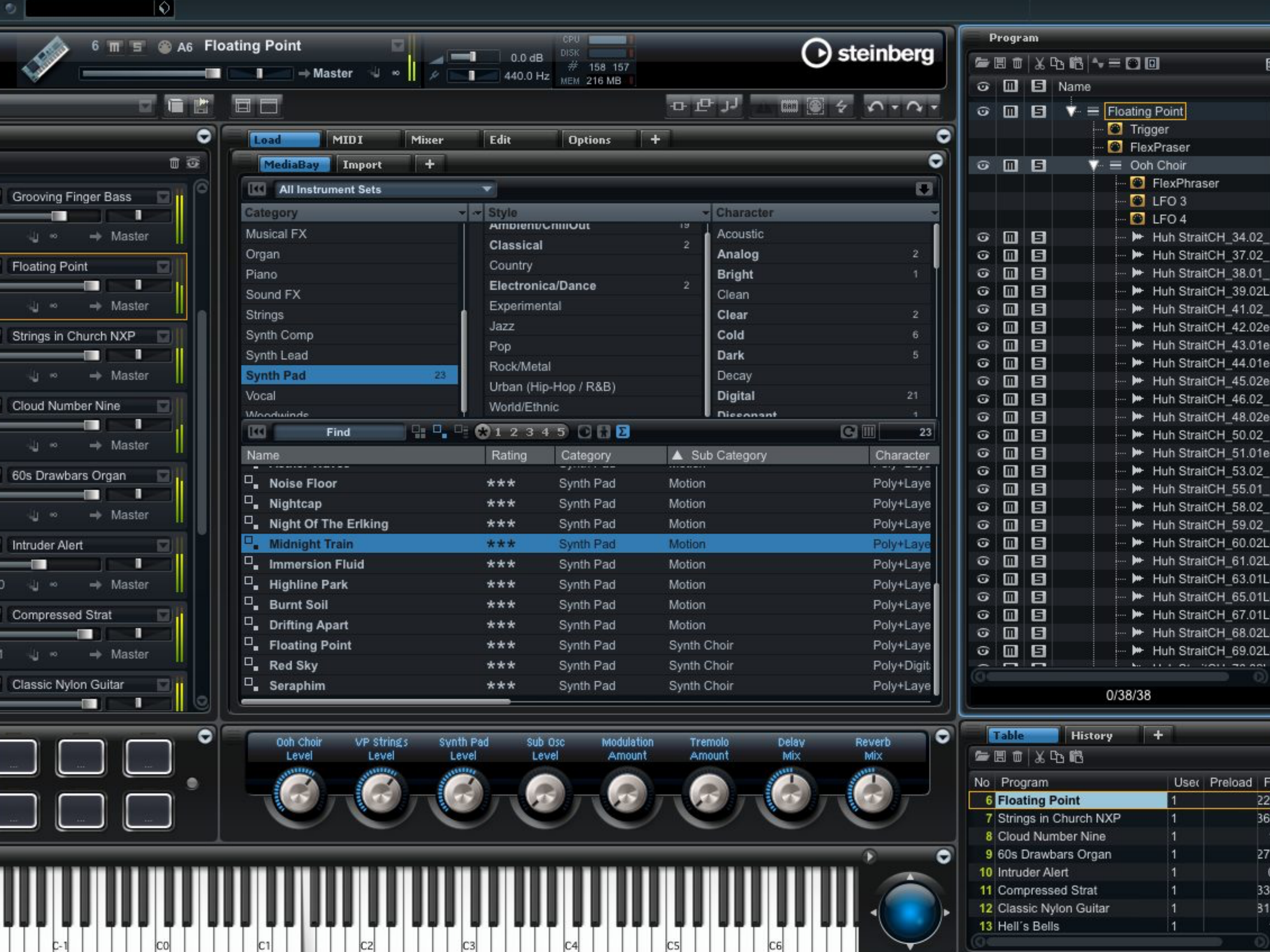
Apollo

manager	Reza Nayebi
architect	Ben Kogan
language	Javier Llaca
integrator	Souren Papazian
validator	Roberto Amorim



Category	
Musical FX	
Organ	
Piano	
Sound FX	
Strings	
Synth Comp	
Synth Lead	
Synth Pad	23
Vocal	
Woodwinds	

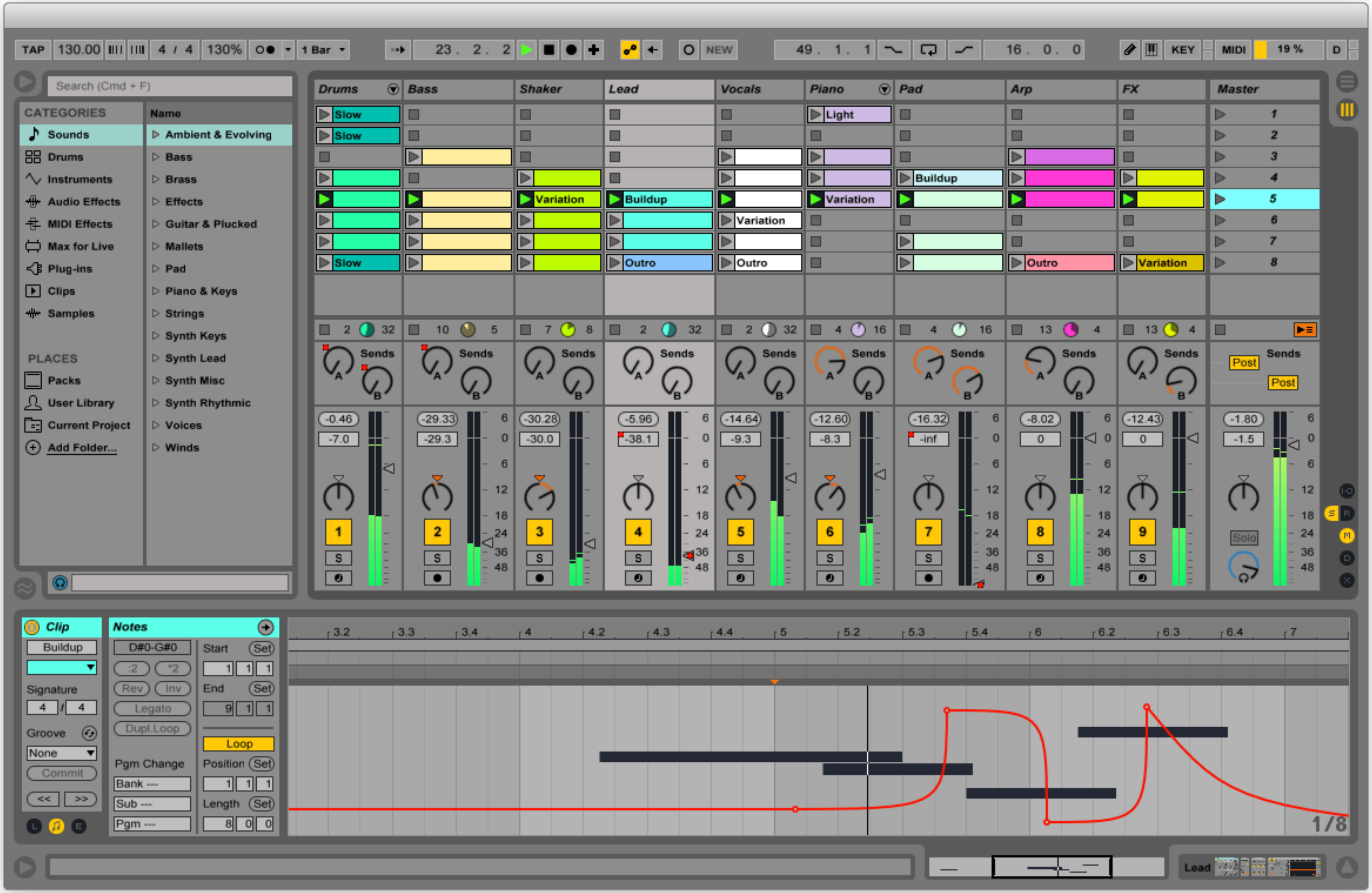
Style	
Ambient/ChillOut	19
Classical	2
Country	
Electronica/Dance	2
Experimental	
Jazz	
Pop	
Rock/Metal	
Urban (Hip-Hop / R&B)	
World/Ethnic	

