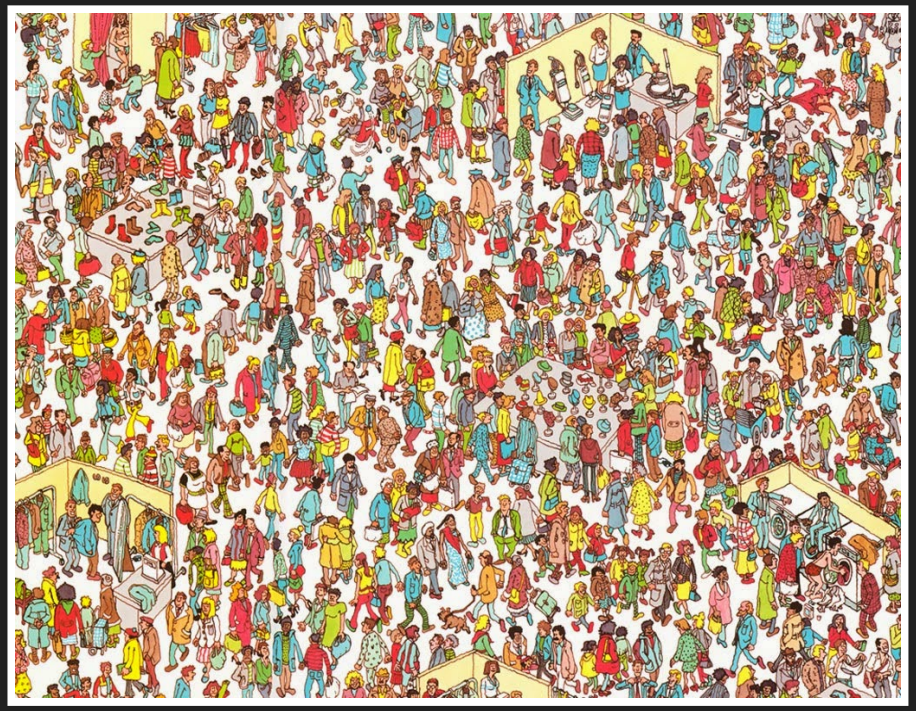
DECOMPILATION, TRANSLATION, AND DEBUGGING

WHERE AM I NOW — REALLY?

(See the references at the end for links to the text of these slides.)



A PYTHON ERROR

```
Traceback (most recent call last):
  File "bug.py", line 2, in <module>
    i / j / k
ZeroDivisionError: integer division or modulo by zero
```

Was `j` zero, or was `k` zero?

MORE ERRORS WHERE LINE NUMBERS ARE NOT GOOD ENOUGH

```
x = prev[prev[0]]                              # which prev ?
```

```
[e[0] for i in d[j] if got[i] == e[i]] # lots going on here
```

```
exec(some_code % 10, namespace)          # code at runtime
```

AN ERROR WHERE NO FILE EXISTS

```
exec(some_code)
```

```
Traceback (most recent call last):
  File "bug-exec.py", line 3, in <module>
    exec(template % 10, namespace)
  File "<string>", line 1, in <module>
NameError: name 'bad' is not defined
```

File "<string>"?

PROGRAMMER TO COMPUTER

A programmer may write:

```
x + y
```

but the C Python interpreter sees:

```
2:   8 LOAD_NAME     x
    10 LOAD_NAME     y
    12 BINARY_ADD
```

SPANISH TO ENGLISH

A person says in Spanish: “mango fragante”
might translate in English to: “fragrant mango”

A person says in Spanish: “Entiendo”
might translate in English to: “I understand”

~~A person says in Spanish: “templado”
might translate in English to: “not hot and not cold”~~

SIDE-BY-SIDE TRANSLATION

1: Whose woods these are I think I know.
2: His house is in the village though

1: A quién pertenece este bosque creo que sé
2: Aunque su casa en el pueblo está

SIDE-BY-SIDE TRANSLATION

1: Whose woods these are I think I know.
2: His house is in the village though

1: A quién pertenece este bosque creo que sé
2: Aunque su casa en el pueblo está

SIDE-BY-SIDE TRANSLATION

1: Whose woods these are I think I know.
2: His house is in the village though

1: A quién pertenece este bosque creo que sé
2: Aunque su casa en el pueblo está

SIDE-BY-SIDE PYTHON TRANSLATION

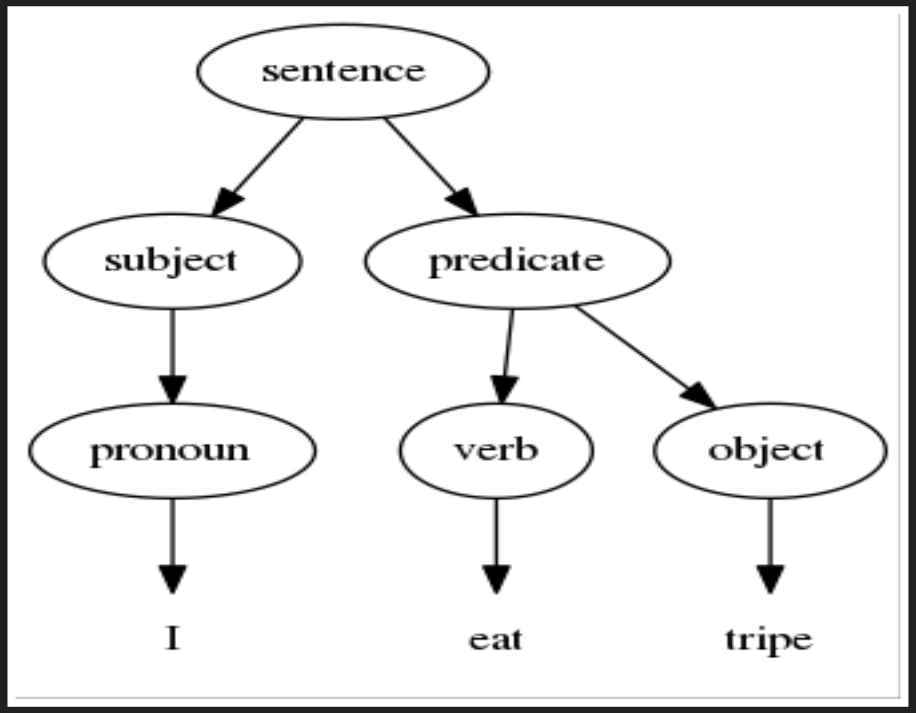
```
x = 1  # line 1
y = 2  # line 2
```

```
1: 0 LOAD_CONST  1
   2 STORE_NAME  x

2: 4 LOAD_CONST  2
   6 STORE_NAME  y
```

ENGLISH SENTENCE PARSE

"I eat tripe."

