```
for i in lon:
    if i > n:
        return true
return false
```

Every recursive program consists of:

- one or more **base cases** that return a pre-defined answer
- one or more **recursive cases** that compute solutions in terms of simpler problems

The recursive case consists of three steps:

1. Split the data into smaller pieces.

2. Solve the pieces.

3. Combine the solutions for the parts into a single answer.

1. Split the data into smaller pieces.
   ... based on **the type of the argument**

2. Solve the pieces.
   ... the "big" sub-problem is
   topologically similar to the original

3. Combine the solutions for the parts
   into a single answer.
   ... based on **the type of function's value**

When writing a program to process an inductively-defined data type,

the structure of the program should follow the structure of the data.

```
<list-of-numbers>
    ::= ()
      | (<number> . <list-of-numbers>)
```

```
<list-of-numbers>
    ::= ()
      | (<number> . <list-of-numbers>)
```

```
(26 37 41 25 12)
```

```
<list-of-numbers>
    ::= ()
      | (<number> . <list-of-numbers>)
```

```
(26   .   (37 41 25 12))
```

```
<list-of-numbers>
    ::= ()
      | (<number> . <list-of-numbers>)
```

```
(26   .   (37 41 25 12))

(26 . <list-of-numbers>)
```

```
<list-of-numbers>
    ::= ()
      | (<number> . <list-of-numbers>)
```

```
(26   .   (37 41 25 12))
```

```
(26 . <list-of-numbers>)
```

```
26    <list-of-numbers>
```

```
<list-of-symbols>
    ::= ()
      | (<symbol> . <list-of-symbols>)
```

```
(a  b  a  c  d))
```

```
(a   .   (b  a  c  d))
```

```
(a   .   <list-of-symbols>)
```

```
(remove-first 'b '(a b c d))
```

```
(remove-first 'b '(b c d))
```