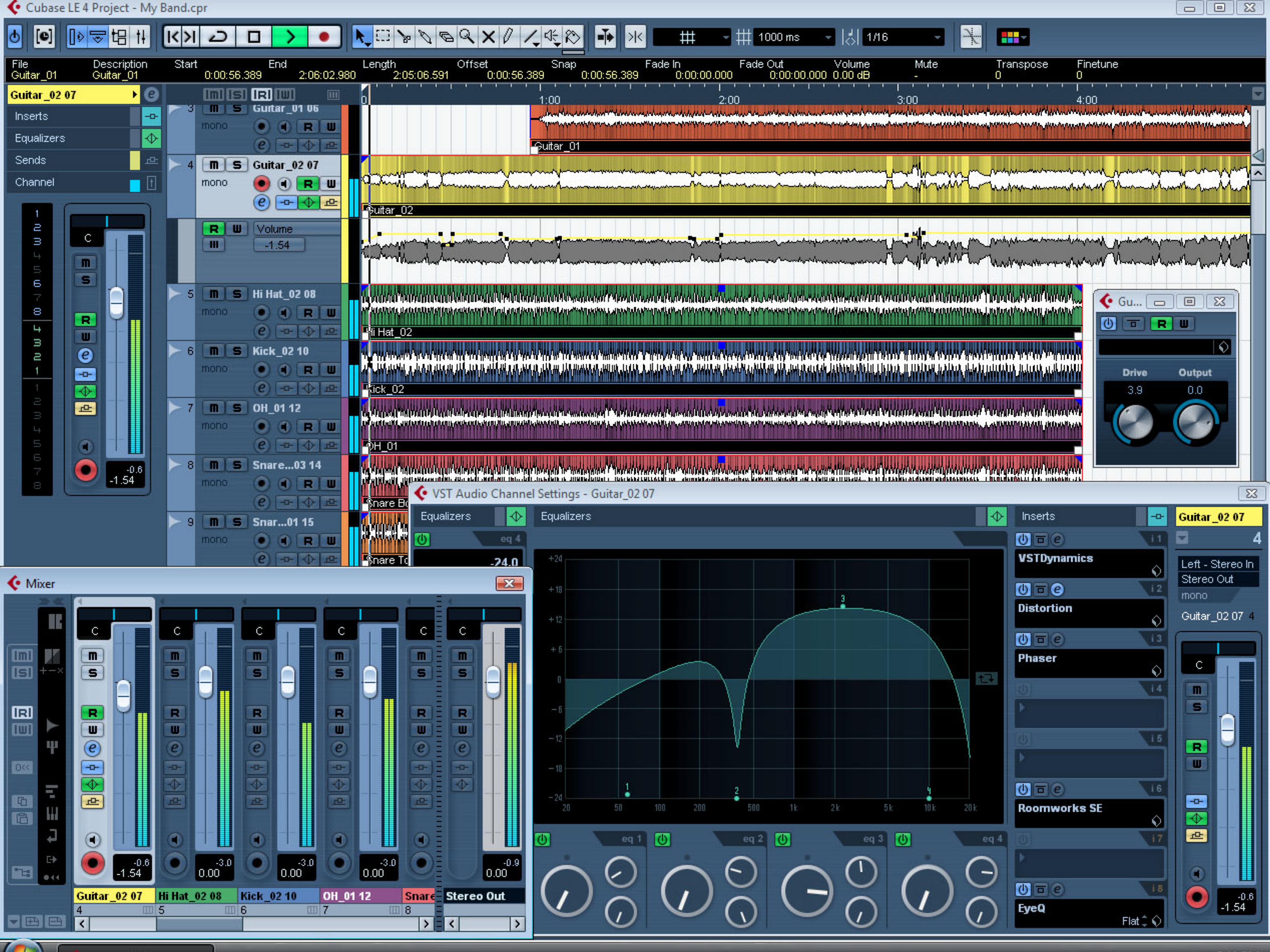
Character	
Acoustic	
Analog	2
Bright	1
Clean	
Clear	2
Cold	6
Dark	5
Decay	
Digital	21
Dissonant	1

Name	Rating	Category	Sub Category	Character
Noise Floor	***	Synth Pad	Motion	Poly+Laye
Nightcap	***	Synth Pad	Motion	Poly+Laye
Night Of The Erlking	***	Synth Pad	Motion	Poly+Laye
Midnight Train	***	Synth Pad	Motion	Poly+Laye
Immersion Fluid	***	Synth Pad	Motion	Poly+Laye
Highline Park	***	Synth Pad	Motion	Poly+Laye
Burnt Soil	***	Synth Pad	Motion	Poly+Laye
Drifting Apart	***	Synth Pad	Motion	Poly+Laye
Floating Point	***	Synth Pad	Synth Choir	Poly+Laye
Red Sky	***	Synth Pad	Synth Choir	Poly+Digit
Seraphim	***	Synth Pad	Synth Choir	Poly+Laye

Floating Point
Trigger
FlexPraser
Ooh Choir
FlexPhraser
LFO 3
LFO 4
Huh StraitCH_34.02
Huh StraitCH_37.02
Huh StraitCH_38.01
Huh StraitCH_39.02L
Huh StraitCH_41.02
Huh StraitCH_42.02e
Huh StraitCH_43.01e
Huh StraitCH_44.01e
Huh StraitCH_45.02e
Huh StraitCH_46.02
Huh StraitCH_48.02e
Huh StraitCH_50.02
Huh StraitCH_51.01e
Huh StraitCH_53.02
Huh StraitCH_55.01
Huh StraitCH_58.02
Huh StraitCH_59.02
Huh StraitCH_60.02L
Huh StraitCH_61.02L
Huh StraitCH_63.01L
Huh StraitCH_65.01L
Huh StraitCH_67.01L
Huh StraitCH_68.02L
Huh StraitCH_69.02L

No	Program	Usec	Preload	F
6	Floating Point	1	22	
7	Strings in Church NXP	1	36	
8	Cloud Number Nine	1		
9	60s Drawbars Organ	1	27	
10	Intruder Alert	1	(	
11	Compressed Strat	1	33	
12	Classic Nylon Guitar	1	31	
13	Hell´s Bells	1		


TAP
130.00
4 / 4
130%
1 Bar
23 . 2 . 2
NEW
49 . 1 . 1
16 . 0 . 0
KEY
MIDI
19 %
D
Search (Cmd + F)
CATEGORIES
Sounds
Drums
Instruments
Audio Effects
MIDI Effects
Max for Live
Plug-ins
Clips
Samples
PLACES
Packs
User Library
Current Project
Add Folder...
Name
Ambient & Evolving
Bass
Brass
Effects
Guitar & Plucked
Mallets
Pad
Piano & Keys
Strings
Synth Keys
Synth Lead
Synth Misc
Synth Rhythmic
Voices
Winds
Drums
Bass
Shaker
Lead
Vocals
Piano
Pad
Arp
FX
Master
Slow
Slow
Slow
Light
Buildup
Variation
Buildup
Variation
Variation
Outro
Outro
Outro
Variation
1
2
3
4
5
6
7
8
Sends
Post
Post
Solo
-0.46
-7.0
-29.33
-29.3
-30.28
-30.0
-5.96
-38.1
-14.64
-9.3
-12.60
-8.3
-16.32
-inf
-8.02
0
-12.43
0
-1.80
-1.5
1
2
3
4
5
6
7
8
9
S
Clip
Buildup
Signature
4 / 4
Groove
None
Commit
Notes
D#0-G#0
:2
*2
Rev
Inv
Legato
Dupl.Loop
Start
Set
End
Loop
Position
Length
Pgm Change
Bank ---
Sub ---
Pgm ---
3.2
3.3
3.4
4
4.2
4.3
4.4
5
5.2
5.3
5.4
6
6.2
6.3
6.4
7
1/8
Lead



Cubase LE 4 Project - My Band.cpr

File	Description	Start	End	Length	Offset	Snap	Fade In	Fade Out	Volume	Mute	Transpose	Finetune
Guitar_01	Guitar_01	0:00:56.389	2:06:02.980	2:05:06.591	0:00:56.389	0:00:56.389	0:00:00.000	0:00:00.000	0.00 dB	-	0	0

VST Audio Channel Settings - Guitar_02 07

But you might know this...

$$\lambda \, x \, . \, (+1) \, x$$

...and maybe a bit of this...

What happens if…

...we do this?

λ x . (♫) x

What is Apollo?

- Functional language for music creation

- Simple to use and understand

- Intended for musicians and non-musicians

What is Apollo?

