# HyperCalc

Originally created by Robert P. Munafo.
Ported to JavaScript by Kenny TM~

---

Go ahead – just *try* to make me overflow!

# Contents

# Chapter 1

# Introducing HyperCalc

Which is bigger: $27^{86!}$ or $\left(27^{86}\right)!$? Most calculators can't even give the value of $27^{86}$ or of 86!.

With HyperCalc you can see that $27^{86}$ is $1.25107\ldots \times 10^{123}$, and 86! is $2.422709\ldots \times 10^{130}$. Some calculators can handle that – the current record-holder is AlCalc for the Pilot, which goes as high as $10^{32767}$ and can handle 9274! (9274 factorial).

But no other calculator can tell you that

$$\left(27^{86}\right)! = 10^{1.534607\ldots\times10^{125}}$$

or that

$$27^{86!} = 10^{3.467778\ldots\times10^{130}}$$

(in other words, the first has over $10^{125}$ digits and the second, with over $10^{130}$ digits is "just a little bit" larger.)

## 1.1 So what is HyperCalc?

HyperCalcis an open-source interpreted calculator program designed to calculate extremely large numbers (such as your phone number raised to the power of the factorial of the Federal budget deficit) without overflowing.

It does this by using a modified form of the level-index number system with a radix of $10^{300}$.

| Year | Model | Overflow |
|---|---|---|
| 1973 | TI SR-50 | $10^{100}$ |
| 1980 | Sharp EL-5100 | $10^{100}$ |
| 1989 | Casio *fx*-7500G | $10^{100}$ |
| ? | Casio *fx*-115D | $10^{100}$ |
| 1995 | Casio CFX-9800G | $10^{100}$ |
| 1997 | Pilot AlCalc | $10^{32768}$ |
| 1998 | Casio *fx*-260 | $10^{100}$ |
| 1998 | Sharp EL-531L | $10^{100}$ |
| 1998 | TI-85 | $10^{1000}$ |
| 1998 | TI-92 | $10^{1000}$ |
| 1999 | TI-89 | $10^{1000}$ |
| 2003 | Mathematica 5 for Windows | $1.92022 \times 10^{646456887}$ |
| 1998 | HyperCalc PalmPilot (for Palm) | 32768^^(300) |
| 1999 | HyperCalc Perl (for UNIX) | $10^{10}$^^(300) |
| 2004 | HyperCalc JavaScript (for WWW) | $(1.79769 \times 10^{308})$^^(300) |

Table 1.1: Performance statistics for other calculators

## 1.2 Representing Numbers in HyperCalc

The overflow value for HyperCalcis so large it can't be represented in the standard way. If we use HyperCalc's internal "PT" (Power Tower) format it's easy.

HyperCalc handles numbers with absolute value greater than the range supported by the floating point library by storing the numbers in many different formats. When the numbers are within normal floating-point range (less than $10^{300}$) they are stored in the normal floating-point format. Between $10^{300}$ and $10^{10^{300}}$ they are stored as (common) logarithms, and Logarithmic Number System (LNS) algorithms are used. When the logarithm gets too big to store as a floating point number, the logarithm is taken again, and so on. An integer field is used to keep track of how many times the logarithm has been taken. Table 1.2 shows some examples:

Each time we transition from the top of one PT range to the bottom of the next, about 2.5 digits of precision are lost as the information formerly stored in the exponent has to be absorbed by the mantissa. Then, as we proceed up the range digits are gradually gained back until we reach the top of the range and we once again have a 2.5 digit exponent. So, for example at the top of the PT = 0 range the values are things like $1.23456789012345 \times 10^{299}$, and there are 53 binary digits of precision in the mantissa, or almost 16 decimal digits. Then we cross over into the PT-1 range and store the logarithm instead, which becomes a value like 301.456789012345 – we still have 15 or more digits to work

