# Quackery Quick Reference

## NAMES

### stack

```
dup ( a --> a a )
drop ( a --> )
swap ( a b --> b a )
rot ( a b c --> b c a  )
unrot ( a b c --> c a b )
over ( a b --> a b a )
nip ( a b --> b )
tuck ( a b --> b a b )
2dup ( a b --> a b a b )
2drop ( a b --> )
2swap ( a b c d --> c d a b )
2over ( a b c d --> a b c d a b )
pack ( a b 2 --> [ a b ] )
unpack ( [ a b ] --> a b )
dip ** ( a b c --> a**b c )
```

### arithmetic

```
1+ ( a --> a+1 )
+ ( a b --> a+b )
negate ( a --> -a )
abs ( a --> |a| )
- ( a b --> a-b )
** ( a b --> a**b )
/mod ( a b --> a/b remainder )
/ ( a b --> a/b )
mod ( a b --> remainder of a/b )
```

### comparison

```
= ( a b --> a=b )
oats ( a b --> "OneAndTheSame" )
!= ( a b --> a≠b )
< ( a b --> a<b )
> ( a b --> a>b )
min ( a b --> smaller of a & b )
max ( a b --> larger of a & b )
clamp ( a b c --> a if c<a,
                  b if c>b else c)
within ( a b c --> a<c<b )
$< ( a b --> a<b a&b = strings )
$> ( a b --> a>b a&b = strings )
```

### boolean

```
true ( --> 1 )
false ( --> 0 )
not ( a --> not(a) )
and ( a b --> a and b )
nand ( a b --> a nand b )
or ( a b --> a or b )
xor (a b --> a xor b)
```

### bitwise

```
~ ( a --> bitwise not(a) )
& ( a b --> bitwise a and b )
| ( a b --> bitwise a or b )
^ ( a b --> bitwise a xor b )
<< ( a b --> leftshift a by b )
>> ( a b --> rightshift a by b )
bit ( a --> bit a=1, rest=0 )
64bits ( a --> a & (2**64)-1 )
64bitmask ( --> (2**64)-1 )
rot64 ( a b --> rotate a by b )
```

### random

```
random ( a --> b, in 0 to a-1 )
randomise ( --> )
shuffle ( a --> reordered nest )
```

### ancillary stacks

```
[ stack ] is ancillary-stack
put ( a stack --> )
take ( stack --> a )
release ( stack --> )
share ( stack --> a )
replace ( a stack --> )
move ( stack-a stack-b --> )
tally ( n stack -- )
temp  ( --> stack )
base ( --> stack )
decimal ( == "10 base put" )
```

### control flow

```
done ( jump to ] )
again ( jump to [ )
if ( skip one item if ToS false )
iff ( skip two items if false )
else ( skip one item )
until ( jump to [ if ToS false )
while ( jump to ] if ToS false )
```

### meta control flow

```
]done[ ]again[ ]if[ ]iff[ ]else[
]do[ ]'[ ]this[
( grant control flow properties
 to calling nest. Also ]bailby[ )
```

### self-reference

```
' x ( --> x )
do ( x --> , do x )
this ( --> enclosing-nest )
[ table 10 11 12 ] ( 0 --> 10 )
recurse ( this do )
decurse ( recurse, limit=depth )
depth ( --> stack )
```

### iteration

```
times x ( n --> , do x n times )
i ( --> n , descending in x )
i^ ( --> n , ascending in x )
step ( n -->, i, i^ step size )
refresh ( times count = 0 )
conclude ( times count = limit )
```

### text

```
space ( --> 32 )
carriage ( --> 13 )
upper ( char --> CHAR )
lower (CHAR --> char )
printable ( char --> boolean )
qacsfot ( char --> n )
digit ( n --> digit )
char->n (digit --> n or -1 )
number$ ( n --> $ )
$->n ( numeric$ --> n boolean )
trim ( $ --> $ )
nextword ( $ --> $ $ )
```