- Functional language for music creation
- Simple to use and understand
- Intended for musicians and non-musicians

Let’s look at Apollo in action

Example

```
pitches: [Pitch] = [C4, E4, G4, C5, G4, E4, C4]

rhythm: [Duration] = uniform(\4, 7)

arpeggio: [Atom] = zip(pitches, rhythm)

main: Music = [arpeggio]
```

Data Types

- Int, Pitch, Duration
- Atom
- List
- Music

Data Types

```
note: Atom = (A4, \4)

chord: Atom = ([A4, C#5, E5], \4)

lead: [Atom] = [note, note]

back: [Atom] = [chord, chord]

song: Music = [lead, back]
```

Data Types

```
x: Int = 3

a: Pitch = A4       -- A in Octave 4 (69)

q: Duration = \4    -- Quarter Note (16)

aMajor: [Pitch] = [A4, C#4, E4]
```

Functions

```
square: (n: Int) -> Int = n * n

fac: (n: Int) -> Int =
   case (n == 0)
     1
    otherwise
      n * fac(n - 1)
```

Functions

Higher-order

```
g: (f: (Int) -> Int, x: Int) -> Int = f(f(x))
```

Typed lambda expressions

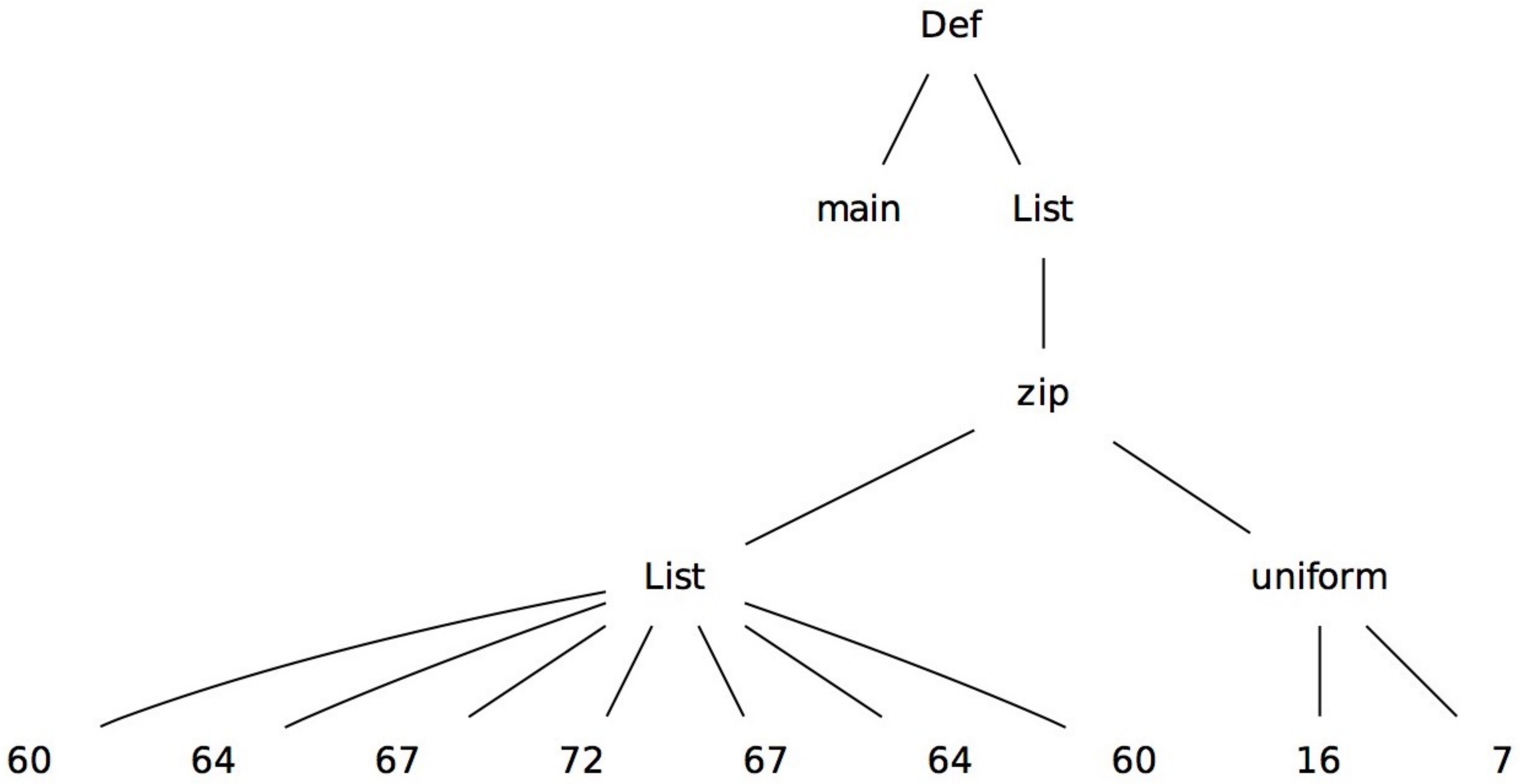
```
\x: Int, y: Int -> Int = x + y
```

Example Revisited

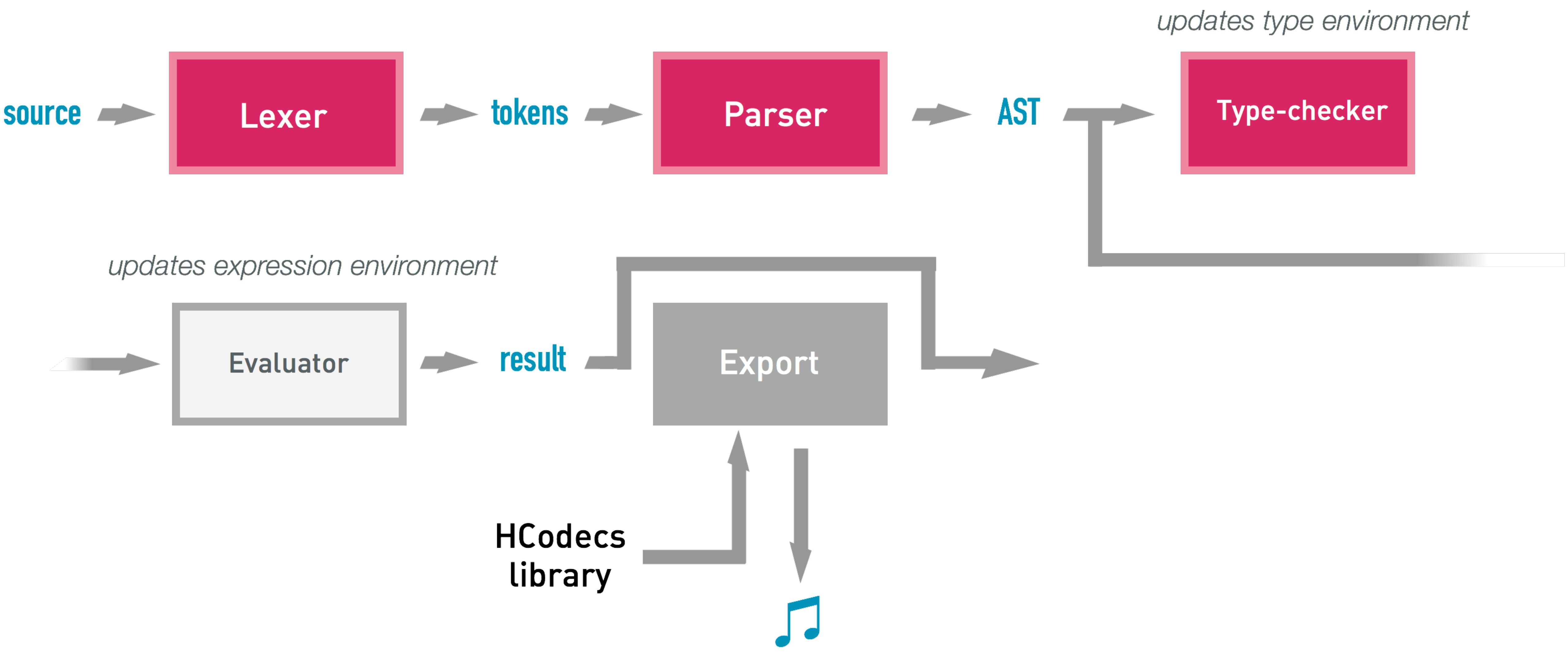
```
pitches: [Pitch] = [C4, E4, G4, C5, G4, E4, C4]

rhythm: [Duration] = uniform(\4, 7)

arpeggio: [Atom] = zip(chord, rhythm)

main: Music = [arpeggio]
```

Example Revisited

```
main: Music = [zip(
  [C4, E4, G4, C5, G4, E4, C4],
  uniform(\4, 7))]
```


