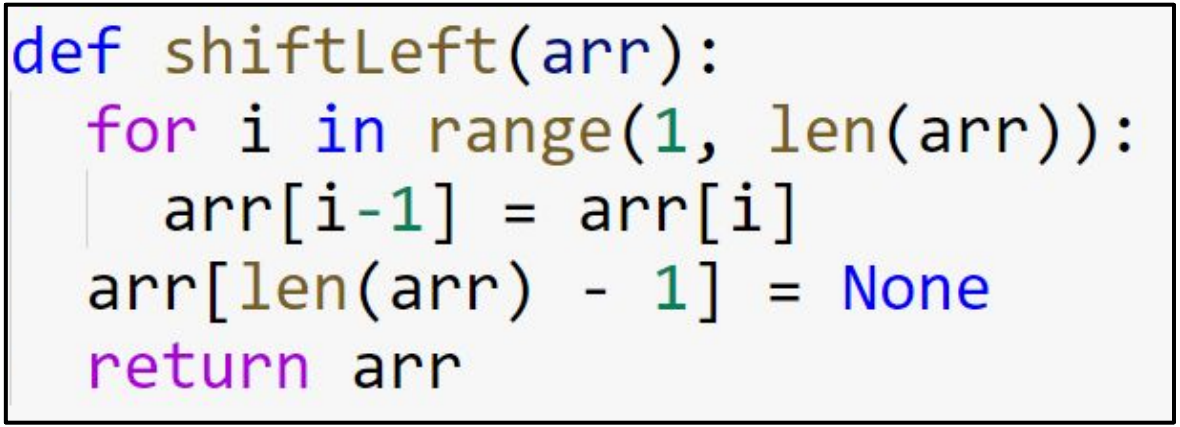
```python
def revArrInPlace(arr):
  i = 0
  j = len(arr) - 1
  while i < j:
    temp = arr[i]
    arr[i] = arr[j]
    arr[j] = temp
    i += 1
    j -= 1
```

Array - Functions (Shift Left)



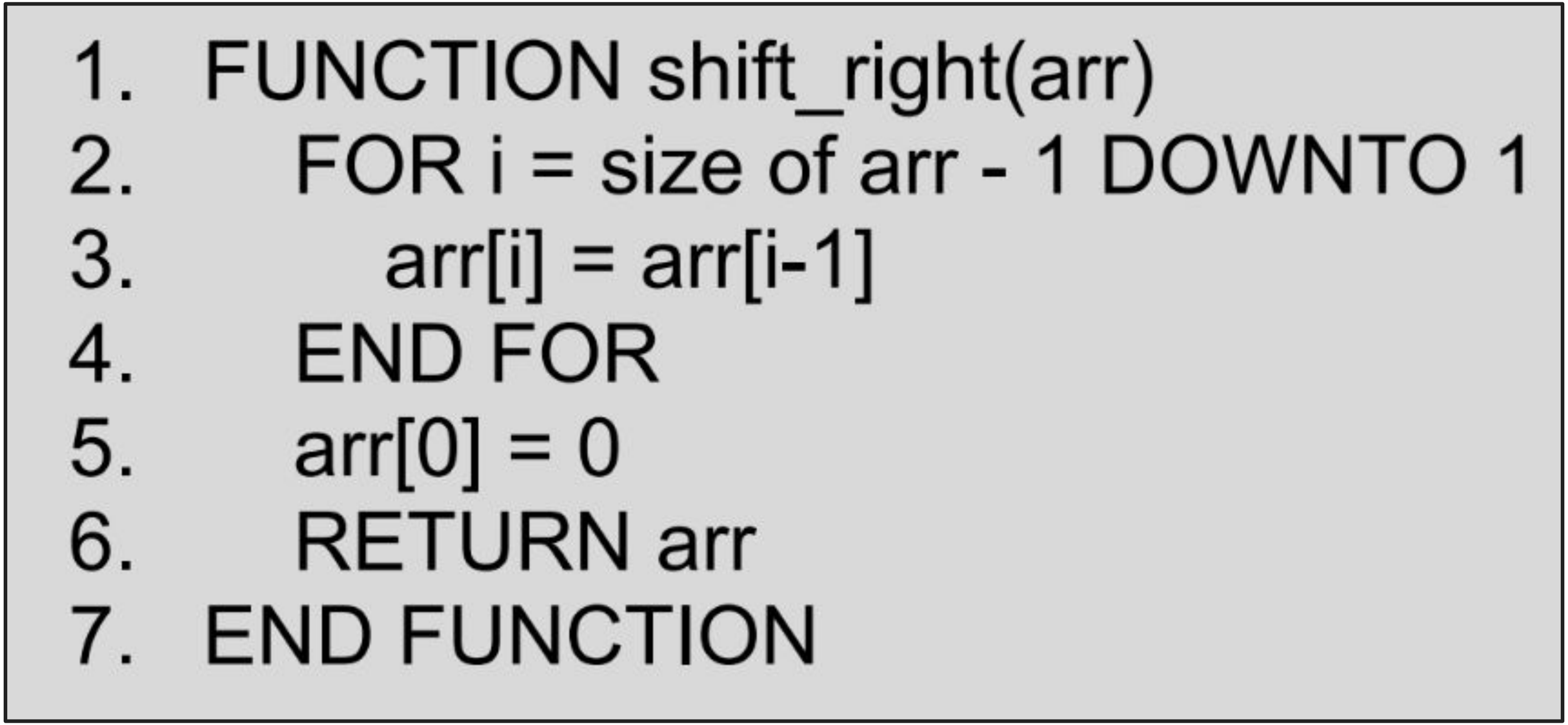
```
FUNCTION shift_left(arr)
    FOR i = 1 TO size of arr - 1
        arr[i-1] = arr[i]
    END FOR
    arr[size of arr - 1] = 0
    RETURN arr
END FUNCTION
```

