UMass Boston Computer Science

CS450 High Level Languages (section 2)

Interpreters and “eval”

Monday, November 13, 2023

Logistics

- HW 6 in
  - ~~due: Sun 11/12 11:59 pm EST~~

- HW 7 out
  - due: Sun 11/19 11:59 pm EST
  - Really due: Wed 11/22 11:59 pm EST

  - (no hw over Thanksgiving)

Last Time

Syntax vs Semantics (Spoken Language)

Syntax

- Specifies: **valid language constructs**
    - E.g., **sentence** = (subject) **noun** + **verb** + (object) **noun**

"the ball threw the child"

- Syntactically: **valid!**
- Semantically: **???**

Semantics

- Specifies: **"meaning" of language (constructs)**

Last Time

Syntax vs Semantics (Programming Language)

Syntax

- Specifies: **valid language constructs**
  - E.g., **valid Racket program: s-expressions**
  - **Valid python program: follows python grammar** (including whitespace!)

Semantics

- Specifies: **"meaning" of language (constructs)**

Last Time

Syntax vs Semantics (Programming Language)

Syntax

- Specifies: **valid language constructs**
  - E.g., **valid Racket program: s-expressions**
  - **Valid python program: follows python grammar** (including whitespace!)

Q: What is the "meaning" of a program?

A: The result of "running" it!

... but how does a program "**run**"?

Semantics

- Specifies: **"meaning" of language (constructs)**

From
Lecture 1

Programs run on CPUs

```
0000000 0000 0001 0001 1010 0010 0001 0004 0128
0000010 0000 0016 0000 0028 0000 0010 0000 0020
0000020 0000 0001 0004 0000 0000 0000 0000 0000
0000030 0000 0000 0000 0010 0000 0000 0000 0204
0000040 0004 8384 0084 c7c8 00c8 4748 0048 e8e9
0000050 00e9 6a69 0069 a8a9 00a9 2828 0028 fdfc
0000060 00fc 1819 0019 9898 0098 d9d8 00d8 5857
0000070 0057 7b7a 007a bab9 00b9 3a3c 003c 8888
0000080 8888 8888 8888 8888 288e be88 8888 8888
0000090 3b83 5788 8888 8888 7667 778e 8828 8888
00000a0 d61f 7abd 8818 8888 467c 585f 8814 8188
00000b0 8b06 e8f7 88aa 8388 8b3b 88f3 88bd e988
00000c0 8a18 880c e841 c988 b328 6871 688e 958b
00000d0 a948 5862 5884 7e81 3788 1ab4 5a84 3eec
00000e0 3d86 dcb8 5cbb 8888 8888 8888 8888 8888
00000f0 8888 8888 8888 8888 8888 8888 8888 0000
0000100 0000 0000 0000 0000 0000 0000 0000 0000
*
0000130 0000 0000 0000 0000 0000 0000 0000
000013e
```

Machine code

"low level"

cpu

"result"

Q: What is the "meaning" of a program?

A: The result of "running" it!

... but how does a program "**run**"?

Programmers don't write machine code!

Running Programs: `eval`

```
;; eval : Program -> Result
;; "runs" a given "program", producing a "result"
```

More generally:
An **interpreter**, i.e., an "**eval**" function, turns a "**program**" into a "**result**"

(But programs are usually not directly interpreted either)

More commonly, **a** <u>high-level</u> program is first **compiled** to a <u>lower-level</u> language (and then **intrepreted**)

Q: What is the "meaning" of a program?

A: The result of "running" it!

... but how does a program "**run**"?

From Lecture 1

"high" level
(easier for humans to understand)

NOTE: This hierarchy is *approximate*

English	
Specification langs	Types? pre/post cond?
Markup (html, markdown)	tags
Database (SQL)	queries
Logic Program (Prolog)	relations
Lazy lang (Haskell, R)	Delayed computation
Functional lang (Racket)	Expressions (no stmts)
JavaScript, Python	"eval"
C# / Java	GC (no alloc, ptrs)
C++	Classes, objects
C	Scoped vars, fns
Assembly Language	Named instructions
Machine code	Binary

"declarative"

ive"

More commonly, **a** high-level program is first **compiled** to a lower-level language (and then **intrepreted**)

(runs on cpu)

“high” level
(easier for humans
to understand)

surface language

“decl ative”

compiler

target language

ive”