| PT-Notation | PT | Value | Representation |
|---|---|---|---|
| 0^^(1.0) | 0 | 1.0 | 1.0 |
| $0\hat{}\hat{}\left(3.45\times10^{10}\right)$ | 0 | $3.45\times10^{10}$ | $3.45\times10^{10}$ |
| $0\hat{}\hat{}\left(1.0\times10^{299}\right)$ | 0 | $1.0\times10^{299}$ | $1.0\times10^{299}$ |
| $0\hat{}\hat{}\left(9.9\times10^{299}\right)$ | 0 | $9.9\times10^{299}$ | $9.9\times10^{299}$ |
| 1^^(300) | 1 | 300 | $10^{300}$ |
| 1^^(300.301) | 1 | 300.301 | $2\times10^{300}$ |
| 1^^(301) | 1 | 301 | $10^{301}$ |
| 1^^(834.173) | 1 | 834.173 | $1.489\times10^{834}$ |
| 2^^(79) | 2 | 79 | $10^{10^{79}}$ |
| 3^^(34) | 3 | 34 | $10^{10^{10^{34}}}$ |
| $254\hat{}\hat{}\left(10^{10}\right)$ | 254 | $10^{10}$ | $\underbrace{10^{10^{10^{\cdot^{\cdot^{\cdot^{10}}}}}}}_{256}$ |
| $32767\hat{}\hat{}\left(10^{300}\right)$ | 32767 | $10^{300}$ | $\underbrace{10^{10^{10^{\cdot^{\cdot^{\cdot^{10^{300}}}}}}}}_{32768\text{ tens}}$ |

(To read about even larger numbers, go to `www.mrob.com` and click on "Large Numbers".)

Table 1.2: Examples of PT-Notation

with, but the first three correspond to the exponent of the number and there are only 12 or 13 digits left for expressing the mantissa. Of course as we keep going up we get to values like 123456.789012345 (which represents $6.15\times10^{123456}$) we lose even more mantissa digits to exponent, but eventually we'll get to values like $123456789012345000000 = 1.2345\ldots\times10^{20}$, which represents $10^{1.2345\ldots\times10^{20}}$ and as we go on up to even bigger numbers we see that since the exponent needs to be printed it once again holds information equivalent to 2.5 digits.

This entire issue of variable number of digits and the associated problems it causes with non-intuitive round-off performance would be avoided if one used a "natural" PT storage format, where $e$ (base of natural logarithm) is the base and the representation is such that the floating point value is always in the inteval $[1,e]$. So, for example, the number 143 would be represented as 2^^(1.601979...) because $e^{e^{1.601979\ldots}}$ is 143. Such a format would be unwieldy for normal calculations, however, because you'd have to keep doing $e^x$ and $\ln x$ all over the place when doing simple calculations like 25 + 2.

# Chapter 2

# History of HyperCalc

Notice that HyperCalc PalmPilot and HyperCalc Perl are created by Mr. Munafo, while HyperCalc JavaScript is written by Kenny TM~.

## 2.1 Revision history of HyperCalc PalmPilot

**Oct 1?th, 1998** Start project from "SampleCalc" example.

**Oct 18th, 1998** Fairly complete scientific calculator, except trigonometric functions.

**Oct 21st, 1998** Start implementing PT functions, get `pt_exp` and `pt_mul` working.

**Oct 22nd, 1998** Implement addition, subtract, power, common logarithm (base 10), common antilogarithm, and gamma function.

**Oct 24th, 1998** Pretty much complete on the PT functions; they even handle infinity. Also, add a "tiny" font to print exponents when using the `stdFont`.

**Oct 25th, 1998** Refine the formatting code for PT-1's and higher so it computes exactly how many digits of mantissa can be shown. Add some more buttons, but most not implemented yet. Implement rounding (incredibly complex!). Add inverse trigonometric function, hyperbolic function, variable definition, and reciprocal keys. (but only reciprocal is implemented).

**Oct 26th, 1998** Add the same formatting refinements to PT-0's, so it can print contents of memories (which have fewer pixels available). Implement variable defintion.

**Oct 28th, 1998** Add hyperbolic functions and inverse trigrinometric functions (but not inverse hyperbolic functions).

**Oct 30th, 1998** Add inverse hyperbolic functions.

**Oct 31st, 1998** Put `f_` and `pt_` routines in their own files. Implement floating-point square root based on the grammar school algorithm (greatly increases speed of inverse trigonometric functions).

## 2.2 Revision history of HyperCalc Perl

**Jun 10th, 1999** Start writing a simple Perl calculator program using a new concept: expression evaluation via regular expressions (I got the idea while writing the `top100` movie statistics program). Right now it just does addition and multiplication.

**Jul 1st, 1999** Break the addition operator into a separate subroutine `add1` (eventually all operators will be done this way).

**Jul 20th, 1999** Add all the code from the HyperCalc PalmPilot, to eventually merge and translate into Perl.

**Jul 21st, 1999** Parsing routine is fairly complete and now includes nested loop to handle parentheses. Subroutines for all four operators $(+, -, \times, \div)$. "*e*" and "%" in an expression represent 2.71828 . . . and previous result, respectively.

