Big-O notation

CS135 Lecture 08

L08.0 Leveling up

List abbreviations

The expression

(cons v_1 **(cons** v_2 **(...** **(cons** v_n **empty)...)))**

can be abbreviated as

(list v_1 v_2 ... v_n**)**

For example

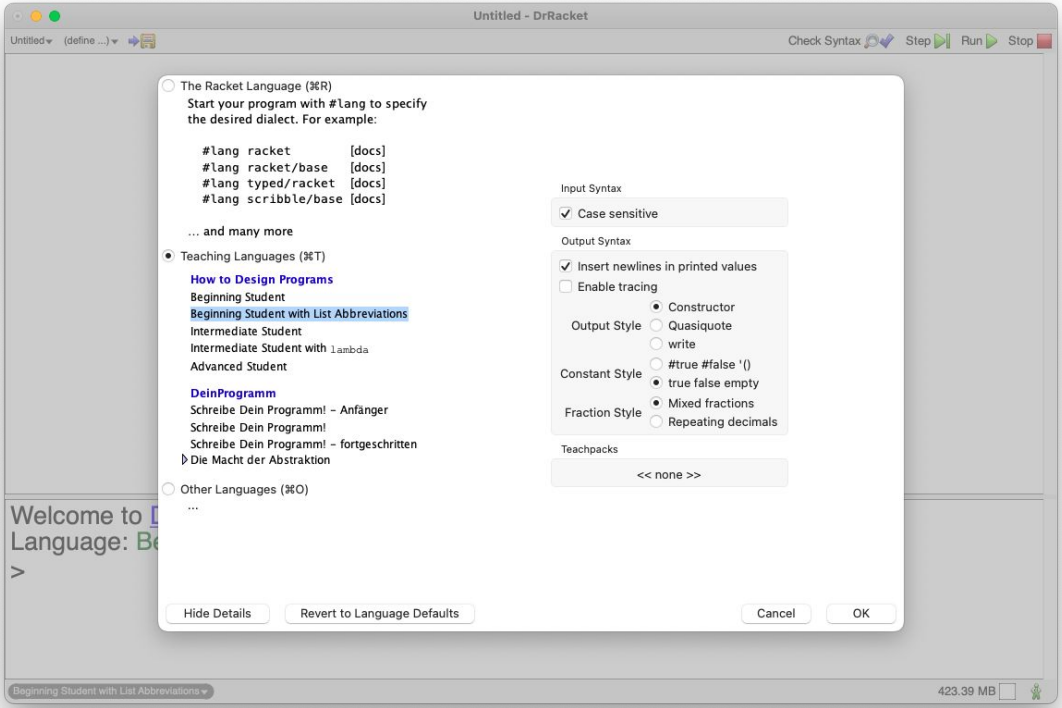
```
(cons 4 (cons 3 (cons 2 (cons 1 empty)
```

can be abbreviated as

```
(list 4 3 2 1)
```

To use list abbreviations we have to adjust our language level.

Adjusting the language level



"Beginning student with List Abbreviations"

You must have exactly this setting for the count of the steps to be the same as in the examples that follow.

L08.1 Measuring efficiency

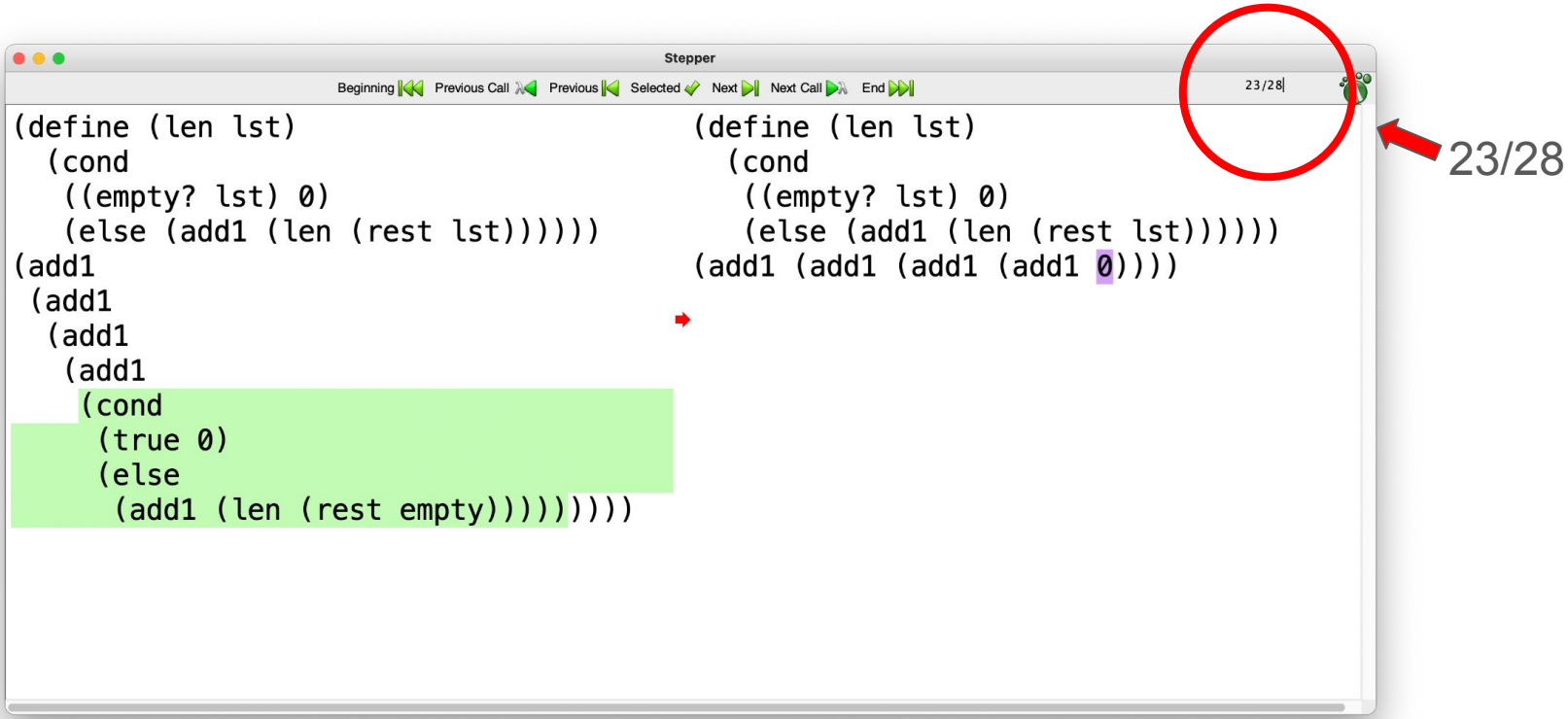
To measure efficiency, we count substitution steps

```
(define (len lst)
  (cond [(empty? lst) 0]
        [else (add1 (len (rest lst)))]))
```

