



FinnAPL spring 2025

# Dyalog version 20.0 & □SHELL

Peter Mikkelsen

# Introduction

- A quick look at Dyalog version 20.0
- Overview of `⎕SHELL`
  - "An improved version of `⎕SH`/`⎕CMD`, with more control over a lot of things"
  - This part is a re-run of a presentation done at Dyalog'24
  - The recording exists online

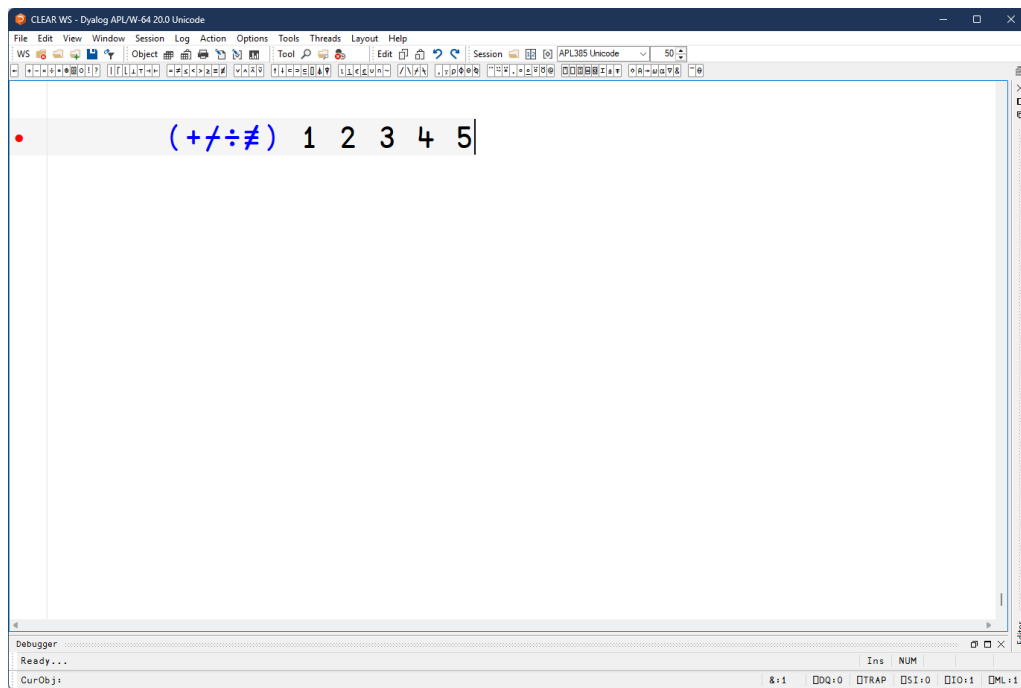
# Dyalog Version 20.0

- Expected to be released later this year
- Contains a handful of new features
  - Inline tracing
  - Behind operator `_`
  - Array notation
  - `⌈VGET` and `⌈VSET`
  - `⌈SHELL`
  - (the list is not complete)

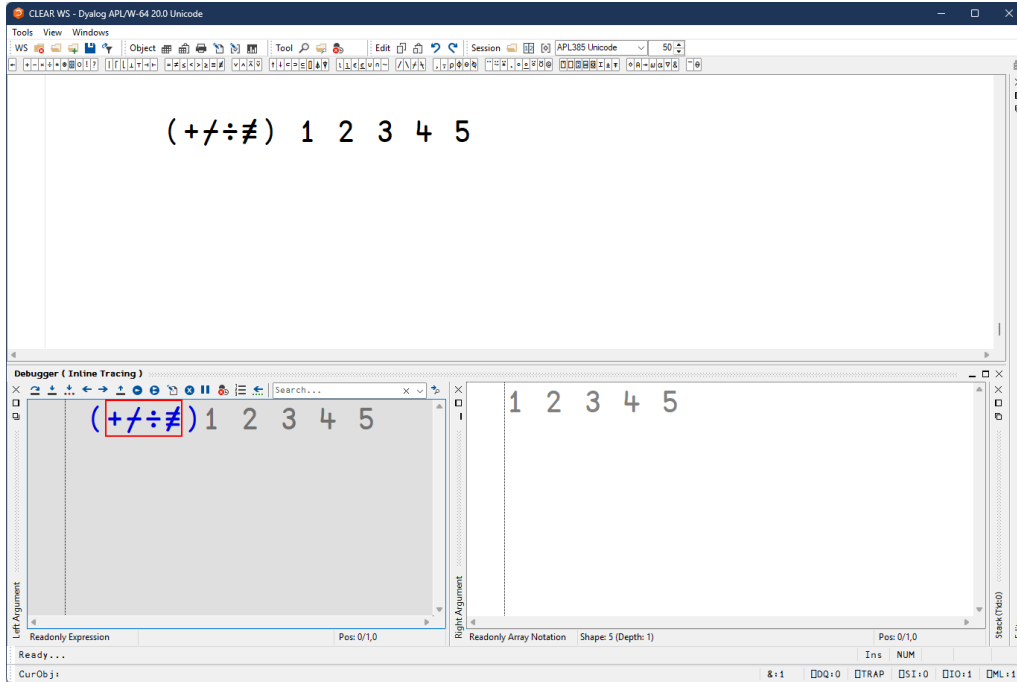
# Version 20.0: Inline tracing

- ◆ An extension to the tracer
  - ◆ Possible to step through the execution of individual primitives
  - ◆ Examination of intermediate results
  - ◆ Very helpful when working with tacit code
  - ◆ Ctrl-Alt-Enter

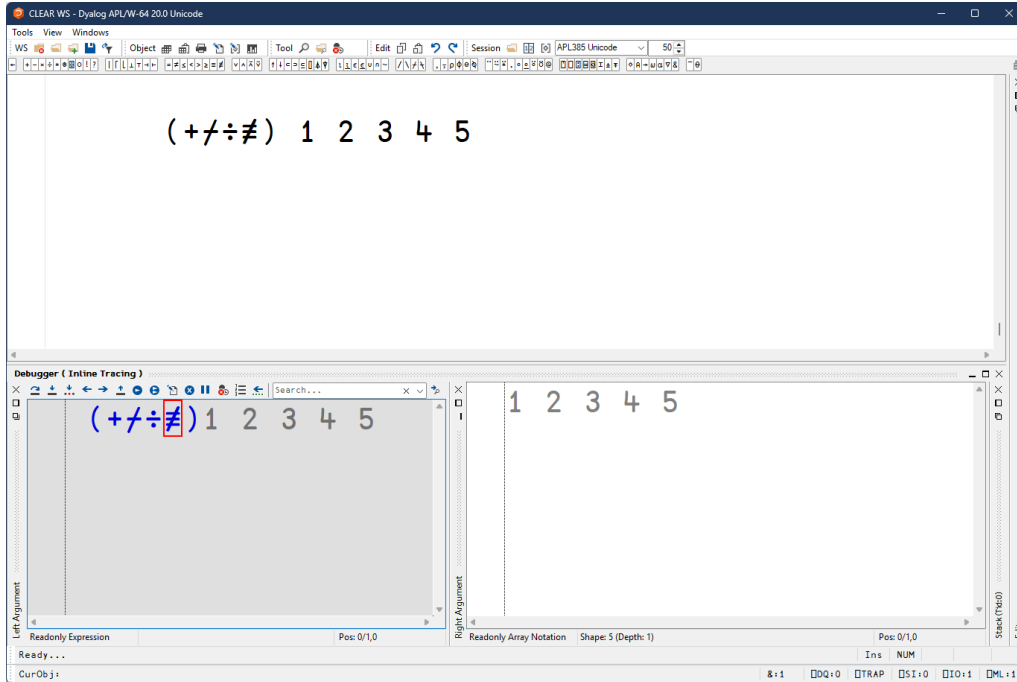
# Version 20.0: Inline tracing



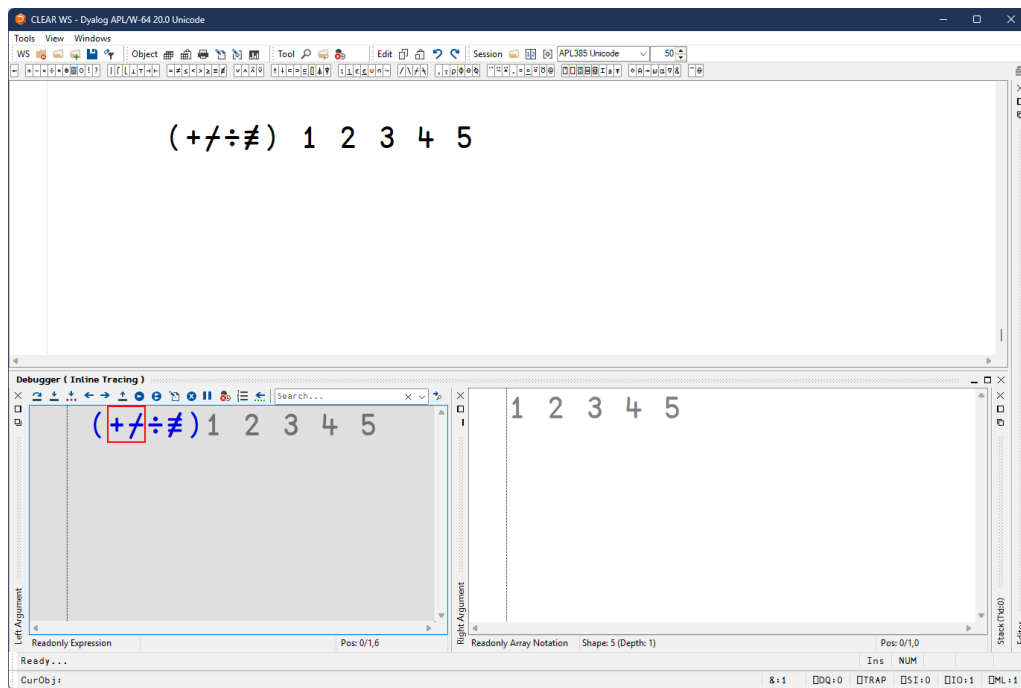
# Version 20.0: Inline tracing



# Version 20.0: Inline tracing

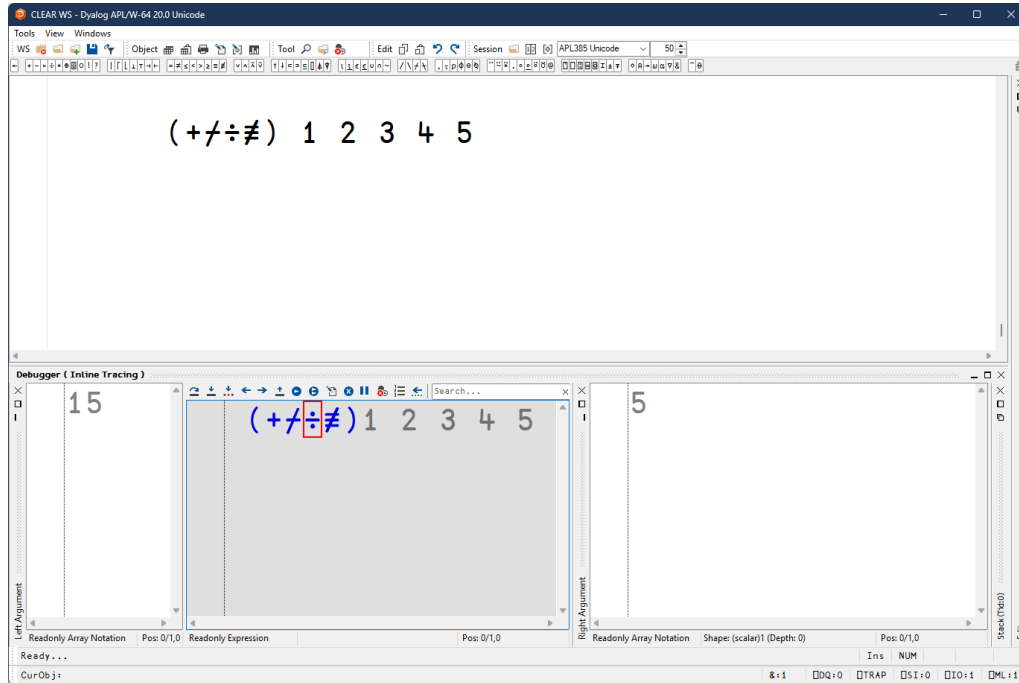


# Version 20.0: Inline tracing

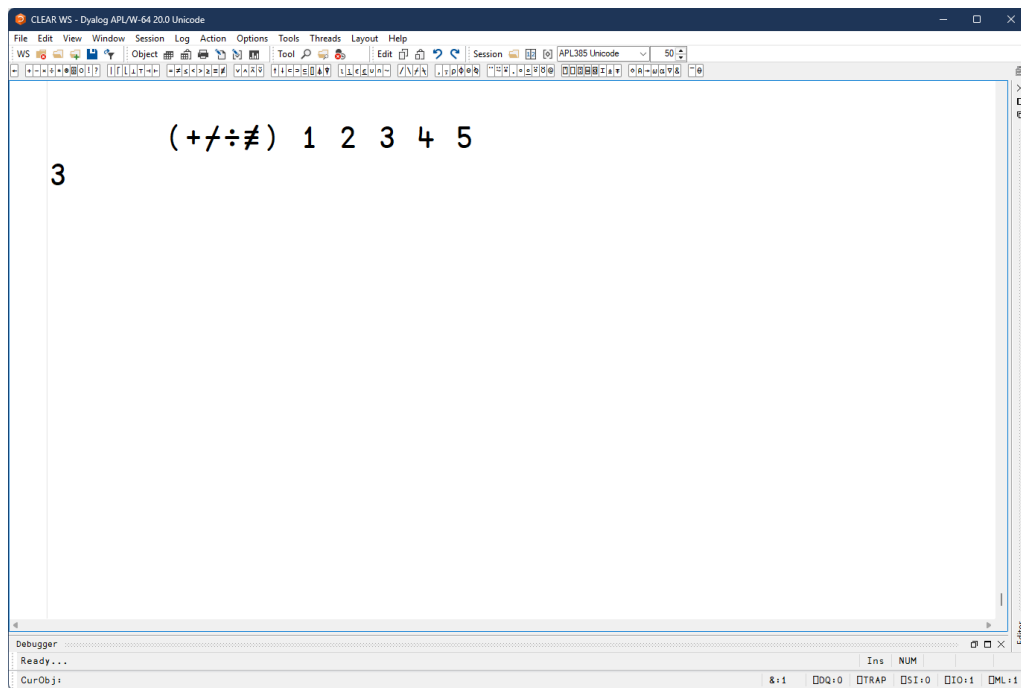




# Version 20.0: Inline tracing



# Version 20.0: Inline tracing



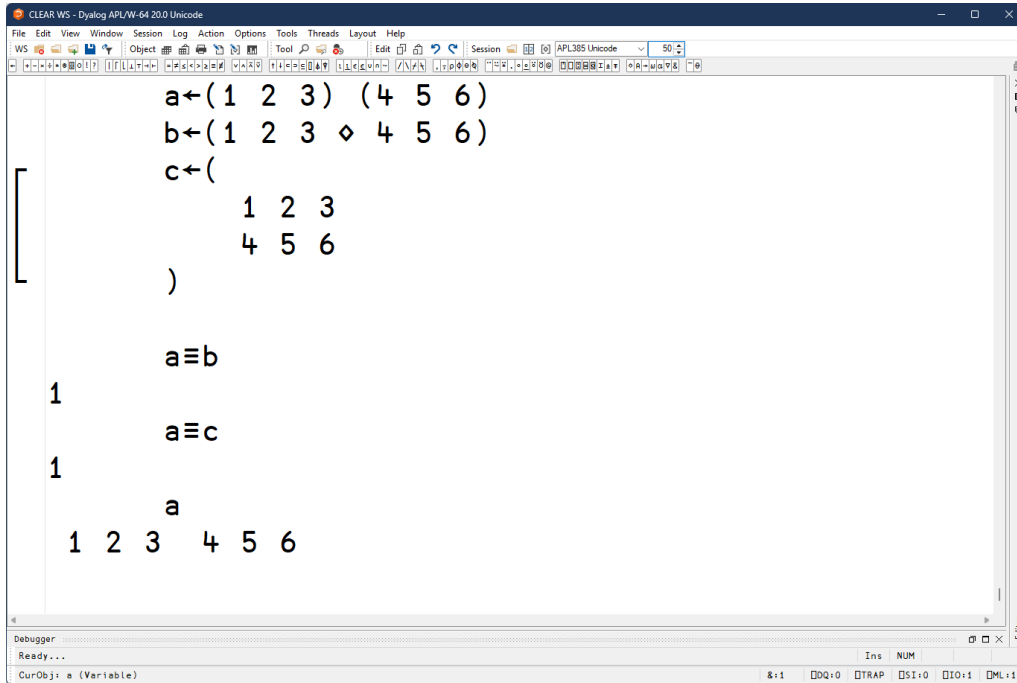
# Version 20.0: Behind operator $\circ$

- A new composition operator
- $f \circ g \ Y$  is equivalent to  $(f \ Y) \ g \ Y$ 
  - $\phi \circ \equiv$      $\mu$  Is it a palindrome?
- $X \ f \circ g \ Y$  is equivalent to  $(f \ X) \ g \ Y$ 
  - $- \circ \downarrow$      $\mu$  Drop from the right
- Mostly useful if you write tacit code
- Ctrl-Shift-f

# Version 20.0: Array notation

- ◆ A new way of writing arrays and namespaces
- ◆ Items are separated by ◆ or newlines

# Version 20.0: Array notation: vectors



The screenshot shows the Dyalog APL/WS 20.0 Unicode editor window. The main text area contains the following APL code:

```
a←(1 2 3) (4 5 6)
b←(1 2 3 ⋄ 4 5 6)
c←(
    1 2 3
    4 5 6
)

a≡b
1

a≡c
1

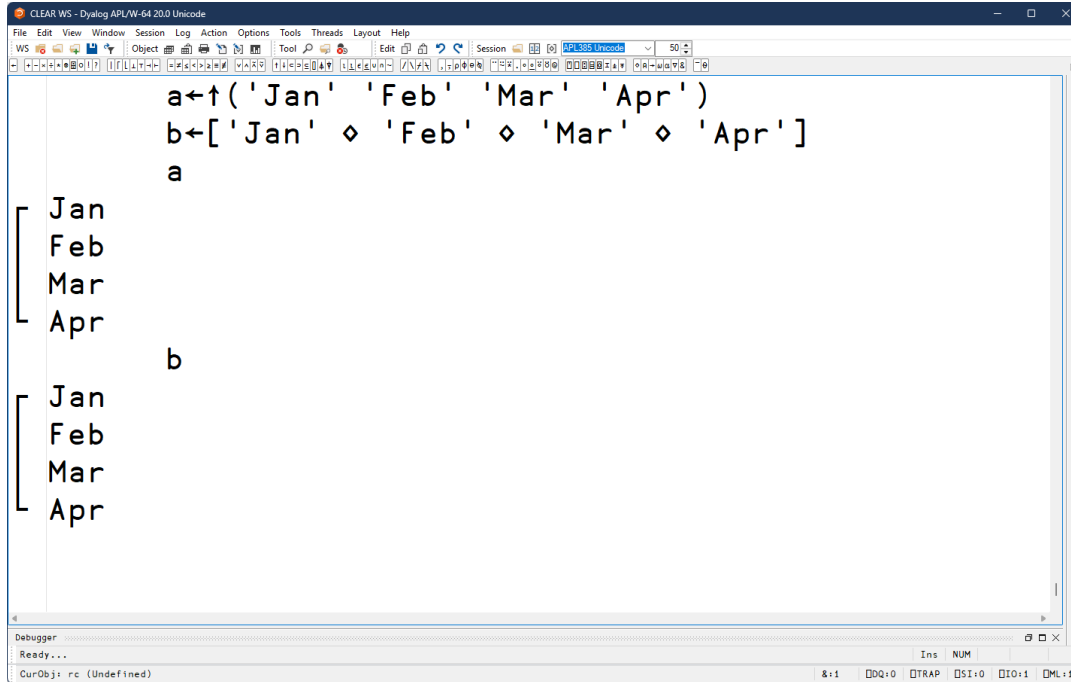
a
1 2 3 4 5 6
```

The execution results are displayed on the left side of the editor, corresponding to the last three lines of code. The results are:

- 1 (result of `a≡b`)
- 1 (result of `a≡c`)
- 1 2 3 4 5 6 (result of `a`)

The bottom status bar shows the current object is a (Variable).

# Version 20.0: Array notation: high-rank



```
CLEAR WS - Dyalog APL/WS-64 20.0 Unicode
File Edit View Window Session Log Action Options Tools Threads Layout Help
WS Edit View Window Session Log Action Options Tools Threads Layout Help
Object Tool Find Session Edit 524.335 Unicode 50
a←↑('Jan' 'Feb' 'Mar' 'Apr')
b←['Jan' ⋄ 'Feb' ⋄ 'Mar' ⋄ 'Apr']
a
b
```

Visual representation of the arrays:

a

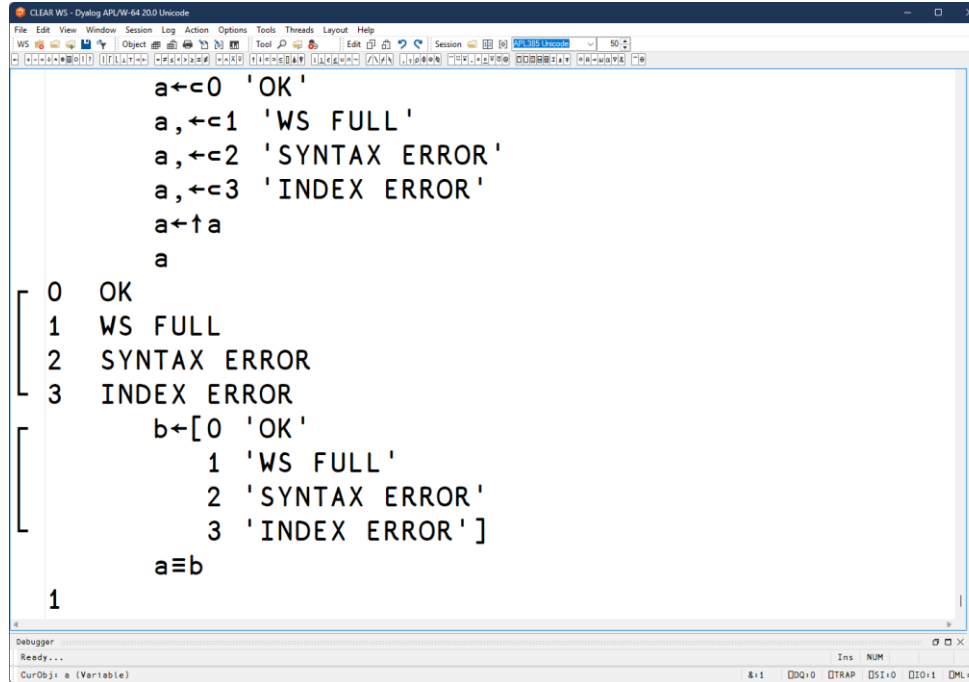
Jan
Feb
Mar
Apr

b

Jan	Feb	Mar	Apr
Jan	Feb	Mar	Apr

Debugger: Ready...  
CurObj: rc (Undefined)

# Version 20.0: Array notation: high-rank

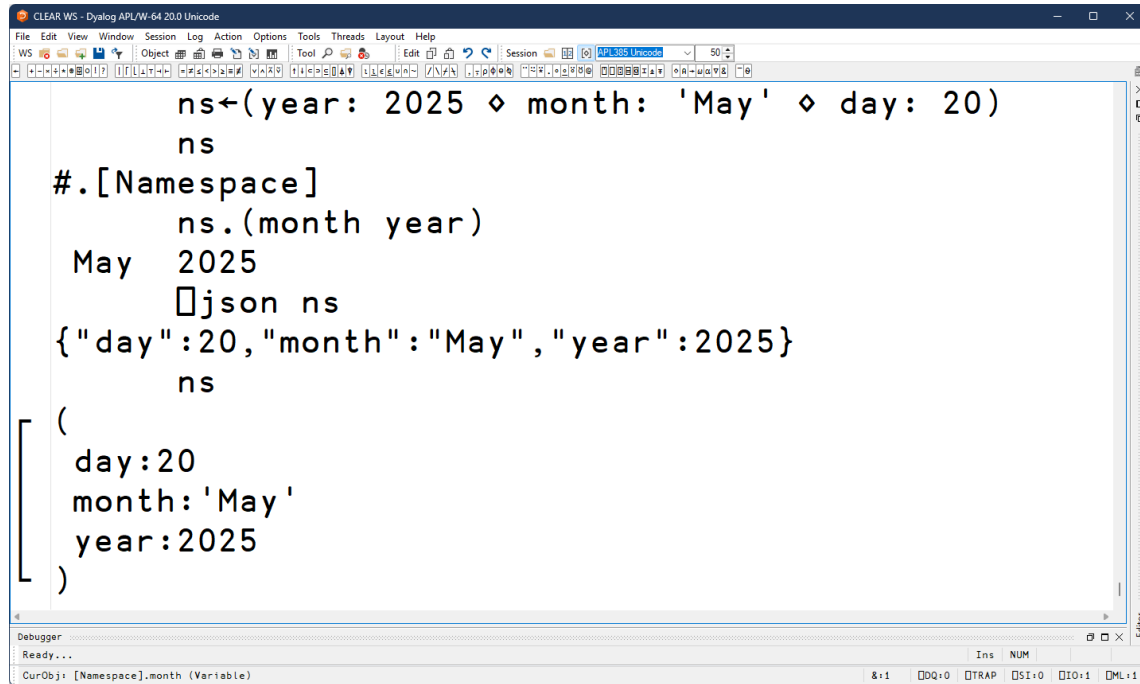


The screenshot shows the CLEAR WS - Dyalog APL/W-64 20.0 Unicode editor. The main window contains the following code:

```
a←c0 'OK'  
a,←c1 'WS FULL'  
a,←c2 'SYNTAX ERROR'  
a,←c3 'INDEX ERROR'  
a←↑a  
a  
  
[ 0 OK  
  1 WS FULL  
  2 SYNTAX ERROR  
  3 INDEX ERROR  
  b←[0 'OK'  
      1 'WS FULL'  
      2 'SYNTAX ERROR'  
      3 'INDEX ERROR']  
  a≡b  
1
```

The bottom of the window shows a Debugger panel with 'Ready...' and 'CurObj: a (Variable)'.

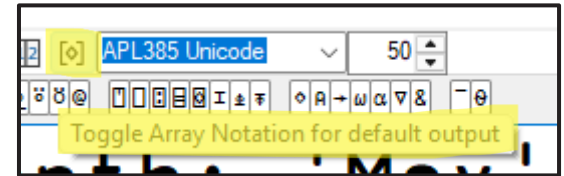
# Version 20.0: Array notation: namespaces



```
ns←(year: 2025 ⋄ month: 'May' ⋄ day: 20),
ns
#.[Namespace]
ns.(month year)
May 2025
⌞json ns
{"day":20,"month":"May","year":2025}
ns
(
 day:20
 month:'May'
 year:2025
)
```

Debugger: Ready...