**Jul 25th, 1999** Add `split` and start writing first operator that handles PT types: `p_add`, `pt_add`, `pt_addpos`.

**Jul 27th, 1999**

**21:25** Do lots of porting work: put all routines in "proper" (Pascal) order; lots of global replaces to change things like `x.pt` to `$x_pt`; replace Taylor and Newton algorithms with builtin functions where available; minimum work to get `pt_addpos` working. It now properly adds $10^{300} + 10^{300}$ (and gets 1^^(300.3010299 . . .)).

**21:54** `pt_add` fully works; `pt_div` works.

**22:35** `pt_sub` and `pt_mul` work now. Output formatting handles some of the special cases to print values like 1^^(2345.6789) as $4.77 \times 10^{2345}$ rather than as "1 PT 2345.6789".

**27:22** `pt_ln` works; parser handles `ln()` and `log()`.

**Jul 28th, 1999**

**13:33** It now handles `exp()` and `pow()`, so I can compute really big values without lots of repetitious keystrokes.

**25:??** `eval2()` now stores all operator results into an array, and stores the array index into the expression string. This is to avoid numbers getting converted from strings into floating point and back again, and that dramatically reduces roundoff error.

**Jul 29th, 1999** Start editing all the `f_` routines so the primitive floating-point type can be changed easily later. This involves implementing a minimal set of "primitives" like `f_int`, `f_le`, `f_neg`, `f_mul`, etc. and making all the other `f_` routines do all their operations by calling these primitives. Also, inline constants like "10" are replaced with globals.

**Aug 1st, 1999** `pt_root` and `pt_log_n` work. All of the `f_` routines are "primitivized", but `pt_` routines still need some work. Also added "debug" command. Put most of `f_` primitives inside `$f64_prim` so they can be defined and redefined via `exec`. Create `$g_pt_inf` to distinguish uses of infinity in PT field from its uses in VAL field. A few other changes to support switching VAL primitive precision. Make it auto-promote inlines like "`23E+456`".

**Aug 2nd, 1999** Pretty much finished making the `pt_` routines call `f_`.

**Aug ?th, 1999** Use `open2()` to launch bc. Write `bce`.

**Aug 5th, 1999** Write `fbc_fix2sci`, `fbc_split`, `f_cmp`, comparison primitives, `f_neg`, `me_magcompare`, `m_truncround`, `me_addpos`, `f_add` and `f_sub`. `fbc_encode` renamed to `fbc_sci2fix`. Redirect `stderr` when launching bc.

**Aug ?th, 1999** Write `me_subpos`.

**Aug 11th, 1999** Add `HC_LOG` debug log, lots of calls to `dbg1`. Fix lots of bugs. Write bc version of `f_mul` and `f_div`.

**Oct 15th, 1999** Fix bug that caused small PT-1's to be printed as e.g. $10^{301.30103}$. Make `dbg1flag` a bitmask to allow debugging functions, expression parsing, or both routines explicitly.

**Oct 17th, 1999** Add variables (currently limited to all-alphabetic starting with "v").

**Oct 18th, 1999** Change single-letter function abbreviations and special letters like "e", "p" etc. to uppercase, to clear the lowercase namespace for use by user variables.

**Oct 19th, 1999** Fix some bugs relating to infinity handling and conversion in `fbc` routines. Four basic functions almost work (subtraction still seems to have problems).

**Nov 17th, 1999** Variables no longer need to start with "v". Add square root function.

**Nov 24th, 1999** Combine parsing of $e, \pi, \phi$ with the variable and function parsing; add error-check for undefined variables.

**Jan 20th, 2000** Write fbc versions of `f_ln` and `f_exp`; fix bugs in `fix2sci` and `sci2fix`; it now correctly computes $2^{100}$ in scale 30. Fix bugs in switching back and forth between `f64` and `fbc`.

**Feb 6th, 2000** Fix bug that prevented `sqrt(1+2)` from working.

**Mar 4th, 2000** Square root now goes through `f_sqrt`. Fix bugs that made `bc hi_init` not compute `g_pi` properly.

**Jul 28th, 2000** Remove dependency on "`rpmlib.pl`".

**Jan 2nd, 2001** Add `ERASE_BS` test.

**Jan 3rd, 2001** Clean up internals of `eval_2`. Fix "right-to-left precedence bug": $4-3-2$ used to give 3, and 4/3/2 used to give 2.66667 . . .. I am deliberately leaving exponents that way: 4^3^2 still gives 262144.

