U*   (u1 u2 -- du)   " u times"

Same as UM*

Included for compatibility with FIG and '79

U.   (n --)   "u dot"

Print N as an UNSIGNED number. The number printed will be positive even if the highest bit, the sign bit, is on.

Definition: : U. (n --) 0 D. ;

```
     -3 U.  ( prints 4,294,967,293 )
```

`Related Words: D.R .HX .R D. .HEX`

U/   (du u -- u-rem u-quotient)

```
     du = double cell unsigned dividend
      u = unsigned divisor
     u-rem = unsigned remainder
     u-quotient = unsigned quotient
```

Divide a double unsigned number by an single precision unsigned number leaving an unsigned remainder and quotient.

Standards: Called U/MOD in '79, called UM/ in '83

`Related Words: */MOD  M/  /  M/MOD CELL/ D2/ 2/ W/ /MOD DU2/  U2/`

U2*   (u -- u*2)   "u 2 star"

U2* multiplies the value on the top of the stack by 2 (which is really a left shift). The MSB is shifted out and thrown away. The LSB gets a 0

Standards: JForth unique

Usage: For left shift of bits, or address manipulations and fast math.

`Related Words: U* DU2* 2* ASHIFT SHIFT`

U2/   (u -- u/2)   "u 2 slash"

U2/ shifts the top value on the stack one place to the right. Least significant bit LSB is shifted out and discarded. 0 is shifted in to the MSB position. This is the same as dividing by 2 .

Standards: JForth unique

Usage: When shift right is needed for low level code writing or for dividing by 2.

`Related Words: CELL/ D2/ 2/  DU2/ ASHIFT SHIFT`

U<   (u1 u2 -- flag)   " u less than"

Unsigned less-than test. True if U1 < U2, otherwise false.

```
     5 -3 .S U<  ( strange but TRUE )
```

`Related Words: U. U* U> > <`

U>   (u1 u2 -- flag)   "u greater than"

Unsigned greater-than test. True if U1 > U2, otherwise false .

`Related Words: U<`

UM*   (u1 u2 -- du)   "u m times"

UM* inputs two single cell unsigned numbers and returns an unsigned double cell product. From file JU:MULTISTANDARD

Standards: '83, called U* in FIG and '79

M* is to be used if a signed product is needed.

Related Words: M* * */ */MOD U2* 2* DU2*

UM/   (du u -- u-rem u-quotient)   "u m slash"

Same as U/

UNMARKFCLOSE (file-pointer --) "un mark f close"

UNMARKFCLOSE is used to remove the file-pointer from the list of files marked to be automatically closed by (QUIT). File-pointers may be added to the list with MARKFCLOSE.

Forth programs usually execute (QUIT) as the final operation in restarting an application, when an error requires restarting (usually indirectly, via QUIT or ?ABORT").

When an application terminates normally, it should use UNMARKFCLOSE to remove the file-pointer from the QUIT list, just prior to closing the file if it was placed there by MARKFCLOSE.

For additional information, see the appendix "JForth File Manager".

Standard: JForth Unique

```
: EXAMPLE ( <filename> -- )
   FOPEN ?DUP         \ get the filename, did it open?
   IF   DUP MYFILE !  \ yes, save pointer in my variable
       MARKFCLOSE     \ set it to auto-close at quit
       RUN-MY-PROGRAM \ do whatever I want with the file
       MYFILE @ DUP   \ need to do TWO thing to the pointer:
       UNMARKFCLOSE   \ remove it from the QUIT list...
       FCLOSE         \ and close the file
   ELSE ." can't open the file "
   THEN
;
```

Related Words: MARKFCLOSE QUIT ABORT   ABORT"

UNMARKFREEBLOCK (mem-block --) "un mark free block"

UNMARKFREEBLOCK is used to remove the memory-block from the list of memory-areas marked to be automatically freed by (QUIT). Memory-blocks may be added to the list with MARKFREEBLOCK. When an application terminates normally, it should use UNMARKFREEBLOCK to remove it from the QUIT list just prior to freeing the memory if it was placed there by MARKFREEBLOCK.

For additional information, see the appendix "JForth Memory Manager".