CurObj: [Namespace].month (Variable)

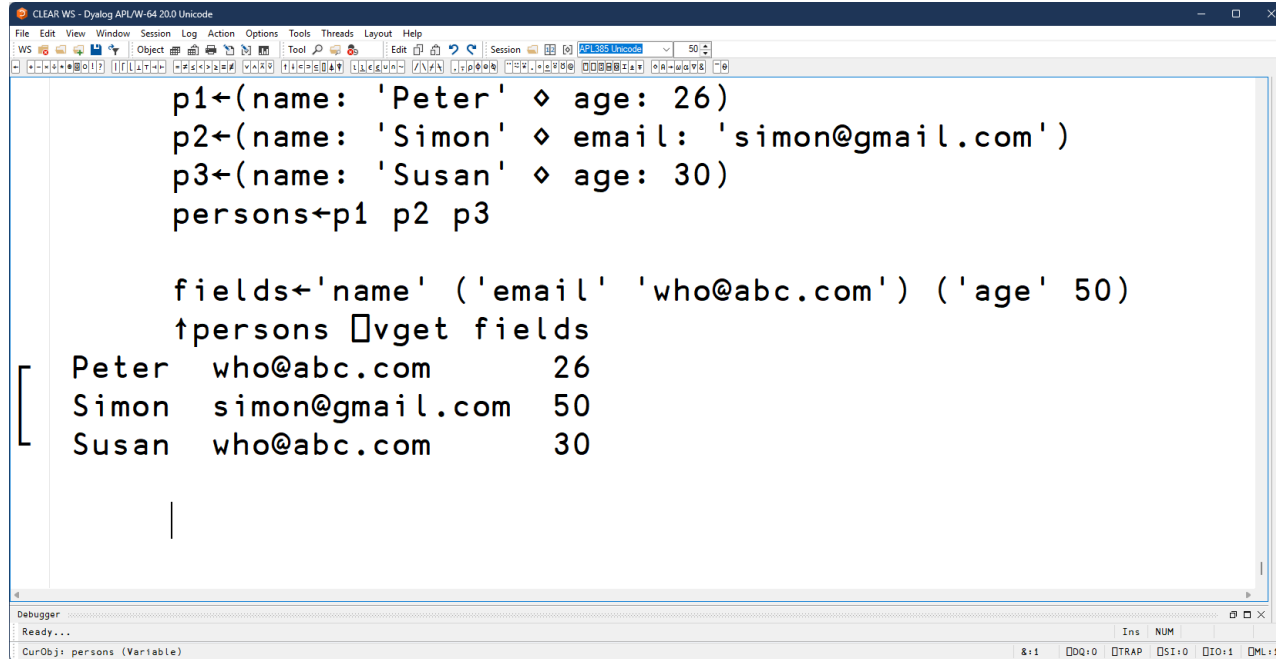




# Version 20.0: `▯VGET` and `▯VSET`

- ◆ System functions which make it possible to set and get values of variables by name
  - ◆ Without the use of `▯` which can be dangerous
  - ◆ With the possibility of providing a default value
  - ◆ In a number of namespaces at once
  - ◆ Very useful for looking up data in external data sources where some data is missing, such as JSON from a web API

# Version 20.0: □VGET and □VSET



The screenshot shows the CLEAR WS - Dyalog APL/W-64 20.0 Unicode editor. The main window contains the following APL code:

```
p1←(name: 'Peter' ♦ age: 26)
p2←(name: 'Simon' ♦ email: 'simon@gmail.com')
p3←(name: 'Susan' ♦ age: 30)
persons←p1 p2 p3

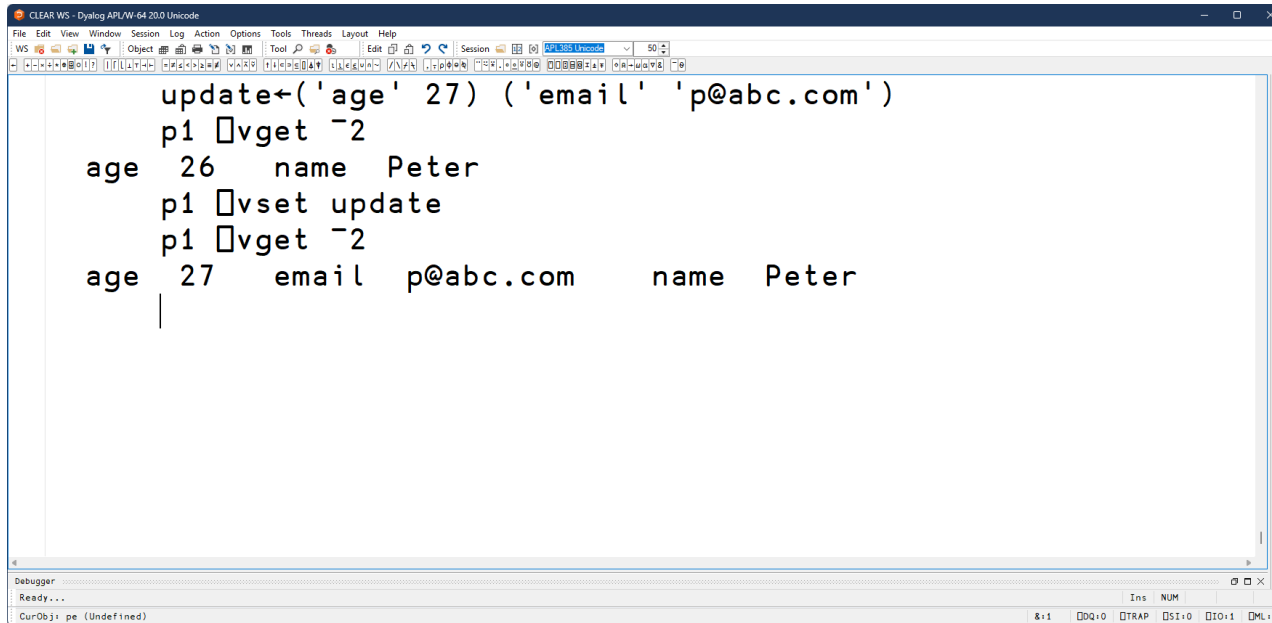
fields←'name' ('email' 'who@abc.com') ('age' 50)
↑persons □vget fields
```

The output of the code is displayed in a table format:

Peter	who@abc.com	26
Simon	simon@gmail.com	50
Susan	who@abc.com	30

The bottom of the window shows a Debugger panel with the status "Ready..." and a variable list at the bottom right with columns for Ins, NUM, and DML.

# Version 20.0: □VGET and □VSET



```
update←('age' 27) ('email' 'p@abc.com')
p1 □vget ~2
age 26    name Peter
p1 □vset update
p1 □vget ~2
age 27    email p@abc.com    name Peter
|
```

The screenshot shows the Dyalog APL/W-64 20.0 Unicode editor window. The title bar reads 'CLEAR WS - Dyalog APL/W-64 20.0 Unicode'. The menu bar includes File, Edit, View, Window, Session, Log, Action, Options, Tools, Threads, Layout, and Help. The toolbar contains various icons for file operations, editing, and session management. The main text area contains the APL code shown in the code block. The bottom status bar displays 'Debugger Ready...' and 'CurObj: pe (Undefined)'. The right side of the status bar shows various flags: &1, □□0, □TRAP, □SI+0, □IO+1, and □ML+1.

# Version 20.0: □SHELL

- ◆ The rest of the presentation 😊

# Why provide an alternative to `□SH`?

- ❖ Short version: `□SH` makes too many choices for us
  - ❖ If we disagree, there is no way to change them
- ❖ Examples:
  - ❖ Always picks up standard output as lines of text with specific encoding
  - ❖ Assumes non-zero exit codes are always bad
  - ❖ Blocks all other thread while it waits
- ❖ Difficult to extend `□SH` enough, without breaking changes

# □SHELL overview

- Monadic system function `R←□SHELL cmd`
  - `'Command arg1 arg2'` - run via system shell
  - `'Command' 'arg1' 'arg2'` - execute command directly
- Supports many variant options to change the defaults
- Result is a five-element nested vector
  - `(StreamData StreamIds ExitCode ExitReason Pid)`
  - More complicated, but `⇒⇒□SHELL` is almost equivalent

# A look at the result

- ◆ `StreamData` and `StreamIds`
  - ◆ Vectors of the same length
- ◆ `□SHELL` can collect output from different "streams"
  - ◆ Controllable via variant options
- ◆ Each collected stream has its number in `StreamIds`
- ◆ Data at the corresponding position of `StreamData`
  - ◆ Default is to redirect `stderr` (2) to `stdout` (1), and collect `stdout`

# A look at the result

- ExitCode and ExitReason

- Integers that tells us why □SHELL stopped

- ExitReasons:

- 0: normal exit. ExitCode is the exit code
  - 1: terminated by signal. ExitCode is the negated signal number
  - 2: □SHELL timed out. ExitCode is always -1006
  - 3: □SHELL interrupted. Exitcode is always -1002



# A look at the result

- ◆ `P i d`
  - ◆ Process ID if the child process is still running
- ◆ This is only relevant when `Ex i t Reason` is 2 or 3
- ◆ Additional cleanup might be required to get rid of the process
  - ◆ Some I-beams will be provided to help with that

# Demo

- ◆ Let's have a quick look in the session
- ◆ (Screenshots of the demo)



Activities

Ride-4.5

Sep 16 17:57

da

CLEAR WS - Dyalog APL/5-64

File Edit View Window Action Threads Help

← + - × ÷ \* ⊗

⊖ ∘ ! ?

| [ ] ⊥ ⊤ ⊢ ⊣

= ≠ ≤ < > ≥ ≡ ≠

√ ∨ ∧ ∼ ∨

↑

⌵ ⌶ ⌷ ⌸ ⌹ ⌺ ⌻ ⌼ ⌽ ⌾ ⌿

⋈ ⋉ ⋊ ⋋ ⋌ ⋍ ⋎ ⋏ ⋐ ⋑ ⋒ ⋓ ⋔ ⋕ ⋖ ⋗ ⋘ ⋙ ⋚ ⋛ ⋜ ⋝ ⋞ ⋟ ⋠ ⋡ ⋢ ⋣ ⋤ ⋥ ⋦ ⋧ ⋨ ⋩ ⋪ ⋫ ⋬ ⋭ ⋮ ⋯ ⋰ ⋱ ⋲ ⋳ ⋴ ⋵ ⋶ ⋷ ⋸ ⋹ ⋺ ⋻ ⋼ ⋽ ⋾ ⋿ ⋰ ⋱ ⋲ ⋳ ⋴ ⋵ ⋶ ⋷ ⋸ ⋹ ⋺ ⋻ ⋼ ⋽ ⋾ ⋿

⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿ ⊿

⊞ ⊟ ⊠ ⊡ ⊢ ⊣ ⊤ ⊥ ⊦ ⊧ ⊨ ⊩ ⊪ ⊫ ⊬ ⊭ ⊮ ⊯ ⊰ ⊱ ⊲ ⊳ ⊴ ⊵ ⊶ ⊷ ⊸ ⊹ ⊺ ⊻ ⊼ ⊽ ⊾ ⊿ ⊿

␣ I need to quote the argument

⊞SH'unicode --brief "APL FUNCTIONAL SYMBOL OMEGA"'

␣ μ U+2375 APL FUNCTIONAL SYMBOL OMEGA

␣ ¹ U+2379 APL FUNCTIONAL SYMBOL OMEGA UNDERBAR

␣⊞SH'unicode --brief "APL FUNCTIONAL SYMBOL OMEGA"'

␣ μ U+2375 APL FUNCTIONAL SYMBOL OMEGA

␣ ¹ U+2379 APL FUNCTIONAL

SYMBOL OMEGA UNDERBAR

&: 2

⊞DQ: 0

⊞TRAP

⊞SI: 0

⊞IO: 1

⊞ML: 1

Pos: 197/198,0





SYMBOL OMEGA UNDERBAR

Ⓐ Now using ⚡SHELL

⚡SHELL'unicode --brief "APL FUNCTIONAL SYMBOL OMEGA"'

ω U+2375 APL FUNCTIONAL SYMBOL OMEGA

⏟ U+2379 APL FUNCTIONAL SYMB

OL OMEGA UNDERBAR

Ⓐ I can now avoid quoting

⚡SHELL'unicode' '--brief' 'APL FUNCTIONAL SYMBOL OMEGA'|

&: 2 ⚡DQ: 0 ⚡TRAP ⚡SI: 0 ⚡IO: 1 ⚡ML: 1 Pos: 206/207,63



OL OMEGA UNDERBAR

⠁ I can now avoid quoting

⇒⠈SHELL'unicode' '--brief' 'APL FUNCTIONAL SYMBOL OMEGA'

ω U+2375 APL FUNCTIONAL SYMBOL OMEGA

⠈ U+2379 APL FUNCTIONAL SYMB

OL OMEGA UNDERBAR

⠁ We can look at the full result

⠈SHELL'unicode' '--brief' 'APL FUNCTIONAL SYMBOL OMEGA'|

&: 2 ⠈DQ: 0 ⠈TRAP ⠈SI: 0 ⠈IO: 1 ⠈ML: 1 Pos: 214/215,61

Activities

Ride-4.5

Sep 16 17:57

da

CLEAR WS - Dyalog APL/S-64

File Edit View Window Action Threads Help

← + - × ÷ \* ⊗ □ ○ ! ? | ⌈ ⌊ ⊥ ⊤ ⊢ ⊣

↓ < > ≤ ≥ ≡ ≠ ∨ ∧ ∼ ∨ ↑

□ □ □ □ □ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥ ⊥

□SHELL'unicode' '--brief' 'APL FUNCTIONAL SYMBOL OMEGA'

ω U+2375 APL FUNCTIONAL SYMBOL OMEGA

ω U+2379 APL FUNCTIONAL SY

MBOL OMEGA UNDERBAR

1 0 0 -1

&: 2 □DQ: 0 □TRAP □SI: 0 □IO: 1 □ML: 1 Pos: 229/230,0





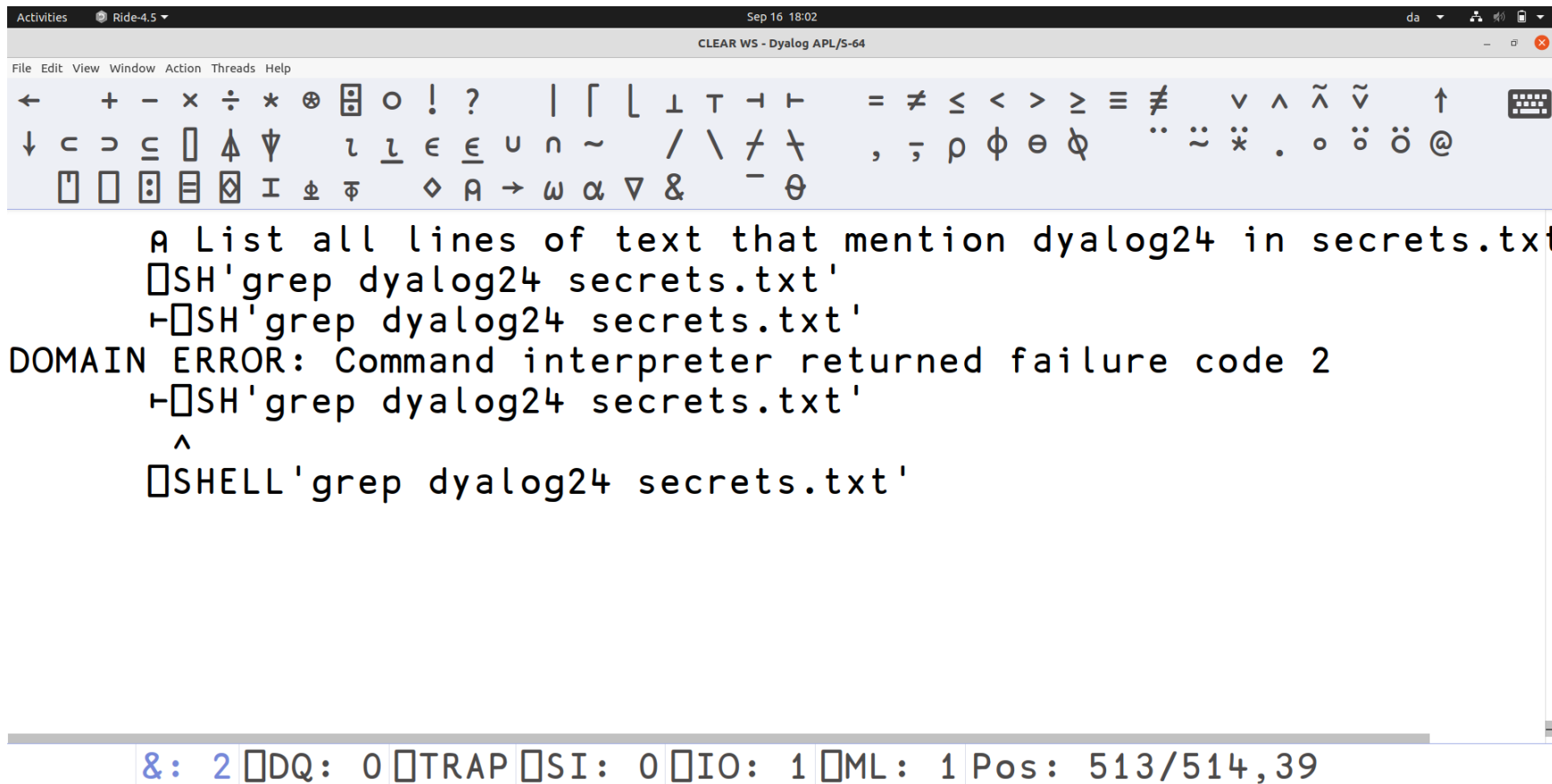
ÿþ

⠇⠈SHELL'unicode --brief -iUTF32 0x2379'

ω U+2379 APL FUNCTIONAL SYMBOL OMEGA UNDERBAR

⠇ List all lines of text that mention dyalog24 in secrets.txt

&: 2 ⠈DQ: 0 ⠈TRAP ⠈SI: 0 ⠈IO: 1 ⠈ML: 1 Pos: 238/239,67



```
h-SH'grep dyalog24 secrets.txt'
```

```
❏ SHELL 'grep dyalog24 secrets.txt'
```

```
grep: secrets.txt: No such file or directory
```

1	2	0	-1
---	---	---	----

```
&: 2 □DQ: 0 □TRAP □SI: 0 □IO: 1 □ML: 1 Pos: 521/522,0
```

# Overview of the variant options

- ◆ Many variant options
- ◆ Typical usage won't require much, if any at all

# 'WorkingDir' path

- Simply changes the working directory of the child process
- By default, the same as the interpreter

# 'InheritEnv' bool

- ✧ Every process has a set of environment variables
- ✧ By default, inherits the set from the interpreter

# 'Env' namesAndValues

- ✧ Allows the user to add/set additional environment variables
  - ✧ Overwrites any inherited values
- ✧ Examples:
  - ✧ Provide configuration options
  - ✧ Set an API key
- ✧ 2 column matrix or name-value pairs

# ❏ 'Shell' shellSpec

- ❖ Makes it possible to use another shell than powershell or /bin/sh
  - ❖ Only relevant when right argument is simple
- ❖ Examples:
  - ❖ '/bin/bash' '-c'
  - ❖ 'CMD.EXE' '/C'



# ❏ 'ExitCheck' bool

- ❏ ❏SH checks the exit code and produce domain error if non-zero
- ❏ In ❏SHELL that is optional
  - ❏ Makes it possible to deal with non-zero exits and still get the output produced
  - ❏ Much easier figure out what went wrong when the error messages aren't lost

# ⏱ 'Timeout' milliseconds

- Some programs finish "quickly"
- Some might hang or take a very long time
- Set a timeout and cause the ⏏SHELL call to return after a while
  - No TIMEOUT error
  - Data collected so far is available

# ❏ 'Signal' number

- ❏ ❏SHELL can stop before the child process
- ❏ Child process potentially still running
- ❏ Requires cleanup
- ❏ Automatically send a signal to the child process
  - ❏ SIGTERM by default
  - ❏ On windows we don't have signals, but 9 (SIGKILL) calls `TerminateProcess`
  - ❏ Often the child process dies on own

# ❏ 'Window' mode

- ❏ ❏CMD on windows supports specifying initial window mode
  - ❏ Hidden, Maximized ...
  - ❏ Done via a nested right argument
- ❏ Exactly the same thing
  - ❏ Variant option because right argument means something else

# ☐ 'Output' redirections

- Which output streams to collect, and what to do with the data
- Default is to collect stdout as text, and redirect stderr to stdout
- 2-column matrix or vector of streamId-target pairs
- Quite a few possible targets

# ⚙️ 'Output' redirections

- ❖ ('Stream' StreamNum) or just StreamNum
- ❖ ('File' TieNum) or just TieNum
- ❖ ('File' FilePath)
- ❖ ('Array' DataType)
- ❖ ('Array' TextEncoding) or just 'Array'
- ❖ 'Null'

# ❏ 'Output' redirections

- ❖ ('Callback' Fn EncodingOrType)
- ❖ ('Callback' Fn)
  - ❖ Whenever data is produced, run a callback function to deal with it
- ❖ Fn is name of function, or (FnName LeftArg)
- ❖ Callback passed a namespace with information
- ❖ Data does not show up in the ❏SHELL result value

# ⌘ 'Input' redirections

- Controls what the child process sees when it tries to read from its input streams
- By default, only stdin is setup, such that the child process gets no data
- 2-column matrix or vector of streamId-source pairs



# 'Input' redirections

- ('File' TieNum) or just TieNum
- ('File' FilePath)
- ('Array' Data Type)
- ('Array' TextData Encoding)
- ('Array' TextData Encoding Newline)
- 'Null'

# Input redirections

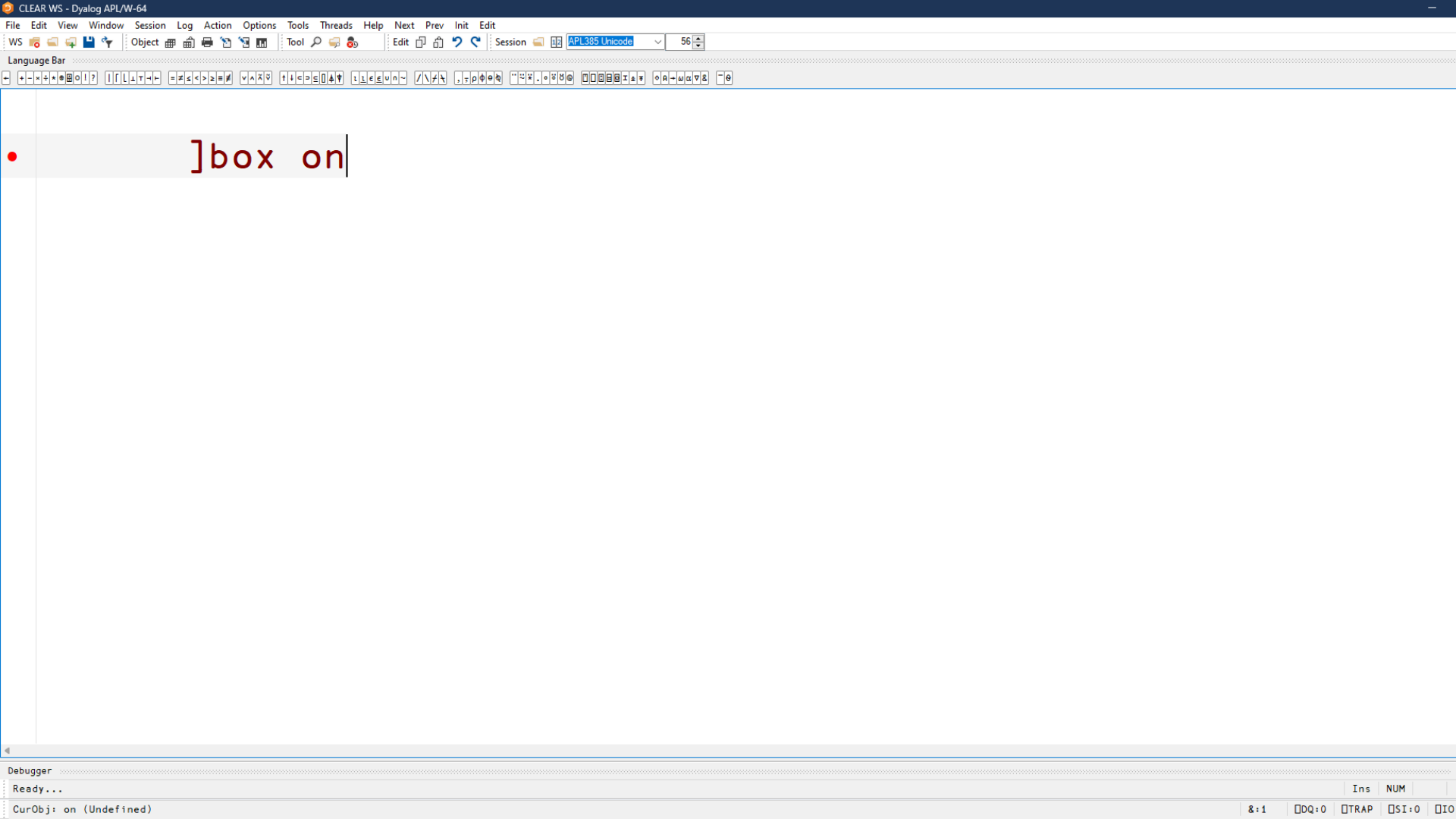
- ( 'Token' tokenNumber )
- All the other sources required all data to be specified before running `□SHELL`
- Send additional data:
  - ( 'Array' ...) `□TPUT tokenNumber`
- Tell `□SHELL` that no more data will appear:
  - `□TPUT tokenNumber`

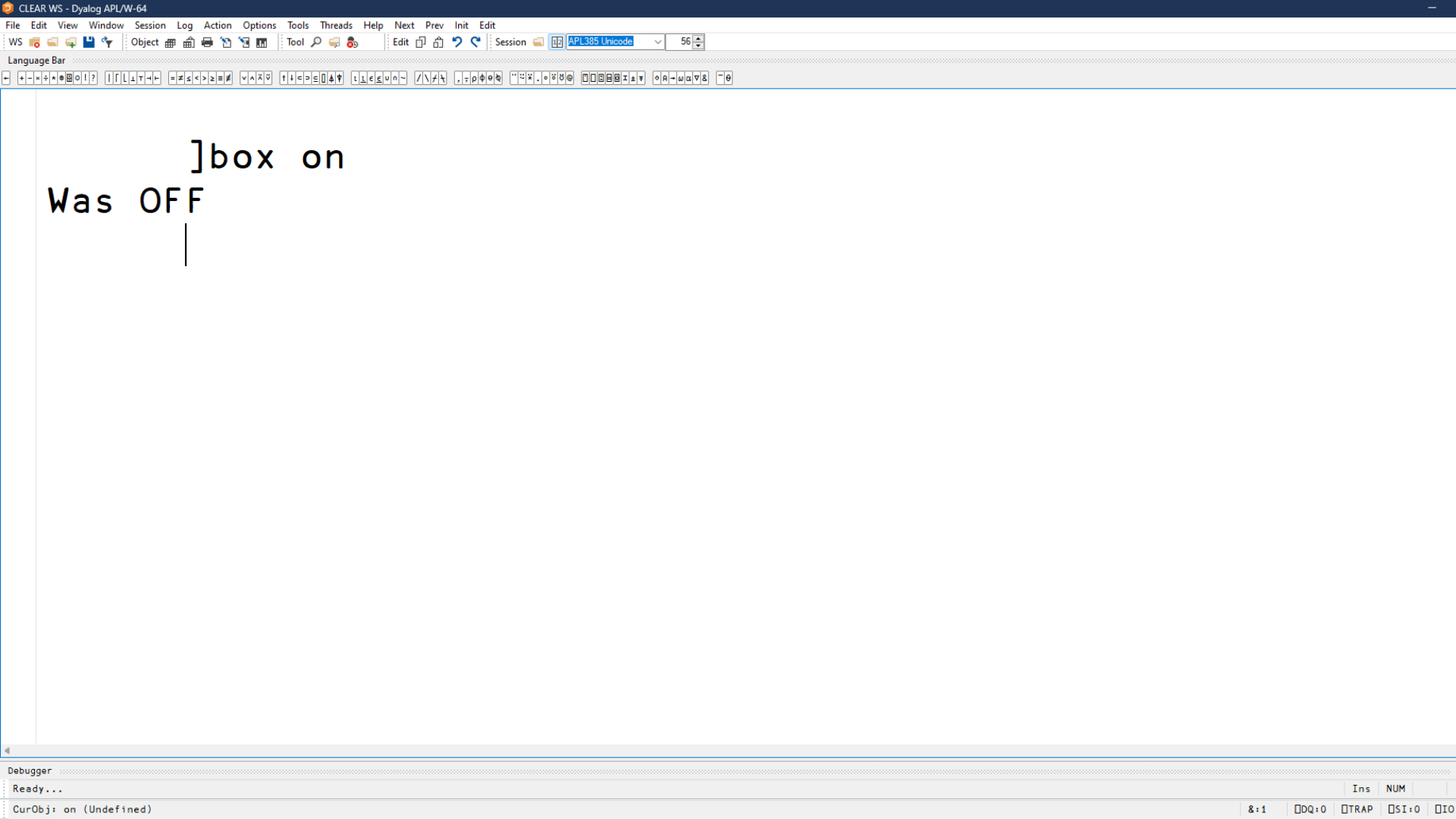
# Default redirections

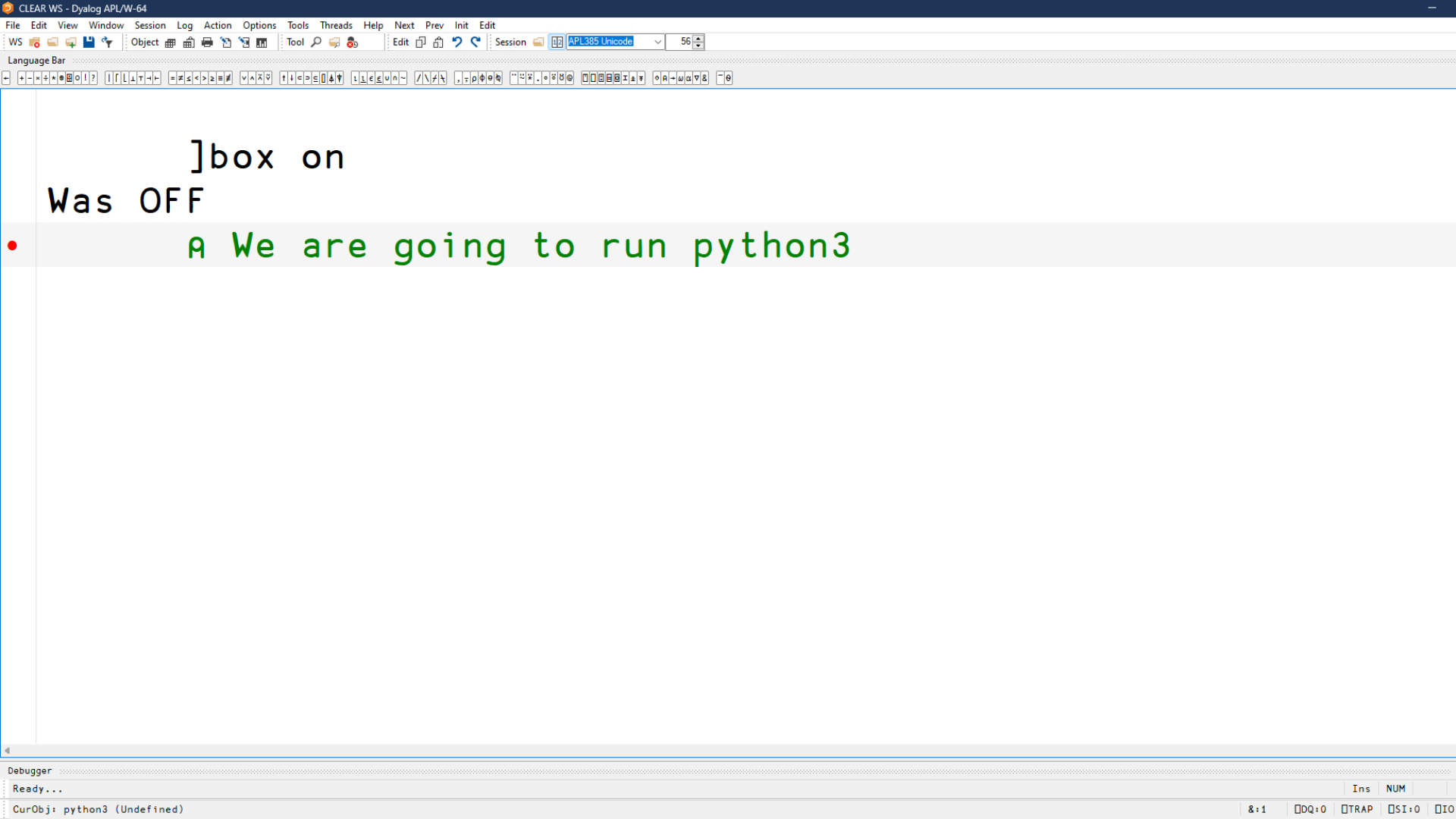
- ❖ `□SHELL` always sets up redirections for the three standard streams if not explicitly setup
- ❖ Input:
  - ❖ `0 'Null'`
- ❖ Output:
  - ❖ `2 ('Stream' 1)`
  - ❖ `1 'Array'`