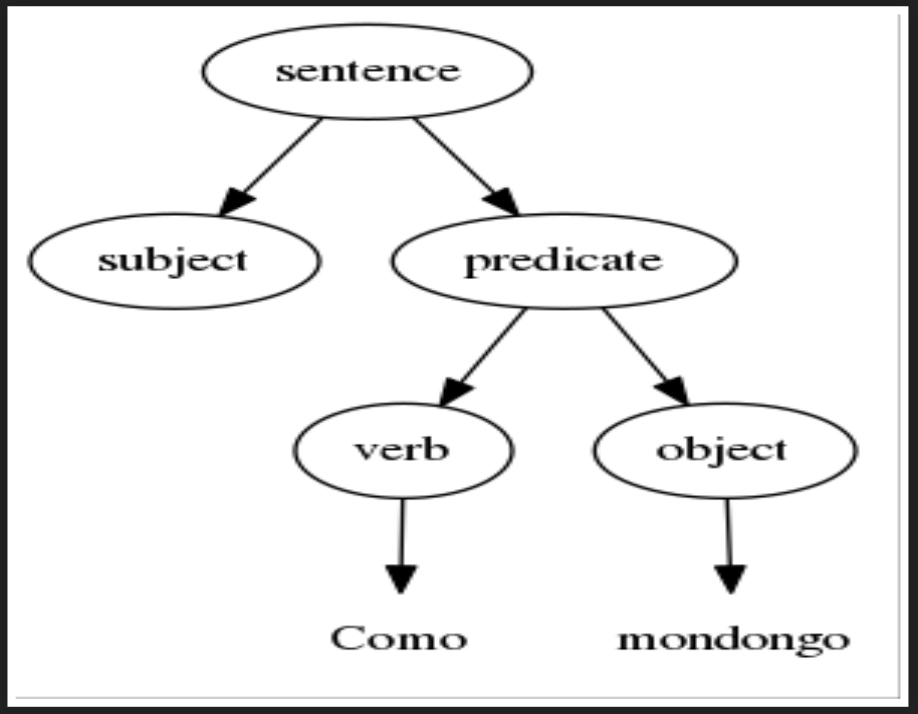


“I eat tripe.”

```
sentence
  0. subject
    pronoun
      I
  1. predicate
     0. verb
        eat
     1. object
        tripe
```

SPANISH SENTENCE PARSE

“Como mondongo.”



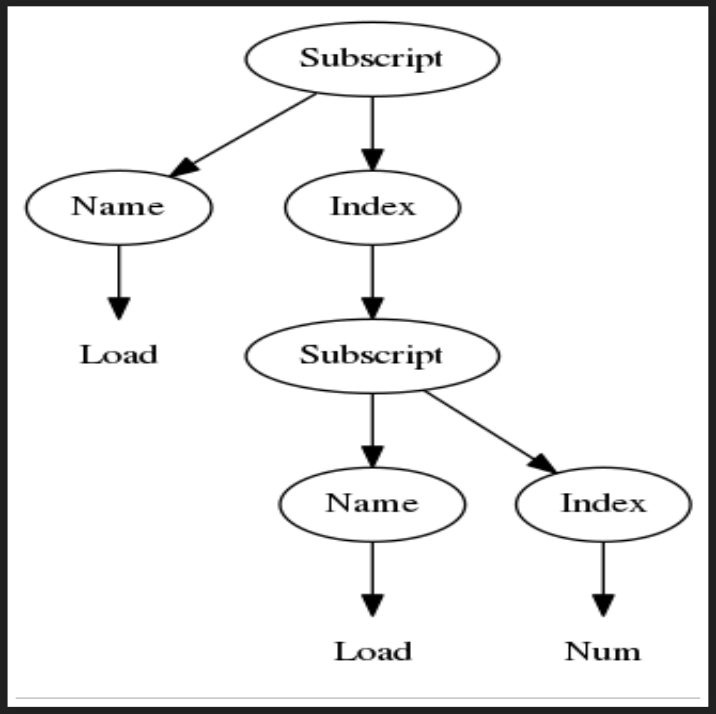
“Como mondongo.”

```
sentence
0. subject
      I
  1. predicate
     0. verb
        eat
     1. object
        mondongo
```

BYTECODE PARSE

```
prev[prev[0]]
```

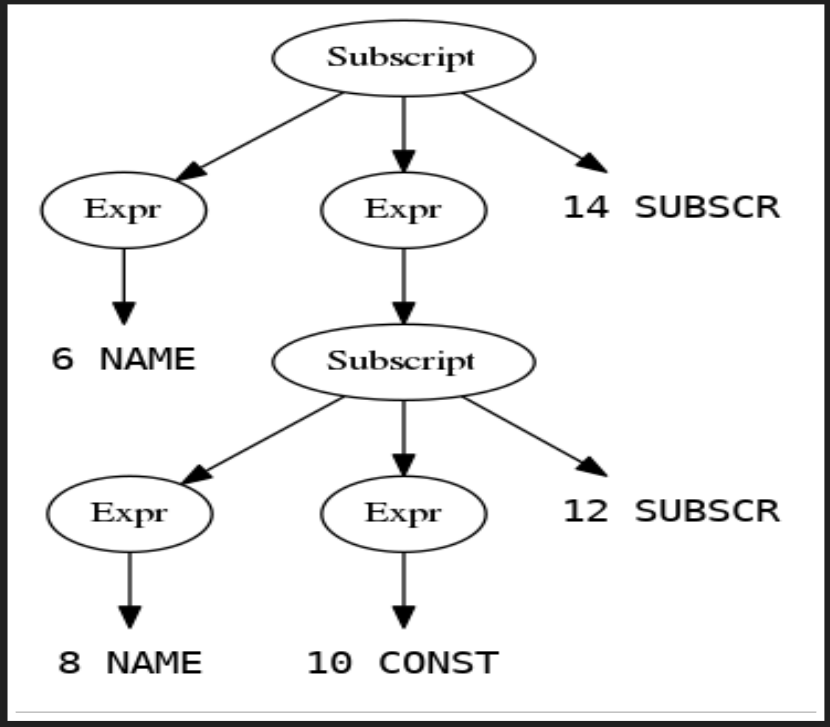
parses to:

