1. Implement a function make-clock that builds a clock object with the following behavior:

```
(define c1 (make-clock))
(define c2 (make-clock))

(c1) => ‘tick
(c1) => ‘tock
(c1) => ‘tick
(c1) => ‘tock
(c2) => ‘tick
(c1) => ‘tick
(c2) => ‘tock
(c1) => ‘tock
```

2. Implement a function previous. An example of its behavior is as follows:

```
(define foo (previous (lambda (y) y))

(foo 1) => #f
(foo 2) => 1
(foo 3) => 2
(foo 1) => 3

(define foo (previous (lambda (y) (square y))))

(foo 1) => ‘false
(foo 2) => 1
(foo 3) => 4
(foo 1) => 9
```

So previous takes a single-argument procedure f as its one argument, and returns an object that always returns the previous value of (f x). The object returns #f the first time it is called.

```
(define (previous f)
```
3. Object-oriented stacks. Define a function `create-stack` that creates a stack object. Initially the stack should be empty. The function should support `push`, `pop`, `peek`, and `clear`. (push adds an item on the stack; pop removes the top item on the stack, and returns its value; peek returns the value of the top item, but does not remove it; clear resets the stack to be empty.)

The stack should be implemented internally using a list data-structure. An example usage:

```
(define s (create-stack))
(s 'push 5)
(s 'push 3)
(s 'pop) => 3
(s 'push 1)
(s 'pop) => 1
(s 'pop) => 5
```