Blending Forth

mixing other languages and Forth

EuroForth'25 conference 2025-09

Ulrich Hoffmann

uho@ .de

Overview

- introduction
- implementing Forth in other languages
- abstraction and representation
- blending Forth
- demo
- conclusion

Day of the Week and Zeller’s congruence

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

```
function ZellerDayOfWeek(q, m, y: Integer): Integer;
…
begin

  K := y mod 100;   // year of the century
  J := y div 100;   // zero-based century

  h := (q + ((13 * (m + 1)) div 5) + K +
                           (K div 4) + (J div 4) - (2 * J)) mod 7;
  …
  ZellerDayOfWeek := h
end;
```

Day of the Week and Zeller’s congruence

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

In **standard Pascal (ISO 7185 and its descendants like Free Pascal, Turbo Pascal, Delphi)** the `mod` operator always returns a result with the **same sign as the dividend** (the left operand).

```
Writeln(  7 mod 3 );    // 1
Writeln( -7 mod 3 );    // -1
Writeln(  7 mod -3 );   // 1
Writeln( -7 mod -3 );   // -1
```

Day of the Week and Zeller’s congruence

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

```
function ZellerDayOfWeek(q, m, y: Integer): Integer;
…
begin

  K := y mod 100;   // year of the century
  J := y div 100;   // zero-based century

  h := (q + ((13 * (m + 1)) div 5) + K +
                           (K div 4) + (J div 4) - (2 * J)) mod 7;
  …
  ZellerDayOfWeek := h
end;
```

Day of the Week and Zeller’s congruence

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor - 2J\right) \bmod 7,$$

```
function ZellerDayOfWeek(q, m, y: Integer): Integer;
…
begin

  K := y mod 100;   // year of the century
  J := y div 100;   // zero-based century

  h := (q + ((13 * (m + 1)) div 5) + K +
                           (K div 4) + (J div 4) + (5 * J)) mod 7;
  …
  ZellerDayOfWeek := h
end;
```

Implementing Forth in Python

- ongoing adventure to implement Forth in different languages
  - Assembler
  - Forth itself
  - Emacs-Lisp
  - Golang
  - Python
- Insightful discoveries

Implementing Forth in Python

- How to implemement stack and return-stack?
- How primitives?
- How the dictionary?
- How the inner and out interpreter?
- What about BASE and STATE?
- How to read characters one-by-one?