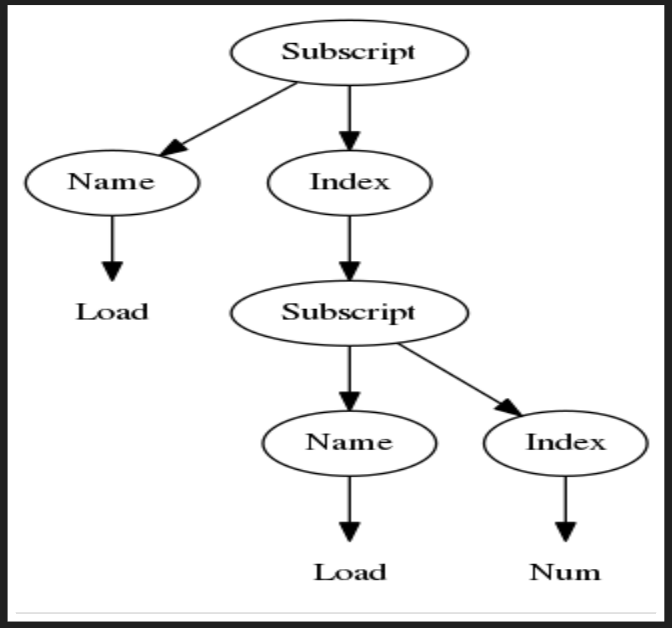


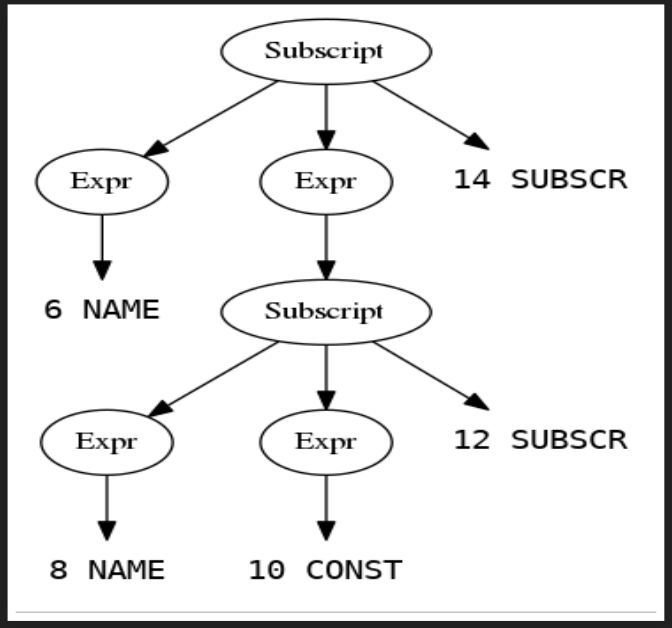
translates to:

```
2   6 LOAD_NAME "prev"
    8 LOAD_NAME "prev"
   10 LOAD_CONST 0
   12 BINARY_SUBSCR
   14 BINARY_SUBSCR
```

uncompyle6 (de)parses to:



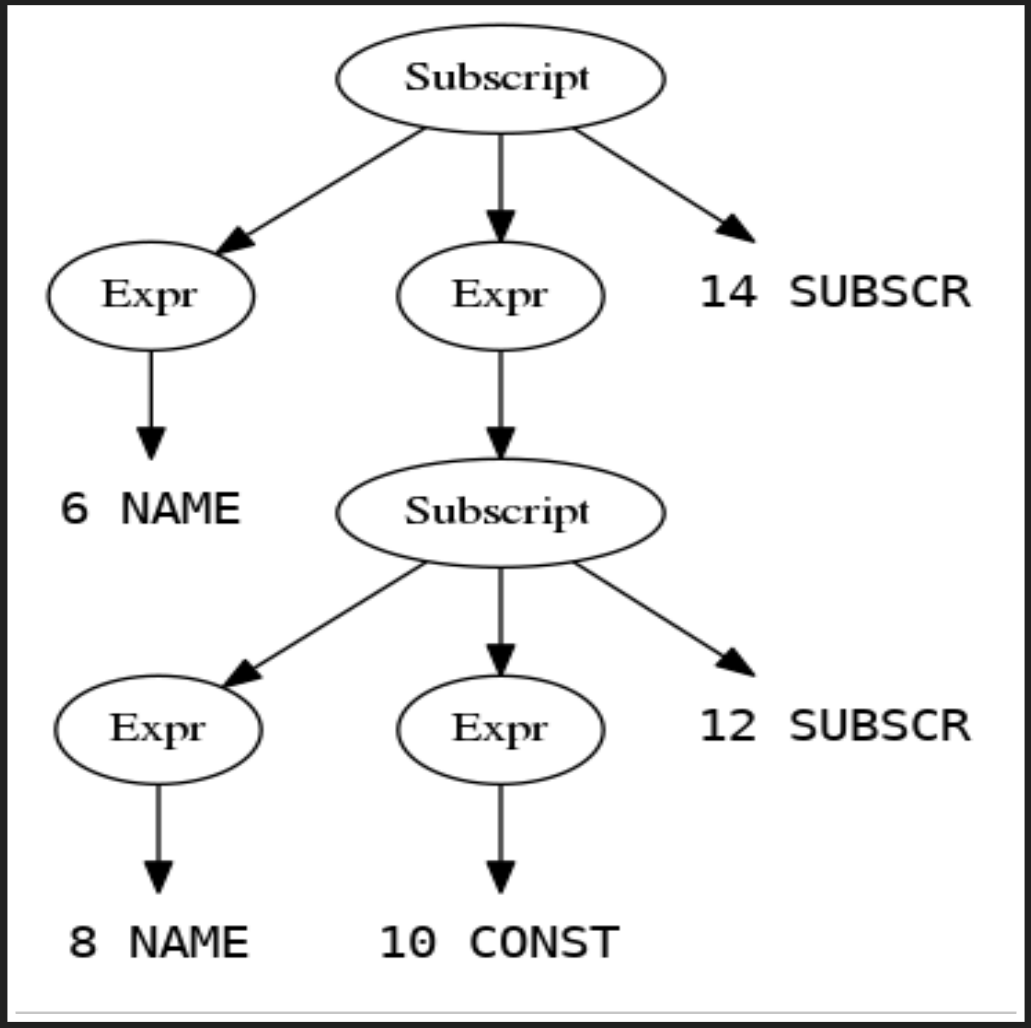

Subscript
Name
Index
Load
Subscript
Name
Index
Load
Num


Subscript
Expr
Expr
14 SUBSCR
6 NAME
Subscript
Expr
Expr
12 SUBSCR
8 NAME
10 CONST

