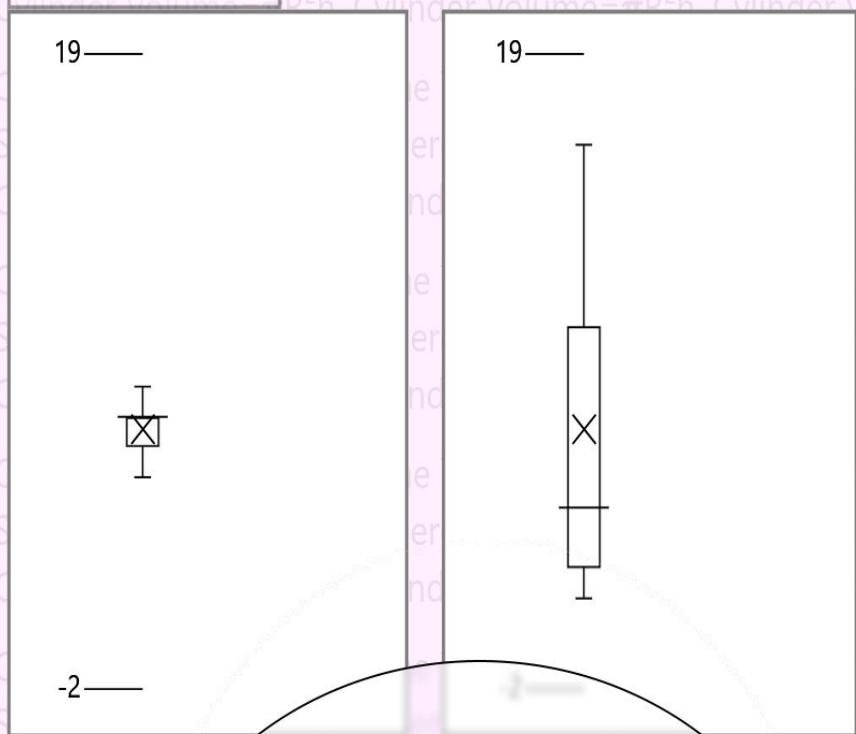


### Boxplots (compare) ✓



## Results

### Robust

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

### Classical

$\bar{x}$	(mean)	5.0
N	(count)	33
$\Sigma$	(sum)	165
s	(sample stddev)	1.1402
$\sigma_n$	(pop. stddev)	1.0198
RSD	(rel. stddev)	0.1728
	(sample stddev)	6.3087
	(pop. stddev)	5.6427
	(rel. stddev)	0.9559

Guide to using  
Best Calculator  
for windows and windows Phone  
2016 Edition

BC BASIC Reference manual and tutorial

including

The Best Calculator Reference Manual

© 2016, Peter D Smith

The Best Calculator app is available for Windows and Windows Phone  
from the Microsoft App Store at

<https://www.microsoft.com/en-us/store/apps/best-calculator/9wzdnrcrdfd6x>

Visit the Best Calculator web site at

<https://bestcalculator.wordpress.com/>

## TABLE OF CONTENTS

---

1	A Quick Tour of Best Calculator .....	15
2	Numbers and Common Calculations .....	17
2.1	Simple Arithmetic.....	17
2.2	Chain calculations.....	17
2.3	Algebraic Entry and Parentheses.....	17
2.4	Editing errors and clearing the display .....	18
3	Memory Keys .....	19
3.1	Memory operation .....	19
3.2	Example .....	19
3.3	Memory Add and Subtract .....	20
4	More Math .....	21
4.1	Square ( $x^2$ ) key.....	21
4.2	Square Root ( $\sqrt{x}$ ) key .....	21
4.3	Inverse (1/x) key .....	21
4.4	Change Sign ( $\pm$ ) key .....	21
5	Scientific Notation.....	22
5.1	Using the EE key .....	22
6	Percent Key % .....	23
6.1	Percent (%) key.....	23
6.2	Calculating sales tax .....	23
6.3	Calculating sales with a percent discount .....	23
7	Trigonometry keys on the Advanced calculator .....	24
7.1	Calculate in degrees or radians .....	24
7.2	Sin, Cos, Tan .....	24
7.3	Inverse Sin, Cos, Tan .....	25
8	Logarithms on the Advanced calculator .....	26
9	Powers, roots, $x!$ , mod on the Advanced calculator .....	27
9.1	Factorial ( $x!$ ) key .....	27

	Page   4
Guide to Using Best Calculator	
9.2 Mod Key .....	27
9.3 Cube ( $x^3$ ) and Cube Root ( $\sqrt[3]{x}$ ) keys.....	27
9.4 Arbitrary Power ( $x^y$ ) and Root ( $y\sqrt{x}$ ) keys.....	28
10 Rounding and abs on the Advanced calculator .....	29
11 Random Numbers on the Advanced calculator .....	30
12 Formatting.....	31
12.1 Standard formatting .....	31
12.2 Exponent formatting .....	32
12.3 Fixed formatting .....	32
12.4 Natural formatting.....	32
12.5 Percent formatting .....	33
13 Constants .....	34
13.1 $\pi$ and e.....	34
13.2 $g_n$ c and $N_a$ .....	34
13.3 $NaN$ , $\infty$ and $-\infty$ .....	34
14 Memory Page.....	35
14.1 Show the operation of Memory0 .....	35
14.2 Saving and Roaming .....	35
14.3 Name a memory cell .....	35
14.4 Get from display and Copy to display.....	36
14.5 Clear .....	36
14.6 Memory and BC Basic.....	36
15 Dates Page.....	37
15.1 Compare Dates .....	37
15.2 Convert to Gregorian.....	37
15.3 Add days .....	38
16 Hex, Decimal, Octal, Binary on the Programmer's calculator.....	39
16.1 Setting the mode .....	39
16.2 Converting between bases .....	39
17 Bit Operators on the Programmer's calculator.....	40

	Page   5
Guide to Using Best Calculator	
17.1    B# (count bits) .....	40
17.2    Inverse ( $\sim$ ) and 2's complement .....	40
17.3    And (&), Or ( ) , Xor (^).....	40
18      Bytes and Swabbing on the Programmer's calculator .....	41
18.1    Bytes .....	41
18.2  SWAB (Swap Bytes) key.....	41
19      Shift operators on the Programmer's calculator .....	42
20      Programmer's Math.....	43
21      Statistical calculator .....	44
21.1    Parts of the Statistical calculator screen .....	44
21.2    Entering Data .....	45
21.3    Classical Statistics (Results) .....	46
Robust Statistics (Results) .....	47
Regression (Results) .....	48
21.4    Compare with T-Tests (Results).....	49
21.5    Boxplots (Graph).....	50
21.6    XY Scatter-plots (Graph) .....	51
22      Conversions.....	52
22.1    Converting .....	52
22.2    Copy to and from the Calculator results display .....	52
22.3    Area conversions .....	53
22.4    Energy Conversions .....	53
22.5    Length Conversions .....	53
22.6    Temperature Conversions .....	54
22.7    Weight .....	54
22.8    Farm Volumes (US).....	55
23      Ascii Table .....	56
24      Unicode .....	57
24.1    Search Rules .....	57
24.2    Copying characters .....	57

Guide to Using Best Calculator	Page   6
25 Advanced Windows Features.....	58
25.1 Shortcut on the desktop .....	58
25.2 Set as the Calculator Key .....	59
26 A brief historical note.....	60
27 Equation Input: your first program .....	61
27.1 What is a program? .....	61
27.2 How do I run my program? .....	61
27.3 How can I print some text? .....	61
27.4 How can my program use the calculator value? .....	62
27.5 How can I write an equation? .....	62
27.6 How can I write to the calculator? .....	62
27.7 Why do some examples start with an = sign? .....	63
27.8 How can I make several different programs?.....	63
27.9 What are all the keys on the screen? .....	63
27.10 How can I learn more? .....	64
28 Sigma Function: advanced programming .....	65
29 What all can you do in the BC BASIC environment? .....	67
29.1 All the main edit dialogs .....	67
29.2 Library of Packages.....	68
29.3 Library Properties (Import BC BASIC package) .....	70
29.4 Bind a program to a key .....	71
29.5 About this package .....	72
29.6 List of programs.....	74
29.7 About this program .....	76
29.8 Edit Program.....	78
29.9 Output screen.....	80
30 BASIC Language Reference .....	82
30.1 Program structure .....	82
30.2 Flags.....	83
30.3 Numbers and Strings and Variables .....	83
30.3.1 Numbers.....	83

Guide to Using Best Calculator	Page   7
30.3.2    Strings.....	84
30.3.3    Variables, GLOBAL, and DIM .....	84
30.4    <expression> overview .....	86
30.4.1    Quick introduction to expressions .....	86
30.4.2    Expression Rules.....	86
30.4.3    negate, power, root prefix and postfix operators .....	87
30.4.4    Operators + - * / and more .....	88
30.4.5    The INPUT expression .....	92
30.5    Math Functions.....	92
30.5.1    Trigonometry functions SIN COS TAN ASN ACS ATN .....	92
30.5.2    Logarithm and Power functions LN EXP SQR .....	93
30.5.3    Rounding and sign functions SGN ABS INT .....	93
30.6    String functions (LEFT, MID, RIGHT, LEN, CHR\$, CODE, VAL) .....	94
30.6.1    LEFT (string, count)., MID (string, index, count) and RIGHT (string, count)	94
30.6.2    LEN string .....	95
30.6.3    CHR, CODE.....	95
30.6.4    VAL(string).....	96
30.7    <statement> overview.....	97
30.8    Packages and Programs.....	98
30.8.1    Inside a package file .....	99
30.9    Picking a package for your program .....	99
30.9.1    Creating common functions for several programs .....	99
30.9.2    Exporting packages .....	99
30.10    All of the Special Symbols .....	100
31    BASIC Statements Reference .....	101
31.1    [CALL] <function> (<expression>, ...)	101
31.2    CLS [<color>] and PAPER <color> .....	102

	Page   8
<b>Guide to Using Best Calculator</b>	
31.3 CONSOLE <expression> [, <expression>].....	103
31.4 DIM <name> ([size]) .....	104
31.5 DUMP .....	105
31.6 FOR <variable> = <start> TO <end> [STEP <step>] ... NEXT <variable> .....	106
31.7 FUNCTION <name> ( <args> ) ... RETURN [<value>] .....	109
31.8 GLOBAL <variable> .....	109
31.9 GOSUB <linenumber> and RETURN .....	110
31.10 GOTO <linenumber> .....	110
31.11 IF (<expression>) THEN <statement> [ELSE <statement>].....	111
31.12 IF (<expression>) ... [ELSE ...] ENDIF .....	111
31.13 IMPORT FUNCTIONS FROM “program” .....	112
31.14 INPUT <variable>.....	112
31.15 (LET) <variable> = <expression>.....	113
31.16 PAUSE <frames>.....	114
31.17 PRINT [AT row,col] <expression> [ (, or ;) [AT row,col] <expression>]* ....	114
31.18 RAND <seed> & the RND value .....	115
31.19 REM comment words to the end of the line.....	116
31.20 STOP and END .....	117
<b>32 Extensions Reference.....</b>	<b>118</b>
32.1 Calculator.Value and Calculator.Message Extension .....	118
32.2 Memory Extension .....	119
32.2.1 Memory[<expression>] and Memory.<name> .....	119
32.2.2 GetOrDefault and IsSet functions .....	120
32.2.3 Memory technical details.....	121
32.3 Math Extension .....	123
32.3.1 Trigonometry (Math.Sin (radians) and more).....	123
32.3.2 Rounding and sign (Floor(), Round() and more) .....	123
32.3.3 Logarithm and power functions (Math.Log, Math.Exp, and more)	
126	
32.3.4 Math.Factorial and Math.IsNotNull.....	127

	Page   9
<b>Guide to Using Best Calculator</b>	
32.4    Math.PI and Math.E values .....	127
32.5    Screen.ClearLine(<line>) and Screen.ClearLines(<from>, <to>) .....	128
32.6    Screen.RequestActive() and Screen.RequestRelease().....	128
32.7    Screen.H and Screen.W Extension .....	128
<b>33      Bluetooth Programming with Best Calculator, IOT edition .....</b>	<b>130</b>
33.1    Programming Bluetooth using BC BASIC .....	130
33.2    Initializing your device and available properties .....	131
33.2.1    Error handling and Bluetooth .....	133
33.3    The objects you use when programming your Bluetooth device.....	134
33.3.1    The Bluetooth object .....	134
33.3.2    The Bluetooth.Devices object (Array / ObjectValueList) .....	135
33.3.3    Individual Bluetooth devices from Bluetooth.Devices.....	135
33.3.4    Specializations.....	136
33.4    Selecting a device with PickDevicesNames and more.....	136
33.4.1    Bluetooth.PickDevicesName(<name pattern>) .....	136
33.4.2    Bluetooth.DevicesName(<name pattern>).....	137
33.4.3    Bluetooth.Devices().....	138
33.5    Reading data from raw Bluetooth services and characteristics .....	138
33.5.1    Direct device Read routines .....	139
33.6    Using callbacks to read data.....	141
33.7    Using the specializations for specific devices .....	144
<b>34      Bluetooth Specializations for specific devices .....</b>	<b>145</b>
34.1    beLight CC2540T Light development kit.....	146
34.2    DOTTI device.....	147
34.3    Hexiwear wearable platform.....	150
34.4    MagicLight and Flux light.....	152
34.5    NOTTI device .....	153
34.6    TI SensorTag 2541 (original version) .....	156
<b>35      Graphics and BC BASIC.....</b>	<b>159</b>

	Page   10
Guide to Using Best Calculator	
35.1    Fixed character screen commands.....	160
35.2    Console Commands.....	162
36    Using the library, step by step .....	162
36.1    Add a new package for your program .....	164
36.2    Add a new program to your package .....	165
36.3    Edit the program to do the conversion .....	165
36.4    Run the program to test it.....	167
36.5    Bind the program to a key.....	168
36.6    Next steps.....	169
37    Appendix: Sample programs .....	170
37.1    BT: An Overview of Bluetooth.....	170
37.1.1    List Bluetooth devices .....	170
37.1.2    Pick a Bluetooth device .....	171
37.1.3    Power .....	171
37.2    BT: beLight.....	172
37.2.1    Green.....	172
37.2.2    Red .....	173
37.3    BT: Dotti.....	173
37.3.1    An introduction .....	173
37.3.2    Change Mode.....	174
37.3.3    List BT Devices.....	175
37.3.4    Load screen from memory .....	175
37.3.5    Raw Bluetooth commands .....	177
37.3.6    SetAnimationSpeed.....	178
37.3.7    SetColumn and SetRow to random lines .....	178
37.3.8    SetName of a Dotti device .....	179
37.3.9    SetPixel to random green dots.....	180
37.3.10    SetPixel to write a single green dot .....	180

Guide to Using Best Calculator	Page   11
37.3.11 Sync Time .....	181
37.4 BT: Hexiwear.....	181
37.4.1 Accelerometer.....	182
37.4.2 Compass .....	183
37.4.3 List Information.....	184
37.4.4 Raw Access to Hexiwear .....	185
37.4.5 Read All .....	186
37.4.6 Set notification count.....	188
37.4.7 SetTime .....	189
37.5 BT: MagicLight .....	189
37.5.1 Green.....	189
37.5.2 Off .....	190
37.5.3 On.....	190
37.5.4 Red .....	191
37.6 BT: Notti .....	191
37.6.1 An introduction .....	191
37.6.2 Change Mode .....	192
37.6.3 NOTTI timer.....	192
37.6.4 Raw Bluetooth commands .....	194
37.6.5 Set Alarm .....	194
37.6.6 SetColor to change the NOTTI color to blue .....	195
37.6.7 SetColor to change the NOTTI color to green .....	195
37.6.8 SetColorCustom animates the color from red to blue and back again	196
37.6.9 SetName of a NOTTI device .....	196
37.6.10 Sync Time .....	197
37.7 BT: SensorTag 2541 .....	198
37.7.1 Accelerometer.....	198

Guide to Using Best Calculator	Page   12
37.7.2 Accelerometer Off.....	199
37.7.3 Accelerometer to Magic Light.....	199
37.7.4 Barometer .....	201
37.7.5 Button .....	202
37.7.6 Gyroscope .....	203
37.7.7 Humidity.....	204
37.7.8 IR .....	205
37.7.9 Magnetometer .....	206
37.7.10 Raw access to SensorTag .....	207
37.7.11 Raw callback with the SensorTag.....	208
37.8 EX: BC BASIC Quick Samples.....	210
37.8.1 Colorful Countdown.....	210
37.8.2 Grams of Fat to Calories.....	211
37.8.3 Miles per Gallon .....	212
37.8.4 Right Triangle calculator .....	213
37.8.5 Tip Calculator .....	214
37.8.6 Welcome to BC BASIC .....	215
37.9 EX: Financial.....	215
37.9.1 Common Tip Values .....	215
37.9.2 Compound Interest .....	217
37.9.3 Doubling Time .....	218
37.9.4 Future Value.....	219
37.9.5 Money Conversion .....	221
37.9.6 Present Value .....	221
37.9.7 Return on Investment .....	223
37.10 EX: Real Estate.....	223
37.10.1 Acres to square feet.....	223

Guide to Using Best Calculator	Page   13
37.10.2    Debt to Income calculations .....	223
37.10.3    Minimum and Maximum density.....	224
37.10.4    Rectangle in feet to acres.....	225
37.10.5    Square feet to acres .....	226
37.11    EX: Space and Astronomy .....	226
37.11.1    Arc Length .....	226
37.11.2    AU to Meters.....	227
37.11.3    Conversion Library .....	228
37.11.4    Distance to horizon .....	229
37.11.5    Lightyears to Parsecs.....	230
37.11.6    Meters to AU .....	230
37.11.7    Parsecs to Lightyears.....	230
37.11.8    Rocket Equation .....	231
37.12    EX: Statistics .....	233
37.12.1    Finite Population Correction .....	233
37.12.2    Margin of Error.....	233
37.12.3    Pfail.....	234
37.12.4    Sample Size .....	235
37.12.5    Sample Size Library .....	236
Index to Best Calculator Manual.....	239

## Guide to Using Best Calculator

# Best Calculator editions

Page | 14

There are two editions of Best Calculator: the classic Best Calculator which has been shipped, free and with no ads, for several years. It's recently been extended to be programmable in BASIC. This simple language lets anyone write small programs to help automate some of the specialized math.



The standard edition splash screen with yellow "Programmable" stripe

The new Best Calculator, IOT edition takes the familiar and powerful Best Calculator and adds the ability to connect to a range of IOT (Internet of Thing) devices. All the parts in common with the standard edition of Best Calculator are free; the rest require a small payment.

As of December, 2016, the IOT capabilities are the ability to communicate to Bluetooth LE devices. Both low level access and simplified, higher-level *specializations* are available for a number of Bluetooth devices. There is a free trial so you can make sure that the app has the capabilities you need.



The IOT edition with the blue Bluetooth stripe

This manual describes both editions. There will be a note for each feature that is only part of a paid edition.

# 1 A QUICK TOUR OF BEST CALCULATOR

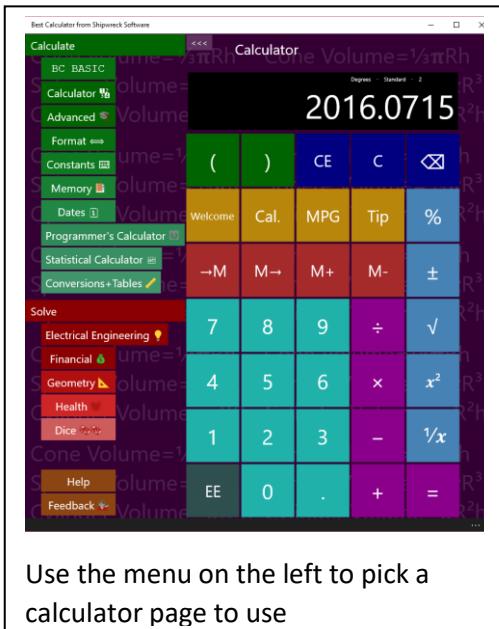
Best Calculator has *Pages* (in green) for common calculations, *Solvers* (in red) for specialized calculations, and Help and Feedback keys (in brown).

The most common pages that you'll use in Best Calculator are the *Calculator* and *Advanced* pages. The Calculator page (shown in the example) is a simple calculator.

The Advanced page is where you can access the angle (trigonometry), logarithm and other advanced math functions. When you are on a wide screen, selecting Advanced will display both the advanced functions and the regular calculator.

The available pages are:

1. BC BASIC lets you program your calculator using the popular BASIC language.
2. Calculator is a common calculator
3. Advanced includes trigonometry and logarithm functions
4. Format lets you change how the results are displays
5. Constants lists common physical constants (like gravity)
6. Memory lets you save, recall and change memory slots
7. Programmer's calculator is for hex and binary operations
8. Statistical calculator performs basic statistics on columns of numbers
9. Conversions and tables lets you convert between common units like converting liters to gallons. It also includes an ASCII table and Unicode lookup



Use the menu on the left to pick a calculator page to use

The available solvers are

1. Electrical Engineering
  - a. Voltage, Current and Resistance ( $V=IR$ )
  - b. Resistors
  - c. Resistor color code
  - d. Capacitors
2. Financial
  - a. CAGR (Compound Annual Growth Rate)
  - b. Mortgage calculations
  - c. WACC (Weighted average cost of capital)
3. Geometry
  - a. Circles
  - b. Right Triangles
  - c. Slope
4. Health
  - a. Ideal Heart Rate
  - b. BMI (Body Mass Index) and BMI for Kids
  - c. Pulse
5. Dice

The feedback page lets you give feedback about Best Calculator. We always include requested features in each new release!

**Are you using a Microsoft keyboard?** You can program in the Best Calculator as the calculator program. See the Advanced Windows Features chapter for details.

**Back Button on the Phone.** The Back button lets you quickly switch between any two calculator pages or solvers. Once in a page, pressing back button once shows the menu. Pressing back button again switches you to the page you were at before.

For example, from the menu, go to the Calculator page. Tap the back button, and you are at the menu. Tap the Advanced page to go to the advanced page. Tap the Back button again to get to the menu, and once more to get to the Calculator page.

## 2 NUMBERS AND COMMON CALCULATIONS

### 2.1 SIMPLE ARITHMETIC

A room is 15 feet by 20 feet. How many square feet of carpet is required to cover the floor?

**Key in:**  $15 \times 20 =$

**Answer:** 300

Pressing the = key gives the answer to the entered formula. The calculator includes the normal + - × ÷ = math calculations.



Addition, Subtraction, Multiplication, Division, Equals, Entering numbers

### 2.2 CHAIN CALCULATIONS

Press the = key again to repeat the last calculation ( $\times 20$ )

**Key in:**  $15 \times 20 = =$

**Answer:** 6000

### 2.3 ALGEBRAIC ENTRY AND PARENTHESES

Calculations are performed as they are entered (“chain input”, or the confusingly-named “Algebraic entry”)

**Key in:**  $2 + 3 \times 4 =$

**Answer:** 20

The  $2 + 3$  is calculated first and that result is multiplied by 4; this is different from “school” arithmetic where multiplication and division are calculated first.

You can force the order of operations by using the left and right parenthesis keys.

---

**Key in:** 2 + ( 3 × 4 ) =

**Answer:** 14

---

The calculator will empty the display when you type the left parenthesis, will display the partial calculation (12) when you type the right parenthesis, and the final result when you type the equals sign (=) .

Best Calculator doesn't limit how deeply nested the parentheses are.

Best Calculator's Equation Input mode uses operator precedence where  $=1+2\times3$  is calculated as 7. This is similar to what some calculators call an "Algebraic Entry System with Hierarchy", "Algebraic Operating System", "Direct Algebraic Logic" or "Visually Perfect Algebraic Method".

## 2.4 EDITING ERRORS AND CLEARING THE DISPLAY

The C, CE and  $\boxtimes$  keys are used to edit input errors and clear the calculator. The Clear All key clears the calculator to its default state.

---

**Key in:** 22 + 33 = C

**Result:** 0

---

---

**Key in:** 22 + 33 CE 44 =

**Result:** 66

---

The CE (Clear Entry) key clears the current entry. In the example, it clears just the '33' entry. When you type in '44' and '=', the calculation finished as if you had just entered 22 + 44. The answer is '66'.

---

**Key in:** 22 + 33  $\boxtimes$  7 =

**Result:** 59

---

The  $\boxtimes$  (Delete) key deletes just the last number entered. In the example, it clears just the second '3' in the '33' entry. When you type in '7' and '=', the calculator finishes its calculation as if you had just entered 22 + 37. The answer is 59.

## 3 MEMORY KEYS

### 3.1 MEMORY OPERATION

The four memory keys let you save and recall values from the calculator.

Save a number to memory

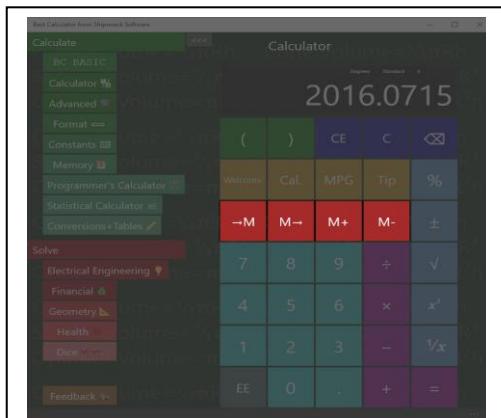
**Key in:**  $45 \rightarrow M$

**Answer:** 45 is displayed

Recall a number from memory

**Key in:**  $C M\rightarrow$

**Answer:** the screen is cleared and then 45 is displayed



Basic Memory operations include get from memory, store into memory

The C key will clear the display.  $M\rightarrow$  will copy the memory value to the display.

See the Memory Page section for advanced memory usage.

### 3.2 EXAMPLE

You are planning on doing a number of calculations using your local tax rate (in the example, 8.25%). Store the tax rate into memory and then use it later on.

Calculate the tax (8.25%) on two different values (10 and 20)

**Key in:**  $8.25 \rightarrow M$

**Key in:**  $10 + M\rightarrow \% =$

**Answer:** 10.825

**Key in:**  $20 + M\rightarrow \% =$

**Answer:** 21.65

The  $\rightarrow M$  key places the current number into memory. The  $M\rightarrow$  key retrieves the memory value and places it into the display just as if you had typed it.

### 3.3 MEMORY ADD AND SUBTRACT

The M+ key adds and the M- key subtracts the current number to or from the existing memory value.

**Key in: 45 →M M+ M→**

**Answer: 46**

First the number 45 is put into memory (45 →M). Then the memory value is incremented (M+). Finally, the incremented value is displayed (M→)

The M- key subtracts the current number to the existing memory value.

## 4 MORE MATH

### 4.1 SQUARE ( $x^2$ ) KEY

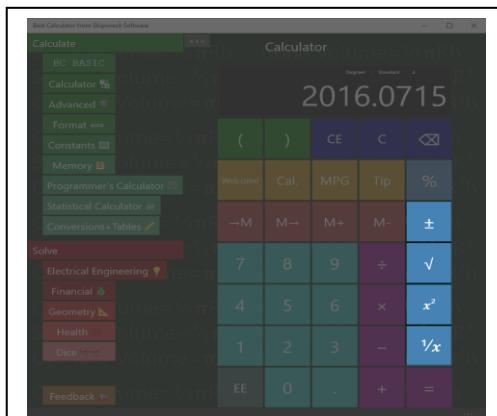
If you have a square 5 inches on a side, how many square inches is the square?

**Key in:**  $5 \times^2$

**Answer:** 25

The  $x^2$  key is the same as multiplying the current number by itself. In the example, it's the same as entering  $5 \times 5 =$

The  $x^2$  key squares the number instantly without you tapping the = key.



Best Calculator includes math functions like square root, square, inverse, change sign

### 4.2 SQUARE ROOT ( $\sqrt{}$ ) KEY

The  $\sqrt{}$  square root key finds the square root of the current number.

If you have a tile and know that it is 25 square inches, how long is it on a side?

**Key in:** 25  $\sqrt{}$

**Answer:** 5

### 4.3 INVERSE ( $1/x$ ) KEY

What's  $\frac{1}{3} + \frac{1}{3}$ ?

**Key in:** 3  $1/x$  + 3  $1/x$  =

**Answer:** 0.666

### 4.4 CHANGE SIGN ( $\pm$ ) KEY

The  $\pm$  key will change the current number from positive to negative or from negative to positive.

## 5 SCIENTIFIC NOTATION

---

### 5.1 USING THE EE KEY

Some numbers are too large or too small to enter conveniently. This is where you can use the EE key to enter your number in exponential (scientific) notation.

Avogadro's number is about  $6.022 \times 10^{23}$  and the atomic weight of water is about 18. How many molecules of water are in a single gram of water?

---

**Key in:** `6.022 EE 23 ÷ 18 =`

**Answer:** `3.34 E 22, or 3.34 × 1022`

---

To enter a negative exponent, use the – key. The ± will change the whole number from positive to negative.

The mass of a single oxygen atom is about  $2.68 \times 10^{-26}$  kilograms. What is the mass of 20 atoms of oxygen?

---

**Key in:** `2.68 EE - 26 × 20`

**Answer:** `5.36E-25 or 3.34 × 10-25`

---

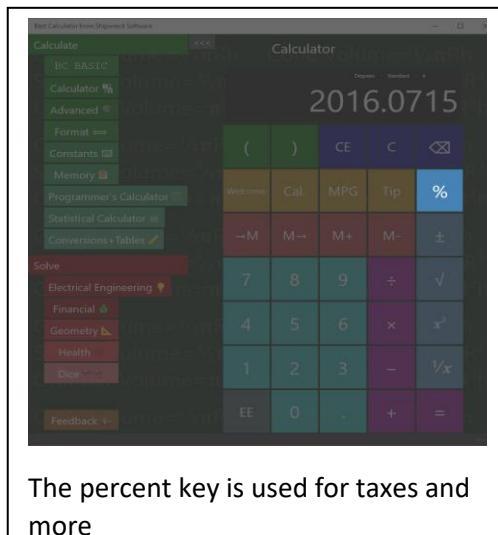
Best Calculator will display the result based on the selection in the Format section. If you are dealing with small numbers and are just seeing a "0" instead of a result in scientific notation, go to the Format screen and enter "Exponent".

## 6 PERCENT KEY %

### 6.1 PERCENT (%) KEY

Given that you're already entered an equation (for example  $72 - 20$ ), pressing the % key will convert the 20 into 20% of 72.

If you have entered only a single number (e.g., just "5"), then the % key will convert the 5 into 0.05 (as if calculating 5% of 100)



The percent key is used for taxes and more

### 6.2 CALCULATING SALES TAX

You are buying an item that costs \$72 and the sales tax is 5%. What is the total cost?

Key in:  $72 + 5 \% =$

Answer: 75.60

### 6.3 CALCULATING SALES WITH A PERCENT DISCOUNT

You are buying an item with a 20% pre-tax discount, and the sales tax is 5%. What's the total?

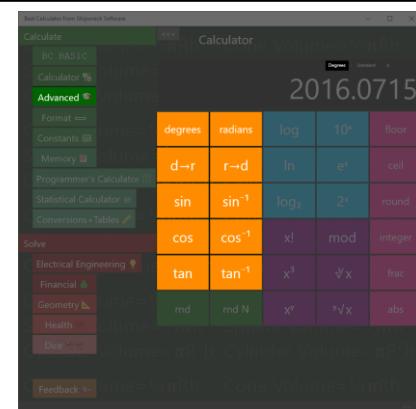
Key in:  $72 - 20 \% + 5 \% =$

Answer: 60.48

## 7 TRIGONOMETRY KEYS ON THE ADVANCED CALCULATOR

The trigonometry (angle) keys are part of the Advanced calculator.

Best Calculator includes sin, cos and tangent and their inverses.



Angle keys include sin, cos, tan, inverses, and degrees, radians and conversions.

### 7.1 CALCULATE IN DEGREES OR RADIANS

Calculations can be done in either degrees or radians. The display will show (in small type) whether you are currently in **degree** mode or **radian** mode.

Press the degrees key to switch to degree mode; press the radians key to switch to radian mode.

Press the **d→r** key to convert a number in degrees to a number in radians.  
Press the **r→d** key for the reverse conversion from radians to degrees.

Convert  $30^\circ$  to radians

**Key in:** 30 **d→r**  
**Answer:** 0.5236

The **d→r** and **r→d** keys work regardless of the **degrees** and **radians** settings.

### 7.2 SIN, COS, TAN

Calculate the sin of  $30^\circ$

**Key in:** 30 **sin**  
**Answer:** 0.50

If you get an answer of -0.9880, the calculator is in *radians* mode; key in **degrees** to switch to calculating in degrees

**Key in:** radians  $0.25 \times \pi = \cos$   
**Answer:** 0.7071

You can perform the same calculation using parenthesis

**Key in:** radians  $(0.25 \times \pi) \cos$   
**Answer:** 0.7071

The result of sin, cosine and tangent are always between -1 and 1.

### 7.3 INVERSE SIN, COS, TAN

(Also called arcsine, arccosine and arctangent)

Given the sin, cosine or tangent of an angle, you can get the original angle back out. The result will be an angle between  $0^\circ$  and  $360^\circ$  degrees or 0 to  $2\pi$  radians

Calculate the inverse sin (arcsine) of 0.5

**Key in:** 0.5  $\sin^{-1}$   
**Answer:** 30

If you get the answer 0.5236, the calculator is in radians mode; key in **degrees** to switch to degree mode.

The input values must be between -1 and 1; otherwise a NaN value is calculated.

The notation  $\sin^{-1}$  is the John Herschel notation; it means “inverse sine” and not “raised to a power”.

## 8 LOGARITHMS ON THE ADVANCED CALCULATOR

Logarithm keys are part of the advanced calculator.

Calculate the logarithm (base 10) of 100, 1000 and 100000,

**Key in:** 100 log

**Answer:** 2

**Key in:** 1000 log

**Answer:** 3

**Key in:** 100000 log

**Answer:** 5



Best Calculator includes logarithm and exponent keys

With base-10 logarithms, the log of a number is related to how many digits long the number is.

Best Calculator lets you calculate logs in three bases:

- The log key calculates using base 10
- The ln key calculates using base e (also called the natural logarithm)
- The log<sub>2</sub> key calculates using base 2 (binary, also called lb)

Each key is paired with the corresponding power key: 10<sup>x</sup>, e<sup>x</sup>, and 2<sup>x</sup>.

The base 2 logarithm is used by computer programmers to determine how many *bits* are required to hold a number of a certain magnitude.

How many bits are needed to hold a number that hold 26 distinct values?

**Key in:** 26 log2

**Answer:** 4.7004

## 9 POWERS, ROOTS, X!, MOD ON THE ADVANCED CALCULATOR

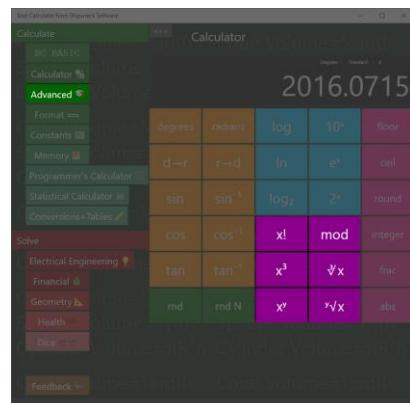
### 9.1 FACTORIAL (X!) KEY

Calculate  $6!$  (6 factorial)

**Key in:**  $6 \times!$

**Answer:** 720

$6!$  is another way of writing  $6 \times 5 \times 4 \times 3 \times 2 \times 1$



Powers and roots plus factorial and the mod operator.

### 9.2 MOD KEY

The Mod (Modulo) key calculates the remainder of a number. The remainder is the part that's left over when one number is divided by another.

What is the reminder of  $7 / 4$ ?

**Key in:** 11 Mod 4 =

**Answer:** 3

4 goes into 11 2 times with 3 left over.