Def
main
List
zip
List
uniform
60
64
67
72
67
64
60
16
7

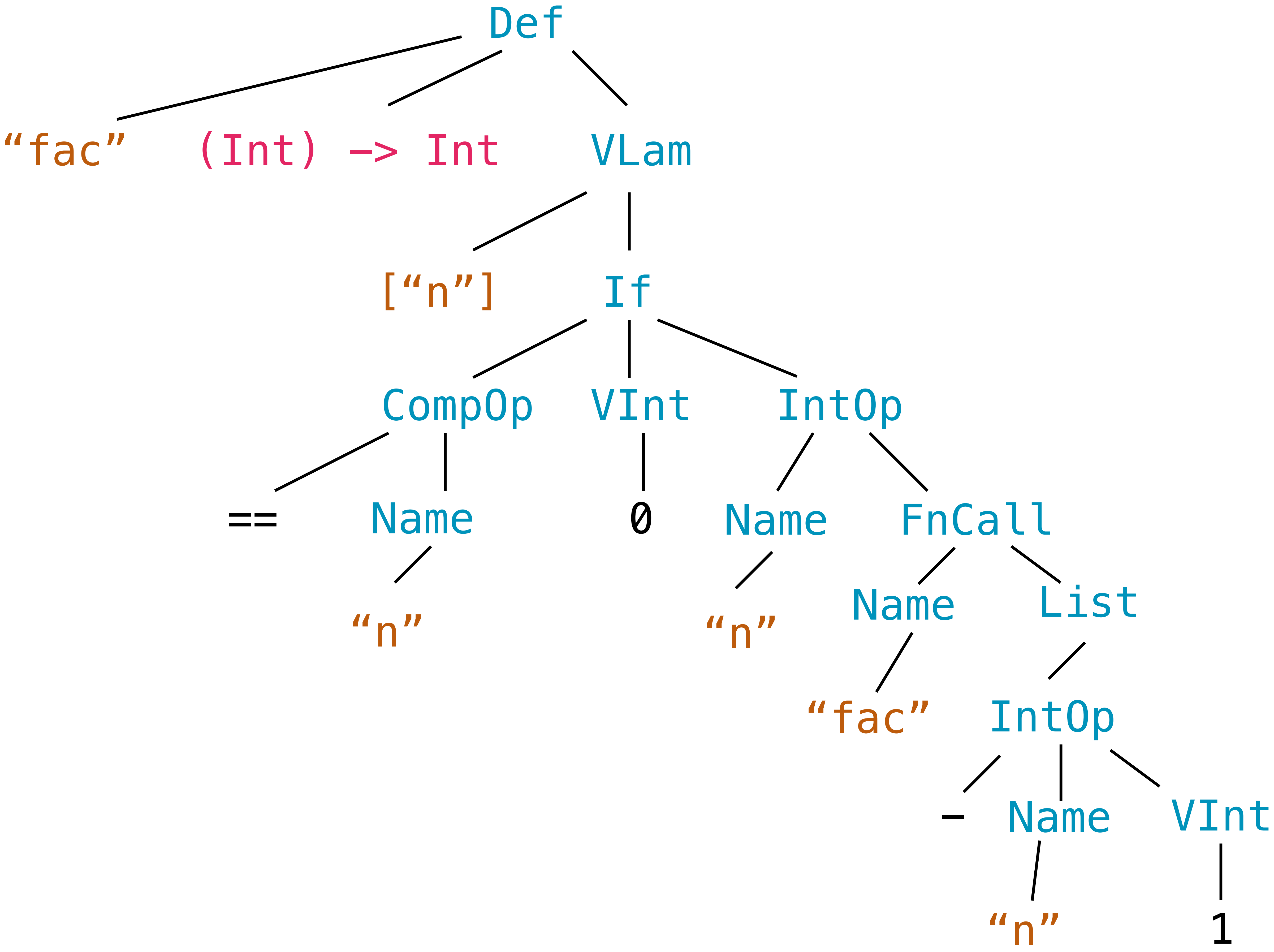
Interpreter Architecture

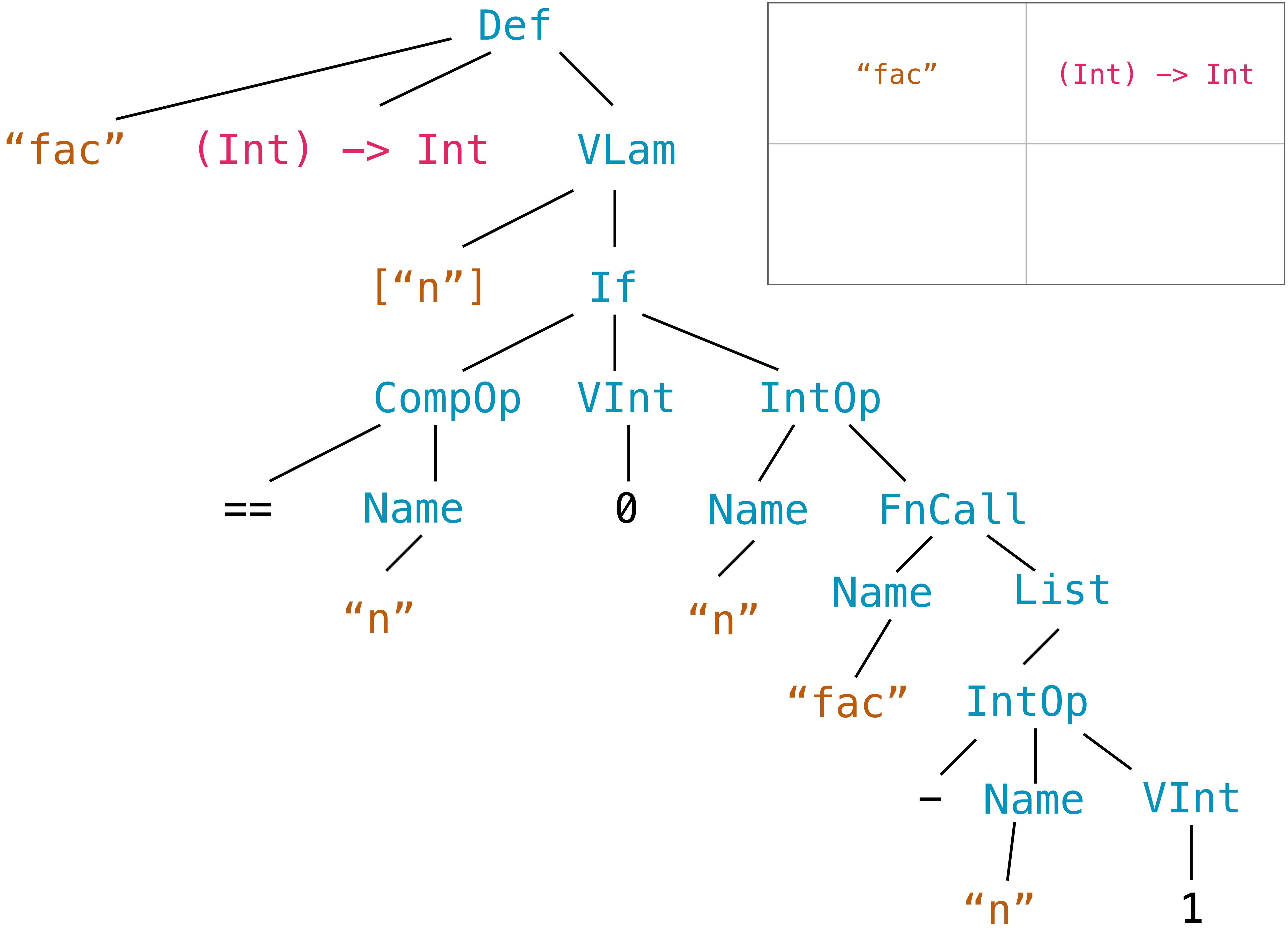


```
toAst t src >>= execAst env >>= \result ->
  handleExport env ofile "main" >> return result
```