Factorial

```
: fac ( n -- n! )
    ?dup IF dup 1- recurse * exit THEN 1 ;
```

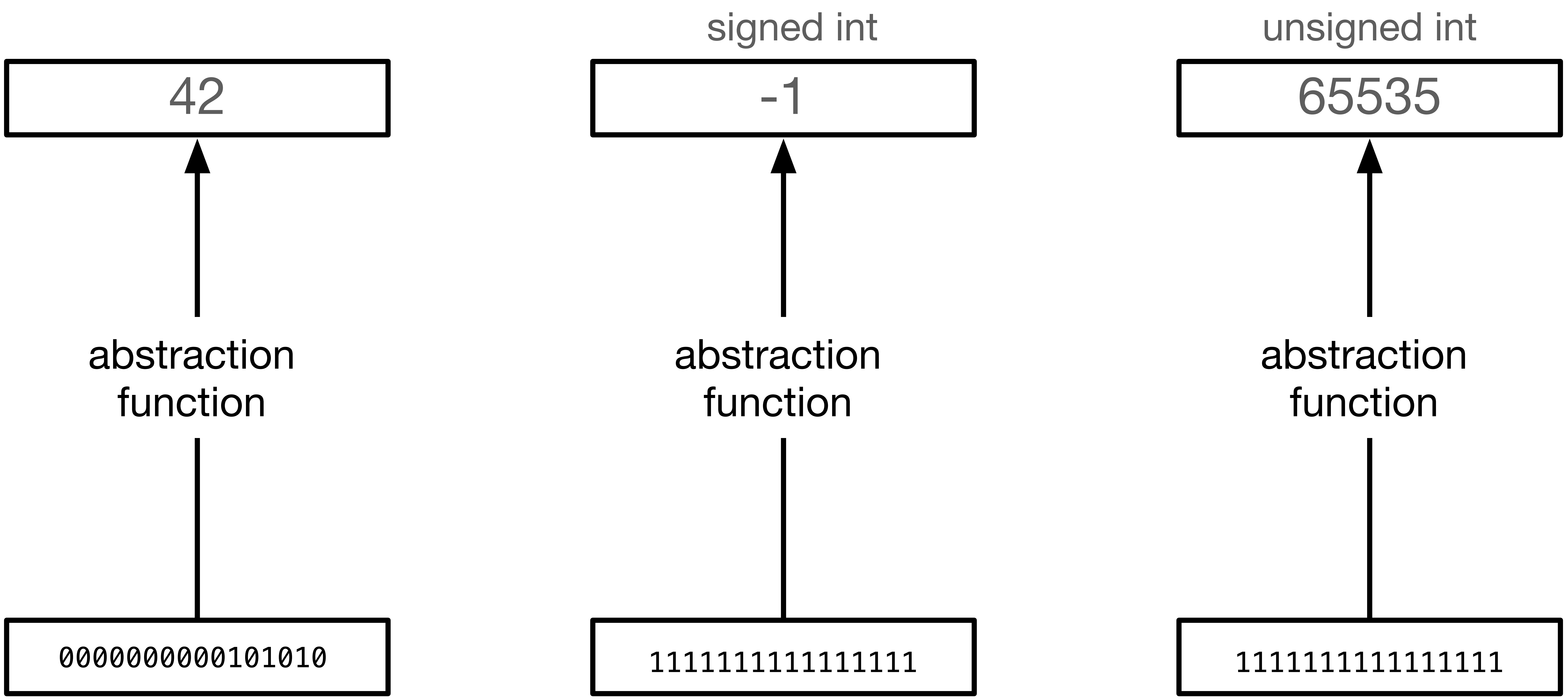
```
10 fac . 3628800  ok 🙂
```

Abstraction and Representation

- You take the elements of the implementation language to realize the elements of the target language Forth.
- data refinement
- operator refinement