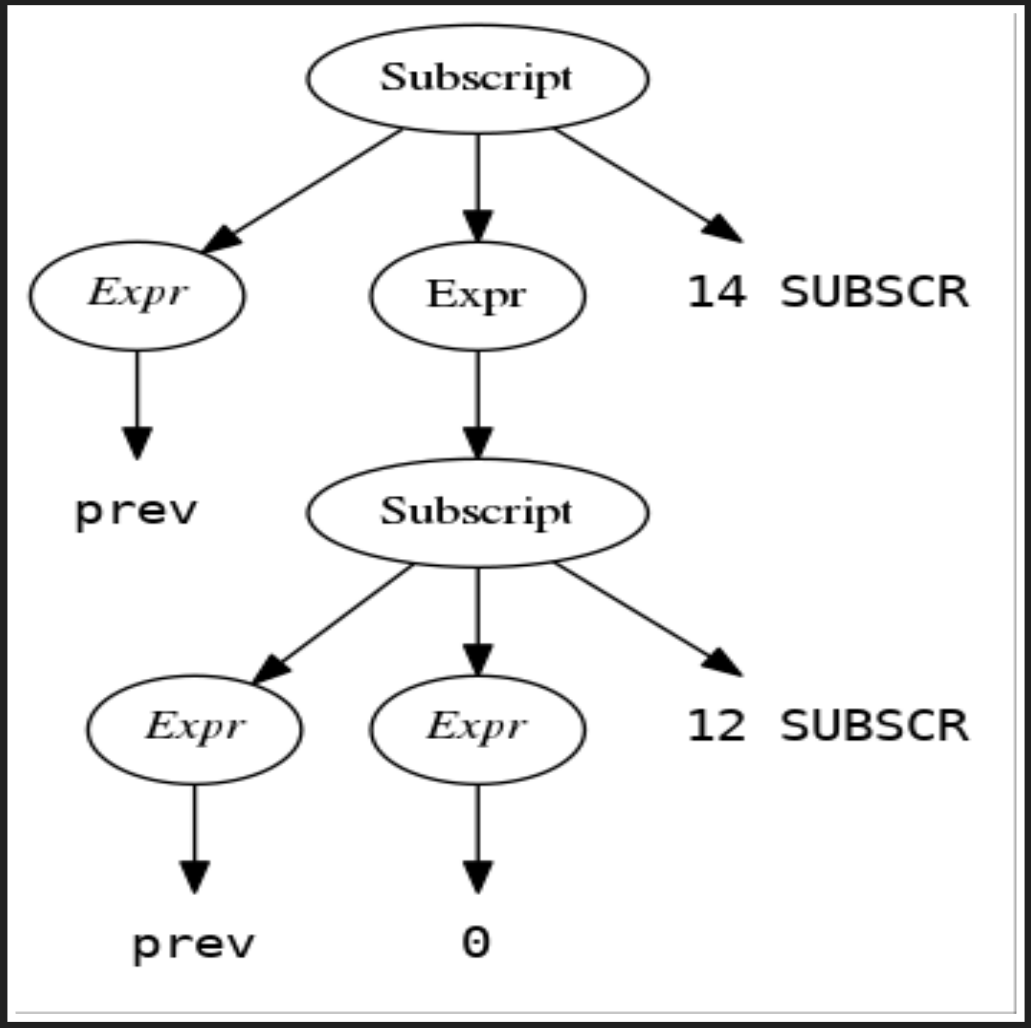
In listing form:

```
        Subscript
          0. Expr
             L.   2         6  LOAD_NAME 'prev'
          1. Expr
             Subscript
             0. Expr
                           8  LOAD_NAME 'prev'
             1. Expr
                          10  LOAD_CONST 0
             2.           12  BINARY_SUBSCR
          2.          14  BINARY_SUBSCR
...
```