### 9.3 CUBE ( $x^3$ ) AND CUBE ROOT ( $\sqrt[3]{x}$ ) KEYS

What is  $4.5^3$ ? (4.5 raised to the 3<sup>rd</sup> power, or  $4.5 \times 4.5 \times 4.5$  )

**Key in:** 4.5 x<sup>3</sup>

**Answer:** 91.125

What is cube root of 27?

**Key in:** 27  $\sqrt[3]{}$

**Answer:** 3

## 9.4 ARBITRARY POWER ( $x^y$ ) AND ROOT ( $y\sqrt{x}$ ) KEYS

Best Calculator calculates numbers raised to arbitrary powers and take arbitrary roots. The powers do not have to be integers.

What is  $6^4$

---

**Key in:** 6 x<sup>y</sup> 4 =

**Answer:** 1296

---

What is the 4<sup>th</sup> root of 32?

---

**Key in:** 32 y<sup>v/x</sup> 4

**Answer:** 2.3784

---

# 10 ROUNDING AND ABS ON THE ADVANCED CALCULATOR

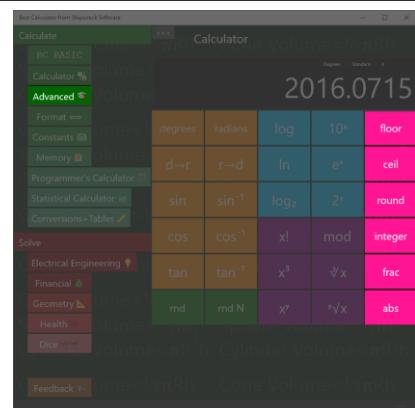
Rounding keys are in the advanced calculator.

**Floor:** round downwards to be a smaller number. The floor of a negative number (like -3.5) is rounded to a smaller number (-4)

**Ceil:** round upwards to a larger number. The ceil of a negative number (like -3.5) is rounded up to a larger number (-3)

**Round:** round towards to closest number. For example, 3.2 round is 3; 3.8 round is 4. Numbers that are exactly half-way between will round to the nearest even number (1.5 rounds to 2 and 4.5 rounds to 4)

**Integer:** rounds towards zero. The integer value of 4.5 is 4; and the value of -5.5 is -5. It is like removing everything past the decimal point.



Rounding, remainder and absolute value

(smaller) (larger)

The number line. Numbers further to the right are larger; numbers further to the left are smaller. -3 is smaller than -2 because it's further left.

**Frac:** returns the fraction part of a number

Calculate the remainder of 5.83

**Key in:** 5.83 remain

**Answer:** 0.83

**Abs:** convert negative numbers into positive numbers.

What is the absolute value of -4.5?

**Key in:** 4.5 +- abs

**Answer:** 4.5

# 11 RANDOM NUMBERS ON THE ADVANCED CALCULATOR

---

Best Calculator has two different random number keys.

The rnd key will place a random number between 0 and 1 into the result.

The rnd N key will place a random integer into the display; it will be between 1 and the number in the display.

Make a random number between 1 and 12

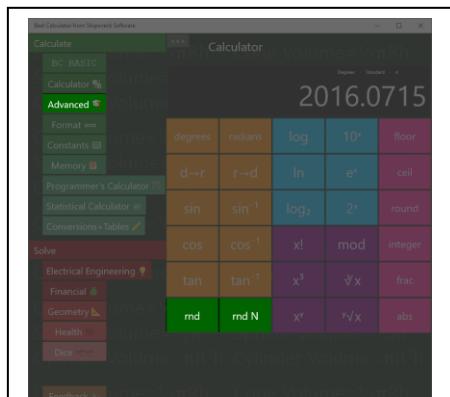
**Key in:** 12 rnd N

**Answer:** 3 (might be any number 1 to 12)

Make a random number between 0 and 1

**Key in:** rnd

**Answer:** 0.841 (might be any number 0 to 1)



Random Numbers

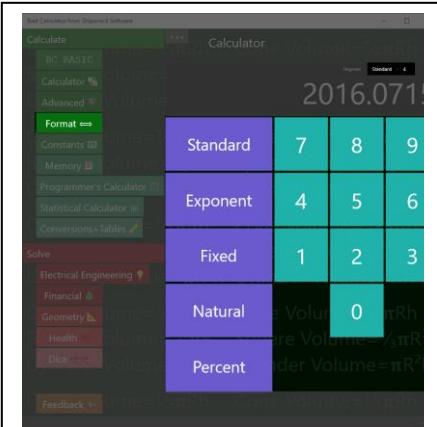
## 12 FORMATTING

Set how your results are displayed with the Format page.

Best Calculator can display numbers as regular numbers ("456.123") or as exponents ("4.56123E+002")

You can pick how many numbers after the decimal place to display.

The current formatting is always shown on the result display, right above the result. In the example, the display is "standard with 4 digits past the decimal point"



Change how your results are displayed with the Format page.

### 12.1 STANDARD FORMATTING

Regular number or exponent?	Automatically selected
Special features	"zero suppress" removes extra zeros after the decimal point.

The standard format is the one that's on by default when you first start the calculator. It will automatically switch between regular number and exponential form. The number of digits past the decimal place is a *maximum* number of digits; extra zeros are automatically suppressed (as is the decimal point).



In the example, up to 4 digits will be printed after the decimal point. The actual number (456.123000) has only zero beyond the ".123", and so they are suppressed.

## 12.2 EXPONENT FORMATTING

Regular number or exponent?	Always exponent
-----------------------------	-----------------

Sometimes called scientific notation, the exponent format is useful when you are dealing with large numbers much of the time.



In the example, the display (4.5612E+002) represents the number  $4.5612 \times 10^2$ . This in turn is  $4.5612 \times 100$ , or 456.12. The value is rounded from its real value (456.123) because the display has been set to only show 4 digits of precision.

A negative exponent (4.5612E-002) is less than zero (0.045612)

Exponent notation is equal to .NET's "E" format

## 12.3 FIXED FORMATTING

Regular number or exponent?	Regular number
Special features	Can overflow the display

The fixed format always displays using regular notation with a certain number of figures after the decimal point. It's often used when dealing with repeated calculations where each calculation should display the same way



In the example, exactly 4 digits will be printed after the decimal point.

If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

Fixed formatting is equal to .NET's "F" format

## 12.4 NATURAL FORMATTING

Regular number or exponent?	Regular number
Special features	Displays with commas

Similar to Fixed formatting, Natural formatting will display the number with commas separating the units.



If the number is too large to fit into the display, the display will be resized. After a certain point, the number will no longer fit, and will be truncated.

Fixed formatting is equal to .NET's "N" format

## 12.5 PERCENT FORMATTING

Regular number or exponent?	Regular number
Special features	Number is displayed as a percent: it's multiplied by 100 and displayed with a percent sign

The percent format is used when you need to show a number as a percent.



In the example, the number has been automatically multiplied by 100 and a percent sign is displayed. Note that the "real" number inside the calculator isn't changed: adding 1 to the example (456.123) results in 457.123 which is displayed as 457,12.3000%

Percent formatting is equal to .NET's "P" format

## 13 CONSTANTS

The Constants page lets to pick common physical constants often used in calculations.

### 13.1 $\pi$ AND e

Press the  $\pi$  key to set the display to the value of  $\pi$  (about 3.1416).

Press the e key to set the display to the value of e (about 2.7183)

Calculate  $4 * \pi$

**Key in:**  $4 \times \pi =$

**Answer:** 12.5664

$\pi$ 3.14159...	$e$ 2.71828...
$g_n$ Earth gravity	$c$ speed of light
NaN Not a number	$\infty$ Infinity
	$-\infty$ Minus infinity

Best Calculator includes useful constants

### 13.2 $g_n$ , c AND $N_a$

$g_n$  (about 9.8) is the standard gravity in metric units; it's the gravitational acceleration on Earth, measured in meters/second<sup>2</sup>.

c is the speed of light (about 299792458) in meters/seconds

$N_a$  is Avogadro's number (about  $6.022 \times 10^{23}$ ) is the number of atoms or molecules in one *mole* of a substance. For example, there are  $N_a$  atoms of carbon (specifically, <sup>12</sup>C) in 12 grams of carbon.

### 13.3 NaN, $\infty$ AND $-\infty$

Best Calculator includes three special constants that can't be typed in otherwise.

NaN (Not a Number) means that a numeric value is not meaningful. For example, an inverse sin ( $\sin^{-1}$ ) is only meaningful for input values -1 to 1. Calculating  $\sin^{-1}$  of 2 will result in a NaN result.

$\infty$  is positive infinity, and  $-\infty$  is negative infinity.

## 14 MEMORY PAGE

The Memory page gives you access to 10 named memory slots. Each slot can be set, changed, and named.

The first memory slot (normally called Memory0) is the memory used by the main Calculator screen for memory operations.

### 14.1 SHOW THE OPERATION OF MEMORY0

The main calculator screen includes 4 memory keys ( $\rightarrow M$ ,  $M \rightarrow$ ,  $M+$  and  $M-$ ). These directly manipulate the first memory slot.

Enter 45 into memory slot 0. Go to the Calculator page.

**Key in:** 45 →

**Answer:** 45 is displayed

Now go to the Memory page. The first memory slot is set to 45.

### 14.2 SAVING AND ROAMING

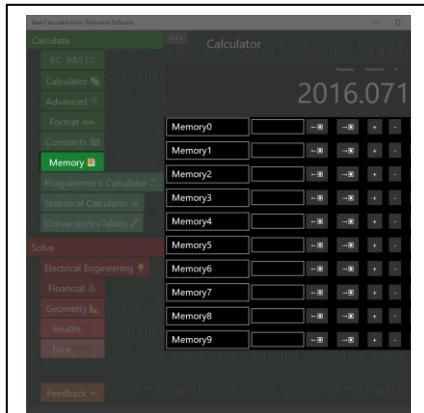
The memory values are roamed: when you set a memory value, the value and name are roamed to all of the computers where you're logged in with the same Microsoft Account. The values will be saved between calculator runs; you can exit the calculator and the values will be preserved for the next time you run Best Calculator.

### 14.3 NAME A MEMORY CELL

Click on the name of a memory cell to rename it.

Set, Increment, Decrement

Click on a memory value to change it. You should only enter numeric values!



Memory operations: rename, set, get from display, copy to display, increment, decrement, clear.

Click on the + key to increment a memory value

Click on the – key to decrement a memory value

#### **14.4 GET FROM DISPLAY AND COPY TO DISPLAY**

The  $\leftarrow\square$  key gets data from the display into the memory slot.

The  $\rightarrow\square$  key copies data from the memory slot to the display

#### **14.5 CLEAR**

The C key clears the memory slot

#### **14.6 MEMORY AND BC BASIC**

The values in the memory page are shared with the BC Basic Memory extension. You can get and set values from the calculator memory slots from within BC BASIC. Changes you make there will be reflected in the memory page.

## 15 DATES PAGE

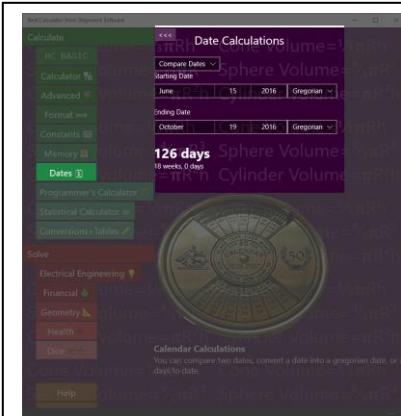
The Dates page lets you perform various date calculations.

### 15.1 COMPARE DATES

How many days are there between June 15<sup>th</sup> and October 19<sup>th</sup>?

**Key in:** Select Compare Dates from the drop-down. Set the Starting Date to June 15<sup>th</sup>, and the Ending Date to October 19<sup>th</sup>. Keep the Calendar type as Gregorian.

**Answer:** 126 days (18 weeks exactly)



Date calculations include calculating days between dates, adding days and calendar conversions

You can compare dates in multiple formats

- Gregorian is the standard calendar used in America and Europe
- Hebrew (הַלְוִת קָעֵבָרִי) is a lunisolar Jewish calendar
- Hijiri (گامشماری هجری خورشیدی) is a calendar used in Iran and Afganistan
- Japanese calendar
- Julian is the common calendar used in Europe and America until being replaced by the Gregorian calendar
- Korean
- Persian
- Taiwan
- Thai
- Umm Al-Qura

### 15.2 CONVERT TO GREGORIAN

The Great Feast of Theophany is celebrated on the 6<sup>th</sup> of January, Julian. In 2016, what day is this in the Gregorian calendar?

**Key in: Select Convert to Gregorian from the drop-down. Set the Starting Date to January 6<sup>th</sup>, 2016 and set the calendar type to Julian.**

**Answer: January 19, 2016**

---

### **15.3 ADD DAYS**

A project is started on April 19, 2016, and will take 21 days. What day will the project end on?

**Key in: Select Add days from the drop-down. Set the Starting Date to April 19<sup>th</sup>, 2016 and keep the calendar type as Gregorian. Set the Number of days to add to 21.**

**Answer: Tuesday, May 10, 2016**

---

# 16 HEX, DECIMAL, OCTAL, BINARY ON THE PROGRAMMER'S CALCULATOR

The Hex, Decimal, Octal and Binary entry and conversions are part of the Programmers Calculator.

## 16.1 SETTING THE MODE

Press the bin, oct, dec or hex keys to switch to binary (base-2), octal (base-8), decimal (base-10) or hexadecimal (base 16). The display will show which mode the Programmer's Calculator is in.

To enter a value, simply press the keys 0 to 9 or A-F. Valid keys will be displayed in Cyan; invalid keys are gray. In the example, the calculator is in decimal mode so that keys 0 to 9 are all valid. In binary mode, only keys 0 and 1 are valid; in octal mode only keys 0 to 7, and in hexadecimal mode keys 0 to 9 and A to F.

Add the number 1A to the number 12

---

**Key in:** hex 1 A + 1 2 =  
**Answer:** 2C

---

## 16.2 CONVERTING BETWEEN BASES

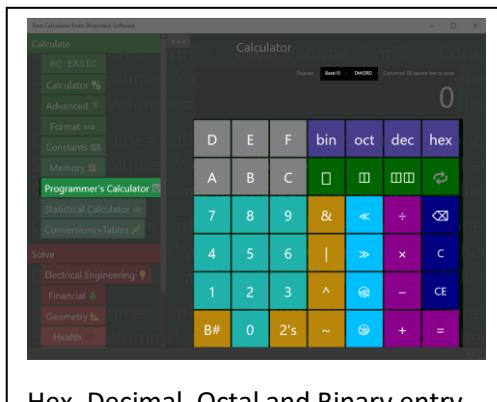
To convert from the current base to a different base, simply press the new base number. The existing display will be converted to the new base

Convert  $2C_{16}$  into decimal

---

**Key in:** hex 2 C dec  
**Answer:** 44

---



Hex, Decimal, Octal and Binary entry and conversions are built in

# 17 BIT OPERATORS ON THE PROGRAMMER'S CALCULATOR

The Bit operators are part of the Programmer's calculator

## 17.1 B# (COUNT BITS)

Use the B# key to count the number of '1' bits.

How many bits are '1' bits in the hex number 44?

**Key in:** hex 4 4 B#

**Answer:** 2

Hex 44 is binary 01000100. Only two of those bits are '1' bits.

## 17.2 INVERSE ( $\sim$ ) AND 2'S COMPLEMENT

The  $\sim$  key will invert each of the bits of the current value. This is also called the "1's complement" of the number.

The 2's key calculates the "2's complement" of a number. This is what (almost every) computer stores as the negative value of a number.

How does a computer store -1?

**Key in:** dec 1 2's

**Answer:** FFFFFFFF

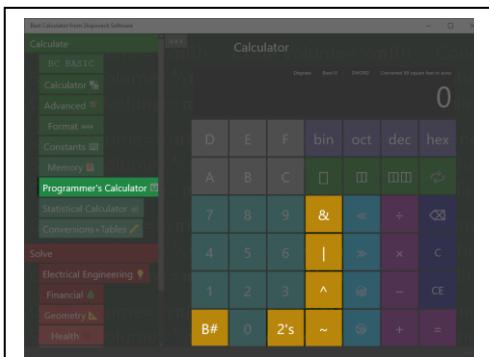
## 17.3 AND (&), OR (|), XOR (^)

The And, Or and Xor keys will and, or xor (exclusive-or) two numbers.

What is 3 AND 4?

**Key in:** 3 & 4 =

**Answer:** 7



Basic Memory operations: get from memory, store into memory

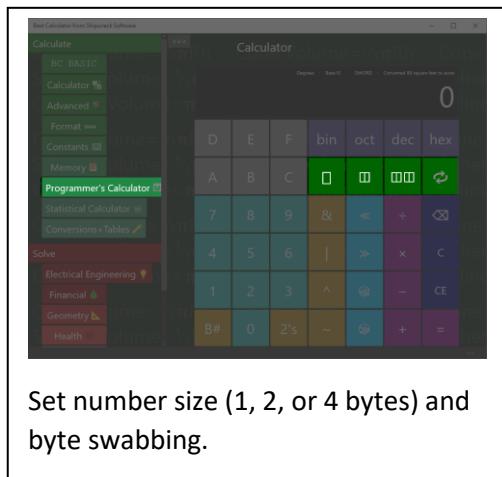
# 18 BYTES AND SWABBING

## ON THE PROGRAMMER'S CALCULATOR

The bytes and swap operators are part of the Programmer's calculator.

### 18.1 BYTES

Many programmer calculations change depending on the number of *bytes* that are involved in an operation. Common sizes are byte (□), word (□□) and dword (4 bytes, □□□□). The result display shows the number of bytes as “BYTE”, “WORD” and “DWORD”. Results that are larger than the current setting will be truncated.



Set number size (1, 2, or 4 bytes) and byte swabbing.

### 18.2 ⚡ SWAB (SWAP BYTES) KEY

The key performs the SWAB (swap bytes) operation. This is commonly used when dealing with network operations: most network protocols send data using “big endian” format while most PCs are in “little endian” format.

What's the network byte order for port 80? Port numbers are sent as 2 bytes.

**Key in:** □ dec 8 0 ⚡

**Result:** 20480

Explanation:

- switches to WORD (2 byte) mode
- dec switches to decimal mode
- 8 0 puts the number 80 into the display
- ⚡ (SWAB) switches the first and second byte around

# 19 SHIFT OPERATORS ON THE PROGRAMMER'S CALCULATOR

The Shift operators are part of the Programmer's calculator

The shift operators (and especially the rotate keys) use the byte setting.

`<<` is shift left

`>>` is shift right

`<<+o` is rotate left

`>>+o` is rotate right

Shift hex F0 left by 2 bits

**Key in:** hex F 0 << 2 =

**Answer:** 3C0

In binary, F0 is 1111 0000. Shifted left 2 bits, the result is 11 1100 0000, which is 3C0. Bits introduced on the right are zero.

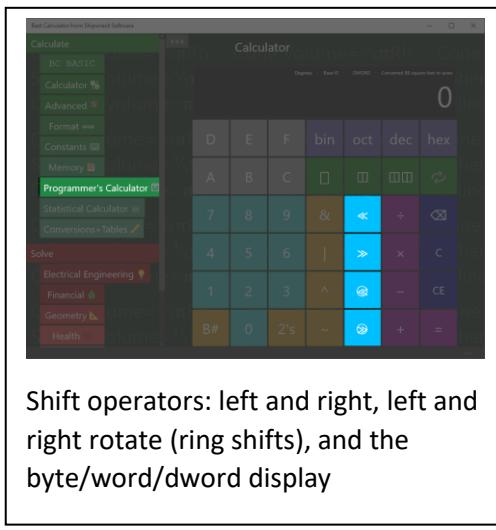
Ring-shift left F0 by 1 bit as a single byte

**Key in:** hex F 0 ⌘ 1 =

**Answer:** E1

In binary, F0 is 1111 0000. With a normal shift, the bits that "shift out" will be dropped and new bits are zero. With rotate (ring shift), the bits would have been dropped are reintroduced on the other side.

In binary, F0 is 1111 0000. When rotated, the top bit (1) is put in as the lowest bit, resulting in 1110 0001, or E1.



# 20 PROGRAMMER'S

## MATH

The programmer's calculator can act as a regular calculator, but only for integers.

Divide 7 by 3

**Key in:**  $7 \div 3 =$

**Answer:** 2

Unlike the regular calculator, where  $7 \div 3$  is 2.3333, the Programmer's Calculator is strictly an integer-only calculator.

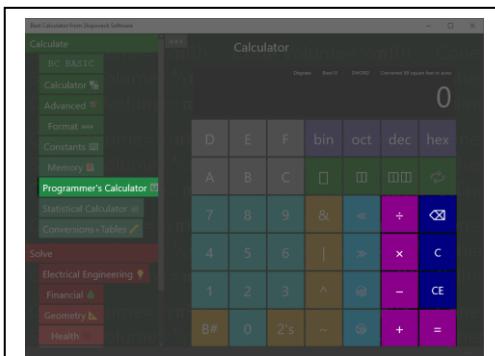
Additionally, the byte settings will force some results to be truncated

Multiply  $99 \times 99$  in byte mode

**Key in:**  $99 \times 99 =$

**Answer:** 73

In the regular calculator,  $99 \times 99$  would be 9801. This is  $(256 \times 38 + 73)$ . Only the bottom byte (the 73) is kept from the calculation. The rest is discarded.



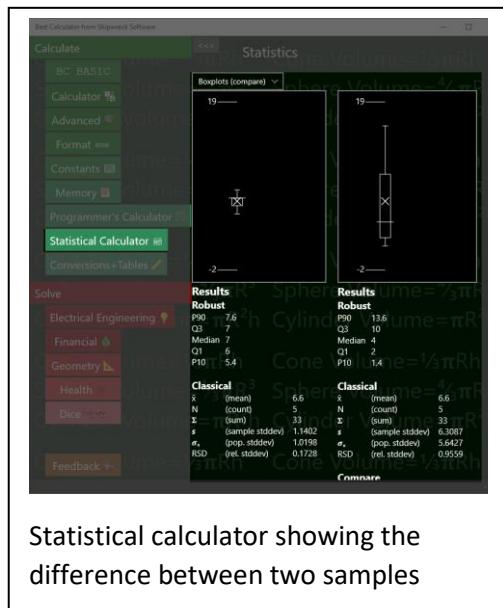
The programmer's calculator performs standard math operations

# 21 STATISTICAL CALCULATOR

The Statistical calculator takes in one or two lists of numbers and computes robust and classical statistics, T-tests and linear regression between two paired groups of numbers

The statistical calculator will display the data either as one or two boxplots (specifically, Tukey boxplots) or as an XY scatterplot.

Best Calculator always computes all statistics. As you type numbers into the data boxes Best Calculator computes and displays the results.



Statistical calculator showing the difference between two samples

## 21.1 PARTS OF THE STATISTICAL CALCULATOR SCREEN

The Statistical calculator screen is divided into three parts: the graphs, the computed results, and the data. There are two data entry boxes, a left box and a right box, resulting in a left hand data set and a right-hand data set.

1. **Graphs at the top** present a pictorial view of your data. You can choose the kind of graphs: two independent boxplots, two boxplots for comparison (their Y axes will be set to be the same), an XY Scatter plot, or no graph
2. **Computed results** for the left and right data sets. Both data sets will always compute Robust and Classical statistics. The right-hand data set in addition will compute a T-Test comparison between the left and right data sets and a linear regression between the left and right data sets.
3. **Left and right data sets** and their data entry boxes. The data is simply a list of numbers; you can copy and paste from Excel or simply type the data in.

## 21.2 ENTERING DATA

Enter data into the left and right data boxes.

When Best Calculator starts, each box is marked “Data” and contains some default values so you can see what kind of statistics and graphs Best Calculator will produce.

To enter data, simply click on either box. Data should be entered with one item per line. To enter a lot of data, you can paste the data into the text box.

Best Calculator is a calculator, not a spreadsheet, and does not understand file

formats like CSV (comma separate value) files, and does not have a way to read a file from disk.

You can paste values directly from Excel. In Excel, highlight and copy the column of numbers you want to compute statistics for. Then click on the text box in Best Calculator and paste.

Data which are not understood as numbers will be ignored. The first line with text in it will be the title of the data. For example, if you copy a column of numbers from Excel, you can include the column header and it will be used as the data title. As you enter data, Best Calculator will automatically recalculate your statistics.

Enter data either into the left or right text boxes. Statistics are automatically computed as you type.

The left box still has the default data in it; the right box has been cleared. The graphs have been turned off.

## 21.3 CLASSICAL STATISTICS (RESULTS)

Both the left and right hand data sets will compute classical statistics. Classical statistics work best with symmetrical, bell-shaped **normally distributed** data.

Lines that are text, not numbers, or which are blank are never included in the values.

### Classical

$\bar{x}$	(mean)	6.6
N	(count)	5
$\Sigma$	(sum)	33
s	(sample stddev)	1.1402
$\sigma_n$	(pop. stddev)	1.0198
RSD	(rel. stddev)	0.1728

### Mean

The mean ( $\bar{x}$ ) of the numbers is the arithmetic mean, or average of the numbers. It's calculated by adding up all the numbers and dividing by the count of the numbers.

### Count

The count (N) of the number of numbers in the data box.

### Sum

The sum ( $\Sigma$ ) is the total of all of the numbers added together.

### Sample Standard Deviation

The sample standard deviation (sample stddev, or s) is very similar to the population standard deviation, but the N value used is (N-1). The standard deviation informs you of the overall spread of the data.

### Population Standard Deviation

The population standard deviation (pop. stdev , or  $\sigma_n$ ) is the standard deviation assuming that the data in the data box is the entire population and not a sample. It's similar to but always a little smaller than the sample standard deviation.

### Relative Standard Deviation

The Relative standard deviation (RSD) is a normalized version of the sample standard deviation. It's the computed by dividing the sample standard deviation by the mean. It's useful for determining the "spread" of a sample without having to mentally compare the standard deviation with the mean.

**ROBUST STATISTICS (RESULTS)**

Both the left and right hand data sets will compute robust statistics. Robust statistics are less sensitive to outliers than classical statistics. The values computed by Best Calculator are used to display the Boxplot.

Lines that are text, not numbers, or which are blank are not included in the value.

**Robust**

P90	7.6
Q3	7
Median	7
Q1	6
P10	5.4

Computed robust statistics

The **median** value is often used as a representative measure of the data. When the data is symmetrical, the mean (classical) and median (robust) are the same; when data is skewed, the median represents a more typical member of the data while the mean is more weighted towards the high end.

To calculate the robust statistics, the data is sorted. Each robust data point is the value that is a certain percentile of the overall data. For example, the median is the 50% percentile; half of the data points are larger than the median value, and half smaller.

Point	Percentile	Comments
P90	90 <sup>th</sup>	The P90 measure is used to help estimate the spread of the data. In the boxplot, the P90 point is marked with a small circle.
Q3	75 <sup>th</sup>	The Q3 (third quartile) is the $\frac{3}{4}$ mark. The box in the boxplot is bounded by the Q# and Q1 points.
Median	50 <sup>th</sup>	The median (Q2) is taken from the exact middle of the sorted data set. It's marked by a long horizontal bar in the box of the boxplot.
Q1	25 <sup>th</sup>	The Q1 (first quartile) is the $\frac{1}{4}$ mark
P10	10 <sup>th</sup>	The P10 measure is the opposite of the P90 measure. In the boxplot, the P10 is also marked with a small circle.

## REGRESSION (RESULTS)

Use the regression data and the linear regression chart to tell if two data sets are related.

The data shows the thickness (in mils) of a layer of silver deposited onto a computer chip after a certain amount of time in a furnace.

The number of minutes is entered into the left hand data box; the thickness is entered into the right-hand data box. If both data sets have the same number of data points, Best Calculator will compute the linear regression statistics.

Values calculated are

**Slope** and **Intercept** is the best fit

for a single straight line through the data. These values can be placed directly into the standard formula for a line,  $y = mx + b$ . The slope value is the  $m$  value, and the intercept value is the  $b$  value.

The **correlation** coefficient says how close of a match the data is. If the data isn't correlated at all, the correlation is 0. A negative correlation means that as one value increases, the other decreases (also called a negative correlation). A value of 1 or -1 means that the data is perfectly correlated.

The **StdErr Line** (standard error of the line) and **StdErr Slope** (standard error of the slope) are tests of how noisy the data is and how good of a fit the computed slope and intercepts are.

## Regression

Slope	0.0037
Intercept	0.0094
Correlation	0.9814
StdErr Line	0.0007
StdErr Slope	0.0004

Computed linear regression statistics

Sample data:

Minutes	Thickness
1	0.0132
1.5	0.0151
2	0.0167
2.5	0.0177
3	0.0211

## 21.4 COMPARE WITH T-TESTS (RESULTS)

When two data sets are entered, Best Calculator will compute a Welch's t-test value.

Welch's t-test are used to decide if the two data sets are "the same" or "different". The boxplots are used informally for the same purpose.

### Compare

The two values are probably DIFFERENT.

The p-value 0.0145 is <= target 0.05

test Welch's t-test

p 0.0145

df 13.9359

t 2.7894

Computed robust statistics

Note that the t-test is not perfect way to tell if two samples are "different". For example, the numbers [10, 20, 30, 40] will be reported to be "possibly the same" as [25, 25] even though a person would declare them to be very different.

In the example, the two data sets are from two types of chemical analysis (<http://www.fao.org/docrep/w7295e/w7295e08.htm>)

The computed p-value helps answer the question, "are these two data sets likely to be from the "same" kind of data. If the p value computed is small (set to 0.05 in Best Calculator), the two data sets must not be the same and are therefore different. If the p value is large, then no conclusion can be drawn: perhaps the data is different, but perhaps it's not.

Welch's t-test is a more modern version of the Student's t-test. It's been shown to be as good as the Student's t-test when the sample variances are the same, and better than they aren't.

Although the primary calculation from the Welch's t-test is the p value, the df (degrees of freedom) and t statistic are also presented.

## 21.5 BOXPLOTS (GRAPH)

Boxplots are a simple way to visually compare two data sets.

The tick marks on the left (1.4 and 3 in the example) tell you about what the range of data is.

The central box shows the inter-quartile range (IQR) for the data. Exactly  $\frac{1}{2}$  of the data falls within the central box;  $\frac{1}{4}$  is less than and  $\frac{1}{4}$  is more than the box.

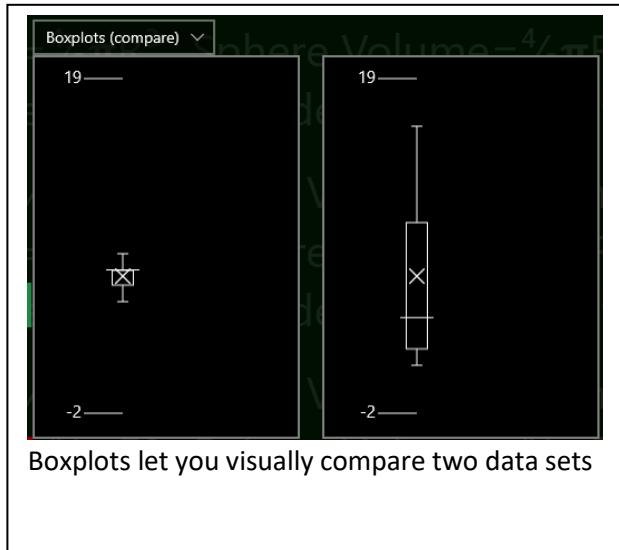
The horizontal line in the center of the box is the median of the data set.

The X in the boxplot (usually, but not always, somewhere in the box) is the average value. The X is the only part of the boxplot computed with classical statistics instead of robust statistics.

The whiskers can each be as long as  $1.5 \times$  the IQR. They are trimmed back to show the furthest-away data point that fits in the whisker. Outliers are not drawn.

The plots also shows the 10% and 90% percentile data points. These are useful when determining whether a process has a significant number of outliers or not. These are shown as the small circles, and will only be displayed when the data has at least 10 data points.

When you're displaying two data sets, you can **compare** the two boxplot or display them as **independent**. In compare mode, the two boxplots are display using the same scale. In independent mode, the two boxplots are display with their own scale.



## 21.6 XY SCATTER-PLOTS (GRAPH)

XY Scatterplots are a quick way to compare *paired* data.

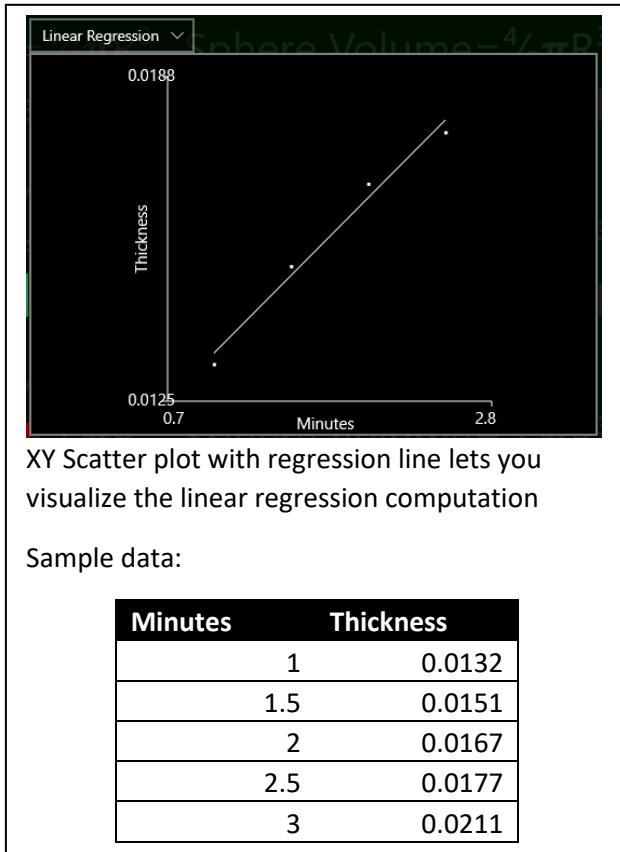
The sample data used in the example is the same data used in the Regression example.

To make a scatterplot from your data, enter the *independent* data into the left hand data box and the *dependent* data into the right hand data box. Then chose “Linear Regression” as your graph type.

Each pair of data is plotted as a small dot, and the results of the regression analysis are drawn as a straight line. The headers will be displayed as the X and Y axis labels.

The min and max axis values are picked based on the data and are not settable.

Highly correlated data will be plotted close to the regression line; uncorrelated data will be plotted randomly, neither close to nor avoiding the regression line. The *correlation coefficient* is computed and displayed as part of the correlation results.



## 22 CONVERSIONS

Conversions (like inches to centimeters) are in the Conversions and Tables section.

Tap that section first, and then tap the conversion page you want.

### 22.1 CONVERTING

All of the converters work the same way.

**Conversions include area, energy, length and more.**

To convert a value, enter in the value you have (like “Square inches”) and read off from the resulting values.

Example: An apartment is 330 square feet. How many square yards is that?

**Key in:** tap ‘Conversions and Tables’ and then tap ‘Area’. Tap the text box next to ‘Square Feet’ and type in 330.

**Answer:** read off 36.67 from the Square yards box

As you enter a number, the surrounding values are automatically calculated.

How many acres are in a square mile?

**Key in:** tap ‘Conversions and Tables’ and then tap ‘Area’. Tap the text box next to ‘Square Miles’ and type in 1.

**Answer:** read off 640 from the Acres box

### 22.2 COPY TO AND FROM THE CALCULATOR RESULTS DISPLAY

You can copy a number from the calculator results display into any of the text boxes with the  $\leftarrow$  key that's next to each text box.

Your apartment is 10 yards by 4 yards. How many square feet is it?

**Key in:** in the main calculator, enter  $10 \times 3 =$

Tap ‘Conversions and Tables’ and then tap ‘Area’. Tap the  $\leftarrow$  key next to the text box marked ‘Square Yards’. It will be set to

360.

**Answer: read off 360 from the Square Feet box**

---

A square foot of carpet costs \$2.04. What is the cost to carpet your apartment?