Data Refinement

depends on type



signed int

unsigned int

42

-1

65535

abstraction function

abstraction function

abstraction function

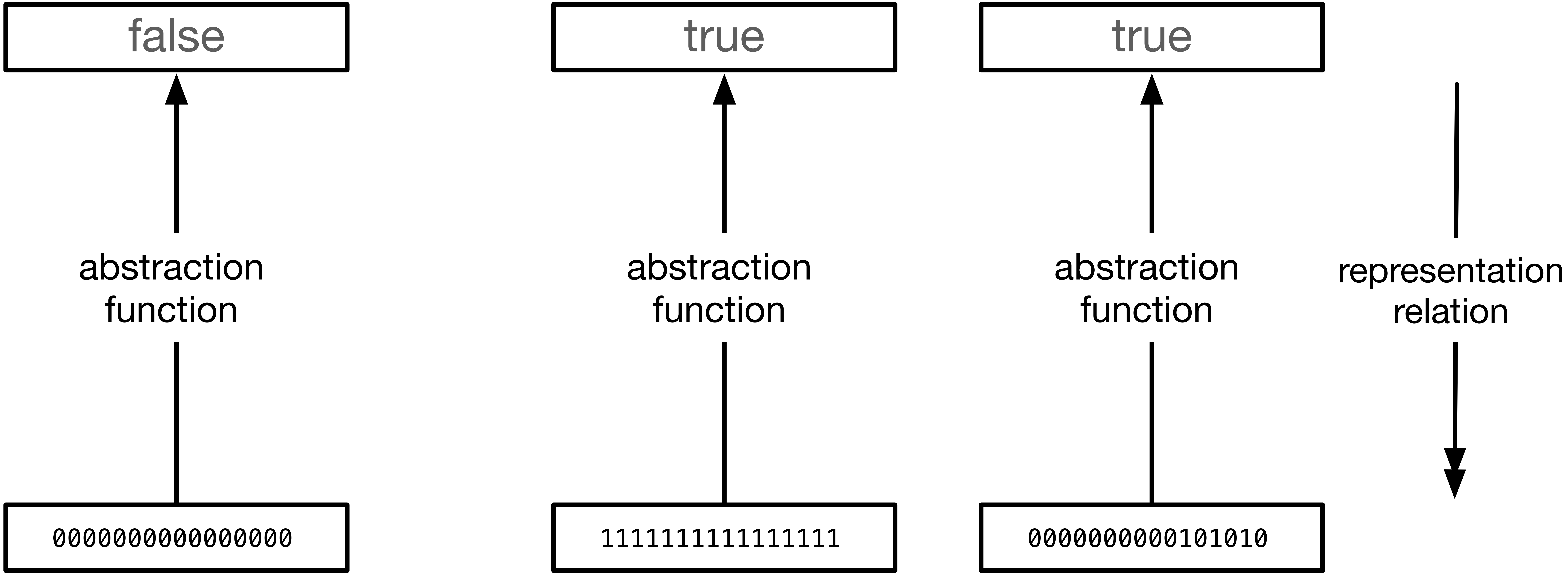
0000000000101010

1111111111111111

1111111111111111

abstraction function sometimes called *retrieval* function

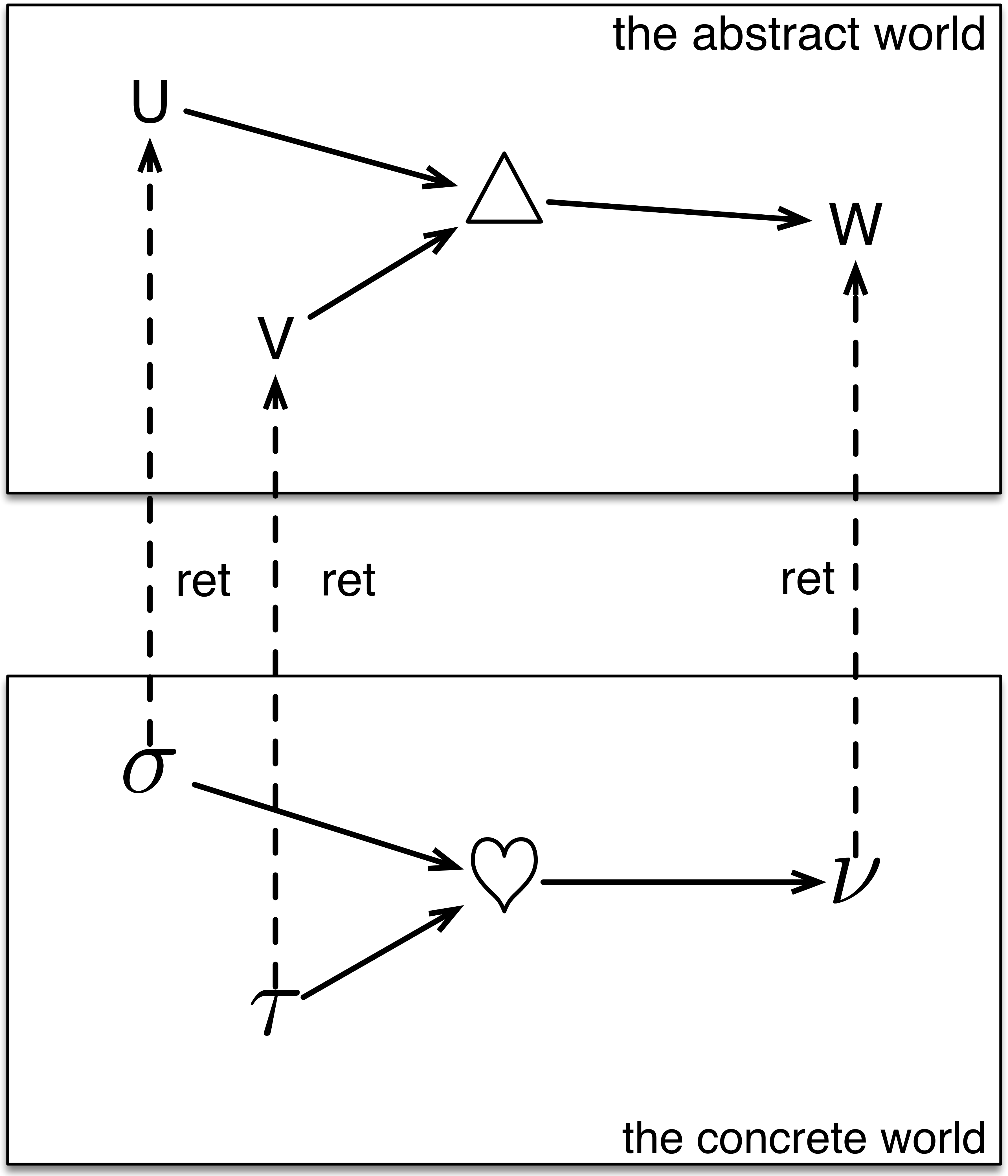
Data Refinement



representation relation sometimes called *refinement relation*

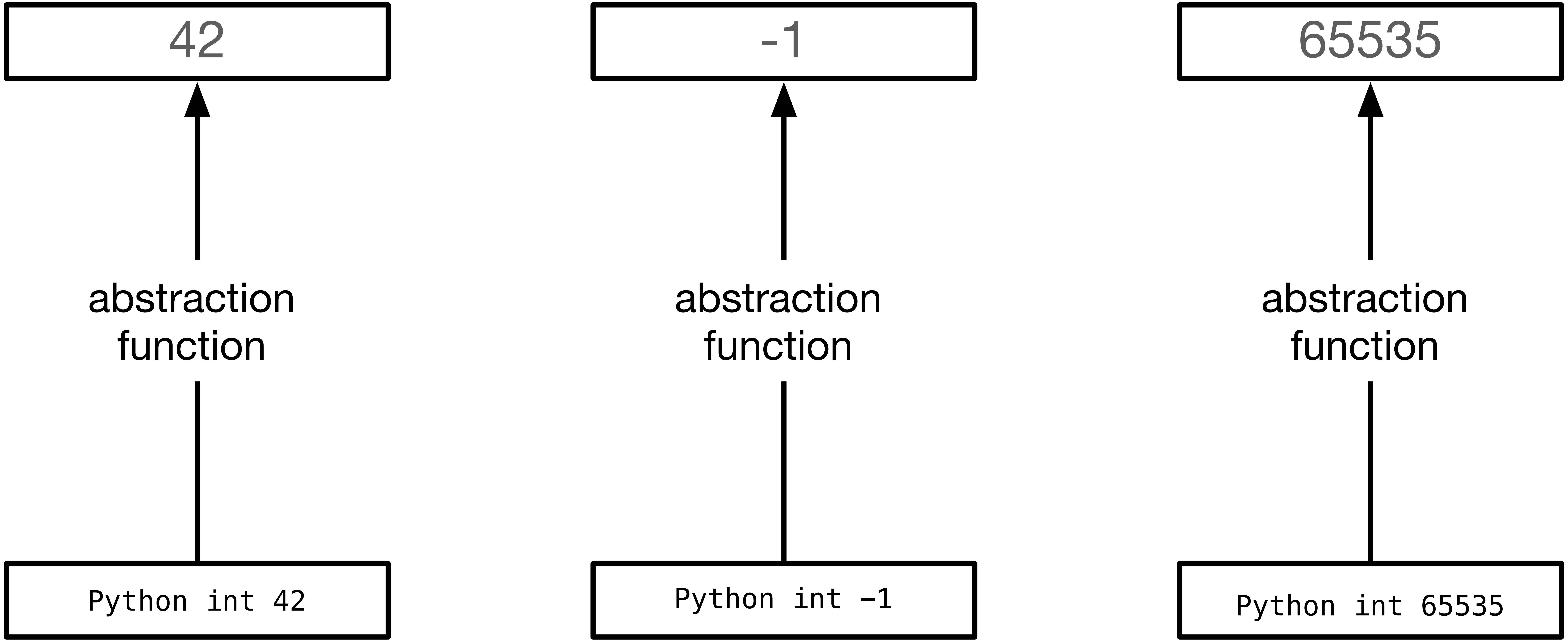
Operator Refinement

$$\mathit{ret}(\sigma \heartsuit \tau) = (\mathit{ret}\ \sigma)\triangle(\mathit{ret}\ \tau)$$



$$\begin{tikzcd}
(U, V) \arrow[r, "\triangle"] & W \\
(\sigma, \tau) \arrow[u, "\mathit{ret}"] \arrow[r, "\heartsuit"] & \nu \arrow[u, "\mathit{ret}"']