**Jan 7th, 2001** Fix bugs: 2+2/(1+1) gave 2; 7^−1 didn't parse; scale 50, $27^{27}$ printed in scientific notation. Write `pt_roundup`. Fix `prnt1` handling of high PT-1's. `fbc`-based PT calculation is actually usable now!

**Jan 8th, 2001** Add history array and `define_hist`. Conversion across scale changes works, at least in the cases I checked. Fix bug in `eval_2`: Square root and other functions had become broken as a result of yesterday's fixes. Clean up `fbc` version of `f_gamma` a little, but it still suffers from a fundamental limit of the Stirling formula method, which basically requires that the number being factorialed must be at least as big as the 15th root of $10^{\texttt{curscale}}$. Combined with the current limit of $10^{300}$ for the `fbc` float data type, that means we can't get more than 33 digits of accuracy out of the `f_gamma` function. Increasing the exponent limit would fix it, but that poses another problem with the scaling loop – for 50 digits of accuracy, the scaling loop has to loop 2154 times (because $2154 = 10^{\frac{50}{15}}$). Finish implementing `format` command.

**Jan 9th, 2001** Fix bug that made history list usable only for first 9 items.

**Jan 15th, 2001** Write `init_pi_2`, which calculates $\pi$ much more quickly. Decrease `gammalim`.

**Jan 16th, 2001** Add input history.

**Jan 17th, 2001** Change letters I/H for input and output history to C/R (commands and results).

**Feb 10th, 2001** Fix "`c2`" in case where `c2` is a variable assignment, and add ";" symbol to separate commands.

**Feb 16th, 2001** Add ability to take `1E9` as input (used to require `1.E9`).

**May 21st, 2001** Make `x` a synonym for `*`. This works pretty well, in fact you can even define a variable `x`, and the expressions `2 x 4`, `2 x x` and `x x x` all do the right thing! But, that's not recommended. Also, change default output format to format 1, and make it print multiplication as `x` because it looks better. Also, mapped `[]` in input to `()`. This almost solves the problem of having output and input formats match – the one missing piece is allowing the user to type "PT", such as `3 PT 1.2 x 10^45`.

**May 30th, 2001** Almost fix the ambiguity of "!!": You can now type `4!!` and it will give you (4!)!, rather than "4" followed by the previous typed line.

**Jun 1st, 2001** When ";" is present in input, print each of the commands with its `C# =` label as they're being added to `input_history`.

**Jun 10th, 2001** Detect presence of UNIX and doesn't try to run `bc` if not on UNIX.

**Jun 13th, 2001** Fix some of the bugs in handling of "−". Add `pt_negate`.

**Oct 26th, 2001** Fix some bugs in command history expansion.

**Nov 4th, 2001** Add autodetect of `^H` and call `stty erase` if they type it (UNIX only)

**Jan 29th, 2002** Move automatic `stty erase` fix to subroutine `fixerase`.

**Mar 1st, 2002** Read first expression from command line.

**Mar 5th, 2002** Fix some bugs in rounding and `prnt2` – but it still has the problem that `scale=15` prints the same number of digits as the default `scale=14`.

**Mar 6th, 2002** Now can put multiple commands including `scale=` and `quit` on command line.

**Jul 11th, 2002** Convert tabs to spaces in input.

## 2.3 Revision history of HyperCalc JavaScript

**Oct 18th, 2004** Started to convert HyperCalc Perl into Visual Basic.

**Oct 21st, 2004** Convert into JavaScript instead, since the language of VB does not really match that of Perl but JS. Moreover, JS has built-in support of Regular Expressions while VB not.

**Nov 4th, 2004** HyperCalc JavaScript basically finished. Started documentation.

**Nov 5th, 2004** Now the program displays $9^{9^9}$ as $4.2812\ldots \times 10^{369693099}$ instead of $10^{369693099.631\ldots}$. (i.e., will use scientic notation as much as possible.)

**Nov 7th, 2004** Improved output history out-of-range detection. Handles `1E+12345` correctly. Can use Mathematica-Style `2*^6` for normal `2E6`. 5^^(5) now displays $10^{10^{10^{10^{100000}}}}$ instead of $10^{10^{10^{10^{10^5}}}}$ (i.e., try to collapse PT level as much as possible. The current routine is not perfect yet, however). Implemented input history. Fixed a bug that causes functions not working.

**Jan 11th, 2005** Now the program outputs $10^{-8}$ instead of **1e-8**. Improved the input-review system that it won't wait too long when calling several `$` repeatedly.