Array - Functions (Shift Left)



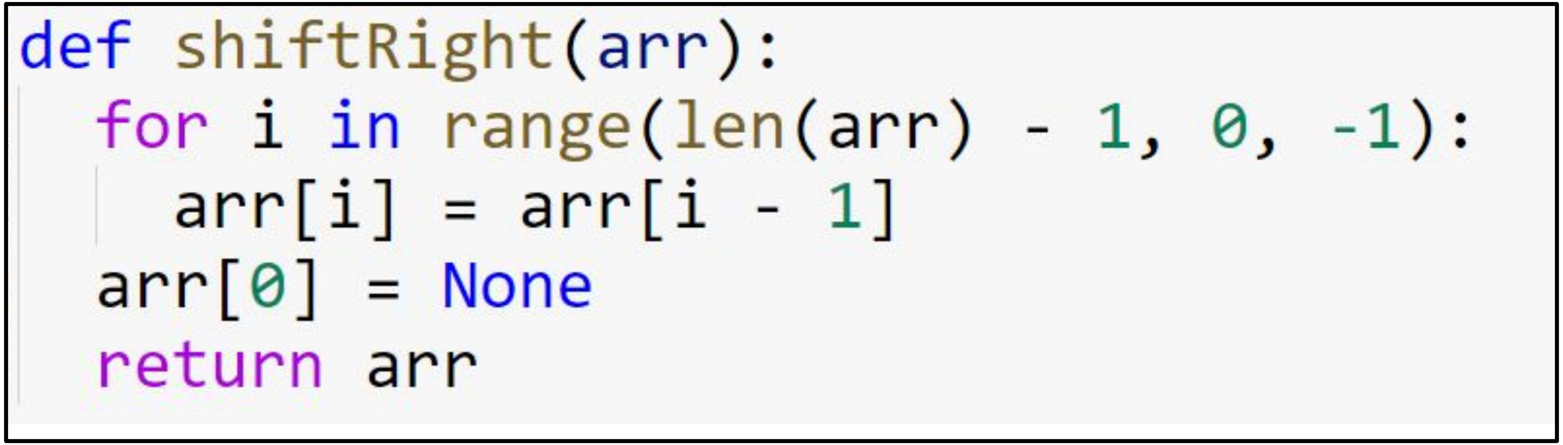
```
def shiftLeft(arr):
  for i in range(1, len(arr)):
    arr[i-1] = arr[i]
  arr[len(arr) - 1] = None
  return arr
```

Array - Functions (Shift Right)



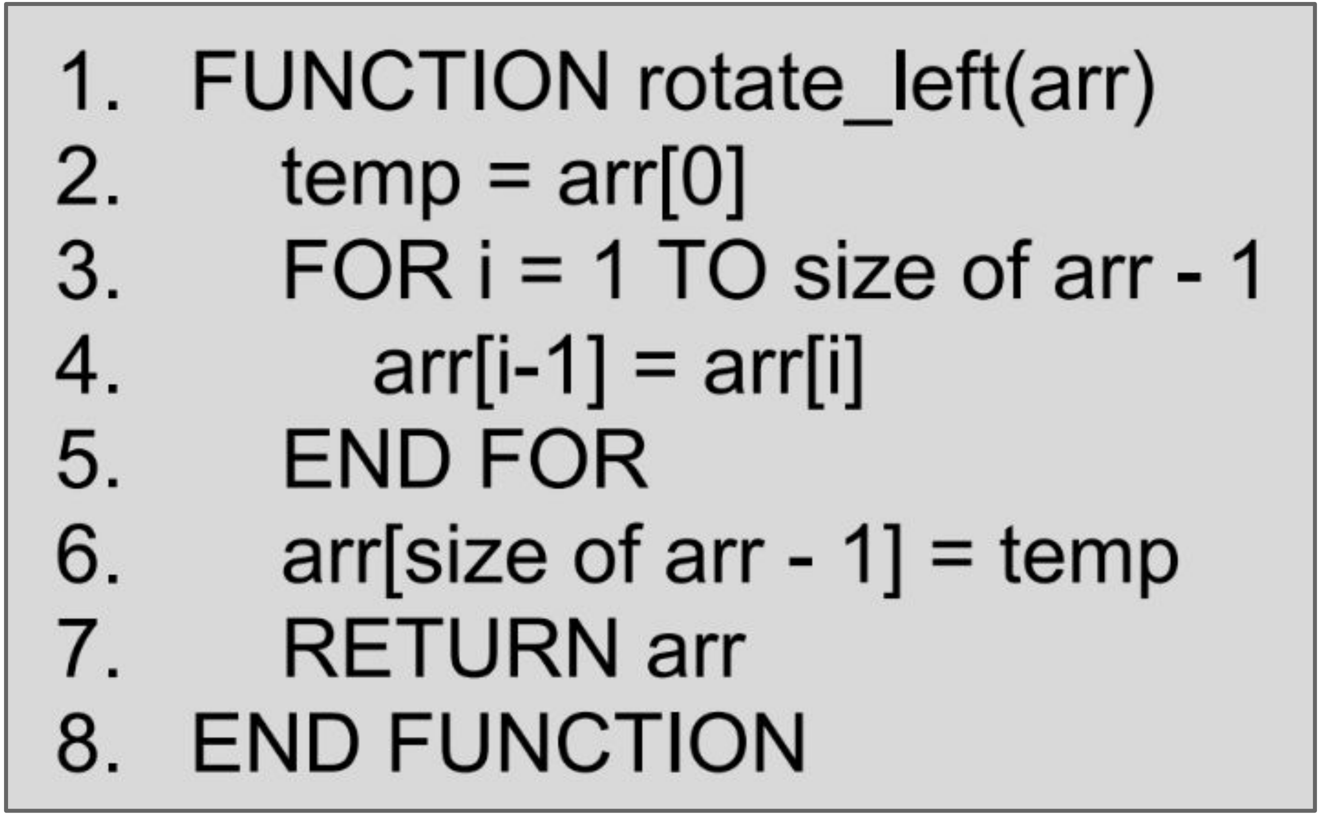
```
FUNCTION shift_right(arr)
    FOR i = size of arr - 1 DOWNTO 1
        arr[i] = arr[i-1]
    END FOR
    arr[0] = 0
    RETURN arr
END FUNCTION
```