\end{tikzcd}$$

Data Refinement



42

abstraction function

Python int 42

-1

abstraction function

Python int -1

65535

abstraction function

Python int 65535

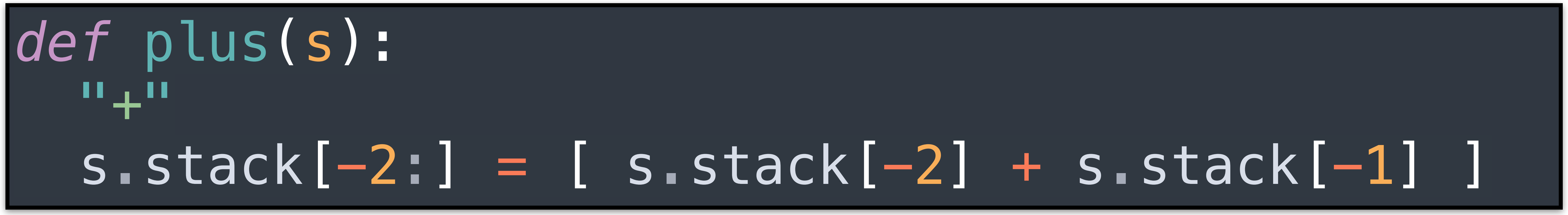
Factorial

```
: fac ( n -- n! )
    ?dup IF dup 1- recurse * exit THEN 1 ;
```

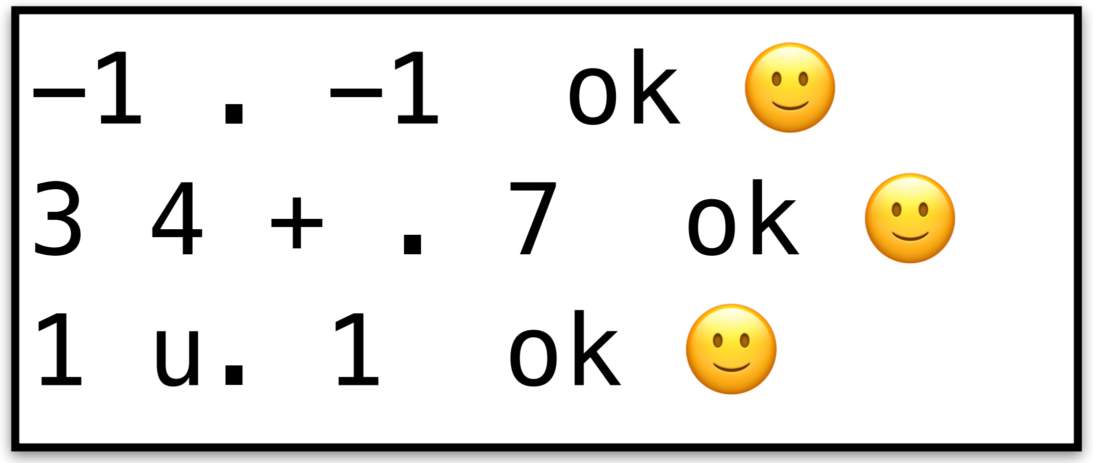
```
10 fac . 3628800  ok 🙂
100 fac .
93326215443944152681699238856266700490715968264381621468592963895217599993229915608
941463976156518286253697920827223758251185210916864000000000000000000000000  ok 🙂
```

Implementing Forth in Python

- Stack
- Primitives



```
def plus(s):
  "+"
  s.stack[-2:] = [ s.stack[-2] + s.stack[-1] ]
```