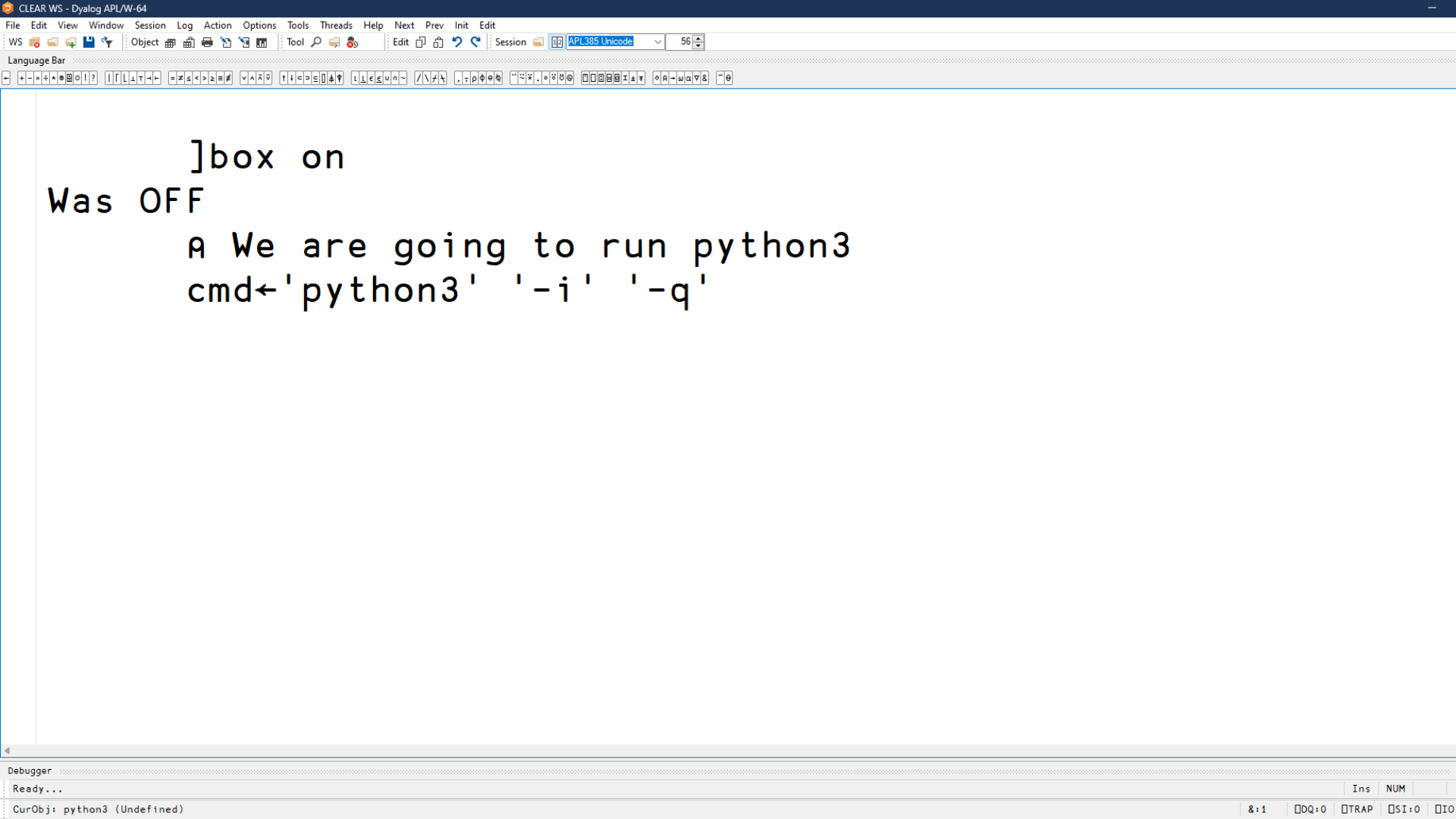
# Demo

- How to use `□SHELL` to run `python3`
- Some variant options will be used

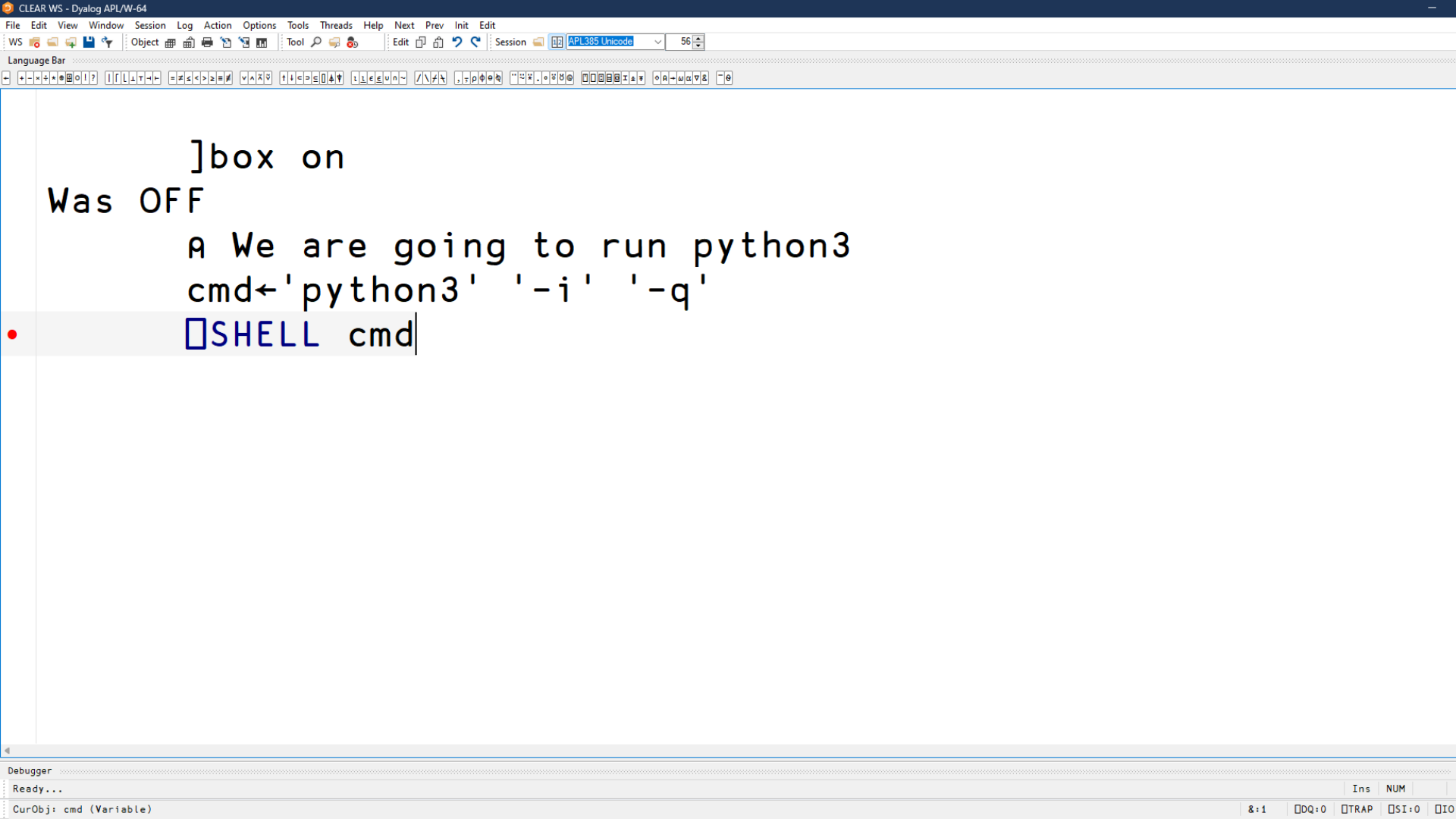


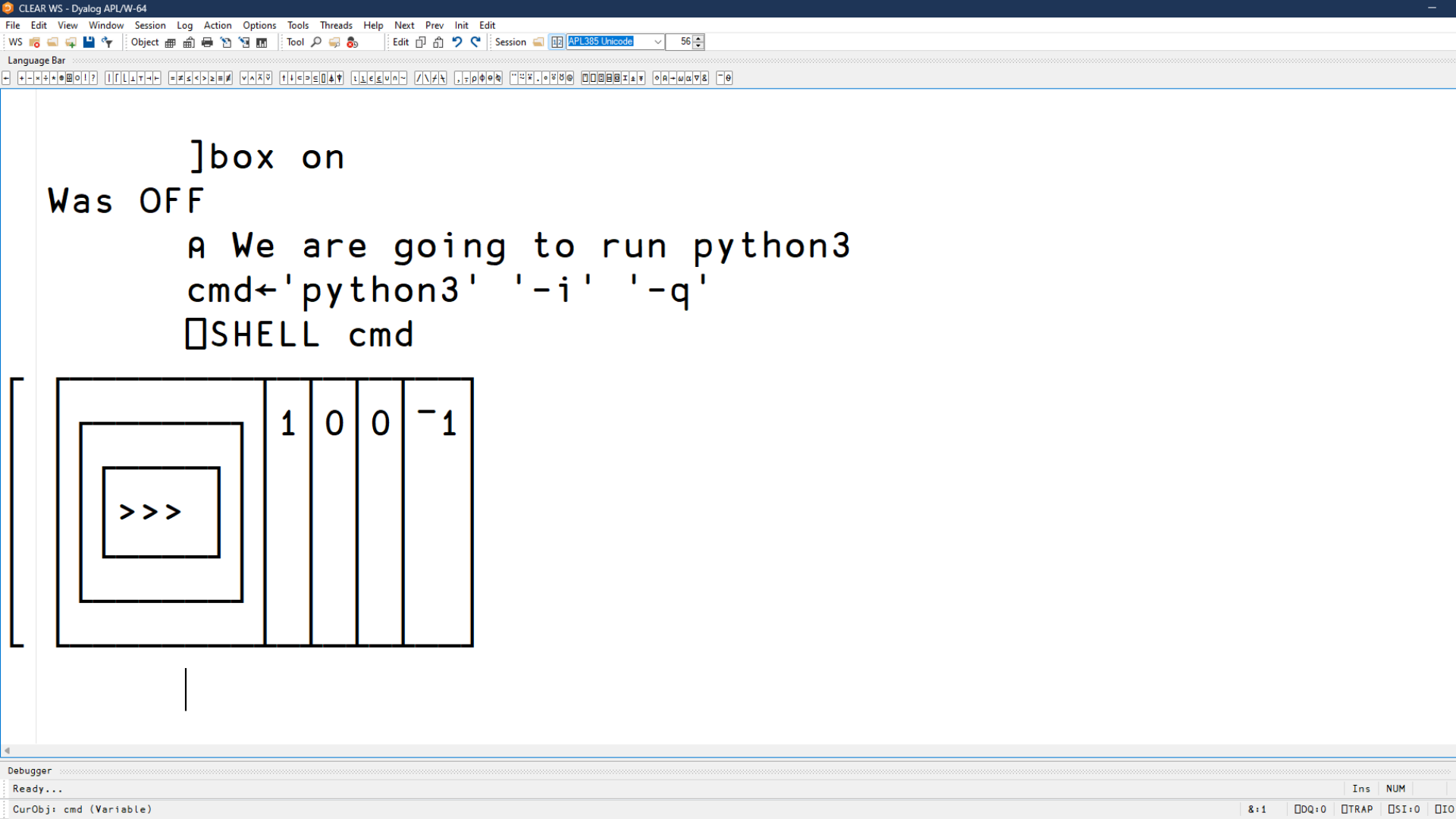


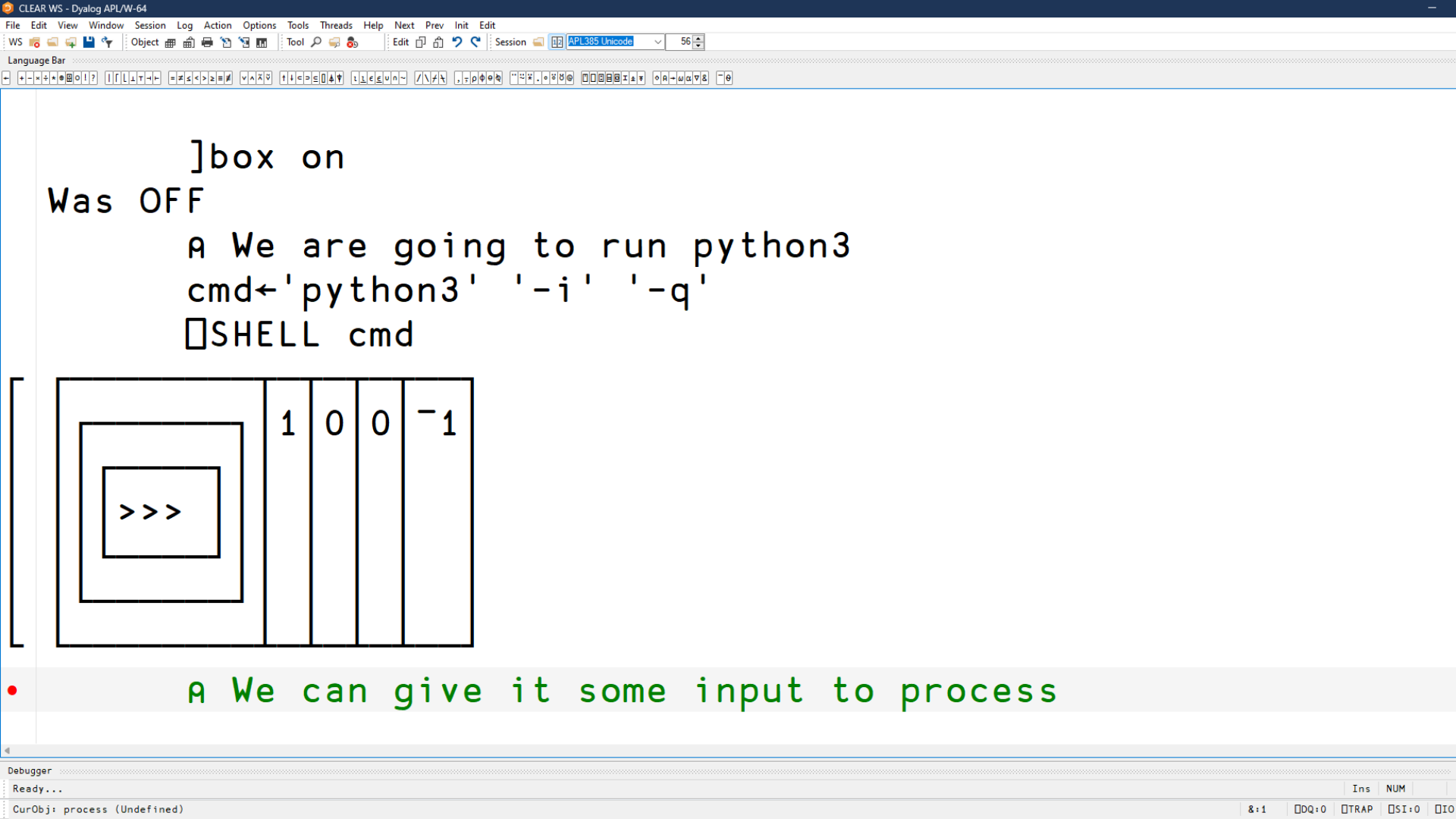


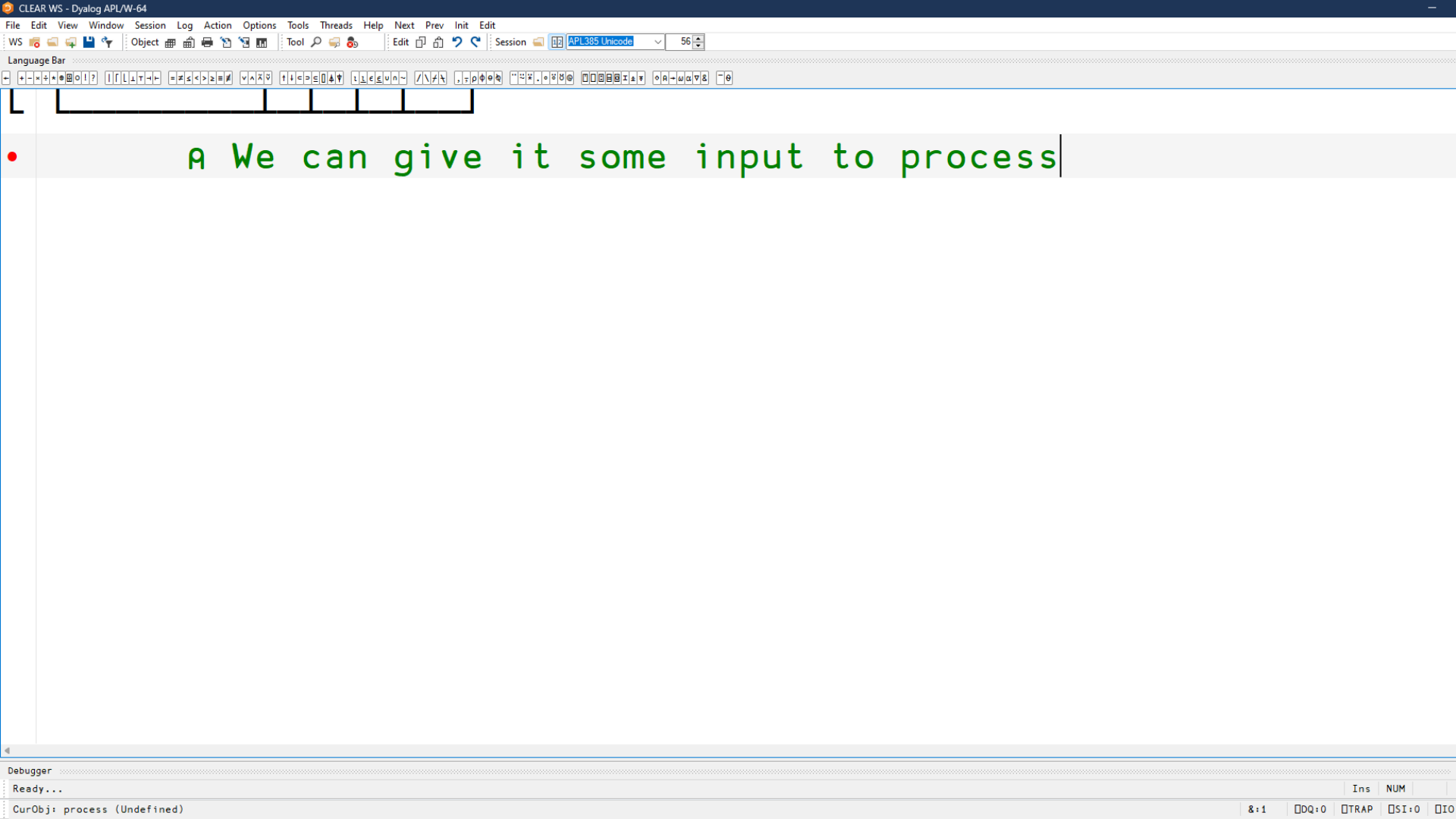




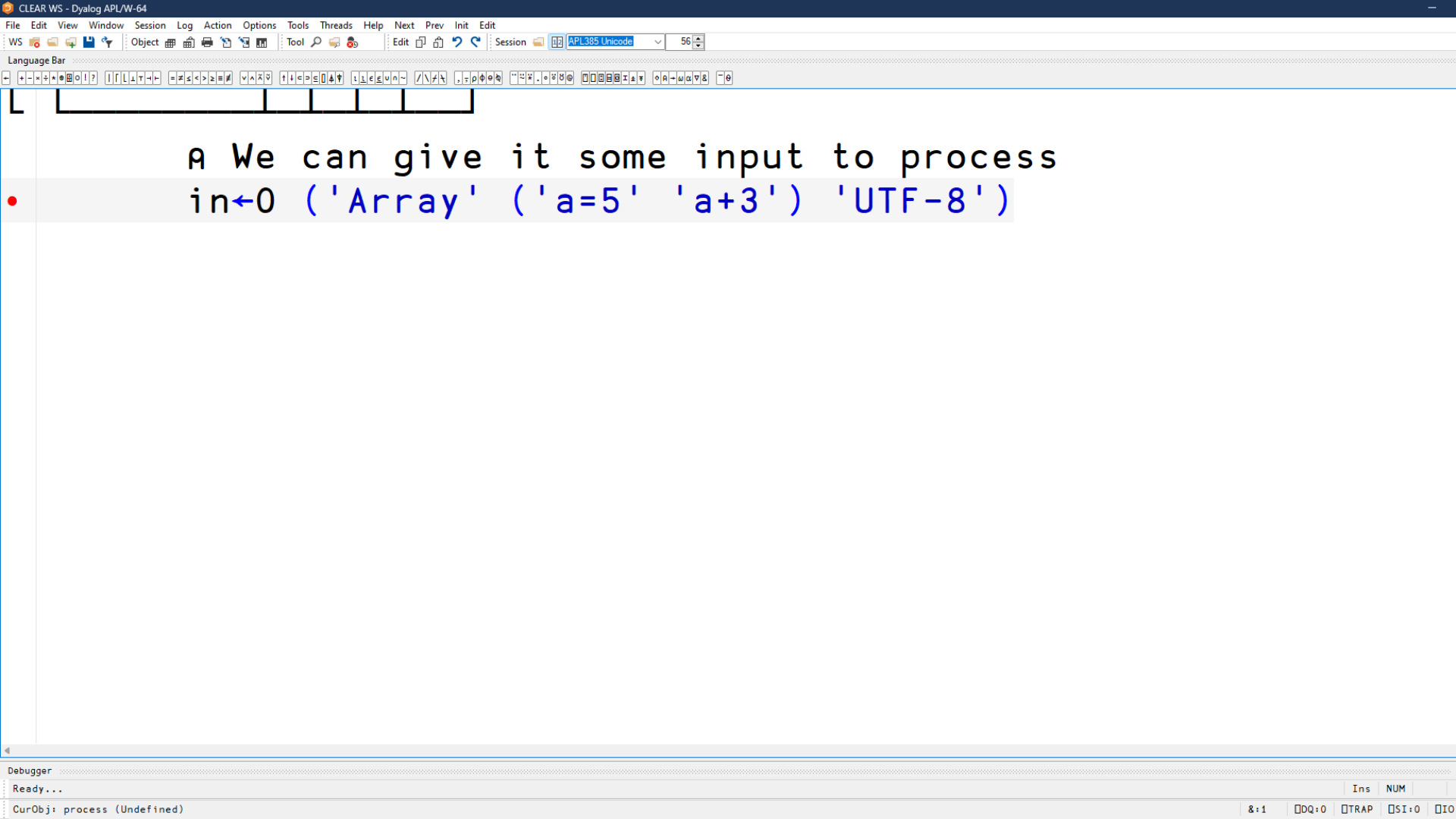


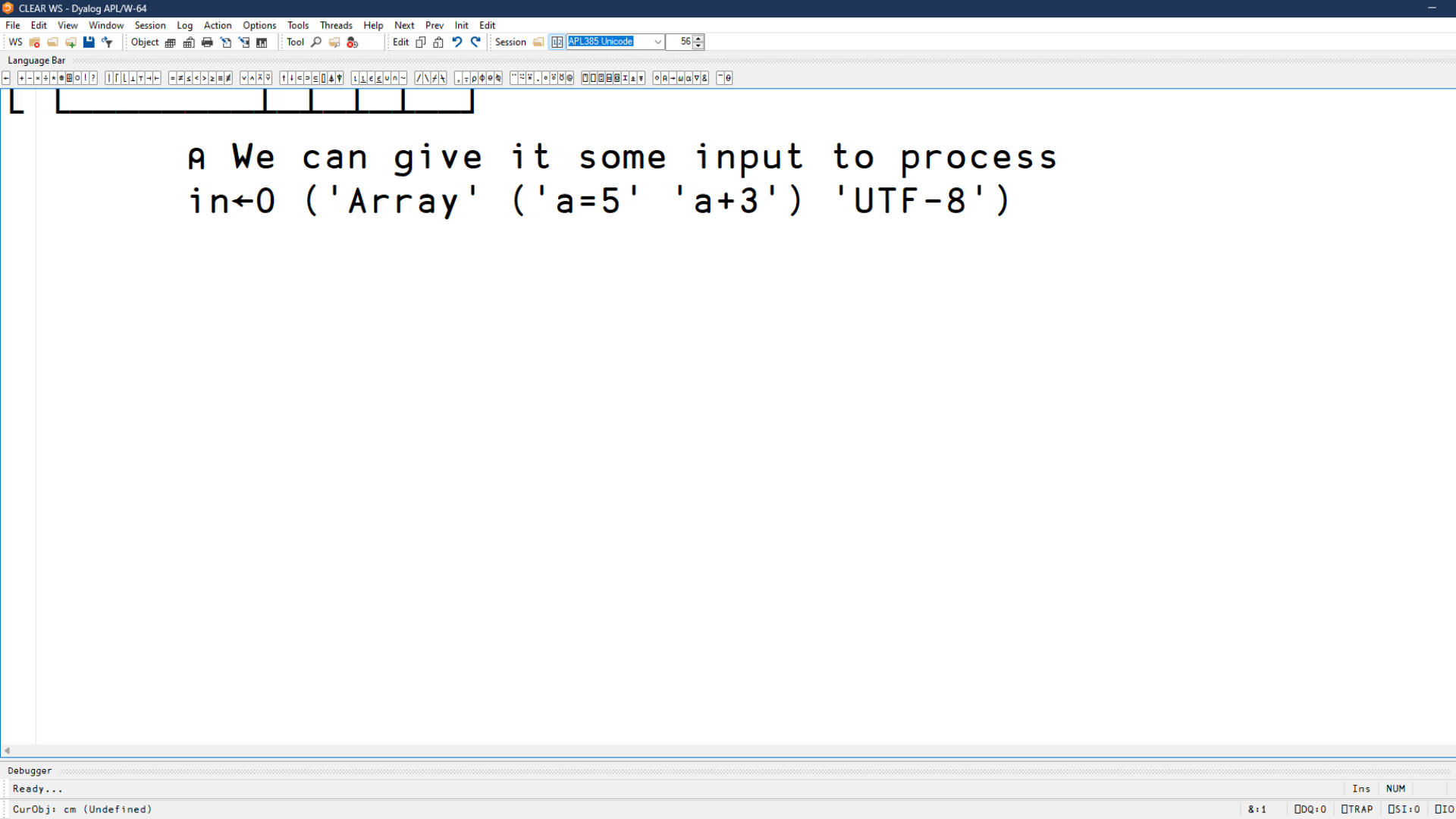


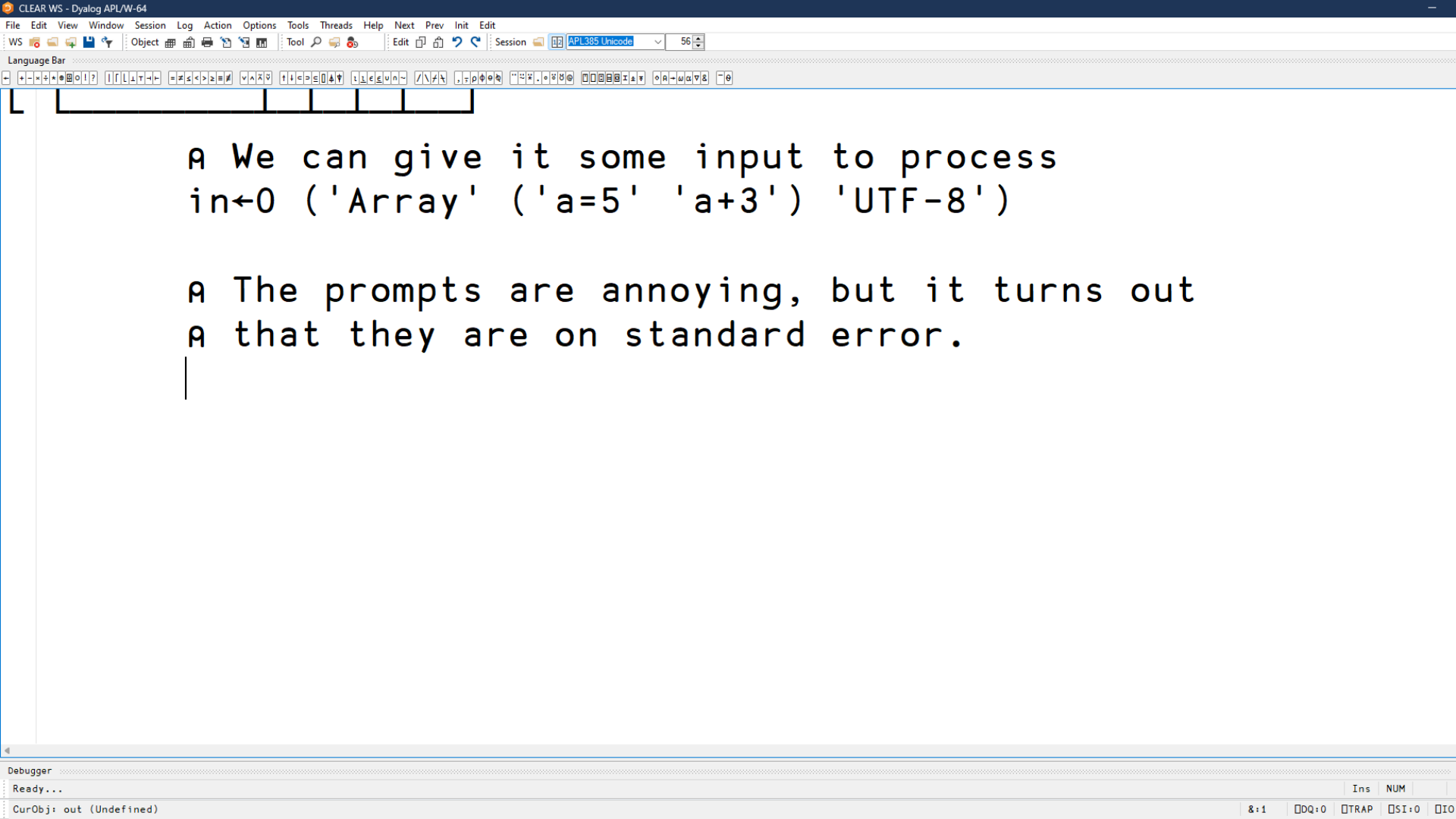


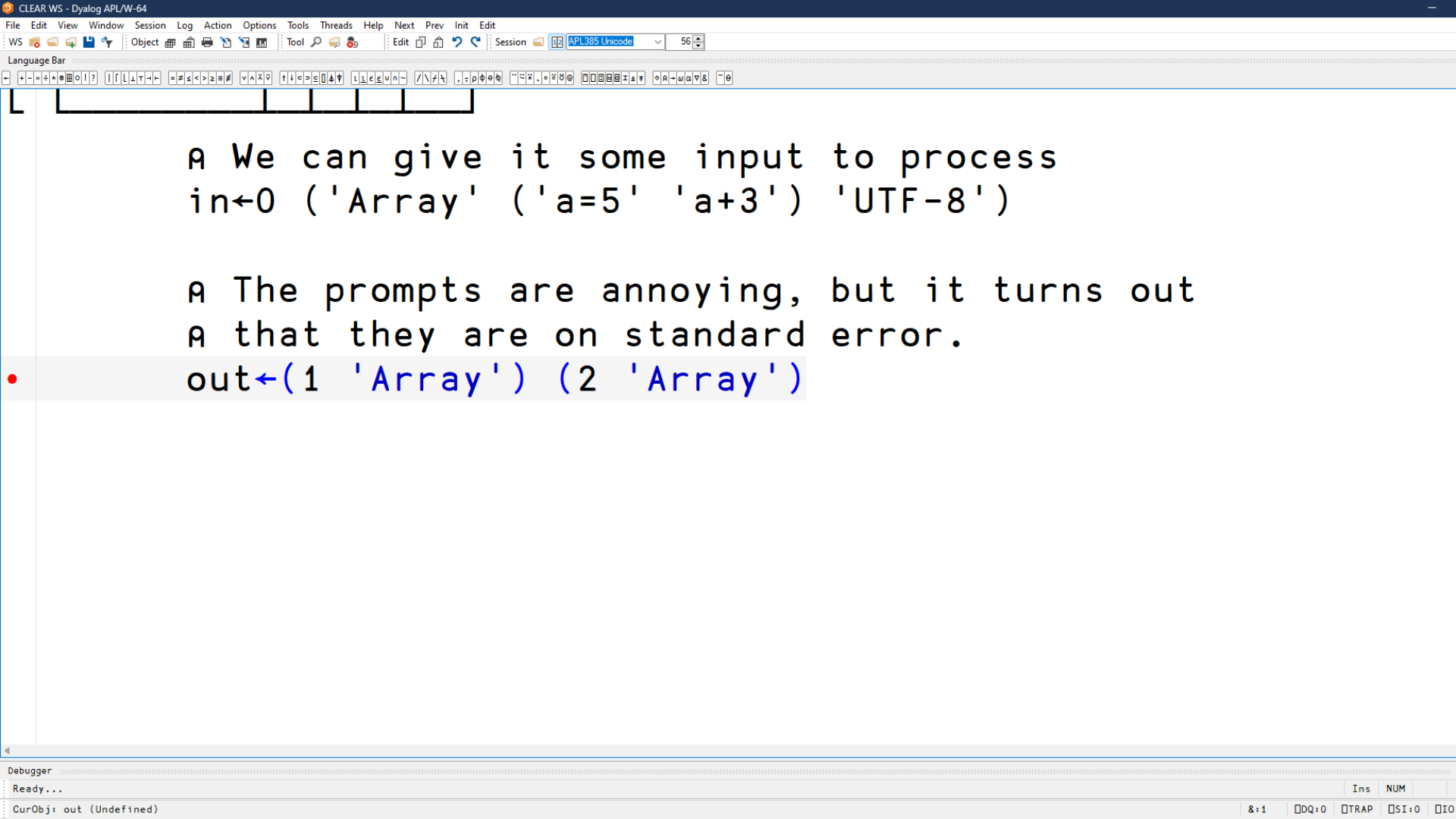


⌵ We can give it some input to process

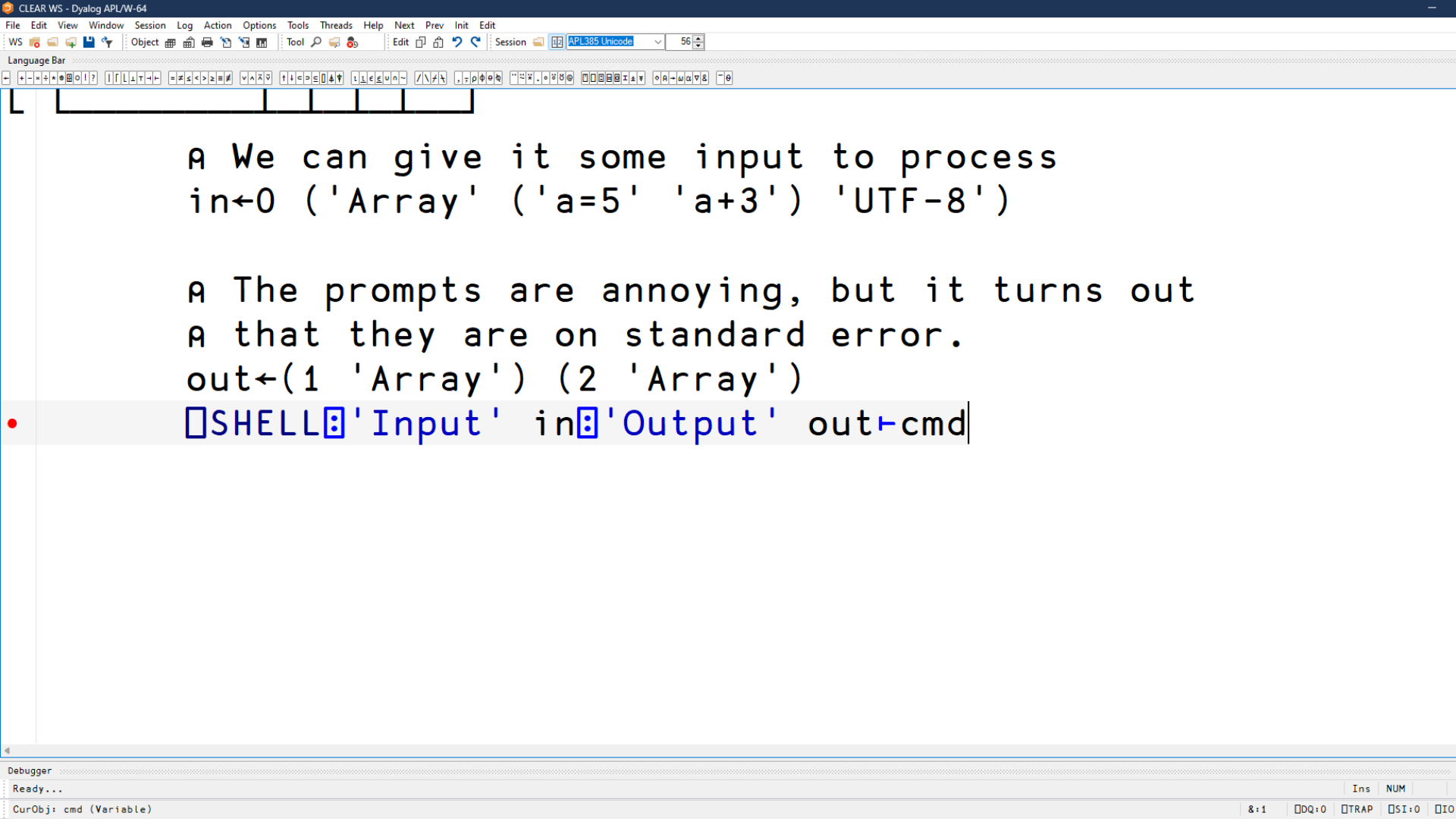


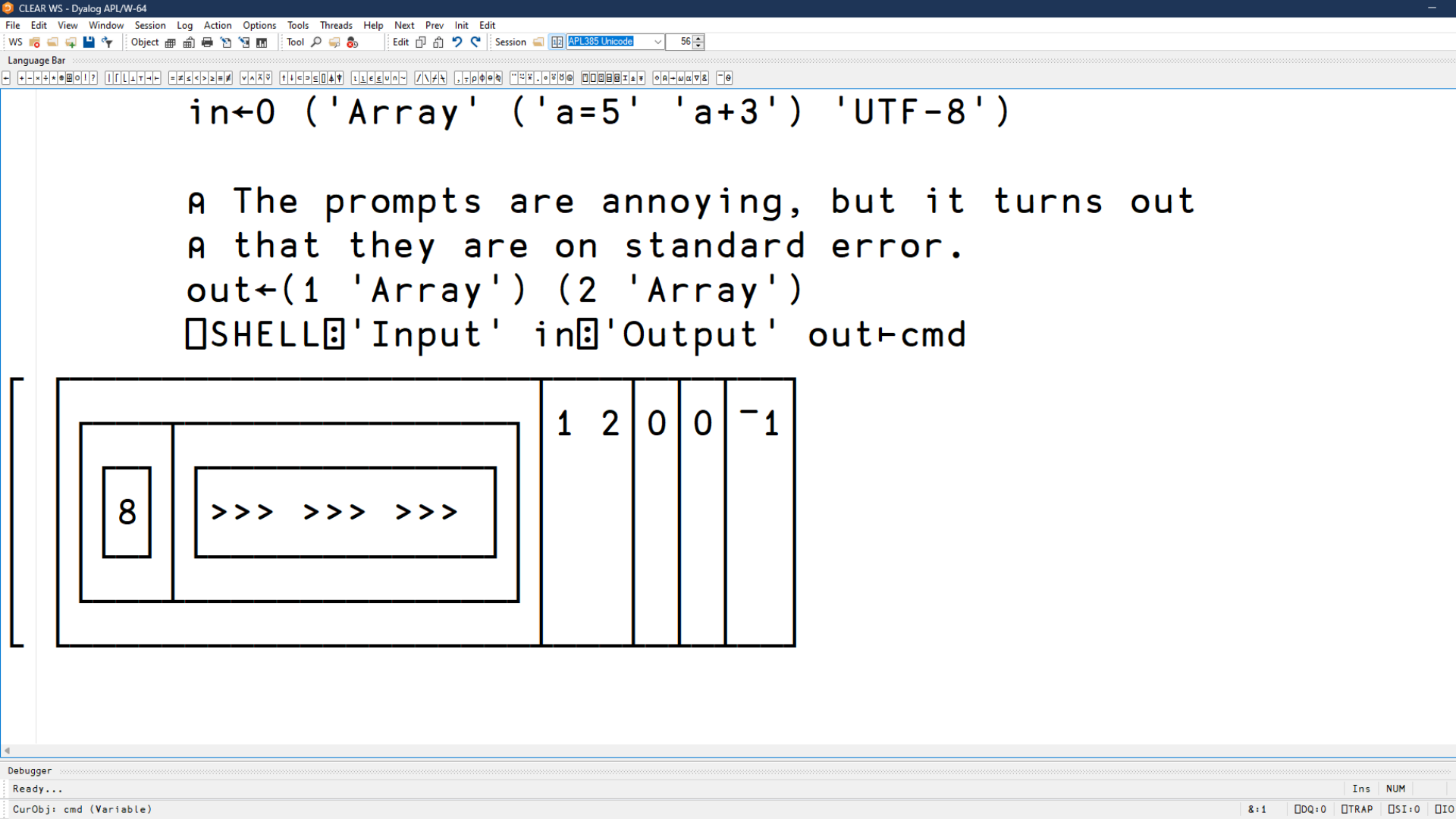