Array - Functions (Shift Right)



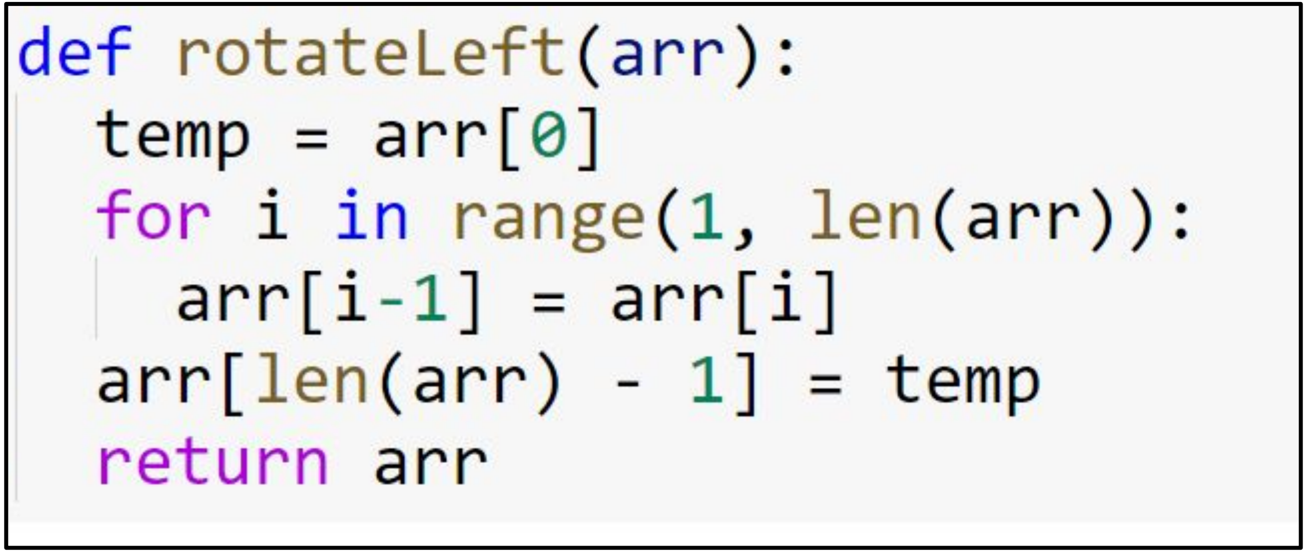
```python
def shiftRight(arr):
  for i in range(len(arr) - 1, 0, -1):
    arr[i] = arr[i - 1]
  arr[0] = None
  return arr
```

Array - Functions (Rotate Left)



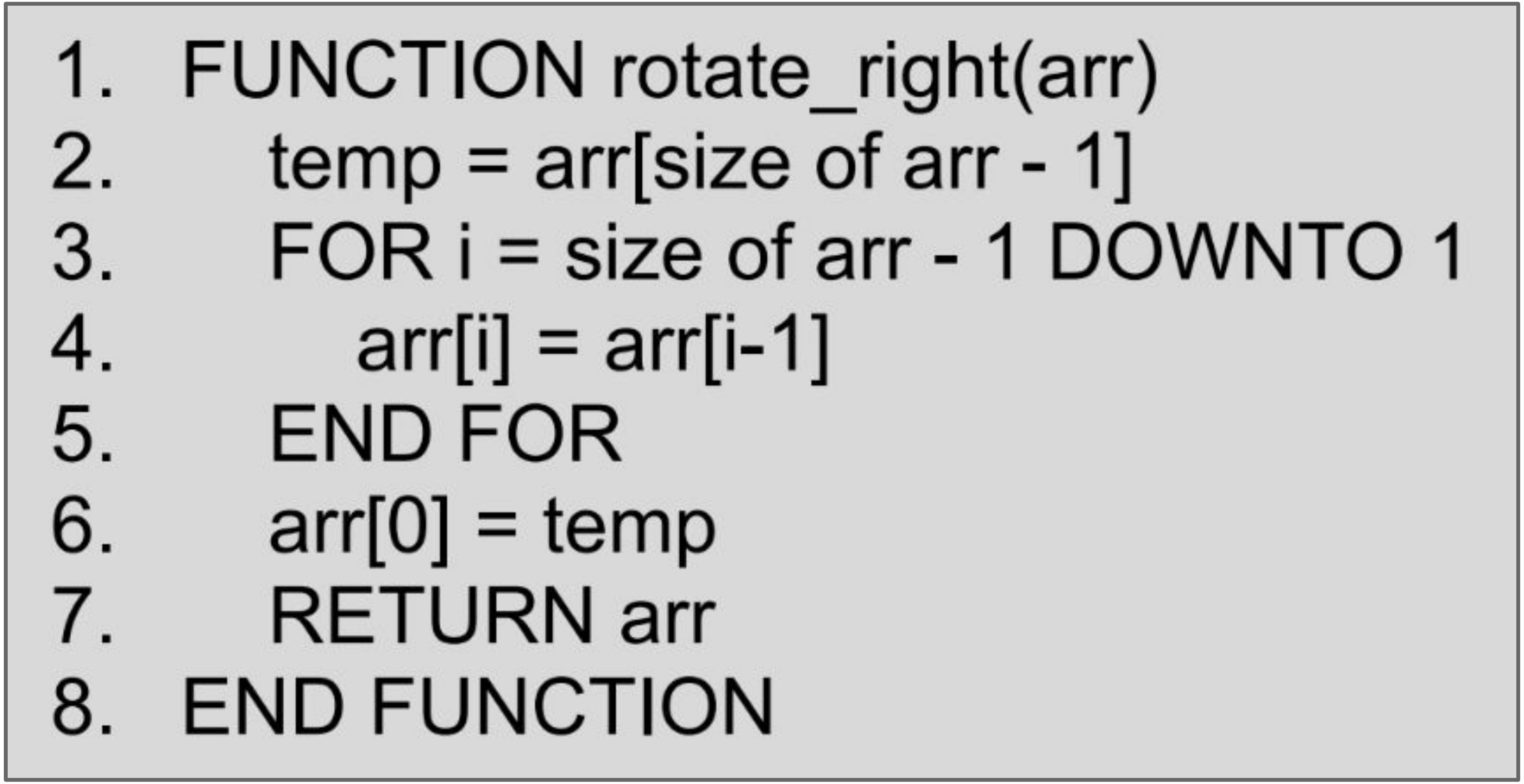
```
FUNCTION rotate_left(arr)
    temp = arr[0]
    FOR i = 1 TO size of arr - 1
        arr[i-1] = arr[i]
    END FOR
    arr[size of arr - 1] = temp
    RETURN arr
END FUNCTION
```

