# CompSci 101 Assignment 3

## Some helpful information about Questions 4, 5 and 6

# A3 Q4 - `get_every_third_even_element()`

**parameter** - a list of integers

**returns** - a list of numbers which includes every third element if it is an even number starting from the **last** element in the list going down to the beginning of the list. The original list should not be changed.

```
list1 = [23, 12, 6, 5, 8, 9, 7, 4]
list2 = get_every_third_even_element(list1)
print("1.", list1)
print("2.", list2)
list1 = [22, 12, 6, 5, 11, 9, 7, 4]
list2 = get_every_third_even_element(list1)
print("3.", list1)
print("4.", list2)
```

```
1. [23, 12, 6, 5, 8, 9, 7, 4]
2. [4, 8, 12]
3. [22, 12, 6, 5, 11, 9, 7, 4]
4. [4, 12]
```

# A3 Q4 - `get_every_third_even_element()`

## How to create a new list, how to add elements to the new list.

## See Slide 2 for A3 Q1

### 1. Create an empty Python list:

```
result_list = []
```

### 2. Fill the list with the relevant elements using the `append()`

```
. . .
    result_list.append(an_element)
```

### 3. Usually the filled Python list is returned as a result of the function:

```
return result_list
```

# A3 Q4 - `get_every_third_even_element()`

## How to go through a list backwards? Using the index and a loop.

Think of the output to the following skeleton code using different values for the variables.

```
my_list = . . .     #a list of elements
index_last_element = . . . #the index of the last element

for index in range(index_last_element,
                           down_to_one_before_zero,
                                        step_downwards):
    print(index, end = " ")
```

If you have the index, the list element can be accessed using square brackets and the index number.

```
result_list = […]
index = … #a valid index for the list
element = result_list[index]
```

# A3 Q4 - `get_every_third_even_element()`

# How to test if a number is exactly divisible by a number.

## See Slide 3 for A3 Q1

**Check if there is zero remainder when the number is divided by another number, use the % operator**

```
number1 = . . . #some integer
number2 = . . . #some integer
if number1 % number2 == 0:
    #Process if number1 is exactly
    #divisible by number2
```

# A3 Q5 - get_sequencial_nums_sums()

**parameter** – a list of integers

**returns** – a new list where the first element is the sum of the first two elements of the parameter list, the second element is the sum of the next two elements of the parameter list, and so on.

```
list1 = [3, 3, 2, 3, 4, 3]
print("1. ", list1)
print("2. ", get_sequencial_nums_sums(list1))

list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print("3. ", list1)
print("4. ", get_sequencial_nums_sums(list1))
```

```
1.  [3, 3, 2, 3, 4, 3]
2.  [6, 5, 7]
3.  [1, 2, 3, 4, 5, 6, 7, 8, 9]
4.  [3, 7, 11, 15, 9]
```

# A3 Q5 - get_sequencial_nums_sums()

# How to create a new list, how to add elements to the new list.

## See Slide 2 for A3 Q1

### 1. Create an empty Python list:

```
result_list = []
```

### 2. Fill the list with the relevant elements using the `append()` method. Usually this happens over and over, i.e. inside a loop:

```
. . .
    result_list.append(an_element)
```

### 3. Usually the filled Python list is returned as a result of the function:

```
return result_list
```

# A3 Q5 - get_sequencial_nums_sums()

## How to go through a list skipping elements.

(Example list: `[3, 6, 8, 2, 7, 1, 2]`)

Think of the output to the following skeleton code using different values for the variables.

```
my_list = … #a list of elements
index = … #index of the second element in the list

while the_index_is_a_valid_index_for_the_list:
    print(index, end = " ")
    #move the index up by 2
```

If you have the index, the list element can be accessed using square brackets and the index number.

```
result_list = […]
index = … #a valid index for the list
element = result_list[index]
```

# A3 Q5 - get_sequencial_nums_sums()
# How to see both elements to be added at the same time.

Think of the output to the following skeleton code using different values for the variables.

```
my_list = … #a list of elements
index = … #index of the second element in the list

while the_index_is_a_valid_index_for_the_list:
    #In here you can access the element at index
    #you can access the element before it, index minus …

    print(index, end = " ")
    #move the index by 2
```