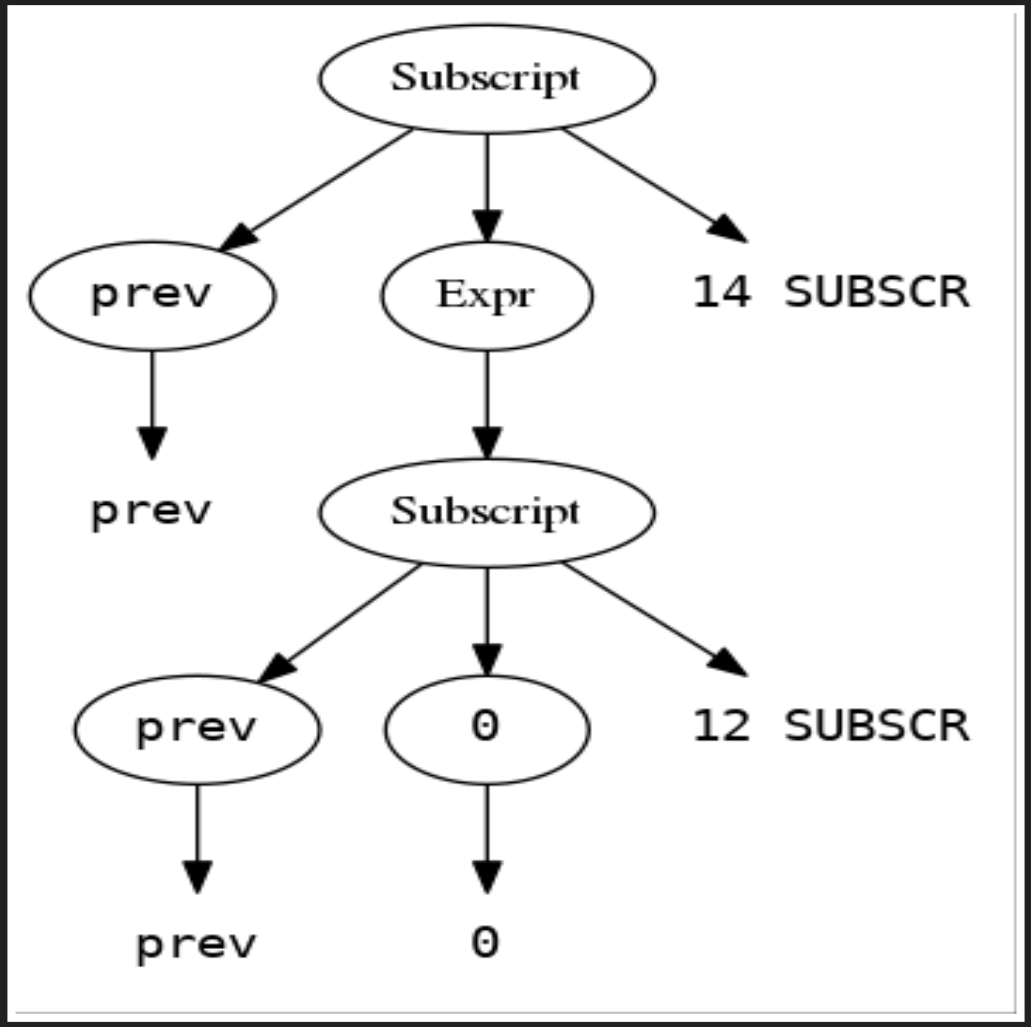
PARSE TO TEXT



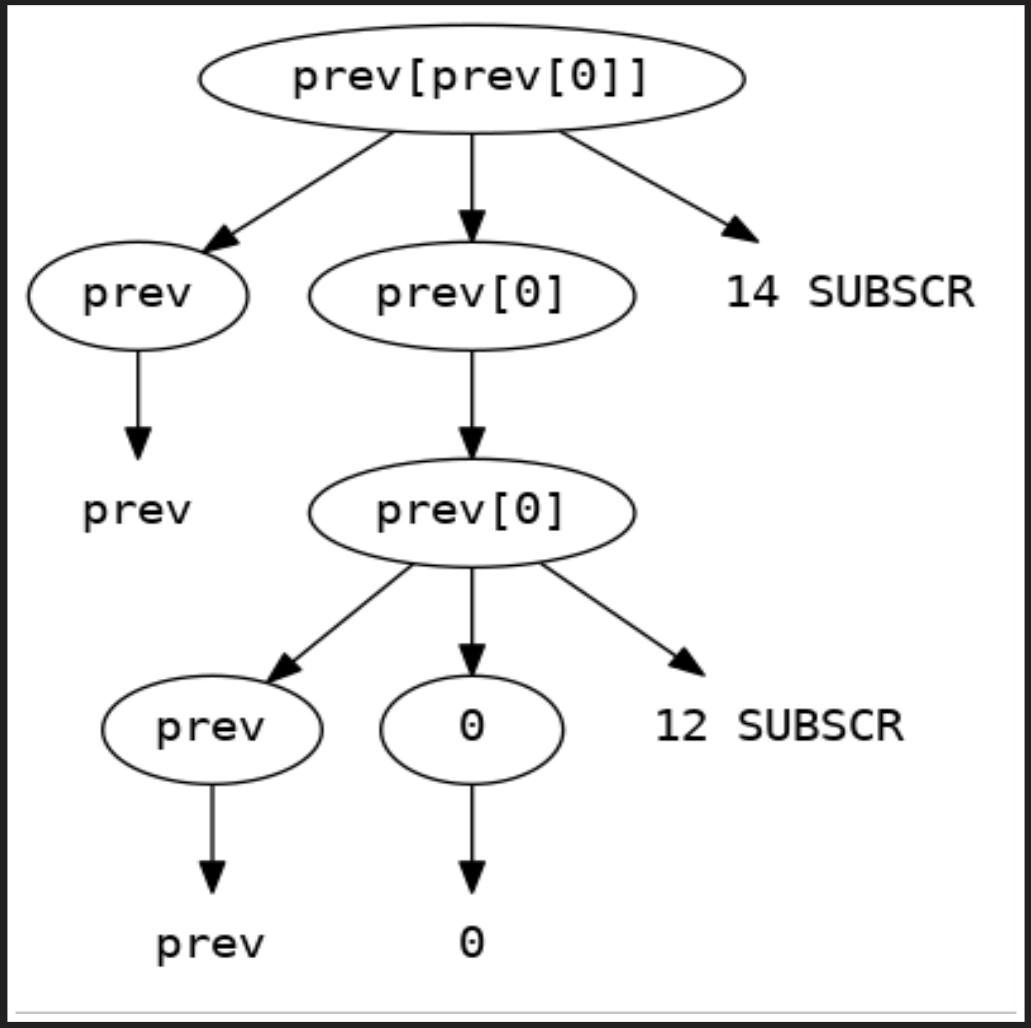
WITH INSTRUCTION OPERANDS



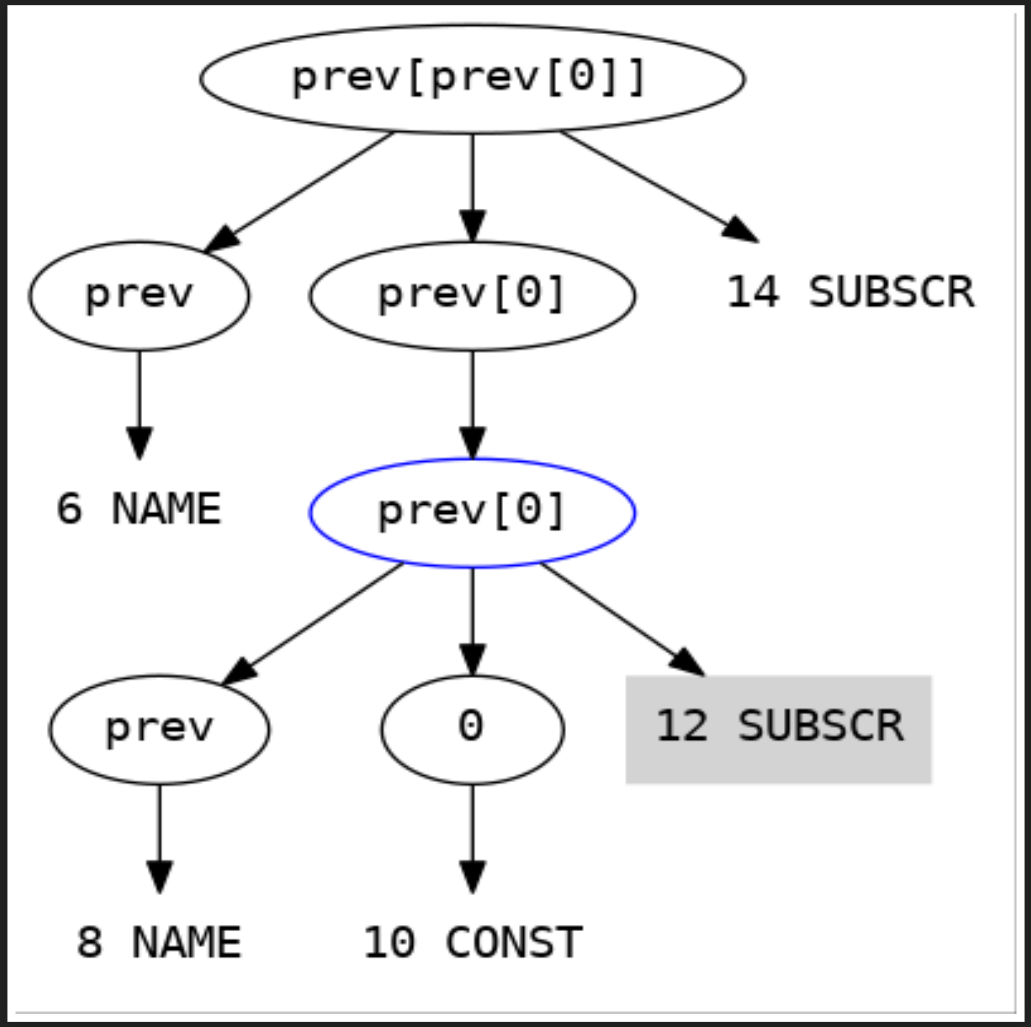
WITH "EXPR" RULE



WITH "SUBSCR" RULE



PARSE AT OFFSET 12

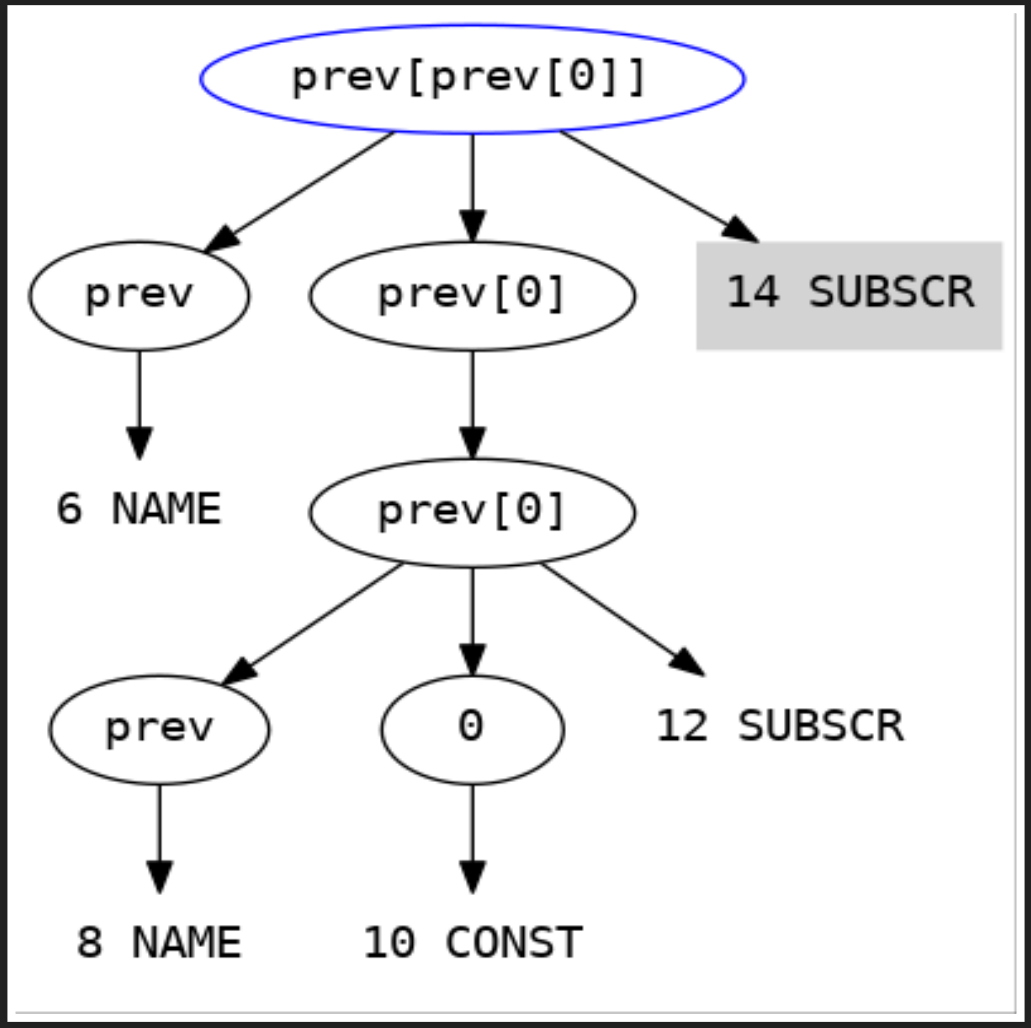