Array - Functions (Rotate Left)



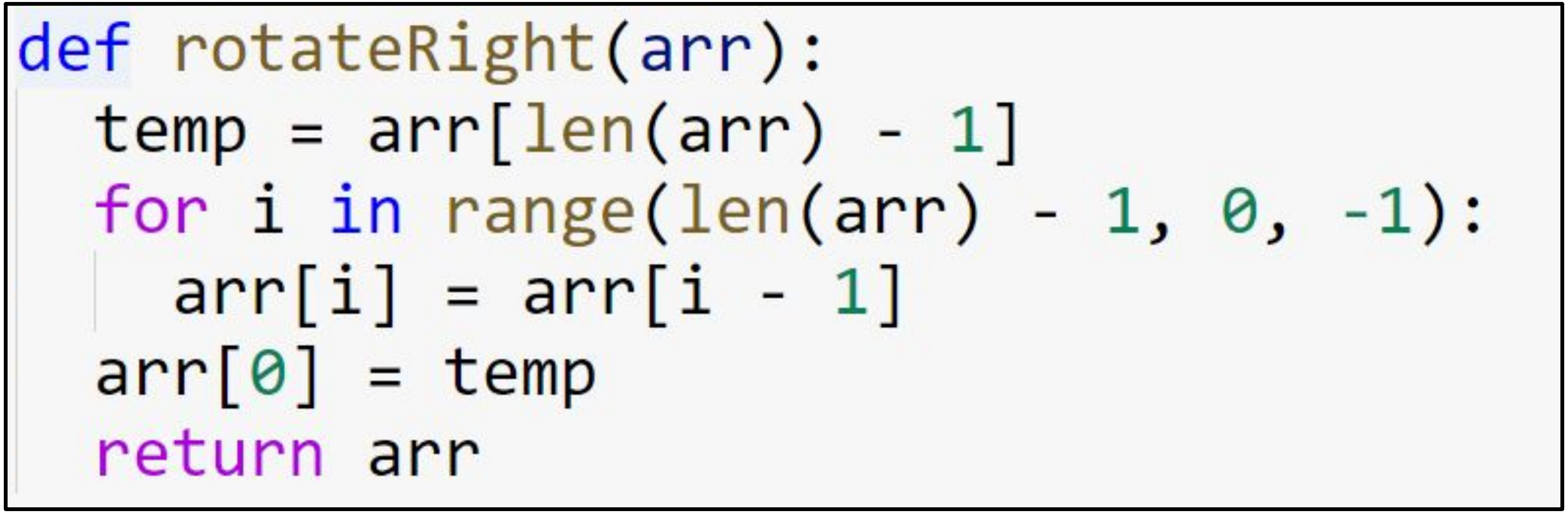
```python
def rotateLeft(arr):
    temp = arr[0]
    for i in range(1, len(arr)):
        arr[i-1] = arr[i]
    arr[len(arr) - 1] = temp
    return arr
```

Array - Functions (Rotate Right)



```
FUNCTION rotate_right(arr)
    temp = arr[size of arr - 1]
    FOR i = size of arr - 1 DOWNTO 1
        arr[i] = arr[i-1]
    END FOR
    arr[0] = temp
    RETURN arr
END FUNCTION
```

Array - Functions (Rotate Right)