**Jan 16th, 2005** Fixed a bug that calculates $\frac{e^{10^{86}}}{2}$ wrong (resulting a PT-0). Added the ? command.

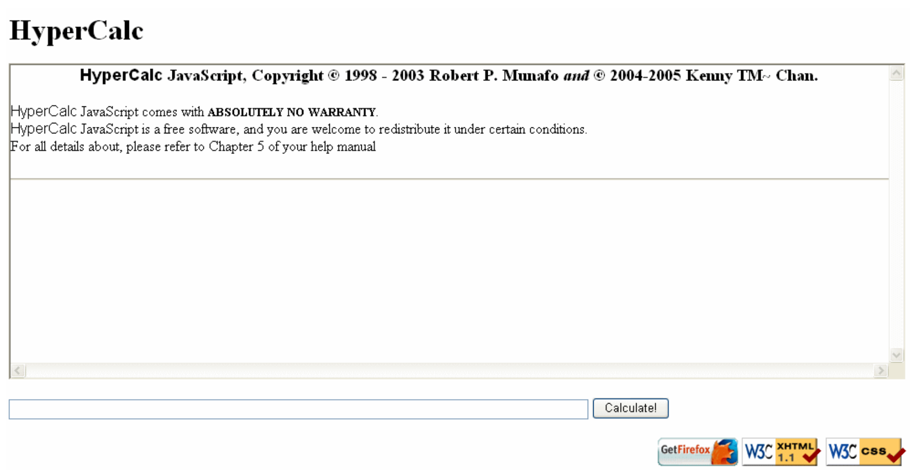


Figure 2.1: A typical screen of HyperCalc JavaScript

# Chapter 3

# Using HyperCalc

*Big notice to HyperCalc Perl users: I've basically changed the interface of HyperCalc JavaScript from the original versions because I haven't copied those functions after* `eval_1()`*. So if you use the input like HyperCalc Perl you'll probably get a wrong answer or error.*

## 3.1 Evaluating Simple Expressions

It is easy to use HyperCalc. After HyperCalc is loaded, you should be able to see a large blank in the middle, a text field under the blank and a button called "Calculate!" on the right of the field. The large blank is the **output screen** of HyperCalc that all results will be displayed there. The text field is for entering expression, and the button is to evaluate the expression you entered.

You can just enter your expressions like the ones displayed in textbook. For example, to calculate 1 + 2, you enter

- `1 + 2`

in the textfield and press the "Calculate!" or hit ENTER. The followings will be shown in the output screen:

---

In[1] := $1 + 2$
Out[1] = **3**

---

The following lists all available operations in HyperCalc:

| Operator | Purpose | Example | Result |
|---|---|---|---|
| `+` | Addition | `1 + 2` | 3 |
| `-` | Subtraction | `6 - 7` | $-1$ |
| `*` | Multiplication | `4 * 2` | 8 |
| `/` | Division | `3 / 5` | 0.6 |
| `^` | Raising power | `2 ^ 10` | 1024 |
| `e` | Base of natural logarithm ($e = 2.71828\ldots$) | `e ^ 5` | $148.413\ldots$ |
| `pi` | Pi ($\pi = 3.14159\ldots$) | `pi / 2` | $1.57079\ldots$ |
| `phi` | Golden ratio ($\phi = \frac{\sqrt{5}+1}{2}$) | `1 / phi` | $0.618033\ldots$ |
| `eulerGamma` | Euler's gamma constant ($\gamma = 0.577215\ldots$) | `-eulerGamma` | $-0.577215\ldots$ |
| `!` | Factorial | `8!` | 40320 |
| `inf` | Infinity | `1 / inf` | 0 |
| `(...)` | Parenthesis (Grouping) | `5*(1-6)` | $-25$ |
| `exp` | Natural anti-logarithm ($e^x$) | `exp(5)` | $148.413\ldots$ |
| `ln` | Natural logarithm | `ln(10)` | $2.32585\ldots$ |
| `log` | Common logarithm | `log(e)` | $0.434294\ldots$ |
| `logb` | Logarithm of specific base | `logb(2,64)` | 6 |
| `sqrt` | Square root | `sqrt(3)` | $1.73205\ldots$ |
| `root` | Taking root | `root(3, 8)` | 2 |
| `sin`, `cos`, `tan` | Trigonometric functions | `sin(pi/3)` | $0.866025\ldots$ |
| `asin`, `acos`, `atan` | Invserse trigonometric functions | `atan(inf)` | $1.57079\ldots$ |
| `gamma` | Gamma function | `gamma(0.5)^2` | $3.14159\ldots$ |
| `deg` | Degree sign ($^\circ = \frac{\pi}{180}$) | `sin(60deg)` | $0.866025\ldots$ |