**Key in: from the last example, tap the  $\rightarrow\blacksquare$  key next to the square feet box. Go to the main calculator; note that it has been set to 360. Key in  $\times 2.04 =$**

**Answer: \$734.40**

---

## 22.3 AREA CONVERSIONS

Best Calculator can convert between

- Square inches
- Square feet
- Square yards
- Acres
- Square miles
- Square centimeters, meters, hectares, and square kilometers.

A hectare is 10,000 square meters, or 1/100 of a square kilometer.

## 22.4 ENERGY CONVERSIONS

Best Calculator can convert between:

- Ergs
- Joules
- Kilowatt-Hours
- Calories
- Food calories (KCAL)
- Donuts
- BTUs
- Therms

## 22.5 LENGTH CONVERSIONS

Best Calculator can convert between

- Inches

- Feet
- Feet + Inches (output only)
- Yards
- Miles
- Centimeters
- Meters
- Kilometers

## 22.6 TEMPERATURE CONVERSIONS

Best Calculator can convert between

- Degrees Celsius (also called Centigrade)
- Fahrenheit
- Kelvin (absolute temperature in the metric system)
- Rankine (absolute temperature in Fahrenheit)

## 22.7 WEIGHT

Best Calculator can convert between

- Ounces
- Pounds
- Pounds + Ounces (output only)
- Short tons (2000 pounds)
- Long tons (2240 pounds)
- Grams
- Kilograms
- Tonnes (1000 kilograms; 2204 pounds)
- MMT (millions of metric tons)
- Grains
- Troy Ounces
- Troy Pounds
- Tolä (about 0.41 ounces)
- Sèr (80 Tolä)
- Maund (40 Sèr)

## 22.8 FARM VOLUMES (US)

Best Calculator can convert between

- Cups
- Pints (2 Cups)
- Quarts (2 Pints)
- Gallons (4 Quarts)
- Pecks (1 Gallons)
- Bushels (4 Pecks)
- Liters

Given a weight in pounds per bushel, Best Calculator will also calculate the weight of an amount in bushels.

## 23 ASCII TABLE

The ASCII (American Standard for Computer Information Interchange) is a common way for computers to encode English-language text as numbers suitable for computer operation.

Programmers often need to convert from characters (like “@”) into their ASCII equal (the number 64).

The example shows the conversions for the “@” characters, sometimes called the “Ray” in honor of Ray Tomlinson, the creator of Internet email in the 1971.

The screenshot shows the 'ASCII Table' section of the Best Calculator software. The table is a grid where each row represents a character or symbol and each column represents its ASCII value. The first few rows are as follows:

Value	Character	Value	Character	Value	Character	Value	Character
0	0	1	@	P	~	p	+
2	!	3	A	Q	^	a	>
4	"	5	B	R	_	b	r
6	#	7	C	S	o	c	s
8	\$	9	D	T	oo	d	t
*	%	:	E	U	e	u	?
:	&	:	F	V	f	v	=
)	(	)	G	W	g	w	
*	*	:	H	X	h	x	
:	:	:	I	Y	i	y	
+	+	:	J	Z	j	z	
,	,	:	K	[	k	{	
-	-	:	L	\	l		
=	=	:	M	]	m	}	
.	.	:	N	^	n	~	
>	>	:	O	-	o	%	

ASCII table

The screenshot shows the 'ASCII Table' section of the Best Calculator software. The table is a grid where each row represents a character or symbol and each column represents its ASCII value. The first few rows are as follows:

Value	Character	Value	Character	Value	Character	Value	Character
0	0	1	@	P	~	p	+
2	!	3	A	Q	^	a	>
4	"	5	B	R	_	b	r
6	#	7	C	S	o	c	s
8	\$	9	D	T	oo	d	t
*	%	:	E	U	e	u	?
:	&	:	F	V	f	v	=
)	(	)	G	W	g	w	
*	*	:	H	X	h	x	
:	:	:	I	Y	i	y	
:	:	:	J	Z	j	z	
+	+	:	K	[	k	{	
,	,	:	L	\	l		
-	-	:	M	]	m	}	
=	=	:	N	^	n	~	
.	.	:	O	-	o	%	

64 is the decimal value of @  
40 is the hex value of @  
100 is the octal value of @

In the example, the “@” is shown along with the decimal (64), hex (40) and octal (100) values.

Example: replace a character with its percent encoding (used in encoding URLs from arbitrary characters). The rules for percent encoding is to replace the single character with three characters: a “%” sign and then two digits with the character’s hex value. The @ hex value is 40, so to replace a single “@” in a URL, you need to replace it with “%40”.

# 24 UNICODE

Unicode is the worldwide specification for characters from all languages, including emoji and useful math symbols. Best Calculator supports Unicode 9.0, introduced in 2016

## 24.1 SEARCH RULES

Enter a set of search terms into the search box. As you type, the display is updated with matching characters.

A search term is a match if it's any part of the character description; short terms (1 or 2 characters long) and terms that start with = have to match a word exactly. Examples: a matches LATIN CAPITAL LETTER A and =phi matches LATIN SMALL LETTER PHI. Longer terms will match anywhere in a description.

Example: LATI matches LATIN CAPITAL LETTER A

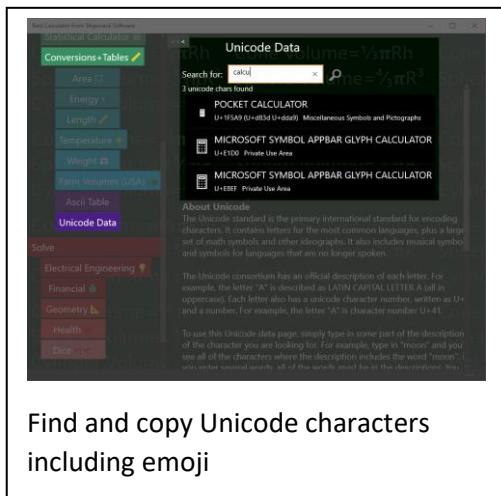
Terms that start with a minus sign (-) are anti matches; they invert the normal processing.

Terms that start with U+ will match a Unicode number exactly.

Each Unicode character includes its Unicode number (like U+41 for LATIN CAPITAL LETTER A), the character itself, the official description and the official alternate description. In addition, searches include the Unicode “Block” name. For example, IPA will match all of the characters in the IPA Extensions block.

## 24.2 COPYING CHARACTERS

Right-click on a result to bring up the app bar. Tap the clipboard to copy the character to the clipboard.

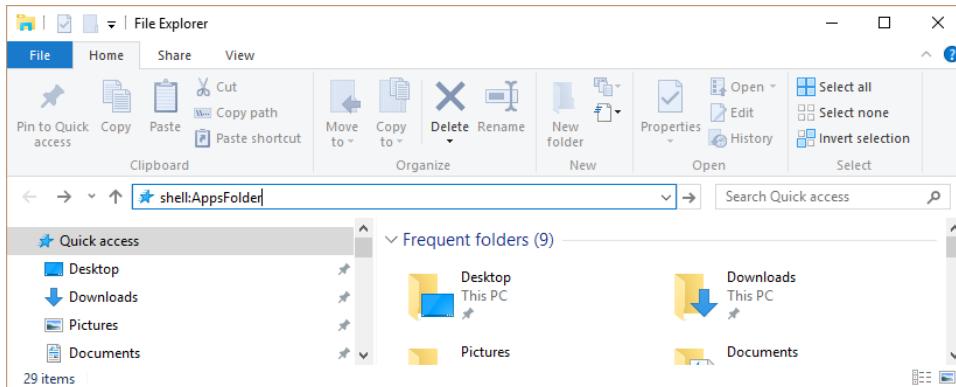


## 25 ADVANCED WINDOWS FEATURES

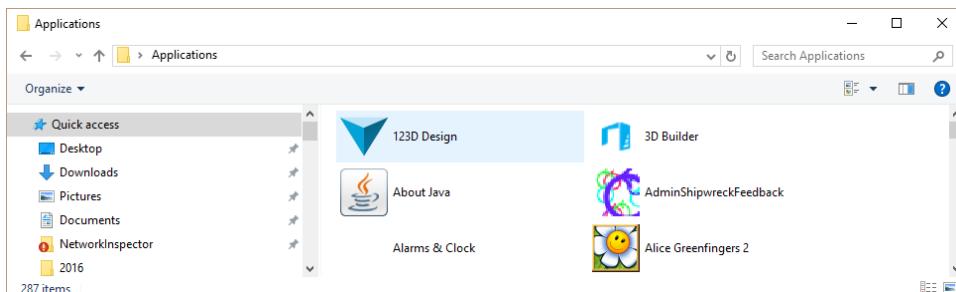
### 25.1 SHORTCUT ON THE DESKTOP

You can add Best Calculator as a shortcut on your desktop.

The easiest way is to start the regular Windows Explorer (press Windows-E). In the address bar, enter shell:AppsFolder (see the picture below).



The Explorer will show you all of your installed apps.



Right-click Best Calculator and select “Create a Shortcut”. You will be told that you can only create a shortcut on the desktop; click “Yes”. A shortcut to Best Calculator will be placed on your desktop.

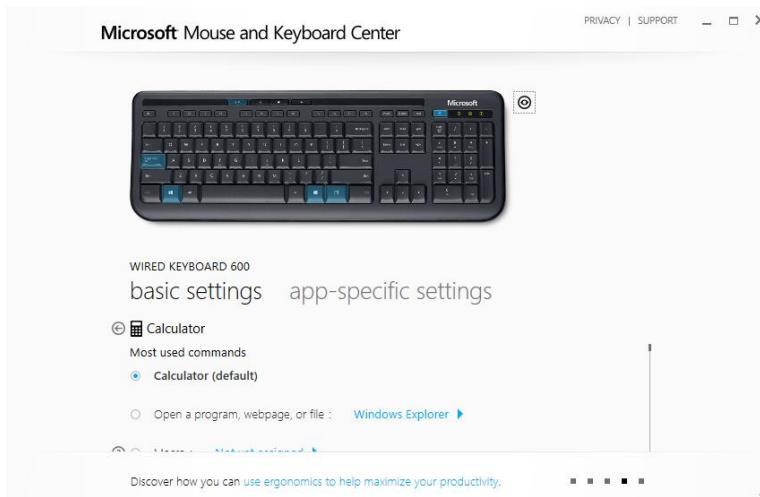
## 25.2 SET AS THE CALCULATOR KEY

Windows keyboards often include pre-programmed buttons to launch common applications.

You can program the launch buttons to launch Best Calculators.

Run the Microsoft Mouse and Keyboard Center by pressing the Windows key and typing “Mouse and Keyboard”. The Mouse and Keyboard center program will show up.

In the example, the center is programming a Microsoft Wired Keyboard 600 with a Calculator key. We’re going to program the calculator key to start Best Calculator.



Under the Calculator setting, tap “Open a program, webpage, or file”. Then tap the Windows Explorer button and *carefully* enter

```
explorer shell:AppsFolder\48425ShipwreckSoftware.BestCalculator_jh2negtepkzpr!App
```

Then press the back button.

That’s it! The calculator button should now launch the calculator app. If you mis-type the long command line, the Windows Explorer will launch instead.

## 26 A BRIEF HISTORICAL NOTE

---

As the first programming language designed for students, BASIC holds a special place in the history of programming languages.

```
LET X = (7+8)/3
PRINT X
END
```

*Sample BASIC program from the first Dartmouth BASIC instruction manual, 1964*

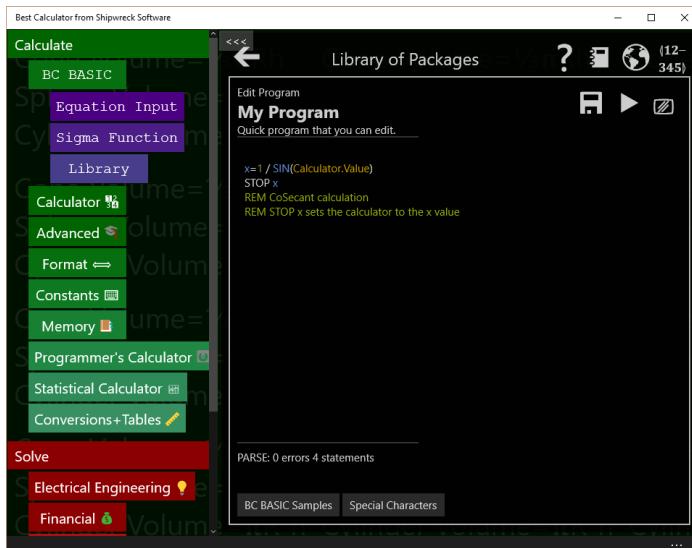
Calculator BASIC lets you program your own simple functions into the Best Calculator, extending its abilities to exactly match your needs. Everyone who's a specialist has the same problem with typical calculators: there's some standard calculation in your profession, but no calculator includes those particular functions on the keyboard. Best

Adding a program to the Best Calculator is easy. The BC BASIC Library key shows you all the programs you've written, neatly organized into individual *packages*. The BC BASIC environment gives you access to a wide variety of sample programs. Naturally, Help is just a click away. Within the programming environment you can add or edit your new program, run your program, and bind your program to any of the five programmable keys on the keyboard.

The programs you write will *roam* between your computers and your phone. You can write a program on one computer, and it will automatically roam to your other computers. (This requires that you've signed in with a Microsoft Account, of course). You can also *Export* your packages to a file, and then *Import* the package into Best Calculator running on another computer.

## 27 EQUATION INPUT: YOUR FIRST PROGRAM

Tap the BC BASIC and then the Equation Input keys. An Edit windows is displayed with a mini program. Your first program is written for you so you can see what a program looks like.



### 27.1 WHAT IS A PROGRAM?

A program is a list of *statements*, each of which performs some action. Many useful programs are just a single equation – for example, to convert feet to acres, or do a financial calculation

### 27.2 HOW DO I RUN MY PROGRAM?

Press the ► (Run Program) key to run the program. When you're in the editor, the final result will be displayed. The final result is either the result of the STOP statement or the result of the last assignment statement.

### 27.3 HOW CAN I PRINT SOME TEXT?

Use the PRINT statement like this:

```
PRINT "Calculation complete!"
```

## 27.4 HOW CAN MY PROGRAM USE THE CALCULATOR VALUE?

The best way is the *Input expression*.

```
X = INPUT DEFAULT 3.4 PROMPT "Enter a tax rate"
```

Input is only for numeric values (you can't get a person's name, for example)

## 27.5 HOW CAN I WRITE AN EQUATION?

Use LET, the *assignment statement*, and an *expression* (equation). For example, suppose you need to calculate the circumference of a circle and the equation is PI times the diameter. Your equation might be

```
LET circumference = PI * diameter
```

Or suppose you need to calculate the inner diameter of a pipe given the outer circumference and the pipe thickness.

```
LET thickness = 3  
LET outerCircumference = 30  
LET outerDiameter = outerCircumference / PI  
LET innerDiameter = outerDiameter - 2 * thickness
```

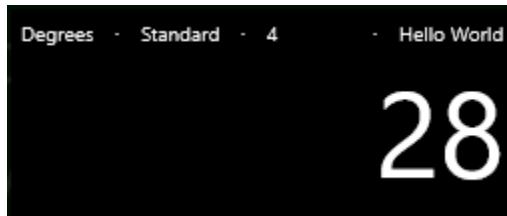
## 27.6 HOW CAN I WRITE TO THE CALCULATOR?

You can write to the calculator in three ways.

You can write a little text message to the top of the calculator display with Calculator.Message.

```
Calculator.Message = "Hello World"
```

Which then show up on the calculator results windows



Use Calculator.Value to get or set the result value

Calculator.Value = 7337



If you end your program with a STOP <value>, the value will be placed into the calculator display. Or, if there isn't a stop, then the last assignment statement (LET statement) evaluated is sent to the calculator. The *equation* or expression form (starting a line with just an equals sign), is considered an assignment even though the value isn't assigned to a variable.

## 27.7 WHY DO SOME EXAMPLES START WITH AN = SIGN?

Lines that just start with an equals sign are the “equation” form. You can put any expression (equation) on the right of the equals, and the value will be computed. If the

## 27.8 HOW CAN I MAKE SEVERAL DIFFERENT PROGRAMS?

See the section on using the library. You can write different programs and give them all different names. They will even roam between your different devices.

## 27.9 WHAT ARE ALL THE KEYS ON THE SCREEN?

The ↺ (Back) key will take you to the Library.

The ? (Help) key pops up this manual

The 🌐 (Web) key takes you to the Best Calculator web site.

↳

The 345↳ (Bind key) key lets you program one of the goldenrod (dark yellow) keys with your program. That way you can run your program straight from the regular calculator.

The  (Save) key will save your program.

The  (play) key will run your program. You can also press the F5 key.

The  (screen) key will clear the colorful output screen. That screen is only displayed if you PRINT output.

Under the edit area is the “PARSE” output; it tells you if your program “parses” correctly – it tells you if and when you make a mistake. Don’t fret too much about making parse errors; experienced programmer make them all the time

The BC BASIC Samples key show you a few simple samples to get started.

The Special Characters key lets you insert some of the hard-to-type characters that BC BASIC can use.

## 27.10 HOW CAN I LEARN MORE?

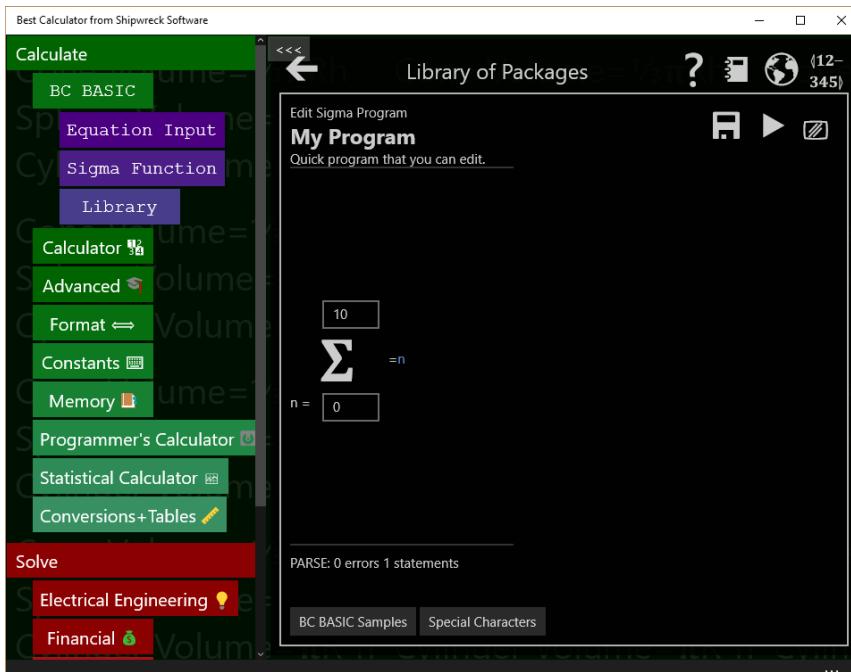
The Reference: Language Basics section tells you everything that makes up a valid BC BASIC program. As a helpful reminder, press the BC BASIC Samples key

to get a little pop-up with some common code snippets. Or press the big  key to see this manual.

A BC BASIC program is a series of statements (lines); each line is an equation. A common statement is *assignment*, which lets you do math. Other common statements are PRINT and INPUT to print results for the user and get numerical values from the user.

## 28 SIGMA FUNCTION: ADVANCED PROGRAMMING

Press the BC BASIC and then the Sigma Function Input keys. An Edit windows is displayed with a mini sigma expression program. Your first expression is written for you so you can see what a program looks like.



The Sigma Function page will run an expression that you provide, summing up the response.



Press ► (Play) or F5 and see the result

The Sigma function is run 11 times. Each time the variable n is set, first to 0, then to 1, and so on up to and including 10. The value of the expression is summed, and the overall value displayed.

Press the →□ key to copy the result into the calculator display.

The rest of the Sigma Function page is the same as the Equation Input page. The Samples include an example of the Taylor expansion to calculate the Sin of a value

```
REM TAYLOR expansion for SIN
REM Convert n(0,1,2,3,4) into series (1,3,5,7,9...)
series = n*2 + 1
x = Calculator.Value
val = x**series / Math.Factorial(series)
isOdd = (Math.Mod (n,2) = 1)
IF (isOdd) THEN val = -val
=val
```

The Taylor expansion for Sin requires a series of numbers 1, 3, 5, 7, and so on. The Sigma function only produces numbers 0, 1, 2, 3, ... .The first step is to convert the **n** value (0, 1, 2, 3, ...) into a series value (1, 3, 5). This is done with the line **series = n\*2+1**.

Next the code calculates the **x<sup>series</sup>** value and divides by **series!**. Odd parts of the sequence are supposed to be subtracted from the sequence, so those are negated.

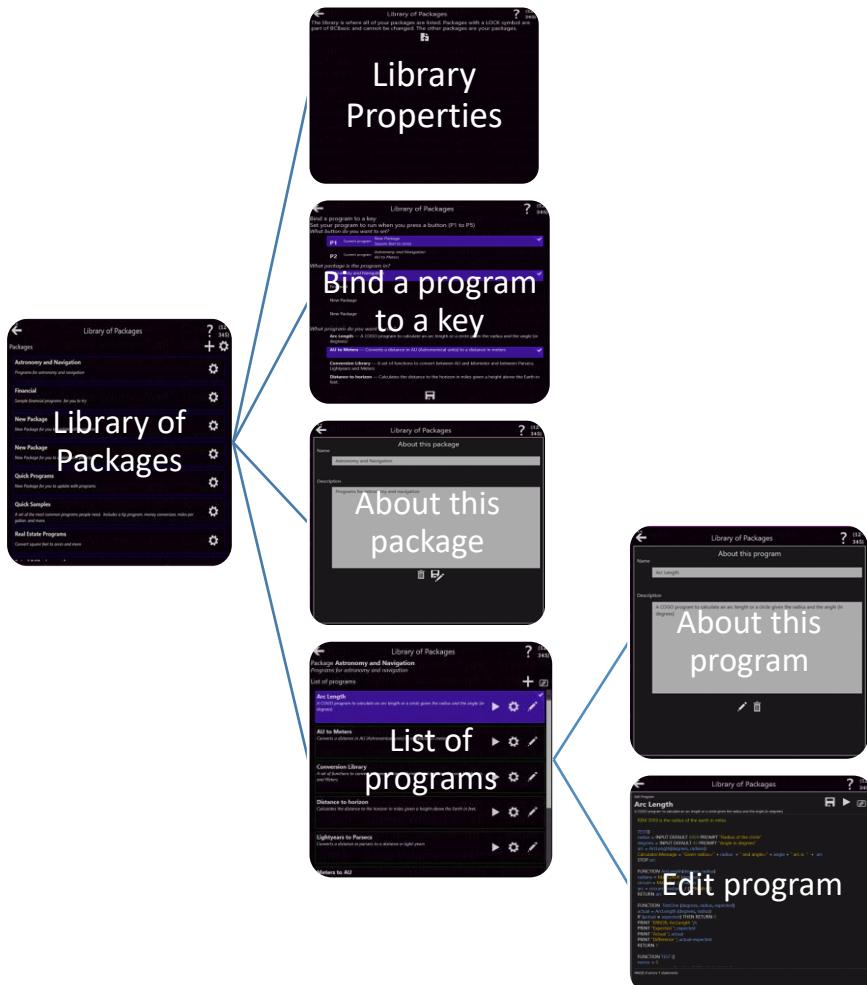
The **x** value is taken from the **Calculator.Value** number from the calculator display.

Lastly the resulting value for a single series value is return (=val).

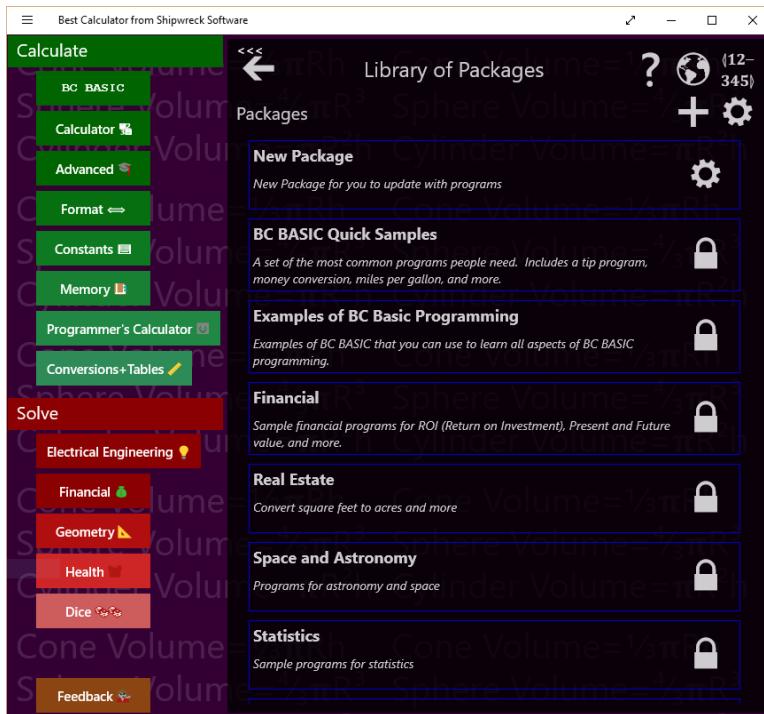
# 29 WHAT ALL CAN YOU DO IN THE BC BASIC ENVIRONMENT?

It's time for a introduction to all of the different dialogs you will use to create and run your programs.

## 29.1 ALL THE MAIN EDIT DIALOGS



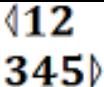
## 29.2 LIBRARY OF PACKAGES



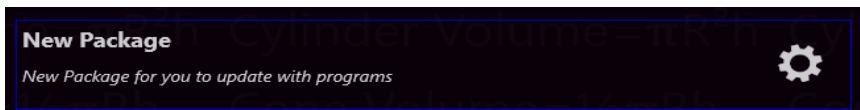
Library of Package is the first library dialog you see. It lists all of the packages that you can run, examine, or change.

In the Library of Packages dialog there are many important controls.

Control	How and when to use it
 Back arrow	<p>The big back arrow is present on all of the dialogs. Press it to return to the parent dialog. If you are in the Library of Packages dialog, the BC BASIC environment will be hidden, and you can see the Best Calculator display.</p> <p>Don't worry! None of your changes are lost. Just press the BC BASIC key to see the Library of Packages dialog again</p> <p>The little "chevron" arrow is a Best Calculator arrow; it hides the Best Calculator menu of calculators.</p>

 Help	Displays the main HELP PDF page using the default PDF reader (often a web browser).
 Buy the manual	Goes to Amazon.com. You can buy a copy of the Best Calculator manual from Amazon.
 Web site	Goes to the Best Calculator web site
 Bind Key	Displays the Bind a program to a key dialog. This lets you bind one of your programs to one of the P1, P2 and so on key in the Best Calculator keyboard. When you press a key, the program you selected will be run.
 Create new package	Creates a new package. The new package gets a name which you should change. Use the GEAR icon (  ) to change the name.
 Properties	Displays up the Library Properties dialog. From this dialog you can Import a BC BASIC package.

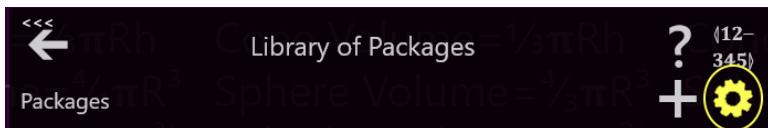
You can also tap on individual packages in the list of packages. Each package entry shows its name and description and includes a GEAR icon to display the package details.



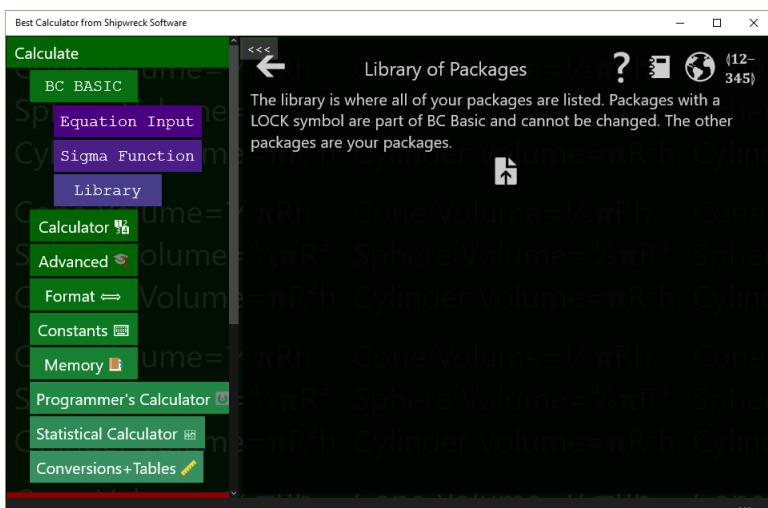
Control	When and how to use it
Tap on a package	Brings up the List of programs for that package. From the list you can add new programs and edit and run your programs.
 Properties	Bring up the About this package dialog. From that dialog you can change the name and description for a package.

## 29.3 LIBRARY PROPERTIES (IMPORT BC BASIC PACKAGE)

Bring up the Library Properties dialog by tapping the Library of Packages GEAR



key (  , highlighted). The Library of Packages header lets you go to the help file, bind a program to a key, add a package for your programs or display the library properties screen.



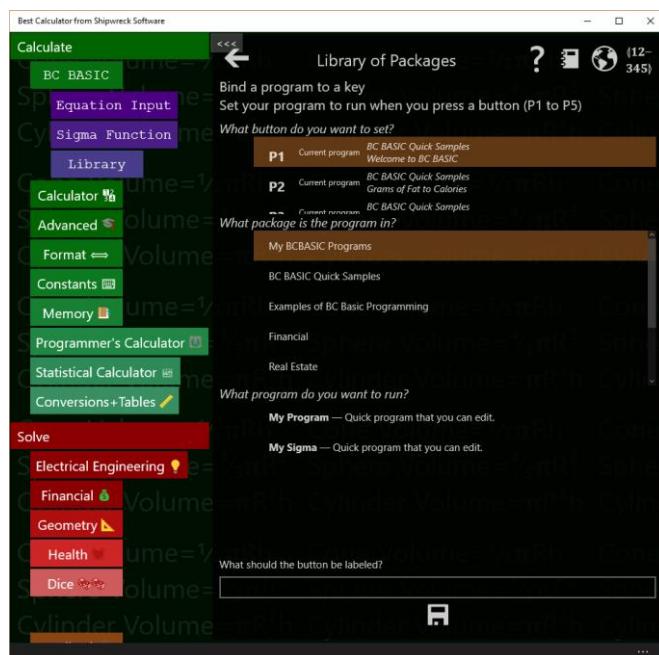
In the Library properties there is a single control

Control	When and how to use it
 Import	Brings up the Import dialog. Use this to import (read in) new BC BASIC packages from your job, your school, your friends, or even the internet.

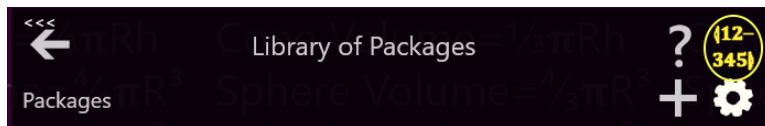
# Guide to Using Best Calculator

## 29.4 BIND A PROGRAM TO A KEY

Page | 71



Bring up the Bind a program to a key screen by tapping the Library of Packages BIND key (highlighted below)



There are several programmable keys which you can bind a program to. When you tap one of those keys, the program that has been bound will be run. When you first get Best Calculator, a selection of programs has already been bound.

To pick a program to run when a programmable key is pressed, select an answer to the three questions in the Bind a program to a key screen and tap SAVE.

The first question is *What key do you want to bind to?* Tap one of the keys in the key list (labeled P1, P2, P3 and so on) to pick a key to bind to. People often just pick key P1. The key list tells you what package and program the key is currently bound to. This helps you pick the right key to use.

The second question is *What package is the program in?* All the possible packages are listed. As you tap on a package, the next list changes to show the programs in that package. Tap on the package with the program you want to run.

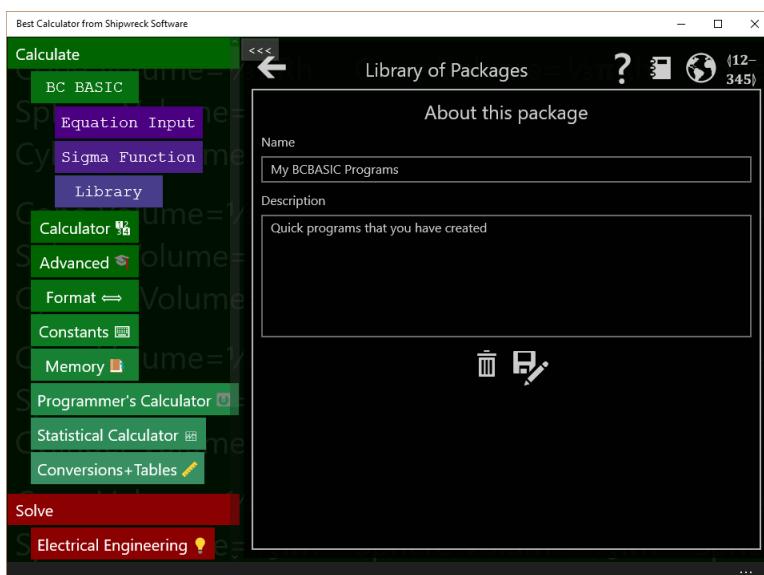
The third question is *What program do you want to run?* Tap the program to run.

Lastly, **be sure to tap the SAVE key (  ).** Your selection isn't saved until you press save.

You can keep on binding programs to more keys, or press the BACK ARROW (  )to go back to your last dialog box.

## 29.5 ABOUT THIS PACKAGE

Bring up the About this package by tapping either the GEAR (  ) or LOCK (  ) in the package listing (see highlighted) in the Library of Packages screen.



The About this package screen lets you change the name and description of a package, delete it, or export it (save it to an external file).

You can only modify your own packages (the ones with a GEAR icon).

Packages that came with Best Calculator (with a LOCK icon) cannot be modified.

### BC BASIC Quick Samples

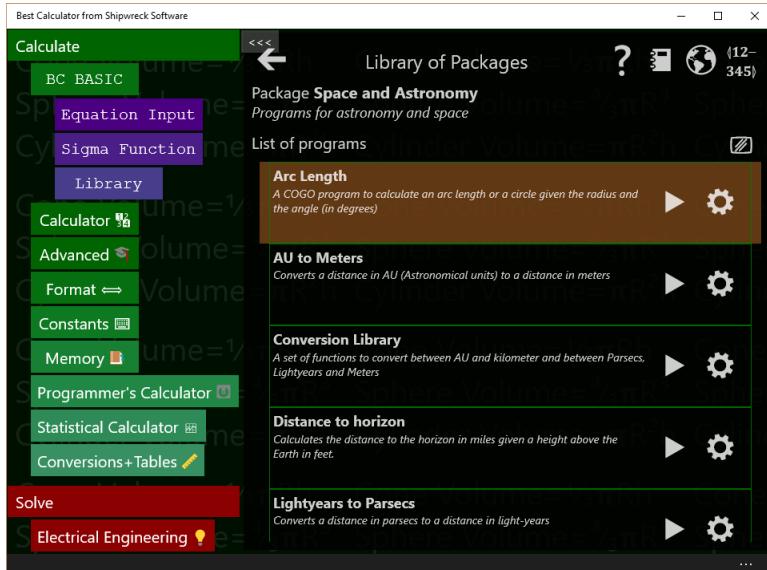
A set of the most common programs people need. Includes a tip program, money conversion, miles per gallon, and more.



Control	When and how to use it
Name	You can change the name of the package here. Just enter a new name. The name is automatically saved.
Description	You can change the description of the package here. Just enter or modify the description. It will be automatically saved.
	Deletes the package. Once deleted, you will not be able to bring the package back. You will be prompted to confirm the delete.
	Saves (exports) the package (and all the programs in it). You will be prompted for a file name to save to.  Once you Save (Export) a package, you can store it in OneDrive, save to a web page, or email to a friend or coworker. They can Import the package from the Library Properties page.