Specification langs
Markup `(html, markdown)`
Database `(SQL)`
Logic Program `(Prolog)`
Lazy lang `(Haskell, R)`
Functional lang `(Racket)`
JavaScript, Python
C# / Java
C++
C
Assembly Language
Machine code

More commonly**, a** <u>high-level</u> program is first **compiled** to a <u>lower-level</u> language (and then **intrepreted**)

(runs on cpu)

Common **target** languages:

- bytecode (e.g., JS, Java)
- assembly
- machine code

A **virtual machine** is just a **bytecode interpreter**

(A (hardware) **CPU** is just a **machine code interpreter)!**

surface language

input

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

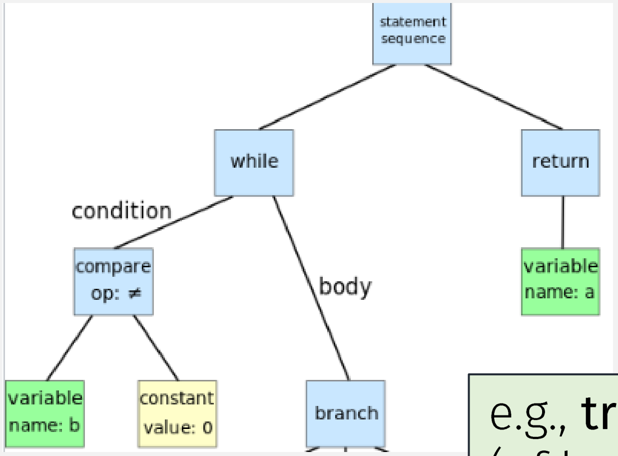
e.g., **string of chars**
(no language constructs)

compiler

This itself is a program

target language

output



e.g., **tree**
(of language constructs)

Semantics

- Specifies: **meaning of language <u>constructs</u>**
- So: **to "run" a program, we need to construct the constructs first**

input

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

e.g., **string of chars**
(no language constructs)

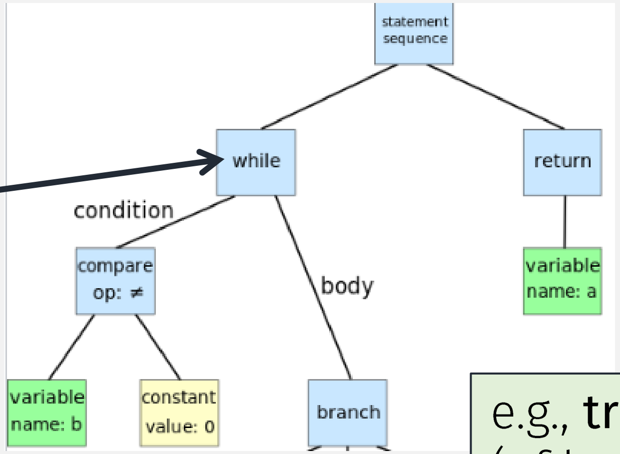
surface language

Compiler, step 1
= **parser**

a **compiler** actually
has <u>many steps</u>
(take a compilers course!)

abstract syntax tree
(AST)

output



e.g., **tree**
(of language constructs)

These must have representations
(**data definitions!**)

input

SExpr

```
(define (count-sl sl sy)
  (cond
    [(empty? sl) 0]
    [else
     (+ (count (first sl) sy) (count-sl (rest
```

surface language

This is a program!

parser

abstract syntax tree
(AST)

output

These must have representations
(**data definitions!**)

These must have representations (**data definitions!**)

input

SExpr

surface language

This is a program!

parser

abstract syntax tree (AST)

output

These must have representations (**data definitions!**)

```
;; A SimpleSexpr (Ssexpr) is one of:
;; - Number
;; - (list ‘+ Ssexpr Ssexpr)
;; - (list ‘- Ssexpr Ssexpr)
```

These must have representations
(**data definitions!**)

input

SExpr

surface language

This is a program!

parser

abstract syntax tree (AST)

output

These must
(**data defin**

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

In-class Coding 11/8: parser

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to a language AST
```

```
;; A SimpleSexpr (Ssexpr) is a:
;; - Number
;; - (list ‘+ Ssexpr Ssexpr)
;; - (list ‘- Ssexpr Ssexpr)
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