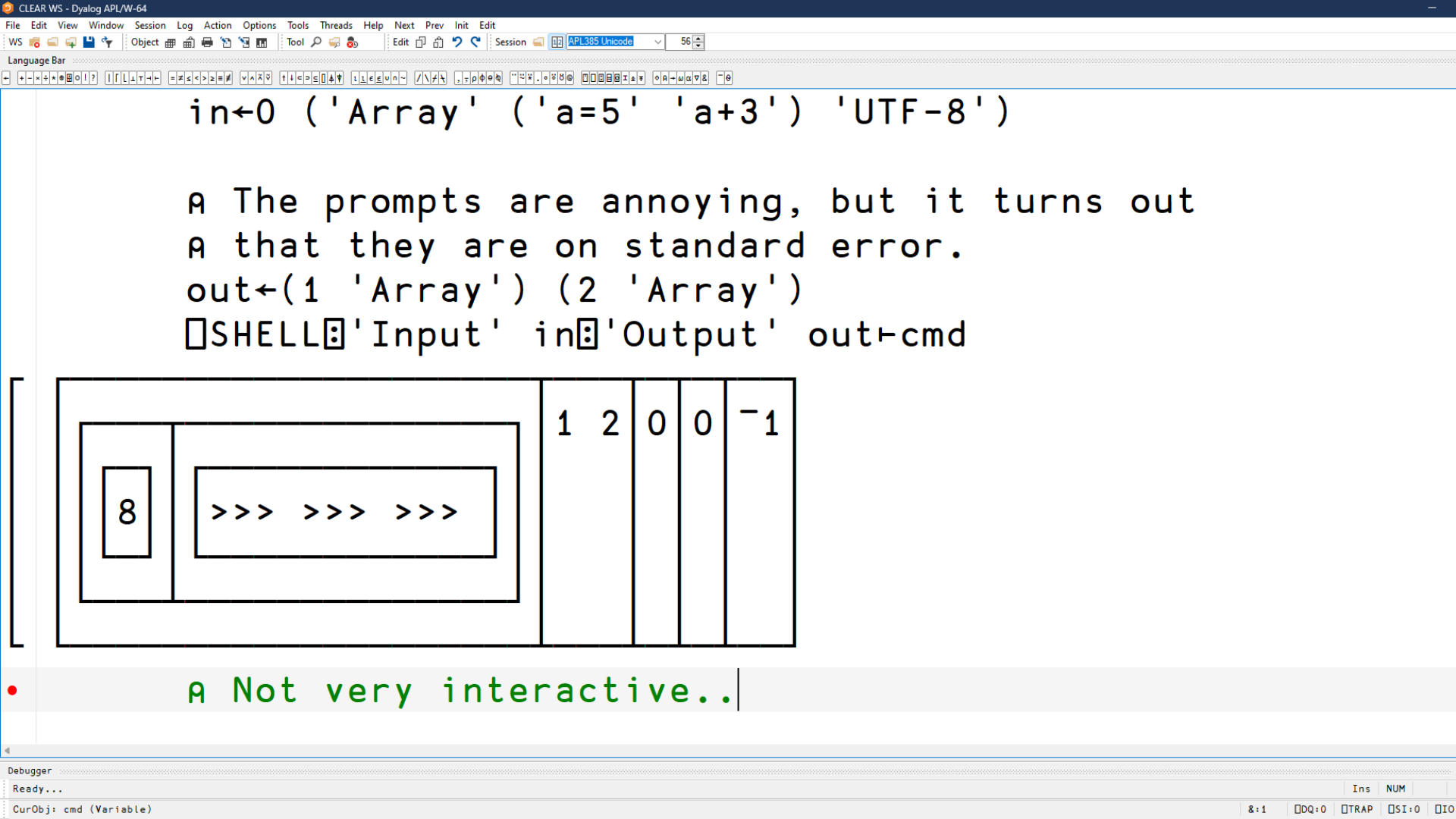




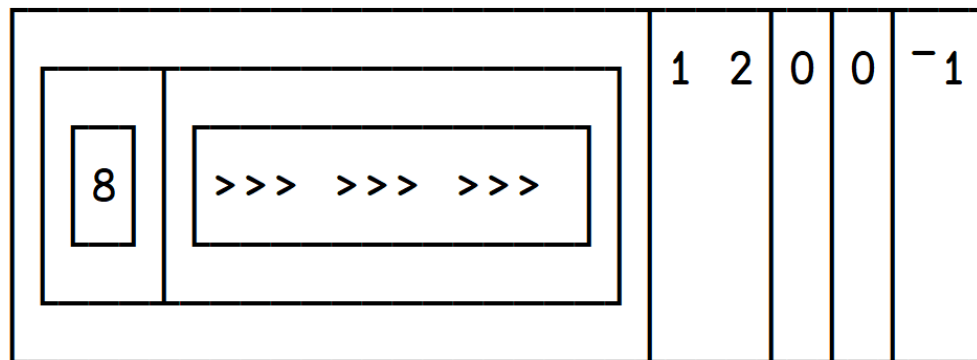








```
A The prompts are annoying, but it turns out
A that they are on standard error.
out←(1 'Array') (2 'Array')
□SHELL□'Input' in□'Output' out←cmd
```

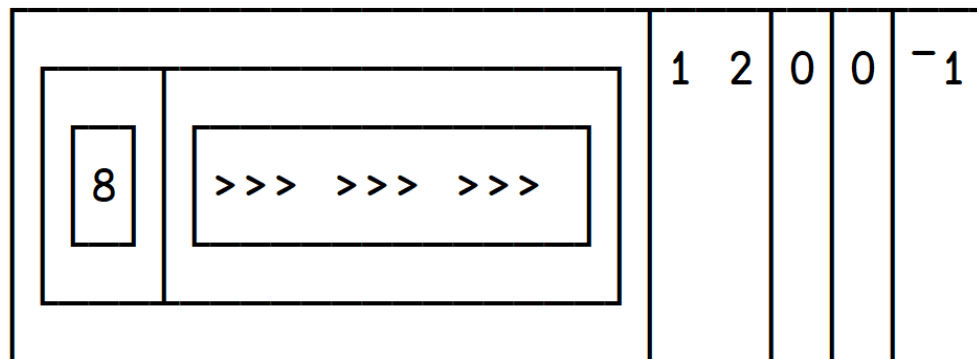


A Not very interactive..

A The prompts are annoying, but it turns out  
A that they are on standard error.

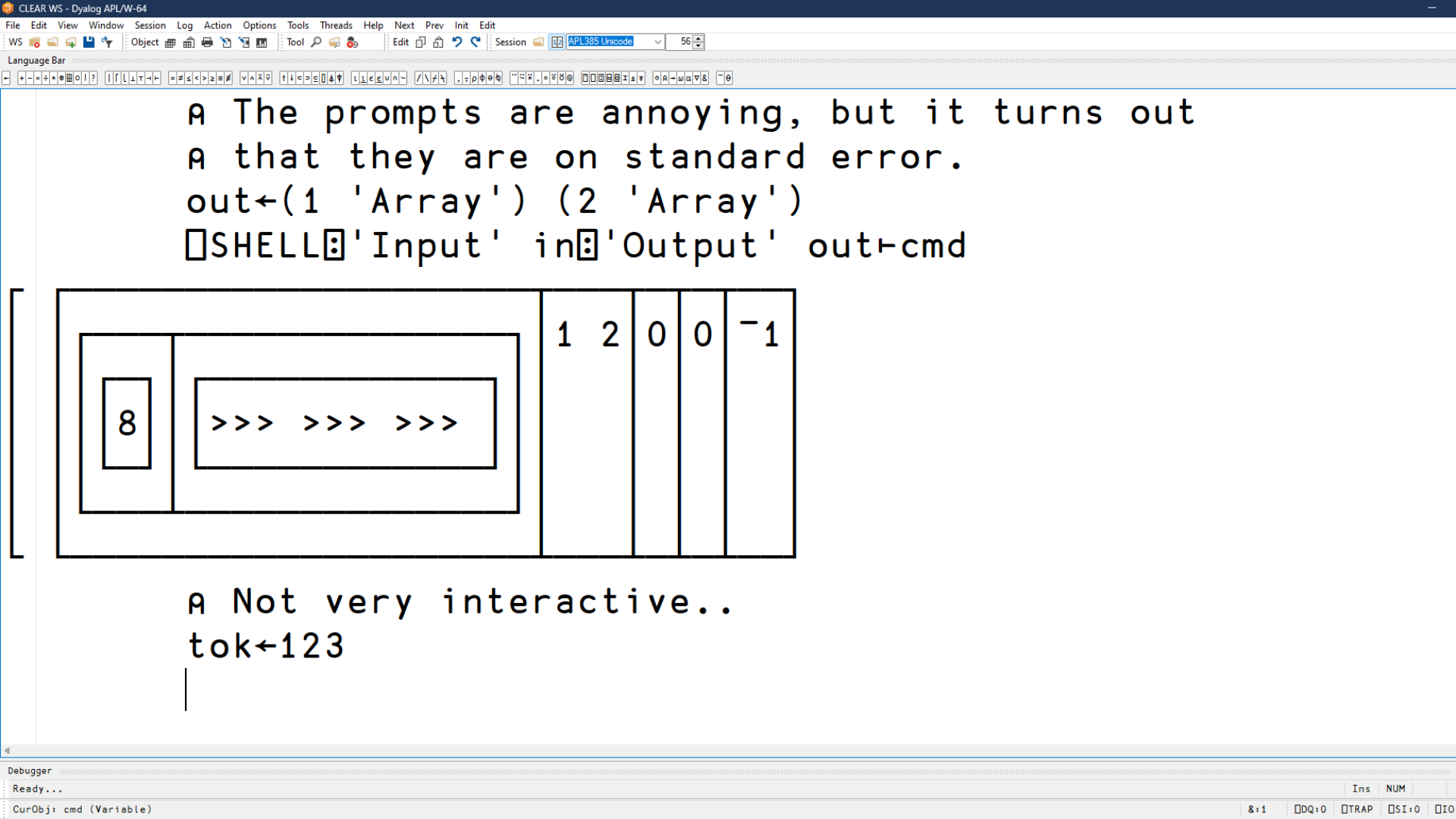
```
out←(1 'Array') (2 'Array')
```

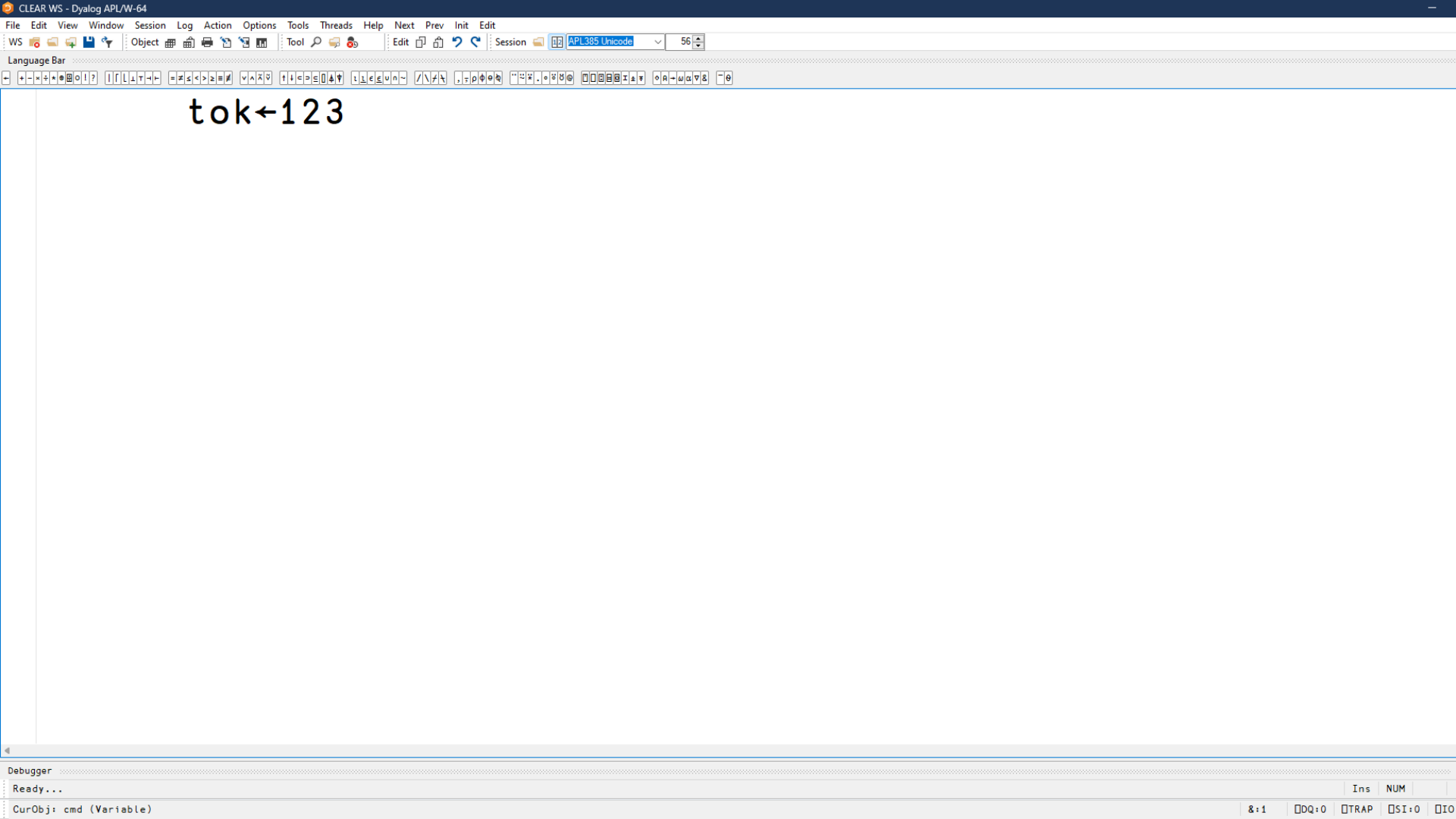
```
⊠SHELL⊠'Input' in⊠'Output' out←cmd
```