PARSE AT OFFSET 12 DISPLAY

```
instruction       12  BINARY_SUBSCR

prev[prev[0]]
     -------
```

PARSE AT OFFSET 14



PARSE AT OFFSET 14 TEXT DISPLAY

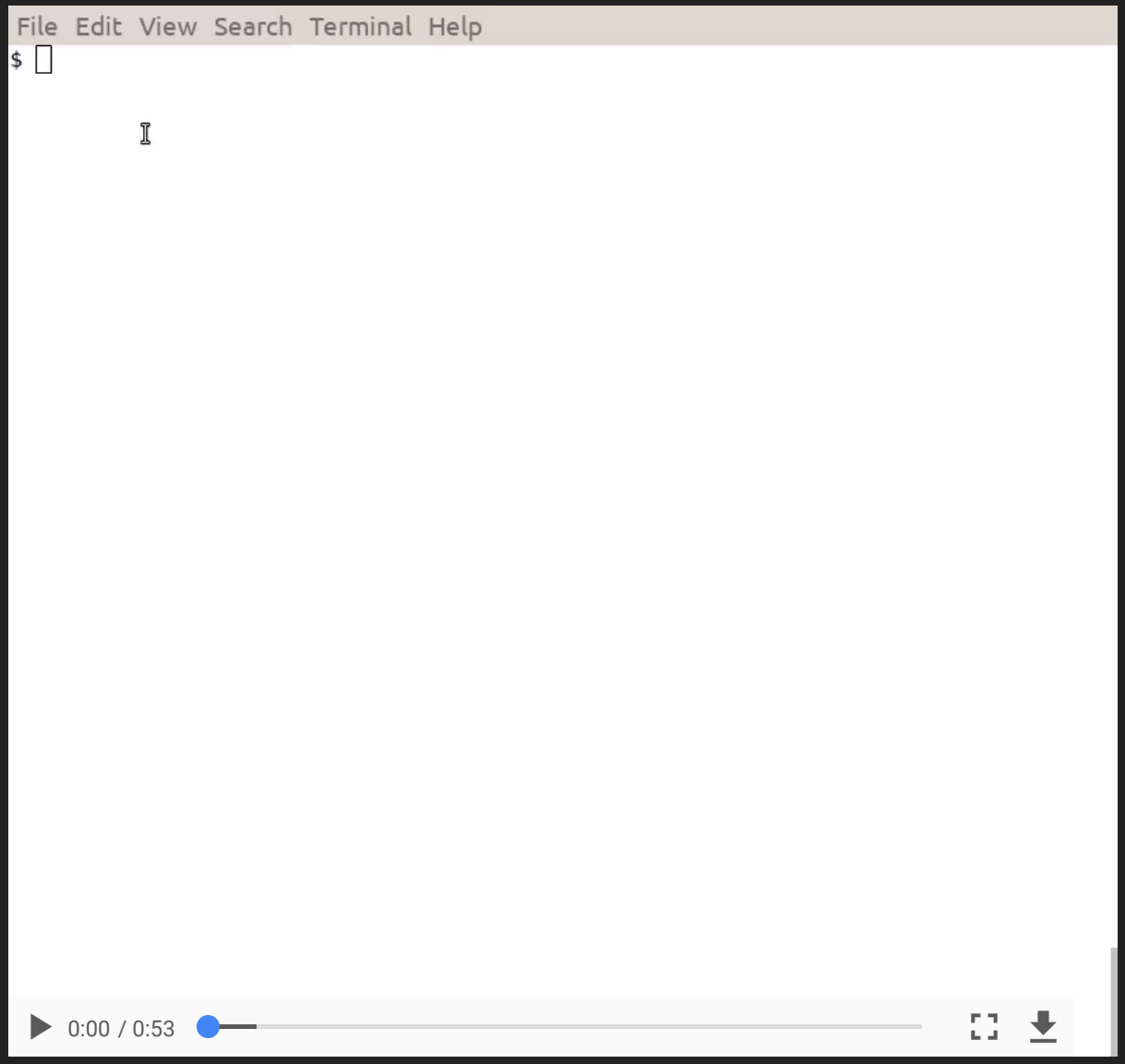
```
instruction       14  BINARY_SUBSCR

prev[prev[0]]
-------------
```