Standard: JForth Unique

```
: EXAMPLE ( - )
   0 1024 ALLOCBLOCK ?DUP \ alloc 1K, did it allocate?
   IF  DUP MYMEM !       \ yes, save pointer in my variable
       MARKFREEBLOCK     \ set it to auto-free at quit
       RUN-MY-PROGRAM    \ do whatever I want to do with the
mem
```

```
       MYMEM @ DUP       \ need to do TWO thing to the
pointer...
       UNMARKFREEBLOCK   \ remove it from the QUIT list...
       FREEBLOCK         \ and free the memory
   ELSE ." can't allocate memory "
   THEN
;
```

Related Words: MARKFREEBLOCK QUIT ABORT   ABORT"

UNRAVEL (--)

Analyses the return stack and prints a list of the words currently executing. Often used when reporting errors to find out what word caused the error.

```
INCLUDE? UNRAVEL JU:UNRAVEL
: STOPIT  ( -- ) ." Error!" UNRAVEL QUIT ;
: FOO  23 45 + . STOPIT ;
: ZAPPER  ." Test" CR FOO ;
ZAPPER
```

If you do this you should see the names of ZAPPER, FOO and STOPIT in the list. To install UNRAVEL as part of QUIT, use START.UNRAVEL.QUIT. To remove it, use STOP.UNRAVEL.QUIT.

Related Words: R@ RDEPTH >NAME

UNSMUDGE   (--)

Clear the smudge bit in the LATEST definition so FIND will locate it. Same as REVEAL.

Standards: JForth

Related Words: SMUDGE REVEAL

UNTIL   (flag --)

Will loop back to BEGIN if FLAG = FALSE. Otherwise continue.

Standards: Run time action is '79, '83, FIG.

```
Compile time action is JForth unique.

: YAKYAK
    BEGIN ." Hit key to stop!" CR
        ?TERMINAL   ( leave flag )
    UNTIL KEY DROP  ( clean up character hit )
;
```

At compile time: (address-of-loop-body begin_flag --)

BEGIN_FLAG is for compiler security. ADDRESS-OF-LOOP-BODY is address to branch to if FALSE. UNTIL puts a ?BRANCH into the dictionary.

Related Words: BEGIN WHILE REPEAT UNTIL-NOT END WHILE-NOT DO LOOP AGAIN

UNTIL-NOT   until not

Same as UNTIL only stays in the loop under the opposite condition as UNTIL . Simple concatenation of NOT and UNTIL .

Standards: NOT COVERED. JForth EXTENSION.

```
    : EXAMPLE  ( -- )
        5 BEGIN  ." loop-body" 1- DUP 0>
          UNTIL-NOT DROP
    ;
```

Related Words: UNTIL BEGIN

UP!   (addr --)   "u p store"

UP! is not called in JForth V2.0.

Standards: JForth unique

Related Words: UP@

UP@   (-- addr)   "u p fetch"

Fetch the user area pointer address. All user-variables (storage area accessible by only the owning task) are indexed from this JForth-relative base pointer.

Standards: JForth unique

Related Words: USER USP US> >US UP!

UPPER   (address count --)

Convert a string at address that is count characters long to upper-case.

```
    : GETPASSWORD  ( -- flag )
        BL WORD DUP COUNT UPPER  ( convert mixed case to upper)
        " SECRET-PASSWORD"  $=
    ;
```

Related Words: MAKEUCASE TEXT=? $= OUTPUT-CASE NOCASE HICASE LOCASE TOUPPER

UPPERC@   (addr -- upper-case-character)

UPPERC@ is used to fetch a character from addr, converting it in the process to upper-case if it is alphabetic. If the ASCII character is not between lower-case a to z, it is not changed. The contents of addr are not affected.

Related Words: C@ UPPER MAKEUCASE LWORD

US-DEPTH   (-- user-stack-depth-in-cells)   "u s depth"

US-DEPTH returns the number of cells that have been pushed on the user stack, analogous to DEPTH.

Related Words: US> US@ USP US-PICK DEPTH

US-PICK   (n -- nth-item-of-user-stack)   "u s pick"

US-PICK picks the nth cell off user stack, much like PICK does for the data stack.

Standards: JForth unique.

Related Words: US> US@ USP US-PICK

US>   (n --US--)   (-- n)   "u s from"

This is the user stack equivalent of R> . It pops one item off the user stack onto the data stack .

Standards: JForth unique.

`Related Words: USP US> >US INIT-USP`

`US@   ( -- n )  ( n --US-- n )`

US@ copies the top of the user stack onto the stack . This is the user stack equivalent of R@ .