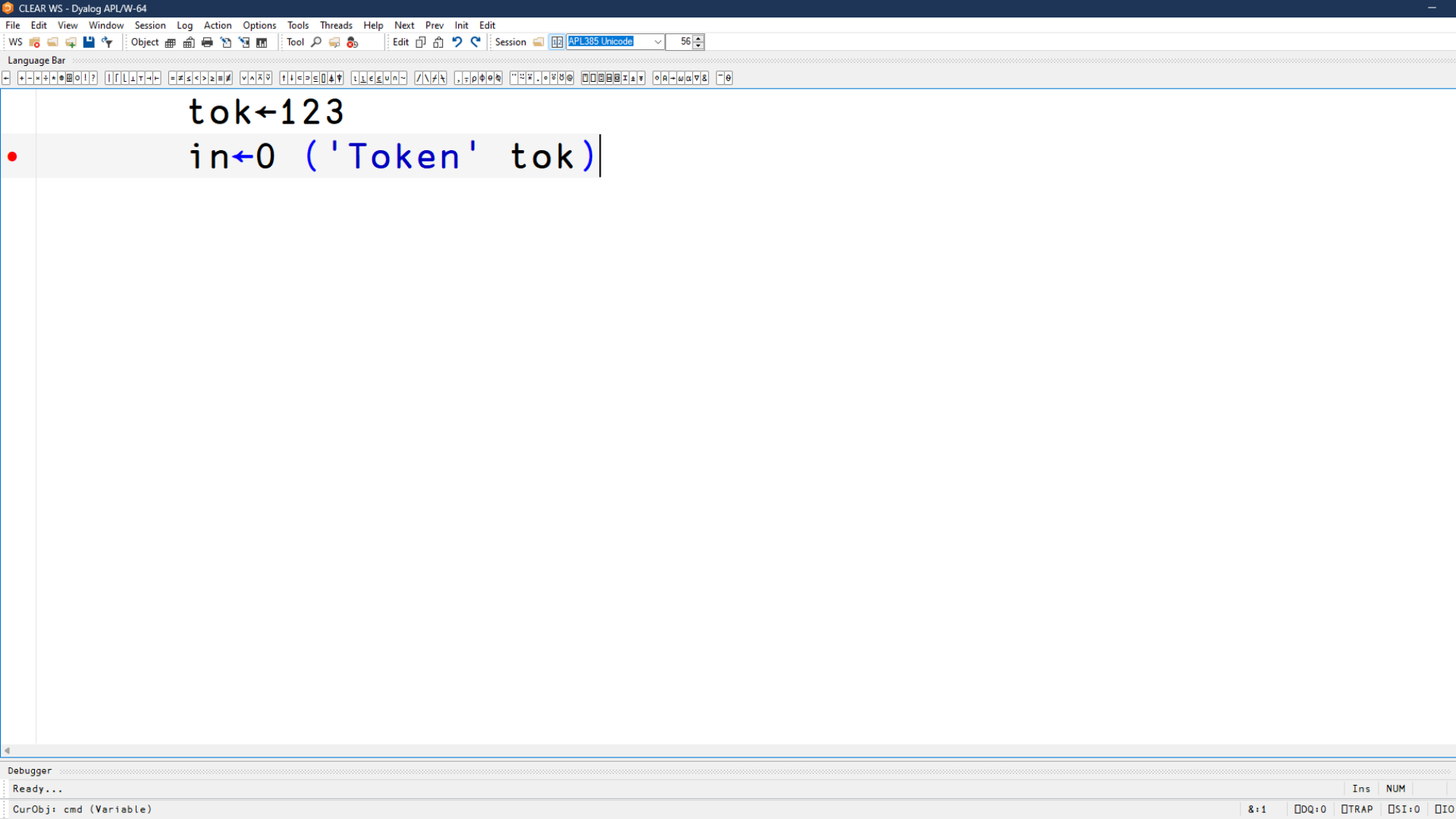


A Not very interactive..

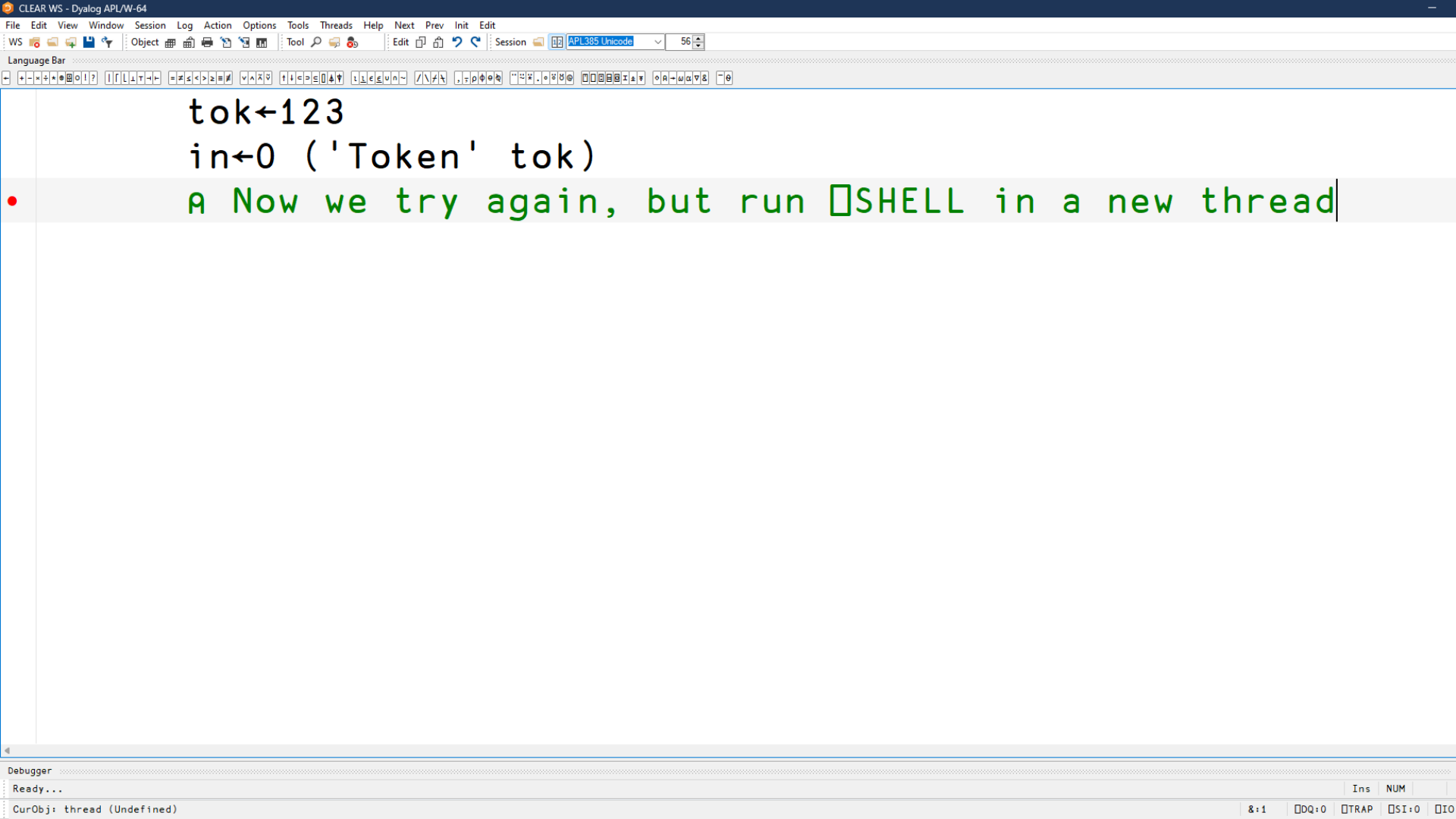
tok ← 123











```
tok←123
```

```
in←0 ('Token' tok)
```

```
⌘ Now we try again, but run ⌘SHELL in a new thread
```

Debugger

Ready...

CurObj: thread (Undefined)

Ins

NUM

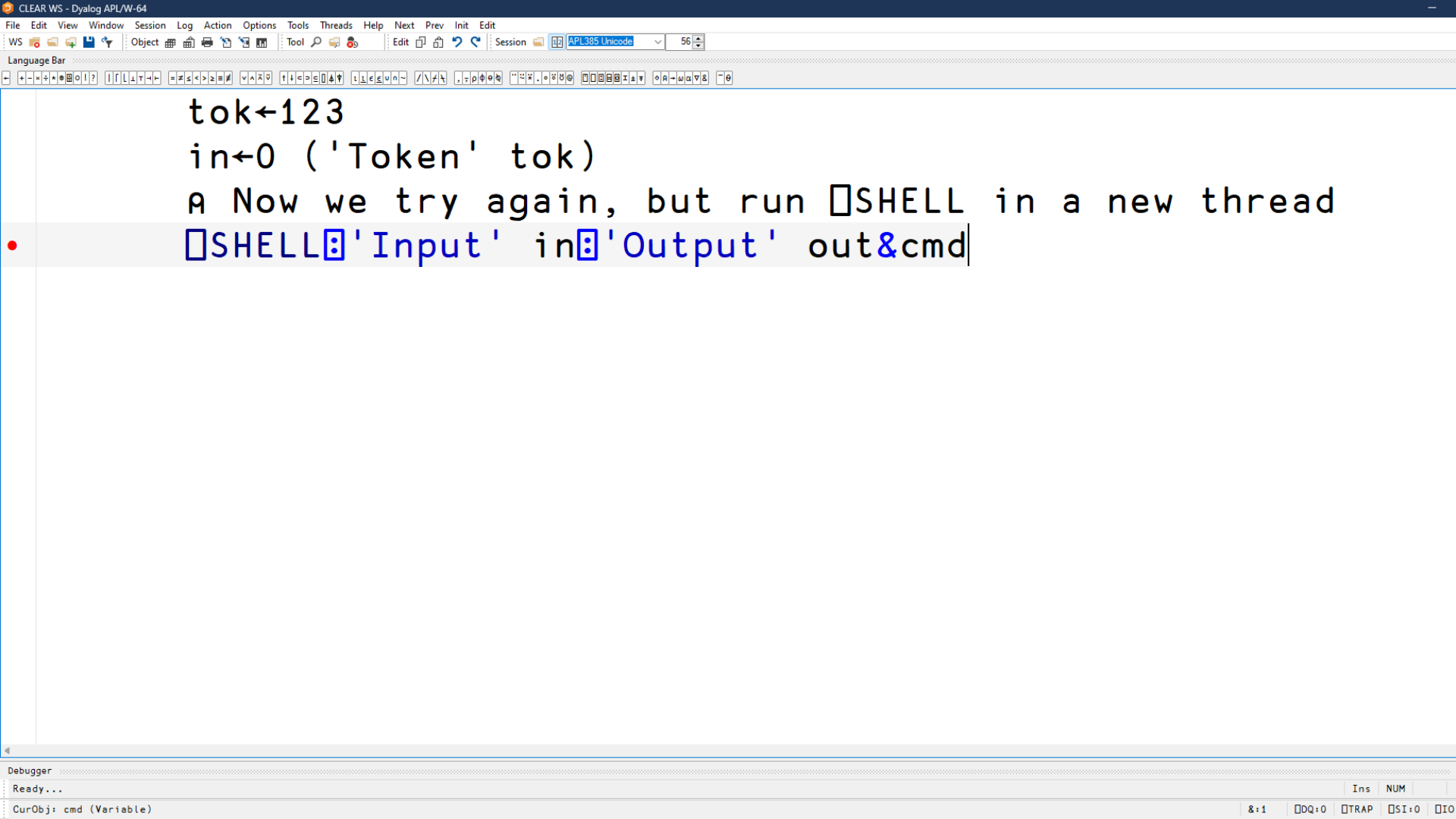
&:1

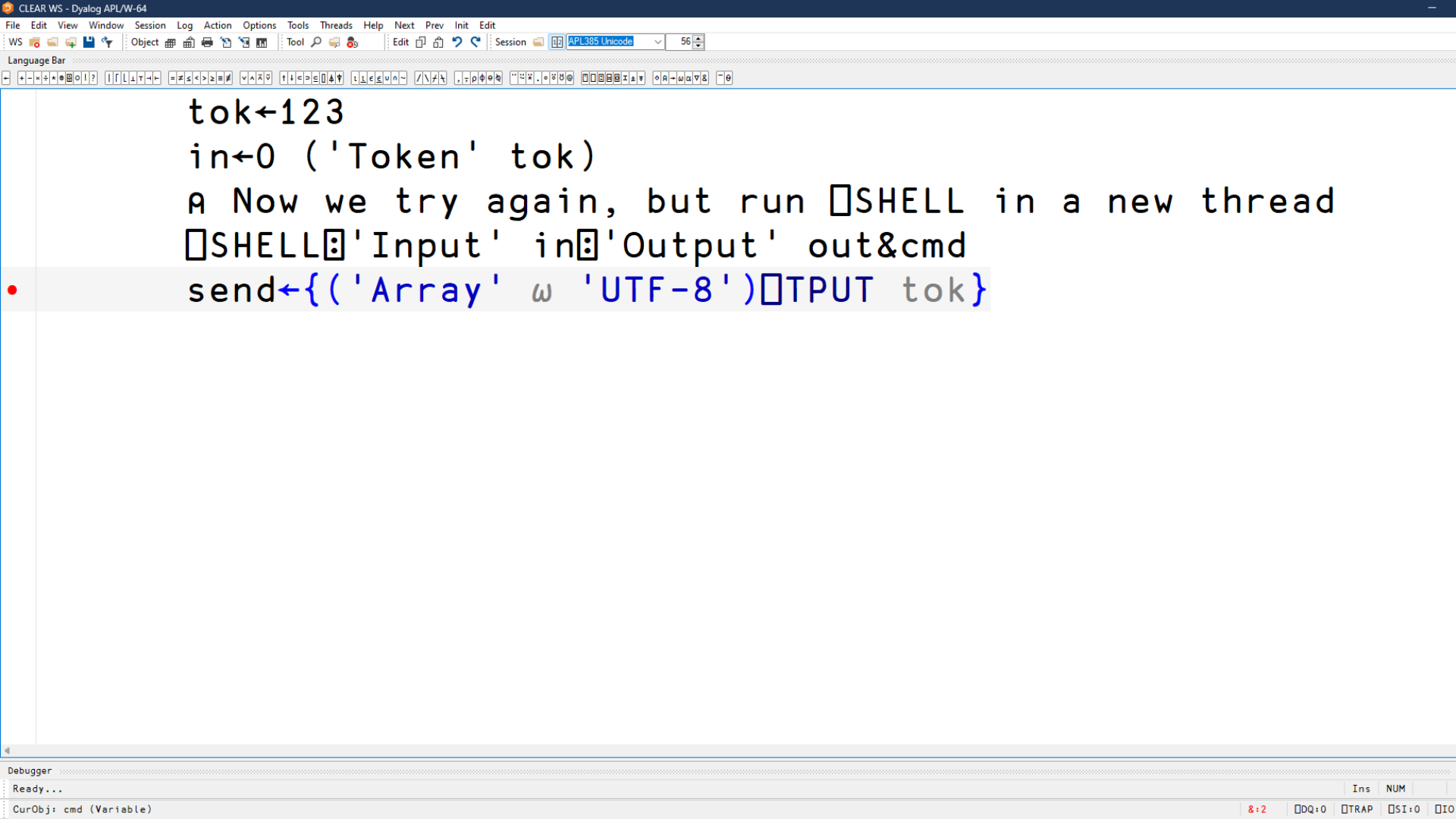
⌘DQ:0

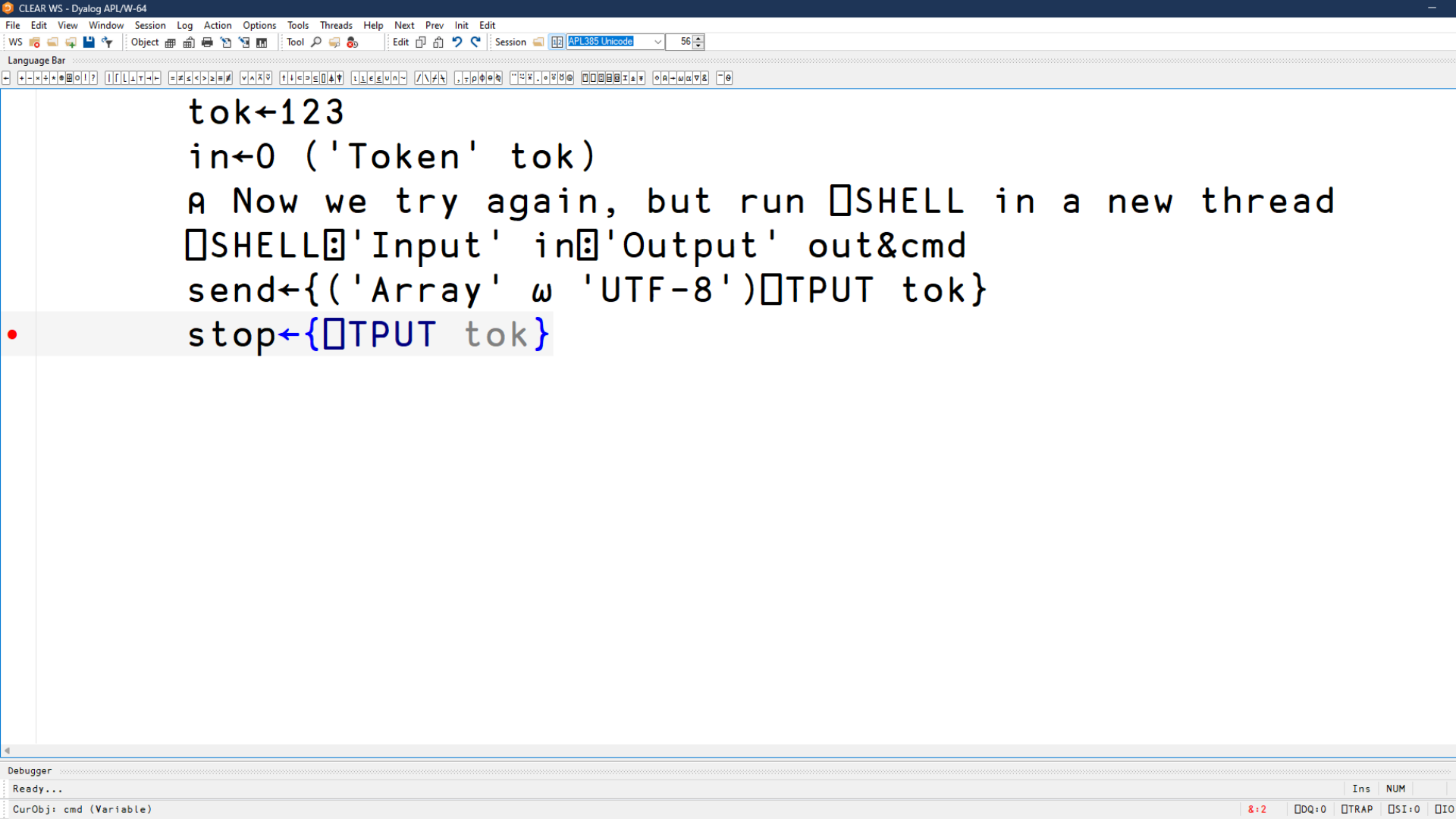
⌘TRAP

⌘SI:0

⌘IO







tok ← 123

```
in ← 0 ( 'Token' tok)
```

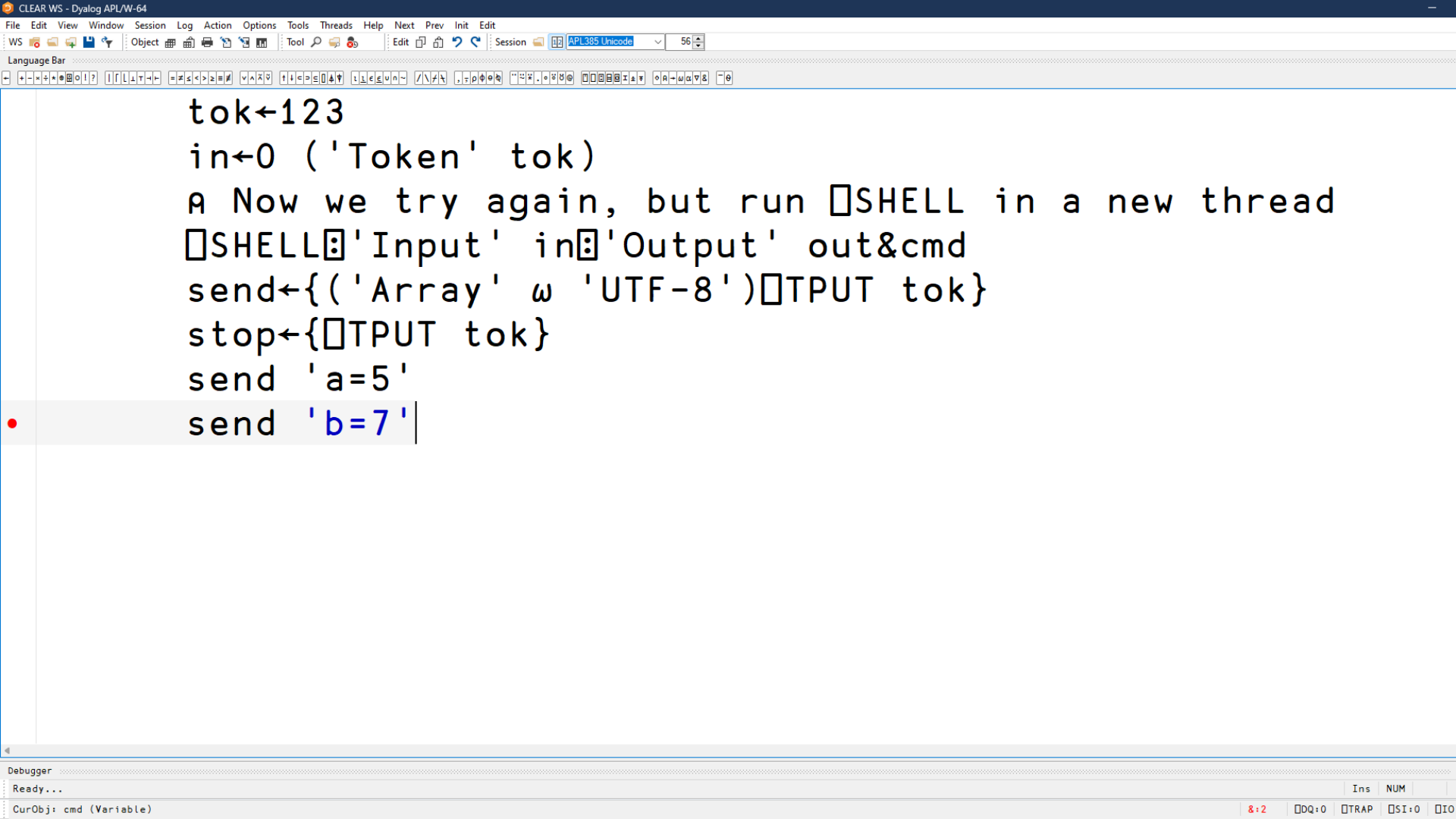
A Now we try again, but run `□SHELL` in a new thread

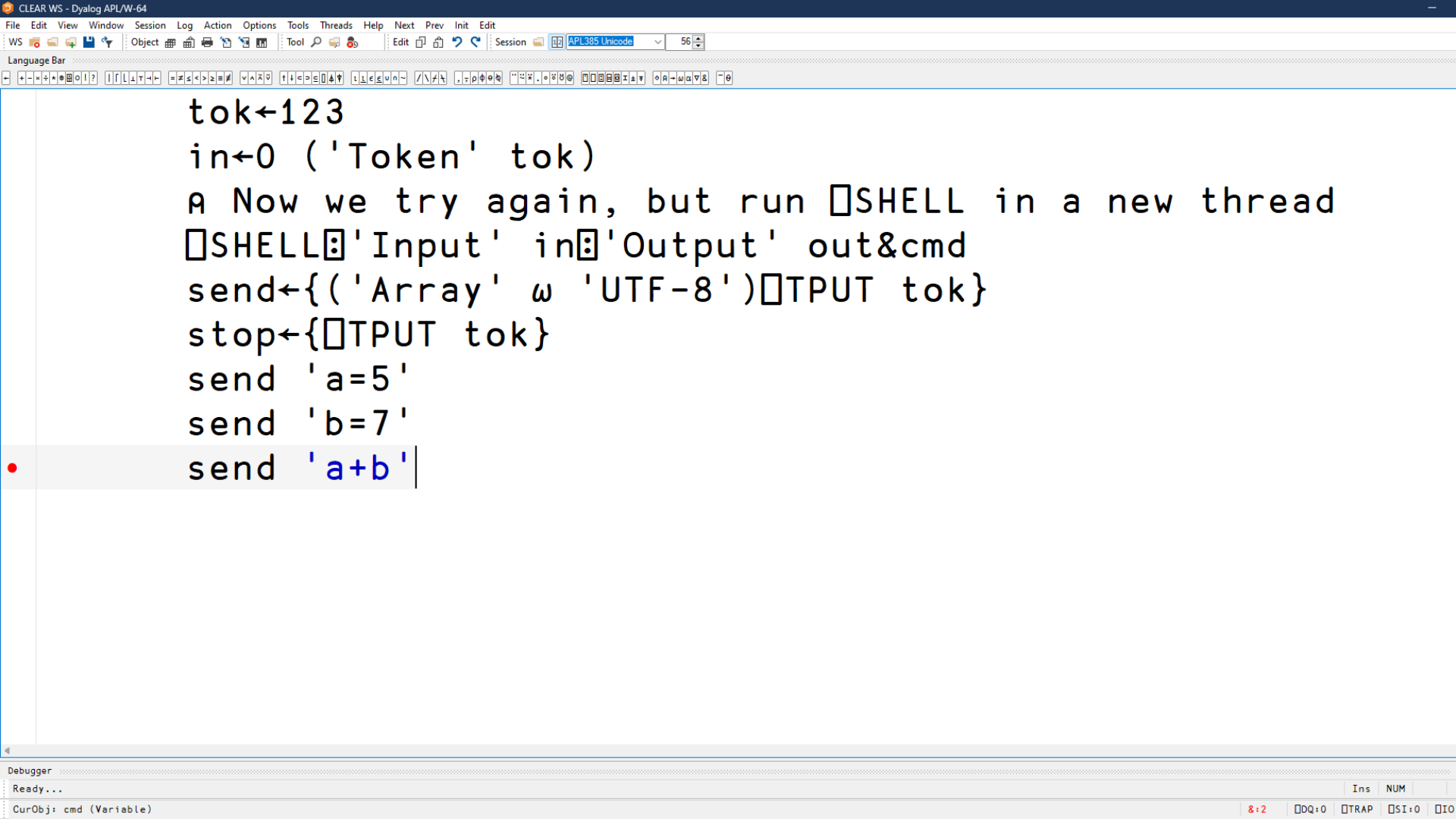
```
□SHELL@'Input' in@'Output' out&cmd
```