### nests

```
nest$ ( $ --> [ )
[] ( --> [ ] )
nested ( a --> [ a ] )
join ( a b --> [ a b ] )
split ( [ a b c ] 2
                --> [ a b ] [ c ] )
size ( [ a b c ] --> 3 )
peek ( [ a b ] 1 --> b )
poke ( 2 [ 1 ] 0 --> [ 2 ] )
pluck ( [ a b ] 1 --> [ a ] b )
stuff ( a [ b ] 1 --> [ b a ] )
behead ( [ a b ] --> [ b ] a )
of ( [ a ] 3 --> [ a a a ] )
reverse ( [ a b ] --> [ b a ] )
reflect ( [ [ a b ] c ]
                --> [ c [ b a ] ] )
copy ( a --> a' )
```

### search

```
makewith ( witheach, but with x
           on top of stack )
witheach x ( a --> do x to each
                 item in nest a )
matchitem ( findwith, but with
            comparison and
            cleanup on stack )
findwith [ over = ] drop == find
find ( 12 [ 10 11 12 ] --> 2 )
findseq ( [ 2 3 ] [ 1 2 3 ] --> 1
found ( result nest --> bool )
```

### sort

```
sortwith < ( [ 1 3 2 ]
                --> [ 3 2 1 ] )
sort ( [ 1 3 2 ] --> [ 1 2 3 ] )
sort$ ( == sortwith $> )
```

### I/O

```
input ( prompt$ --> $ )
sp ( --> , print space )
cr ( --> , carriage return )
emit ( char --> , print char )
echo$ ( $ --> , print string )
wrap$ ( [$$] n --> print $s )
echo ( x --> , print x )
ding ( --> , sound sys alert )
putfile ( $ file$ --> bool )
takefile ( file$ --> $ bool )
sharefile ( file$ --> $ bool )
releasefile ( file$ --> bool )
replacefile ( $ file$ --> bool )
loadfile (file$ --> )
filepath ( --> stack )
```

### exceptions

```
protect ancillary-stack ( --> )
backup ( n --> )
2 ]bailby[ ( == ]done[ ]done[ )
bail ( --> )
bailed ( --> boolean )
message ( $ --> )
history ( --> stack )
backupwords ( --> )
restorewords ( --> )
releasewords ( --> )
protected ( --> stack )
fail ( problem$ --> )
```

### to-do stacks

```
to-do ( --> stack )
new-do (  stack --> )
add-to ( n*items action n s --> )
now-do ( stack --> )
do-now ( stack --> )
not-do ( stack --> )
```

### internal

```
quid ( x --> n )
operator? ( x --> boolean )
number? ( x --> boolean )
nest? ( x --> boolean )
name? ( x --> boolean )
builder? ( x --> boolean )
immovable ( --> )
```

### dictionaries

```
names ( --> nest-of-strings )
actions ( n --> x )
builders ( --> nest-of-strings )
jobs ( n --> x )
namenest ( --> stack )
actiontable ( --> ' actions )
buildernest ( --> stack )
jobtable ( --> ' jobs )
```

### building

```
build ( $ --> nest )
quackery ( == build do )
unbuild ( nest --> $ )
quackify ( x --> $ )
unresolved ( --> )
nesting ( --> stack )
```

```
time ( -- ms_since_epoch )
```

### development

```
empty ( * --> )
words ( --> )
shell  ( --> )
leave ( --> )
stacksize ( --> n )
echostack ( --> )
nestdepth ( --> n )
return ( --> nest )
return$ ( --> $ )
echoreturn ( --> )
python ( --> $ )
```

## BUILDERS

```
[ and ] - enclose a nest
x is name - makes a name
x builds name - makes a builder
forward is name - makes a forward
                  reference
x resolves name - resolves a
                 forward reference
char c - makes a character
         literal
$ "string" - makes a string
             literal
say "string" - makes code to echo
               a string literal
hex 7FF - makes a hex literal
x now! - does x immediately
x constant - does x immediately
             and makes a literal
( and ) - enclose a comment
```