`(len empty)` ⇒ 4 steps
`(len (list 1))` ⇒ 10 steps
`(len (list 1 2))` ⇒ 16 steps
`(len (list 1 2 3))` ⇒ 22 steps
`(len (list 1 2 3 4))` ⇒ 28 steps

If n is the length of the list, number of steps = $f(n) = 6n + 4$

To measure efficiency, we count substitution steps



Adding one to each element of a list of numbers

```
(define (add1-list lst)
  (cond [(empty? lst) empty]
        [else (cons (add1 (first lst))
                    (add1-list (rest lst)))]))
```

`(add1-list empty)` ⇒ 4 steps
`(add1-list (list 0))` ⇒ 12 steps
`(add1-list (list 0 1))` ⇒ 20 steps
`(add1-list (list 0 1 2))` ⇒ 28 steps
`(add1-list (list 0 1 2 3))` ⇒ 36 steps

A substitution step is a unit of time.

The efficiency of a computation is usually described in terms of the “time” it takes for the computation to complete.

While this time could be measured in milliseconds, nanoseconds, or even hours, it's often measured more abstractly—as the number of some basic computational operation required to complete the computation, such as substitution steps.

When we talk about the number of substitution steps required to perform a computation, it’s normal (and even common) to talk about the “time” taken to complete the computation.

We are usually interested in the worst case time to perform a computation in terms of *n*, where *n* is often a measure of the data size, such as the length of a list.

L08.2 Math class

Huh? Why are there no slides?

We treat this module as a traditional math class.

Your instructor will write on the board.

Practical examples and a summary of key ideas for assignments and exams will be provided in Lecture 09 (the next lecture).

If you work with Accessibility Services. They will have a copy of the notes for you.

L08.3 Applying Big-*O*

Big-*O* in practice

Name	Big-O Notation
Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
"n log n"	$O(n \log n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$

Since a substitution step is a unit of time, we often talk about the "time complexity" of a function.

For example, **`add1-list`** has linear time complexity.

In the next lecture, we will look at specific examples of linear, quadratic and exponential functions.

Efficiency of built-in **`length`** vs. our **`len`** function

`(length empty)` ⇒ 1 step
`(length (list 1))` ⇒ 1 step
`(length (list 1 2))` ⇒ 1 step
`(length (list 1 2 3))` ⇒ 1 step
`(length (list 1 2 3 4))` ⇒ 1 step

Built-in functions take one step, but you should consider their efficiency to be the same as if you had written the equivalent function using directly on a list, i.e., you should consider the built-in **`length`** function to be linear in the length of the list.

You can assume that all other currently allowed built-in functions (other than **`length`**) are constant time, i.e., *O*(1). Future lectures will have other examples.

Lecture 8 Summary

L08: You should know

- How categorize the behaviour of functions using “Big-O notation”.
- How to use list abbreviations to write lists.
- How to use the stepper to measure efficiency.

We will see many more examples of measuring efficiency in Lecture L09, and we provide a summary of key points for assignments and exams at the end of that lecture.

L08: Allowed constructs

Newly allowed constructs:

```
list
```

Previously allowed constructs:

```
( ) [ ] + - * / = < > <= >= ;
abs acos add1 and asin atan boolean? check-expect
check-within cond cons cons? cos define e else empty empty?
even? exp expt false first inexact? integer? length list?
log max min not number? odd? or pi quotient rational?
remainder rest second sin sqr sqrt sub1 symbol? symbol=? tan
third true zero?
listof Any anyof Bool Int Nat Num Rat Sym
```

Recursion **must** follow the Rules for Recursion (second version)