In HyperCalc, multiplication signs can be omitted. For instance, the expressions `3 * tan(30 * deg)`, `3 tan(30 deg)` and `3tan(30deg)` all result in $\sqrt{3}$. You can even type `7 4` for $7 \times 4$. However, the parenthasis around arguments of functions cannot be omitted, i.e., `log(5)` must be typed as is, and `log 5` will be interpreted as "log ×5" and result in **NaN**.

HyperCalc follows the precedence like normal algebraic calculation. To explain explicitly, the *functions* and *parenthesis*, are handled first, then *factorial*, then *negation* (e.g., `-123`), then *power raising*, then *multiplication* and *division* and finally *addition* and *subtraction*. When operators of the same precedence go together, they are handled from left to right **except** power raising, which is handled from right to left.

HyperCalc is case-insensitive, that means `gamma`, `Gamma`, `GAMMA` and `gAmMA` are all the same. Also, many functions in HyperCalc possesses **alias** that do the same job as the original. The following lists all aliases available:

| Function | Alias |
|---|---|
| `(` | `[` |
| `)` | `]` |
| `inf` | `infin`, `infty`, `infinity` |
| `phi` | `goldenRatio` |
| `ln` | `loge` |
| `log` | `log10` |
| `logb` | `logn` |
| `asin` | `asn`, `arcsin` |
| `acos` | `acs`, `arccos` |
| `atan` | `atn`, `arctan` |
| `sqrt` | `sqr` |
| `root` | `rt` |

## 3.2 Big Numbers in HyperCalc

### 3.2.1 Entering Big Numbers

Since HyperCalc is designed for calculation with really big numbers. To enter a big number, the most common method is using scientific form:

- *mantissa*`E`*exponent*

Here "mantissa" and "exponent" are two real number. This represents $m \times 10^e$. For example, `5E+16` means $5 \times 10^{16}$. The value of "exponent" is not limitted as for many other calculators. You can set it as high as you want — there is no problem in handling `1E+1234567890`.

However, the scientific form cannot be used to enter *really* big numbers, say, $10^{10^{1234567890}}$ would require you to enter $\texttt{1E+1}\underbrace{\texttt{0000...0000}}_{\text{1234567890 zeros}}$. This is clearly impossible. However, we can use the PT notation to indicate these kinds of numbers. (See section 1.2 for details of PT notation.) To enter a PT number, use

- *pt*`P`*value*

which represents

$$\underbrace{10^{10^{10^{\cdot^{\cdot^{\cdot^{10^{v}}}}}}}}_{p\text{ tens}}$$

So for $10^{10^{1234567890}}$ we can just input `2P1234567890`. Note that the "value" must be positive.

The alias of `E` is `*^` and `P` is `PT` and `^^`.

### 3.2.2 Displaying Big Numbers

HyperCalc will display numbers as natural as possible. But sometimes the number will be too "big" to display in radix form, and it will be "collapsed" into a single PT notation. To be clear, try evaluate `6P1` and `7P1`. The former will result in $10^{10^{10^{10^{10000000000}}}}$, but the latter will become 6^^(10). This is because the latter is too "big" and using PT notation would be better. By default HyperCalc will only display values in radix form upto PT-5.

## 3.3 I/O History

The I/O history is the list of input/output results on the output screen. You can use I/O history retrieval commands to get those values.

### 3.3.1 Output History

The last output can be obtained by entering `%`. For example:

- `pi^pi`
- `%^2 - 2% + sin(%)`

will evaluate **36.4621596072079** and then **1255.6199743011982**. If you want to refer to one specific output at line $n$, use

- %$n$

### 3.3.2 Input History

The last input can be re-evaluated by entering `$`. For example:

- `3`
- `3^%`
- `$`

will evaluate **3**, **27** and **7625597484987**. If you want to refer to one specific input at line $n$, use

- `$`*n*

## 3.4 Variables and Functions

The internal variables and functions are never enough for pratical use. Because of this, you can define your *own* variables and functions in HyperCalc.