# A3 Q5 - get_sequencial_nums_sums() What if the list has an odd number of elements?

What to do if the length of the list is odd, e.g. for a list such as

```
[3, 6, 8, 2, 7, 1, 2]
```

Loop for index 1, then index 3, then index 5 (index 7 is not valid). The resulting list is

- Element at index 1 plus element at index 0
- Element at index 3 plus element at index 2
- Element at index 5 plus element at index 4

```
[9, 10, 8]
```

If the list has an odd number of elements then the last element of the parameter list needs to be appended to the resulting list.

```
[9, 10, 8, 2]
```

# A3 Q6 - `remove_triplets()`

**parameters** – a list of integers

**returns** – None

The function removes triplets made up of three sequential identical elements

```
a_list = [6, 6, 6, 7, 6, 6, 6, 3, 3, 3, 8, 8, 8, 3]
remove_triples(a_list)
print("1.", a_list)

a_list = [6, 6, 6, 7, 6, 6, 6, 6, 6]
remove_triples(a_list)
print("2.", a_list)

a_list = [6, 6, 6, 7, 6, 6, 4, 3, 3, 3, 8, 8, 8, 3]
remove_triples(a_list)
print("3.", a_list)

a_list = [1, 1, 1, 4, 4, 4, 1, 1, 1]
remove_triples(a_list)
print("4.", a_list)
```

```
1. [7, 3]
2. [7, 6, 6]
3. [7, 6, 6, 4, 3]
4. []
```

# A3 Q6 - `remove_triplets()`
# Need to look at the indices.

In CompSci 101 we use the `pop()` method to remove an element at a given index from the list.

The `pop()` method requires the index number as an argument.

```
result_list = […]
index = … #a valid index for the list
result_list.pop(index)
```

When using the `pop()` method to remove elements you need to process the list from the end of the list.

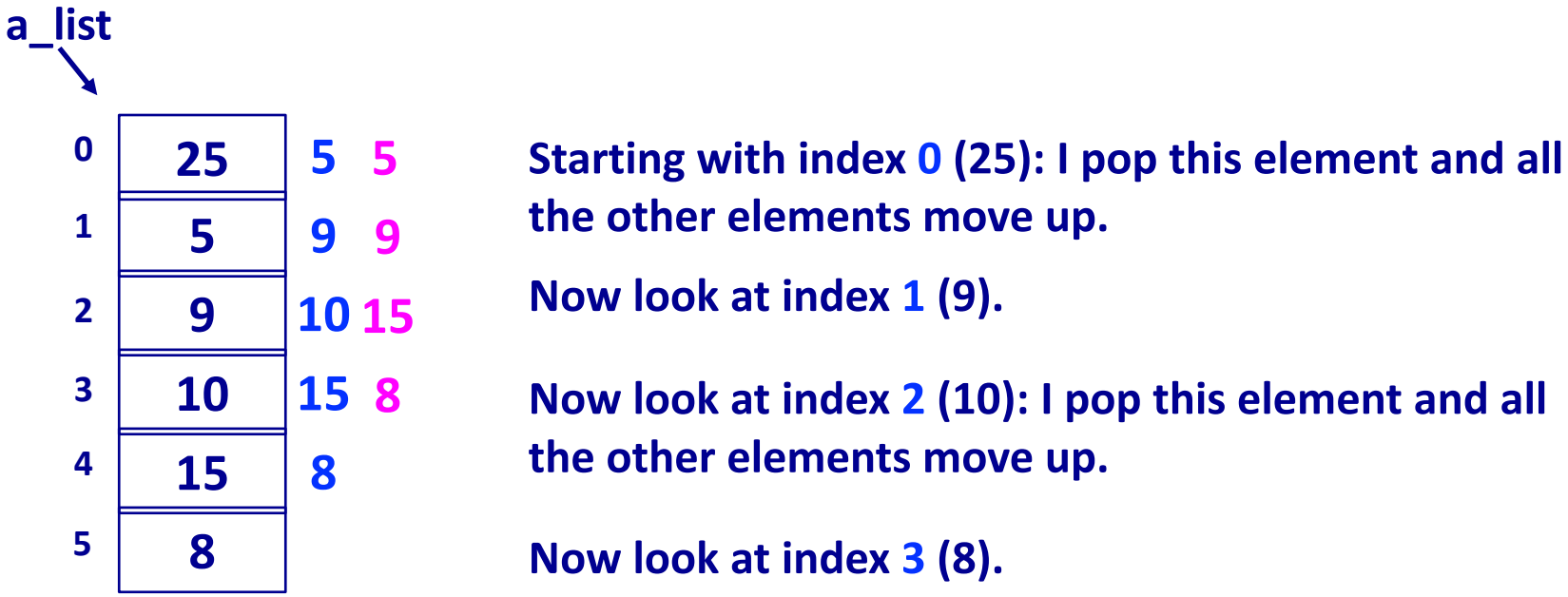
```
my_list = … #a list of elements
index = … #the last index of the list

while the_index_is_a_valid_index:
      …
     #move the index down the list
```

