Homework 6
Seth Denney

1.) Add the method outside: thru: to the small integer class that returns true if the small integer k is outside: m thru: n (e.g. 4 outside: 5 thru: 45 should be true since 4 < 5 or 4 > 45 is true

```
outside: low thru: high
        | n l h |
        n := self.
        l := low.
        h := high.
        ^((n < l) | (n > h))
```

2.)Add these messages to extend the Array class to a new class MyArray of integers

a) Implement the message addFirst that when sent to the array of integers will return an array in which the element has been added to the array.

#(7 6 5) addFirst: 8 => (8 7 6 5)

```
addFirst: element
        |n arr|
        n := Array new: 1.
        n at: 1 put: element.
        arr := self.
        ^(n, self)
```

b) Implement the message altElement that returns an array containing every other element of the list, starting with the first.

#(1 4 9 11 25) altElement => (1 9 25)

```
altElement
        |i arr s n result|
        i := 3.
        arr := self.
        s := arr size.
        (s < 2) ifTrue: [^arr].
        result := Array new: 1.
        result at: 1 put: (arr at: 1).
        [i <= s] whileTrue: [
                        n := Array new: 1.
                        n at: 1 put: (arr at: i).
                        result :=  result append: result with: n .
                        i := i + 2.
                        ].
        ^result.

append: a with: b
        ^(a, b)
```

c) Implement a message altElement2 that returns an array containing every other element of the list starting with the second.

#(1 4 9 11 25) altElement2 => (4 11)

```
altElement2
        |i arr s n result|
        i := 4.
        arr := self.
        s := arr size.
        (s < 2) ifTrue: [^#()].
        result := Array new: 1.
        result at: 1 put: (arr at: 2).
        (s < 4) ifTrue: [^result].
        [i <= s] whileTrue: [
                        n := Array new: 1.
                        n at: 1 put: (arr at: i).
                        result :=  result append: result with: n .
                        i := i + 2.
                        ].
        ^result.

append: a with: b
        ^(a, b)
```

d) Write (or find) a method named max that will return the maximum value in an array of integers.

#(2 32 4) max: => 32

```
max
        |max arr s i|
        arr := self.
        s := arr size.
        (s < 1) ifTrue: [^nil].
        max := arr at: 1.
        i := 1.
        [i <= s] whileTrue: [
                ((arr at: i) > max) ifTrue: [max := arr at: i].
                i := i + 1.].
        ^max
```

3.) In Smalltalk, a block may be used to define a single valued function. For example:

```
|f|
f := [:x | 3 * x].        to define a function f(x) = 3x .
f value: 4.               will cause an evaluation yielding 12.
```

A block is an instance of the BlockContext class. So a BlockContext object should be able to receive a knice: a to: b for: k and return true if for some integer x between a and b and some integer n <= k, n applications of the block function will yield x. Extend the BlockContext class to a BlockFunction class that contains the knice method.

```
knice: a to: b for: k
        |x n r|
        x := a.
        [x <= b] whileTrue: [
                n := 1.
                r := x.
                [n <= k ] whileTrue: [
                        r := self value: r.
                        (r == x) ifTrue: [^true].
                        n := n + 1.
                ].
                x := x + 1.
        ].
        ^false.
```