### 3.4.1 Custom Variables

To define a custom variable, enter

- *name = def*

Here, "name" is the name of the variable and "def" is its definition. To use the variable, just type its name. For example,

- `c = 299792458`
- `m = 9.10938188E-31`
- `massEnergyOfElectron = m c^2`

will define three variables: $c$, $m$ and `massEnergyOfElectron` and are assigned to be 299792458, $9.10938 \times 10^{-31}$ and $mc^2 = 8.18710 \times 10^{-14}$ respectively.

Notice that the internal variables ($e$, $\pi$, $\phi$, $\gamma$ and $\infty$) will *never* be overridden. If you call `pi = 22/7` then use `pi` in later evaluations you will still get 3.14159 . . . but not 3.142857 . . ..

### 3.4.2 Custom Functions

To define a custom variable, enter

- *name* `:=` *def*

Here, "name" is the name of the function and "def" is its definition. You can use any numbers of arguments, and use #*n* to substitute them (the *n* corresponds to the *n*th argument). `#1` can be entered as just `#`. To use the function, type its name then followed by the list of arguments enclosed inside the parenthesis. For example,

- `cosineLawS := sqrt(#1^2 + #2^2 - 2#1#2cos(#3))`
- `cosineLawA := acos((#1^2 + #2^2 - #3^2)/(2#1#2))`
- `cosineLawA(5, 6, 7)/deg`

will define two functions: `cosineLawS` and `cosineLawA` that both take three arguments. Their definitions are:

$$\texttt{cosineLawS}(x_1, x_2, x_3) = \sqrt{x_1^2 + x_2^2 - 2x_1x_2\cos x_3}$$

and

$$\texttt{cosineLawA}(x_1, x_2, x_3) = \frac{x_1^2 + x_2^2 - x_3^2}{2x_1x_2}$$

The last statement evaluates the `cosineLawA` function and set the arguments $(x_1, x_2, x_3)$ to be $(5, 6, 7)$. The result of this function would be $1.36943\ldots$ and the final result would be **78.46304096718451**.

A function can take no arguments as well. For example,

- `f := %^%`

To call these kinds of functions, you do not need to place a pair of parenthesis after them, i.e.,

- `12`
- `5 + f`

works and results **8916100448261**.

As with variables, the internal functions cannot be overridden either.

### 3.4.3 Variables vs. Functions

At a first glance, a function with no arguments seems to have the same meaning as variable. This is totally wrong. To major difference of variables and functions is that variables are evaluated once they are assigned while functions are evaluated only when they are called. Compare the followings:

- `5`
- `myVar = 4 + %`
- `myFunc := 4 + %`
- `18`

If you call `myVar` after "18", you get 9 because when it is defined to be *the result of* `4 + %` in the second line, which is 9. But if you call `myFunc` you will get 22 because when it is defined to be*the pattern* `4 + %`.

### 3.4.4 Reviewing Custom Variables and Functions

To know what custom variables have been defined, enter

- `!=`

Similarly, to know definitions of all custom functions, enter

- `!:=`

### 3.4.5 Removing Custom Variables and Functions

To remove a variable or function, enter

- *name* `=.`

To remove all variables, enter

- `!!=`

or

- `!=.`

To remove all functions, enter

- `!!:=`

or

- `!:=.`

## 3.5 Miscellaneous

To clear the output screen, enter

- `!!`

To clear the I/O history, enter

- `!!%`

or

- `!!$`

To view all commands preset in HyperCalc, enter

- `?`

# Chapter 4

# Troubleshooting

## 4.1 Non-Intuitive Results when Working with Huge Numbers

If you spend a while exploring the ranges of huge numbers HyperCalc can handle, you will probably start noticing some paradoxical results and might even start to think the calculator is giving wrong answers.

For example, try calculating 27 to the power of *googolplex* (a googolplex is 10 to the power of *googol* and a googol is $10^{100}$). Key in:

- `27^10^10^100`

and it prints $10^{10^{10^{100}}}$. So the calculator thinks that:

$$27^{10^{10^{100}}} = 10^{10^{10^{100}}}$$

This is clearly wrong — and it doesn't even seem to be a good approximation. What's going on?

