Creative Exercise 1a: Drawing a Triangle

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

## Defining Functions (a.k.a. Methods)

The following four pieces of code are all equivalent in Python, MATLAB, Scheme, and Java. Each defines a function that returns the maximum of two values and then prints the maximum of 5 and 15.

### Python

def max(x, y):

if (x > y):

return x

return y

print(max(5, 15))

### MATLAB

function m = max(x, y)

if (x > y)

m = x

else

m = y

end

end

disp(max(5, 15))

### Scheme

(define max (lambda (x y) (if (> x y) x y)))

(display (max 5 15)) (newline)

### Java

public static int max(int x, int y) {

if (x > y) {

return x;

}

return y;

}

public static void main(String[] args) {

System.out.println(max(10, 15));

}

## Arrays

Optional Supplementary Reading: [*Shewchuk*](https://sp18.datastructur.es/materials/hw/hw0/hw0_supplementary_arrays.txt)

Our final new syntax item of this HW is the array. Arrays are like vectors in Scheme, lists in Python, and arrays in MATLAB.

The following four programs in Python, MATLAB, Scheme, and Java declare a new array of the integers 4, 7, and 10, and then prints the 7.

### Python

numbers = [4, 7, 10]

print(numbers[1])

### MATLAB

numbers = [4 7 10]

disp(numbers(2))

### Scheme

(define numbers #(4 7 10))

(display (vector-ref numbers 1)) (newline)

### Java

int[] numbers = new int[3];

numbers[0] = 4;

numbers[1] = 7;

numbers[2] = 10;

System.out.println(numbers[1]);

Or in an alternate (but less general) shorthand:

### Alternate Java

int[] numbers = new int[]{4, 7, 10};

System.out.println(numbers[1]);

([program link](http://goo.gl/U6xLky))

You can get the length of an array by using .length, for example, the following code would print 3:

int[] numbers = new int[]{4, 7, 10};

System.out.println(numbers.length);

## Optional: Exercise 4

This is a particularly challenging exercise, but strongly recommended.

Write a function windowPosSum(int[] a, int n) that replaces each element a[i] with the sum of a[i] through a[i + n], but only if a[i] is positive valued. If there are not enough values because we reach the end of the array, we sum only as many values as we have.

For example, suppose we call windowPosSum with the array a = {1, 2, -3, 4, 5, 4}, and n = 3. In this case, we’d:

* Replace a[0] with a[0] + a[1] + a[2] + a[3].
* Replace a[1] with a[1] + a[2] + a[3] + a[4].
* Not do anything to a[2] because it’s negative.
* Replace a[3] with a[3] + a[4] + a[5].
* Replace a[4] with a[4] + a[5].
* Not do anything with a[5] because there are no values after a[5].

Thus, the result after calling windowPosSum would be {4, 8, -3, 13, 9, 4}.

As another example, if we called windowPosSum with the array a = {1, -1, -1, 10, 5, -1}, and n = 2, we’d get {-1, -1, -1, 14, 4, -1}.

public class BreakContinue {

public static void windowPosSum(int[] a, int n) {

/\*\* your code here \*/

for (int i = 0; i < a.length; i++) {

if (a[i] < 0) {

continue;

}

int limit = i + n + 1;

if (limit > a.length) {

limit = a.length;

}

for (int j = i+1; j < limit; j++) {

a[i] += a[j];

}

}

}

public static void main(String[] args) {

int[] a = {1, 2, -3, 4, 5, 4};

int n = 3;

windowPosSum(a, n);

// Should print 4, 8, -3, 13, 9, 4

System.out.println(java.util.Arrays.toString(a));

}

}