## 29.6 LIST OF PROGRAMS

Bring up the list of programs by tapping on a package in the Library of Packages



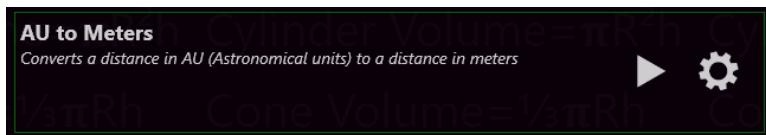
screen.

Packages contain multiple programs which you can run. The List of programs screen lets you see all and run the programs in a package, add new programs, edit programs, and more.

Control	When and how to use it
 Add Program	Adds a new program to the package. Once created, you will need to tap the GEAR (  ) to set the program's name and description.  You can only add programs to your own packages. Packages that came with Best Calculator cannot be modified.
 Clear Screen	Clears the output screen. If the output screen is not visible, you won't see a change (but it will be cleared nonetheless)

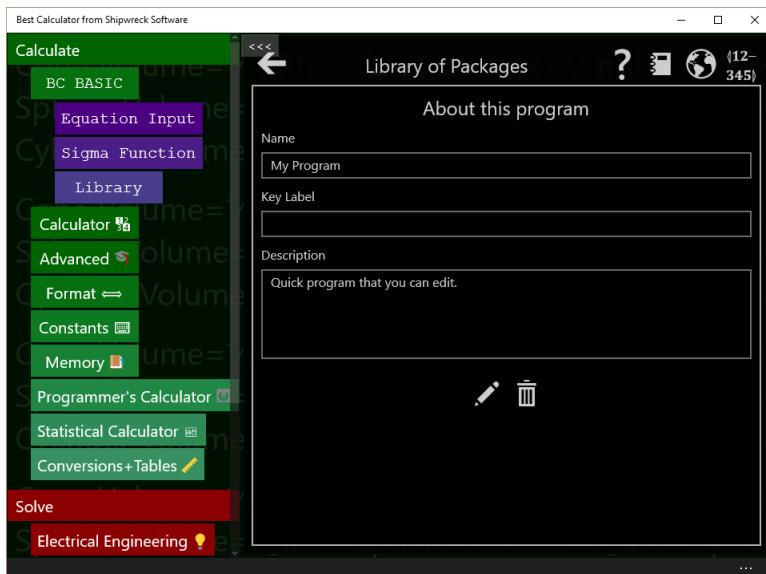
You can also tap on individual programs in the list of programs. Each program entry shows the name and description of the program and lets you run and

examine the properties of the program. If the program is one you wrote, you can also edit it.

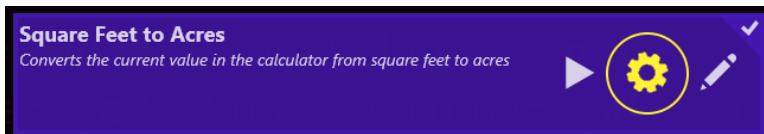


Control	When and how to use it
Tap on a program	Does nothing in particular ☺
 Run program	Runs the program from the start
 Properties	Displays the About this program dialog. From that dialog you can edit the program name and description.
 Edit program	Display the Edit Program dialog. From that dialog you can edit and run the program.  Packages that come with Best Calculator cannot be modified. For those programs, the Edit key will show you the source code for the program but will not let you change it.

## 29.7 ABOUT THIS PROGRAM



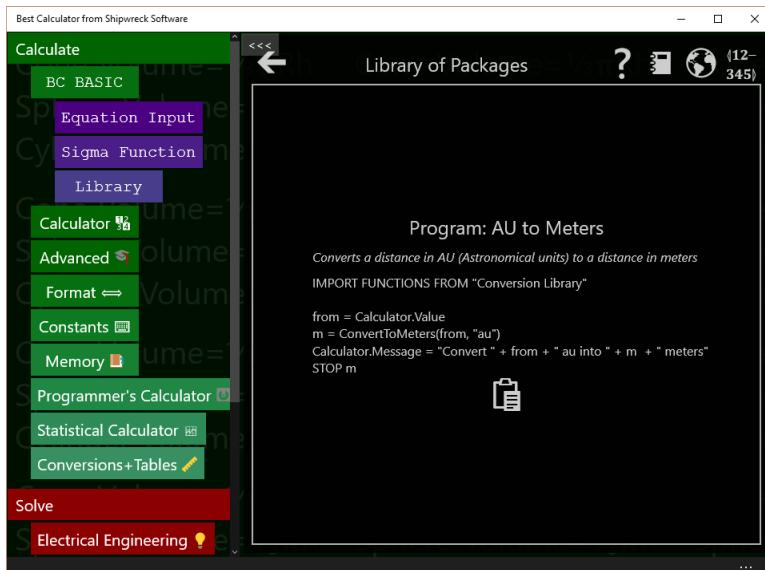
Bring up the About this program screen by clicking the GEAR icon in the program entry in the List of programs (see highlighted). The About this program screen lets you modify the name and description of a program, go straight to the edit program screen, or delete the program.



Control	When and how to use it
Name	You can change the name of the program here. Just enter a new name. The name is automatically saved when the program is saved.
Description	You can change the description of the program here. Just enter or modify the description. It will be automatically saved. When the program is saved.
	Display the Edit Program dialog. From that dialog you can edit and run the program.

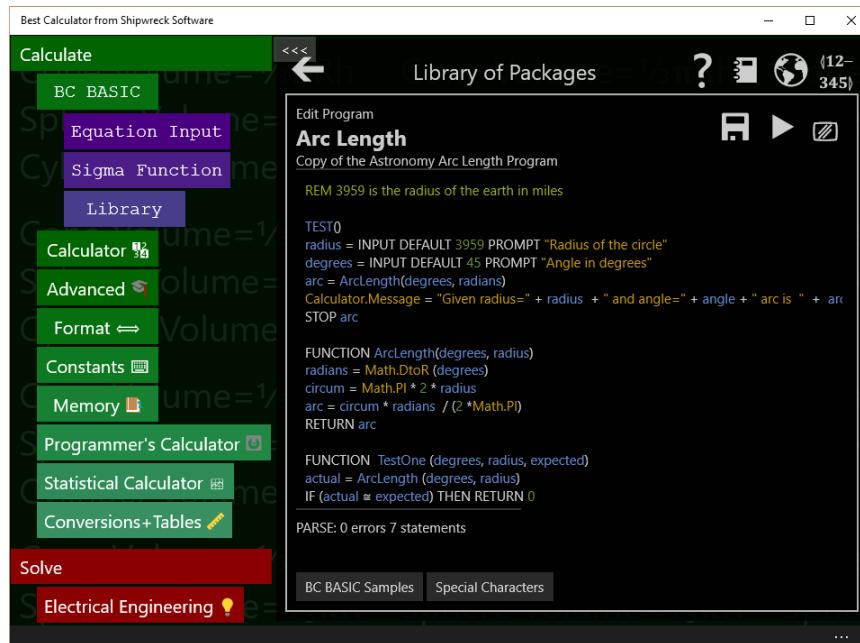
Edit program	You cannot edit programs that come with Best Calculator
 Delete	Deletes the program. Once deleted, you will not be able to bring the program back. You will be prompted to confirm the delete.

Programs that come with Best Calculator are locked and cannot be changed. Their About screen is a little different.



Control	When and how to use it
 Copy to clipboard	Copies the source code for the program to the clipboard.

## 29.8 EDIT PROGRAM



The Edit program dialog is where you enter your BC BASIC code.

Bring up the Edit program dialog by tapping the EDIT key on either the program

list or by tapping the EDIT ( ) key in the About this program screen.

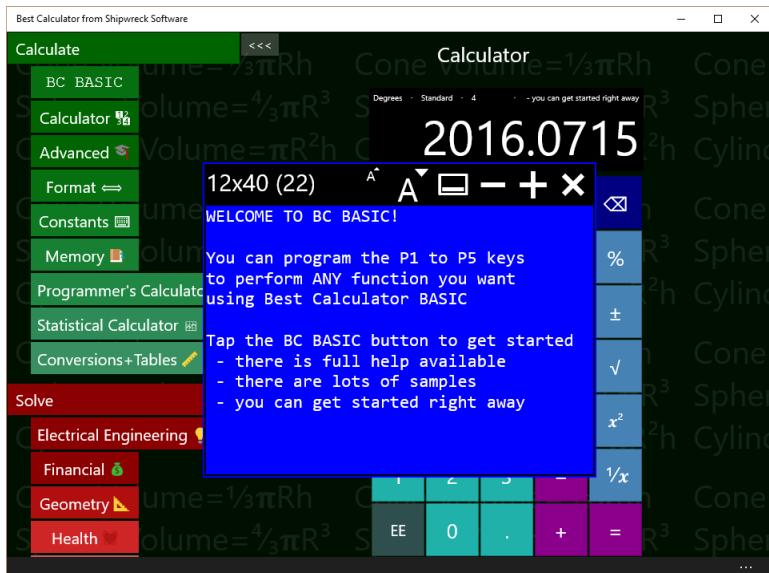
Control	How and when to use it
	Saves your changes. Your changes are automatically saved when you press RUN. Your program is saved as part of the package file; this is managed for you.
	Runs the program from the start. Some common errors when you press RUN but your program does not appear to run are: <ol style="list-style-type: none"> <li>Your program might have a syntax or other error that prevents it from running. You will know because the parse indicator will show an error</li> </ol>

	2. Your program might have run, but didn't display anything to the screen. The output is often displayed on the calculator display. Press the Calculator key to see the calculator screen.
 Clear screen	Clears the output screen. If the output screen is not visible, you won't see a change (but it will be cleared nonetheless)
<b>PARSE: 0 errors 7 statements</b> Parse indicator	<p>This indicates if your program compiles. BC BASIC automatically compiles your program as you type it, and tells you of any syntax errors.</p> <p>The editor uses <i>syntax coloring</i> on your code; different parts of the code will be displayed in different colors. When a syntax error is discovered, only the program up to the syntax error is colored; the rest shows up in white.</p>

There are two special keyboard keys while you are editing a program

Key	How and when to use it
Escape	Stops the program that's currently running. This is useful when you've written an infinite loop (a program that doesn't end)
F5	Acts like the Run program key. Press F5 to start the program running.

## 29.9 OUTPUT SCREEN



The output screen is where the results of the PRINT, CONSOLE and DUMP commands are written. It is the screen that's cleared or whose color changes when CLS or PAPER is run. You can only write to the screen; you can't read back from it.

The output screen contains both the fixed-character size screen (which is the normal output screen) and the scrolling console output. The console output is mostly intended for debugging.

The controls at the top of the screen let you modify the screen's looks.

Control	How and when to use it
<b>12x40 (26)</b> Screen Size	The output screen always tells you how large the screen is and the font size. Screen sizes include 12x20, 12x40, 16x60 and 24x80
<b>A</b> Smaller font	Reduces the font size, automatically making the output screen smaller.
<b>A</b>	Increases the font size, automatically making the output screen larger

Larger font	
	Toggles the console portion of the output screen on and off. Unlike the output screen, the console is a long scrolling list of output.
	Reduces the number of characters on the screen.
	Increases the number of characters on the screen.
 Close	Closes the screen. The screen contents are not lost; when the screen is re-shown, the original contents will still be present. When you PRINT to the screen, it will be displayed automatically.

## 30 BASIC LANGUAGE REFERENCE

---

### 30.1 PROGRAM STRUCTURE

A BC BASIC program is a list of *statements* and *functions*. Each statement can optionally start with a *line number*; line numbers are simple integers. Line numbers do not have to be in any special order (this is unlike many version of BASIC where the lines must be in numerical order).

#### Examples of statements

```
FOR I = 1 TO 4
PRINT "Hello World"
NEXT I
100 REM this statement has a line number
```

Statements are normally exactly one line. They can be extended with visible “return” type characters:

↓	U+21B2	DOWNWARDS ARROW WITH TIP LEFTWARDS
↙	U+21B5	DOWNWARDS ARROW WITH CORNER LEFTWARDS
↴	U+2936	ARROW POINTING DOWNWARDS THEN CURVING LEFTWARDS

Statements do not have a statement terminator (for example, the “C” language terminates statements with semicolons). Statements start with a command name like PRINT except that the CALL and LET command names are optional.

Examples of statements include **CLS BLUE** and **LET a=10**. The CALL and LET commands are optional; **LET a=10** is the same as plain **a=10**, and **CALL PrintName()** is the same as **PrintName()**.

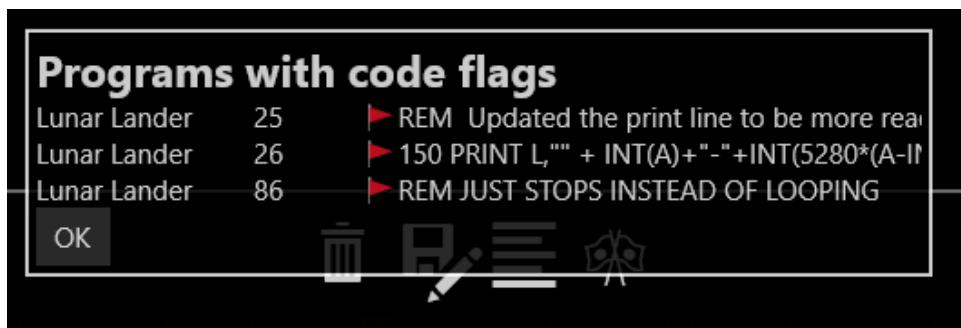
Lines are technically a sequence of characters that ends with a \n, \r or \v. The \v (vertical tab) is sometimes used by Microsoft Word when cut-and-pasting text.

## 30.2 FLAGS

BC BASIC lets you include flags (like ▶ and ☀) in your source code. Adding flags let you mark different lines of your program, either for you to find again later or to let other people review your code. The BC BASIC editor includes a **Flags** button that lets you easily insert a variety of flags into your code.

Comments start with the REM command; the rest of the line is the comment.

The properties page for each package includes a ☀ button; press it and each line of each program in the package that includes a flag character will be displayed.



## 30.3 NUMBERS AND STRINGS AND VARIABLES

Numbers and strings are the two most common things people want to manipulate. Inside a BC BASIC program, you can write number constants (like, 4 or 3.14) and string constants (like, “apple”).

### 30.3.1 Numbers

Number can be any of

- Integers like 3 or -5
- Integers using hexadecimal (base 16) notation like 0x65
- Floats (which are stored as double-precision) like 1.2, or -7.8
- Numbers in exponential notation like 45E3 (which is equal to 45000)

Doubles which are smaller than 1 can start with just a decimal place (0.3 and .3 are both OK). To help improve the look of your programs, BC BASIC allows for either a computer-style minus or a wider Unicode minus and dash characters.

Example: computer-style is - and wider Unicode is – or –. Microsoft’s Word program sometimes converts one into the other.

### Example of using different types of minus signs:

```
REM Constant numbers
```

```
V1 = -12.34E-6
```

```
V2 = -12.34E-6
```

```
V3 = -12.34E-6
```

```
REM Expressions
```

```
E1 = V1 - V2
```

```
E2 = V1 - V2
```

```
E3 = V1 - V2
```

```
REM Negated Values
```

```
N1 = - E1
```

```
N2 = - E2
```

```
N3 = - E3
```

Technically, these are Unicode characters HYPHEN MINUS (U+2D), EN DASH (U+2013) and MINUS SIGN (U+2013). The special Unicode plus signs are not accepted or the other Unicode minus signs.

#### 30.3.2 Strings

String constants can use either computer-style quotes like "" or “smart” quotes. A string that starts with an opening smart must be ended with a smart end quote. You can’t nest smart quotes: the string “hello “special” world” is incorrect. The string will be ended at the end of the word special; the rest of what you think is the string will be misinterpreted.

BC BASIC includes functions for manipulating strings. You can concatenate strings together with the + operator, get the length of a string with the LEN function, and extract parts of a string with the LEFT, MID and RIGHT functions.

#### 30.3.3 Variables, GLOBAL, and DIM

BC BASIC allows you to create and use variables at any time. Variables are given names; names must start with a letter (a to z and A to Z) and from then on can contain letters, numbers, and underscores. Variables can optionally end with a dollar sign (\$). Variables are case sensitive; the variable “name” is different from the variable “NAME”.

In BC BASIC the “\$” at the end of a variable has no special meaning. The \$ is allowed for compatibility with other versions of BASIC. In some versions of BASIC, a “\$” indicates that a variable is a string. In BC BASIC, any variable can be any kind of variable.

Variables created inside of functions are *scoped* to the function; they cannot be used out of the function.

Use the DIM statement to make array variables. Array variables let you get and set a lot of values in one variable. They are often used in mathematical analysis.

Use the GLOBAL statement in a function to use variable declared outside of the function. Normally you can use the same variable name both globally and in a function and they will refer to different variables. Sometimes in a function you need to refer to a global variable (for example, this is common in callback functions). In these cases, use **GLOBAL variable**. After that, references to the variable will be to the global variable.

### Example of using variables:

```
LET a = 10
LET b = a + 3
LET c = 7.89 / b
LET bignum = -1.3E23
LET smallnum = 2.78E-23

LET name = "Person of interest"
LET information$ = "You can use smart quotes"

LET dog_name = "Sumi"
LET check9 = 99

CLS GREEN
PRINT "Sample variables"
DUMP
```

Each variable includes some properties that tell you about the variable.

**variable.Type** says what the type of the variable is. Possible results are NUMBER, STRING, ARRAY, OBJECT, ERROR, and NOTHING. Other objects (like the different Bluetooth objects) may return other values.

`variable.IsNumber`, `variable.IsString`, `variable.IsObject` and `variable .IsError` are true if the variable is a number, string, object or error respectively.

`variable.IsNotNull` is true if the variable is number which is a NaN (not a number) value. If the number is a string or something that isn't a number, will return true.

## 30.4 <EXPRESSION> OVERVIEW

### 30.4.1 Quick introduction to expressions

"1 + 1" is one of the simplest expressions; it adds two *constants* (the two 1 values). Best Calculator BASIC has the normal set of operators and precedence rules for modern computer languages, plus a few extra convenience functions.

### 30.4.2 Expression Rules

BC BASIC is designed to make most expressions work normally.  $1+2*3$ , for example, will multiple the  $2*3$  first, and then add the 1. You can put parenthesis around your expressions. You can use either curved parenthesis or square brackets.

Expression type	Explanation and Sample
Variable	When evaluated, is the value of the variable. Examples: A B\$
Constant	A numeric or string constant Examples: 1 3.14 0xFF "Hello World" "She said, &QUOT;Hello&QUOT;"
Named values PI and RND	PI is always equal to PI (3.1415...). It's more common in BC BASIC to use the Math.PI value and not PI by itself. RND is a new random number between 0 and 1.
( expression ) [ expression ]	You can place expression inside of parenthesis () or square braces [] to show which operations should happen first.

<i>expression OPERATOR expression</i>	Any of the standard math operators like + and -. See the table below for a full list. BC BASIC also includes a variety of comparison and logical operators, and the INPUT operator.
PREFIX <i>expression expression POSTFIX</i>	Use minus (-) to negate a number. Use power and root operators to take a square root, or raise a value to a power.
Function ( argument, argument )	The value of the given function. There are a set of built-in functions (SGN ABS SIN COS TAN ASN ACS ATN LN EXP SQR INT LEN CODE CHR\$), or you can define your own. The parentheses are normally required. Unlike some other versions of BASIC, you cannot omit the parenthesis for the built-in functions.

Constants are always handled as doubles (1.2, or 0.1, or -56.7, or just plain 4 or -2

Example of expressions are

Example	Explanation
1	A constant
apple	The value of the variable "apple". The variable should have already been defined; otherwise it's assumed to be a "NaN" (Not a Number)
SIN (PI / 4)	The SIN of $\frac{1}{4}$ PI radians. The trigonometry functions all take their values in radians. Use the Math.DtoR() function to convert degrees to radians.
1 + 2 * 3	Is the value 7 (and not 9); multiplication is higher priority than addition so the $2*3$ is done first and then the 1 is added to it.
(1+2) * 3	Is the value 9; the parenthesis force the addition to be done first
"A" < "B"	Is the value 1 (for 'TRUE') because A is sorted before B.

### 30.4.3 negate, power, root prefix and postfix operators

Operator	Explanation and Sample
----------	------------------------

$2^3 4$	Square, cube and fourth power operators. BC BASIC is special among most programming languages for allowing these superscripts to be used. You can also use the Math.Pow() function or the $^{**}$ operator  Example: $5^2$ is 25 because $5*5$ is 25 Math.Pow(5, 2) is also 25 $5^{**}2$ is also 25
$\sqrt[3]{\sqrt[4]{}}$	Square root, cube root and fourth root operators. You can also use the Math.Pow () function or the $^{**}$ operator  Example: $\sqrt{64}$ is 8 because $8*8$ is 64 Math.Pow(64, 1/2) is also 8 $64^{**}0.5$ is also 8
-	Unary minus converts any number to its negative  Example: $12 + -3$ is 9 because $12 - 3$ is 9

### 30.4.4 Operators + - \* / and more

Each operator has a precedence value; operators with a higher precedence will be done before operators with a lower precedence.

Operator	Precedence	Explanation and Sample
$^{**}$	10	$^{**}$ is the “raise to the power” operator  Example: $2^{**} 6$ is 64 because $2*2*2*2*2*2$ is 64  Historical note: early versions of BC Basic includes a “root finding” operator. This proved to be confusing in practice, and has been removed.
$* /$	9	$*$ is the computer sign for multiply $/$ is the standard computer sign for divide Examples:

		3 * 4 is 12 10 / 2 is 5
+ -	6	<p>+ is the standard computer sign for addition or string concatenation - is the standard computer sign for subtraction</p> <p>Example: 1+2 is 3 10 - 2 is 8</p> <p>“1” + 2 is “12”, but 1 + “2” is 12. When the left side of a + is a string, both sides are treated as string and concatenated together; when the left side is a number, both sides are treated as numbers. Strings that cannot be converted are handled as a NaN</p> <p>Extra bonus: The Unicode character set designates three characters that are commonly used for minus signs:</p> <ul style="list-style-type: none"> <li>HYPHEN-MINUS (U+2D), the normal minus sign</li> <li>MINUS SIGN (U+2212)</li> <li>EN DASH (U+2013)</li> </ul> <p>Any of these can be used for a minus sign. Among other things, this makes it easier to copy your programs back and forth between BC BASIC and Microsoft Word.</p>
< <= = >= > <> ≈	5	<p>The normal set of operators for less than, less-than-or-equal, and so on. The &lt;&gt; operator is for “not equals”.</p> <p>The ≈ is for “approximately equals”; for numbers it means that the two numbers are within 5 significant figures of each others, and for strings that they compare equal using the CurrentCultureIgnoreCase flag.</p>

		Note that two numbers which are different signs are never considered approximately equal even if they are both really, really close to zero.
NOT	4	<p>Inverts the logical value of its argument.</p> <p>Example</p> <p>IF NOT A=B THEN &lt;statement&gt; will do the statement if A is NOT equal to B.</p> <p><b>Note on logical values:</b></p> <p>The logical operators (NOT, AND, and OR) can take any numerical value and treat it as a logical value; anything that's zero is treated as FALSE and all other numeric values are TRUE. The operators will only ever produce a 0 or 1.</p>
AND	3	A logical AND operation; A AND B will be 1 if both A and B are TRUE (non-zero).
OR	2	A logical OR operation. A OR B will be 1 if either A or B are TRUE (or if both of them are).
INPUT		<p>INPUT is a complex operator with two optional values, a PROMPT and a DEFAULT value. When your program runs, a dialog box will pop up with the prompt you specific and with a starting value of whatever the default value was (it can be an expression). The user then enters a value and presses the OK key; the resulting value is the value of the INPUT expression.</p> <p>Example:</p> <p>birth_year = INPUT DEFAULT 1967 PROMPT "Enter your birth year: "</p>

### Examples of expressions:

REM Multiplication binds more than addition  
 REM a will be 7 ( $1 + (2 * 3)$ ) and not 9 ( $(1+2) * 3$ )  
 $a = 1 + 2 * 3$

```
b = 10 - 4 / 2
```

REM Demonstrate cube root and raise to 4<sup>th</sup> power

```
c = 3 ∛ 10
```

```
d = 6 ** 4
```

REM Comparing values. PI is not about 22/7

REM but it is about 355/113

```
e = PI ≈ 22/7
```

```
f = PI ≈ 355/113
```

REM These are all false (except j)

```
g = PI > 22.7
```

```
h = PI >= 22.7
```

```
i = PI = 22.7
```

```
j = PI <> 22.7
```

REM Logical operators

```
k = c > 2 AND c < 4
```

```
l = c > 2 OR d < 10
```

```
m = NOT (c > 2 OR d < 10)
```

REM You can ask for input from the user

```
n = INPUT DEFAULT 5 PROMPT ↴
```

"Please enter a number"

```
CLS BLUE
```

```
PRINT "All the variables"
```

```
DUMP
```

**Example of using the RND and PAUSE statement to make a little random animated display:**

```
CLS BLUE  
COUNT = 0
```

REM You can also use a FOR..NEXT loop

```
10 A = DrawDot()
```

```
COUNT = COUNT + 1
```

```
IF (COUNT > 100) THEN GOTO 20
```

```
PAUSE 1
```

```
GOTO 10
```

```
20 PRINT AT 1,1 "DONE"
```

```
FUNCTION DrawDot()
col = INT (RND * Screen.W) + 1
row = INT (RND * Screen.H) + 1
ch = "*"
PRINT AT row,col ch
RETURN
```

### 30.4.5 The INPUT expression

The INPUT expression lets you prompt your user to enter a value. You can use DEFAULT <value> to supply a default value and a PROMPT <string> to specify a prompt string. You must specify the DEFAULT before the PROMPT.

If the DEFAULT value is a string, the user will be allowed to enter either a number or a string. Otherwise, the user may only enter a number. The original version of BCBASIC only allowed for numeric entry.

There is also an INPUT statement that is less powerful than the INPUT expression. It's provided for compatibility with other versions of BASIC.

#### Example of the INPUT expression:

```
LET interest = INPUT DEFAULT 3.5 ↴
PROMPT "Interest rate"
```

The user will enter a value which will be interpreted as a number.

## 30.5 MATH FUNCTIONS

BC BASIC includes a small but powerful set of math functions. These are designed to be compatible with other versions of BASIC.

Some version of BASIC let you use these functions without parenthesis; BC BASIC does not.

There are even more functions available in the Math extension.

### 30.5.1 Trigonometry functions SIN COS TAN ASN ACS ATN

Function	Notes
ACS(value)	Calculates the inverse of COS; given a value will compute the corresponding angle in radians.

ASN(value)	Calculates the inverse of SIN; given a value will compute the corresponding angle in radians.
ATN(value)	Calculates the inverse of TAN; given a value will compute the corresponding angle in radians. Note that TAN of 90° is infinite.
COS (radians)	Calculates the cosine of an angle given in radians
SIN (radians)	Calculates the sin of an angle given in radians
TAN (radians)	Calculates the tangent of an angle given radians

### 30.5.2 Logarithm and Power functions LN EXP SQR

Function	Notes
EXP (value)	Calculates the value $e^{\text{value}}$ for any given value. This is the inverse of the LN function This is the same as Math.Exp(value)
LN (value)	Calculates the natural (base e) logarithm of the given value. This is the inverse of LN. This is the same as Math.Log(value)
SQR (value)	Calculates the square root of a value. You can also use the √ square root operator, or use the ** power operator. This is the same as Math.Sqrt(value).

### 30.5.3 Rounding and sign functions SGN ABS INT

Function	Notes
ABS (value)	Calculate the absolute value of a number. This is the same as Math.Abs(value)
INT (value)	Calculates the floor of a number. The floor is the number rounded down to the nearest integer. For example, INT (12.8) is 12 and INT (-12.8) is -13. This is the same as Math.Floor (value)
SGN (value)	Return the sign of a number. The sign is 1 for positive values, -1 for negative values, and 0 for zero. This is the same as Math.Sign(value)

## 30.6 STRING FUNCTIONS (LEFT, MID, RIGHT, LEN, CHR\$, CODE, VAL)

### 30.6.1 LEFT (string, count), MID (string, index, count) and RIGHT (string, count)

The LEFT, MID and RIGHT functions get data from the start (LEFT), middle (MID) or end (RIGHT) ends of a string. The return value is always a string.

For the MID function, you don't need to provide the count value; it will be

assumed to be 1. The index for the MID function starts at 1; **MID (string, 1,**

**count**

is exactly the same as **LEFT (string, count)**.

#### Examples of the string functions:

```
REM LEFT(string, count) returns string 'count' long
REM from the left part of the input string
REM The example will print AB
PRINT "LEFT("ABCDE", 2) = "; LEFT("ABCDE", 2)
```

```
REM MID(string, index, count) returns
REM      a string 'count' long
REM from the middle part of the input string
REM starting at 'index'. The first letter is index 1.
REM The example will print BCD
PRINT "MID("ABCDE", 2, 3) = "; MID("ABCDE", 2, 3)
```

```
REM RIGHT(string, count) returns
REM      a string 'count' long
REM from the right part of the input string
REM The example will print DE
PRINT "RIGHT("ABCDE", 2) = "; RIGHT("ABCDE", 2)
```

Each of the functions has similar special cases.

1. The count is always truncated so that the return value doesn't go past the size of the string
2. If the count value is less than one, then a blank (zero length) string is returned

3. If the index for MID is more than the length of the string, then a blank string is returned

### 30.6.2 LEN string

The LEN function (which has optional parenthesis) returns the length of a string. Blank strings have a length of zero.

#### Examples of the LEN function:

```
REM The length is always 3
PRINT "LEN of string ABC is 3"
PRINT LEN "ABC"
PRINT LEN ("ABC")

REM Length of a blank string is zero
PRINT LEN ""

REM LEN PI is the length of the string
REM      representing the number PI
REM (the answer is 16 printed digits)
PRINT LEN Math.PI
```

Special cases include:

1. When given a number, returns the length of the string that represents the number
2. Blank strings have zero length
3. Unicode strings that use *surrogate pairs* will count each surrogate pair as two characters. For example, the Unicode U+1F60B character point (FACE SAVORING DELICIOUS FOOD) is presented as two characters, U+D83D and U+DE0B. These two characters are called a surrogate pair, and represent the Unicode character.

### 30.6.3 CHR, CODE

CHR converts a series of numerical Unicode values into a string. There is also, for compatibility, a CHR\$() that takes a single argument. The CHR\$ function is technically an expression operator and does not require parenthesis. That is, CHR\$ (65) can be written as CHR\$ 65.

The CODE function returns the Unicode code point of the first character in a string. The CODE function is technically an expression operator and does not require parenthesis. That is, CODE ("A") can be written as CODE "A".

In Windows, Unicode code points outside the Basic Multilingual Plane are encoded as surrogate pairs and are handled as two characters.

#### Example of the CHR, CHR\$ and CODE functions:

```
REM CHR converts a Unicode character number
REM      to a string
REM 65 is the ASCII A
REM Unicode U+1F60B is
REM      "FACE SAVOURING DELICIOUS FOOD".
REM It is converted into two chars
REM      (a surrogate pair).
REM CHR$ is the same function, but takes in
REM      only one parameter
PRINT "CHR (65) = "; CHR (65)
PRINT "CHR Unicode: ";
      CHR(65, 66, 0x1F60B, 67, 68)

REM CODE converts the first character of
REM      a string to a code
PRINT "CODE "ABC" = "; CODE "ABC"
```

#### 30.6.4 VAL(string)

The VAL function evaluates the string as a BC BASIC expression and return the value. For example, VAL("1 + 2") will return 3. The expression can use variables that you have set.

Note that VAL can be slow.

#### Example of the VAL function:

```
REM VAL will evaluate an expression
a = 1
b = 2
PRINT "VAL ("a + b") = "; VAL ("a + b")
```

## 30.7 <STATEMENT> OVERVIEW

Statements are the building blocks of a BC BASIC program. They let you perform calculations, assign variables, loop until a condition is true, define functions and more.

### Examples of statements:

```
10 CLS
A = 3
B = 4
20 C = A + B
PRINT "C is "; C
```

BC BASIC does not allow for multiple statements on a single line.

Statements can include an optional line number. They are the targets for GOTO and GOSUB statements. They do not need to be in any particular order. Other versions of BASIC always require line numbers and automatically order all lines by line number. BC BASIC instead lets you use any line numbers you want in any order.

### Good line number practices:

1. Only number a statement when you have to.
2. Keep your line numbers in numerical order
3. Make your line numbers divisible by 10 or 100; that way you can add new line numbers in between existing values.

Line numbers in a function are *scoped* to that function; two functions can use each other's line number. You cannot GOTO or GOSUB into or out of a function.

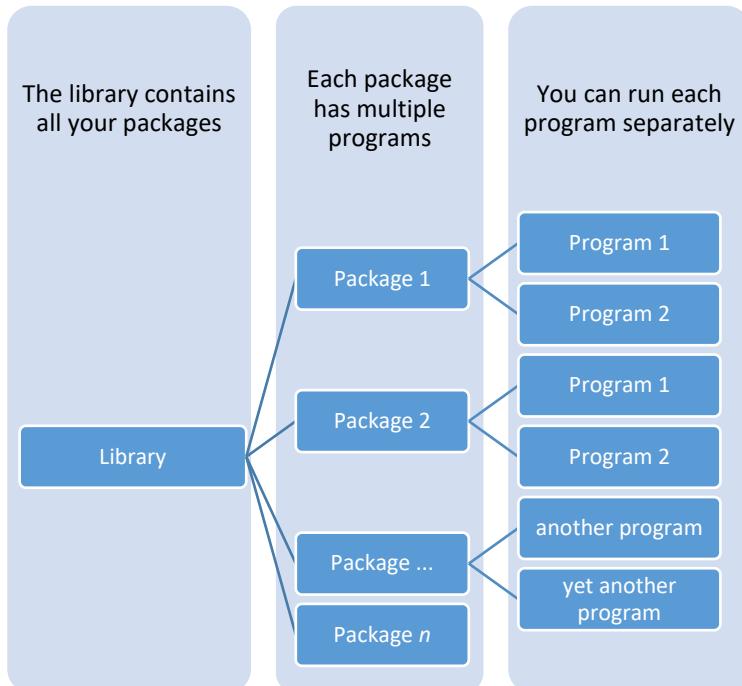
Statements are separated by new-lines; a newline is one or more of carriage-return, new-line, or vertical-tab. (Pressing ALT-newline in Word can separate lines with vertical tabs). You can continue a statement from one line to another by placing a visible enter symbol ( $\downarrow$ , U+21B2) at the end of the line where white space would fit.

## 30.8 PACKAGES AND PROGRAMS

The BC BASIC *Package* and *Program* concepts are special to the BC BASIC environment.

If you're just getting started, you should feel free to just place the programs you write into the single package that you made when you wrote your first program. You don't even need to rename it; you can continue to use the New Package name.

But if you are going to write more than a few programs, you should spend a few minutes understanding the BC BASIC *Package* and *Program* concepts. These are explained more fully later on.



The BC BASIC environment tracks your programs for you. You do not have to deal with their file names or try to remember where the packages are or what programs they have. When you make a new package, the BC BASIC environment makes a file for you in the app roaming directory, and picks the name for you with a file extension of *.bcbasic*. Best Calculator will automatically read in all of the packages you've created (and they roam, too, so you can make

or edit a package on one computer and it will be automatically sent to your other computers).

### 30.8.1 Inside a package file

There is one file per package (and therefore multiple programs are placed into a single file). When Best Calculator starts, it reads in all the *.bcbasic* files both in the BC BASIC directory (these are the samples shipped with Best Calculator) and all the *.bcbasic* files in the app data directory (these are your packages).

## 30.9 PICKING A PACKAGE FOR YOUR PROGRAM

When you make a new program, you should add it to a package where it will logically fit. For your first few package, that will probably be the “New Package” that you created. You will often find that you make a number of programs that you will be using together. For example, you’ve made one program that converts square feet to acres. The reverse program (acres to square feet) would naturally fit into the same package.