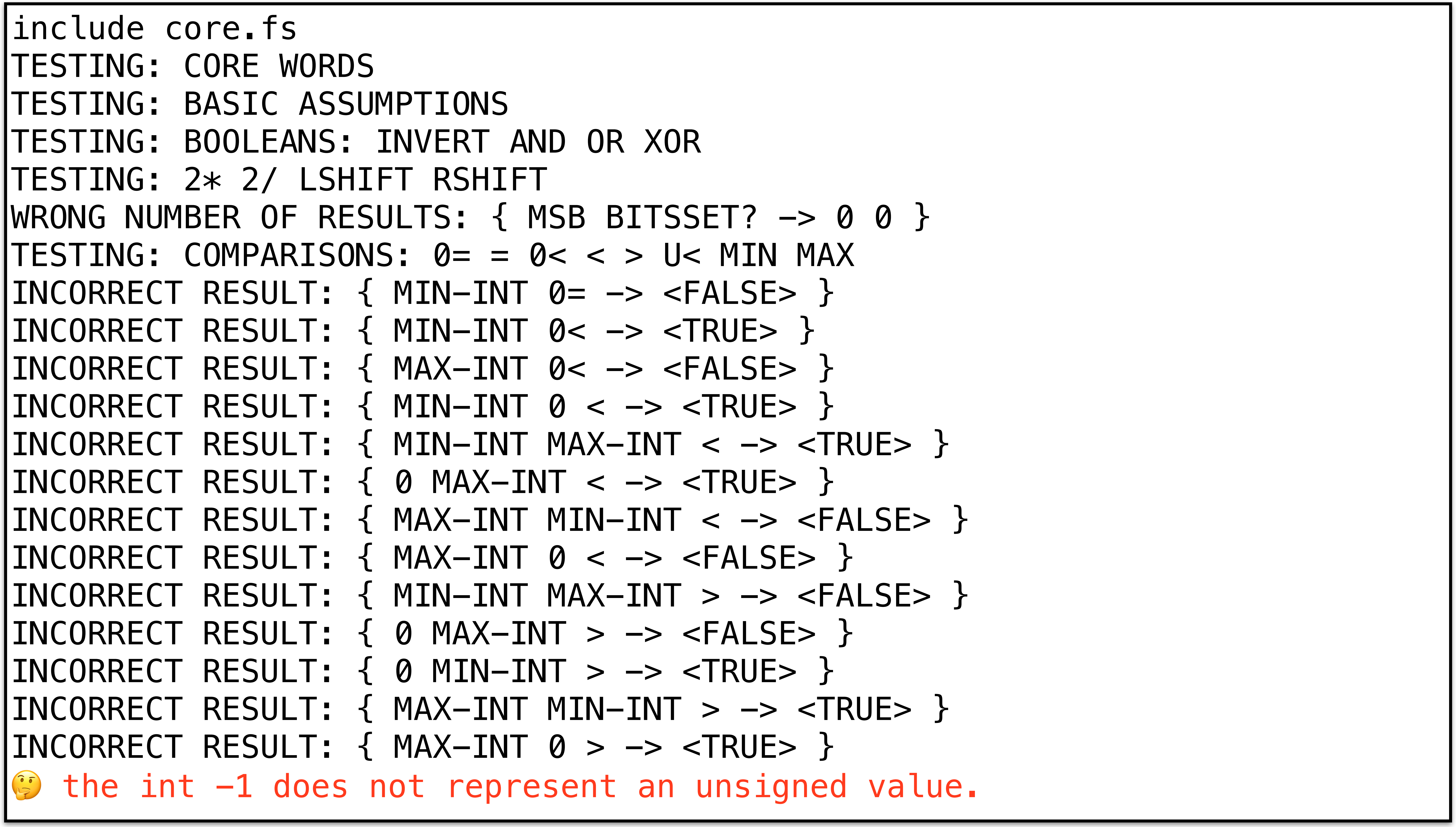


```
-1 . -1  ok 🙂
3 4 + . 7  ok 🙂
1 u. 1  ok 🙂
```

But is it Forth?

Let’s run the Forth-94 core test.