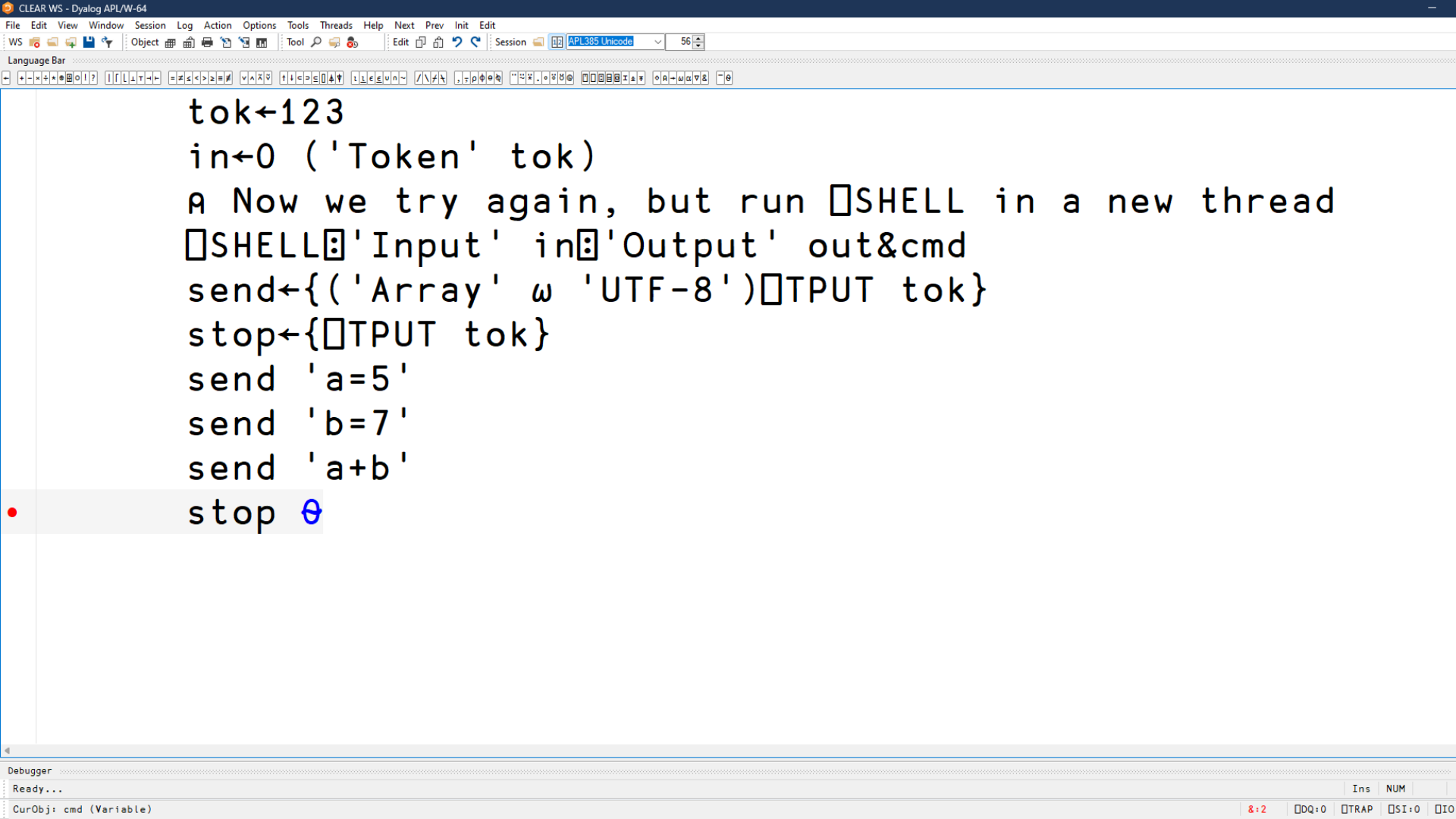
```
send←{('Array' w 'UTF-8')⊠TPUT tok}
```

```
stop ← {⊥ TPUT tok}
```

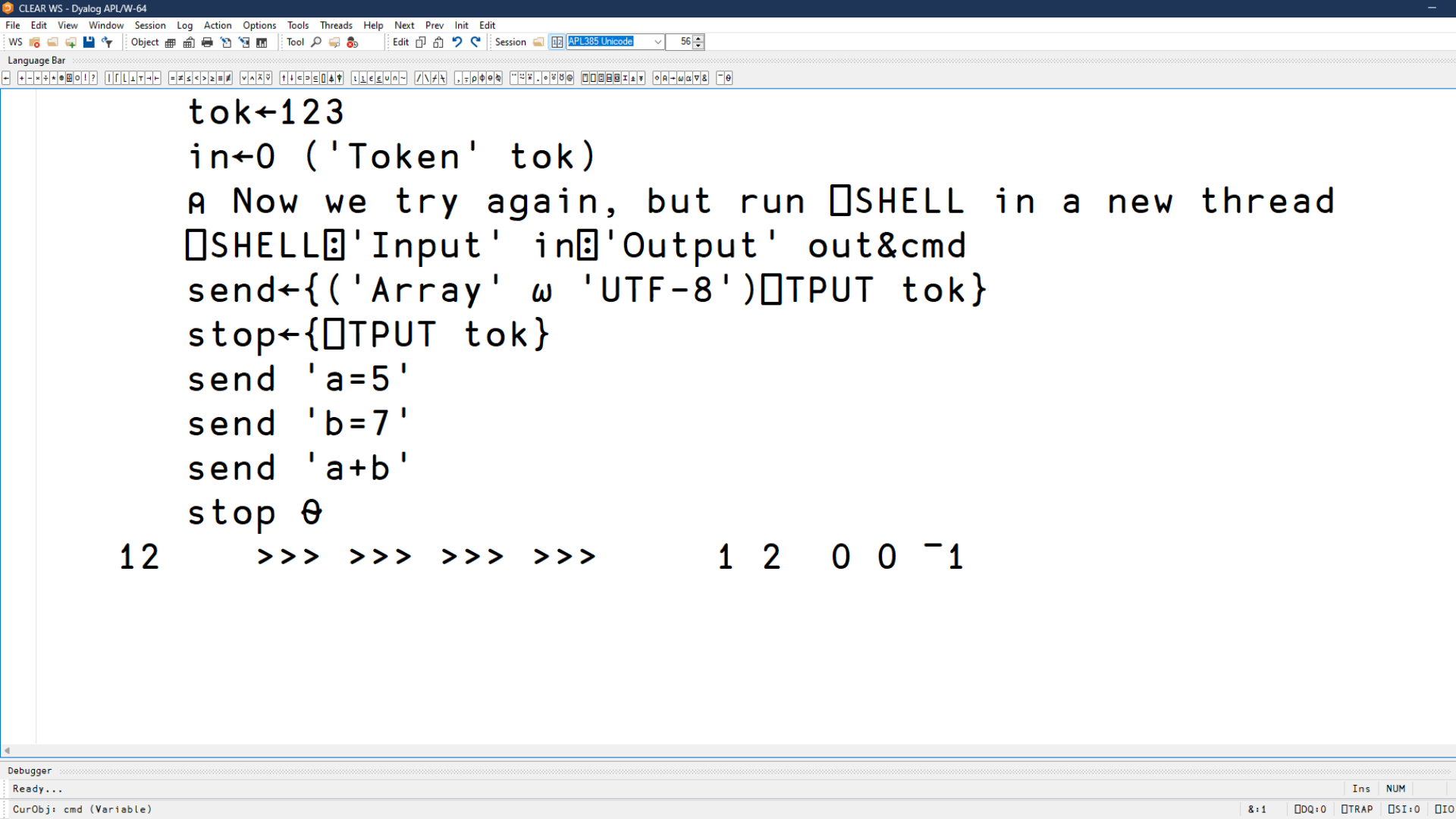
```
send 'a=5'
```

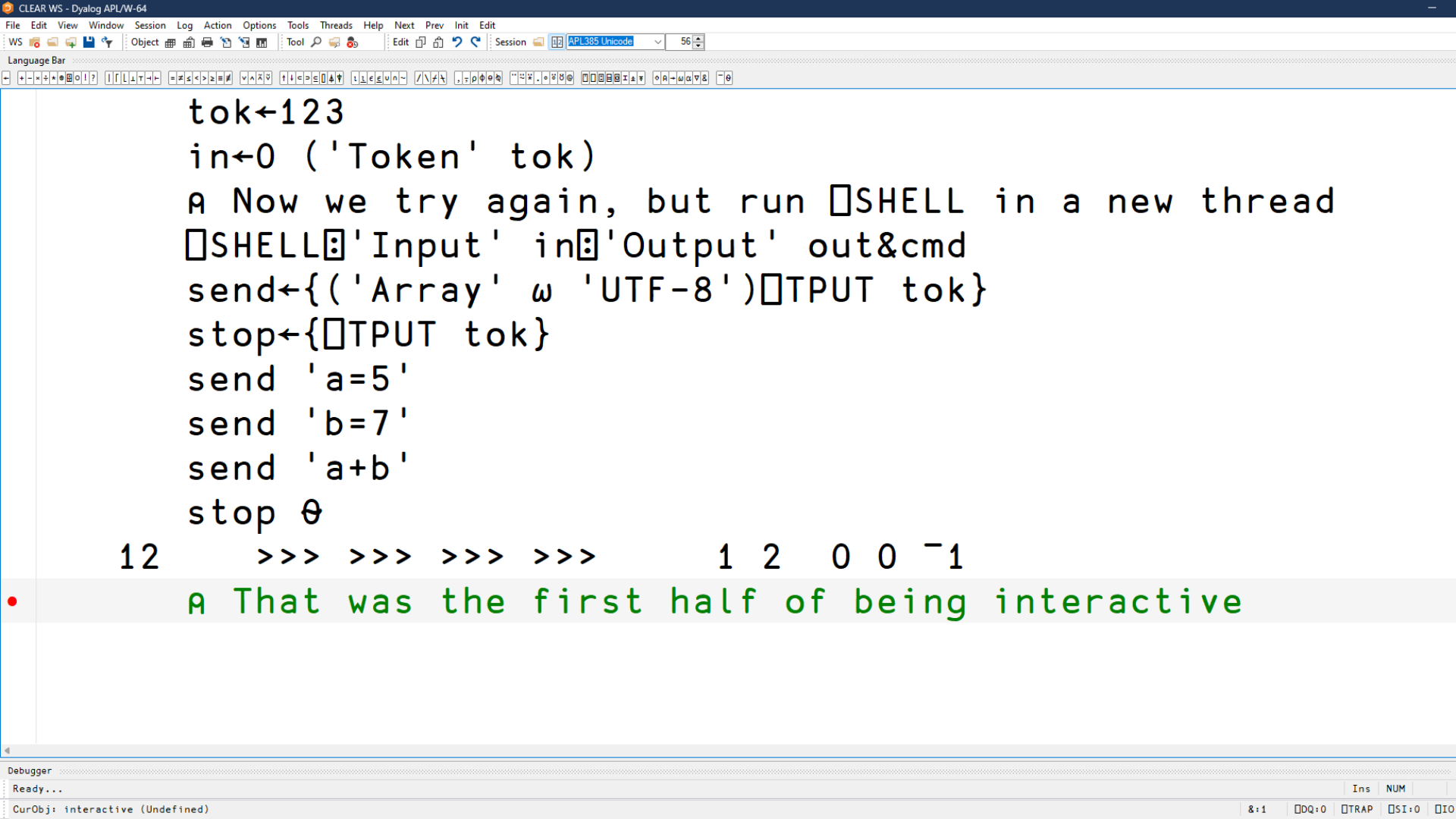












```
tok←123
```

```
in←0 ('Token' tok)
```

```
⌞ Now we try again, but run ⌞SHELL in a new thread
```

```
⌞SHELL⌞'Input' in⌞'Output' out&cmd
```

```
send←{('Array' ω 'UTF-8')⌞TPUT tok}
```

```
stop←{⌞TPUT tok}
```

```
send 'a=5'
```

```
send 'b=7'
```

```
send 'a+b'
```

```
stop 0
```

```
12      >>> >>> >>> >>>      1 2  0 0  -1
```

```
⌞ That was the first half of being interactive
```

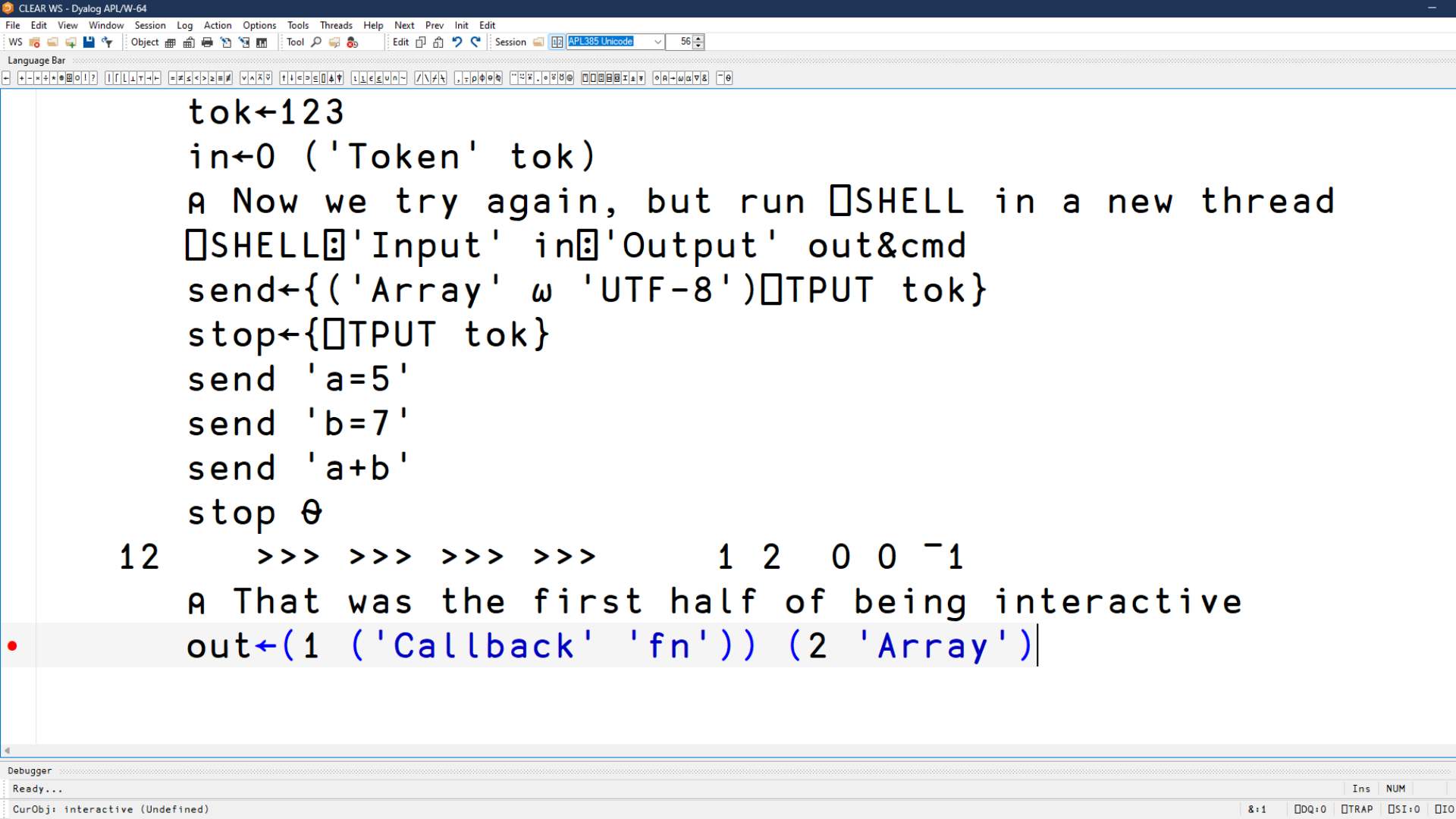
Debugger

Ready...

CurObj: interactive (Undefined)

Ins NUM

&:1 DQ:0 TRAP SI:0 IO



```
tok←123
```

```
in←0 ('Token' tok)
```

```
⌞ Now we try again, but run ⌞SHELL in a new thread
```

```
⌞SHELL⌞'Input' in⌞'Output' out&cmd
```

```
send←{('Array' ⍵ 'UTF-8')⌞TPUT tok}
```

```
stop←{⌞TPUT tok}
```

```
send 'a=5'
```

```
send 'b=7'
```

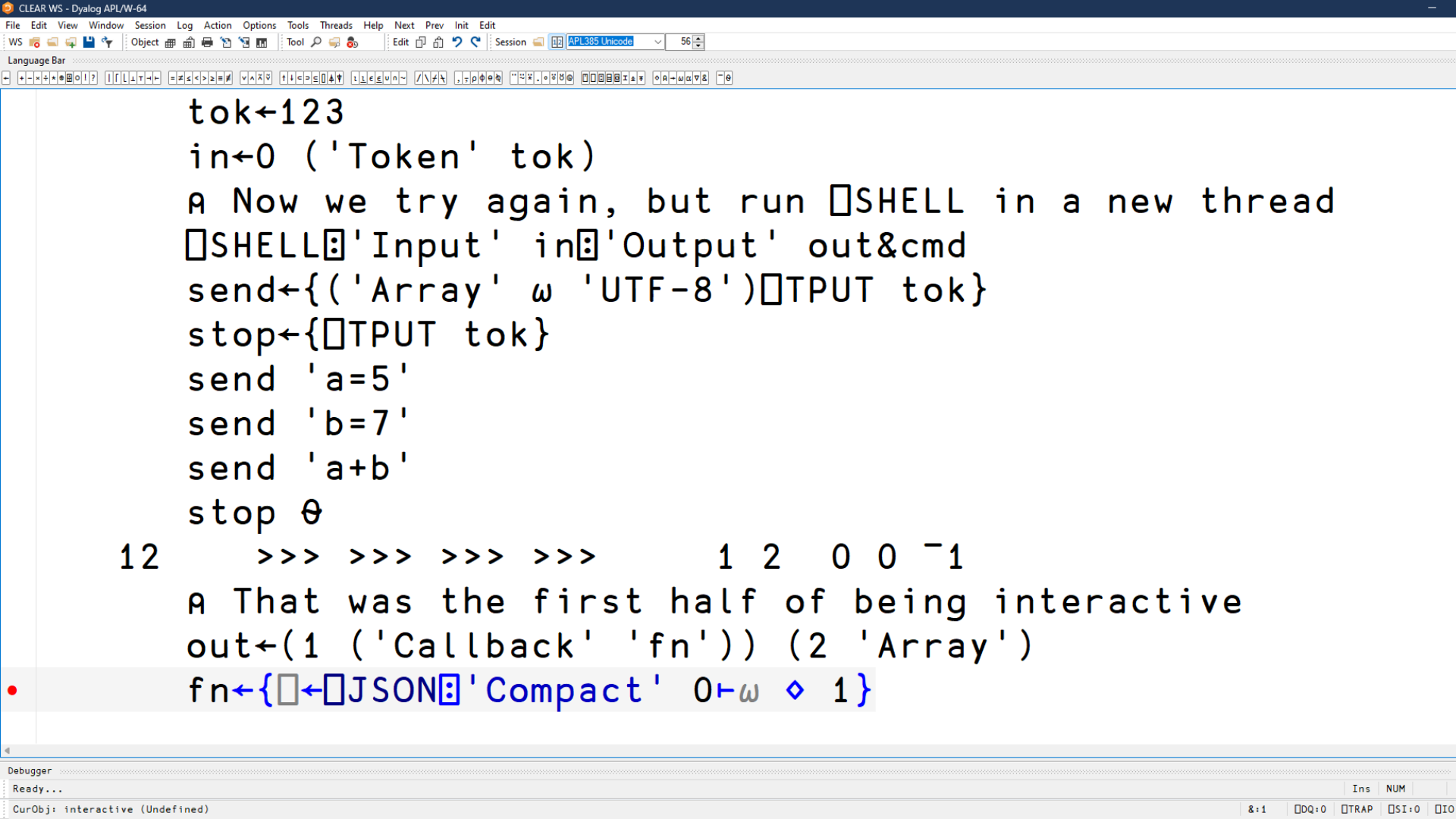
```
send 'a+b'
```

```
stop 0
```

```
12      >>> >>> >>> >>>      1 2  0 0  -1
```

```
⌞ That was the first half of being interactive
```

```
out←(1 ('Callback' 'fn')) (2 'Array')|
```



```
tok←123
```

```
in←0 ('Token' tok)
```

```
A Now we try again, but run ⎕SHELL in a new thread
```

```
⎕SHELL⎕'Input' in⎕'Output' out&cmd
```

```
send←{('Array' ⍵ 'UTF-8')⎕TPUT tok}
```

```
stop←{⎕TPUT tok}
```

```
send 'a=5'
```

```
send 'b=7'
```

```
send 'a+b'
```

```
stop 0
```

```
12      >>> >>> >>> >>>      1 2 0 0 -1
```

```
A That was the first half of being interactive
```

```
out←(1 ('Callback' 'fn')) (2 'Array')
```

```
fn←{⎕←⎕JSON⎕'Compact' 0←⍵ ⋄ 1}
```

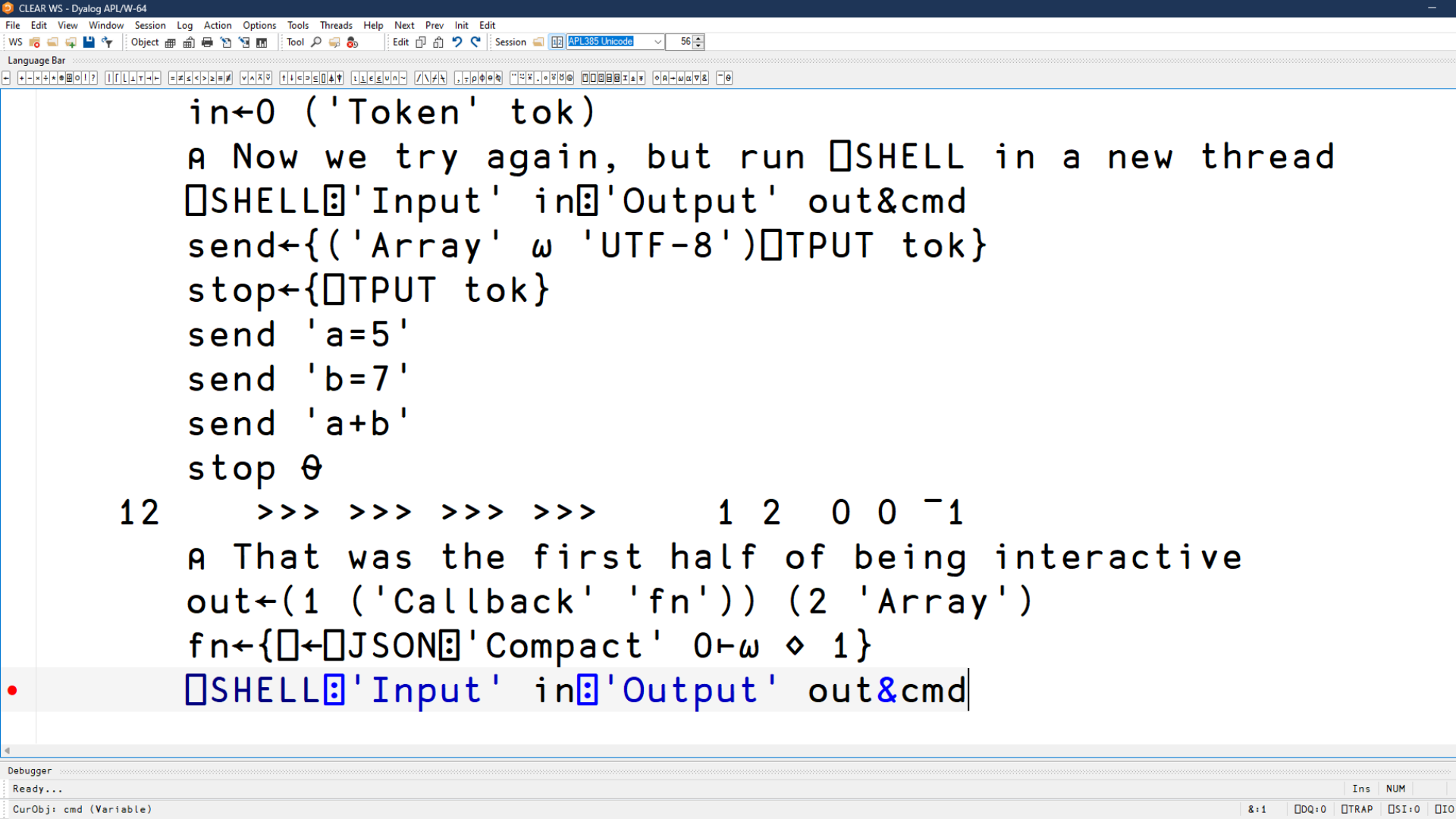
Debugger

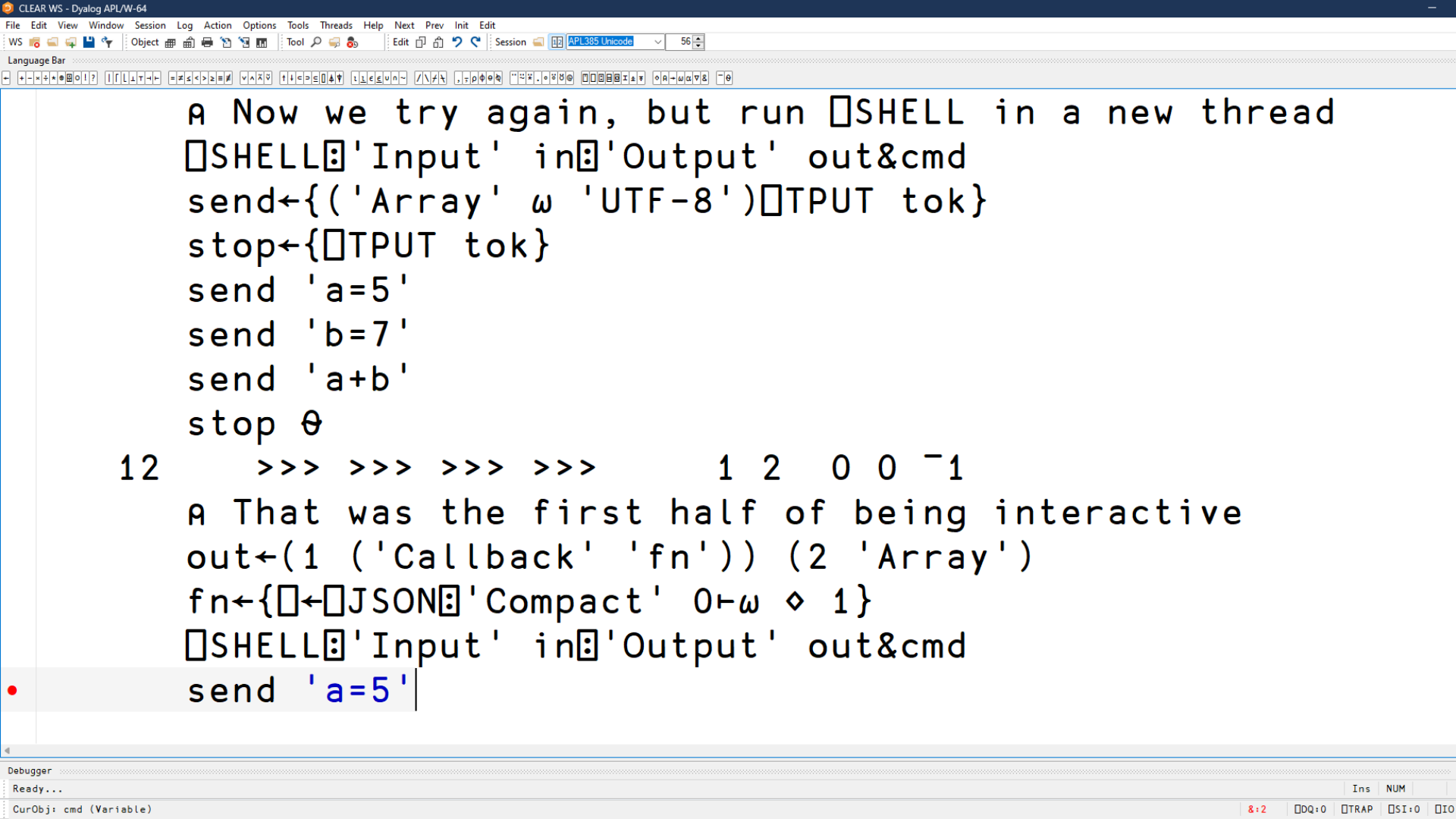
Ready...