Definition: : US@ (-- N) USP @ @ ;

Standards: JForth unique.

Usage: Control of an extra stack .

`Related Words: USP US> >US INIT-USP`

`USER`

Compile time: (name --IN-) Eats name for header from input stream.

Run time: (-- addr) Leaves address of user-owned data area.

USER is a defining word similar to VARIABLE. It defines a data type that is only accessible to the task that created it. This is done by providing each task with it's own USERAREA, from which storage space is allocated.

Each user area contains a counter (called USER#) which is used to point to the next available cell to be allotted. Each time USER is invoked:

1. A name is created in the dictionary, from that which follows the USER declaration.
2. The current value of USER# is fetched, installed into the just-created definition, then incremented by 4 (1 cell) and placed back in USER#.

Note that when the user-variable later executes, it will place the address of the user-specific data area on that users stack. This address will be equal to the user offset stored in the definition, PLUS the address of that users user area (gotten by UP@).

Standards: JForth UNIQUE. FIG USER expects an offset (n -- <name>).

```
USER MY-VAR
23 MY-VAR !
```

`Related Words: USER# UP@ #U`

`USER#   ( -- user-addr )`

This user-variable is used by the USER facility to track the allotment of the user area. See USER.

`USERCLEANUP   ( -- )`

See the chapter on CLONE.

`USP@  ( -- addr-of-top-of-user-stack )  "u s p fetch"`

Returns the address of the top of the user stack, as SP@ does for the data stack.

Standards: JForth unique.

`Related Words: USP US> >US INIT-USP`

`VALID-NAME?   ( possible-NFA -- seems-valid-flag )`

VALID-NAME? takes a possible NFA and returns a true if it appears to be a valid header. It conducts a thorough test on the passed in address, testing if it physically meets the qualities of a real name field.

Standard: JForth Unique.

`Related Words: ID.  >NAME`

`VALUE   ( n <name> -- )`

Creates a self fetching variable. VALUEs act like a constant except you can change them using the word -> .

```
INCLUDE? VALUE JU:VALUE
765 VALUE MY-VALUE
MY-VALUE .    ( will print 765 )
19 -> MY-VALUE    ( change value )
MY-VALUE .    ( will print 19 )
```

`Related Words: -> VARIABLE CONSTANT CREATE { DOES>`

`VARIABLE   ( <name> -- )`

Define a named 32 bit storage location. You can store values into this area for later use. The initial value is zero. When the word is executed it leaves its address on the stack.

Standards: '79 '83. fig expected an initial value for the variable at compile time.

```
VARIABLE MY-VAR
MY-VAR .   ( print address of MY-VAR )
73 MY-VAR  !  ( set value )
MY-VAR @ .    ( prints 73 )
```

`Related Words: USER CONSTANT CREATE VALUE`

`VLINK>VLATEST   ( voc-link-addr -- addr-of-name )`

VLINK>VLATEST converts a VOC-LINK, vocabulary link pointer, address, to the address of the NAME of the latest definition in that vocabulary. See section on Vocabularies for more information.

Standards: JForth UNIQUE

`Related Words: VOC-LINK VLINK>VLATEST  VLATEST>VLINK`

`VLINK>'   ( voc-link-addr -- cfa-of-vocabulary )`

VLINK>' converts a VOC-LINK, vocabulary link pointer, address to the CFA address for that vocabulary. See section on Vocabularies for more information.

Standards: JForth UNIQUE

```
VOC-LINK @ VLINK>' >NAME ID.
```

`Related Words: VOC-LINK VLINK>VLATEST  VLATEST>VLINK`

`VLATEST>VLINK  ( nfa-of-vocabulary -- voc-link-addr )`

VLATEST>VLINK converts the address of the NAME of a vocabulary to the VOC-LINK address, then vocabulary link pointer address, of that vocabulary. See section on Vocabularies for more information.

Standards: JForth UNIQUE

`Related Words: VOC-LINK VLINK>VLATEST  VLATEST>VLINK`

`VLIST   ( -- )`

See WORDS.

Related Words: WORDS ALL-WORDS WORDS-LIKE ORDER

VOC-LINK (-- addr)

VOC-LINK is a USER variable pointing to the linked list of vocabularies in reverse order of definition. See section on Vocabularies for more information.

Standards: Not covered . But common in FIG '79 '83 systems.