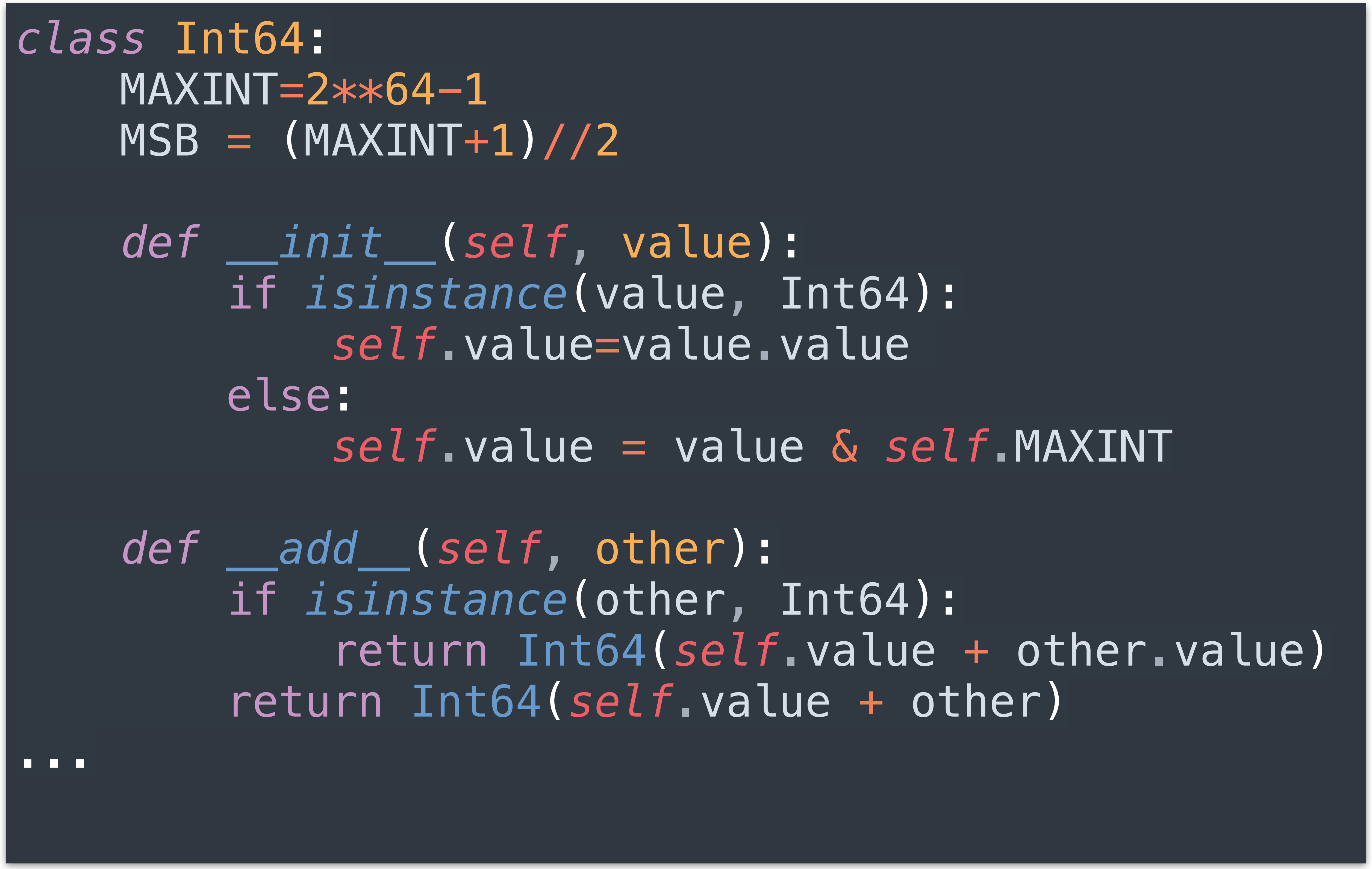
But is it Forth?



```
include core.fs
TESTING: CORE WORDS
TESTING: BASIC ASSUMPTIONS
TESTING: BOOLEANS: INVERT AND OR XOR
TESTING: 2* 2/ LSHIFT RSHIFT
WRONG NUMBER OF RESULTS: { MSB BITSSET? -> 0 0 }
TESTING: COMPARISONS: 0= = 0< < > U< MIN MAX
INCORRECT RESULT: { MIN-INT 0= -> <FALSE> }
INCORRECT RESULT: { MIN-INT 0< -> <TRUE> }
INCORRECT RESULT: { MAX-INT 0< -> <FALSE> }
INCORRECT RESULT: { MIN-INT 0 < -> <TRUE> }
INCORRECT RESULT: { MIN-INT MAX-INT < -> <TRUE> }
INCORRECT RESULT: { 0 MAX-INT < -> <TRUE> }
INCORRECT RESULT: { MAX-INT MIN-INT < -> <FALSE> }
INCORRECT RESULT: { MAX-INT 0 < -> <FALSE> }
INCORRECT RESULT: { MIN-INT MAX-INT > -> <FALSE> }
INCORRECT RESULT: { 0 MAX-INT > -> <FALSE> }
INCORRECT RESULT: { 0 MIN-INT > -> <TRUE> }
INCORRECT RESULT: { MAX-INT MIN-INT > -> <TRUE> }
INCORRECT RESULT: { MAX-INT 0 > -> <TRUE> }
🤔 the int -1 does not represent an unsigned value.
```