In-class Coding 11/8: parser

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to a language AST
```

```
;; A SimpleSexpr (Ssexpr) is a:
;; - Number
;; - (list ‘+ Ssexpr Ssexpr)
;; - (list ‘- Ssexpr Ssexpr)
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

TEMPLATE

```
(define (parse s)
  (match s
    [(? number?) … ]
    [`(+ ,x ,y)
      … (parse x) … (parse y) … ]
    [`(- ,x ,y)
      … (parse x) … (parse y) … ]))
```

In-class Coding 11/8: parser

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to a language AST
```

```
;; A SimpleSexpr (Ssexpr) is a:
;; - Number
;; - (list ‘+ Ssexpr Ssexpr)
;; - (list ‘- Ssexpr Ssexpr)
```

```
(define (parse s)
  (match s
    [(? number?) (num s)]
    [`(+ ,x ,y)
      … (parse x) … (parse y) … ]
    [`(- ,x ,y)
      … (parse x) … (parse y) … ]))
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

In-class Coding 11/8: parser

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to a language AST
```

```
;; A SimpleSexpr (Ssexpr) is a:
;; - Number
;; - (list '+ Ssexpr Ssexpr)
;; - (list '- Ssexpr Ssexpr)
```

```
(define (parse s)
  (match s
    [(? number?) (num s)]
    [`(+ ,x ,y)
     (add (parse x) (parse y))]
    [`(- ,x ,y)
     … (parse x) … (parse y) … ]))
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

In-class Coding 11/8: parser

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to a language AST
```

```
;; A SimpleSexpr (Ssexpr) is a:
;; - Number
;; - (list ‘+ Ssexpr Ssexpr)
;; - (list ‘- Ssexpr Ssexpr)
```

```
(define (parse s)
  (match s
    [(? number?) (num s)]
    [`(+ ,x ,y)
     (add (parse x) (parse y))]
    [`(- ,x ,y)
     (sub (parse x) (parse y))]))
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

In-class Coding 11/8 #2: run

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is one of:
;;
;; ???
;;
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
```

TEMPLATE

```
(define (run p)
  (cond
    [(num? p) … (num-val p) … ]
    [(add? p)  … (run (add-lft p)) …
               … (run (add-rgt p))  … ]
    [(sub? p)  … (run (sub-lft p)) …
               … (run (sub-rgt p))  … ]))
```

In-class Coding 11/8 #2: run

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
```

```
(define (run p)
  (cond
    [(num? p) p]
    [(add? p)  … (run (add-lft p)) …
               … (run (add-rgt p))  … ]
    [(sub? p)  … (run (sub-lft p)) …
               … (run (sub-rgt p))  … ])
```

In-class Coding 11/8 #2: run

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
```

```
;; A Result is a:
;; - Number
```

```
(define (run p)
  (cond
    [(num? p) p]
    [(add? p) (+ (run (add-lft p))
                 (run (add-rgt p)))]
    [(sub? p)  … (run (sub-lft p)) …
               … (run (sub-rgt p))  … ])
```

In-class Coding 11/8 #2: run

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
```

```
;; A Result is a:
;; - Number
```

```
(define (run p)
  (cond
    [(num? p) p]
    [(add? p) (+ (run (add-lft p))
                 (run (add-rgt p)))]
    [(sub? p)  … (run (sub-lft p)) …
               … (run (sub-rgt p))  … ])
```

In-class Coding 11/8 #2: run

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
```

```
;; A Result is a:
;; - Number
```

```
(define (run p)
  (cond
    [(num? p) p]
    [(add? p) (+ (run (add-lft p))
                 (run (add-rgt p)))]
    [(sub? p) (- (run (sub-lft p))
                 (run (sub-rgt p)))]))
```

The "CS450 LANG" Programming Language

cs450-lang.rkt

```
;; parse : Ssexpr -> AST
(define (parse ssexpr) … )
;; run: AST -> Result
(define (run ast) … )

(define (eval450 p) (compose run parse))

(define-macro (module expr ...)
  (eval450 expr) ...))
```

```
((compose f g) x) = (f (g x))
```

cs450-prog1.rkt

```
#lang s-exp "cs450-lang.rkt"

(+ 1 2) ; => 3
```

A program! written in "CS450 LANG"!

“CS450 Lang” Demo

- See `cs450f23/inclass-lecture19` github repository

The "CS450 Lang + Strings" PL