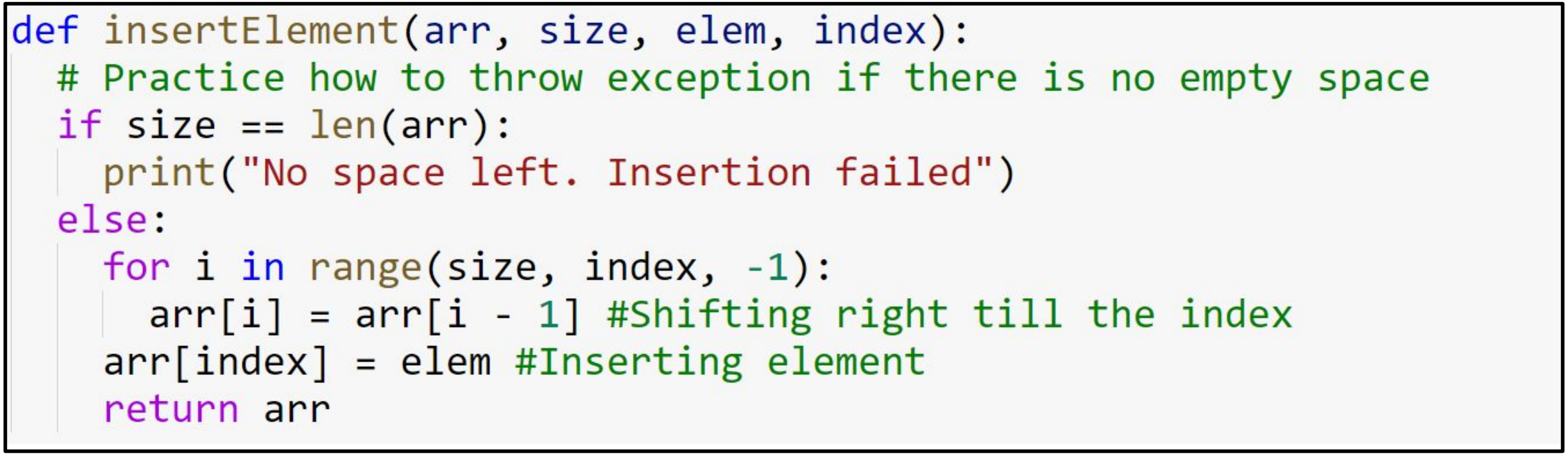


```python
def rotateRight(arr):
  temp = arr[len(arr) - 1]
  for i in range(len(arr) - 1, 0, -1):
    arr[i] = arr[i - 1]
  arr[0] = temp
  return arr
```

Array - Functions (Insert)

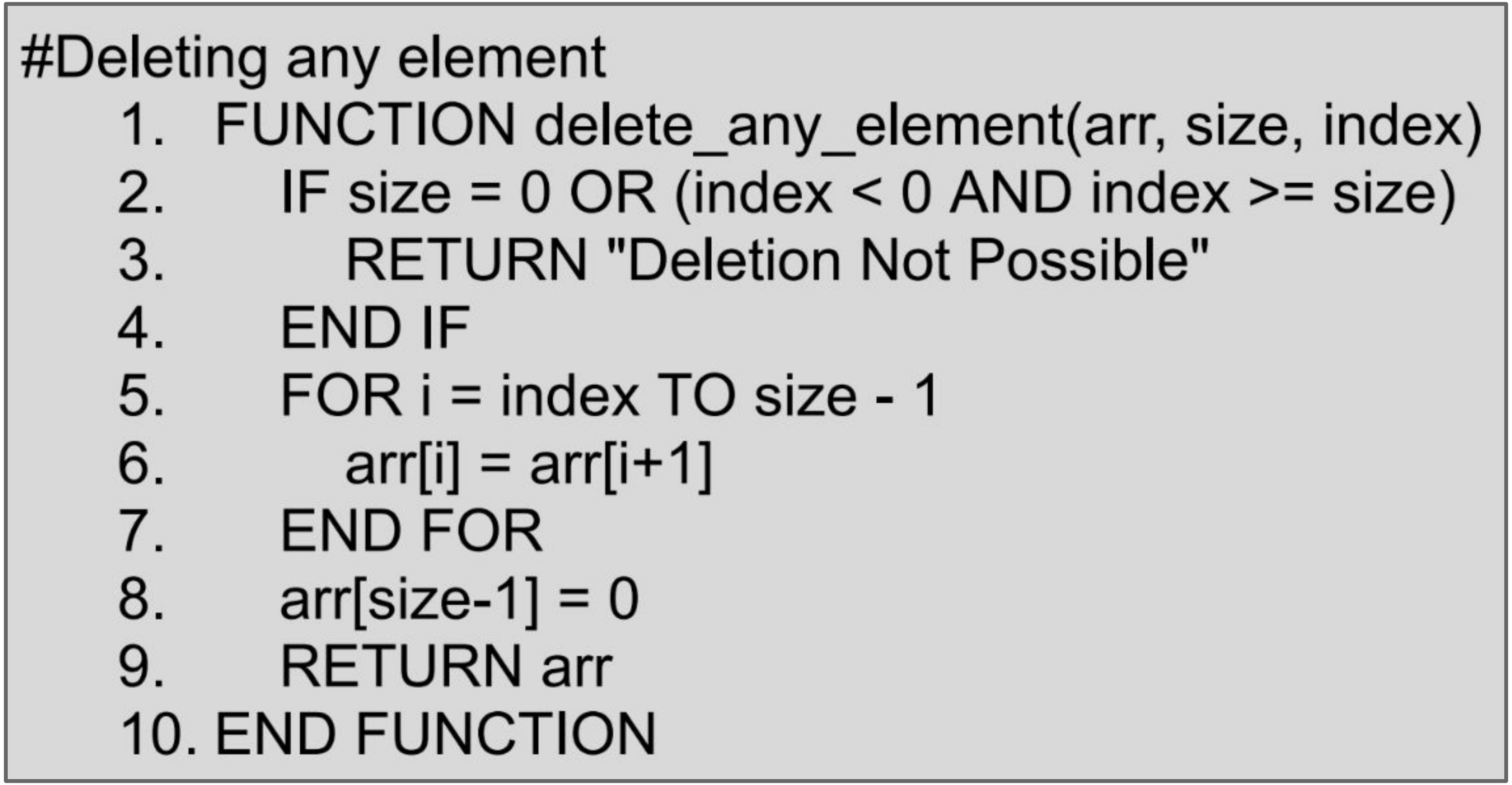
```
#Inserting anywhere
FUNCTION insert_anywhere(arr, size, index, elem)
    IF index < 0 OR index > size
        RETURN "Insertion Not Possible"
    END IF
    IF size >= size of arr
        arr = resize_array(arr, size of arr + 3)
    END IF
    FOR i = size DOWNTO index + 1
        arr[i] = arr[i-1]
    END FOR
    arr[index] = elem
    RETURN arr
END FUNCTION
```