### 30.9.1 Creating common functions for several programs

The programs in a package are normally independent of each other. They don’t share variables or functions, and you can GOTO or GOSUB from one to the other. There are exceptions to this general rule. The IMPORT statement can read in the functions of another program in the same package. This lets you make a “library” of functions that many programs can use.

### 30.9.2 Exporting packages

You can export (write) packages out to a file and then later import (read) the files back in. When you export, you create a JSON file that lists all the programs along with their names and descriptions and BC BASIC code. The export and import mechanisms mean that you can share code with your colleagues, coworkers, fellow students, or friends. You might even find useful packages on the internet. Be careful though: although a BC BASIC program cannot damage your computer, it’s possible for one package to delete your memory variables and conceptually possible to use up excessive disk space.

## 30.10 ALL OF THE SPECIAL SYMBOLS

Best Calculator supports a number of special symbols both to help create good-looking program and so that your programs can be copy and pasted into word processors like Microsoft Word.

Symbol	Name	How it's used
$\sqrt{ }$	Square Root U+221A	$C = \sqrt{A^2 + B^2}$
$\sqrt[3]{ }$	Cube Root U+221B	$X = \sqrt[3]{Y}$
$\sqrt[4]{ }$	Fourth Root U+221C	$W = \sqrt[4]{X+Y}$
<sup>2</sup>	Superscript 2 U+B2	$C=X^2$
<sup>3</sup>	Superscript 3 U+B3	$C=X^3$
<sup>4</sup>	Superscript 4 U+2074	$C=X^4$
-	Hypen-Minus U+2D	The minus sign on a keyboard
–	Minus Sign U+2212	Alternate minus sign
–	En dash U+2013	Alternate minus sign
$\approx$	Approximately equal to U+2245	Helpful to compare floating point numbers.
“ ”	Left and right double quotation marks U+201C and U+201D	Smart Quotes
\v	Line Tabulation U+B	Can be used just like a normal carriage return (or Enter, Return or Line feed). Word sometimes converts those into a line tabulation
↓	Downwards arrow with tip leftwards U+21B2	Line continuation: use at the end of a line to continue onto the next line.
↳	Downwards arrow with corner leftwards U+21B5	Line continuation
↙	Arrow pointing downwards then curving leftwards U+2936	Line continuation

## 31 BASIC STATEMENTS REFERENCE

---

In the descriptions of statements

- Words in <angle brackets> describe what to add.
- Words in square brackets are optional
- The punctuation ,... means the previous item can be part of a list separated by comma. The list can include no items at all.
- The punctuation ... means a series of statements on new lines
- Otherwise, the items are to be entered as they appear

For example, the CALL statement is described as **[CALL] <function> (<expression>, ...)**.

The word CALL is optional; you can include it or not as you please.

The <function> means you have to enter the name of a function

( ) are parenthesis; they are required

<expression> is an expression (they are explained in an earlier section)

The following are valid statements

```
CALL myfunction ()  
CALL myfunction (1)  
CALL myfunction (1+2)  
CALL myfunction (1, 2)  
myfunction ()
```

### 31.1 [CALL] <FUNCTION> (<EXPRESSION>, ...)

Calls the function by name, passing in the given parameters. The word CALL is optional, but the parentheses are required.

**Example of using CALL and defining a FUNCTION:**

```
CALL Hello("Mom")  
Hello("Dad")  
  
FUNCTION Hello (name)  
PRINT "Hello ",name  
RETURN
```

The example will print Hello Mom and Hello Dad.

### 31.2 CLS [<COLOR>] AND PAPER <COLOR>

CLS will clear the scrolling console and screen and potentially change the screen color. PAPER will change the screen color without clearing the screen.

Normally after you press 'Run' the scrolling console will have the results of previous runs and will also have the results of evaluating your program. You can change the background color of the screen by specifying either a color name or a color index from the table below.

Pick the color for CLS and PAPER with either a color number (0 to 7) or a color name.

Color Number	Color Name
0	BLACK
1	BLUE
2	RED
3	MAGENTA
4	GREEN
5	CYAN
6	YELLOW
7	WHITE

**Example of using CLS to clear the screen:**

```
CLS BLUE
```

**This example is like a color stroboscope:**

```
REM The screen only shows up if you PRINT
REM      something to it.
REM PAUSE delay is in "frames"; there are
REM      50 frames per second.
```

```
PRINT ""
delay = 25
FOR i=1 TO 3
```

```
FOR color = 0 TO 7  
CLS color  
PAUSE delay  
NEXT color
```

```
REM You can specify colors with names  
CLS BLACK  
PAUSE delay  
CLS BLUE  
PAUSE delay  
CLS RED  
PAUSE delay  
CLS MAGENTA  
PAUSE delay  
CLS GREEN  
PAUSE delay  
CLS CYAN  
PAUSE delay  
CLS YELLOW  
PAUSE delay  
CLS WHITE  
PAUSE delay  
NEXT i  
CLS BLACK
```

### 31.3 CONSOLE <EXPRESSION> [, <EXPRESSION>]

Writes the expressions out to the scrolling console. Any number of expressions can be given (including none)

**Example of writing to the console:**

```
CLS BLUE  
ANGLE = 45  
RADIAN = Math.DtoR (ANGLE)
```

```
REM PRINT to the screen to see the console.  
PRINT ""  
CONSOLE "SIN(45 degrees)", SIN(RADIAN)
```

The CONSOLE command prints out about 0.707. You might need to tap the



console key ( ) to see the console. See '[Graphics and Best Calculator BASIC](#)' for a description of the console screen.

### 31.4 DIM <NAME> ([SIZE])

Creates an array variable. Use array variables when you need to store a set of values and then index them. You will often use arrays when you do data analysis.

Example

```
DIM a()  
a(1) = "First"  
a(2) = "2.2"  
PRINT a(1)  
PRINT a(2)
```

You can find out the length of an array using the Count value.

```
FOR i = 1 to a.Count  
    PRINT a(i)  
NEXT i
```

You can also specify the size of the array in the DIM statement

```
REM Make an array that's exactly 10 items long  
DIM a(10)  
a(8) = 8.8
```

```
REM Nope! You can't add an element beyond 10.  
a(20) = 20.20
```

Here's a real-world example showing how calculate the average of a list of number. The DIM statement isn't given a specific value, so the array can hold any number of elements.

```
CLS BLUE
DIM a()
REM Use DIM a(3) if you know the array will always
REM be 3 elements long. In that case, all three
REM elements will be initialized to NaN.
a(1) = 10
a(2) = 20
a(3) = 90

average = Average(a)
PRINT "Average is ";average
REM The average of 10, 20, 90 is 40

FUNCTION Average (data)
    sum = 0
    FOR i=1 TO data.Count
        sum = sum + data(i)
    NEXT i
    RETURN sum/data.Count
```

## 31.5 DUMP

Prints all the variables to the scrolling console. This is a common mechanism to see what your program is doing. DUMP will also print out all of the memory values.

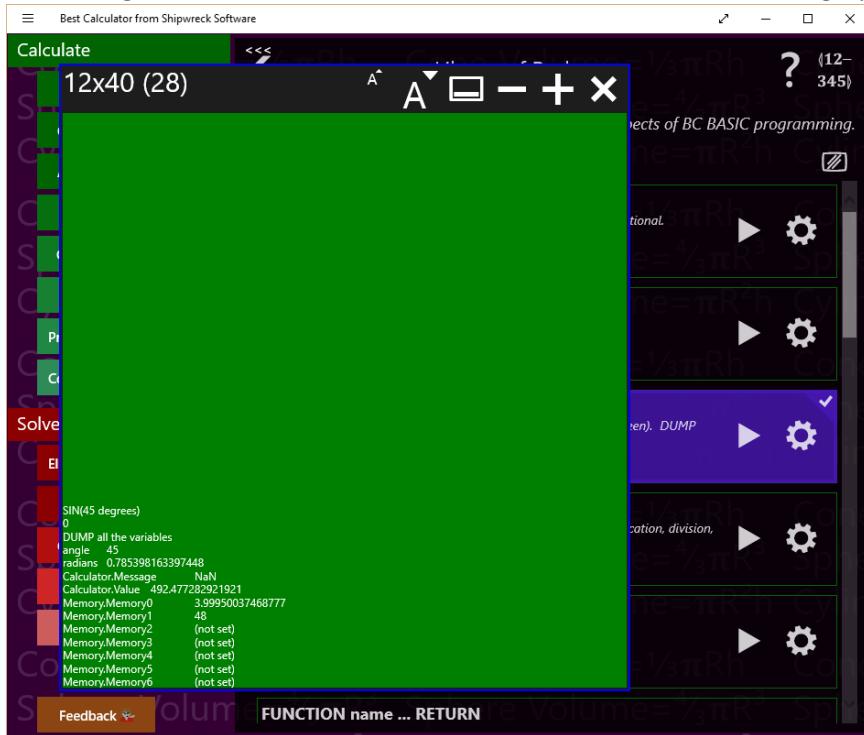
### Example of using DUMP:

```
CLS BLUE
ANGLE = 45
RADIAN = Math.DtoR (ANGLE)

REM PRINT to the screen to see the console.
PRINT ""
CONSOLE "SIN(45 degrees)", SIN(RADIAN)

CONSOLE "DUMP all the variables"
DUMP
```

The variables are then printed to the scrolling console.



You might need to press the console key (  ) to see the console.

### 31.6 FOR <VARIABLE> = <START> TO <END> [STEP <STEP>] ... NEXT <VARIABLE>

Use the FOR and NEXT statements to form loops. The variable is the name of the *loop variable*; it will start at the <start> value. Each time through the loop, it will be incremented by the <step> amount (the default is 1) until it's more than the <end> amount. The start, end and step values are all expressions.

If the <step> value is negative, then the loop is changed slightly. The variable starts at the start value, is decremented by the <step> value, and the loop ends when the variable is less than, not greater than the <end> value.

The end of the loop is the NEXT <variable> statement; the variable is the exact same as in the FOR statement. For example `FOR i=1 TO 10` is ended at a later `NEXT I` statement.

## Common errors:

1. Never GOTO or GOSUB into the middle of a FOR ... NEXT loop, and never jump out.
2. FOR ... NEXT loops can be nested inside each other, but always nest them correctly. The first FOR must match the last NEXT
3. You can reuse a variable name in several loops, but not nested.
4. You should not modify the variable inside the loop.
5. You will always go through the loop at least once

**Example of a simple FOR loop:**

```
PRINT "Value", "Squared"
FOR I=1 TO 10
PRINT I, I**2
NEXT I
```

The example prints a table of numbers and their squares.

Value	Squared
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

**Example of going backwards through a loop:**

```
PRINT "Value", "Squared"
FOR I=10 TO 1 STEP -1
PRINT I, I**2
NEXT I
```

To go backwards, you have to specify a negative STEP value and TO value that's less than the starting value. In the example, the STEP value is -1, and the TO value (1) is less than the start value (10).

**Example of a nested FOR loop:**

```
CLS GREEN
PRINT "Value **2    **3    **4    **5"
FOR N=1 TO 10
PRINT AT N+2,1 N
FOR POWER = 2 TO 5
PRINT AT N+2, (POWER-1)*8 N**POWER
NEXT POWER
NEXT N
```

This function prints a table of numbers; each number is printed along with the number raised to a power of 2, 3, 4 and 5.

Value	**2	**3	**4	**5
1	1	1	1	1
2	4	8	16	32
3	9	27	81	243
4	16	64	256	1024
5	25	125	625	3125
6	36	216	1296	7776
7	49	343	2401	16807
8	64	512	4096	32768
9	81	729	6561	59049
10	100	1000	10000	100000

### 31.7 FUNCTION <NAME> ( <ARGS> ) ... RETURN [<VALUE>]

Creates a function with the given name (expressions are not allowed), taking the arguments. A single value can be returned.

When you CALL a function, you pass in data; that data is then given the names in the function. The names in the argument list are only valid in the function.

It's considered a best practice to have a single RETURN in a function; it's easier to understand a function that has only a single return point. However, it's also sometimes easier to RETURN early.

**Example of using FUNCTION to print to the screen:**

```
CALL Hello("Mom")
Hello("Dad")

FUNCTION Hello (name)
PRINT "Hello ",name
RETURN
```

In the example, the function HELLO will print whatever value is passed in. It's called twice, and therefore the program will print out two messages: Hello Mom and Hello Dad.

### 31.8 GLOBAL <VARIABLE>

The GLOBAL statement lets you to use global variables from within a function without having to pass the variable in as a parameter. For example, the callback functions used when a Bluetooth device has a changed characteristic value have no way to pass arbitrary data in and might need access to variables declared at the global level.

To use a global variable, use the GLOBAL <variable name> command inside your function.

Example

```
LET message = "This is a variable at the global scope"
CALL PrintMessage ()
STOP
```

```
FUNCTION PrintMessage()
    GLOBAL message
    PRINT message
END
```

## 31.9 GOSUB <LINENUMBER> AND RETURN

*Tip: you should almost always use a function instead of using GOSUB. Many earlier versions of BASIC (including Sinclair BASIC and MSW BASIC) used GOSUB instead of FUNCTIONS.*

GOTO and GOSUB both jump to the line number indicated. GOSUB remembers where you jumped from. When the program encounters a RETURN statement, the program will continue at the line after the GOSUB line. You can nest a GOSUB inside a GOSUB routine.

It's considered good practice that each block of code that you will GOSUB to has a single entry point and usually will have just the one RETURN. Otherwise, your code will get very complex.

### Example of using GOSUB:

```
REM Calculate the hypotenuse of a triangle
A = 3
B = 4
GOSUB 100
PRINT ""
DUMP
STOP

100 REM Calculate the hypotenuse from A and B
C=2 √(A**2 + B**2)
RETURN
```

## 31.10 GOTO <LINENUMBER>

Jumps to the line number indicated. Unlike GOSUB, you can't RETURN from a GOTO.

GOTO is often considered harmful.

In BC BASIC, the primary values of a GOTO is compatibility with earlier versions of BASIC and to help “extend” the statements in an IF statement. BC BASIC IF statements can only conditionally run a single statement after the THEN, and do not have ELSE clauses.

### 31.11 IF (<EXPRESSION>) THEN <STATEMENT> [ELSE <STATEMENT>]

If the expression is TRUE (not zero), then the statement will be run; otherwise it will not be. Often the statement will be a GOTO or GOSUB. The expression often uses the comparison operators (< > > plus AND OR and NOT).

**Example of an IF statement:**

```
a = 15  
IF a > 12 THEN PRINT "A is more than a dozen"
```

In this example, the variable ‘a’ is set to the value 15. The PRINT part of the IF statement will only be executed if the variable a is greater than 12. Since 15 is more than 12, the PRINT statement is executed, and “A is more than a dozen” is printed.

The statement after the ELSE is optional; it will be run if the expression is false.

```
IF (x >= 1) THEN PRINT "Single line: x >= 1" ↴  
ELSE PRINT "Else: x NOT >=1"
```

### 31.12 IF (<EXPRESSION>) ... [ELSE ...] ENDIF

The IF ... ENDIF statement is very similar to the IF statement but it lets you include multiple IF statements in a block instead of just one. It also lets you add statements in an ELSE clause. The statements in the ELSE clause will only be run if the expression is false.

```
IF (x < 1)  
    PRINT "Multi-line IF statement (expression is true)"  
    PRINT "x<1"  
ELSE  
    PRINT "Multi-line IF statement (expression is false)"
```

```
PRINT "NOT x<1"  
END IF
```

### 31.13 IMPORT FUNCTIONS FROM “PROGRAM”

The IMPORT statement will read in all of the functions from a specified program in the same package. This lets you make a package with a common set of functions that all the programs in the package can use. This is useful when several programs in one package all need to perform the same calculation.

The STATISTICS samples do this. There’s a program called “Sample Size Library” that consists of a set of useful statistical functions (MarginOfError, SampleSize, GetZ, and more). The programs that you might bind onto a key then just IMPORT the functions from that program and can call them.

Technical details: as needed, the library program is compiled and the functions remembered. When the IMPORT FUNCTIONS FROM “program” is run, the functions are imported by name. All functions are imported automatically. You can’t pick just one or two functions from a program.

You always IMPORT a program from the same package. You cannot IMPORT FUNCTIONS from a different package. You can IMPORT as many programs as you like; newer imported functions will override older ones. Deliberately doing this is not a best practice.

#### **Example of the IMPORT statement:**

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

The example is taken directly from the AU to Meters program in the Astronomy library. As soon as it’s run, the program can call any function from the “Conversion Library” program.

### 31.14 INPUT <VARIABLE>

*Note: the expression <variable> = INPUT DEFAULT <value> PROMPT <prompt> is a more flexible and powerful way to read in data. The INPUT statement is included to improve compatibility with other versions of BASIC.*

The INPUT statement asks the user to enter a value. If the variable name ends with a \$ (dollar sign) the user may enter non-numeric values and the value will be a string value. Otherwise, the user can only enter a number.

#### Example of the INPUT statement:

```
REM The a=INPUT expression has more power
REM than INPUT statement. The expression version
REM lets you specify a prompt and a default.

REM The INPUT statement has no default value
REM and no prompt.
INPUT taxrate

REM The INPUT expression has box a default
REM and a prompt. The user has an easier time
REM remembering what to enter.
income = INPUT DEFAULT 40000 ↓
    PROMPT "Enter your income for the year"

PRINT "Owe="; taxrate*income
IF (taxrate*income > 100) THEN ↓
    PRINT "You owe more than 100"
```

### 31.15 (LET) <VARIABLE> = <EXPRESSION>

LET is the *assignment* statement. It sets (assigns) the value of the expression to a variable. The variable might or might not already exist. If the variable didn't already exist, it will be created. If it did exist, the old value is discarded and overwritten with the new value. Some languages call this an *assignment* statement.

Variables start with a letter, and then can be any combination of letters, digits, and underscores. Variables are case sensitive; myage is different from MYAGE and is different from myAge. Expression can include requests for user input.

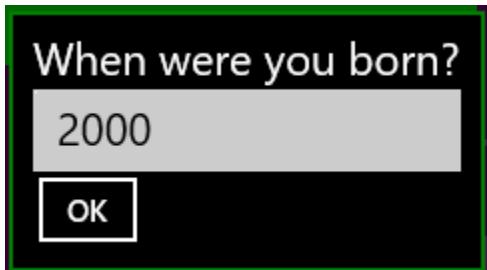
The word LET is optional in BC BASIC. You will find that it is required in many other variants of BASIC.

#### Examples of the LET statement:

```
LET year = 2015
birth_year = INPUT DEFAULT year - 15 ↓
```

```
PROMPT "When were you born?"  
age = year - birth_year  
PRINT ""  
DUMP
```

There are 3 LET statements in the example. The first uses the LET word (LET year = 2015). The other two leave off the LET word. When the INPUT expression is run, BC BASIC will pop up a dialog for the user to enter a value.



### 31.16 PAUSE <FRAMES>

Pauses the screen. This is useful when animating the screen. A value of 1 is one “frame”; there are about 60 frames per second. The value is not exact.

You can see an example of the PAUSE being used in the Colorful Countdown sample in the Quick Samples package.

### 31.17 PRINT [AT ROW,COL] <EXPRESSION> [ (, OR ; ) [AT ROW,COL]<EXPRESSION>]\*

The PRINT command will print one or more expressions to the output screen. BC BASIC remembers where the last thing was printed, and prints the next thing on the next line.

**PRINT expression** prints an expression (either a number or a string) onto the next line of the screen. This is the most common use of the PRINT command. If the screen is already full, nothing happens. The screen won’t clear or scroll to make room for the new text.

**PRINT expression; expression** to print two values next to each other

**PRINT expression, expression** to print values in columns

**PRINT AT [row, col] for exact placement.** You can print each expression at a particular point on the screen with the AT row, col syntax. The rows and column values start with 1,1 at the upper-left corner of the screen. You can tell how large the screen is with the Screen.H and Screen.W values.

**Simplest example of PRINT:**

```
PRINT "Hello, World"
```

**Example of PRINT with an AT:**

```
PRINT AT 1,1 "HELLO"; AT 2,4 "WORLD"
```

**Example of doing a PRINT with multiple expressions and commas:**

```
PRINT "HELLO", "WORLD"
```

Note that there's a big gap between the "HELLO" and the "WORLD". The comma means to print at the next tab stop; those are each 16 characters apart.

**Example of doing a PRINT with multiple expressions and semicolons:**

```
PRINT "HELLO"; "WORLD"
```

With semicolons the two words are printed right next to each other without even a single space between them.

### 31.18 RAND <SEED> & THE RND VALUE

The RND value (like PI) acts like a variable in expressions; unlike PI (which is always the same value), RND will provide a stream of different random numbers between the values 0.0 and 1.0.

**Example of using RND:**

```
REM Print some random numbers  
PRINT RND, RND, RND, RND
```

This prints numbers like so:

16x60 (26)

<sup>A</sup> A<sup>V</sup> □ - + ×

0.911517164163067  
0.542218610431169

0.852741063038232  
0.0266883256969453

### Technical details

BC BASIC provides access to a single pseudo-random number generator; the stream of values is determined entirely by the value of the initial seed value. All expressions in all functions are connected to the same stream of random numbers. The RAND statement will re-seed the random number generator. The seed value 0.0 is special; it will re-seed the generator with a time-dependant value. All other seed values simply reset the seed.

*Tip: the random number generator is not suitable for any cryptographic or security uses.*

### 31.19 REM COMMENT WORDS TO THE END OF THE LINE

The REM (remark) statement lets you put comments into your code. The words after the REM, and up to the carriage return, are entirely ignored by the program,

**Example of a REM comment statement:**

**REM Calculate the hypotenuse given A and B**

$$C=2 \sqrt{(A^{**2} + B^{**2})}$$

The REM statement will help you understand what your program does. The common practice is to explain the *why* of a program, not the *how*. It's also common to comment on unusual or clever mathematical techniques. For example, some of the programs in the Real Estate section specifically refer to the exact regulations that are being implemented; this helps later on when you need to verify that the program is implemented in accordance with the laws.

### 31.20 STOP AND END

The STOP statement halts execution of the program. If rerun, the program will start from the very beginning again. It's common practice for programs to have their main logic at the start, and a number of subroutines (called by using GOSUB) at the end. A STOP statement is placed after the main logic, and before the subroutines.

**Example of using STOP to return a value to the calculator:**

```
IMPORT FUNCTIONS FROM "Conversion Library"  
from = Calculator.Value  
m = ConvertToMeters(from, "au")  
Calculator.Message = "Convert " + from ↓  
    + " au into " + m + " meters"  
STOP m
```

When the example is done, it returns the “m” value to the calculator. The calculator then prints the value into the display. The example is from the Space and Astronomy built-in package. It will not just work if you type it into a new program. To work, it requires that you have a program “Conversion Library” in your package.

END is permitted as a synonym for STOP. It is added for compatibility with other version of BASIC including the first Dartmouth BASIC program (see the historical note).

## 32 EXTENSIONS REFERENCE

BC BASIC includes *extensions* to the core BASIC language. The extensions let you

- Set and get values from the calculator screen
- Perform advanced math
- Set and get memory values that persist from one session to another and roam to your other computers
- Get information about the screen

### 32.1 CALCULATOR.VALUE AND CALCULATOR.MESSAGE EXTENSION

You can get and set the current numeric calculator value with the Calculator.Value value.

You can set and get the Calculator.Value; it is always a double. If you try to set it to a string value, the string will be converted to a double (e.g., the string “3.14” becomes the double 3.14; the string “apple” becomes the double NaN).

The Calculator.Message is the small display above the calculator value. When you start Best Calculator, it will say “System test passed”.



You can set (but not get) the Calculator.Message value.

**Examples of using the Calculator.Value and Calculator.Message extensions:**

```
value = Calculator.Value
retval = value * Math.PI
Calculator.Message = "Converted diameter " +
    + value + " to circumference"
Calculator.Value = retval
```

First the variable called “value” is set to the current value in the calculator window. It’s multiplied by PI. Then a message is printed to the calculator window, and the calculator value is set to the variable called “retval”.

## 32.2 MEMORY EXTENSION

You can read and write numeric values to any of the calculator memory. In the example, there are 8 unnamed memory cells (Memory0 to Memory7) plus two named cells (PipeHeight with a value of 45.2 and PipeFraction with a value of 54.4).



Some named memory values are displayed in the Memory screen of Best Calculator. Display the Best Calculator memory screen by tapping the **Memory**  menu item.

### 32.2.1 Memory[<expression>] and Memory.<name>

There are three ways to access a cell: by number, by name, and by simple name.

**Access cells by number:** `Memory[<expression>]`. The named cells can also be accessed by number; in the picture, PipeHeight is the cell right after Memory7 and is accessed as `Memory[8]`.

**Access cells by name:** `Memory[<expression>]`. For example, the PipeHeight cell can be accessed as `Memory["PipeHeight"]`

**Access cells by simple name:** `Memory.<constant_name>` where the constant looks like a variable name (without double quotes) and not a string or number. The memory cell name must be compatible with the rules for variable names. For example, the name can't have spaces or start with a number. The PipeHeight cell can be accessed as `Memory.PipeHeight`.

### 32.2.2 GetOrDefault and IsSet functions

**Is the memory set?** You can tell if a memory cell is set or not in two ways:

`Memory.GetOrDefault(<expression>, <default value>)` returns either the memory value (if it's set) or the supplied default value if not.

`Memory.IsSet(<expression>)` returns true or false (1 or 0) if the memory value is already set.

These functions are commonly used to let you “smart initialize” a value. For example, some calculations use a seldom-changed value (for example, money conversions). You can use `Memory.GetOrDefault` as the default value in an input expression and then save the value that the user enters.

#### Example money conversion program:

```
REM
REM The defaults here are roughly the conversion
REM rate from yen to australian dollars.
REM 1 yen is about 0.011 australian dollar;
REM 10000 yen is about 110 australian dollars.
REM
prompt1 = "Conversation rate <from> to <to>" ↴
    + "[e.g., yen to australian dollars]"
prompt2 = "Amount to convert [e.g., amount in yen]"
rate = INPUT DEFAULT Memory.GetOrDefault ↴
    ("ConversionRate", 0.011) PROMPT prompt1
Memory.ConversionRate = rate
amount = INPUT DEFAULT Memory.GetOrDefault ↴
    ("ConversionAmount", 10000) PROMPT prompt2
Memory.ConversionAmount = amount
value = amount * rate
Calculator.Message = "Convert " + amount + ↴
    " at a rate of " + rate + " is " + value
Calculator.Value = value
```

### 32.2.3 Memory technical details

**The calculator memory is both persistent and roaming.** Persistent means that it keeps its value between program runs; it's never automatically reset to zero or other default states. Roaming means that the data roams between your sessions on different computers.

The DUMP command will print out all the memory values to the scrolling console screen.

Best Calculator Memory display will show the first 10 memory slots. However, you can actually access more than that with BC BASIC. Memory slots in BC BASIC can be numbered up to 100. You can show the Best Calculator Memory screen by pressing the Memory  key on the left menu.

Interesting cases for programmers:

1. The memory can only store doubles; it does not store strings.
2. Using indexes less than 0 or more than 100 will silently fail, as will numeric indexes which are not integer (e.g., 3.5). These reads will always return a no such value.
3. An integer index and the string version of the index will refer to the same cell. For example, Memory[1] and Memory["1"] refer to the same memory cell.
4. If you used a named cell and there isn't a cell already with that name, the cell won't be visible in the display. If the user then renames a cell in the memory display with that name, future reads and writes will be to that user-named cell. BC BASIC doesn't place any limit on how many cells there are. Given any particular name, BC BASIC will prefer to save and load from the visible memory cells, but will use the non-visible cells if it has to. BC BASIC won't ever change the name of a cell; that's up to you.
5. There isn't any way to delete memory cells. Once set, the memory cell is created forever. You can, of course, override the memory value.

**Examples of using the Memory extension:**

CLS

REM

```
REM You can use integer index values
REM
Memory[0] = Memory[0] + 1
Memory[1] = Memory[1] +10
PRINT "Numeric Index: "; Memory[0]; " "; Memory[1]

REM
REM You can use simple index names
REM
Memory.PipeHeight = Memory.PipeHeight + 1
PRINT "Simple name: "; Memory.PipeHeight

REM
REM You can use index names with square brackets
REM
Memory["PipeHeight"] = Memory["PipeHeight"] + 1
PRINT "Const Index: "; Memory["PipeHeight"]

REM
REM You can use variables and expressions
REM      in the index name
REM
name = "PipeHeight"
Memory[name] = Memory[name] + 1
PRINT "Variable Index: "; Memory[name]

prefix = "Pipe"
suffix = "Height"
Memory[prefix + suffix] = ↓
    Memory[prefix + suffix] + 1
PRINT "Expression Index: "; ↓
    Memory[prefix + suffix]

REM Some memory isn't set
a = Memory.NotSet
isset = Memory.IsSet ("PipeHeight")
isnotset = Memory.IsSet ("NotSet")
ns = Memory.NotSet =Memory.NotSet

REM Memory.GetOrDefault returns either the
REM memory value or the default value
REM depending on whether the memory was
REM set or not.
default = Memory.GetOrDefault ("NotSet", 34)
notdefault = ↓
    Memory.GetOrDefault ("PipeHeight", 34)

DUMP
```

### 32.3 MATH EXTENSION

BC BASIC includes a full set of math functions and constants divided into categories for trigonometry, rounding, logarithm and power functions and other functions.

#### 32.3.1 Trigonometry (Math.Sin (radians) and more)

BC BASIC does all trigonometry calculations in radians. The function Math.DtoR (degrees) will convert degrees to radians and Math.RtoD(radians) will convert radians to degrees.

Function	Notes
Math.Acos(value)	Calculates the inverse of Math.Cos; given a value will compute the corresponding angle in radians.
Math.Asin(value)	Calculates the inverse of Math.Sin; given a value will compute the corresponding angle in radians.
Math.Atan(value)	Calculates the inverse of Math.Tan; given a value will compute the corresponding angle in radians. Note that Math.Tan of 90° is infinite.
Math.Atan2(y, x)	Calculates the inverse tangent of given an Y and X value. Note that Y is given first. This matches most common program languages including C#, Java, Fortran, C and JavaScript. Unlike the Math.Atan function, Math.Atan2 can handle angles of 90°
Math.Cos (radians)	Calculates the cosine of an angle given in radians
Math.Cosh (radian)	Calculates the hyperbolic cosine of an angle given in radians
Math.DtoR (degrees)	Converts degrees to radians
Math.RtoD (radians)	Converts radians to degrees
Math.Sin (radians)	Calculates the sin of an angle given in radians
Math.Sinh (radians)	Calculates the hyperbolic sin of an angle given in radians
Math.Tan (radians)	Calculates the tangent of an angle given radians
Math.Tanh (radians)	Calculates the hyperbolic tangent of an angle given radians

#### 32.3.2 Rounding and sign (Floor(), Round() and more)

Function	Notes
Math.Abs (value)	Calculate the absolute value of a number.

Math.Ceiling (value)	Calculates the ceiling of a number. The ceiling is the number rounded up to the nearest integer. For example, Math.Ceiling (2.2) is 3. Ceilings of negative numbers round up (e.g., to be closer to zero), so Math.Ceiling (-2.2) is -2.
Math.Floor (value)	Calculates the floor of a number. The floor is the number rounded down to the nearest integer. For example, Math.Floor (2.8) is 2; Math.Floor (-2.8) is -3.
Math.Frac (value)	Returns the fractional part of a number (the part after the decimal sign). Math.Frac(3.456) is 0.456.  For negative numbers, Math.Frac returns the difference between the number at the next higher number. For example, Math.Frac(-7.4) is 0.6.  Math.Floor(value) + Math.Frac(value) is equal to the original value.
Math.Max (value, ...)	Returns the largest value of a set of numbers. You may give one or more values to Math.Max()
Math.Min (value, ...)	Returns the smallest value of a set of numbers. You may give one or more values to Math.Max()
Math.Mod (v1, v2)	Returns the remainder when v1 is divided by v2. For example, Math.Mod(7,3) is 1 because 3 goes into 7 2 times with a remainder of 1. Math.Mod(7.6, 3.1) is 1.4 because it's the remainder after 3.1 is multiplied by 2.
Math.Round (value)	Calculates the rounded value of a number. The rounded value is the one closest to an integer. Math.Round (2.2) is 2; Math.Round (2.8) is 3. If a number is a ".5" number, it is rounded down (technically, rounded towards zero; Math.Round (2.5) is 2, and Math.Round (-2.5) is -2.)

Math.Sign (value)	Return the sign of a number. The sign is 1 for positive values, -1 for negative values, and 0 for zero.
Math.Truncate (value)	Calculates the truncated value of a number. The truncated value is the integer value closest to zero. For positive numbers, this is like Math.Floor (for example, Math.Truncate (2.8) is 2). For negative numbers, this is like Ceiling (for example, Math.Truncate (-2.8) is -2, the integer closer to zero)

## 32.3.3 Logarithm and power functions (Math.Log, Math.Exp, and more)

Function	Notes
Math.Exp(value)	Calculates the value $e^{value}$ for any given value. This is the reverse of the Math.Log function
Math.Log (value) Math.Log (value, base)	Can do two different calculations. When given just one value, calculates the natural (base e) logarithm of the given value. When given two numbers, calculates the $\log_{base}$ of the value.
Math.Log2 (value)	<p>Calculates the base-2 logarithm of a number. This is useful when dealing with computer math. Simple example: you're writing a program, and certain variable will hold a number from 0 to 934. How many bits are needed to hold this value? Answer: Math.Log2(934) is about 9.87; rounding up, you discover than you will need a 10-bit field to hold the number.</p> <p>Sophisticated example: You need to store 200 numbers, each of which is a value 0 to 11 (inclusive, 12 total values). Assuming best bit packing but no compression, how much space do you need? The answer is that <math>200 * \text{Math.Log2}(12) = 717</math> bits; divide by 8 to discover that your data will fit into 90 bytes of space.</p>
Math.Log10(value)	<p>Calculates the base-10 logarithm of a number. This is useful when rounding a number up to the nearest power-of-ten.</p> <p>For example, you want the first power of ten (e.g., 10, 100, 1000) of 783. Math.Log10 (783) is 2.89; rounded up this is 3. Math.Pow (10, 3) is 1000, and that's the closest larger power of ten of the number.</p>
Math.Pow (x, y)	Calculates $x^y$ . For example, Math.Pow (10, 3) is 1000.
Math.Sqrt(value)	Calculates the square root of the value.

### The Math.Factorial Function

Math.Factorial(value) is the  $n!$  function. For any given integer, it returns the product of all the integers less than or equal to  $n$ . For example, Math.Factorial(5) is  $5 * 4 * 3 * 2 * 1$ , or 120.

Math.Factorial returns NaN (not a number) for any input that isn't an integer or is less than zero. Math.Factorial(0) is 1.

#### Example of using Math.Factorial

```
REM simple binding for X! so it's on the
REM main calculator page
x=Math.Factorial(Calculator.Value)
STOP x
```

### The Math.NaN function

Math.IsNaN(value) returns true (1) when the given value evaluates to a NaN value and 0 otherwise.

NaN values will propagate their values in BC BASIC, and don't compare equal to each other. The only simple way to tell a number is a NaN value is to use the Math.IsNaN function.

## 32.4 MATH.PI AND MATH.E VALUES

Two constants, Math.PI and Math.E are available from the Math extension.

#### Example of using Math.PI and Math.E:

```
REM Converts a circle AREA to DIAMETER
REM area = Math.PI * R**2, which means
value = Calculator.Value
retval = 2 * SQR (value / Math.PI)
Calculator.Message = "Converted area " + value + " to
diameter"
Calculator.Value = retval
```

### 32.5 SCREEN.CLEARLINE(<LINE>) AND SCREEN.CLEARLINES(<FROM>, <TO>)

You can clear an entire line on the screen with Screen.ClearLine (<linenumber>). This is very useful with doing data-collection work; you can provide a constantly updated display on the screen with minimal effort

Example:

```
Screen.ClearLine(3)  
PRINT AT 3,1 "Newest Data";X
```

The Screen.ClearLines(from, to) function is similar but it clears an entire set of rows at once.