Related Words: VOCABULARY ORDER WORDS ALL-WORDS VLINK>VLATEST VLINK> VLATEST>VLINK

VOCABULARY (<name> --)

VOCABULARY is a defining word that creates new vocabularies. Vocabularies are used to collect a group of word that relate to a specific area, eg. music, robotics. You can turn on or off a vocabulary to make these words available or unavailable. See section on Vocabularies for more information.

Standard: '83.

Related Words: VOCABULARY ORDER WORDS ALL-WORDS VLINK>VLATEST VLINK> VLATEST>VLINK

W! (n even-addr --) "w store"

Store lower 16 bits of N to an even address.

Related Words: W@ W, W->S

W, (n --) "w comma"

Add the lower 16 bits of N to the dictionary at HERE. Same as , only a 16 bit value.

```
\ Create a table of 16 bit values
CREATE TABLE16  23 W, 497 W, 71 W,
TABLE16 1 2* + W@ .  ( print 497 )
```

Related Words: , ALLOT HERE DP CFA,

W->S (w -- n)

Sign extend a 16 bit value W to a 32 bit value N. See B->S .

```
VARIABLE MY-VAR
HEX -27 MY-VAR W!
MY-VAR W@ .   ( not -27 !!! )
MY-VAR W@ W->S . ( print -27 )
```

Related Words: S->D B->S W@ W!

W/ (n w -- n/w) "w slash"

```
n = dividend ( one cell or 32 bits in size)
w = divisor (one half cell or 16 bits in size)
n/w = 16 bit quotient in lower half of cell
```

32 bit by 16 bit divide. Several times faster than / because it can use the 68000 hardware divide. Does not sign extend 16 bit result n/w . Dividing by zero will cause a trap. The remainder is discarded.

```
DECIMAL  50 4 W/ . ( displays 12 )
```

Related Words: / U/ D/ 2/ ASHIFT /MOD W->S

W@ (even-addr -- w) "w fetch"

Fetch a 16 bit value from the address given. W@ will not sign extend the word. Use W->S to sign extend.

Standards: same as standard @ in most 16 bit Forths.

```
CREATE SHORTVAR -34 W,
SHORTVAR W@  W->S .
```

Related Words: W! W, W->S

WARNING" (flag <string"> --)

Types the quote delimited string if flag is true.

```
: TEST.OVERFLOW
    DEPTH 100 >
    WARNING"  Lots of stuff on stack!" ;
```

Related Words: ABORT" QUIT ."

WHAT'S (<name> -- cfa)

Return the CFA that a deferred word is set to execute. This is useful when you want to set a deferred word temporarily then restore its original value.

Standards: JForth UNIQUE

```
WHAT'S EMIT  >NAME ID.
```

Related Words: IS DEFER GLOBAL-DEFER

WHEN-SCANNED (nfa --)

WHEN-SCANNED is a DEFERed word that is executed by SCAN-VOC and SCAN-ALL-VOCS for each word that they examine. It is used to write WORDS and ALL-WORDS and WORD-LIKE . See SCAN-ALL-VOCS .

Standard: JForth UNIQUE.

Related Words: WHEN-VOC-SCANNED SCAN-VOC SCAN-ALL-VOCS

WHEN-VOC-SCANNED (voc-addr --)

WHEN-VOC-SCANNED is a DEFERed word that is executed by SCAN-ALL-VOCS and SCAN-ALL-VOCS for each vocabulary that is scanned. It is used to write WORDS and ALL-WORDS and WORD-LIKE . See SCAN-ALL-VOCS .

Standard: JForth UNIQUE.

Related Words: WHEN-SCANNED SCAN-VOC SCAN-ALL-VOCS WORDS ALL-WORDS

WHILE (flag --)

Used in the form:

```
BEGIN xxxx WHILE yyyy REPEAT
```

The code XXXX must leave a flag. If the flag is TRUE then YYYY will be executed. Otherwise, execution will jump to the code after REPEAT.

```
: BIGMISTAKE  ( -- , loop while answering yes )
    BEGIN ." Launch a missile?" Y/N
    WHILE ." Bombs away!" cr
    REPEAT ;
```

`Related Words: BEGIN REPEAT WHILE-NOT UNTIL`

WHILE-NOT (flag --)

Simple concatenation of NOT and WHILE . Instead of

```
     BEGIN xxxx NOT WHILE yyyy REPEAT
```

you can use

```
     BEGIN xxx WHILE-NOT yyy REPEAT
```

which is faster.