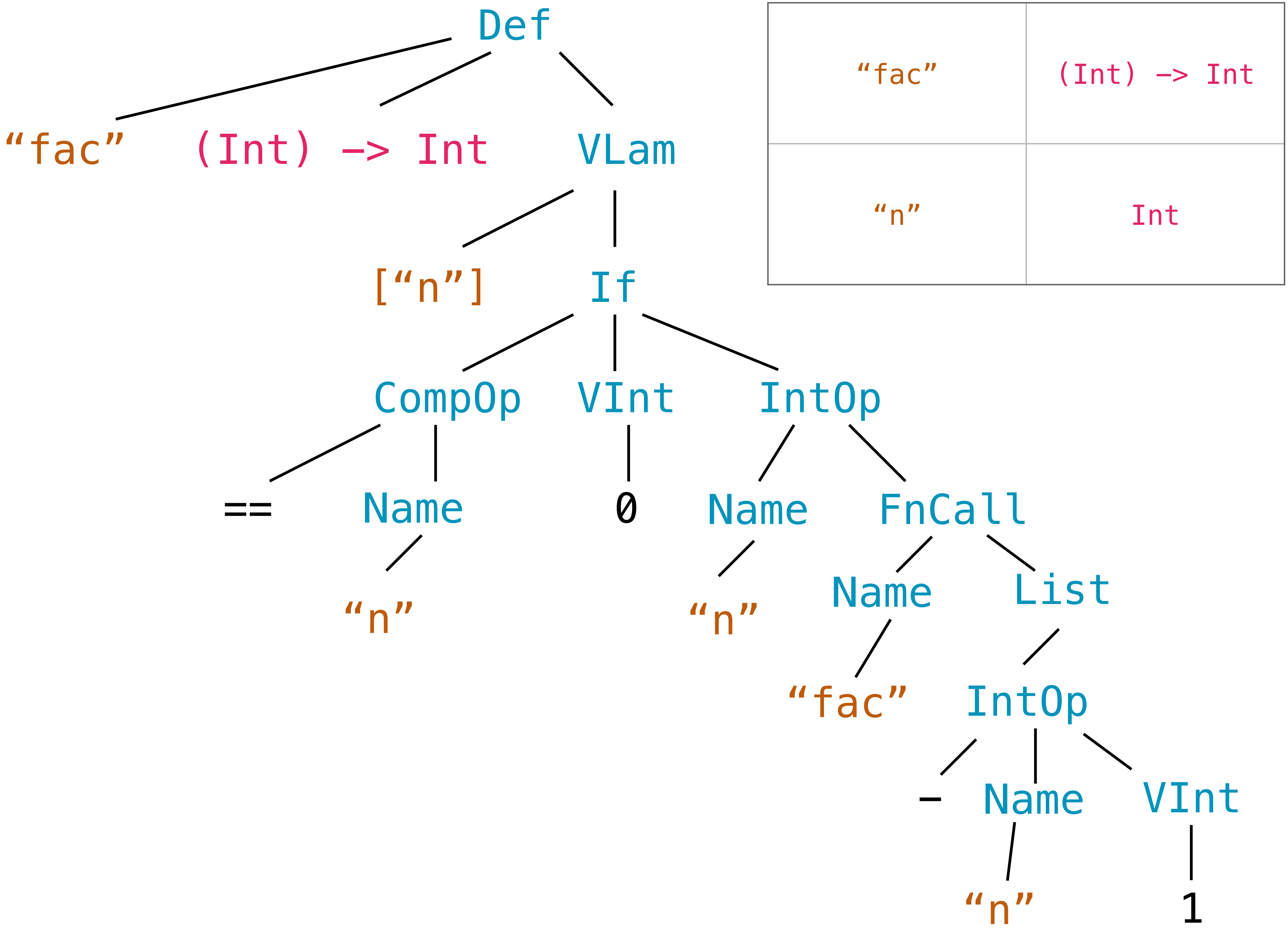
Extendable Architecture

```
typecheck :: Env Type -> Expr -> IOThrowsError Type
typecheck env expr = case expr of

  VInt{}  -> return TInt
  VBool{} -> return TBool

  Neg e -> do
    t <- typecheck env e
    if t == TInt
    then return TInt
    else throwError (TypeUMismatch "-" t)

  Head l -> do
    tl <- typecheck env l
    case tl of
      (TList t) -> return t
      _         -> throwError (TypeUMismatch "h@" tl)

  BoolOp op a b -> do
    ta <- typecheck env a
    tb <- typecheck env b
    case (ta, tb) of
      (TBool, TBool) -> return TBool
      _              -> throwError (TypeMismatch (show op) ta tb)

  Block body ret -> do
    env' <- clone' env
    mapM_ (typecheck env') body
    typecheck env' ret

  FnCall (Name name) args -> do
    TFunc tps tr <- getVar env name
    checkFn env (name, tps, tr) args

  FnCall (VTLam tps tr _ _) args ->
    checkFn env ("<lambda>", tps, tr) args
```

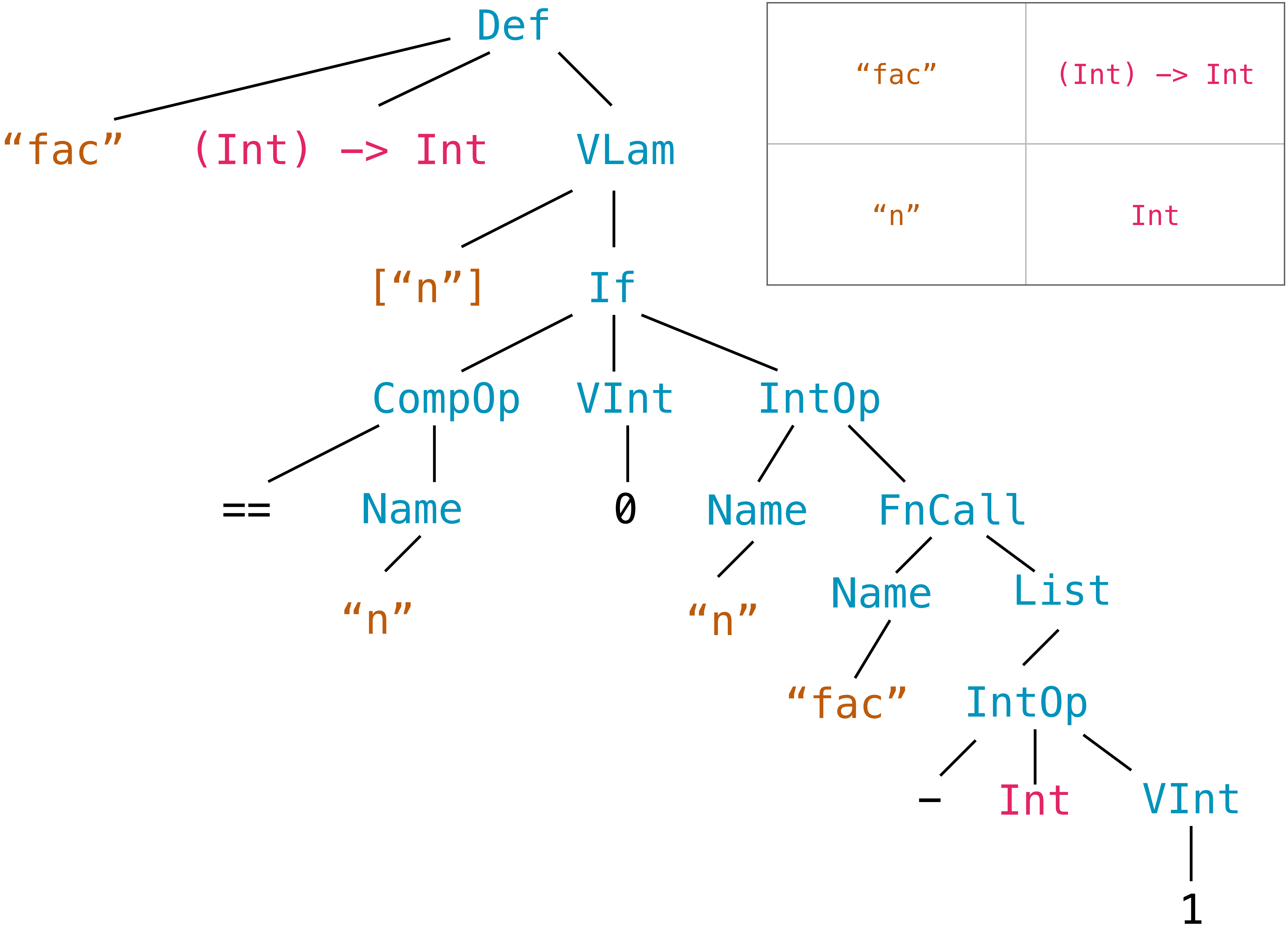

Def
“fac”
(Int) -> Int
VLam
[“n”]
If
CompOp
VInt
IntOp
==
Name
0
Name
FnCall
“n”
“n”
Name
List
“fac”
IntOp
-
Name
VInt
“n”
1


Def
“fac”
(Int) -> Int
VLam
[“n”]
If
CompOp
VInt
IntOp
==
Name
0
Name
FnCall
“n”
“n”
Name
List
“fac”
IntOp
-
Name
VInt
“n”
1