Array - Functions (Insert)



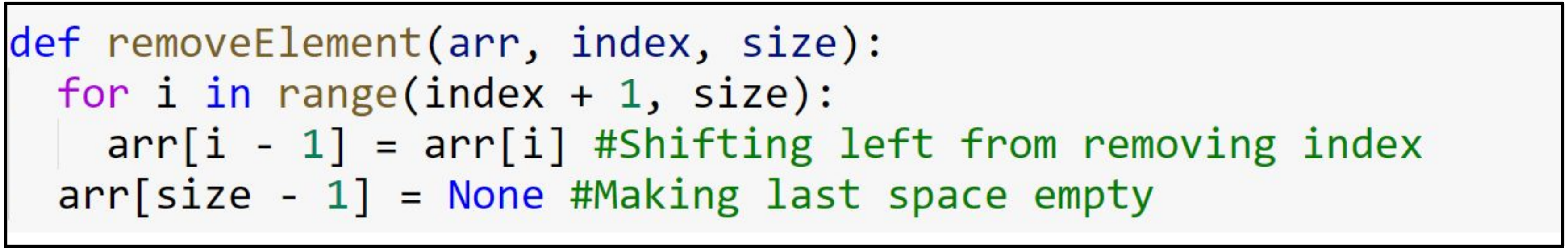
```python
def insertElement(arr, size, elem, index):
 # Practice how to throw exception if there is no empty space
 if size == len(arr):
  print("No space left. Insertion failed")
 else:
  for i in range(size, index, -1):
   arr[i] = arr[i - 1] #Shifting right till the index
  arr[index] = elem #Inserting element
  return arr
```

Array - Functions (Remove)



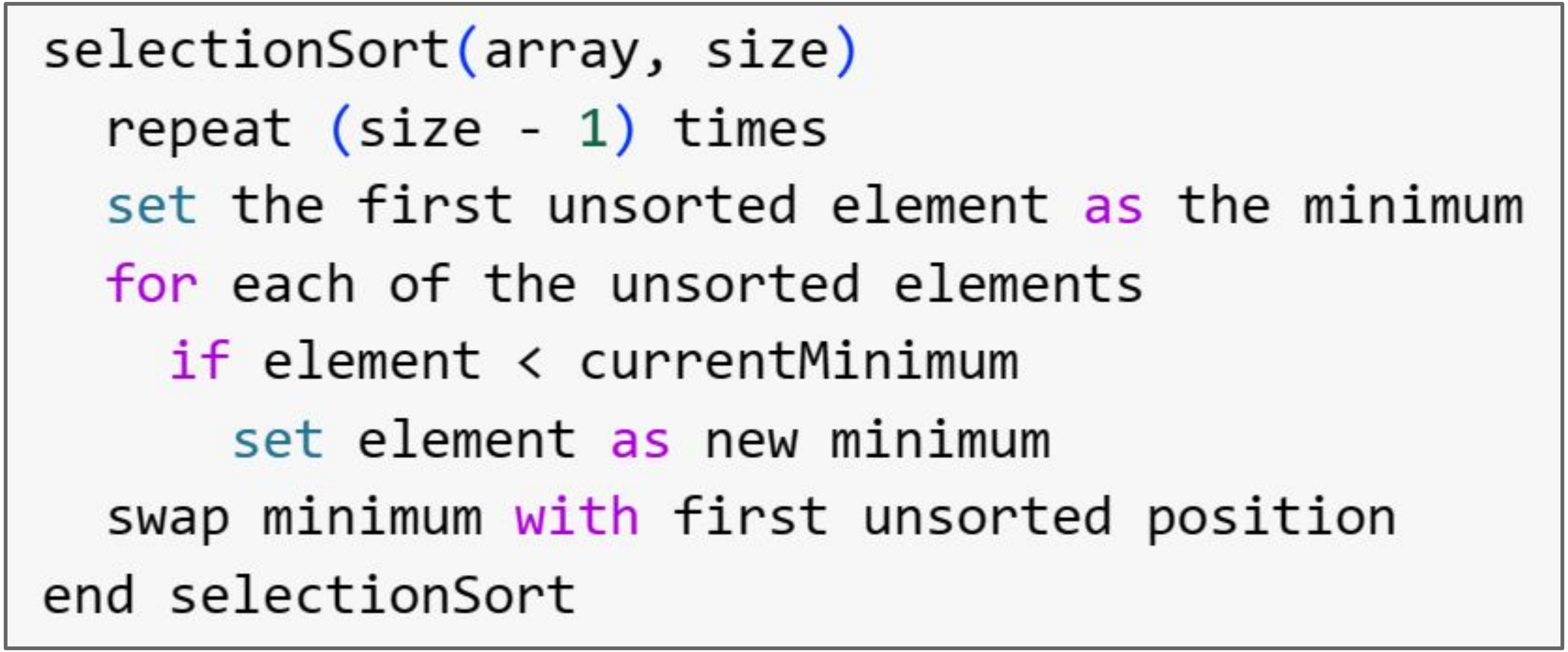
```
#Deleting any element
FUNCTION delete_any_element(arr, size, index)
    IF size = 0 OR (index < 0 AND index >= size)
        RETURN "Deletion Not Possible"
    END IF
    FOR i = index TO size - 1
        arr[i] = arr[i+1]
    END FOR
    arr[size-1] = 0
    RETURN arr
END FUNCTION
```