`Related Words: BEGIN REPEAT WHILE UNTIL NOT`

WILLGET (-- , <modulename> <wordname>)

See the chapter on MODULES.

WINDOWSTRING (-- addr)

Returns the address of the string used to open the JForth window. Change this string and SAVE-FORTH if you want to resize or change the title of the default JForth window.

```
     WINDOWSTRING 60 DUMP
```

WITHIN? (N lower upper -- flag)

WITHIN? compares a number to an upper and lower bounds. Returns a TRUE if N is both greater than or equal to lower and less than or equal to upper.

```
     20  -5 100 WITHIN? . ( will be true )
     20  20 30  WITHIN? . ( true )
     20  21 30  WITHIN? . ( false )
```

`Related Words: >= <=`

WMOVE (from-addr to-addr #words --)

WMOVE is similar to MOVE but it moves n words instead of bytes, and requires the to and from addresses to be even (on the 68000). This is faster than CMOVE.

Standard: JForth unique.

`Related Words: MOVE CMOVE CMOVE> $MOVE`

WORD (delim-char <text> -- $addr)

Reads one word from the input stream using the character on the stack as the delimiter. An ASCII blank is usually used. WORD moves the text string <TEXT> to the dictionary address $ADDR with the count in the first byte, and leaves $ADDR on top of the stack. $ADDR typically equals HERE. WORD will ignore leading occurrences of the delimiter character.

WORD will update the TIB pointer >IN to point to the character following the text.

If SKIP-WORD has been called then WORD will ignore the following text and simply return the address of HERE . This can be used to pass a string to a word that normally gets its text from input. See SKIP-WORD for more info.

Here is an example of using WORD to define the comment operator.

```
     : (  ASCII ) WORD DROP ;
```

`Related Words: HERE EXPECT TEXT KEY UNWORD >IN FIND SKIP-WORD`

WORD-SWAP   (abcd -- cdab)

Swap upper and lower 16 bit parts of a 32 bit number. A B C and D in the diagram are bytes.

```
    HEX 12345678 WORD-SWAP .  ( print 56781234 )
```

Standards: JForth unique

`Related Words: W@ W! BYTE-SWAP`

WORDS   (--)

WORDS causes a printout to the current device of all JForth words in the CONTEXT vocabulary.

Standards: '83. Called VLIST in FIG and '79 . Both are included in JForth.

`Related Words: ALL-WORDS VLIST ORDER VOCABULARY SCAN-VOCS`

WORDS-LIKE (<partial-name> --)

Searches dictionary for words whose name contains the partial name given.

```
    WORDS-LIKE EMIT
```

will print FEMIT FLUSHEMIT (EMIT) plus any other words containing "EMIT". WL is a short alias for WORDS-LIKE. WORDS-LIKE is also attached to the function key <F6>.

`Related Words: FILE? WORDS`

X!   (nx ... n2 n1 addr x --)

Store x cells from stack to memory starting at address ADDR. Cells are transferred so that the most significant cell, nx, is at addr, with less significant cells in ascending memory. Works on ODD or EVEN addresses.

Standards: JForth UNIQUE

```
    VARIABLE MY-DVAR CELL ALLOT
    9. MY-DVAR 2 X!  (  stores the double number 9. in MY-DVAR )
```

`Related Words: X@ X>R XR> D!`

X>R   (nx...n2 n1 x --)   (--R-- nx...n2 n1)

Transfers x cells from the data stack to the return stack .

Standards: JForth UNIQUE.

```
    : EXAMPLE  2  46 88 99  3 X>R  . ( displays 2 )
        3 XR>  3   XDROP ( empty stack ) ;
```

`Related Words: XR> XR@ XDROP XDUP`

X@   (addr x -- nx...n2 n1)   "x fetch"

Fetch x cells (nx...n2 n1) from memory starting at address ADDR to the data stack. ADDR is address of nx and addresses are incremented by one cell address space to fetch each new N.   See X! .

Standards: JForth UNIQUE

```
    CREATE FOO  11 ,  22 ,   33 ,
    FOO 3 X@  .S   ( 33 22 11 on stack )
```

`Related Words: XR> X>R XR! XDUP XDROP X!`

XDROP (nx...n2 n1 x --)

Drop X cells from the data stack. This is very fast regardless of the number of items dropped.