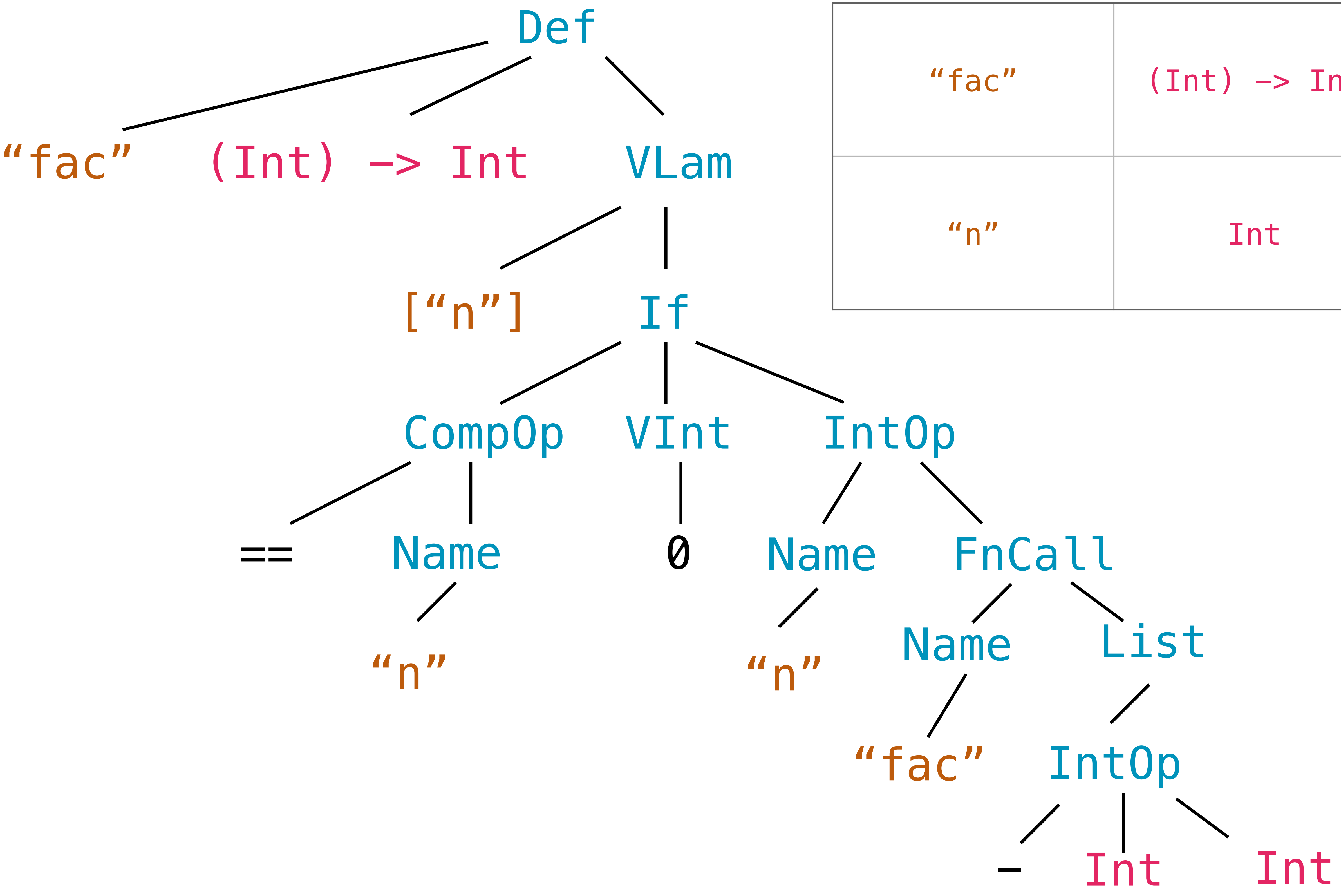
“fac”	(Int) -> Int



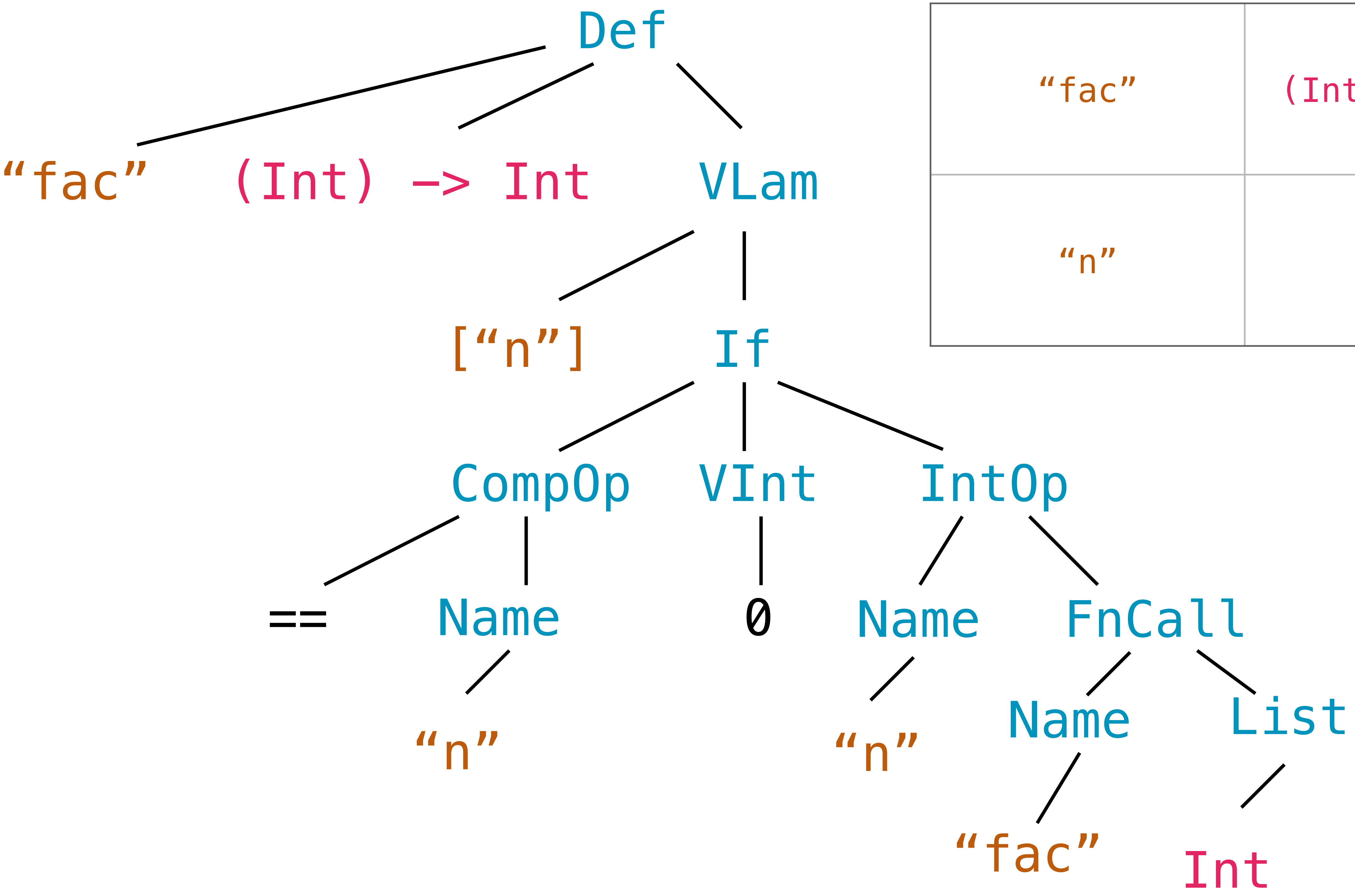
“fac”	(Int) -> Int
“n”	Int



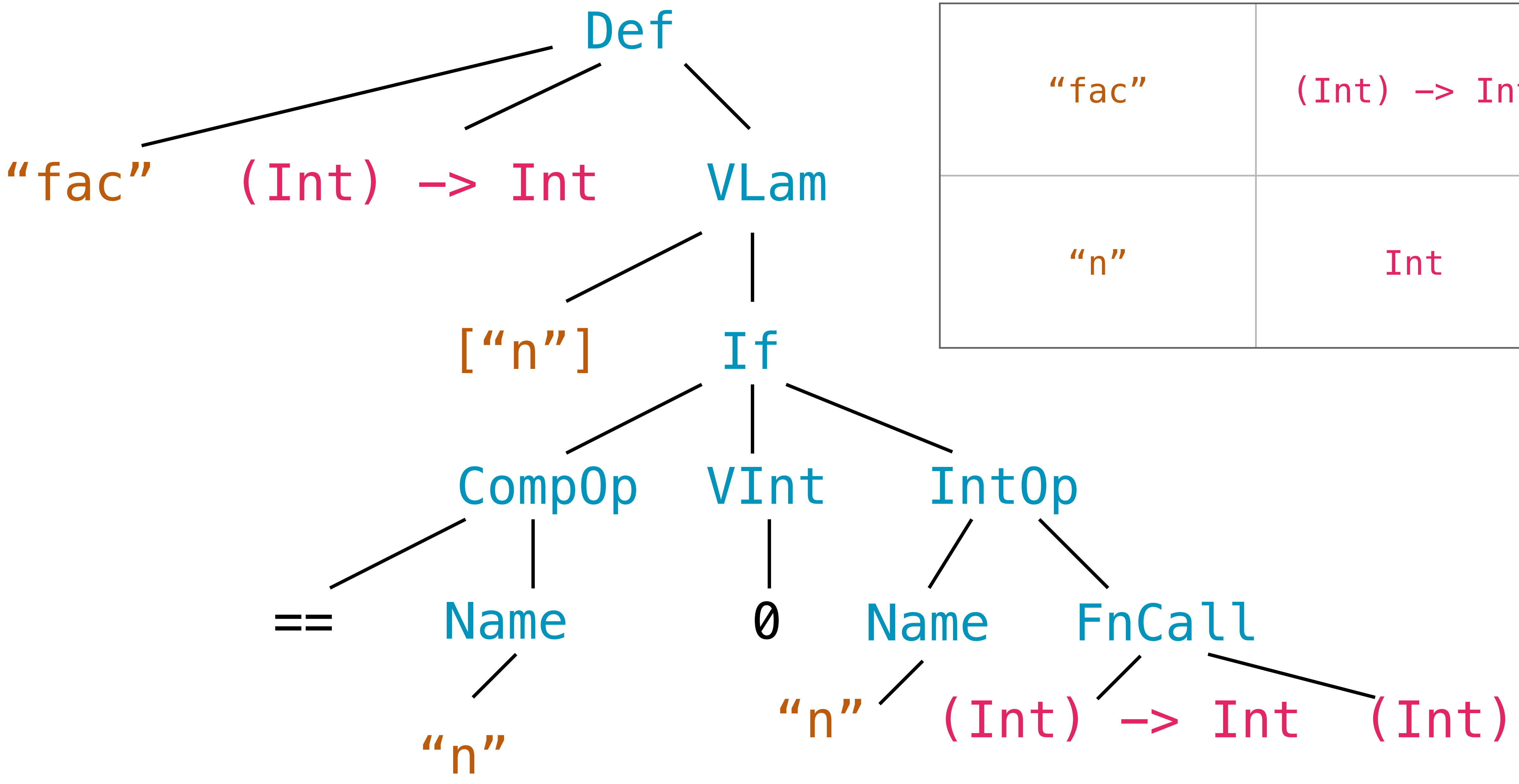
“fac”	(Int) -> Int
“n”	Int



“fac”	(Int) -> Int
“n”	Int



“fac”	(Int) –> Int
“n”	Int



Def

“fac”

(Int) –> Int

VLam

[“n”]

If

CompOp

VInt

IntOp

==

Name

0

Name

FnCall

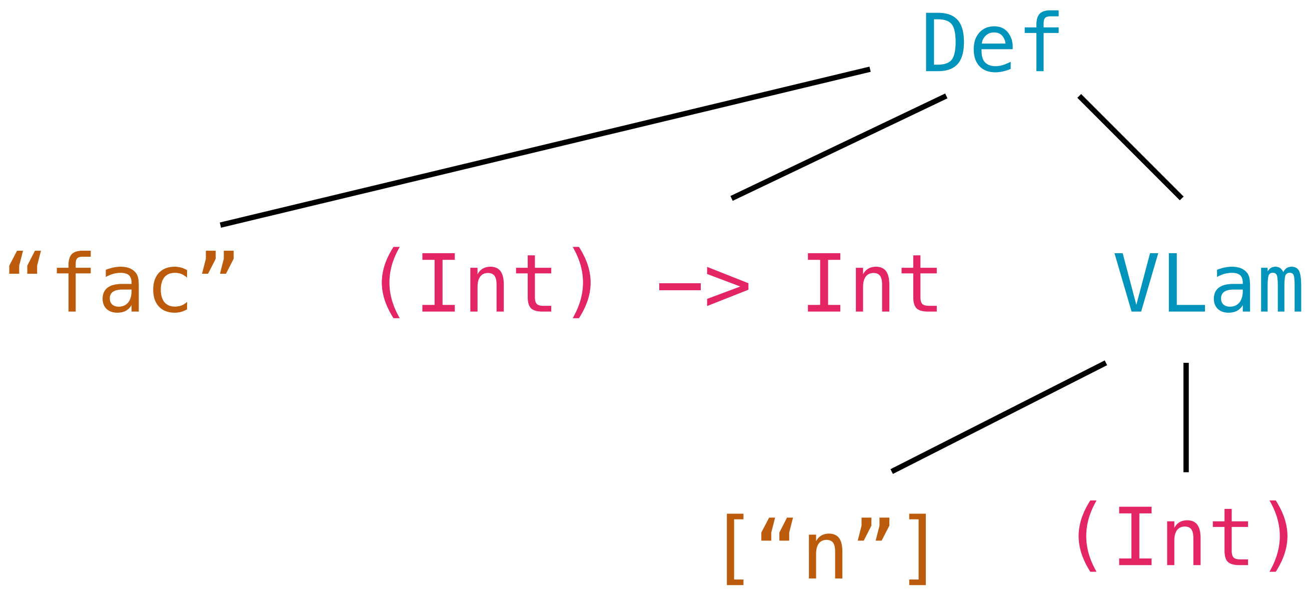
“n”

“n”

(Int) –> Int

(Int)

“fac”	(Int) –> Int
“n”	Int



“fac”	(Int) –> Int
“n”	Int

“fac”   (Int) -> Int   ✔

“fac”	(Int) -> Int
“n”	Int

An interesting case

```
h@[]
```

An interesting case

```
case (!a)
  []
otherwise
  f(h@a) :: mapII(f, t@a)