### 32.6 SCREEN.REQUESTACTIVE() AND SCREEN.REQUESTRELEASE()

These two powerful functions let you keep the computer screen on even when the user is not interacting with the computer. These are very commonly used in Bluetooth and IOT applications when you need your application to run for an extended period of time without any user activity. Normally a computer or phone screen will be automatically turned off by the operating system after a period of time.

The Screen.RequestActive() will ask for the screen to stay on and Screen.RequestRelease() will allow the screen to turn off. They are counted; if you call Screen.RequestActive() twice you have to call Screen.RequestRelease() twice for the screen to be allowed to turn off.

Important: only use these for long-running applications that must remain active with not human interaction. Most programs do not fall into this category. Keeping the screen on will drain your batteries and is only recommended when you know the device will be continuously powered.

### 32.7 SCREEN.H AND SCREEN.W EXTENSION

The Screen.H and Screen.W provide the height and width, in fixed-width characters, of the screen. The most common use is to help lay out text to fix a particular screen size.

**Example: using Screen.H and Screen.W to print in the middle of the screen:**

```
CLS MAGENTA  
PrintCenter ("Hello, world!")
```

```
FUNCTION PrintCenter (str)  
Imargin = 1+INT (( Screen.W - LEN str ) / 2)  
IF (Imargin < 1) THEN Imargin = 1  
row = INT ((Screen.H) / 2)  
PRINT AT row,Imargin str  
RETURN
```

The resulting output shows a neatly-centered message.



## 33 BLUETOOTH PROGRAMMING WITH BEST CALCULATOR, IOT EDITION

Best Calculator, IOT edition is a special version of Best Calculator that lets you control multiple IOT devices using Bluetooth via the built-in BASIC programming language. Unlike the normal Best Calculator, the Best Calculator, IOT edition is a paid app. However, it does have a free trial so that you can make sure that it's suitable for your device. You should have a basic understanding of Bluetooth devices to use these features.

### 33.1 PROGRAMMING BLUETOOTH USING BC BASIC

Let's start with a simple example: let's display the names of each *paired* Bluetooth device

```
CLS BLUE
PRINT "Bluetooth Functions"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices ()
PRINT "Count", devices.Count

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PrintBluetoothInfo (bt)
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT ""
END
```

The `devices = Bluetooth.Devices ()` line gets all the paired Bluetooth devices and puts the results into an array. Then the code simply loops through the devices. We get each device and call `PrintBluetoothInfo`, passing in the device. Each device has a **Name** property with the Windows name of the device. We can also print out the device Id and the Properties.

The `Bluetooth.Devices()` method does not return every possible Bluetooth device. Devices it does not return include

- Devices which the user has not paired. BC BASIC does not include any functions to help the user pair their device
- Devices which are reserved for the system. This includes HID device like mice and keyboards, and audio devices like speakers and headphones
- Bluetooth beacons like the popular Bluetooth enabled luggage tags or the Bluetooth beacons found in some stores, museums, and work places.

Although Best Calculator, IOT edition gives you access to many of the Windows 10 Bluetooth capabilities, there are still areas where you might need to add your own customizations. The paid version of Best Calculator, IOT edition comes with the full source code for the calculator. Advanced programmers will find that they can add additional capabilities in C#.

In this and the next chapters, you'll learn

- An introduction to programming Bluetooth using BC BASIC
- The different stages of initializing and programming your device
- The different *objects* that used for for programming Bluetooth devices
- Reading data from Bluetooth *services* and *characteristics*
- Using the *specializations* for specific devices
- All the specializations for specific devices
- All the different BC BASIC programs that come with Best Calculator, IOT edition

### 33.2 INITIALIZING YOUR DEVICE AND AVAILABLE PROPERTIES

There are three stages of using a Bluetooth device. In the first stage (after calling `Bluetooth.Devices()`), you can get all paired Bluetooth devices. This provides just a few simple properties and methods.

To get to stage two, call `device.Init()` on any particular device. This will verify that your program is allowed to access the device (the user can refuse permission), and that no other program has access to the device. Once you call `Init()`, you are allowed to read and write data to the device. You can call `Init()` as

often as you wish, but you must call it at least once to get full access to the device.

In the third stage, you call the *device*.As("device type") method to get a specialized version of the device. This gives you methods that are customized for a particular device and which are much easier to use.

Example of the second stage, showing the call to Init() and the additional methods available. This is a modification of the starting program; Init() is called before calling PrintBluetoothInfo and the PrintBluetoothInfo function is updated to display the device's Bluetooth address.

```
CLS BLUE
PRINT "Bluetooth Initialization"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices ()
PRINT "Count", devices.Count

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    device.Init()
    PrintBluetoothInfo (device)
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT "ADDRESS", bt.BluetoothAddress
    PRINT "CONN.", bt.ConnectionStatus
    PRINT ""
END
```

Once you initialize an object, you can read and write data. For example, if you have a DOTTI device from wittidesign.com, you can read the power level using service 180f and reading the Power data from *characteristic* 2a19.

### 33.2.1 Error handling and Bluetooth

You must handle Bluetooth errors if you want to create a robust program.

Bluetooth calls can fail in many ways: the computer that your program is run on might not have a Bluetooth radio; the devices might be out of range or turned off or might go out of range while your program is trying to communicate.

You should check the error value returned by the Bluetooth methods.

```
CLS BLUE
PRINT "Bluetooth Initialization"

devices = Bluetooth.Devices ()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    status = device.Init()
    IF (status.IsError)
        PRINT "Unable to initialize", device.Name
    ELSE
        PrintBluetoothInfo (device)
    END IF
NEXT i

FUNCTION PrintBluetoothInfo(bt)
    PRINT "NAME", bt.Name
    PRINT "ID", bt.Id
    PRINT "PROPERTIES", bt.Properties
    PRINT "ADDRESS", bt.BluetoothAddress
    PRINT "CONN.", bt.ConnectionStatus
    PRINT ""
END
```

### 33.3 THE OBJECTS YOU USE WHEN PROGRAMMING YOUR BLUETOOTH DEVICE

You will use four main objects when you program a Bluetooth device. If you've ever done object-oriented programming, you'll recognize the terms *method* and *property* as they are used in BC BASIC. BC BASIC has some simplifications that make BC BASIC a little different from what you've seen before.

A method is like a function call (like `SIN(0.01)`) but where the "function" is the name of a variable followed by a dot and followed by the method name to call. For example, to get a list of all of the available Bluetooth devices, you call the `Devices()` method on the globally available `Bluetooth` object like this: `LET devices = Bluetooth.Devices()`. In this example, a new variable `devices` is made by calling the `Devices()` method on the `Bluetooth` object.

A property is like a method but it doesn't take any arguments and you don't need any parenthesis. For example, once you have the list of Bluetooth devices you can find out its length with the `Count` property like this: `PRINT devices.Count`.

You will always start with a single global object (like `Bluetooth`) which is always available to your program. From there, you can get other objects like an array (list) of devices and the individual devices. Unlike other language, BC BASIC does not make you use the `new` operator just to make an object. BC BASIC also only lets you call a single method at a time.

For example, you must make these calls

```
devices = Bluetooth.Devices()  
device = devices.Get(1)
```

You cannot combine the two method calls, so that you cannot make a single line like `Bluetooth.Devices().Get(1)` and have it work.

BC BASIC does not allow you to define your own classes and objects in BC BASIC.

#### 33.3.1 The Bluetooth object

Subject to your license, the `Bluetooth` object is always available to your program. It contains just a few methods. The `PickDevicesName()` lets the user pick a single Bluetooth device. `Devices()` and `DevicesName()` methods both return a list of Bluetooth devices.

The **Bluetooth.Devices()** method returns a list of all paired Bluetooth devices on the system except for devices reserved to the system.

The **Bluetooth.DevicesName("name")** method returns the same list except that the Windows device name must match the passed-in value. The value must either exactly match or can start with or end with a star to match the end or start of a name. This is very useful when you're trying to control specific devices. For example, to control the DOTTI device, it's handy to use just the devices listed in **Bluetooth.DevicesName("\*Dotti")** because DOTTI devices by default are all called "Dotti" and when changed by the DOTTI app get a name that ends with -Dotti.

The **Bluetooth.PickDevicesName("name")** method uses the same list that DevicesName returns and show a dialog that where the user can pick a single device.

### 33.3.2 The Bluetooth.Devices object (Array / ObjectValueList)

The list of available devices from Bluetooth.Devices and Bluetooth.DevicesName has a single property called Count and a single method called Get(<index>) which gets individual devices.

The *deviceList.Count* property returns the number of devices in the list. An empty list has length zero.

The *deviceList.Get(index)* method will return a single device. The index must be a value that is 1 or more and less than or equal to the Count property. For example, if you call Bluetooth.Devices and get a list whose count is 2, then you can call Get(1) and Get(2) to get the two devices.

If you call Get() with the wrong number of arguments or with an invalid argument (not a number, or out of range), the returned object is an error object.

### 33.3.3 Individual Bluetooth devices from Bluetooth.Devices

You will get individual Bluetooth devices by calling the Bluetooth.PickDevicesName() or by calling the Get() method on the list that you get by calling Bluetooth.DevicesName("<name>") or Bluetooth.Devices(). Devices start off uninitialized. From an uninitialized device, you can read the name and id (plus a little bit more). After you call Init() and the call succeeds, you can read and write data to the device using the Read and Write methods.

Properties that are always available on a device include the Name, Id and Properties of the device. For developers who need very detailed information about their devices, you can also retrieve any property from the underlying Windows.Devices.Enumeration.DeviceInformation value that each device include. A list of these properties can be found on the MSDN web site.

If you've called the *device.Init()* method, then properties from the Microsoft Windows.Devices.Bluetooth.BluetoothLEDevice are also available. Where a property from the DeviceInformation conflicts with a property from BluetoothLEDevice, the DeviceInformation value is returned. When the property names are in conflict, you can access the BluetoothLEDevice property by prepending "BLE\_" to the property name. For example, *device.Name* is the DeviceInformation name, and *device.BLE\_Name* is the BluetoothLEDevice name.

The read and write routines are explained in the "Reading data from raw Bluetooth devices" section further down.

### 33.3.4 Specializations

A number of common Bluetooth devices have *specializations* available using the *device.As("type")* method. The specializations let you control Bluetooth devices without having to know the exact services and characteristics and data formats for individual devices.

The specializations are each documented in a subsequent chapter. Specializations exist for different Bluetooth lights, gadgets and IOT sensor platforms like the TI SensorTag 2541.

## 33.4 SELECTING A DEVICE WITH PICKDEVICESNAMES AND MORE

Often you need to be able to pick a specific device to control. For example, you might have several of the NOTTI illuminated devices. You want to pick which NOTTI device you want to control.

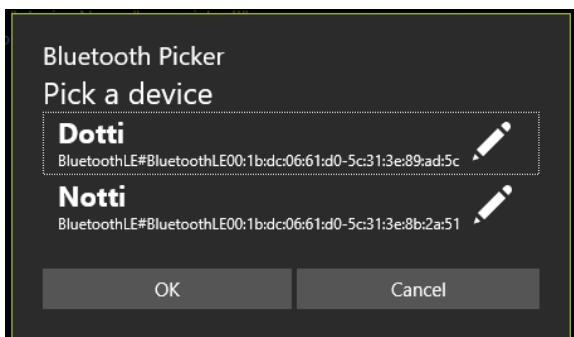
### 33.4.1 Bluetooth.PickDevicesName(<name pattern>)

*Bluetooth.PickDevicesName(<name pattern>)* is often a good choice. It will show a list of available Bluetooth devices that match your name pattern. For example, the NOTTI devices all have a name that ends with Notti. Your BASIC code would look like:

```
device = Bluetooth.PickDevicesName("*ot*")
IF (device.IsError)
```

```
PRINT "Sorry, no device was picked"  
ELSE  
    PRINT "Device ";device.Name;" was picked!"  
END IF
```

When `PickDevicesName` is called, it pops up a dialog to let the user pick a device. In the example, the user can pick any device that matches \*otti. This matches both the Notti and Dotti devices. If the user doesn't pick a device, taps cancel, or there is no matching device, the return value will be an error.



The dialog helps the user pick the correct device in two ways. Firstly, the device ID is listed; these are always unique to the actual device; they are never duplicated.

The user can set the “preferred” name of the device. The edit icon (✍) lets the user set a ‘tag’ for a Bluetooth device. That tag is associated with the Bluetooth id; as long as the ID doesn’t change, the name will remain.

The name is roamed with Best Calculator, IOT Edition data. That means that when the user sets a name on one device, all their devices that are logged into the same MSA (Microsoft) account will have access to the same name. The name will be the same regardless of which BC BASIC program is running.

### 33.4.2 `Bluetooth.DevicesName(<name pattern>)`

`Bluetooth.DevicesName(<name pattern>)` lets you access the same list of devices that the `PickDevicesName` method shows the user. You can use this when you need to perform the same action on multiple devices.

### 33.4.3 Bluetooth.Devices()

Bluetooth.Devices() returns a list of all the available Bluetooth devices regardless of their name.

Just because a device is returned doesn't mean you have full access to the device. When you call the Init method for the first time, the user can choose to let BC BASIC have full access to the device, or can choose to not grant full access. Additionally, the device might already be used by another program.

## 33.5 READING DATA FROM RAW BLUETOOTH SERVICES AND CHARACTERISTICS

What you'll mostly be doing with a Bluetooth device is reading and writing to it. To do that, you will need to know the *service* and *characteristic* that you want to read or write. These take the form of long and short GUIDs. These will be documented in the device documentation somewhere. The data might be in a specific format that you will have to read. Additionally, when you read from a device you can read either the cached data (super fast, but not as fresh) or the raw data (always fresh, but much slower).

Every Bluetooth LE device exposes a set of *services*; each service is specified with a GUID. The Bluetooth functions all take in GUIDs as a string; the string is most commonly the short version of the GUID (e.g., 1800) but will also accept the long version (e.g., 00001800-0000-1000-8000-00805f9b34fb). Each service in turn exposes a set of *characteristics* which are also specified with a short or long GUID.

**Bluetooth global object**

- Lets you find Bluetooth LE devices

**.Device() or .DevicesName() list of devices**

- call Get(index) to get an individual device

**Individual Bluetooth device**

- Call Init() to get full access to the device

**Service (like 180f for power)**

- Each device normally has several services.

**Characteristic (like 2a19, read current power level for service 180f)**

- Some services just have one characteristic; others have many characteristics
- Use the methods like ReadRawByte() to read data

**The hierarchy of Bluetooth features.**

### 33.5.1 Direct device Read routines

#### 33.5.1.1 *device.Read[Cached/Raw]Byte (service, characteristic) → byte*

There are two read methods that read a single byte from the device. Many characteristics have just a single byte of data, so it's handy to be able to just read that byte. The byte is read as an unsigned value from 0 to 255.

There are two separate Read methods available: ReadCachedByte and ReadRawByte. The ReadCachedByte method reads a cached byte; it's fast but the data may not be fresh. The ReadRawByte method asks the device for fresh data; it's likely to be slower but gets the most recent data.

#### 33.5.1.2 *device.Read[Cached/Raw]Bytes (service, characteristic) → array of data*

There are two read methods that read multiple bytes from the device. The data returned is an array (most likely a BCValueList).

The number of bytes read is available with the data.Count property

You can ready individual bytes with the `data.Get(index)` method; the index is 1-based (1 up to and including Count). The data is returned as a unsigned byte with values 0 up to and including 255.

Example:

```
data = device.ReadRawBytes("2000", "2003")
PRINT AT 1,1 data.Get(1)
PRINT AT 2,1 data.Get(2)
PRINT AT 3,1 data.Get(3)
```

As a handy convenience, call `data.GetValue(index, type)` to interpret the data in a variety of ways. Supported types are:

- “int16-le” reads two bytes of data and treats them as a LSB and MSB of a two-byte, signed integer value. For example, if the two bytes are [4 10] the result will be  $(10 * 256) + 4$ . If the second byte is more than 127, it’s treated like a signed value. For example, if the two bytes are [255 255] then the returned value is -1.

There are two separate Read methods available: `ReadCachedBytes` and `ReadRawBytes`. The `ReadCachedBytes` method reads cached data; it’s fast but the data may not be fresh. The `ReadRawBytes` method asks the device for fresh data; it’s likely to be slower but gets the most recent data.

Example: read the Power data from a DOTTI device using the raw Bluetooth read commands

```
CLS BLUE
PRINT "Read Bluetooth Power"

REM
REM How many Bluetooth devices are available?
REM
devices = Bluetooth.Devices()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    IF (device.Name = "Dotti") THEN GetPowerInfo(device)
NEXT i

FUNCTION GetPowerInfo(bt)
```

```
bt.Init()  
PRINT "NAME", bt.Name  
PRINT "POWER", bt.ReadRawByte("180f", "2a19")  
PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")  
END
```

Example: write a red dot to the DOTTI device in position (2,2) using the raw Bluetooth write commands

```
CLS BLUE  
PRINT "Write red dot onto DOTTI device"  
  
devices = Bluetooth.Devices ()  
  
FOR i = 1 TO devices.Count  
    device = devices.Get(i)  
    IF (device.Name = "Dotti") THEN WriteDot(device, 10,  
    255, 0, 0)  
NEXT i  
  
FUNCTION WriteDot(bt, pos, r, g, b)  
    bt.Init()  
    REM The fff0 is the service for many DOTTI commands  
    REM The fff3 is the characteristic used by service fff0  
    REM for many of the DOTTI commands  
    REM the 7 and 2 are the bytes that define the DOTTI  
    REM command to send (0x0702 means set LED color)  
    REM the pos is the position from 1 to 64  
    REM the r g and b are the color to set.  
    bt.WriteBytes ("ffff0", "fff3", 7, 2, pos, r, g, b)  
END
```

### 33.6 USING CALLBACKS TO READ DATA

Instead of polling your device for data you can have the Bluetooth device tell you when the data changes. You have to perform two steps get callbacks:

Tell the device to send data with `device.WriteCallbackDescriptor(service, characteristics, value)`<sup>1</sup>. method. The service and characteristic say which data value to set the notifications on. The value parameter is one of 0 = None,

---

<sup>1</sup> This exactly corresponds with the Windows Runtime

WriteClientCharacteristicConfigurationDescriptorAsync method call. That name is much, much too long for a simple language like BC BASIC!

1=Notify and 2=Indicate. Your Bluetooth device specs will say whether any particular characteristic is Notify or Indicate capable. Many devices support Notify but not Indicate. The callback-name is the name of the BC BASIC function that you want to be called when the data changes; it's ignored when the value is 0.

You can place multiple callbacks on the same service and characteristic.

Example: getting notifications for accelerations on a TI SensorTag 2541

```
CLS BLUE
PRINT AT 5,1 "Acceleration Data"

devices = Bluetooth.DevicesName ("SensorTag*")

REM
REM Constants for TI SensorTag 2541 Accelerometer
REM These are taken from the data sheets.
REM
AccService = "f000aa10-0451-4000-b000-000000000000"
AccData = "f000aa11-0451-4000-b000-000000000000"
AccConfig = "f000aa12-0451-4000-b000-000000000000"
AccPeriod = "f000aa13-0451-4000-b000-000000000000"

PRINT "COUNT", devices.Count
IF devices.Count < 1 THEN STOP

device = devices.Get(1)

PRINT "SensorTag Address", device.Init()

REM Tell the SensorTag to enable the Accelerometer
REM Config=1 means enable
REM Period=20 means get data fast (50 per second)
device.WriteBytes(AccService, AccConfig, 1)
device.WriteBytes(AccService, AccPeriod, 100)

REM 1=Notify (2=Indicate 0=None)
device.WriteCallbackDescriptor (AccService, AccData, 1)
device.AddCallback (AccService, AccData, "WriteAcc")

REM
REM Wait a little while and then turn off the Accelerometer
REM

FOR time = 1 TO 10
    PAUSE 50
```

```
PRINT AT 1,1 time
```

```
NEXT time
```

```
REM
```

```
REM Turn off the accelerometer; turn off notify; remove  
callback
```

```
REM
```

```
device.WriteCallbackDescriptor (AccService, AccData, 0)
```

```
device.WriteByte(AccService, AccConfig, 0)
```

```
device.RemoveCallback (AccService, AccData, "WriteAcc")
```

```
FUNCTION WriteAcc(device, x, y, z)
```

```
    PRINT AT 3,1 " " " " "
```

```
    PRINT AT 3,1 x, y, z
```

```
END
```

### 33.7 USING THE SPECIALIZATIONS FOR SPECIFIC DEVICES

The easiest way to control a Bluetooth device is to use one of the specializations that are available for select Bluetooth devices. Specializations include methods that can easily control the devices without having to know service and characteristics GUIDs for your device. You would normally only use the Raw Bluetooth commands either when there isn't a specialization available for your device or when you need fine-grain control over your device.

To create a specialization, call the *device*.As("⟨device type⟩") method on a device object. Each available specialization is fully described along with the device type you need to provide.

It's important to know that BC BASIC doesn't verify that you are using the right specialization! That's because you might be controlling some new device, or a variant of an existing device. Using the wrong specialization will mostly just result in the commands not working.

Example: the (3,3) pixel to green using the DOTTI specialization.

```
CLS BLUE
PRINT "Write green dot onto DOTTI device"
devices = Bluetooth.DevicesName ("*Dotti")
FOR i = 1 TO devices.Count
    device = devices.Get(i)
    Dotti = device.As ("DOTTI")
    Status = Dotti.SetPixel (3, 3, 0, 255, 0)
    PRINT "status", Status
NEXT i
```

## 34 BLUETOOTH SPECIALIZATIONS FOR SPECIFIC DEVICES

---

Note that Best Calculator, IOT edition has no special relationship with any of the device manufacturers listed below. In all cases, the devices are programmed based on generally available information.

The Network Inspector program, also from Shipwreck Software, is useful when investigating any Bluetooth device. Most devices broadcast their capabilities in a way that any programmer can read and understand how to control these devices.

The appendix includes sample Bluetooth programs for many devices.

## 34.1 bELIGHT CC2540T LIGHT DEVELOPMENT KIT

The beLight CC2540 device is a small high-intensity light development kit from Texas Instruments. It has four built-in LEDs: red, green, blue and high-intensity white. For full information, please see the TI web site at <http://www.ti.com/tool/cc2540tdk-light>.



The Bluetooth PIN for pairing the device is 0.

The default Windows name for the device is `beLight`; to get all of the devices call `devices =`

`Bluetooth.DevicesName ("beLight")`. To get the `beLight` specialization of a device, call `beLight = device.As("beLight")`.

To list available methods, use `beLight.Methods`. The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>SetColor (r, g, b, white)</code>	Sets the color to a given red, green blue and white value.
<code>ToString()</code>	Prints out a little information about your device.

The device is mostly programmed through special service ffb0. The `SetColor` call is characteristic ffb5, and takes in 4 bytes for red, green, blue and white values.

The device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to "beLight"; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

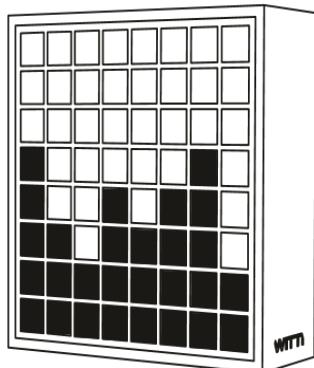
180a Device Info: 2a29 (Manufacturer, but the value is just "Manufacturer name" instead of a specific value)

## 34.2 DOTTI DEVICE

The DOTTI device is a desktop device with an 8x8 array of pixels. Each pixel can be programmed individually. For fully information, see the Witti Design web site at <http://www.wittidesign.com>.

The Bluetooth PIN for pairing the device is 123456.

The default Windows name for the device is Dotti; to get all DOTTI devices call `devices =`



`Bluetooth.DevicesName ("*Dotti")`. The regular DOTTI app can rename the device but will always add a -Dotti to the end.

To get the DOTTI specialization of a device, call `Dotti = device.As("DOTTI")`. The name is in upper case to conform to how the manufacturer describes the device in their manual.

To list available methods, use `Dotti.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>GetPower()</code>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
<code>ChangeMode(mode)</code>	Sets the mode; 0=default on icon 1=animation 2=clock 3=dice game 4=battery indicator 5=screen off

LoadScreenFromMemory(part1, part2)	Loads the visible screen from memory. Part1 and Part2 describe the memory. There is a BASIC program that helps explain what these values should be.
SaveScreenToMemory(part1, part2)	Saves the current visible screen to memory.
SetAnimationSpeed(speed)	Sets the animation speed; 1 is very fast and 6 is very slow.
SetColumn (column, r, g, b)	Sets the given column to the given red, green and blue value. Columns are numbers 1 to 8.
SetName (name)	<p>Sets the Bluetooth name of the device as returned by service 1800 characteristic 2a00. The name will be modified as needed so that it ends with the word “-Dotti” (or is simply Dotti) to match the regular DOTTI app.</p> <p>The Windows name of the device may not change until the device is reset and the re-paired.</p>
SetNameArbitrary(name)	Like SetName, but the name won’t be changed to end with “-Dotti”
SetPanel (r, g, b)	Sets the entire panel color to the given red, green and blue values.
SetPixel (x, y, r, g, b)	Sets the given pixel to the given red, green and blue values. The x and y values must be 1 to 8.
SetRow (row, r, g, b)	Sets the given row to the given red, green and blue values. Rows are numbered 1 to 8.
SyncTime(h,m,s)	Sets the time on your DOTTI device.

ToString()	Prints out a little information about your DOTTI device.
------------	--

The special DOTTI service is fff0. Most commands are sent using characteristics fff3 except for and for the SET NAME command which uses characteristic fff5. By sending command bytes to these characteristics, you can control the DOTTI device.

The DOTTI device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “Dotti”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “Manufacturer name” instead of a specific value)

180f Battery Level: 2a19 (Power, but it always seems to be 100)

fff0: D (fff3=D Data In): [writable], (fff5=C Command Channel)

### 34.3 HEXIWEAR WEARABLE PLATFORM

The Hexiwear from mikroElektronika (<http://hexiwear.com>) is a small wearable platform with weather, health and environmental sensors like accelerometers and pulse measurements. The device generates a unique pairing code each time it's paired.

As of September 2016, the device can pair with a Windows Phone but apparently does not pair with a Windows laptop or desktop.



The default Windows name for the device starts with HEXIWEAR; to get all of the devices call `devices = Bluetooth.DevicesName ("HEXIWEAR")`.

To get the specialization of a device, call `tag = device.As("Hexiwear")`.

To list available methods, use `tag.Methods`

One of the unique things about the Hexiwear is that you can't control what data you can read. There are 4 modes (accessed via `device.ReadMode()`) . The 0=Idle 2=sensor tag 5=heart 6=pedometer. You can only read data when the user has manually set the device to the correct mode.

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached.
<code>GetManufacturerName()</code>	Gets the data from service 180a characteristic 2a29.
<code>GetFirmwareRevision()</code>	Gets the data from service 180a characteristic 2a29. The device I tested against reported being 1.0.1/1.0.0. This doesn't match the spec.
<code>GetPower()</code>	Reads a battery charge percent from 0..100 from service 180f characteristic 2a19.

GetMode()	Gets the Hexiwear mode. Sensors are only available in the right mode. 0 Idle 2 Sensor Tag 5 Heart 6 Pedometer (steps+calories)
GetAccelerometer()	Returns an XYZ value Example: LET d = tag.GetAccelerometer() PRINT d.X
GetGyroscope()	Returns an XYZ of the gyroscope values.
GetMagnetometer()	Returns an XYZ of the current compass setting.
GetLight()	Returns the current brightness value in a range of 0 to 100.
GetTemperature()	Returns the current temperature in degrees C.
GetHumidity()	Returns the current humidity as a percent from 0 to 100.
GetPressure()	Returns the current pressure in
GetHeart()	Gets the heart rate (when mode is 5) in beats per minute
GetSteps()	Gets the current steps count
GetCalories()	Gets the current calorie use estimate.

## 34.4 MAGICLIGHT AND FLUX LIGHT

The MagicLight from Shultz and the Flux light are identical Bluetooth-enabled color-changing light bulbs.

The pairing code is 0.

The default Windows name for the device starts with LEDBlue; to get all of the devices call `devices = Bluetooth.DevicesName` (“LEDBLue\*”).



**MAGIC LIGHT**  
THE WORLD'S SMARTEST LIGHT BULB

To get the specialization of a device, call `light = device.As("MagicLight")`.

To list available methods, use `light.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>SetColor (r, g, b)</code>	Sets the given column to the given red, green and blue value. Columns are numbers 1 to 8.
<code>SetOff()</code>	Turns off the light
<code>SetOn()</code>	Turns on the light to the last color setting
<code>ToString()</code>	Prints out a little information about your device.

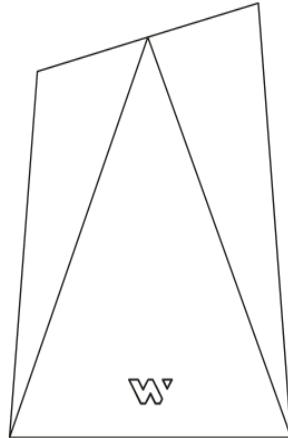
### 34.5 NOTTI DEVICE

The NOTTI device is a desktop device with a single light that can be set to any color. You can also program transitions and for colors changes to happen at a time in the future. For fully information, see the Witti Design web site at <http://www.wittidesign.com>.

The Bluetooth PIN for pairing the device is 123456.

The default Windows name for the device is Notti; to get all NOTTI devices call `devices =`

`Bluetooth.DevicesName ("*Notti")`



To get the NOTTI specialization of a device, call `Dotti`

`= device.As("NOTTI")`. The name is in upper case to conform to how the manufacturer describes the device in their manual.

To list available methods, use `Notti.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>GetPower()</code>	Gets the current battery power of the device using service 180f characteristic 2a19. The value is not cached.
<code>AlarmSetting (type, r, g, b, advance)</code>	Alarm settings. See <code>SetAlarmTime</code> for when the alarm will go off. Type is 0=off 1=every day 2=once only <code>r, g, b</code> is the color <code>advance</code> is how many minutes ahead of time to start changing.

	It's a value between 1 and 10; 1=2.5 minutes and 10=25 minutes.
ChangeMode(mode)	Sets the mode; 0=light on 1=light off 2=animation The mode doesn't seem to have any obvious effect.
SetAlarmTime(h,m)	Sets the time for the next alarm
SetColor (r, g, b)	Sets device color to the given red, green and blue value.
SetColorCustom(r1, g1, b1, r2, g2, b2)	Sets the device color to animate between color 1 (r1, g1, b1) and color 2 (r2, g2, b2)
SetName (name)	<p>Sets the Bluetooth name of the device as returned by service 1800 characteristic 2a00. The name will be modified as needed so that it ends with the word “-Notti” (or is simply Notti) to match the regular NOTTI app.</p> <p>Unlike the other NOTTI commands, SetName uses characteristic fff5, not fff3.</p> <p>The Windows name of the device may not change until the device is reset and the re-paired.</p>
SetNameArbitrary(name)	Like SetName, but the name won't be changed to end with “-Notti”
SyncTime(h,m,s)	Sets the time on your DOTTI device.
ToString()	Prints out a little information about your DOTTI device.

The special NOTTI service is fff0. Most commands are sent using characteristics fff3 except for and for the SET NAME command which uses characteristic fff5. By sending command bytes to these characteristics, you can control the NOTTI device.

The NOTTI device supports all the regular Bluetooth services and characteristics.

1800 Generic Access: 2a00 (Name) defaults to “Notti”; 2a01 (Appearance) defaults to Unknown, 2a02 (Privacy) is False.

180a Device Info: 2a29 (Manufacturer, but the value is just “Manufacturer name” instead of a specific value)

180f Battery Level: 2a19 (Power, but it always seems to be 100)

fff0: D (fff3=D Data In): [writable], (fff5=C Command Channel)

## 34.6 TI SENSORTAG 2541 (ORIGINAL VERSION)

The model 2541 SensorTag from Texas Instruments is a small, battery-powered sensor platform from TI. The sensors include an accelerometer, gyroscope, IR contactless thermometer, humidity sensor, magnetometer, barometer and on-chip temperature sensor.



The pairing code is 0.

The default Windows name for the device starts with SensorTag; to get all of the devices call `devices = Bluetooth.DevicesName ("SensorTag")`.

To get the specialization, call `tag = device.As("SensorTag2541")`.

To list available methods, use `tag.Methods`

The specialization includes the following methods

Method	Description
<code>GetName()</code>	Gets the Bluetooth name of the device using service 1800 characteristic 2a00. The value is not cached and might not be the same as the Windows name for the device from <code>device.Name</code> .
<code>AccelerometerSetup(onoff, period, callback)</code>  <code>callback (tag, x, y, z)</code>	Sets the accelerometer to on (1) or off (0); if on, then also sets the period. The callback is the name of the function to be called when the accelerometer data changes. The period is in 1/100s of a second. The x, y and z values are in "g" values and range +/-2.
<code>BarometerSetup(onoff, period, callback)</code>	Sets the barometer to on (1) or off (0). If on, also sets the period. The callback is the

callback (tag, temp, pressure)	name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C. The pressure is in hectoPascal.
ButtonSetup(onoff, callback)  callback (tag, left, right, side)	Sets the Buttons to on or off. The callback is the name of the function to be called when the button data changes.
GyroscopeSetup (axis, period, callback)  callback (tag, x, y, z)	Sets the gyroscope to on (1 to 7) or off (0). The value says which axis to enable; 7 means x, y and z. If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms.
HumiditySetup (onoff, period, callback)  callback (tag, temp, humidity)	Sets the humidity sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms. The temp is in degrees C. The humidity is relative humidity from 0 to 100.
IRSetup (onoff, period, callback)  callback (tag, ambient, object)	Sets the IR sensor to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second.

	Two temperatures are returned: the ambient (air) temperature and the contactless (object) temperature. Temperature is in degrees C.
MagnetometerSetup (axis, period, callback)  callback (tag, x, y, z)	Sets the magnetometer to on (1) or off (0). If on, also sets the period. The callback is the name of the function to be called with the data changes. The period is in 1/100s of a second; minimum value 10=100 ms.
ToString()	Prints out a little information about your device.

### Example program

```

devices = Bluetooth.DevicesNames ("SensorTag")
FOR i=1 TO devices.Count
    device = devices.Get(i)
    tag = device.As("SensorTag2541")
    tag.AccSetup(1, 100, "Acc")
NEXT i

FUNCTION Acc(tag, x, y, z)
    PRINT AT 1,1 x, y, z
END

```

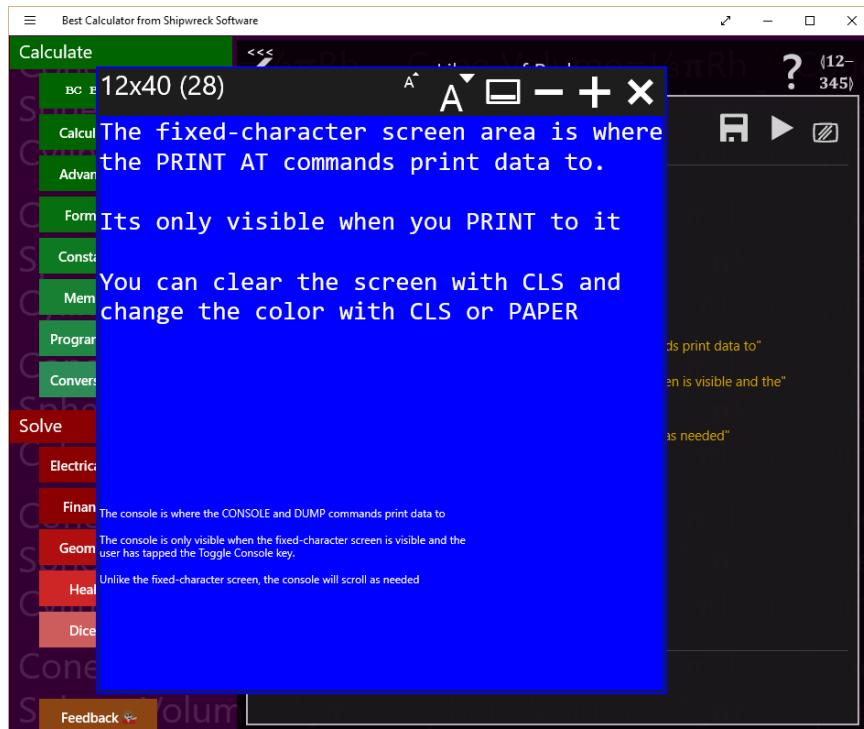
There are a number of sample programs in BC BASIC provided to demonstrate using the TI SensorTag.