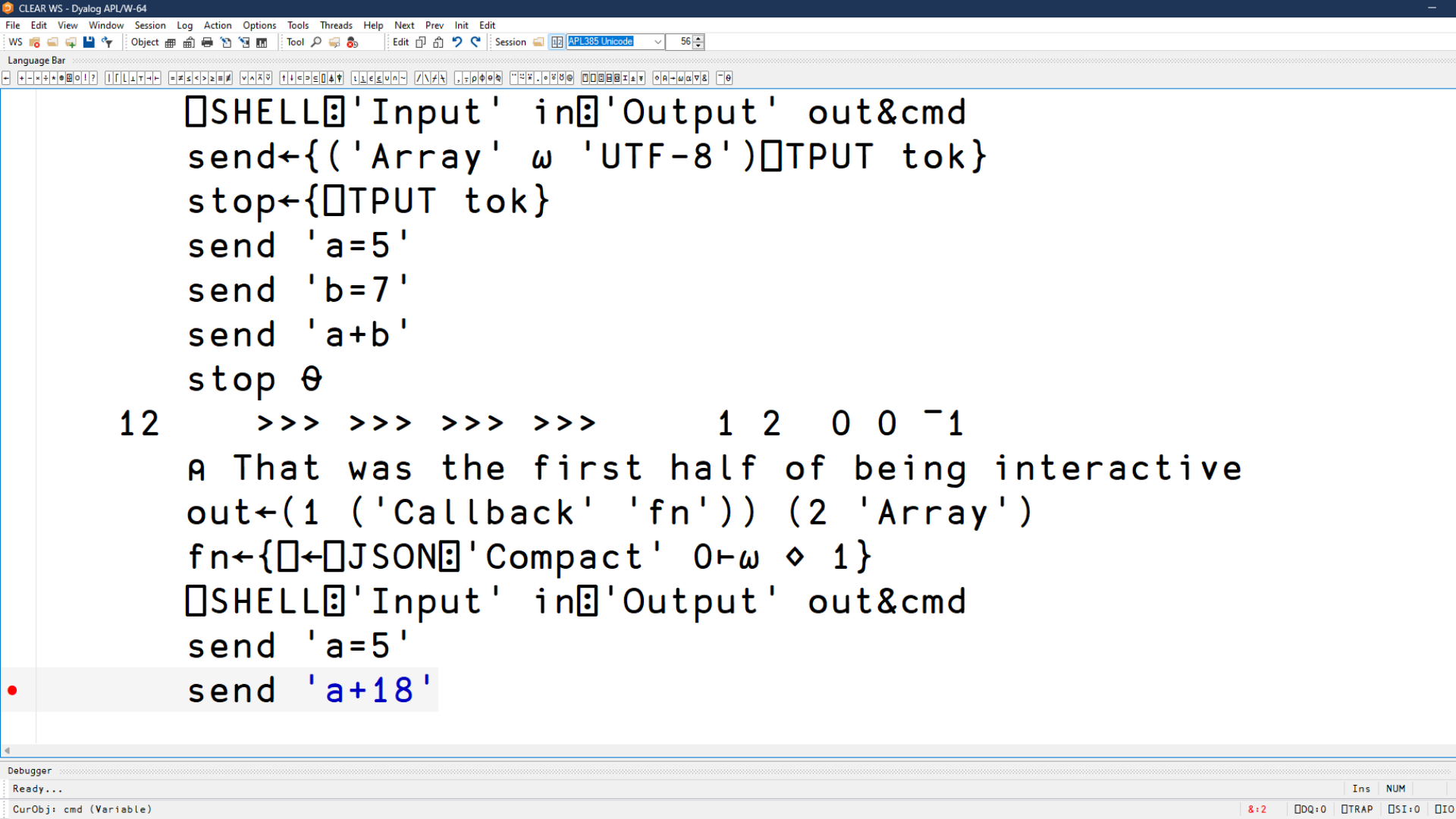
CurObj: interactive (Undefined)

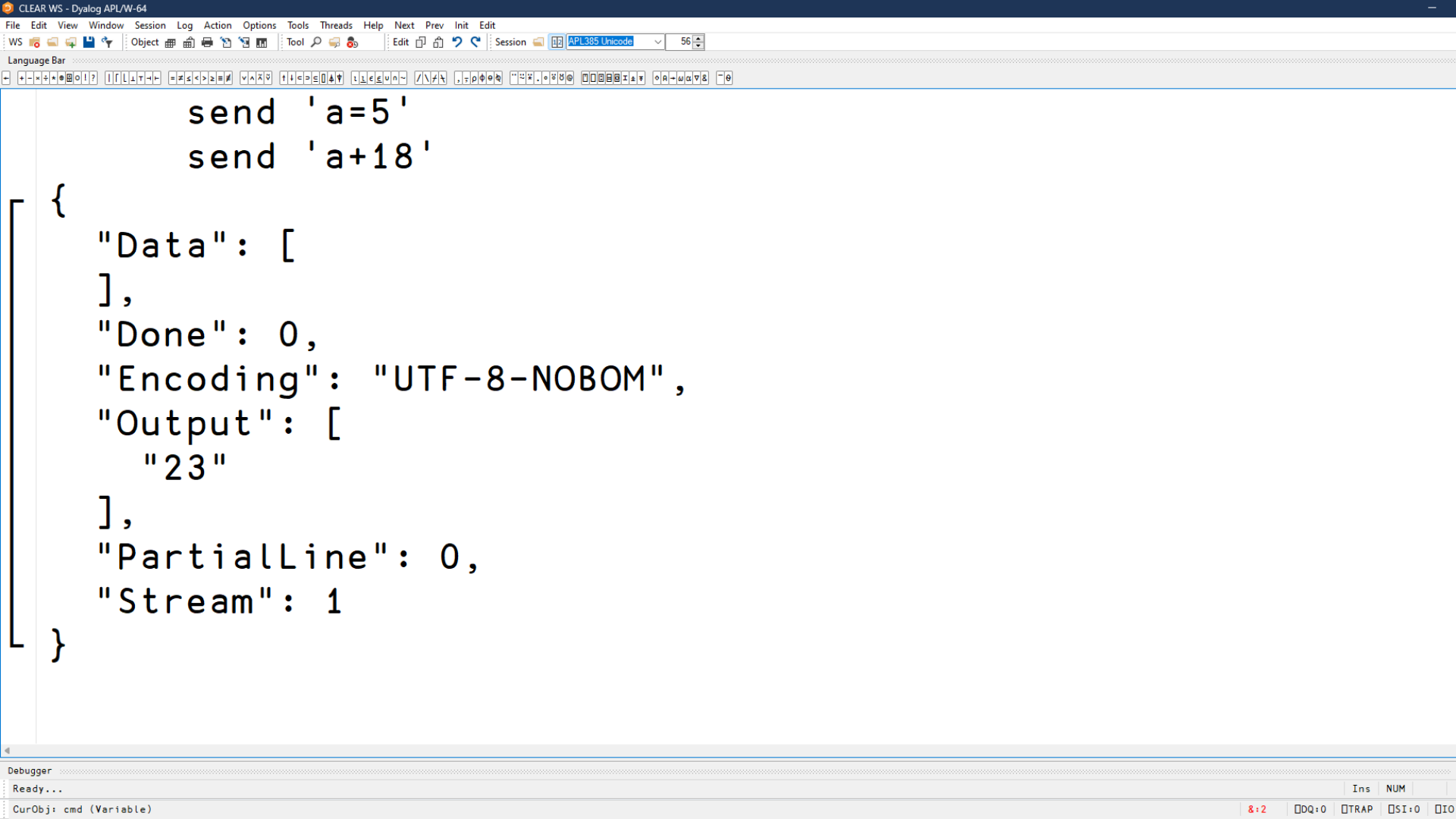
Ins NUM

&:1 ⎕DQ:0 ⎕TRAP ⎕SI:0 ⎕I









```
send 'a=5'
```

```
send 'a+18'
```

```
{
```

```
"Data": [
```

```
],
```

```
"Done": 0,
```

```
"Encoding": "UTF-8-NOBOM",
```

```
"Output": [
```

```
"23"
```

```
],
```

```
"PartialLine": 0,
```

```
"Stream": 1
```

```
}
```

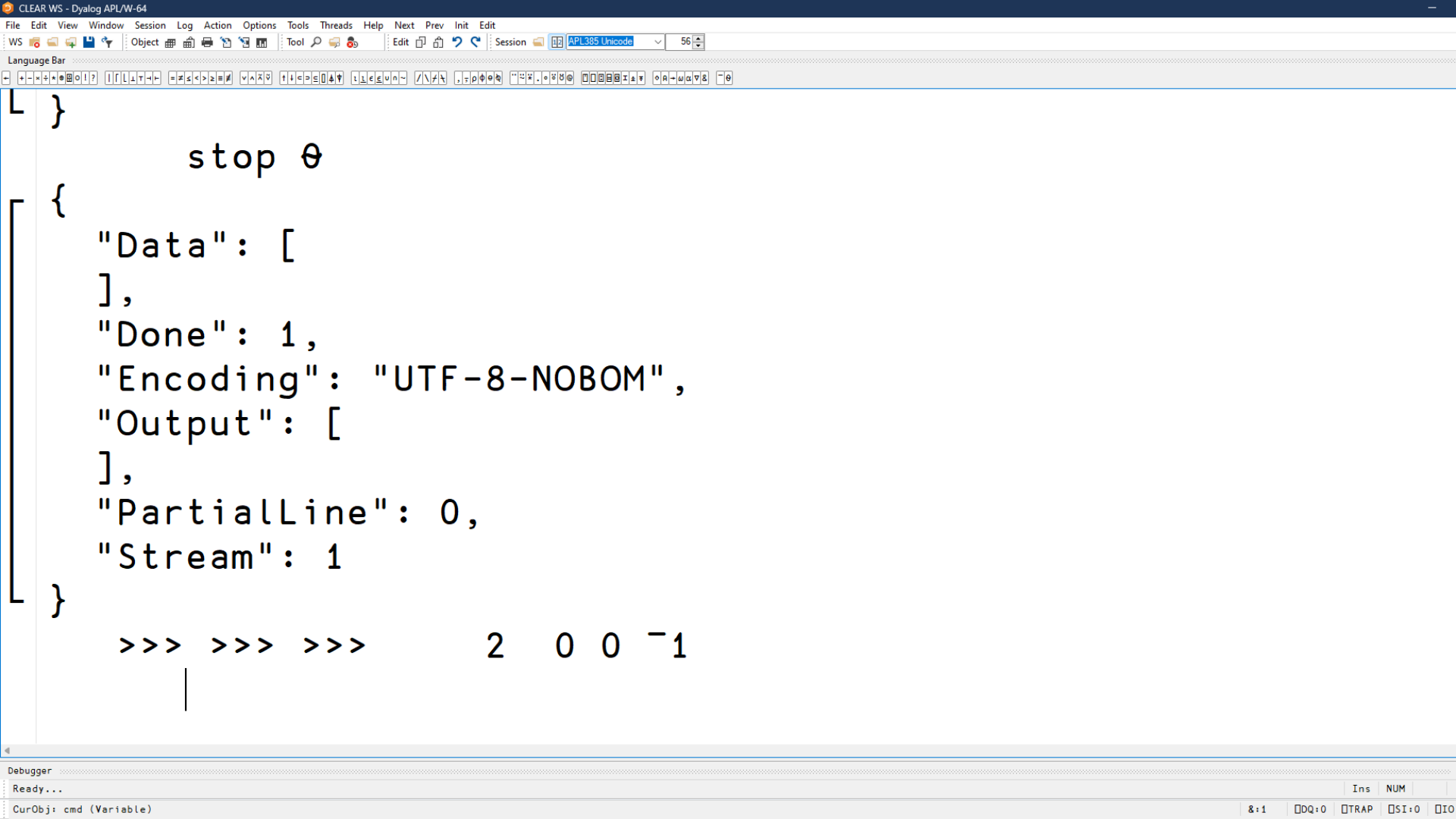


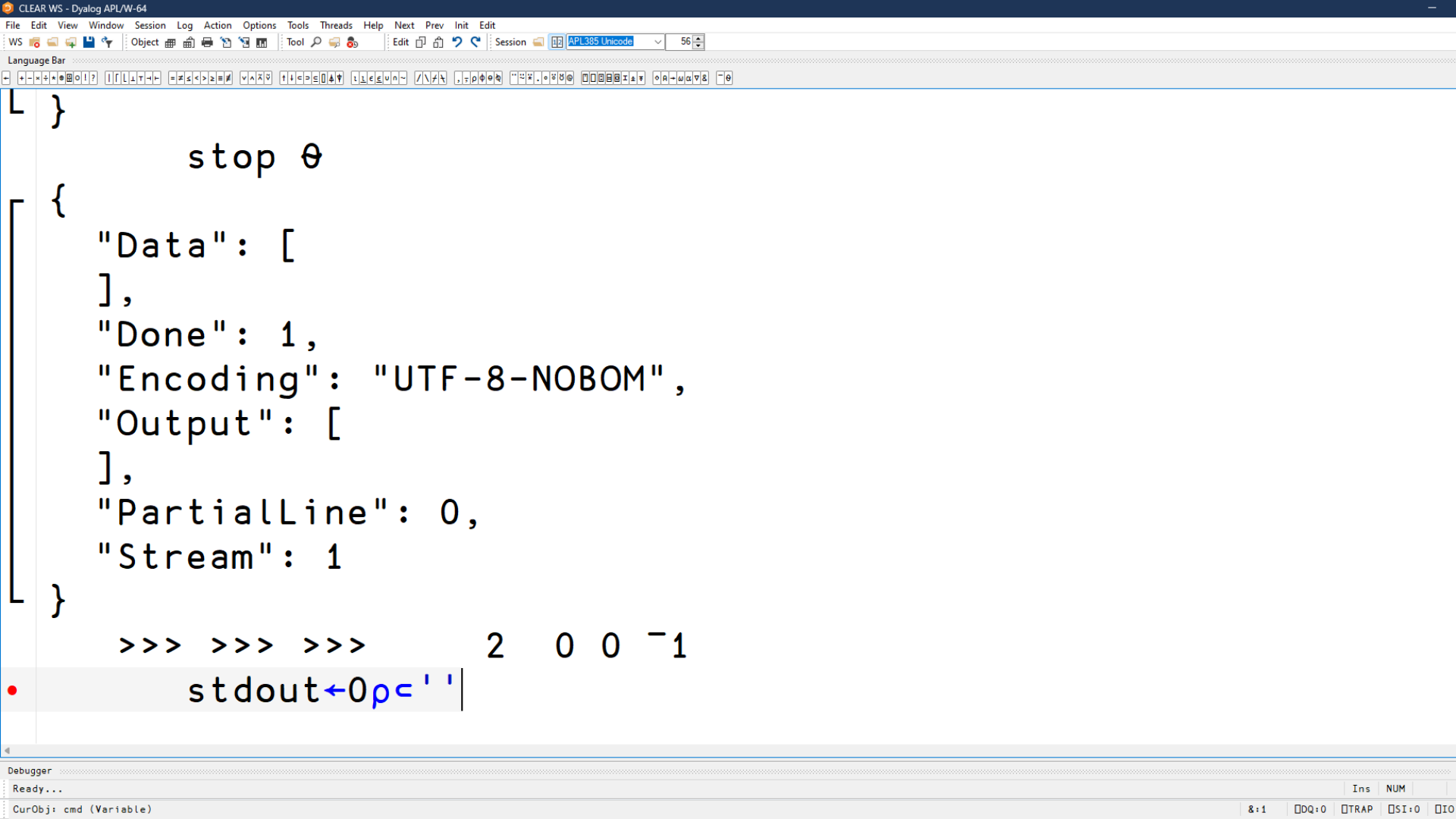
```
send 'a=5'
```

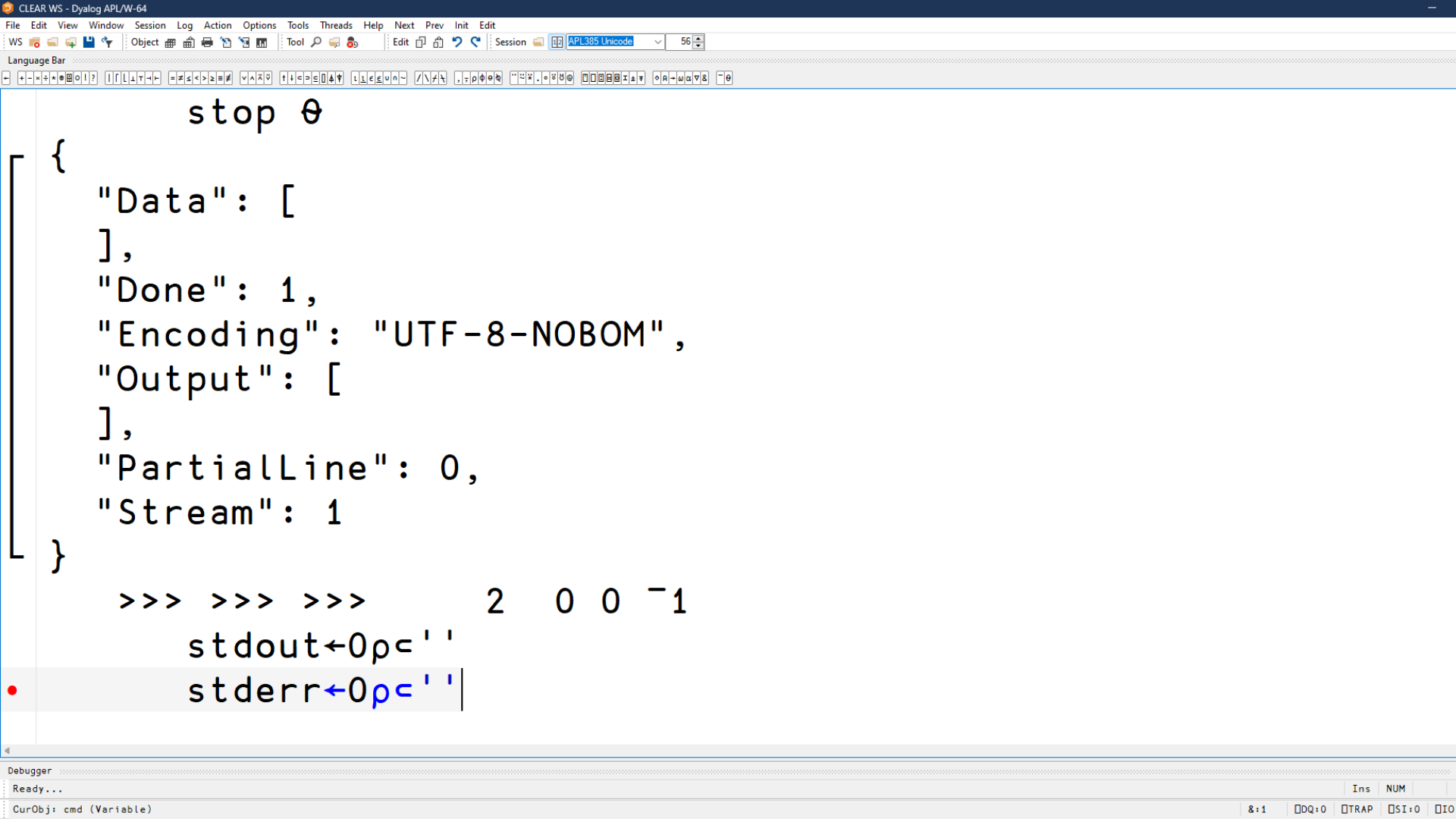
```
send 'a+18'
```

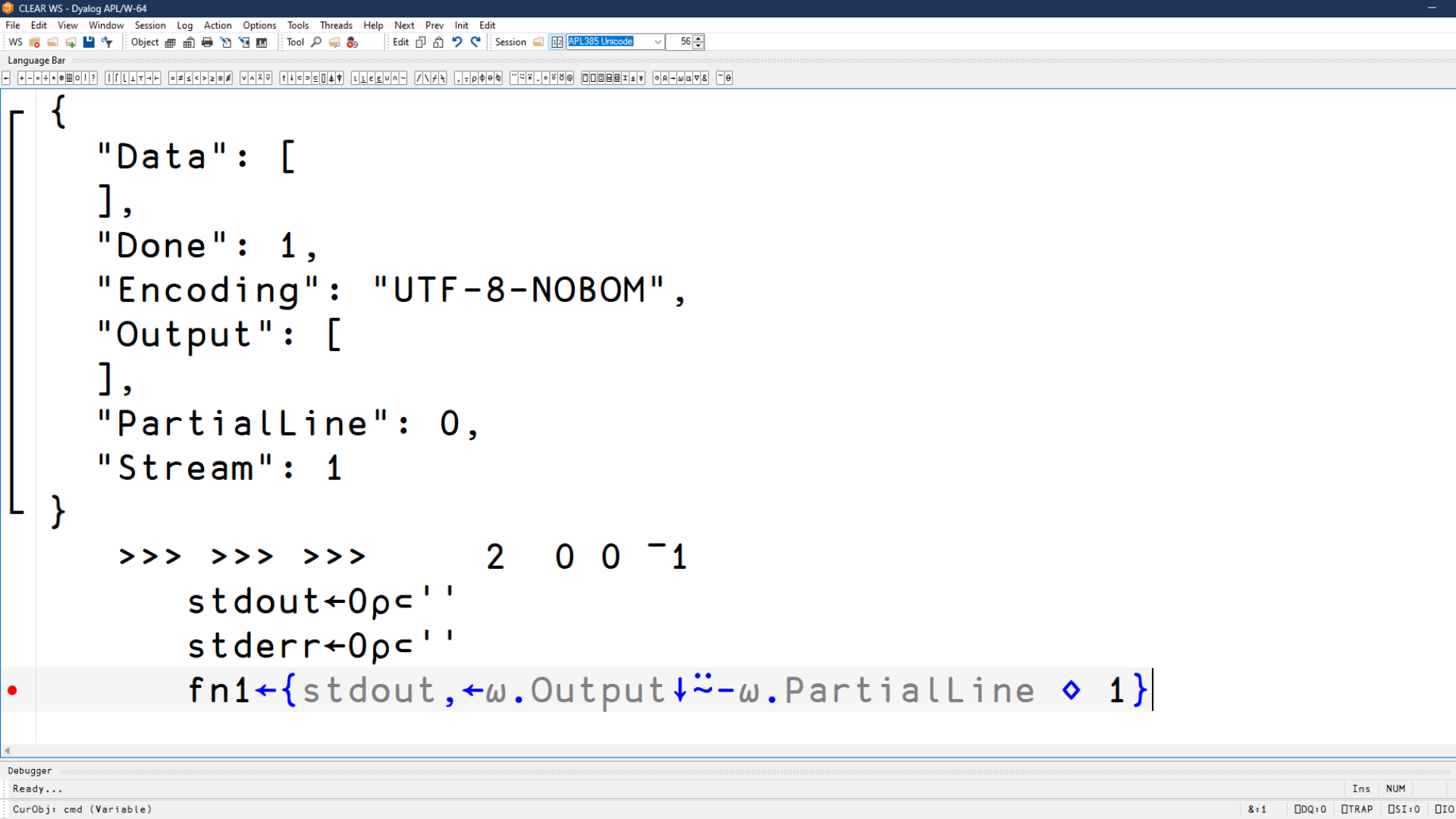
```
{
  "Data": [
  ],
  "Done": 0,
  "Encoding": "UTF-8-NOBOM",
  "Output": [
    "23"
  ],
  "PartialLine": 0,
  "Stream": 1
}
```

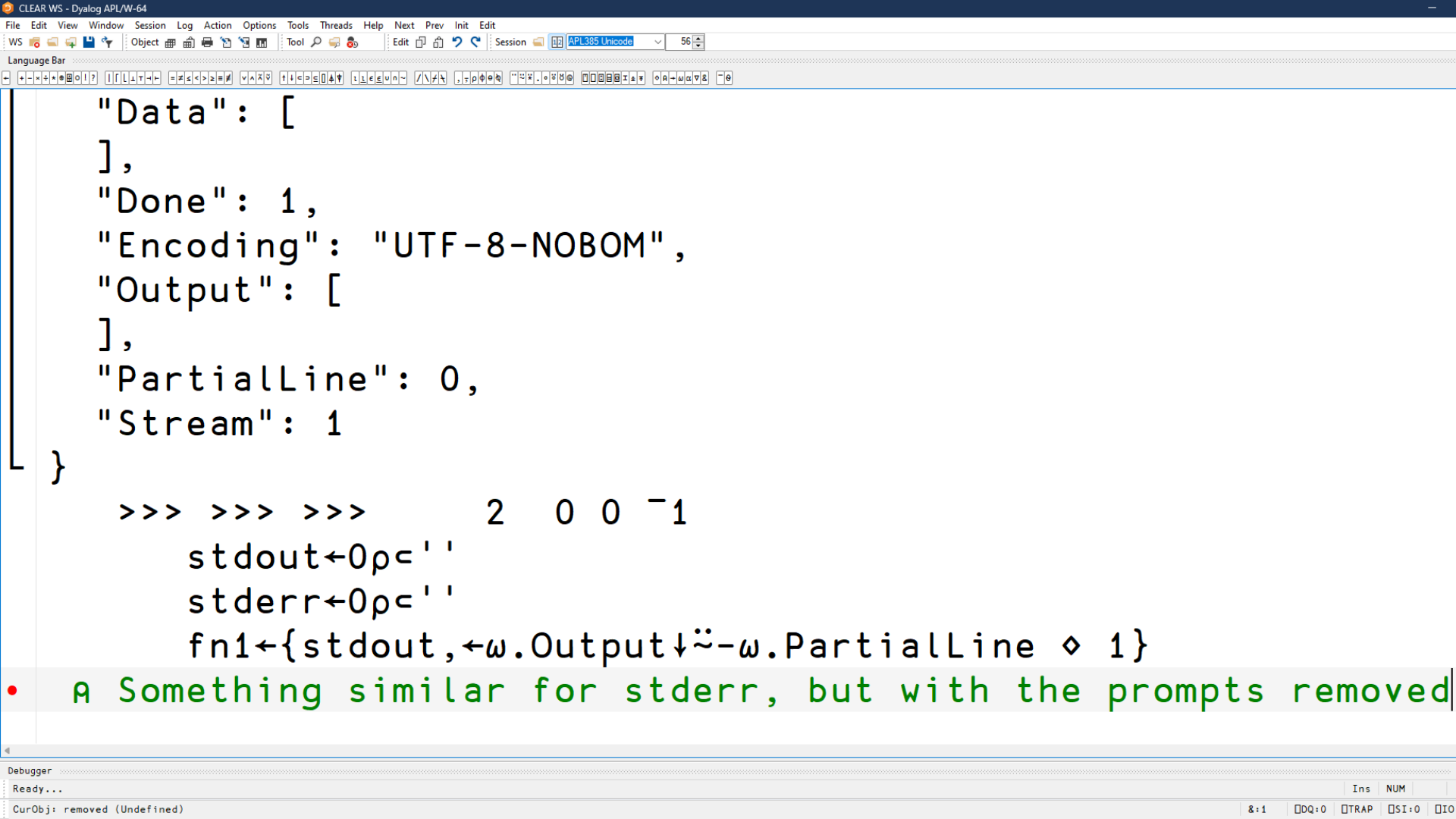
stop  $\theta$

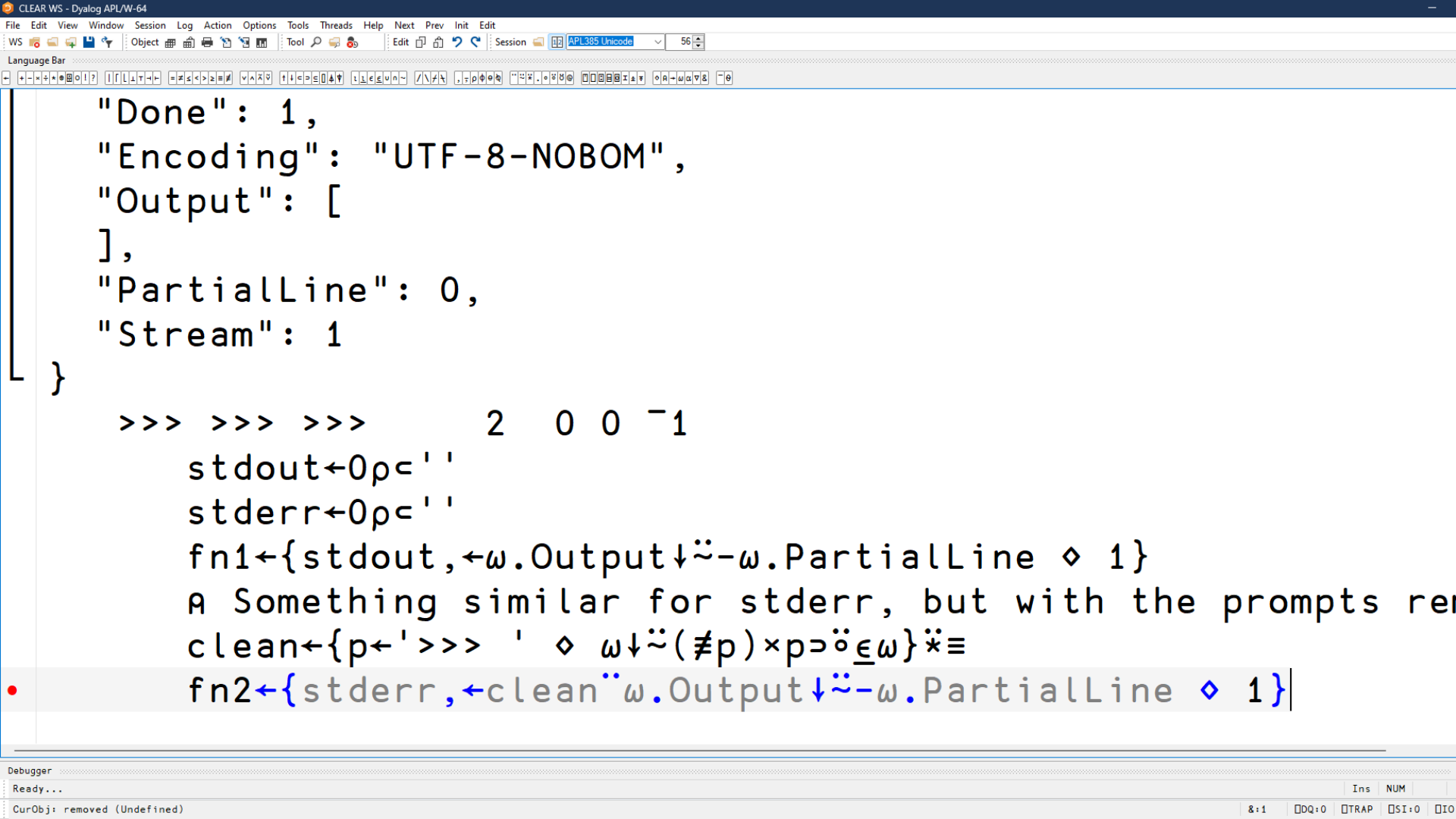












```
"Done": 1,  
"Encoding": "UTF-8-NOBOM",  
"Output": [  
],  
"PartialLine": 0,  
"Stream": 1  
}
```

```
>>> >>> >>> 2 0 0 -1
```

```
stdout←0p<' '
```

