Question 4

We would like to design a function that consumes a person and produces a list of the names of all the people in the tree under 20 ("in the tree" includes the original person).

```
(define P1 (make-person "N1" 5 empty))
(define P2 (make-person "N2" 25 (list P1)))
(define P3 (make-person "N3" 15 empty))
(define P4 (make-person "N4" 45 (list P2 P3)))
```

Here is a partial design:

```
;; Person -> ListOfString
;; ListOfPerson -> ListOfString
;; produce a list of the names of the persons under 20

(check-expect (names-under-20--person P1) (list "N1"))
(check-expect (names-under-20--lop empty) empty)
(check-expect (names-under-20--person P2) (list "N1"))
(check-expect (names-under-20--person P4) (list "N3" "N1"))
```

Question 4

1/1 point (graded)

Complete the function design for `names-under-20--person` and `names-under-20--lop`.

```
(define (names-under-20--person p)
  (if (< (person-age p) 20)
      ( cons  (person-name p)
              (names-under-20--lop (person-children p)))
      (names-under-20--lop (person-children p))))

(define (names-under-20--lop lop)
  (cond [(empty? lop) empty)]
        [else
         (append (names-under-20--person (first lop))
                 (names-under-20--lop (rest lop)))]))
```

append	cons

✓

Submit

✓ Correct (1/1 point)