Let's try calculating the correct answer ourselves. We need to express the answer as 10 to the power of 10 to the power of something, because that's the standard format the calculator is using, and we're going to see how much of an error it made. So, we want to compute $27^{10^{10^{100}}}$ as a tower of powers of 10. The first step is express the power of 27 as a power of 10 with a product in the exponent, using the formula $x^y = 10^{y \log x}$:

$$27^{10^{10^{100}}} = 10^{\log 27 \times 10^{10^{100}}}$$

log 27 is about 1.43, so we have

$$27^{10^{10^{100}}} = 10^{1.43\times 10^{10^{100}}}$$

Now we have a base of 10 but the exponent still needs work. The next step is to express the product as a sum in the next-higher exponent; this time the formula we use is $xy = 10^{\log x + \log y}$:

$$10^{1.43\cdot 10^{10^{100}}} = 10^{10^{\log 1.43 + \log 100}}$$

log 1.43 is about 0.155, and if we add this to $10^{100}$ we get

$$\begin{aligned} & 10^{10^{0.155+10^{100}}} \\ = \; & 10^{10^{1000...000.155}} \\ = \; & 10^{10^{1.000...000155\times 10^{100}}} \end{aligned}$$

where there are 94 more 0's in place of each of the "...". So our final answer is:

$$27^{10^{10^{100}}} = 10^{10^{1.000...000155\times 10^{100}}}$$

Now that we've expressed the value of $27^{\text{googolplex}}$ precisely enough to see the calculator's error — look how small the error is! The calculator would need to have at least 104 digits of precision to be able to handle the value "1.000...000155" accurately — but it only has 16 digits of accuracy. Those 16 digits are taken up by the 1 and the first 15 0's — so when the calculator gets to the step where we're adding 0.155 to $1.0 \times 10^{100}$, it just rounds off the answer to $1.0 \times 10^{100}$ — and produces the answer we saw when we performed the calculation:

$$10^{10^{10^{100}}}$$

Even if it did have the precision, it wouldn't have room to print the whole 104 digits on the screen, so the answer you *see* would look the same. And no matter how many digits of accuracy we try to give the calculator, there's always another even bigger number it wouldn't be able to handle. For example, the calculator would need slightly over a *million* digits of accuracy to distinguish

$$27^{10^{10^{1000000}}} \quad \text{from} \quad 10^{10^{10^{1000000}}}$$

and if we just add one more **10** to that tower of exponents, all hope of avoiding roundoff is lost.

## 4.2 FAQ

### 4.2.1 Why I can't use `x` as the multiplication sign?

If you were switched from HyperCalc Perl, you will notice that `x` can no longer be a substitution of multiplication sign, and you will get an "Undefined variable or function" error. The reason is that HyperCalc JavaScript no longer supports this because of the introduction of implicit multiplication sign (spaces). For instance, if `x` is used as the multiplication sign, then it would be ambigious for what `x x x` means: does it mean $x \cdot x$ or $x \cdot x \cdot x$? Of course, the implicit multiplication sign feature can be removed, but this is a bigger trade-off. Even without the implicit multiplication sign, this feature is still a dirty implementation (at least in my opinion) and should not be used.

### 4.2.2 Why I can't use `c` or `r` as input/output history recall?

They are mapped to the characters `$` and `%` respectively.

### 4.2.3 I entered `!!` for re-evaluating the last statement but the screen was blanked.

You should enter `$` instead. `!!` is for clearing the output screen.

### 4.2.4 Why `7 / 100 * 100` does not give 7?

This is because of how JavaScript handles a number. In JavaScript, a number is in IEEE 1394 Double format, and all key infomation about a number is in **binary** format. Precision is lost because of this. Hence the result will be erred by a little — about $8.88 \times 10^{-16}$ in this case. In order to improve the accuracy, we have started to consider using *arbitrary-precision* float numbers, but this is hard to implement. Hence you should expect waiting for a long period.

### 4.2.5 Can I store my custom variables/functions in a file?

Generally, you can't.
Technically, you can do it by changing the source code (hint: changes line 86 and 87 in the source).

### 4.2.6 Can I redistribute/modify HyperCalc?

Yes. You can redistribute/modify HyperCalc under the terms of the GNU General Public License (See chapter 5).

### 4.2.7 What if I still have questions?

Email it to `casio_fifty@yahoo.com.hk`.

# Chapter 5

# GNU General Public License

Version 2, June 1991

## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

# GNU General Public License

## Terms and Conditions For Copying, Distribution and Modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or

indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

## No Warranty

11. Because the program is licensed free of charge, there is no warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.

## End of Terms and Conditions

## 5.1 How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <*one line to give the program's name and a brief idea of what it does.*>
> Copyright (C) <*year*><*name of author*>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place – Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright (C) <*year*><*name of author*>
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
> This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w` and `show c`; they could even be mouse-clicks or menu items — whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program
> 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> <*signature of Ty Coon*>, 1 April 1989
> Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.