Implementing Forth in Python

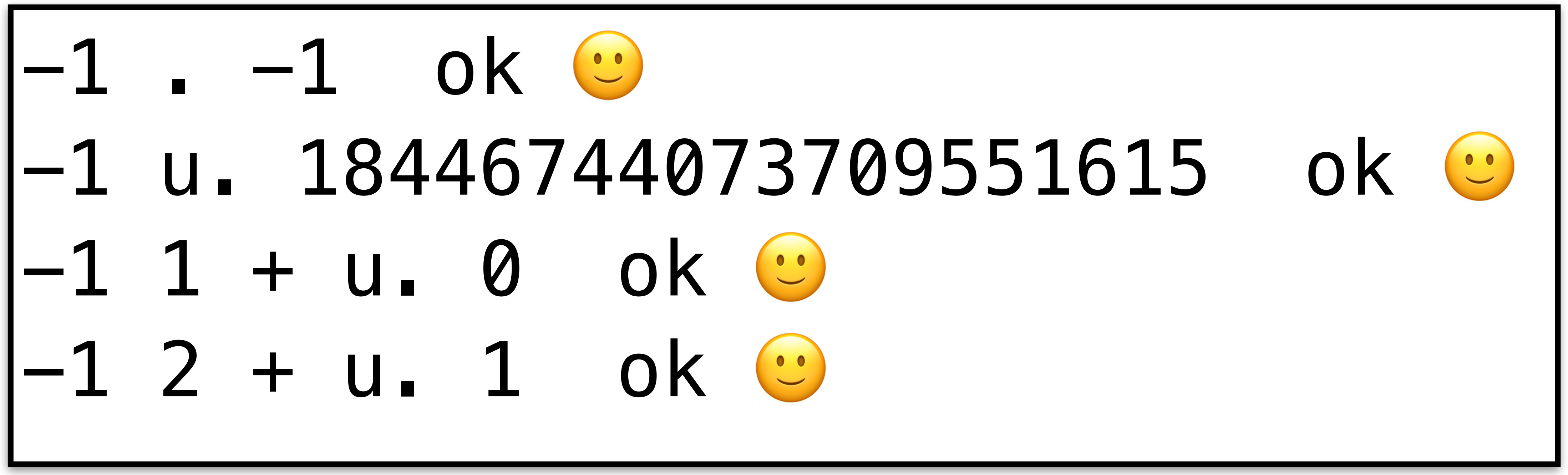
We need to implement cyclic 2’s complement numbers



```
class Int64:
    MAXINT=2**64-1
    MSB = (MAXINT+1)//2

    def __init__(self, value):
        if isinstance(value, Int64):
            self.value=value.value
        else:
            self.value = value & self.MAXINT

    def __add__(self, other):
        if isinstance(other, Int64):
            return Int64(self.value + other.value)
        return Int64(self.value + other)
...
```

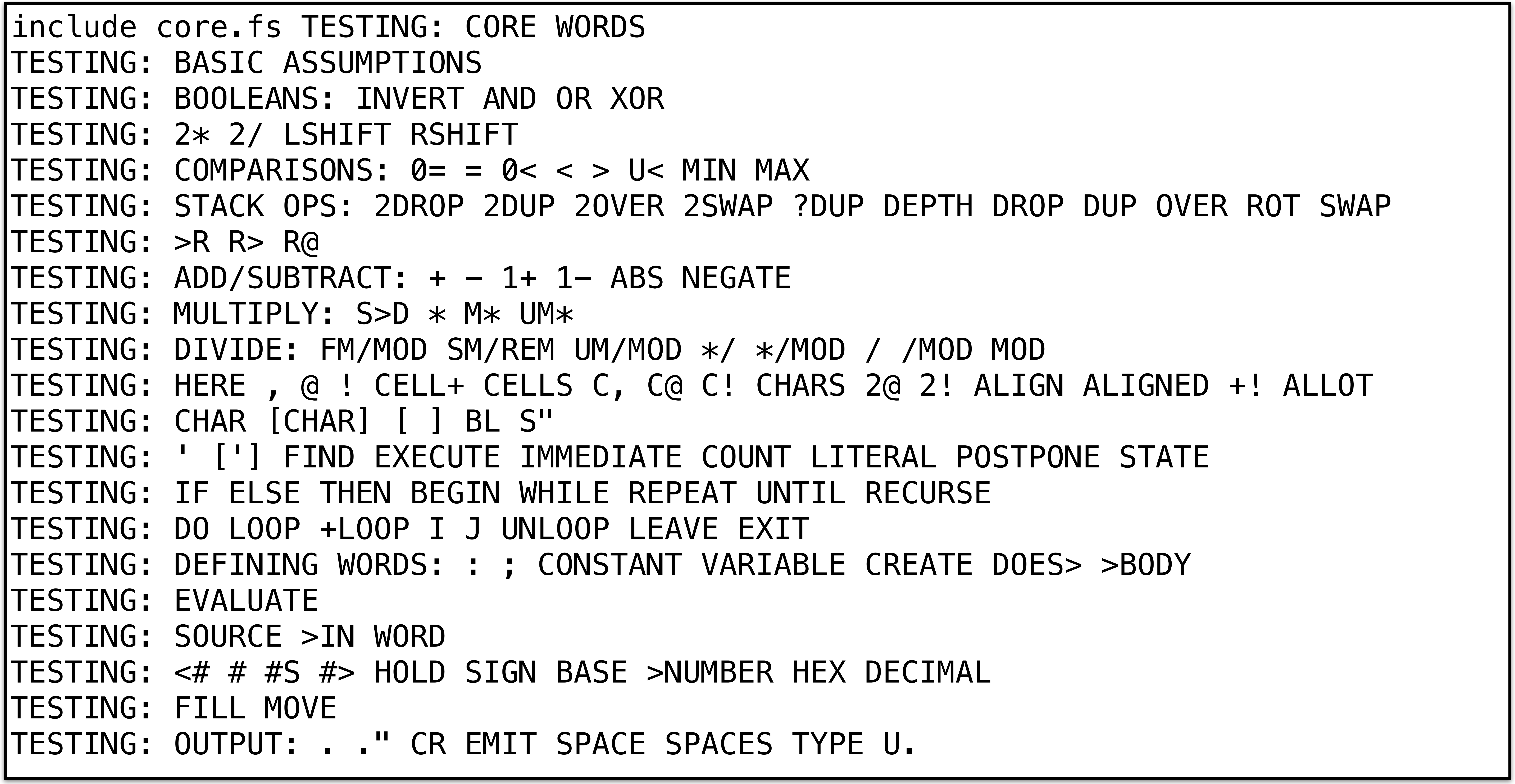


```
-1 . -1  ok 🙂
-1 u. 18446744073709551615  ok 🙂
-1 1 + u. 0  ok 🙂
-1 2 + u. 1  ok 🙂
```

But is it Forth?

