Big-O notation

CS135 Lecture 08

L08.00 Math class

Huh? Why are there no slides?

We treat this module as a traditional math class.

Your instructor will write on the board.

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

L08.01 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

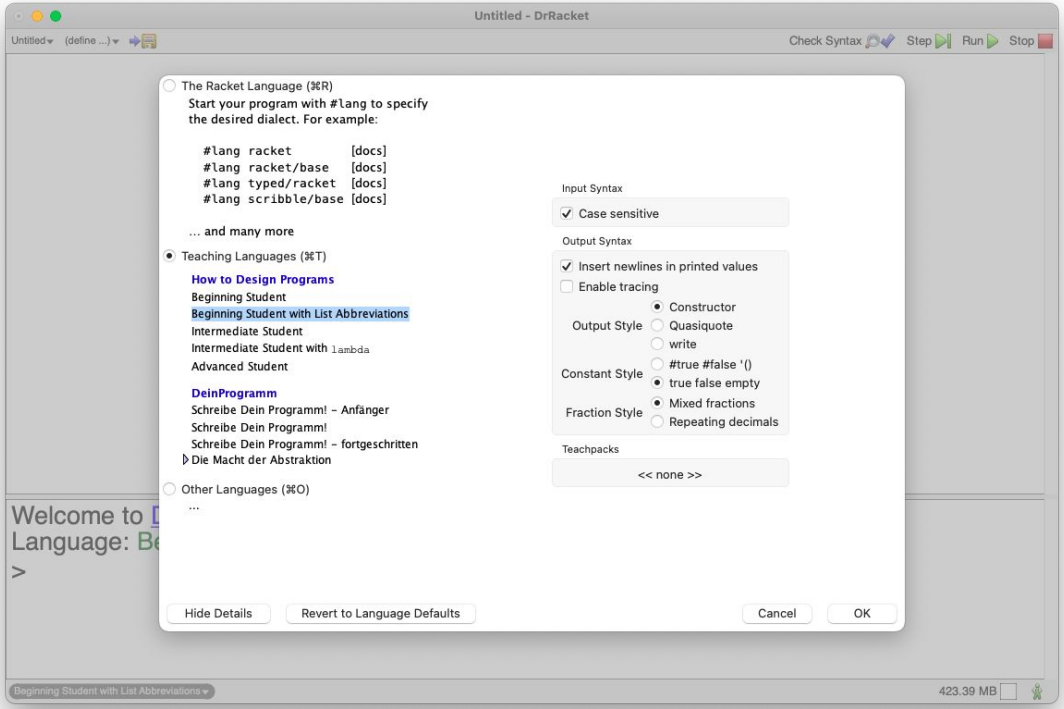
```
(cons 4 (cons 3 (cons 2 (cons 1 empty)
```

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”

L08.02 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 = $f(n) = 6n + 4 = O(n)$

To measure efficiency, we count substitution steps

Stepper

Beginning | Previous Call | Previous | Selected | Next | Next Call | End | 23/28

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

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

23/28

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

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	$O(2^n)$

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.

L08: Allowed constructs

Newly allowed constructs:

```
list
```

Previously allowed constructs:

```
( ) [ ] + - * / = < > <= >= ;
abs acos add1 and asin atan check-expect check-within cond
cons cons? cos define e else empty empty? exp expt false
first inexact? integer? length list? log max min not number?
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 of Recursion (second version)