## 35 GRAPHICS AND BC BASIC

BC BASIC has two ways to print to the screen. The main way is the PRINT and CLS and PAPER commands; they print to the “Fixed-character” screen. The CONSOLE and DUMP commands write to the console.

In the example, the area of the screen in blue with large type is the fixed-character screen. It’s called that because each character prints at the same width. This lets you make tables and diagrams more easily.

Underneath is the console. The primary use of the console is debugging.



Normally the console is not visible.

## 35.1 FIXED CHARACTER SCREEN COMMANDS

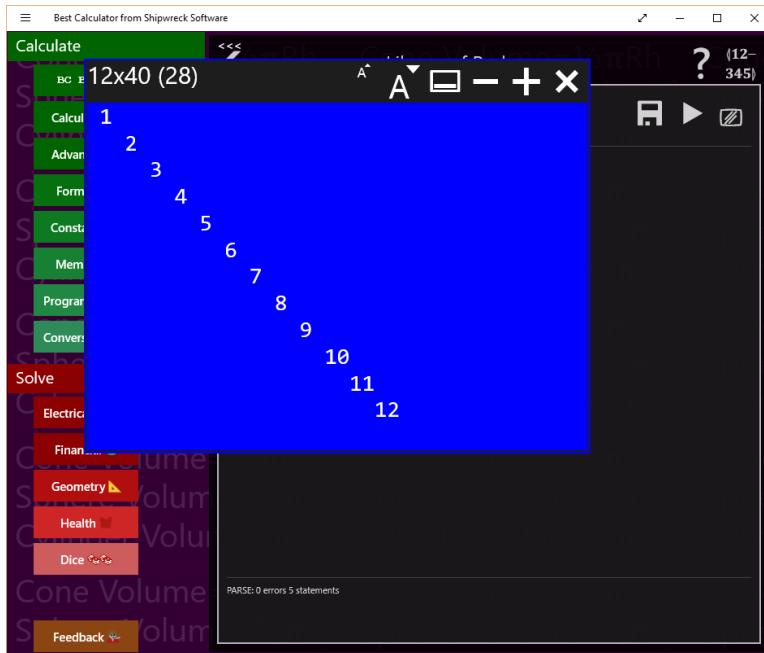
PRINT [AT line, column] <expression>

A comma between expressions will print each expression at 16-character positions

**Simple example:**

```
CLS BLUE
R = 1
10 PRINT AT R, 2*R R
R = R + 1
IF (R < 20) THEN GOTO 10
```

The following output is produced:



**Example that prints circles on the screen:**

```
REM
REM PLOT 3 CIRCLES
REM

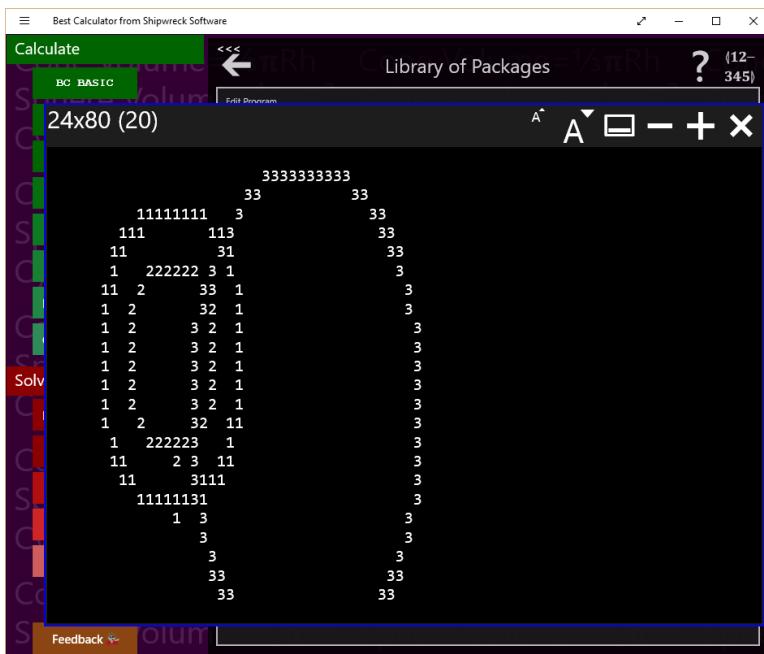
CHAR = 1
R = 8
CX = 15
CY = 12
GOSUB 1000

CHAR = 2
R=5
GOSUB 1000

CHAR=3
R=13
CX=30
CY=15
GOSUB 1000

STOP

1000 REM DO A CIRCLE USING CHAR R CX AND CY
S = 0
1010 REM TOP OF LOOP
COL = R * SIN(S) + CX
ROW = R * COS(S) + CY
PRINT AT ROW,COL CHAR
S = S + 0.05
IF (S < 7) THEN GOTO 1010
RETURN
```



## 35.2 CONSOLE COMMANDS

The console commands work on the console, a scrolling list of small-font output. The primary console commands are CONSOLE (writes to the console), DUMP (writes the name and value of all of the BC BASIC variables to the console) and the CLS and PAPER commands (which clear and possibly change the color).

**Example of using CLS to clear the screen and change the color:**

```
CLS YELLOW
```

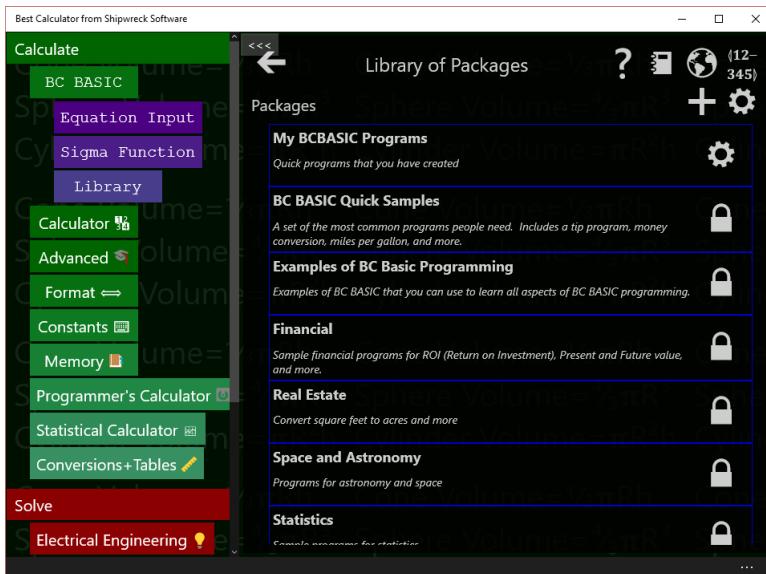
## 36 USING THE LIBRARY, STEP BY STEP

Once you have written a simple program, you might want to write more programs, and keep them all. BC BASIC includes simple Library functionality to keep all your programs. The Library also includes a series of sample programs for you to use.

In this first example, we'll write a simple program to convert square feet to acres. The steps are listed in the diagram and will be described in detail in each section.



## 36.1 ADD A NEW PACKAGE FOR YOUR PROGRAM



In this example, you will create a new program. First you need to select a package to put your program in.

Press the BC BASIC key and select Library. The *Library of Packages* screen will pop up. A package is a bundle of individual *programs*; you're going to make a single new *package* that contains a simple *program*. The program will convert from square feet to acres.

First, you need to make the *package* that your programs will be part of.

Tap the + key to add a new package. A new package will be created with a default name of “New Package”.

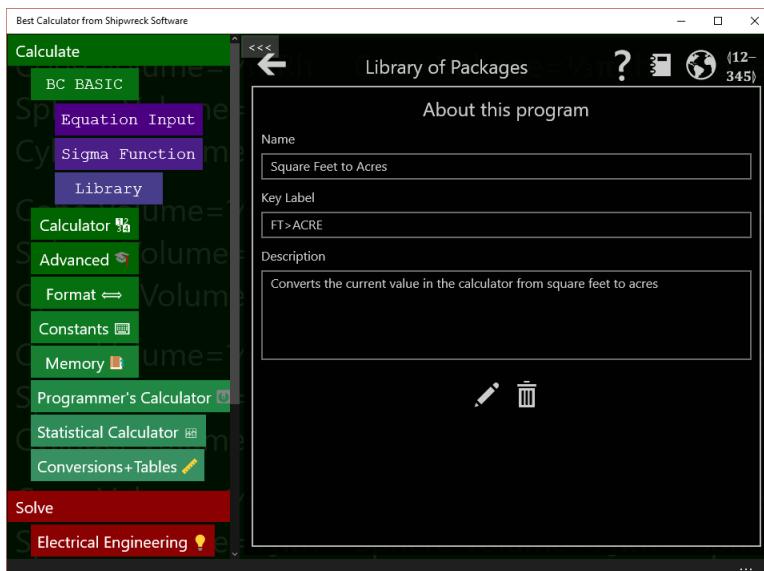
After you've written more programs and want to create more packages, you'll probably want to rename this package. Do that by tapping the

package's GEAR key (  ). Then change the name and description. The changes take place right away. Tap the BACK ARROW to get back to the list of packages.

## 36.2 ADD A NEW PROGRAM TO YOUR PACKAGE

Your new package is now ready for you to add your new program. Tap the package to see the list of programs in the package, and then tap the + to add a new program. It will be given the name NewProgram . It's also got a description and some code.

Tap the program's GEAR (  ) key to bring up the About this program screen.



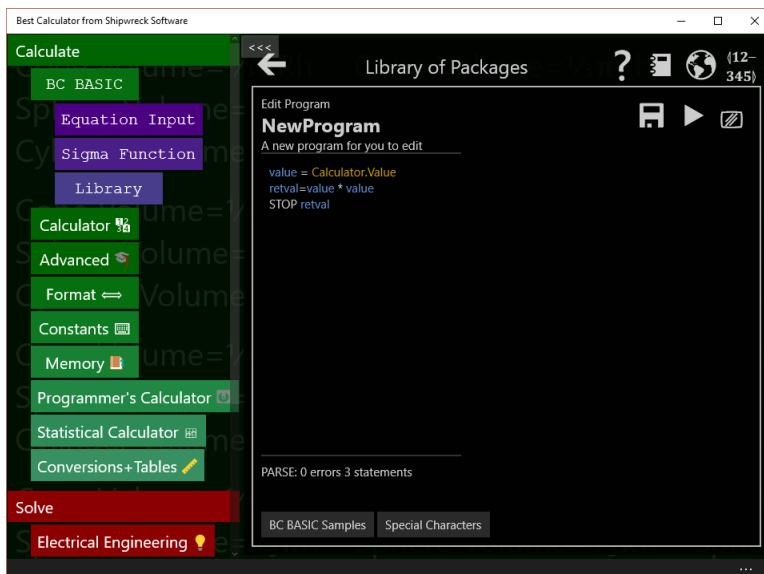
Change the name to "Square Feet to Acres" and the description to "Converts the current value in the calculator from square feet to acres". You don't have to do anything special to save the name and description; they are saved automatically when your program is saved.

You can also set the key label value. If you bind a program to a programmable key, this string will be displayed. Set it now to "FT>ACRE". The string has to be short to fit onto a key.

## 36.3 EDIT THE PROGRAM TO DO THE CONVERSION

To get to the edit screen, you can either tap the EDIT key at the bottom of the About this program screen, or you can tap the BACK ARROW key to get back to the Programs list and then tap the program's EDIT key.

The program starts out with a sort of mini-sample. You'll be deleting the mini-sample code and replacing it with your own.



Most conversion programs follow the same pattern:

1. Get a number from the calculator display. This is the number of square feet.
2. Multiple or divide it to get the new value. To convert square feet to acres, just divide by 43560.
3. Output a string to the calculator display to say what we've done
4. Return the numeric value so it's set into the calculator

The program to do these is:

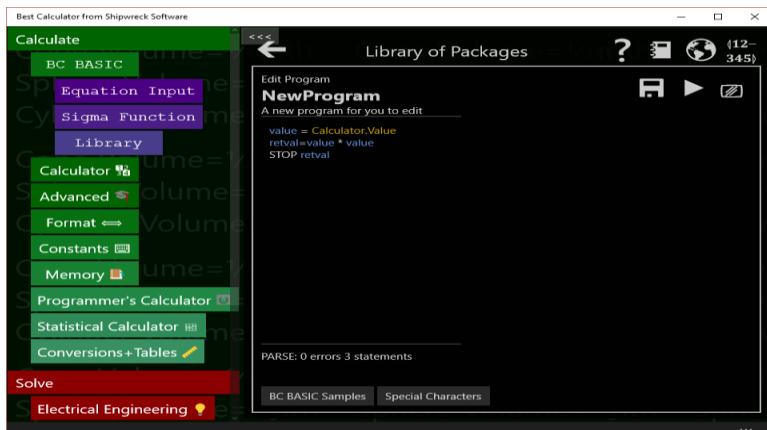
```
value = Calculator.Value
newValue = value / 43560
Calculator.Message = "Converted " + value ↴
    + " square feet to acres"
STOP newValue
```

Each line corresponds to one of our steps. Enter this code into the program area.

## 36.4 RUN THE PROGRAM TO TEST IT

Before we run the program, we need to have a “known good” conversion. Type “Convert 10000 square feet to acres” into a web search; it should tell you that 10000 square feet is 0.229568 acres.

To test the program, tap the Calculator key; this dismisses the BC BASIC programming area and pops up the calculator. Type in the starting value of 10000.



Now tap the BC BASIC key. The BC BASIC programming area pops up again, right where you were. Tap the RUN key to run the program. Your program is automatically saved when you run the program. You can also press the F5 key to run your program.

When you run the program from the editor, it will show a dialog box with the results. It's not displayed when you bind the program to a key. Tap the Calculator key again to see the calculator. You should see this:

The program works! It's done the conversion, and reminded you of



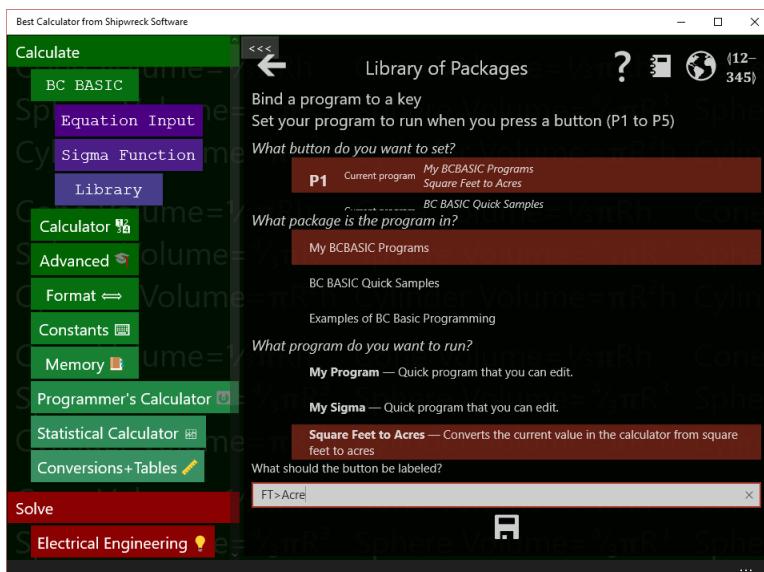
what exactly it did.

## 36.5 BIND THE PROGRAM TO A KEY

Now we're going to *bind* the program to one of the programmable keys on the calculator. They are the ones marked P1 to P4. Best Calculator comes with these keys already programmed to some common tasks.

In any of the programming dialogs, tap the BIND key (); it's the one in the upper-right corner. It brings up the binding dialog.

To bind a key, you first pick the key to bind, the package and program to bind to, and then press save. You can also set what the key should say.



The first question is *What key do you want to bind to?* Tap one of the keys in the key list (labeled P1, P2, P3 and so on) to pick a key to bind to. People often just pick key P1. The key list tells you what package and program the key is currently bound to. This helps you pick the right key to use.

The second question is *What package is the program in?* All the possible packages are listed. As you tap on a package, the next list changes to show the programs in that package. Tap on **New Package** to pick your new package. If you've change the name of the package, pick the new name.

The third question is *What program do you want to run?* Tap **Square feet to acres** to select your new program.

You can optionally set the label of the key. The default value is set from the “Key Label” value when you update the program data (the “About this program” screen). You should have already set it to FT>ACRE.

Lastly, **be sure to tap the SAVE key** (). Your selection isn’t saved until you press save.

Now verify that the key works how it should. Tap the Calculator key to see the calculator again. Now enter a value into the calculator. You might want to enter 10000 since you already know how many acres it is. Now press the key you bound (probably the P1 key). The value in the calculator screen should be replaced with 0.229568 (and there is a message that it just converted 10000 square feet to acres).

## 36.6 NEXT STEPS

Now you’ve seen the dialogs and screens that you need to use to create a BC BASIC program. Your next step is to try it! Pick a problem that you have where you work, at home, for your hobby, or your schoolwork. Conversion programs are often a great way to start; lots of times you have to convert one value to another.

If you need to make a conversion program, take a look at the code in the Astronomy package. It demonstrates a more advanced way how to make a single central library program that handles lots of conversions. Or you can just write each one just like you did this one.

Sometimes you have to enter several numbers. The Arc Length program shows how you can prompt the user for several values. The Money Conversion program in the Quick Samples library shows how you can ask the user for input and remember the last value entered. By setting a default value for the value to be entered, you can really make your work flow go faster.

## 37 APPENDIX: SAMPLE PROGRAMS

---

This appendix includes many of the sample programs that come with Best Calculator and Best Calculator, IOT Edition. You can use these sample to help create your own program and understand how different Bluetooth devices are programmed.

All of the packages that start with BT: require the Bluetooth capabilities of Best Calculator, IOT Edition.

All of the packages that start with EX: work with all the BC Basic that's included in all recent Best Calculator editions.

### 37.1 BT: AN OVERVIEW OF BLUETOOTH

Introduces Bluetooth programming. Call the Bluetooth.Devices() method to get a list of paired Bluetooth devices for a system. For each individual device, you can get the name or you can make Bluetooth calls into individual devices. The device.Init() call is needed to get real Bluetooth device data.

#### 37.1.1 List Bluetooth devices

Call the Bluetooth.Devices() method to get a list of paired Bleutooth devices. For each device in the list you can get the name even without call the device.Init() method. The list.Count property is the way to get the length of the list.

```
CLS BLUE
PRINT "Available Bluetooth devices"

devices = Bluetooth.Devices ()

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PRINT "NAME", device.Name
NEXT i
```

```
n = devices.Count  
PRINT ""  
PRINT "" + n + " devices were found"
```

### 37.1.2 Pick a Bluetooth device

The `Bluetooth.PickDevicesName(<name pattern>)` method lets the user select a single Bluetooth device from a matching list.

```
CLS BLUE  
PRINT "PickDevicesName lets the user select"  
PRINT "a single Bluetooth device from a list"  
PRINT ""  
device = Bluetooth.PickDevicesName("*")  
IF (device.IsError)  
    PRINT "Sorry, no device was picked"  
ELSE  
    PRINT "Device ";device.Name;" was picked!"  
    PRINT device.Properties  
END IF
```

### 37.1.3 Power

Get real data from each Bluetooth device using the raw Bluetooth read commands. This program builds on the List program: each device is initialized with the `device.Init()` call. Once initialized, standard Power data is retrieved from each device. There are two types of reads: cached reads (like `device.ReadRawByte`) are faster because they use the data that the operating already knows. The raw reads will use the Bluetooth radio and will ask the device for data. Each raw call gets the most up to date data (but will be slower).

```
CLS BLUE  
PRINT "Read Bluetooth Power"  
  
REM  
REM How many Bluetooth devices are available?
```

```
REM  
devices = Bluetooth.Devices ()  
  
FOR i = 1 TO devices.Count  
    device = devices.Get(i)  
    PRINT "NAME", device.Name  
    GetPowerInfo(device)  
NEXT i  
  
REM Get power data using the RAW bluetooth routines  
FUNCTION GetPowerInfo(bt)  
    PRINT "Init", bt.Init()  
    PRINT "POWER", bt.ReadRawByte("180f", "2a19")  
    PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")  
    PRINT "BLE_Name", bt.BLE_Name  
END
```

## 37.2 BT: bELIGHT

Supports the bELIGHT CC2540T developer kit from TI. This is a small, Bluetooth-enabled high-output light. Unlike some other lights, it includes a bright white light plus individual red, green and blue lights. This lets you make a light whose color can be adjusted to be cooler (more blue) or warmer (more red). The BC BASIC device.As("bELIGHT") specialization includes just one method, SetColor(r, g, b, w) that lets you set the red, green, blue and white values. Valid values are 0 (off) to 255.

### 37.2.1 Green

Turns the device green

```
CLS BLUE  
PRINT "Sets the bELIGHT to green"  
  
device = Bluetooth.PickDevicesName("bELIGHT")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE
```

```
beLight = device.As ("beLight")
REM The four parameters are Red, Green, Blue and White values.
REM White is very bright
REM They must be in the range 0 to 255
Status = beLight.SetColor (0, 255, 0, 0)
PRINT "status", Status
END IF
```

### 37.2.2 Red

Turns the beLight red

```
CLS BLUE
PRINT "Sets the beLight to red"

device = Bluetooth.PickDevicesName("beLight")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    beLight = device.As ("beLight")
    REM The four parameters are Red, Green, Blue and White values.
    REM White is very bright
    REM They must be in the range 0 to 255
    Status = beLight.SetColor (255, 0, 0, 0)
    PRINT "status", Status
END IF
```

## 37.3 BT: DOTTI

Demonstrates how to control the Dotti device (from Witti design company). The DOTTI device is a small desktop device with an 8x8 array of pixels. Each pixel can be programmed individually. The Pairing code is 123456.

### 37.3.1 An introduction

An introduction to using the DOTTI specialization

```
CLS BLUE
```

```
PRINT "An introduction to the DOTTI specialization"
```

```
device = Bluetooth.PickDevicesName("*Dotti")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    PRINT "Dotti Device"
```

```
    PRINT "Name", device.Name
```

```
    Dotti = device.As ("DOTTI")
```

```
    PRINT "Methods", Dotti.Methods
```

```
    PRINT "GetName()", Dotti.GetName()
```

```
    PRINT "GetPower()", Dotti.GetPower()
```

```
    PRINT " "
```

```
END IF
```

### 37.3.2 Change Mode

Changes the mode of the Dotti device (clock, animation, etc)

```
CLS BLUE
```

```
PRINT "Set DOTTI mode"
```

```
PRINT "0=default on"
```

```
PRINT "1=Animation"
```

```
PRINT "2=Clock"
```

```
PRINT "3=Dice Game"
```

```
PRINT "4=Battery Indicator"
```

```
PRINT "5=Off"
```

```
device = Bluetooth.PickDevicesName("*Dotti")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    device = devices.Get(i)
```

```
    Dotti = device.As ("DOTTI")
```

```
    mode = INPUT DEFAULT 1 PROMPT "What mode?"
```

```
    Status = Dotti.ChangeMode(mode)
```

```
PRINT "status", Status  
END IF
```

### 37.3.3 List BT Devices

Lists all of the available paired Bluetooth devices and prints both the Windows version of the name and the BLE (Bluetooth device) version of the name. These can be different on DOTTI devices: when you use the DOTTI commands to change the name, the BLE name will change. But the Windows name might only change after restarting or re-pairing the device.

```
CLS BLUE  
PRINT "Read Bluetooth Power"  
  
REM  
REM How many Bluetooth devices are available?  
REM  
devices = Bluetooth.Devices ()  
  
FOR i = 1 TO devices.Count  
    device = devices.Get(i)  
    PRINT "NAME", device.Name  
    GetPowerInfo(device)  
NEXT i  
  
REM Get power data using the RAW bluetooth routines  
FUNCTION GetPowerInfo(bt)  
    PRINT "Init", bt.Init()  
    PRINT "POWER", bt.ReadRawByte("180f", "2a19")  
    PRINT "CACHE", bt.ReadCachedByte("180f", "2a19")  
    PRINT "BLE_Name", bt.BLE_Name  
END
```

### 37.3.4 Load screen from memory

Loads the screen from memory (animation, dice, notifications, etc)

```
Status = Dotti.ChangeMode(2)
```

```
CLS BLUE
```

```
PRINT "Load Screen"
```

```
device = Bluetooth.PickDevicesName("*Dotti")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    device = devices.Get(i)
```

```
    Dotti = device.As ("DOTTI")
```

```
    ez = INPUT DEFAULT 24 PROMPT "What icon?"
```

```
    Part1=EZValueToPart1(ez)
```

```
    Part2=EZValueToPart2(ez)
```

```
    PRINT "ez", ez
```

```
    PRINT "Part1", Part1
```

```
    PRINT "Part2", Part2
```

```
    Dotti.LoadScreenFromMemory(Part1, Part2)
```

```
END IF
```

```
FUNCTION ShowAllScreens(Dotti)
```

```
FOR ez = 1 TO 9
```

```
    Part1=EZValueToPart1(ez)
```

```
    Part2=EZValueToPart2(ez)
```

```
    PRINT ez, Part1, Part2
```

```
    Dotti.LoadScreenFromMemory(Part1, Part2)
```

```
    PAUSE 50
```

```
NEXT ez
```

```
END
```

```
FUNCTION EZValueToPart1(ez)
```

```
IF (ez = 0) THEN RETURN 0
```

```
IF (ez < 9) THEN RETURN 2
```

```
IF (ez < 17) THEN RETURN 1
```

```
IF (ez < 23) THEN RETURN 2
```

```
IF (ez = 23) THEN RETURN 0x10
```

```
IF (ez = 24) THEN RETURN 0x20
```

```
IF (ez = 25) THEN RETURN 0x30
IF (ez = 26) THEN RETURN 0x40
IF (ez = 27) THEN RETURN 0x50
IF (ez = 28) THEN RETURN 0x60
IF (ez = 29) THEN RETURN 0x70
IF (ez = 30) THEN RETURN 0x80
IF (ez = 31) THEN RETURN 0x90
PRINT "p1"
RETURN 0

FUNCTION EZValueToPart2(ez)
IF (ez = 0) THEN RETURN 0
IF (ez < 9) THEN RETURN ((ez-1)*16+0x80)
IF (ez < 17) THEN RETURN ((ez-9)*16)
IF (ez < 23) THEN RETURN ((ez-17)*16)
IF (ez < 32) THEN RETURN 0
PRINT "p2"
RETURN 0
```

### 37.3.5 Raw Bluetooth commands

Writes a single red dot into position (2,2) on a Dotti device using the raw Bluetooth commands

```
CLS BLUE
PRINT "Write red dot onto DOTTI device"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    device = devices.Get(i)
    IF (device.Name = "Dotti") THEN WriteDot(device, 10, 255, 0, 0)
END IF

FUNCTION WriteDot(bt, pos, r, g, b)
    bt.Init()
```

```
REM The fff0 is the service for many DOTTI commands  
REM The fff3 is the characteristic used by service fff0  
REM    for many of the DOTTI commands  
REM the 7 and 2 are the bytes that define the DOTTI  
REM command to send (0x0702 means set LED color)  
REM the pos is the position from 1 to 64  
REM the r g and b are the color to set.  
bt.WriteBytes ("fff0", "fff3", 7, 2, pos, r, g, b)  
END
```

### 37.3.6 SetAnimationSpeed

Sets the animation speed (and does a ChangeMode to the animation)

```
CLS BLUE  
PRINT "Set Dotti animation speed"  
  
device = Bluetooth.PickDevicesName("Dotti")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    device = devices.Get(i)  
    Dotti = device.As ("DOTTI")  
    speed = INPUT DEFAULT 1 PROMPT "What speed (1 to 6)?"  
    Status = Dotti.ChangeMode(1)  
    Status = Dotti.SetAnimationSpeed(speed)  
  
    PRINT "status", Status  
END IF
```

### 37.3.7 SetColumn and SetRow to random lines

Draw random color lines on a Dotti device using SetColumn and SetRow

```
CLS BLUE  
PRINT "Write random lines"
```

```
device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")

CLS GREEN
PRINT device.Name
Dotti.SetPanel (50, 50, 50)
REM set to a medium kind of green
FOR n = 1 TO 200
    x = Math.Ceiling(RND * 8)
    green = Math.Ceiling(RND*255)
    red = Math.Ceiling(RND*255)
    blue = Math.Ceiling(RND*255)
    Dotti.SetColumn (x, red, green, blue)

    x = Math.Ceiling(RND * 8)
    green = Math.Ceiling(RND*255)
    red = Math.Ceiling(RND*255)
    blue = Math.Ceiling(RND*255)
    Dotti.SetRow (x, red, green, blue)

NEXT n
END IF
```

### 37.3.8 SetName of a Dotti device

Sets the name of a Dotti device

```
CLS BLUE
PRINT "Change the name of a Dotti device"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
```

```
PRINT device.Name  
Dotti = device.As ("DOTTI")  
Status = Dotti.SetName ("Dotti")  
PRINT "status", Status  
END IF
```

### 37.3.9 SetPixel to random green dots

Displays random green dots on the Dotti using the dotti.SetPixel command.

```
CLS BLUE  
PRINT "Write green dot onto DOTI device"  
  
device = Bluetooth.PickDevicesName("*Dotti")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    Dotti = device.As ("DOTTI")  
  
    CLS GREEN  
    PRINT device.Name  
    Dotti.SetPanel (0, 10, 0)  
    REM set to a medium kind of green  
    FOR n = 1 TO 200  
        x = Math.Ceiling(RND * 8)  
        y = Math.Ceiling(RND*8)  
        green = Math.Ceiling(RND*255)  
        Status = Dotti.SetPixel (x, y, 0, green, 0)  
    NEXT n  
END IF
```

### 37.3.10 SetPixel to write a single green dot

Writes a green dot to a Dotti device using the dotti.SetPixel() command

```
CLS BLUE
```

```
PRINT "Write green dot onto DOTTI device"
```

```
device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")
    Status = Dotti.SetPixel (3, 3, 0, 255, 0)
    PRINT "status", Status
END IF
```

### 37.3.11 Sync Time

Sets the time on the Dotti device

```
CLS BLUE
```

```
PRINT "SyncTime --- set Dotti time"
```

```
device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Dotti = device.As ("DOTTI")
    PRINT "DOTTI", Dotti
    h = INPUT DEFAULT 10 PROMPT "What hour?"
    m = INPUT DEFAULT 10 PROMPT "What minute?"
    s = INPUT DEFAULT 15 PROMPT "What second?"
    Status = Dotti.ChangeMode(2)
    Status = Dotti.SyncTime(h, m, s)
    PRINT "status", Status
END IF
```

## 37.4 BT: HEXIWEAR

The Hexiwear is a small hexagonal "wearable" IOT device from <http://www.hexiwear.com/>. It includes a number of sensors including heart rate, steps, weather and the normal accelerometer and

gyroscope. The Hexiwear device.As("Hexiwear") specialization gives you easy access to all of the Hexiwear data.

### 37.4.1 Accelerometer

This program provides a constant stream of accelerometer updates from the device. In the program, all of the Hexiwear devices are listed (a device is known to be a Hexiwear device if it's name is HEXIWEAR). For each device found, the device.As("Hexiwear") specialization is created. Then the program goes into a loop, getting the data and printing it to the screen.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate Hexiwear Accelerometer"

device = Bluetooth.PickDeviceName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT AT 6,1 "Got a device", device.Name

REM Now poll for data.
FOR time = 1 TO 10
    Screen.ClearLine (3)
    PRINT AT 3, 1 "TIME", time

    Screen.ClearLine(1)
    data = tag.GetAccelerometer()
    PRINT AT 1,1 data.Get(1), data.Get(2), data.Get(3)

    PAUSE 50
NEXT time

PRINT AT 9, 1 "FINISH", status
tag.Close()
```

**END IF**

### 37.4.2 Compass

This program provides a constant stream of compass updates from the device. In the program, all of the Hexiwear devices are listed (a device is known to be a Hexiwear device if its name is HEXIWEAR). For each device found, the device.As("Hexiwear") specialization is created. Then the program goes into a loop, getting the data and printing it to the screen.

```
CLS BLUE
PRINT AT 7,1 "Demonstrate Hexiwear Magnetometer"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT AT 8,1 "Got a device", device.Name

    data = tag.GetMagnetometer()
    REM Now poll for data.
    FOR time = 1 TO 99
        Screen.ClearLine(11)
        PRINT AT 11, 1 "TIME", time
        data = device.ReadRawBytes("2000", "2003")
        FOR i=1 TO 6
            Screen.ClearLine(1)
            PRINT AT i,1 data.Get(i)
        NEXT i

        GOTO 90
    data = tag.GetMagnetometer()
    PRINT AT 1,1 INT (data.Heading)
    Screen.ClearLine(3)
    PRINT AT 3,1 data.X, data.Y
```

```
Screen.ClearLine(4)
PRINT AT 4,1 data.Z
90 REM bottom
PAUSE 10
NEXT time

PRINT AT 11, 1 "FINISH", status
tag.Close()
END IF
```

### 37.4.3 List Information

The List Information program provides information about each Hexiwear device. For each device, a device.As("Hexiwear") specialization is created. The program then prints the device name, battery power level, manufacturer name and firmware revision. To see all of kinds of data you can read from a Hexiwear device, look at the ReadAll program. It prints all of the data that a Hexiwear is capable of producing.

```
CLS BLUE
PRINT "Available Bluetooth devices"

devices = Bluetooth.DevicesName ("HEXIWEAR")

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PRINT "NAME", device.Name
    tag = device.As("Hexiwear")
    PRINT tag.GetName()
    PRINT tag.GetPower()
    PRINT tag.GetManufacturerName()
    REM does not work. PRINT tag.GetHardwareRevision()
    PRINT tag.GetFirmwareRevision()

NEXT i
```

```
n = devices.Count  
PRINT ""  
PRINT "" + n + " devices were found"
```

### 37.4.4 Raw Access to Hexiwear

The Raw Access program demonstrates how you can get information from a Hexiwear device without using the specialization. To do this, you will need know the different Bluetooth services and characteristics that a Hexiwear device exposes and how to read the resulting data. This documentation is available at

<https://www.dropbox.com/s/92tphuymsv0n5kx/HEXIWEAR%20Bluetooth%20Specifications.pdf?dl=0> In this program the accelerometer data is read. The acceleration service is server "2000" and the acceleration data is characteristic "2001". The data is read using the ReadRawBytes() method; that method returns an array of 6 bytes. The array starts at index 1. The bytes of the array must be interpreted as 3 16-bit integers. To interpret the bytes, you can use the built-in GetValue() method on the data array; that method takes in two parameters. The first parameter is the index to start reading at and the second is the interpretation type. Use "int16-le" to interpret the data as a 16-bit signed integer, little endian.

```
CLS BLUE  
ACCSERVICE ="2000"  
ACCDATA ="2001"  
  
device = Bluetooth.PickDevicesName("HEXIWEAR")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    PRINT "DEVICE", device.Name  
  
    address= device.Init()  
    PRINT "Address", address
```

```
REM Don't have to tell the device to turn on accelerometer
```

```
PRINT "X", "Y", "Z"  
FOR time = 1 TO 15  
    PAUSE 50  
    data = device.ReadRawBytes(ACCSERVICE, ACCDATA)  
    x = data.GetValue (1, "int16-le") / 100  
    y = data.GetValue(3, "int16-le") / 100  
    z = data.GetValue(5, "int16-le") / 100  
    PRINT x, y, z  
NEXT time  
  
PRINT "Done"  
END IF
```

### 37.4.5 Read All

The Read All program demonstrates all of the different sensors in the Hexiwear IOT device. In the program, all of the Hexiwear devices are listed and a specialization created. Then the Hexiwear mode is read. Depending on the mode, the heart rate, pedometer or sensor data will be printed.