# A3 Q6 - `remove_triplets()`

## Why should you NOT go forwards through the list when popping elements?

Let's say I wish to remove any element which is exactly divisible by 5 in the following list. If I look at index 0, then 1, then 2, then 3, then 4, then 5. I have a problem.

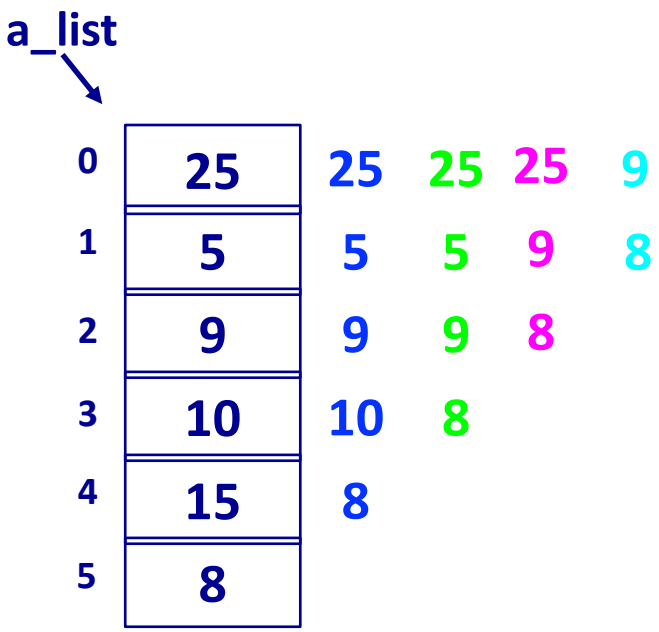



**Problem**: because popping the elements means that the elements are moved up the list, this means that I am not checking all the elements.

# A3 Q6 - `remove_triplets()`

## Why do you go backwards through the list when popping elements?

If instead I process the list from the end. If I look at index 5, then 4, then 3, then 2, then 1, then 0. I do not have a problem.




**Starting with index 5 (8).**

**Now look at index 4 (15): I pop this element and the last element moves up.**

**Now look at index 3 (10): I pop this element and the last element moves up.**

**Now look at index 2 (9).**

**Now look at index 1 (5): I pop this element and all the other elements move up.**

**Now look at index 0 (25): I pop this element and all the other elements move up.**

# A3 Q6 - `remove_triplets()`
# Need a while loop.

This type of problem needs a `while` loop.

Why?