Array - Functions (Remove)



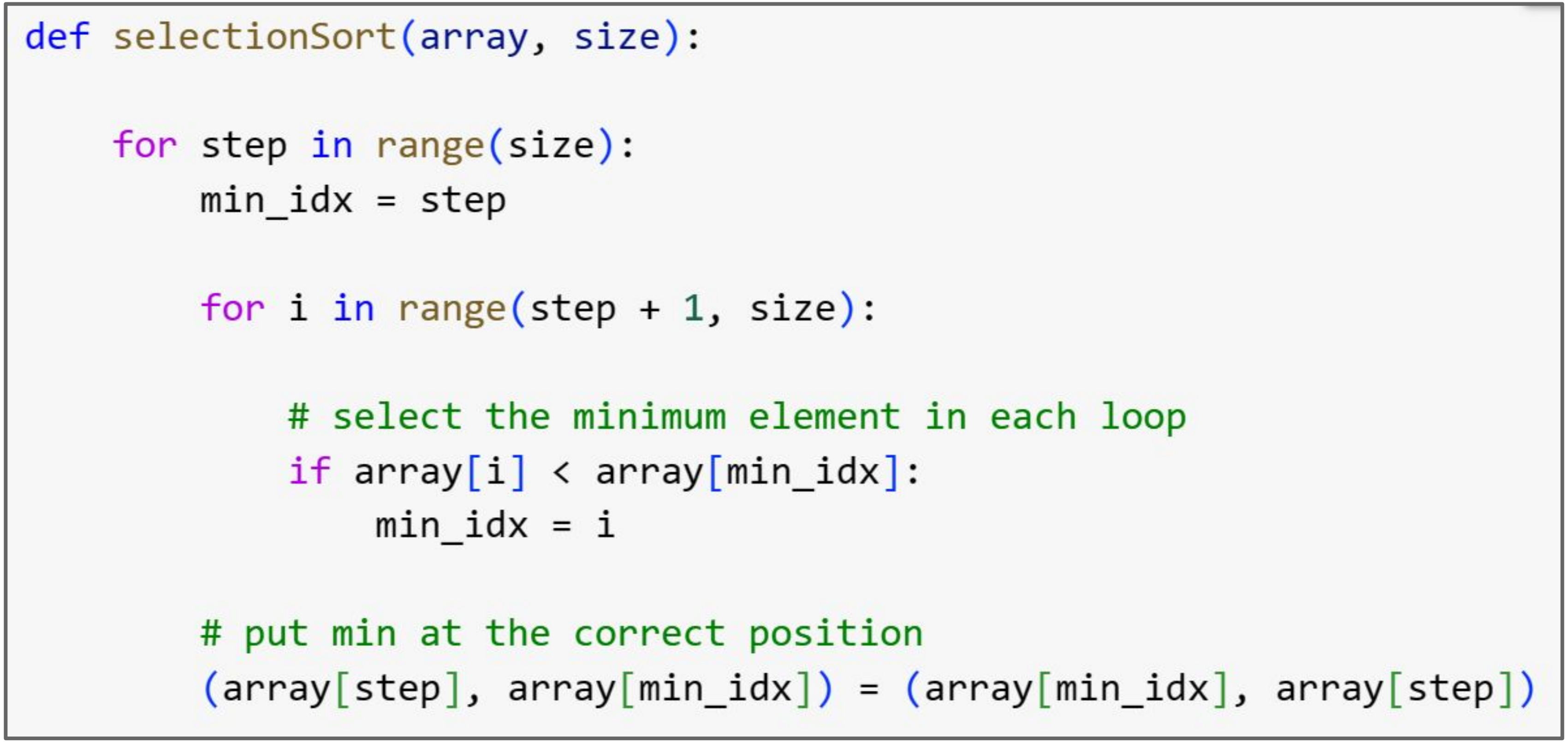
```python
def removeElement(arr, index, size):
  for i in range(index + 1, size):
    arr[i - 1] = arr[i] #Shifting left from removing index
  arr[size - 1] = None #Making last space empty
```

Array - Selection Sort



```
selectionSort(array, size)
  repeat (size - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selectionSort
```