```

Enabling recursion

- Initialize name before storing its value

- Closures:

```
    Def name _ (VLam p b)
       => (Function p b env')
```

- …and recursive closures!

Software Development Environment

- UNIX
- GitHub
- Haskell
- Cabal

Runtime

- REPL
- Output MIDI
- Prelude

Prelude

```
concat: (a: [Int], b: [Int]) -> [Int] =
   case (!a)
      b
   otherwise
      h@a :: concat(t@a, b)
```

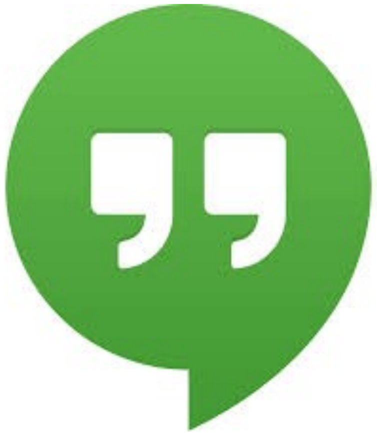
Prelude

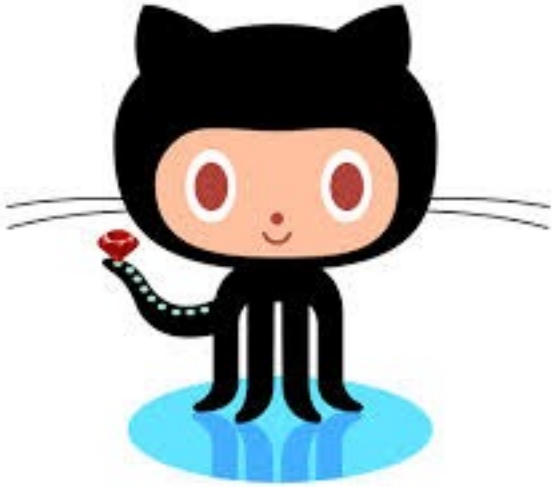
```
filter: (f: (Int) -> Bool, a: [Int]) -> [Int] =
   case (!a)
     []
   case (f(h@a))
     h@a :: filter(f, t@a)
   otherwise
     filter(f, t@a)
```