```
;; A Ssexpr is a:
;; - Number
;; - (list '+ Ssexpr Ssexpr)
;; - (list '- Ssexpr Ssexpr)
```

```
;; A Ssexpr is a:
;; - Number
;; - String
;; - (list '+ Ssexpr Ssexpr)
;; - (list '- Ssexpr Ssexpr)
```

```
;; An AST is one of:
;; - (num Number)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
```

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

Parsing "CS450 LANG + STRINGS" Programs

```
;; parse: SimpleSexpr -> AST
;; Converts a (simple) S-expression to language AST
```

```
(define (parse s)
  (match s
   [(? number?) (num s)]
   [(? string?) (str s)]
   [`(+ ,x ,y) (add (parse x) (parse y))]
   [`(- ,x ,y) (sub (parse x) (parse y))]
```

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)
(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

```
;; A Ssexpr is a:
;; - Number
;; - String
;; - (list '+ Ssexpr Ssexpr)
;; - (list '- Ssexpr Ssexpr)
```

Running “CS450 Lang + Strings” Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
;; - String
```

Let’s look at other languages!

```
;; An AST is one of:
;; - (num Number)
;; - (str String)
;; - (add AST AST)
;; - (sub AST AST)
```

What should happen when two strings are added???

e.g., What is the “meaning” of (+ “hello” “world!”)

```
(define (run p)
  (cond
    [(num? n) n]
    [(str? s) s]
    [(add? p) (???? (run (add-lft p)) (run (add-rgt p)))]
    [(sub? p) (???? (run (sub-lft p)) (run (sub-rgt p)))]))
```

JavaScript Semantics Exploration: “plus”

Introducing: The "CS450JS" Programming Lang!

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
```

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

Parsing: “CS450js” Programs

```
;; parse450js: 450jsExpr -> 450jsAST
;; Converts a CS450js Lang surface program to its AST
```

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - (list ‘+ 450jsExpr 450jsExpr)
;; - (list ‘- 450jsExpr 450jsExpr)
```

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
```

```
(define (parse450js s)
  (match s
   [(? number?) (num s)]
   [(? string?) (str s)]
   [`(+ ,x ,y)  (add (parse450js x) (parse450js y))]
   [`(- ,x ,y)  (sub (parse450js x) (parse450js y))]))
```

Running: “CS450JS” Programs

```
;; run450js: 450jsAST -> 450jsResult
;; computes the result of running a CS450js program AST
```

```
;; A 450jsResult is either:
;; - Number
;; - String
```

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
```

```
(define (run p)
  (cond
    [(num? n) n]
    [(str? s) s]
    [(add? p) (???? (run (add-lft p)) (run (add-rgt p)))]
    [(sub? p) (???? (run (sub-lft p)) (run (sub-rgt p)))]))
```

Running: "CS450JS" Programs

```
;; run450js: 450jsAST -> 450jsResult
;; computes the result of running a CS450js program AST
```

```
;; A 450jsResult is either:
;;  - Number
;;  - String
```

```
;; A 450jsAST is one of:
;;  - (num Number)
;;  - (str String)
;;  - (add 450jsAST 450jsAST)
;;  - (sub 450jsAST 450jsAST)
```

```
(define (run p)
  (cond
    [(num? n) n]
    [(str? s) s]
    [(add? p) (450js+ (run (add-lft p)) (run (add-rgt p)))]
    [(sub? p) (450js- (run (sub-lft p)) (run (sub-rgt p)))]))
```

Running: “CS450js” Programs: “plus”

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together
;; (follows js semantics)
```

```
;; A 450jsResult is either:
;; - Number
;; - String
```

TEMPLATE

```
(define (450js+ x y)
  (cond
    [(number? x) … ]
    [(string? x) … ]))
```

or

```
(define (450js+ x y)
  (cond
    [(number? y) … ]
    [(string? y) … ]))
```

Two-Argument Templates

- Sometimes … a fn must **process <u>two</u> arguments simultaneously**
- **This template should combine templates of <u>both</u> args**
  - (This is only possible if the data defs are simple enough)

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; "adds" two cs450js result values together (following js semantics)
```

(2-argument) TEMPLATE

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) … ]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) … ]
```

(see why this is typically not recommended?)

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) … ]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) … ]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) … ]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; "adds" two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) ??? ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) (string-append (??? x) y)]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) (string-append (num->str x) y)]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) (string-append x y)]
```

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; “adds” two cs450js result values together (following js semantics)
```

```
(define (450js+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (string? y)) (string-append (num->str x) y)]
    [(and (string? x) (number? y)) (string-append x (num->str y))]
    [(and (string? x) (string? y)) (string-append x y)]
```

(can any cond clauses be combined?)