Let’s run the Forth-94 core test - again.

But is it Forth?



```
include core.fs TESTING: CORE WORDS
TESTING: BASIC ASSUMPTIONS
TESTING: BOOLEANS: INVERT AND OR XOR
TESTING: 2* 2/ LSHIFT RSHIFT
TESTING: COMPARISONS: 0= = 0< < > U< MIN MAX
TESTING: STACK OPS: 2DROP 2DUP 2OVER 2SWAP ?DUP DEPTH DROP DUP OVER ROT SWAP
TESTING: >R R> R@
TESTING: ADD/SUBTRACT: + - 1+ 1- ABS NEGATE
TESTING: MULTIPLY: S>D * M* UM*
TESTING: DIVIDE: FM/MOD SM/REM UM/MOD */ */MOD / /MOD MOD
TESTING: HERE , @ ! CELL+ CELLS C, C@ C! CHARS 2@ 2! ALIGN ALIGNED +! ALLOT
TESTING: CHAR [CHAR] [ ] BL S"
TESTING: ' ['] FIND EXECUTE IMMEDIATE COUNT LITERAL POSTPONE STATE
TESTING: IF ELSE THEN BEGIN WHILE REPEAT UNTIL RECURSE
TESTING: DO LOOP +LOOP I J UNLOOP LEAVE EXIT
TESTING: DEFINING WORDS: : ; CONSTANT VARIABLE CREATE DOES> >BODY
TESTING: EVALUATE
TESTING: SOURCE >IN WORD
TESTING: <# # #S #> HOLD SIGN BASE >NUMBER HEX DECIMAL
TESTING: FILL MOVE
TESTING: OUTPUT: . ." CR EMIT SPACE SPACES TYPE U.
```

But is it Forth?

```
include core.fs TESTING: CORE WORDS
...
TESTING: OUTPUT: . ." CR EMIT SPACE SPACES TYPE U.
YOU SHOULD SEE THE STANDARD GRAPHIC CHARACTERS:
 !"#$%&'()*+,-./0123456789:;<=>?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
abcdefghijklmnopqrstuvwxyz{|}~
YOU SHOULD SEE 0-9 SEPARATED BY A SPACE:
0 1 2 3 4 5 6 7 8 9
YOU SHOULD SEE 0-9 (WITH NO SPACES):
0123456789
YOU SHOULD SEE A-G SEPARATED BY A SPACE:
A B C D E F G
YOU SHOULD SEE 0-5 SEPARATED BY TWO SPACES:
0  1  2  3  4  5
YOU SHOULD SEE TWO SEPARATE LINES:
LINE 1
LINE 2
YOU SHOULD SEE THE NUMBER RANGES OF SIGNED AND UNSIGNED NUMBERS:
  SIGNED: -8000000000000000 7FFFFFFFFFFFFFFF
UNSIGNED: 0 FFFFFFFFFFFFFFFF
```

But is it Forth? It passes the Forth-94 Core Test Yes!

```
include core.fs TESTING: CORE WORDS
...
YOU SHOULD SEE THE NUMBER RANGES OF SIGNED AND UNSIGNED NUMBERS:
  SIGNED: -8000000000000000 7FFFFFFFFFFFFFFF
UNSIGNED: 0 FFFFFFFFFFFFFFFF
TESTING: INPUT: ACCEPT

PLEASE TYPE UP TO 80 CHARACTERS:
it works

RECEIVED: "it works"
TESTING: DICTIONARY SEARCH RULES
GDX exists  ok 🙂
```

Blending Forth

- But the stack can hold not just (our) numbers.
- It’s implemented as a Python list that can hold any Python object
  - float numbers
  - strings
  - lists and dictionaries
  - method and functions
  - …

pluggable number system

Blending Forth

Python Objects on the Data Stack

```
1.2 3 + .  4.2  ok 🙂
```

```
# 1.2 3   ok 🙂
1.2 3 # .s
0: (IntXX) 3
1: (float) 1.2 ok 🙂
1.2 3 # +  ok 🙂
4.2 # . 4.2  ok 🙂
```

```
# need now  ok 🙂
# now . datetime.datetime(2025, 9, 13, 6, 56, 15, 133285)  ok 🙂
```

Related Work

- *oforth* by Franck Bensusan
  - objects on the stack
  - no standard forth syntax (control structures)
  - similar enough to be called Forth

But is it Forth?

"*If it walks like a duck and*

it quacks like a duck,

then it must be a duck“

Does it?

Demo

Blending Forth

- Why not use Python in first place?
- Forth is concatenative and allows to execute programs interactively step by step.

Blending Forth

Conclusion

- implementing Forth in other languages
- abstraction and representation
- blending Forth

Forth inherits their properties

the heart of implementation

Where to go from here?

Discussion