# bc Precision calculator

**START HERE**

**`bc -l`**
Calculator with math library

**`define f(x) { return x*x }`**
Own function

**`echo '2^10' | bc`**
Calculate from a pipe

**`quit`**
Leave bc

**`scale=4`**
Digits after the point

**`obase=2`**
Print in binary

**`ibase=16`**
Read hex input

**`last`**
Previous result

**Input** read in ibase → `compute` → **Arbitrary precision** scale decides digits → `print` → **Output** shown in obase

## CONCEPTS

Division truncates to scale digits and scale starts at 0, so 10/3 prints 3. Set scale=N first, or use -l (scale 20).

Set obase before ibase when both change: after ibase=16, the number 10 means sixteen. Use A for ten while ibase is 16.

This sheet covers Gavin Howard's bc (BSD, macOS, many Linux distros). GNU bc lacks some extensions marked here.

## RUNNING BC

**`bc`**
Start interactive calculator

**`bc -l`**
Math library and scale 20

**`echo '<expr>' | bc -l`**
Evaluate an expression from a pipe

**`bc -e '<expr>'`**
Evaluate an expression argument

**`bc <file>`**
Run a script file, then read stdin

**`bc -f <file>`**
Run a file, exit afterwards

## RUNNING BC (cont.)

**`bc -q`**
Skip banner (GNU compatibility)

**`bc -s`**
Accept strict POSIX bc only

**`bc -w`**
Warn about non-POSIX extensions

**`bc -S <n> -I <base> -O <base>`**
Preset scale, ibase, obase

**`bc -z`**
Print leading zero (0.5, not .5)

**`bc -L`**
Do not wrap long output lines

**`BC_LINE_LENGTH=0 bc`**
Same as -L via environment
*Default wrap is 70 digits with a backslash, which breaks piping long results; 0 turns it off.*

**`BC_ENV_ARGS="-l"`**
Default options for every run

## PRECISION AND BASES

**`scale=<n>`**
Digits kept after the decimal point

**`scale`**
Show current scale

**`scale(<x>)`**
Digits after the point of a value

## PRECISION AND BASES (cont.)

**`obase=<n>`**
Output base (2 to max)

**`ibase=<n>`**
Input base (2 to 36)

**`obase=16; <n>`**
Convert decimal to hex

**`ibase=16; <HEX>`**
Read hex as input (uppercase digits)

**`last` or `.`**
Last printed result

## OPERATORS

**`+ - * / %`**
Basic arithmetic and remainder
*Modulus honours scale; with scale=0 it is the integer remainder.*

**`^`**
Power with integer exponent

**`sqrt(<x>)`**
Square root

**`++x x-- x+=1`**
Increment and compound assignment

**`== != < <= > >=`**
Comparisons giving 1 or 0

**`&& || !`**
Boolean and, or, not

**`abs(<x>)`**
Absolute value

## OPERATORS (cont.)

**`length(<x>)`**
Number of significant digits

## MATH LIBRARY (-L)

**`s(<x>)`**
Sine in radians

**`c(<x>)`**
Cosine in radians

**`a(<x>)`**
Arctangent in radians

**`l(<x>)`**
Natural logarithm

**`e(<x>)`**
e to the power x

**`4*a(1)`**
Pi to current scale

**`pi(<p>)`**
Pi at precision p

**`t(<x>)`**
Tangent

**`l(<x>)/l(10)`**
Log base 10 (or l10(x))

**`log(<x>, <b>)`**
Logarithm in any base

**`p(<x>, <y>)`**
Power with non-integer exponent

**`r(<x>, <p>)`**
Round to p places

**`root(<x>, <n>)` `cbrt(<x>)`**
Nth root; cube root

## MATH LIBRARY (-L) (cont.)

**`f(<n>)`**
Factorial

**`gcd(<a>, <b>)` `lcm(<a>, <b>)`**
Greatest common divisor; least common multiple

**`max(<a>, <b>)` `min(<a>, <b>)`**
Larger or smaller of two

**`hex(<x>)` `binary(<x>)`**
Print integer as hex or binary

## PROGRAMMING

**`define <f>(<a>, <b>) { ... }`**
Define a function

**`define void <f>() { ... }`**
Function that returns no value

**`auto <i>, <j>`**
Local variables (first line in function)
*Without auto, variables are global and shared with the caller.*

**`return (<x>)`**
Return a value

**`if (<c>) { } else { }`**
Conditional

**`while (<c>) { ... }`**
Loop while true

**`for (i=0; i<10; i++) { ... }`**
Counting loop

## PROGRAMMING (cont.)

**`break` `continue`**
Leave or skip a loop iteration

**`a[<i>] = <x>`**
Array element (zero-based)

**`print "<text>", <x>, "\n"`**
Print without implicit newline

**`"<text>"`**
String literal prints as is

**`/* ... */` `# ...`**
Comments

**`quit`**
Exit bc

## RECIPES

**Precise division in a script**

```
echo 'scale=6; 22/7' | bc
echo '22/7' | bc -l | cut -c1-8
echo 'scale=2; 100*1.0825' | bc
```

**Convert between bases**

```
echo 'obase=16; 255' | bc
# FF
echo 'ibase=16; FF' | bc
# 255
echo 'obase=2; ibase=16; A5' |
bc     # 10100101
```

**Sum a column of numbers**

```
paste -sd+ numbers.txt | bc
awk '{s+=$1} END {print s}'
numbers.txt
```

**Define a function and use it**

```
bc -l <<'EOF'
define cube(x) { return x^3 }
scale=3; cube(1.5) + e(1)
EOF
```