```
;; 450js+: 450jsResult 450jsResult -> 450jsResult
;; "adds" two cs450js result values together (following is semantics)
```

```
;; res->str: 450jsResult -> String
(define (res->str x)
  (cond
   [(string? x) x]
   [(number? x) (number->string? x)]))
```

```
(define (450js+ x y)
  (cond
    [(or (string? x) (string? y))
     (string-append (res->str x) (res->str x))]
    [else (+ x y)]))
```

Running: "CS450JS" Programs: "minus"

```
;; 450js-: 450jsResult 450jsResult -> 450jsResult
;; "subtracts" 2nd cs450js result from 1st one (following js semantics)
```

```
(define (450js- x y)
  (cond
    [(and (number? x) (number? y)) (- x y)]
    [(and (number? x) (string? y)) … ]
    [(and (string? x) (number? y)) … ]
    [(and (string? x) (string? y)) … ]))
```

JavaScript Semantics Exploration: “minus”

“Not a Number”

NaN

From Wikipedia, the free encyclopedia

In computing, **NaN** (/næn/), standing for **Not a Number**, is a particular value of a numeric data type (often a floating-point number) which is undefined or unrepresentable, such as the result of 0/0. Systematic use of NaNs was introduced by the IEEE 754 floating-point standard in 1985, along with the representation of other non-finite quantities such as infinities.

mdn web docs_

NaN

The `NaN` global property is a value representing Not-A-Number.

Running: "CS450JS" Programs

```
;; run450js: 450jsAST -> 450jsResult
;; computes the result of running a CS450js program AST
```

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
```

```
;; A 450jsResult is either:
;; - Number
;; - String
;; - NaN
(struct nan [])
(define NaN (nan))
```

Don't forget to update all "Result" functions!

```
;; res->str: 450jsResult -> String
(define (res->str x)
  (cond
   [(string? x) x]
   [(number? x) (number->string? x)]
   [(nan? x) "NaN"]))
```

Running: "CS450JS" Programs: "minus"

```
;; A 450jsResult is either:
;;  - Number
;;  - String
;;  - NaN
(struct nan [])
(define NaN (nan))
```

```
;; 450js-: 450jsResult 450jsResult -> 450jsResult
;; "subtracts" 2nd cs450js result from 1st one
;; (following js semantics)
```

```
(define (450js- x y)
  (cond
    [(and (number? x) (number? y)) (- x y)]
    [else NaN]))
```

- Repo: **cs450f23/lecture19-inclass**
- File: **cs450js-**<*your last name*>**.rkt**

In-class Coding 11/13: **put it all together!**

```
;; parse450js: 450jsExpr -> 450jsAST
;; Parses "CS450js Lang" program to AST
```

```
;; run450js: 450jsAST -> 450jsResult
;; Computes result of running CS450js AST
```

```
;; A 450jsExpr is one of:
;; - Number
;; - String
;; - (list '+ 450jsExpr 450jsExpr)
;; - (list '- 450jsExpr 450jsExpr)
```

```
;; A 450jsResult is one of:
;; - Number
;; - String
;; - NaN
```

```
;; A 450jsAST is one of:
;; - (num Number)
;; - (str String)
;; - (add 450jsAST 450jsAST)
;; - (sub 450jsAST 450jsAST)
(struct num [val])
(struct str [val])
(struct add [lft rgt])
(struct sub [lft rgt])
```

Program in "CS450JS LANG"!

- *Change* require in `cs450js-lang.rkt` to point to your `cs450js-`<*your last name*>`.rkt` file
- *Write* "CS450JS LANG" code by putting this at top of any file: `#lang s-exp "cs450js-lang.rkt"`
  - See cs450js-prog1.rkt as an example

No More Quizzes!

but push your in-class work to:
Repo: `cs450f23/lecture19-inclass`