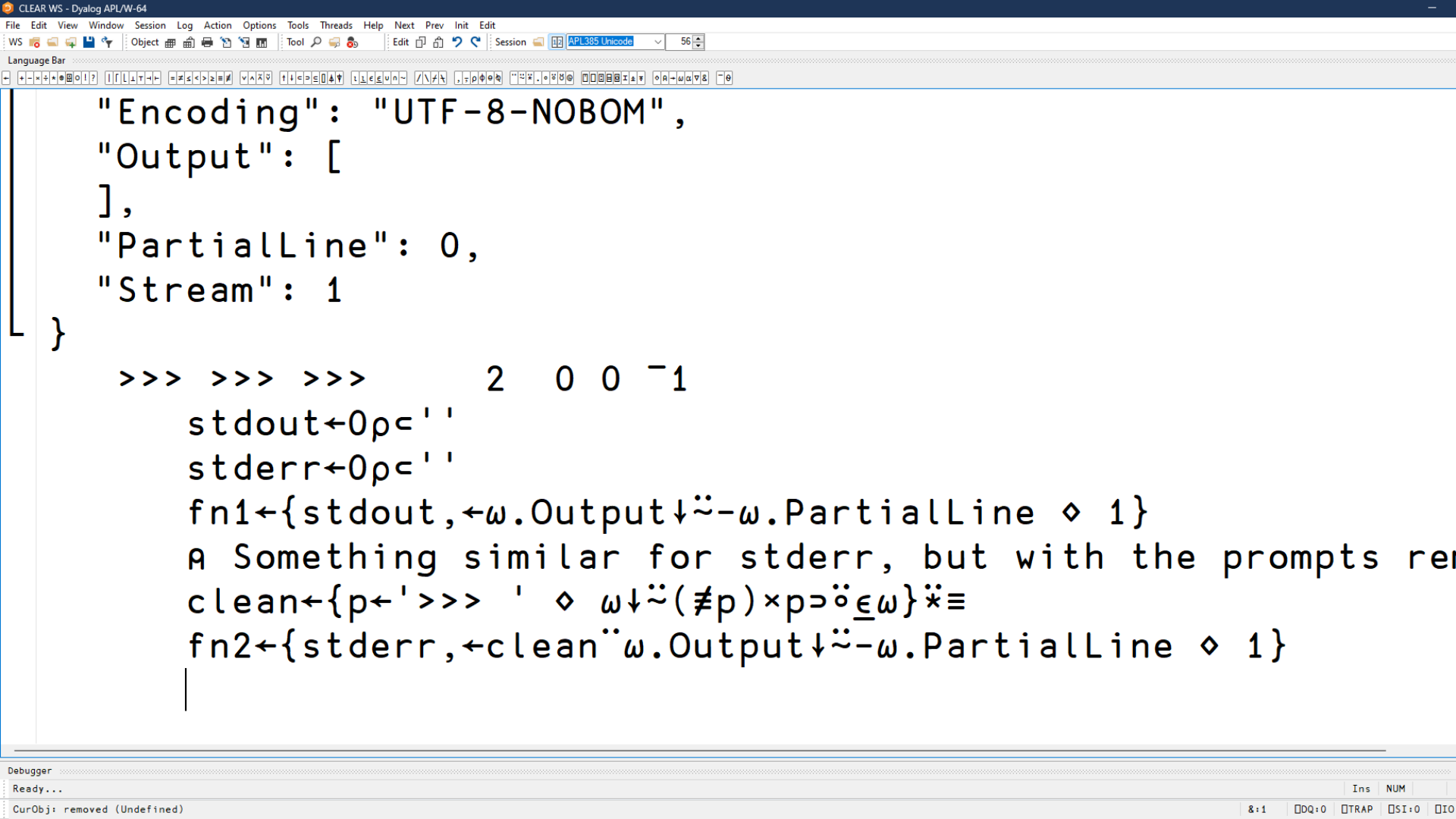
```
stderr←0p<' '
```

```
fn1←{stdout,←ω.Output↓~ω.PartialLine ⋄ 1}
```

```
A Something similar for stderr, but with the prompts re
```

```
clean←{p←'>>> ' ⋄ ω↓~(≠p)×p>∘⊆ω}*≡
```

```
fn2←{stderr,←cleanω.Output↓~ω.PartialLine ⋄ 1}
```



```
"Encoding": "UTF-8-NOBOM",
```

```
"Output": [
```

```
],
```

```
"PartialLine": 0,
```

```
"Stream": 1
```

```
}
```

```
>>> >>> >>>      2  0 0 -1
```

```
stdout<-0p<' '
```

```
stderr<-0p<' '
```

```
fn1<-{stdout,<-w.Output<-w.PartialLine <- 1}
```

```
A Something similar for stderr, but with the prompts re
```

```
clean<-{p<'>>> ' <- w<-w.PartialLine <- 1}
```

```
fn2<-{stderr,<-cleanw.Output<-w.PartialLine <- 1}
```

Debugger

Ready...

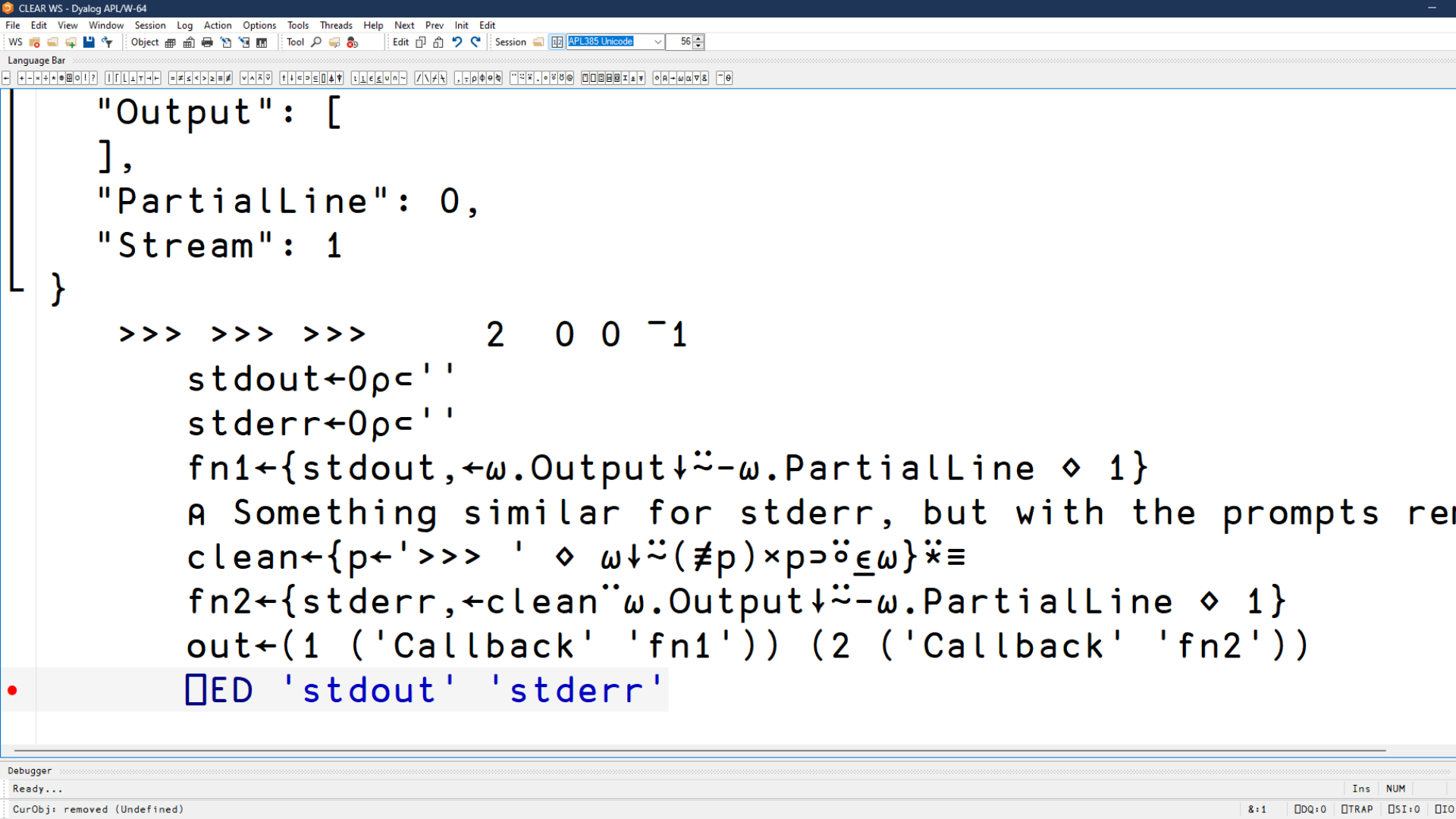
CurObj: removed (Undefined)

Ins NUM

&:1 DQ:0 TRAP SI:0 IO







```
"Output": [
```

```
],
```

```
"PartialLine": 0,
```

```
"Stream": 1
```

```
}
```

```
>>> >>> >>>      2  0  0  -1
```

```
stdout←0p←''
```

```
stderr←0p←''
```

```
fn1←{stdout,←ω.Output↓~-ω.PartialLine ⋄ 1}
```

```
A Something similar for stderr, but with the prompts re
```

```
clean←{p←'>>> ' ⋄ ω↓~(≠p)×p>ö⊆ω}*≡
```

```
fn2←{stderr,←cleanω.Output↓~-ω.PartialLine ⋄ 1}
```

```
out←(1 ('Callback' 'fn1')) (2 ('Callback' 'fn2'))
```

```
ED 'stdout' 'stderr'
```

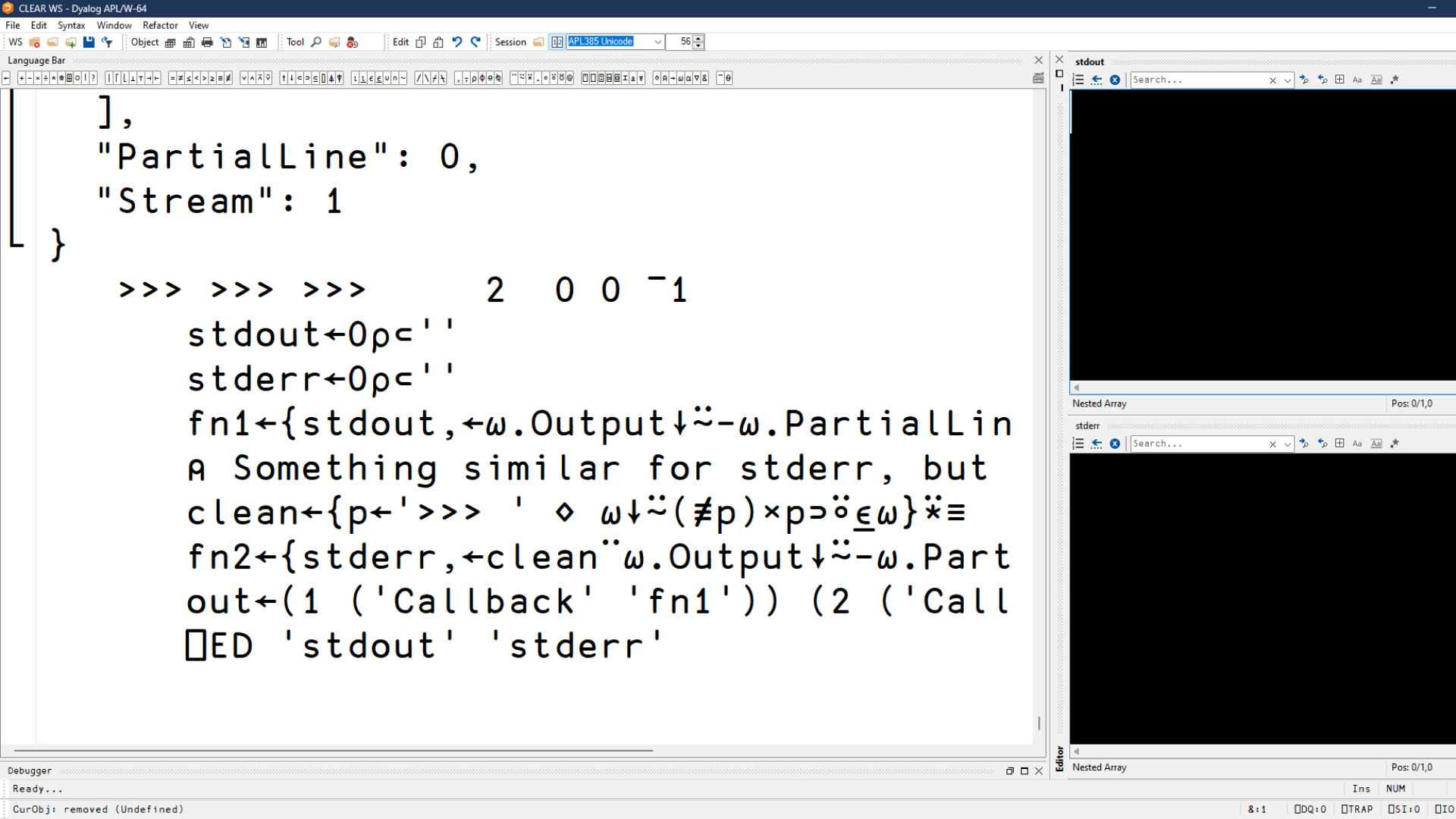
Debugger

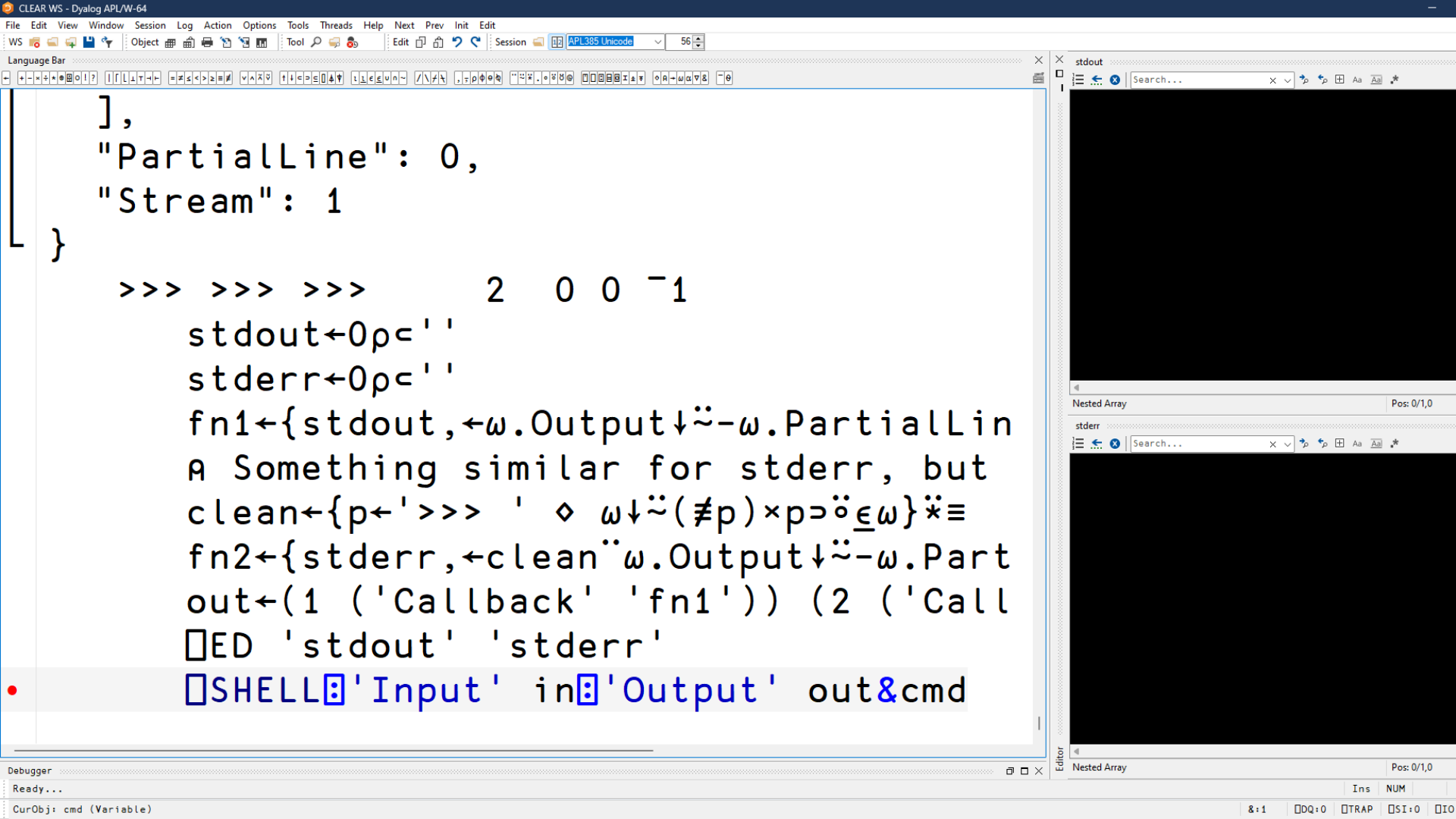
Ready...

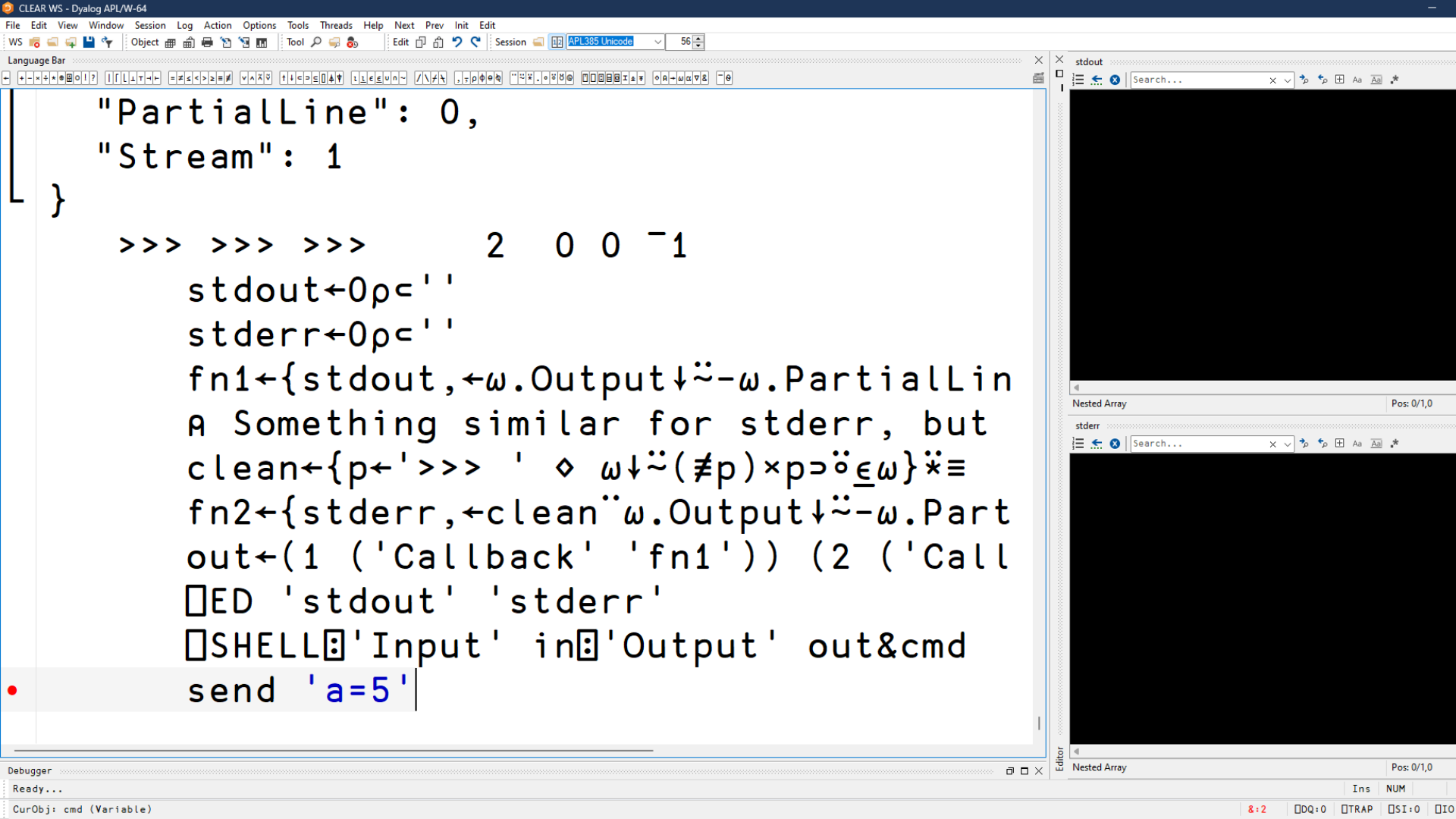
CurObj: removed (Undefined)

Ins NUM

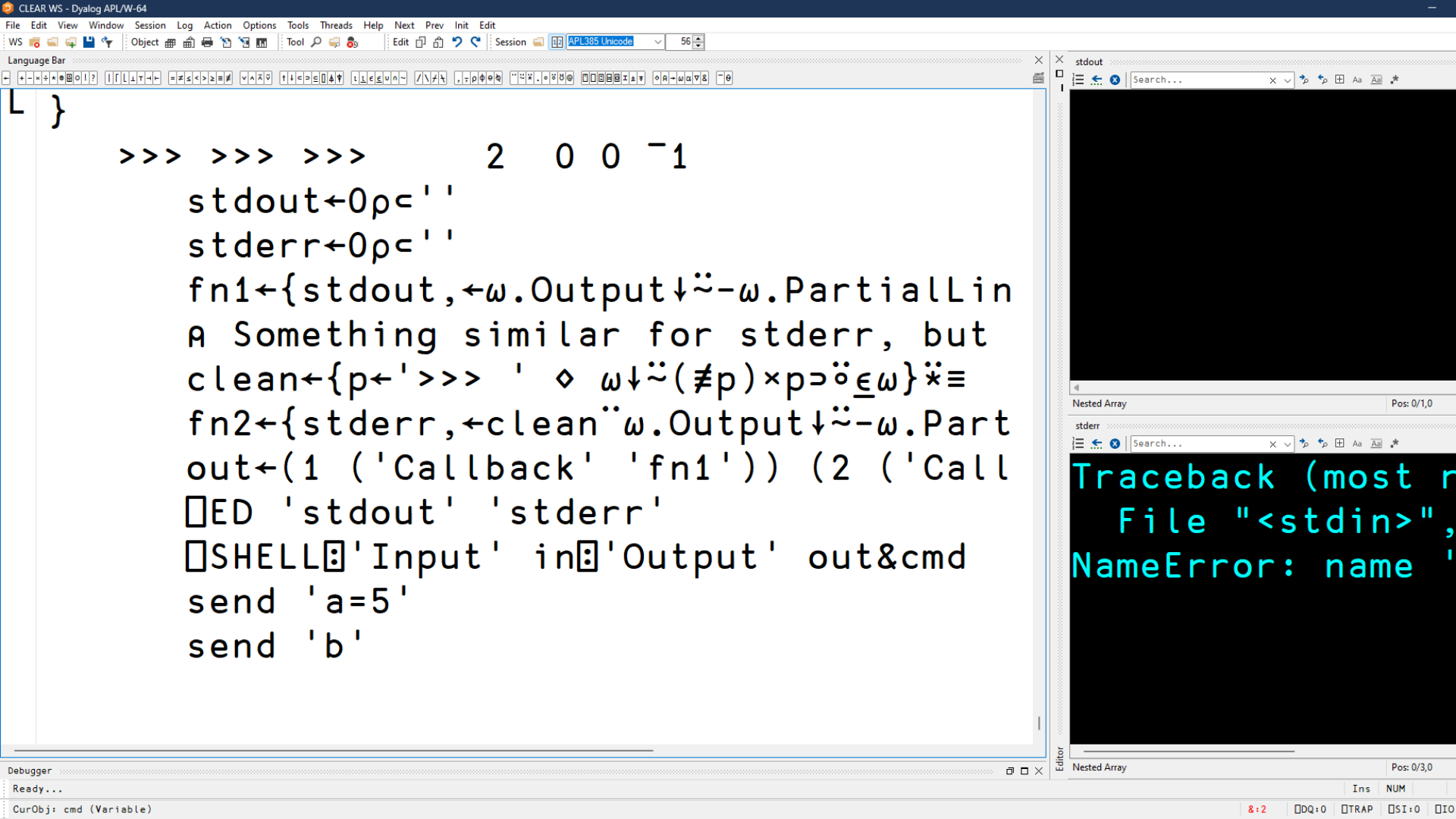
&:1 DQ:0 TRAP SI:0 IO











[illegible]

```
stdout ← 0p ← ' '
```

```
stderr ← 0; c ← ''
```

```
fn1←{stdout,←ω.Output↓~ω.PartialLin
```

A Something similar for stderr, but

$$\text{clean} \leftarrow \{p \leftarrow ' > > > ' \mid \omega \downarrow \sim (\neq p) \times p \supset \ddot{\omega} \in \omega\} \ddot{\equiv}$$

```
fn2←{stderr,←clean"ω.Output↓~−ω.Part
```

```
out←(1 ('Callback' 'fn1')) (2 ('Call
```

```
ED 'stdout' 'stderr'
```

```
□SHELL@'Input' in@'Output' out&cmd
```

```
send 'a=5'
```

```
send 'b'
```

```
send 'b=12'
```

stdout

Nested Array	Pos: 0/1,0
--------------	------------

stderr

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Debugger :

Ready...

```
CurObj: cmd (Variable)
```



### Nested Array

Pos: 0/3,0

Ins

NUM

8:2

□DQ: 0

☐ TRAP

□SI:0

□ IO



```
CurObj: cmd (Variable)
```

[illegible]

8:2	<input type="checkbox"/> DQ:0	<input type="checkbox"/> TRAP	<input type="checkbox"/> SI:0	<input type="checkbox"/> IC
-----	-------------------------------	-------------------------------	-------------------------------	-----------------------------

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```



Pos: 0/3,0

10

IO

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

IO

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

```

A Something similar for stderr, but
clean←{p←'>>> ' ⋄ ω↓~(≠p)×p>ö⊆ω}*≡
fn2←{stderr,←clean`ω.Output↓~-ω.Part
out←(1 ('Callback' 'fn1')) (2 ('Call
□ED 'stdout' 'stderr'
□SHELL□'Input' in□'Output' out&cmd
send 'a=5'
send 'b'
send 'b=12'
send 'a+b'
send 'a-b'
stop θ
0 0 -1

```

$$\begin{array}{r} 17 \\ -7 \\ \hline \end{array}$$

### Nested Array

Pos: 0/2,0

stderr

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Debugger

Ready...

```
CurObj: cmd (Variable)
```

Ins

NUM

8:1

**DQ**

☐ TRAP

SI: 0

IO

# End of demo

- ✧ This is of course only an example
- ✧ Wrap it up in a class
  - ✧ `p ← NEW python`
  - ✧ ...
- ✧ `□SHELL` is designed not to have a specific use-case in mind

# Summary

- ◆ Lots of new stuff in version 20.0
- ◆ Including `□SHELL`, which we took a deeper look at
- ◆ Now is a good time for questions