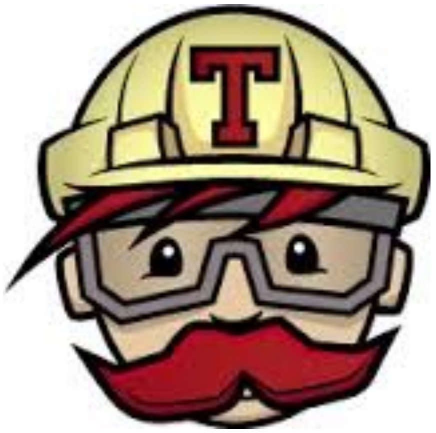
Prelude

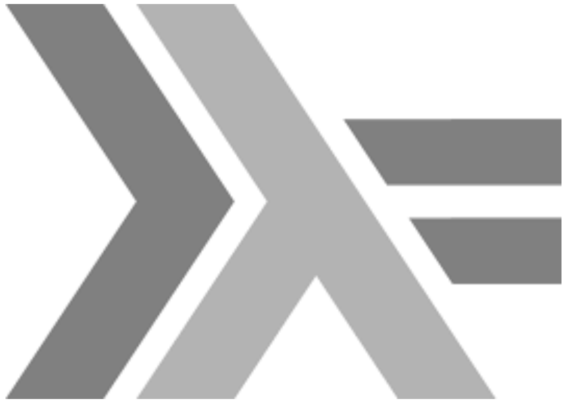
```
sort: (a: [Int]) -> [Int] =
    case (!a)
        []
    otherwise {
        concat(concat(sort(a), [p]), sort(b))
        where
            p: Int = h@a
            a: [Int] = filter(\x: Int -> Bool: x <= p, t@a)
            b: [Int] = filter(\x: Int -> Bool: x > p, t@a)
    }
```

Project Management







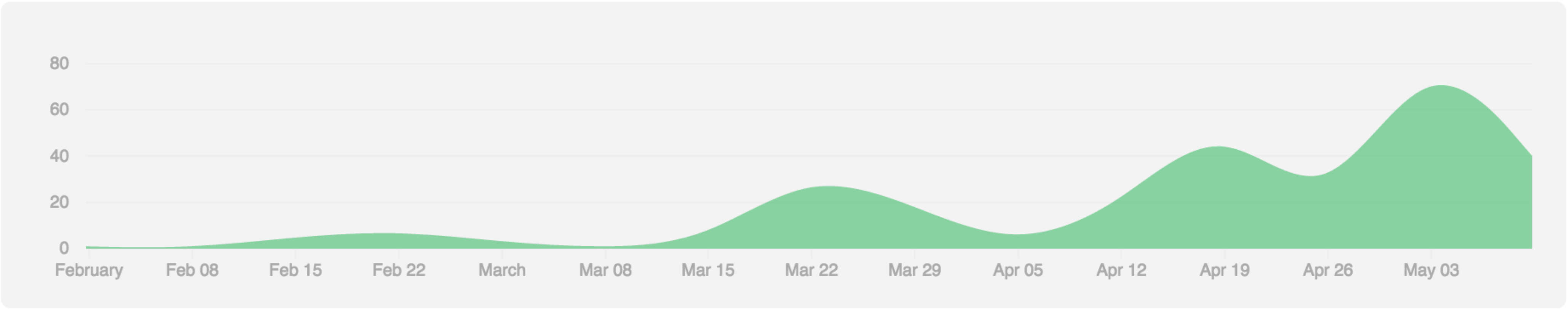


Project Management

- Weekly team meetings
- Git workflow: branch and pull request, wait for validation from other team members before merging
- Travis CI and unit testing helped a lot to catch small errors.

Project Management

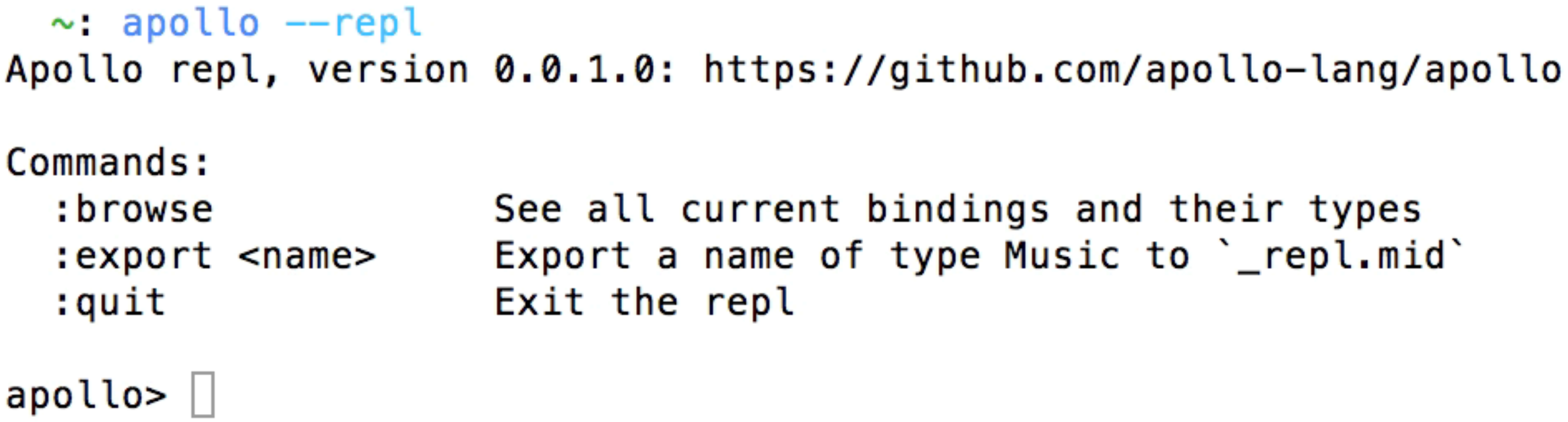
- Project came together nicely at the end
- A lot of the hard work in the architecture that allowed us to add all features easily



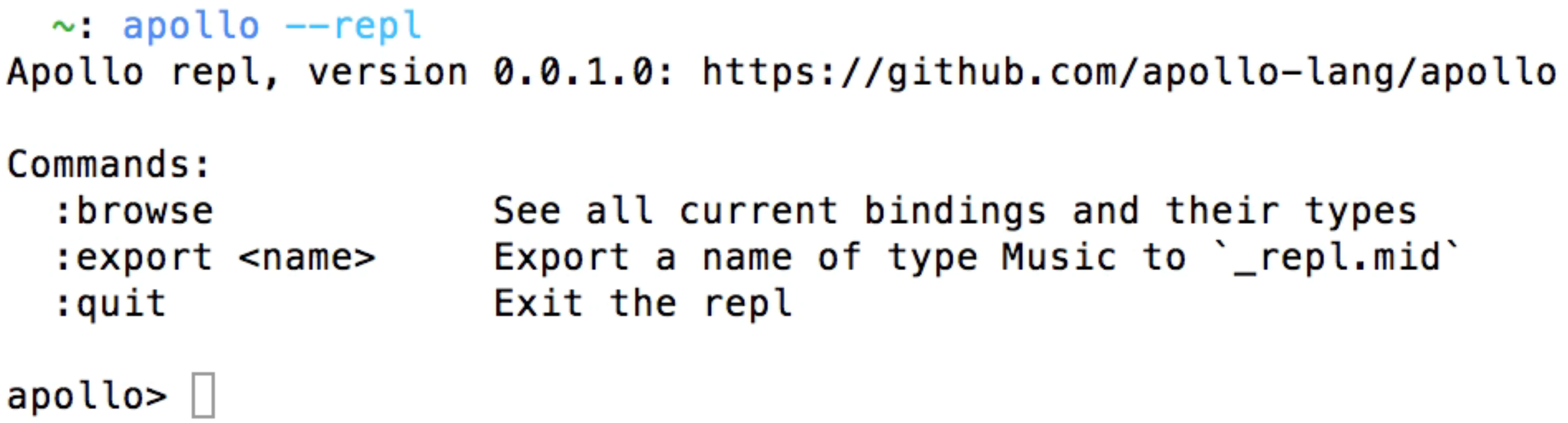
Git commit history

Testing & Validation

```
~:
```



```
  ~: apollo --repl
Apollo repl, version 0.0.1.0: https://github.com/apollo-lang/apollo

Commands:
  :browse              See all current bindings and their types
  :export <name>       Export a name of type Music to `_repl.mid`
  :quit                Exit the repl

apollo> 
```



```
  ~: apollo --repl
Apollo repl, version 0.0.1.0: https://github.com/apollo-lang/apollo

Commands:
  :browse              See all current bindings and their types
  :export <name>       Export a name of type Music to `_repl.mid`
  :quit                Exit the repl

apollo> 
```

Testing and Validation

- 20+ code files for testing features, errors
- Bash script tests, validates, and `diff`s errors
- Test files all at once or line-by-line
- Travis Continuous Integration

```
~/P/apollo master: make test
```