INTERMISSION

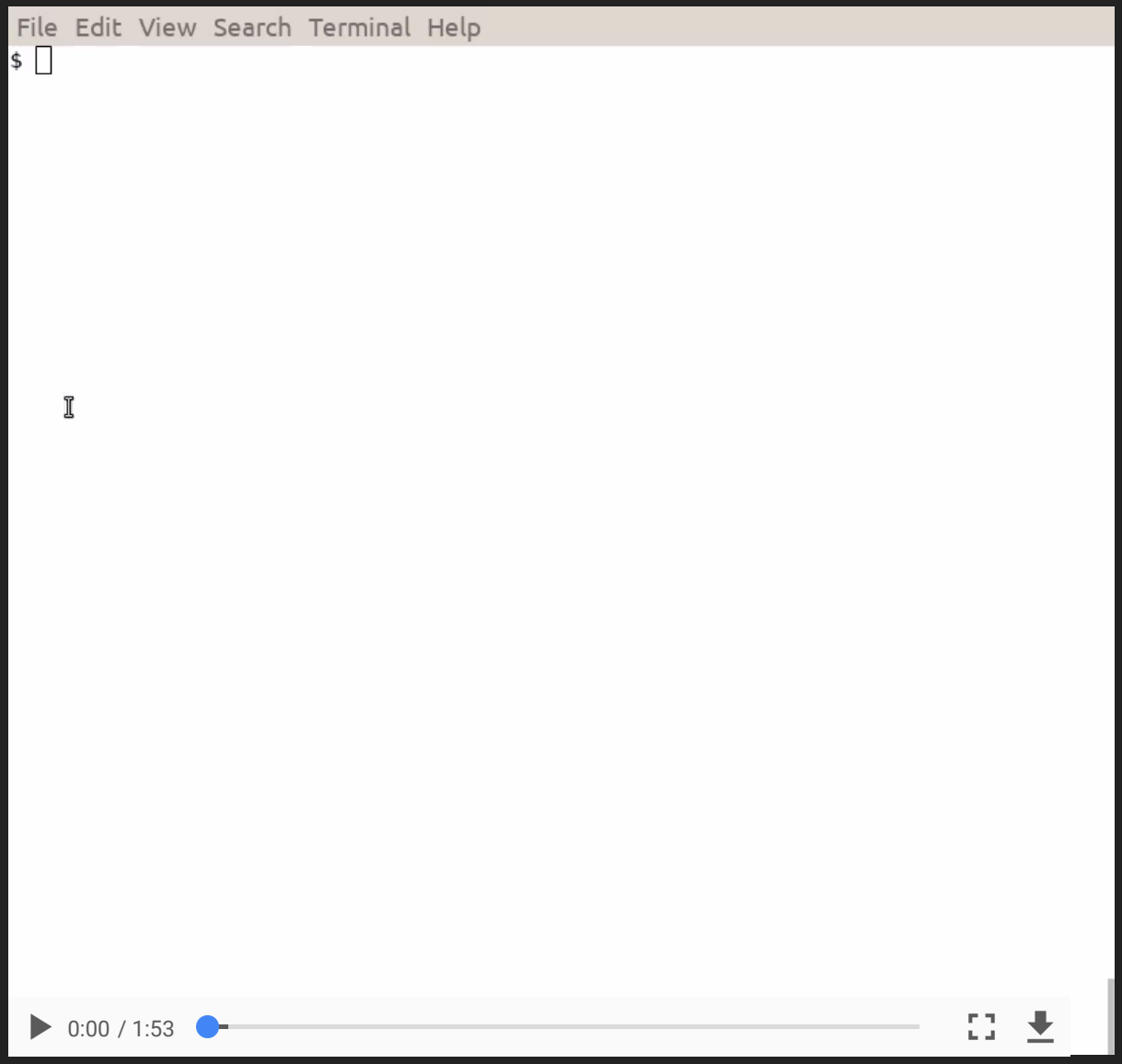
TREPAN2 HELP

link to video



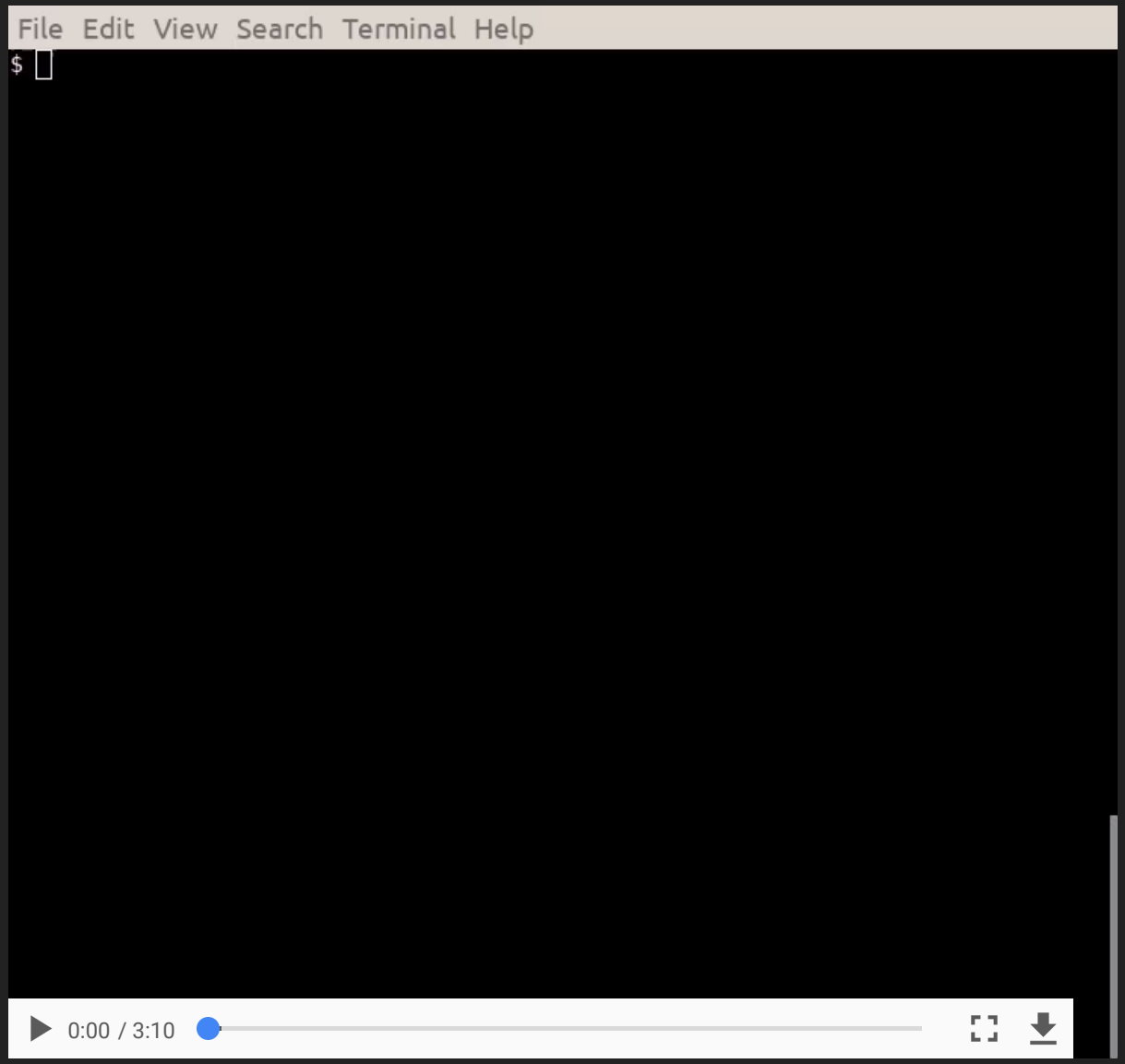
DEPARSING COMPREHENSION ERRORS

link to video



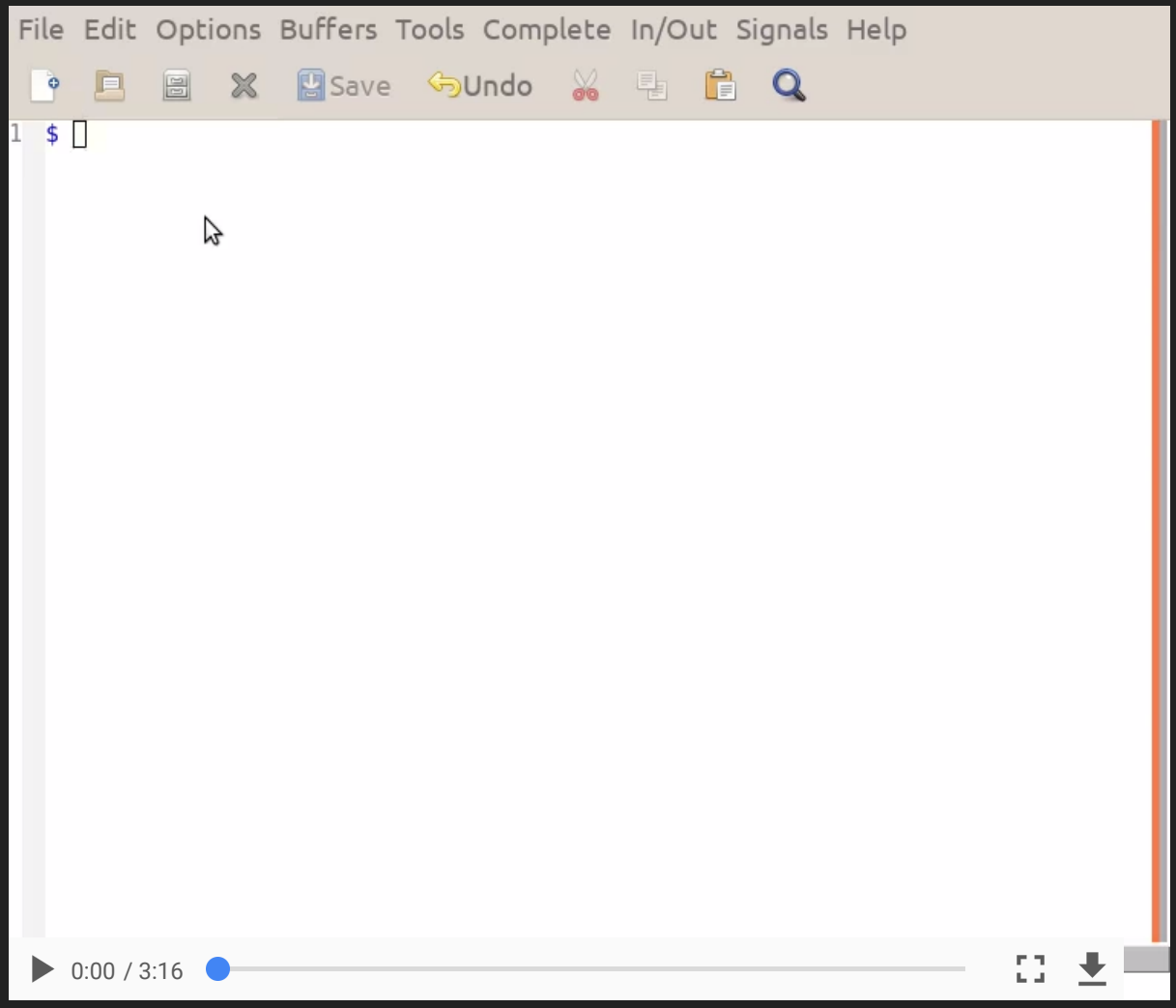
DEBUGGING CODE CREATED AT RUNTIME

link to video



DEBUGGING CODE WHEN THERE IS NO SOURCE

link to video



THERE IS MORE TO DO...

Emacs Lisp:

```
(/ a (/ b c)))
```

gives:

```
0       varref    a
1       varref    b
2       varref    c
3       quo
4       quo
5       return
```

deparses:

```
fn_exprs
  0. expr
    0. binary_expr
      0. expr
        name_expr
          0  VARREF     a
      1. expr
        binary_expr
          0. expr
            name_expr
              1  VARREF     b
          1. expr
              name_expr
              2  VARREF     c
          2. bin_op
```

Note the similarity in parse structure with previous examples

LINKS...

- Text and slides for this presentation
- Draft of an first part, in Spanish
- Python Decompiler
- Python Bytecode Library
- traceback module + deparsing
- Python 3 Debugger
- trepan2 Documentation
- Emacs Interface to Debuggers
- Decompilation Research Paper
- rocky@gnu.org, https://github.com/rocky

- fin