Tutorial 1 - Stacks and Queues

Exercise 1: Implementation of a stack and a queue

Give an implementation of a stack and of a queue by writing the following functions :

```
def create_stack():
    # return a new empty stack.
def push( P, e ):
    # Push the element 'e' in the stack 'P'.
def pop( P ):
    # Pop an element of the stack 'P' and return that element.
def size_of_stack( P ):
    # return the number of elements inside the stack 'P'.
def display_stack( P ):
    # display on the terminal the stack 'P' as a list.
def create_queue():
    # return a new empty queue.
def enqueue( F, e ):
    # Enqueue the element 'e' in the queue 'F'.
def dequeue( F ):
    # Dequeue an element of the queue 'F' and return that element.
def size_of_queue( F ):
    # return the number of elements inside the queue 'F'.
def display_queue( F ):
    # display on the terminal the queue 'F' as a list.
```

Draw the contents of the stack and the queue during the execution of the following programs. Give the content displayed by the terminal.

```
stack = create_stack()
push( stack, 1 )
push( stack, 2 )
push( stack, 3 )
print( size_of_stack( stack ) )
print( pop( stack ) )
print( pop( stack ) )
print( pop( stack ) )
```

```
queue = create_queue()
enqueue( queue, 1 )
enqueue( queue, 2 )
```

```
enqueue( queue, 3 )
print( size_of_queue( queue ) )
print( dequeue( queue ) )
print( dequeue( queue ) )
print( dequeue( queue ) )
```

Exercise 2

Suppose now, you are not allowed to use python list, python dictionary and python tuples. Suppose now, you just can use the following primitives :

```
def create_stack()
def push( P, e )
def pop( P )
def size_of_stack( P )
```

1) Give an implementation of queues using only stacks by writing the following primitives :

```
def create_queue()
def enqueue( F, e )
def dequeue( F )
def size_of_queue_queue( F )
```

2) Give an implementation of array using only the previous primitives. You will implement the following functions :

```
def create_array():
    # Create and return a new empty array.
def insert_element( T, id, e ):
    # Insert a new element 'e' at index 'id' in the array 'T'.
    # The size of the array should increase by 1.
def delete_element( T, id ):
    # Delete an element at index 'id' of the array 'T'.
    # The size of the array should decrease by 1.
def replace_element( T, id, e ):
    # Replace the element at index 'id' of 'T' by the element 'e'.
    # The size of the array should not change.
def get_element( T, id ):
    # Return the element at index 'id' of the array.
```

3) Give an implementation of dictionaries using the previous primitives. You will write the following functions :

```
def create_dictionary():
    # Create and return a dictionary
def insert_association( D, key, value ):
    # Add a new association ('key','value') in the dictionary 'D'.
    # If there exists an association ('key', 'value1') in 'D', then
    # this association should be replaced by ('key','value').
```

```
def delete_association( D, key ):
    # Delete the association of 'D' whose key is 'key'.
def in_dictionary( D, key ):
    # Return true is 'key' is a key of the dictionary 'D'.
def get_value( D, key ):
    # Return the value associated with 'key' in 'D'.
```

Exercise 3

Suppose now, the function `size_of_stack(P)` is not available. That means you just have access to the following function :

```
def create_stack()
def push( P, e )
def pop( P )
```

Make again the questions 1, 2 and 3 of the exercise 2 with that new constraint.