Don't know which sequence of indices you need to access because you don't know where the triples are located in the list.

```
my_list = … #a list of elements
index = … #the last index of the list

while the_index_is_a_valid_index:
    #In here you can access the element at index
    #the element before it, at index minus …
    #the element two before it at index minus …

    #move the index depending on whether the
    #elements are equal or not
```

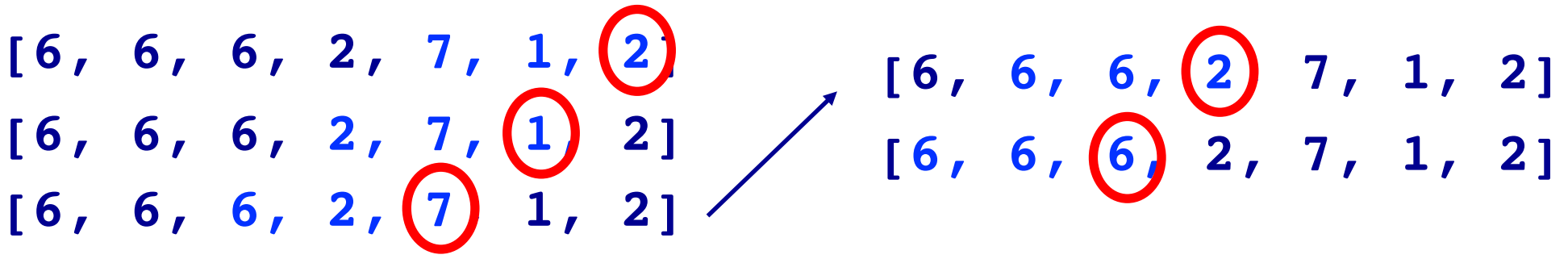
# A3 Q6 - `remove_triplets()`

## Inside the loop body access 3 elements of the list at a time.

```
my_list = … #a list of elements
index = … #the last index of the list

while the_index_is_a_valid_index_for_the_list:
    #In here you can access the element at index
    #the element before it, at index minus …
    #the element two before it at index minus …

    #move the index down the list
```



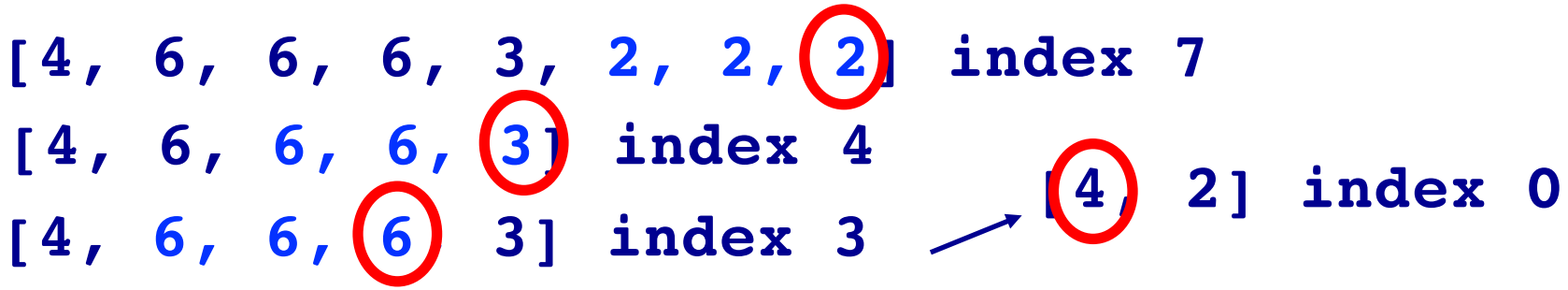
# A3 Q6 - `remove_triplets()`

## How much do you need to change the index if the three elements are equal. If they are not equal?

```
my_list = … #a list of elements
index = … #the last index of the list

while the_index_is_a_valid_index_for_the_list:
    #In here you can access the element at index
    #the element before it, at index minus …
    #the element two before it at index minus …

    #move the index depending on whether the
    #elements are equal or not
```



# A3 Q6 - `remove_triplets()`

## What is the condition in the `while` loop header? When to stop?

```
my_list = … #a list of elements
index = … #the last index of the list

while the_index_is_a_valid_index_for_the_list:
    #In here you can access the element at index
    #the element before it, at index minus …
    #the element two before it at index minus …

    #move the index depending on whether the
    #elements are equal or not
```

```
[4, 6, 6, 5, 3, 2, 2, 2, …] #Need to check
[4, 3, …] #Don't want to continue
[4, 2, …] #Don't want to continue
```