Standards: JForth UNIQUE

```
    11 22 33 44   4 XDROP  ( empty stack )
```

`Related Words: XR> X.R XR@ XDUP`

XDUP (nx ...n2 n1 x -- nx...n2 n1 nx...n2 n1)

Duplicate the top x cells on the stack in order and push the whole set on top of the stack .

Standards: JForth UNIQUE

```
    11 22 33  3 XDUP .S ( print: 11 22 33 11 22 33 )
```

`Related Words: XR> X>R XR@ XDROP`

XOR (n1 n2 -- n3)

Perform bit by bit EXCLUSIVE OR of N1 and N2. Here is the truth table for an EXCLUSIVE OR operation on two bits:

```
    A   B  ->  A.XOR.B
    0   0         0
    1   0         1
    0   1         1
    1   1         0
```

(Performing an XOR with -1 has the effect of reversing the value of the bits in a number.)

Standards: '79 '83 fig

```
    BINARY 0011 1010 XOR .  ( print 1001 )
    010101 111111 XOR . ( print 101010 )
```

`Related Words: OR AND NOT`

XPICK (..... N X -- Xcells-from-Ncell-deep)

XPICK takes the cells N deep and the number of cells X to pick and picks them from the stack. Zero is the 1st cell.

Standards: JForth UNIQUE

```
    99 88 77 66  (  -- 99 88 77 66 )
    1 2 XPICK .S (  -- 99 88 77 66 88 77 )
```

`Related Words: R> >R X>R  XR@ XDROP XDUP PICK RPICK`

XR> (x -- nx ...n2 n1 n0)

```
    ( nx ...n2 n1 n0 --R-- )   "x r move"
```

Move x cells (n1 n2 up through nx) from the return stack to the data stack in order.

Standards: JForth UNIQUE

`Related Words: R> >R X>R  XR@ XDROP XDUP`

XRDROP (x --)   "x r drop"

XRDROP drops X items off the return stack.

Standards: JForth UNIQUE

```
    : EXAMPLE   4 >R  38 >R  30 >R   3 XRDROP ;
```

Related Words: R> >R X>R XR@ XDROP XDUP

Y/N   (-- flag , or quit)   "y slash n"

Y/N is a word that displays the message "...Yes, No or Quit (y/n/q)?" then waits for 1 of 3 keys: y n or q in either upper or lower case. If q it will call QUIT, if y it will return a true flag, and if n it will return a false flag (zero).

Standards: JForth UNIQUE

[(--)   "open bracket"

Suspends the operation of the compiler, causing following input to be INTERPRETed, rather than compiled. Usually paired with the] character, which re-invokes the compiler.

Standard: fig '79 '83

Here is an example where the calculation of a constant is done at compile time to reduce the run time burden.

```
: EXAMPLE   [ HEX 7A43 17 / DECIMAL ] LITERAL +
   [ ." Prints right now!!" ] ;
```

Related Words:] STATE IMMEDIATE INTERPRETING? COMPILING? LITERAL

[COMPILE]   (<name> --)   "bracket compile"

Forces the compilation of the next word in the input stream even if it is an IMMEDIATE word.

```
: VERBOSE.LITERAL   ( N -- , compile N )
    DUP ." N = " . CR
    [COMPILE] LITERAL  ( execute LITERAL later )
; IMMEDIATE
: FOO [ 234 9 / ] VERBOSE.LITERAL . ;
FOO .
```

Related Words: [] CFA

[]   (object --)

Used to indicate late binding in ODE. See chapter on ODE.

```
: PRINT.OBJECT ( object -- )
    PRINT: [] ;
```

Note: This is also sometimes used as a shorthand for [COMPILE] whose use is preferred.

Related Words: [COMPILE]

\   (<rest-of-line> --)   "back slash"

Ignores the rest of the line during compiling or interpreting. Treat it as a comment.

```
\ This is a comment.
." DO THIS!"  \ But not this!
```

Related Words: ( )

]   (--)   "close bracket"

Enter compile mode by setting STATE to TRUE. Used with [to temporarily suspend and resume compilation.

This is sometimes entered accidentally because of it's proximity to the carriage return key. Just enter [to get back your OK prompt.

Standard: fig '79 '83

```
    : example   [ 2 4 + . ]  [ 9 ] literal . ;
    \ Will display 6 when compiled and 9 when run.
```

Related Words: [STATE IMMEDIATE INTERPRETING? COMPILING?