Če funkcije in operacije postavite v en izraz, bodo funkcije izračunane pred operacijami. To pravilo lahko spremenite s pomočjo oklepajev. Oglejmo si dva primera, ki se razlikujeta samo po oklepajih, izračunavanja pa imajo popolnoma drugačen vrstni red (čeprav sta rezultata enaka, kar se včasih dogaja):

```
LEN "Fred"+LEN "Bloggs"          LEN ("Fred"+"Bloggs")
4 + LEN "Bloggs"                 LEN (FredBloggs")
4 + 6                            LEN "FredBloggs"
1Ø                               1Ø
```

Oglejmo si še nekaj funkcij:

STR$ pretvarja števila v stringe: njen argument je število, rezultat pa string, ki se pojavi na ekranu, če to zahtevamo z ukazom PRINT. Bodite pozorni na ime funkcije, ki se končuje na $, da bi se pokazalo, da je njen rezultat string.

Rekli bi lahko na primer:

```
LET a$ = STR$ 1e2
```

kar bi imelo popolnoma isti efekt kot:

```
LET a$ = "1ØØ"
```

Ali pa bi rekli:

```
PRINT LEN STR$ 1ØØ.ØØØØ
```

in dobili odgovor: 3, ker je STR$ 1ØØ.ØØØØ = "1ØØ"

VAL deluje obratno kot STR$, saj pretvarja stringe v števila.

Na primer:

```
VAL "3.5" = 3.5
```

Vendar je VAL obraten od STR$ samo do neke mere, saj če vzamete katerokoli število, pa uporabite STR$ ter nato delujete z VAL, pa zopet s STR$, se ne boste vedno vrnili na začetno število.

VAL je precej močna funkcija saj string, ki je njen argument, ni vedno prosto število - lahko je katerikoli številčni izraz. Na primer:

```
VAL "2 * 3" = 6     ali celo
VAL ("2"+"* 3") = 6
```

Tu sta dva procesa. V prvem je argument funkcije VAL ocenjen kot string izraz