Array - Selection Sort

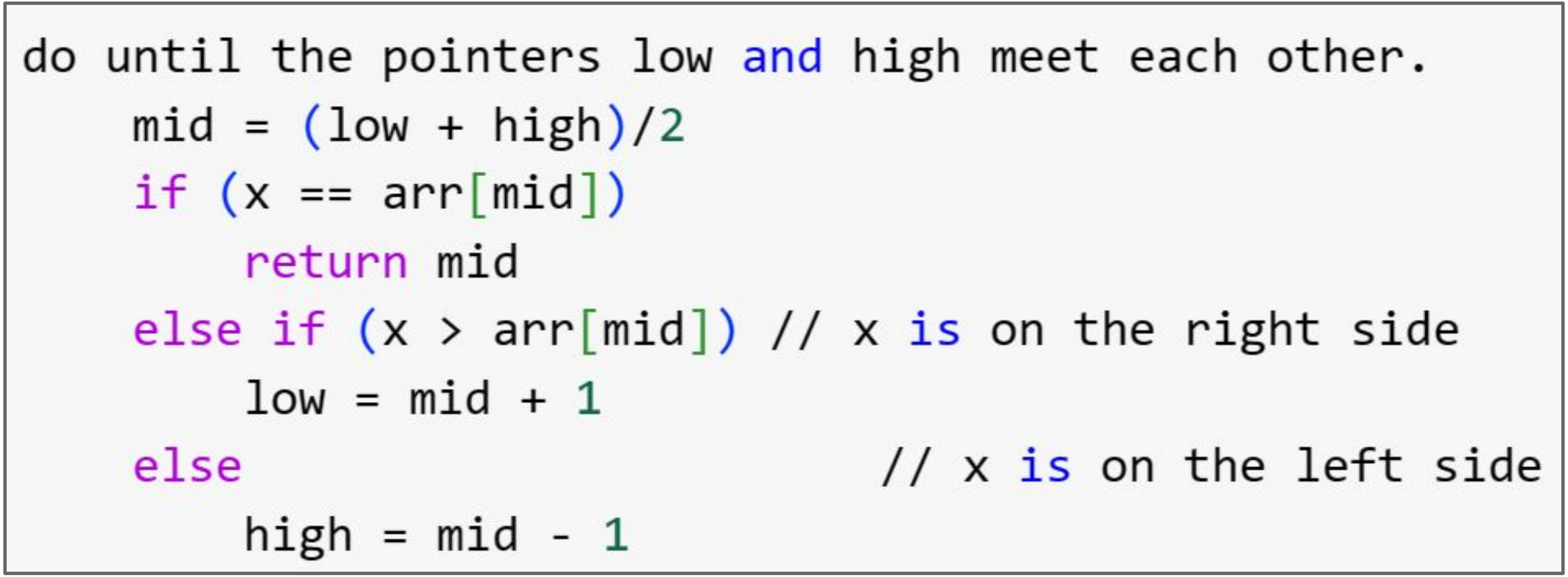


```python
def selectionSort(array, size):

    for step in range(size):
        min_idx = step

        for i in range(step + 1, size):

            # select the minimum element in each loop
            if array[i] < array[min_idx]:
                min_idx = i

        # put min at the correct position
        (array[step], array[min_idx]) = (array[min_idx], array[step])
```

Array - Bubble Sort

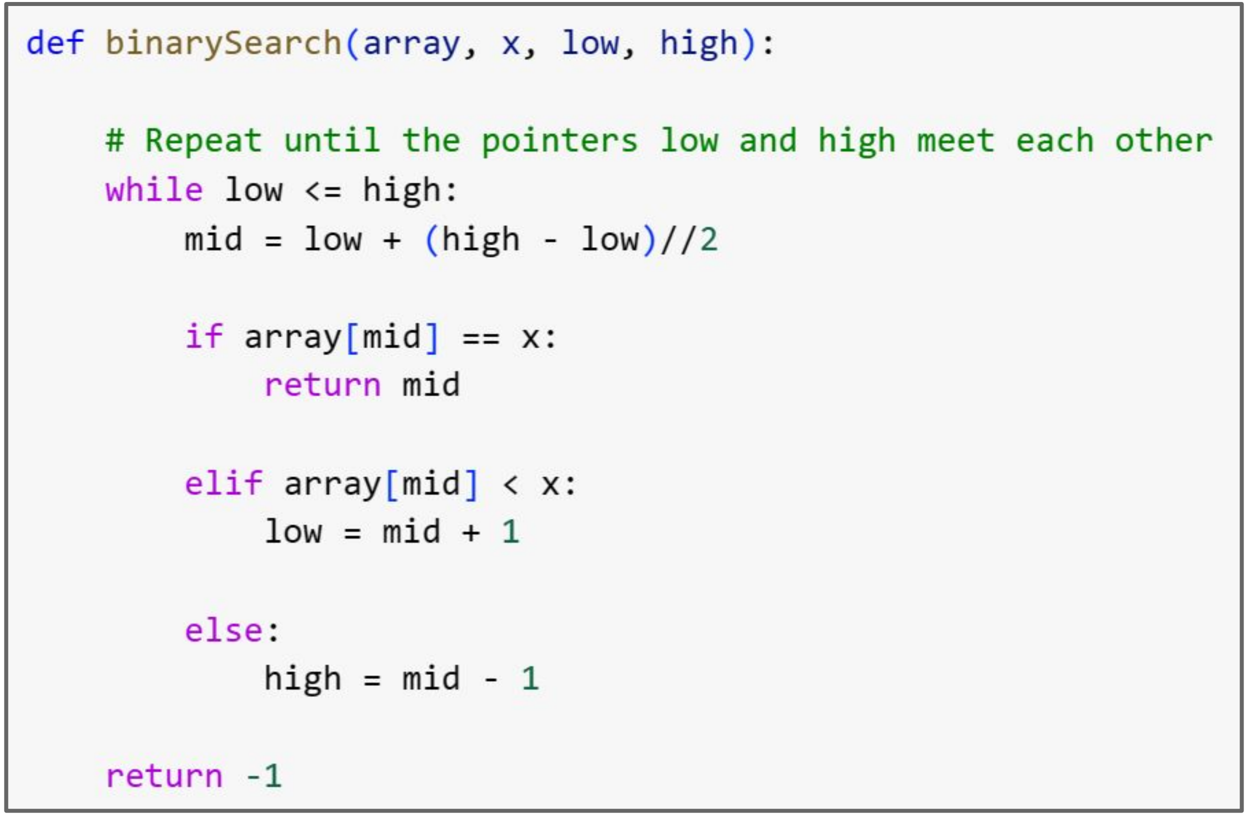
SELF STUDY

Array - Binary Search



```
do until the pointers low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                       // x is on the left side
        high = mid - 1
```

Array - Binary Search



```python
def binarySearch(array, x, low, high):

    # Repeat until the pointers low and high meet each other
    while low <= high:
        mid = low + (high - low)//2

        if array[mid] == x:
            return mid

        elif array[mid] < x:
            low = mid + 1

        else:
            high = mid - 1

    return -1
```