```
CLS BLUE  
PRINT AT 12,1 "Demonstrate Hexiwear sensors"  
  
device = Bluetooth.PickDeviceName("HEXIWEAR")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    tag = device.As("Hexiwear")  
    PRINT AT 1,1 "Got a device", device.Name  
  
  
REM Now poll for data.  
FOR time = 1 TO 4
```

```
Screen.ClearLine(2)
PRINT AT 2, 1 "TIME", time

mode = tag.GetMode()
PRINT AT 3,1 "MODE", mode

IF (mode = 2) THEN ShowSensors(tag)
IF (mode = 5) THEN ShowHeart(tag)
IF (mode = 6) THEN ShowPedometer(tag)

PAUSE 50
NEXT time

PRINT AT 11, 1 "FINISH", status
tag.Close()
END IF

FUNCTION ShowHeart(tag)
Screen.ClearLine(4)
PRINT AT 4,1 "Heart", tag.GetHeart()
END

FUNCTION ShowPedometer(tag)
Screen.ClearLine(4)
Screen.ClearLine(5)
PRINT AT 4,1 "Steps", tag.GetSteps()
PRINT AT 5,1 "Calorie", tag.GetCalories()
END

FUNCTION ShowSensors(tag)
value = tag.GetAccelerometer()
Screen.ClearLine(4)
PRINT AT 4,1 "Accel.", "" + value.X + " " + value.Y + " " + value.Z

value = tag.GetGyroscope()
Screen.ClearLine(5)
PRINT AT 5,1 "Gyro.", "" + value.X + " " + value.Y + " " + value.Z
```

```
value = tag.GetMagnetometer()
Screen.ClearLine(6)
PRINT AT 6,1 "Mag.", "" + value.X + " " + value.Y + " " + value.Z

Screen.ClearLine(7)
PRINT AT 7,1 "Temp", tag.GetTemperature()
Screen.ClearLine(8)
PRINT AT 8,1 "Humidity", tag.GetHumidity()
Screen.ClearLine(9)
PRINT AT 9,1 "Pressure", tag.GetPressure()
Screen.ClearLine(10)
PRINT AT 10,1 "Light", tag.GetLight()

END
```

### 37.4.6 Set notification count

A new program for you to edit

```
CLS BLUE
PRINT "Demonstrate Hexiwear SetNotificationCount"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT "Got a device", device.Name

    REM Now set the notification
    REM 2=missed call 4=social 6=email
    REM second value is the count to set.
    status = tag.SetNotificationCount(6, 17)
    PRINT ". status", status
END IF
```

## 37.4.7 SetTime

A new program for you to edit

```
CLS BLUE
PRINT "Demonstrate Hexiwear SetTimeNow"

device = Bluetooth.PickDevicesName("HEXIWEAR")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("Hexiwear")
    PRINT "Got a device", device.Name

    REM Now set the time
    status = tag.SetTimeNow()
    PRINT ". status", status
END IF
```

## 37.5 BT: MAGICLIGHT

Supports the MagicLight and Flux lights. These are Bluetooth-enabled light bulbs for home use.

### 37.5.1 Green

Turns the device green

```
CLS BLUE
PRINT "Sets the light to green"

device = Bluetooth.PickDevicesName("LEDBlue*")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    light = device.As ("MagicLight")
```

```
REM The three parameters are Red, Green and Blue values.  
REM They must be in the range 0 to 255  
Status = light.SetColor (0, 255, 0)  
PRINT "status", Status  
END IF
```

### 37.5.2 Off

Turns the device off

```
CLS BLUE  
PRINT "Turns the light off"  
  
device = Bluetooth.PickDevicesName("LEDBlue*")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    light = device.As ("MagicLight")  
    Status = light.SetOff ()  
    PRINT "status", Status  
ENDIF
```

### 37.5.3 On

Turns the device on

```
CLS BLUE  
PRINT "Turns the light on"  
  
device = Bluetooth.PickDevicesName("LEDBlue*")  
IF (device.IsError)  
    PRINT "No device was picked"  
ELSE  
    light = device.As ("MagicLight")  
    Status = light.SetOn ()  
    PRINT "status", Status  
ENDIF
```

### 37.5.4 Red

Turns the light red

```
CLS BLUE
PRINT "Sets the light to red"

device = Bluetooth.PickDevicesName("LEDBlue*")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    light = device.As ("MagicLight")
    REM The three parameters are Red, Green and Blue values.
    REM They must be in the range 0 to 255
    Status = light.SetColor (255, 0, 0)
    PRINT "status", Status
END IF
```

## 37.6 BT: NOTTI

Demonstrates how to control the NOTTI device (from Witti design company). The NOTTI device is a desktop device with a single light that can be set to any color. You can also program transitions and for colors changes to happen at a time in the future.

### 37.6.1 An introduction

An introduction to using the NOTTI specialization

```
CLS BLUE
PRINT "An introduction to the NOTTI specialization"

devices = Bluetooth.DevicesName ("*Notti")

FOR i = 1 TO devices.Count
    device = devices.Get(i)
    PRINT "Notti Device"
```

```
PRINT "Name", device.Name
Notti = device.As ("NOTTI")
PRINT "Methods", Notti.Methods
PRINT "GetName()", Notti.GetName()
PRINT "GetPower()", Notti.GetPower()
PRINT " "
NEXT i
```

### 37.6.2 Change Mode

Changes the mode of the NOTTI device (clock, animation, etc)

```
CLS BLUE
PRINT "Set NOTTI mode"
PRINT "0=Light on"
PRINT "1=Light off"
PRINT "2=Full animation"

device = Bluetooth.PickDevicesName("*Notti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
    mode = INPUT DEFAULT 1 PROMPT "What mode?"
    Status = Notti.ChangeMode(mode)
    PRINT "status", Status
END IF
```

### 37.6.3 NOTTI timer

Slowly switches from GREEN to RED over (n) minutes

```
device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    notti = device.As ("NOTTI")
```

```
PRINT "Power", notti.GetPower()

nminutes = Calculator.Value
nminutes = INPUT DEFAULT nminutes PROMPT "How many minutes
to run for?"
END IF
STOP nminutes

FUNCTION DoTimer(device)
rstart = 0
rend = 255
gstart = 255
gend = 0
bstart = 0
bend = 0

nottiSetColor (rstart, gstart, bstart)
PRINT notti
Screen.RequestActive()
tot = nminutes*60
FOR s = 0 TO tot
    pct = s / tot
    r = pct*(rend - rstart) + rstart
    g = pct*(gend - gstart) + gstart
    b= pct*(bend - bstart) + bstart
    IF (Math.Mod(s, 60)= 0) THEN PRINT s/60
    Screen.ClearLine (8)
    PRINT AT 8,1 "Set color", tot-s,INT(pct*1000)/10
    nottiSetColor(r, g, b)
    PAUSE 50
NEXT s
PRINT "done!"
nottiSetColor(0, 0, 255)
Screen.RequestRelease()
RETURN
```

### 37.6.4 Raw Bluetooth commands

Sets the NOTTI color to red using the raw Bluetooth commands

```
CLS BLUE
PRINT "Set a NOTTI device to RED"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    WriteColor(device, 255, 0, 0)
END IF

FUNCTION WriteColor(bt, r, g, b)
    bt.Init()
    REM The fff0 is the service for many NOTTI commands
    REM The fff3 is the characteristic used by service fff0
    REM   for many of the NOTTI commands
    REM the 6 and 1 are the bytes that define the NOTTI
    REM command to send (0x0601 means set LED color)
    REM the r g and b are the color to set.
    bt.WriteBytes ("fff0", "fff3", 6, 1, r, g, b)
END
```

### 37.6.5 Set Alarm

Sets the Alarm on the NOTTI device

```
CLS BLUE
PRINT "ALARM -- set NOTTI alarm"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
```

```
PRINT "NOTTI", Notti
h = INPUT DEFAULT 10 PROMPT "What hour?"
m = INPUT DEFAULT 10 PROMPT "What minute?"
ahead = INPUT DEFAULT 1 PROMPT "How far ahead 1=2.5 minutes"

Status = Notti.SetAlarmTime(h, m)
REM the 2 means it's a one-time alarm (0=off 1=every day)
REM 255, 0, 0 is RED (rgb color)
REM 1 means start up 2.5 minutes ahead of time (1=2.5 minute
10=25 minutes)

PRINT "AT " + h + ":" + m + " turn to RED"
PRINT "START " + (ahead*2.5) + " minutes ahead of that time"
Status = Notti.AlarmSetting (2, 255, 0, 0, ahead)
PRINT "status", Status
END IF
```

### 37.6.6 SetColor to change the NOTTI color to blue

Sets the NOTTI color to Blue

```
CLS BLUE
PRINT "Sets a NOTTI to blue"

device = Bluetooth.PickDevicesName("*Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
    REM The three parameters are Red, Green and Blue values.
    REM They must be in the range 0 to 255
    Status = NottiSetColor (0, 0, 255)
    PRINT "status", Status
END IF
```

### 37.6.7 SetColor to change the NOTTI color to green

Sets the NOTTI color to Blue

```
CLS BLUE
PRINT "Sets a NOTTI to green"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
    REM The three parameters are Red, Green and Blue values.
    REM They must be in the range 0 to 255
    Status = Notti.SetColor (0, 255, 0)
    PRINT "status", Status
END IF
```

### 37.6.8 SetColorCustom animates the color from red to blue and back again

Uses the SetColorCustom command to change colors

```
CLS BLUE
PRINT "Sets a NOTTI from RED to BLUE and back again"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
    REM The three parameters are Red, Green and Blue values.
    REM They must be in the range 0 to 255
    Status = NottiSetColorCustom (255, 0, 0, 0 , 0, 255)
    PRINT "status", Status
END IF
```

### 37.6.9 SetName of a NOTTI device

Sets the name of a NOTTI device. The device must be reset and re-paired for Windows to use the new name.

```
CLS BLUE
PRINT "Change the name of a NOTTI device"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    PRINT device.Name
    Notti = device.As ("NOTTI")
    Status = Notti.SetName ("My-Notti")
    PRINT "status", Status
END IF
```

### 37.6.10 Sync Time

Sets the time on the NOTTI device

```
CLS BLUE
PRINT "SyncTime -- set NOTTI time"

device = Bluetooth.PickDevicesName("Dotti")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    Notti = device.As ("NOTTI")
    PRINT "NOTTI", Notti
    h = INPUT DEFAULT 10 PROMPT "What hour?"
    m = INPUT DEFAULT 10 PROMPT "What minute?"
    s = INPUT DEFAULT 15 PROMPT "What second?"
    Status = Notti.SyncTime(h, m, s)
    PRINT "status", Status
END IF
```

## 37.7 BT: SENSORTAG 2541

Demonstrates how to use the TI SensorTag 2541 (the original version). The model 2541 SensorTag from Texas Instruments is a small, battery-powered sensor platform from TI. The sensors include an accelerometer, gyroscope, IR contactless thermometer, humidity sensor, magnetometer, barometer and on-chip temperature sensor.

### 37.7.1 Accelerometer

Demonstrates the basics of the AccelerometerSetup and using a callback routine.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate TI SensorTag Accelerometer"

device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
    PRINT AT 6,1 "Got a device", device.Name
    REM 1=turn on the on-device accelerometer
    REM 20=accelerometer update speed (in milliseconds)
    PRINT AT 7,1 "SETUP", tag.AccelerometerSetup(1, 20, "Acc")

    PRINT AT 8,1 "Done with setup"

    REM Now wait a little while. The Acc routine will
    REM be called with updates.
    FOR time = 1 TO 10
        Screen.ClearLine(3)
        PRINT "TIME", time
        PAUSE 50
    NEXT time
    REM Undo the accelerometer
    status = tag.AccelerometerSetup(0, 0, "")
    PRINT AT 9, 1 "FINISH", status
```

```
    tag.Close()
```

```
END IF
```

```
FUNCTION Acc(tag, x, y, z)
    Screen.ClearLine(1)
    PRINT x, y, z
END
```

### 37.7.2 Accelerometer Off

Turns off the accelerometer

```
CLS BLUE
```

```
device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
    PRINT "Got a device", device.Name
    tag.SetupAcc(0, 100, "Acc")
ENDIF
PRINT "All done"
```

### 37.7.3 Accelerometer to Magic Light

Uses the TI SensorTag accelerometer to drive a Magic Light

```
CLS BLUE
```

```
PRINT "Demonstrate TI SensorTag Accelerometer"
```

```
lights = Bluetooth.DevicesName ("LEDBlue*")
FOR i=1 TO lights
    device = lights.Get(i)
    light = device.As("MagicLight")
NEXT i
```

```
device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
    PRINT "Got a device", device.Name
    PRINT tag.SetupAcc(1, 20, "Acc")

    PRINT "Done with setup"
    FOR time = 1 TO 20
        PRINT AT 10, 1 time
        PAUSE 50
    NEXT time
    PRINT tag.SetupAcc(0, 0, "")
    tag.Close()
END IF
PRINT "DONE"
```

REM The tag.SetupAcc(1, 20, "Acc") call tells the device to call the "Acc"

REM function when the accelerometer changes.

```
FUNCTION Acc(tag, x, y, z)
```

REM Tell the Acc function that the "light" variable is really a  
REM global variable defined in the main program.  
GLOBAL light

```
Screen.ClearLine(1)
```

```
PRINT x, y, z
```

```
r = ABS (x*120)
```

```
g = ABS(y*120)
```

```
b = ABS (z*120)
```

```
Screen.ClearLine(2)
```

```
PRINT r, g, b
```

```
light.SetColor (r, g, b)
```

**END**

### 37.7.4 Barometer

Demonstrates the basics of the BarometerSetup and using a callback routine.

```
devices = Bluetooth.DevicesName ("SensorTag")

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Barometer measurements"

device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
    REM 100=1000ms=1 second
    PRINT AT 7,1 "SETUP", tag.BarometerSetup(1, 100, "Barometer")

    FOR time = 1 TO 30
        PAUSE 50
        PRINT AT 3,1 "TIME", time
    NEXT time

    PRINT AT 8,1 "CLOSE", tag.BarometerSetup(0, 100, "Barometer")
END IF

REM Temperatures are in degrees C
REM pressure is in hpa
FUNCTION Barometer(tag, temp, pressure)
    Screen.ClearLine(1)
    PRINT temp, pressure

    Screen.ClearLine(2)
    PRINT CTOF(temp), HPATOINCHM(pressure)
END
```

```
FUNCTION CTOF(C)
```

```
    F = C * 9/5 + 32
```

```
    F = Math.Round(F*10) / 10
```

```
END F
```

```
FUNCTION HPATOINCHM(HPA)
```

```
    ATM = HPA / 1013.25
```

```
    INCHM = ATM * 29.9213
```

```
    REM INCHM = Math.Round(INCHM*10) / 10
```

```
END INCHM
```

```
FUNCTION ROUND(val)
```

```
    val = Math.Round(val*10) / 10
```

```
END val
```

### 37.7.5 Button

Demonstrates the "Simple Key Service" on the SensorTag

```
devices = Bluetooth.DevicesName ("SensorTag")
```

```
CLS BLUE
```

```
PRINT AT 5,1 "Demonstrate SensorTag Buttons"
```

```
PRINT AT 6,1 "Count", devices.Count
```

```
device = Bluetooth.PickDevicesName("SensorTag")
```

```
IF (device.IsError)
```

```
    PRINT "No device was picked"
```

```
ELSE
```

```
    tag = device.As("SensorTag2541")
```

```
    PRINT AT 7,1 "SETUP", tag.ButtonSetup(1, "Button")
```

```
FOR time = 1 TO 30
```

```
    PAUSE 50
```

```
    PRINT AT 3,1 "TIME", time
```

```
NEXT time
```

```
PRINT AT 8,1 "CLOSE", tag.ButtonSetup(0, "Button")
END IF

FUNCTION Button(tag, left, right, side)
Screen.ClearLine(1)
IF (left) THEN PRINT AT 1,1 "LEFT"
IF (right) THEN PRINT AT 1,8 "RIGHT"
IF (side) THEN PRINT AT 1,16 "SIDE"
END
```

### 37.7.6 Gyroscope

Demonstrates the basics of the GyroscopeSetup and using a callback routine.

```
CLS BLUE
PRINT AT 5,1 "Demonstrate TI SensorTag Gyroscope"

devices = Bluetooth.DevicesName ("SensorTag")

device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
    PRINT AT 6,1 "Got a device", device.Name
    REM 7=turn on all axis of the gyroscope
    REM 20=accelerometer update speed (in milliseconds)
    PRINT AT 7,1 "SETUP", tag.GyroscopeSetup(7, 20, "Gyroscope")

    PRINT AT 8,1 "Done with setup"

    REM Now wait a little while. The Acc routine will
    REM be called with updates.
    FOR time = 1 TO 10
        PRINT AT 3, 1 "TIME", time
```

```
PAUSE 50
NEXT time
REM Undo the accelerometer
status = tag.GyroscopeSetup(0, 20, "Gyroscope")
PRINT AT 9, 1 "FINISH", status
tag.Close()
END IF

FUNCTION Gyroscope(tag, x, y, z)
Screen.ClearLine(1)
PRINT ROUND(x), ROUND(y), ROUND(z)
END

FUNCTION ROUND(val)
val = Math.Round(val*10) / 10
END val
```

### 37.7.7 Humidity

Demonstrates the basics of the HumiditySetup and using a callback routine.

```
devices = Bluetooth.DevicesName ("SensorTag")

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Humidity measurements"

FOR i=1 TO devices.Count
device = devices.Get(i)
tag = device.As("SensorTag2541")
REM 100=1000ms=1 second
PRINT AT 7,1 "SETUP", tag.HumiditySetup(1, 100, "Humidity")

FOR time = 1 TO 30
PAUSE 50
PRINT AT 3,1 "TIME", time
NEXT time
```

```
PRINT AT 8,1 "CLOSE", tag.HumiditySetup(0, 100, "Humidity")
NEXT i

REM Temperatures are in degrees C
FUNCTION Humidity(tag, temp, humidity)
    Screen.ClearLine(1)
    PRINT temp, humidity

    Screen.ClearLine(2)
    PRINT CTOF(temp), ROUND(humidity)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F

FUNCTION ROUND(val)
    val = Math.Round(val*10) / 10
END val
```

### 37.7.8 IR

Demonstrates the basics of the IRSetup and using a callback routine.

```
devices = Bluetooth.DevicesName ("SensorTag")

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag IR measurements"
PRINT AT 6,1 "Count", devices.Count

device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
    PRINT "No device was picked"
ELSE
    tag = device.As("SensorTag2541")
```

```
REM 100=1000ms=1 second
PRINT AT 7,1 "SETUP", tag.IRSetup(1, 100, "IR")

FOR time = 1 TO 30
    PAUSE 50
    PRINT AT 3,1 "TIME", time
NEXT time

PRINT AT 8,1 "CLOSE", tag.IRSetup(0, 100, "IR")
END IF

REM Temperatures are in degrees C
FUNCTION IR(tag, objTemp, ambTemp)
    Screen.ClearLine(1)
    PRINT objTemp, ambTemp

    Screen.ClearLine(2)
    PRINT CTOF(objTemp), CTOF(ambTemp)
END

FUNCTION CTOF(C)
    F = C * 9/5 + 32
    F = Math.Round(F*10) / 10
END F
```

### 37.7.9 Magnetometer

Demonstrates the basics of the MagnetometerSetup and using a callback routine.

```
devices = Bluetooth.DevicesName ("SensorTag")

CLS BLUE
PRINT AT 5,1 "Demonstrate SensorTag Magnetometer measurements"

device = Bluetooth.PickDevicesName("SensorTag")
IF (device.IsError)
```

```
PRINT "No device was picked"  
ELSE  
    tag = device.As("SensorTag2541")  
    REM 100=1000ms=1 second  
    PRINT AT 7,1 "SETUP", tag.MagnetometerSetup(1, 100,  
    "Magnetometer")  
  
    FOR time = 1 TO 30  
        PAUSE 50  
        PRINT AT 3,1 "TIME", time  
        NEXT time  
  
    PRINT AT 8,1 "CLOSE", tag.MagnetometerSetup(0, 100,  
    "Magnetometer")  
END IF  
  
FUNCTION Magnetometer(tag, x, y, z)  
    Screen.ClearLine(1)  
    PRINT ROUND(x), ROUND(y), ROUND(z)  
END  
  
FUNCTION ROUND(val)  
    val = Math.Round(val*10) / 10  
END val
```

### 37.7.10 Raw access to SensorTag

Shows how to access the SensorTag using just the low-level Bluetooth functions and without using the specialization

```
CLS BLUE  
devices = Bluetooth.DevicesName("SensorTag*")  
ACCSERVICE ="f000aa10-0451-4000-b000-000000000000"  
ACCCONFIG ="f000aa12-0451-4000-b000-000000000000"  
ACCDATA ="f000aa11-0451-4000-b000-000000000000"
```

```
PRINT "COUNT", devices.Count

FOR i=1 TO devices.Count
    device = devices.Get(i)
    PRINT "DEVICE", device.Name

    address= device.Init()
    PRINT "Address", address

    REM Tell the device to send me some acc. data
    PRINT "ACC ON", device.WriteBytes(ACCSERVICE, ACCCONFIG, 1)

    PRINT "COUNT", "X", "Y", "Z"
    FOR time = 1 TO 15
        PAUSE 50
        data = device.ReadRawBytes(ACCSERVICE, ACCDATA)
        PRINT data.Count, data.Get(1), data.Get(2), data.Get(3)
    NEXT time

    REM Turn it back off
    PRINT "ACC OFF", device.WriteBytes(ACCSERVICE, ACCCONFIG, 0)
    PRINT " "
NEXT i
```

### 37.7.11 Raw callback with the SensorTag

A new program for you to edit

```
CLS BLUE
PRINT AT 5,1 "Acceleration Data"

devices = Bluetooth.DevicesName ("SensorTag*")

REM
REM Constants for TI SensorTag 2541 Accelerometer
REM These are taken from the data sheets.
```

REM

AccService = "f000aa10-0451-4000-b000-000000000000"

AccData = "f000aa11-0451-4000-b000-000000000000"

AccConfig = "f000aa12-0451-4000-b000-000000000000"

AccPeriod = "f000aa13-0451-4000-b000-000000000000"

PRINT "COUNT", devices.Count

IF devices.Count < 1 THEN STOP

device = devices.Get(1)

PRINT "SensorTag Address", device.Init()

REM Tell the SensorTag to enable the Accelerometer

REM Config=1 means enable

REM Period=20 means get data fast (50 per second)

device.WriteBytes(AccService, AccConfig, 1)

device.WriteBytes(AccService, AccPeriod, 100)

REM 1=Notify (2=Indicate 0=None)

device.WriteCallbackDescriptor (AccService, AccData, 1)

device.AddCallback (AccService, AccData, "WriteAcc")

REM

REM Wait a little while and then turn off the Accelerometer

REM

FOR time = 1 TO 10

PAUSE 50

PRINT AT 1,1 time

PAUSE 50

PRINT AT 1,1 " (clear)"

NEXT time

REM

REM Turn off the accelerometer; turn off notify; remove callback

**REM**

```
device.WriteCallbackDescriptor (AccService, AccData, 0)
device.WriteBytes(AccService, AccConfig, 0)
device.RemoveCallback (AccService, AccData, "WriteAcc")
```

```
FUNCTION WriteAcc(device, x, y, z)
    Screen.ClearLine(3)
    PRINT x, y, z
END
```

## 37.8 EX: BC BASIC QUICK SAMPLES

A set of the most common programs people need. Includes a tip program, money conversion, miles per gallon, and more.

### 37.8.1 Colorful Countdown

A bright countdown display. Will count down for the number of seconds in the calculator window. The minimum countdown is 5 seconds, and the maximum is 60 seconds.

```
value = Calculator.Value
IF value < 5 THEN value = 5
IF value > 60 THEN value = 60

PRINT "Count down!"
```

```
REM anitime sets the speed of the color changes
REM when set low (like 5), the colors really flash quickly
REM when set to 50, the color changes with the display
anitime = 25
```

```
ctime = 0
display = value
FOR i=0 TO (value*50) STEP anitime
color = color + 1
IF color >= 7 THEN color = 1
CLS color
```

```
PrintTitle (1, "Colorful Countdown")
PrintCenter (display)
PAUSE anitime
ctime = ctime + anitime
IF ctime < 50 THEN NEXT i
ctime = ctime - 50
display = display - 1
NEXT i
PrintCenter ("Countdown Complete")

FUNCTION PrintTitle(row, str)
lmargin = 1+INT (( Screen.W - LEN str) / 2)
IF (lmargin < 1) THEN lmargin = 1
PRINT AT row,lmargin str
RETURN

FUNCTION PrintCenter (str)
lmargin = 1+INT (( Screen.W - LEN str) / 2)
IF (lmargin < 1) THEN lmargin = 1
row = INT ((Screen.H) / 2)
PRINT AT row,lmargin str
RETURN
```

### 37.8.2 Grams of Fat to Calories

Takes the value already in the calculator and converts it from grams of fat to calories. This program assumes that all fat is 9 calories per gram.

```
value = Calculator.Value
retval=value * 9
Calculator.Message = "Converted " + value + " grams of fat to
calories"

CLS BLUE
PRINT "Convert Grams of Fat to Calories"
PRINT "Input="; value
PRINT " "
```

```
PRINT "There are 9 calorie per gram of fat"  
PRINT ""  
PRINT "Calories="; retval  
STOP retval
```

### 37.8.3 Miles per Gallon

Calculates mile per gallon given number of miles driven and total gallons of gas.

```
CLS  
PRINT "Calculating Miles Per Gallon"  
PRINT ""  
miles = INPUT DEFAULT 100 PROMPT "How many miles were driven?"  
gallons = INPUT DEFAULT 4 PROMPT "How many gallons did you  
need?"  
retval = mpg(miles, gallons)  
  
PRINT "Miles driven="; miles  
PRINT "Gallons used="; gallons  
PRINT "MPG="; retval  
  
IF NOT Memory.IsSet ("PreviousMpg") THEN GOTO 40  
lastMpg = Memory.PreviousMpg  
deltaMpg = retval - lastMpg  
PRINT "Last time="; lastMpg  
IF ABS (deltaMpg) > 1.5 THEN GOTO 10  
PAPER GREEN  
PRINT "MPG is about the same"  
GOTO 40  
  
10 IF deltaMpg >= 1.5 THEN GOTO 20  
  
PAPER RED  
PRINT "MPG has decreased!"  
GOTO 40
```

```
20 PAPER GREEN
```

```
PRINT "MPG has increased!"
```

```
GOTO 40
```

```
40 REM
```

```
Memory.PreviousMpg = retval
```

```
STOP retval
```

```
FUNCTION mpg (Miles, Gallons)
```

```
Retval = Miles / Gallons
```

```
RETURN Retval
```

### 37.8.4 Right Triangle calculator

Uses the Pythagorean theorem to calculate the hypotenuse of a right triangle based on the other two sides.

```
REM Calculate the hypotenuse of a triangle
```

```
CLS BLUE
```

```
PRINT "Right triangle calculator"
```

```
PRINT " "
```

```
A = INPUT DEFAULT 3 PROMPT "Enter the first side"
```

```
B = INPUT DEFAULT 4 PROMPT "Enter the second side"
```

```
C = hypotenuse (A, B)
```

```
PRINT "First side=";A
```

```
PRINT "Second side=";B
```

```
PRINT "Hypotenuse=";C
```

```
PRINT " "
```

```
PRINT "Calculation is √(A**2 + B**2)"
```

```
STOP C
```

```
REM Calculate the hypotenuse from A and B
```

```
FUNCTION hypotenuse (A, B)
```

```
C=2 √(A**2 + B**2)
```

```
RETURN C
```

## 37.8.5 Tip Calculator

A new program for you to edit

```
value = Calculator.Value  
CLS GREEN  
  
TEST()  
PRINT "Tip Calculator"  
PRINT " "  
PRINT " 5% tip of "; value; " is "; Tip(value, 5)  
PRINT " 10% tip of "; value; " is "; Tip(value, 10)  
PRINT " 15% tip of "; value; " is "; Tip(value, 15)  
PRINT " 18% tip of "; value; " is "; Tip(value, 18)  
PRINT " 20% tip of "; value; " is "; Tip(value, 20)  
PRINT " "  
Calculator.Message = "15% tip of " + value + " is " + Tip(value, 15)  
STOP 0+Tip(value, 15)
```

REM We need a fancy function because we need to format the number  
REM nice and neat. It should be calculated to the nearest penny  
exactly.

```
FUNCTION Tip(value, percent)  
raw = value * (percent/100)  
round = Math.Round (raw * 100) / 100  
fraction = round - Math.Truncate(round)  
fraction = Math.Round (fraction * 100)  
IF (fraction < 10) THEN fraction = "0" + fraction  
top = "" + Math.Truncate(round) + "." + fraction  
RETURN top
```

```
FUNCTION TestOne (value, percent, expected)  
actual = Tip (value, percent)  
IF (actual ≈ expected) THEN RETURN 0  
PRINT "ERROR; TIP ";value; " pct "; percent  
PRINT "Expected "; expected  
PRINT "Actual "; actual
```

```
PRINT "Difference "; actual-expected
```

```
RETURN 1
```

```
FUNCTION TEST ()
```

```
nerror = 0
```

```
REM Tip returns a string, not a number
```

```
nerror = nerror + TestOne (100, 5, "5.00")
```

```
nerror = nerror + TestOne (76, 15, "11.40")
```

```
nerror = nerror + TestOne (140, 15, "21.00")
```

```
IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror
```

```
IF (nerror > 0) THEN PAPER RED
```

```
RETURN nerror
```

## 37.8.6 Welcome to BC BASIC

Describes BC BASIC for new users

```
CLS BLUE
```

```
PRINT "WELCOME TO BC BASIC!"
```

```
PRINT " "
```

```
PRINT "You can program the P1 to P5 keys"
```

```
PRINT "to perform ANY function you want"
```

```
PRINT "using Best Calculator BASIC"
```

```
PRINT " "
```

```
PRINT "Tap the BC BASIC button to get started"
```

```
PRINT " – there is full help available"
```

```
PRINT " – there are lots of samples"
```

```
PRINT " – you can get started right away"
```

## 37.9 EX: FINANCIAL

Sample financial programs for ROI (Return on Investment), Present and Future value, and more.

### 37.9.1 Common Tip Values

Starts with the value already in your calculator, and comes up with a range of tips (5%, 10%, 15%, 18%, 20%)

```
value = Calculator.Value
CLS BLACK
TEST()
PRINT "5% tip of "; value; " is "; Tip(value, 5)
PRINT "10% tip of "; value; " is "; Tip(value, 10)
PRINT "15% tip of "; value; " is "; Tip(value, 15)
PRINT "18% tip of "; value; " is "; Tip(value, 18)
PRINT "20% tip of "; value; " is "; Tip(value, 20)

Calculator.Message = "15% tip of " + value + " is " + Tip(value, 15)
STOP 0+Tip(value, 15)

REM We need a fancy function because we need to format the number
REM nice and neat. It should be calculated to the nearest penny
exactly.

FUNCTION Tip(value, percent)
raw = value * (percent/100)
round = Math.Round (raw * 100) / 100
fraction = round - Math.Truncate(round)
fraction = Math.Round (fraction * 100)
IF (fraction < 10) THEN fraction = "0" + fraction
top = "" + Math.Truncate(round) + "." + fraction
RETURN top

FUNCTION TestOne (value, percent, expected)
actual = Tip (value, percent)
IF (actual ≈ expected) THEN RETURN 0
PRINT "ERROR; TIP ";value; " pct "; percent
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1

FUNCTION TEST ()
nerror = 0
```

```
REM Tip returns a string, not a number
```

```
nerror = nerror + TestOne (100, 5, "5.00")
```

```
nerror = nerror + TestOne (76, 15, "11.40")
```

```
nerror = nerror + TestOne (140, 15, "21.00")
```

```
IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror
```

```
IF (nerror > 0) THEN PAPER RED
```

```
RETURN nerror
```

### 37.9.2 Compound Interest

Calculates compound interest

```
REM
```

```
REM Calculates the interest earned on a loan.
```

```
REM Loan terms and interest is given per year; the
```

```
REM interest is compounded monthly.
```

```
REM
```

```
TEST()
```

```
P = INPUT DEFAULT 1000 PROMPT "Principal (original balance)"
```

```
RY = INPUT DEFAULT 12 PROMPT "Rate per year (enter 3% as 3)"
```

```
NY = INPUT DEFAULT 1 PROMPT "Number of years"
```

```
C = CompoundInterest(P, RY, NY)
```

```
Calculator.Message = "Compound interest earned"
```

```
STOP C
```

```
FUNCTION CompoundInterest(P, RY, NY)
```

```
R = RY / 1200
```

```
N = NY * 12
```

```
C = P * ( (1 + R)**N - 1 )
```

```
RETURN C
```

```
FUNCTION TestOne (P, RY, NY, expected)
```

```
actual = CompoundInterest(P, RY, NY)
```

```
actual = Math.Round (actual * 100) / 100
```

```
IF (actual ≈ expected) THEN RETURN 0
```

```
PRINT "ERROR; P ";P; " RY "; RY
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1

FUNCTION TEST ()
nerror = 0
nerror = nerror + TestOne (1200, 12.49, .5, 76.92)
nerror = nerror + TestOne (1100, 3.2, 2, 72.60)

IF (nerror > 0) THEN PRINT "CompoundInterest NERROR=";nerror
IF (nerror > 0) THEN PAPER RED
RETURN nerror
```

### 37.9.3 Doubling Time

Calculate the time it takes to double an investment given a rate of return. The rate of return is in percent; 12% is represented as 12.

```
REM Takes the value in the calculator as a percent (e.g., 12% is 12)
REM divides by 100 to get computer-type percents (0.12)
REM values are in YEARS, but interest is assumed to compound
MONTHLY.
```

```
REM Example: at a rate of "6" (6%), money will double in about 11.58
years.
```

```
CLS BLACK
TEST()
```

```
yr = Calculator.Value
dt = DoublingTime(yr)
Calculator.Message = "Doubling Time in years at " + yr + "% per year"
STOP dt
```

REM Doubling time in years given a per-year interest rate that compounds monthly

REM yr is percent interest rate; e.g., give 4.25% as 4.25

FUNCTION DoublingTime(yr)

r = yr / 100

REM divide by 12 to get the monthly rate

r = r / 12

dt = LN(2) / LN(1 + r)

REM dt starts off in months, but we want to present years, so divide by 12.

dt = dt / 12

Calculator.Message = "Doubling Time in years at " + yr + "% per year"

RETURN dt

FUNCTION TestOne (yr, expected)

actual = DoublingTime(yr)

actual = Math.Round (actual \* 100) / 100

IF (actual ≈ expected) THEN RETURN 0

PRINT "ERROR; DoublingTime "; yr

PRINT "Expected "; expected

PRINT "Actual "; actual

PRINT "Difference "; actual-expected

RETURN 1

FUNCTION TEST ()

nerror = 0

nerror = nerror + TestOne (6, 11.58)

nerror = nerror + TestOne (6.35, 10.94)

IF (nerror > 0) THEN PRINT "DoublingTime NERROR=";nerror

IF (nerror > 0) THEN PAPER RED

RETURN nerror

### 37.9.4 Future Value

Calculates the future value of money today given a period of time and an interest rate.

```
REM Future value uses the standard formula FV = PV * (1+r)**n  
REM where PV = present value (e.g., money to invest)  
REM r is the interest rate (3% is .03 in the formula, but this program  
lets the user enter '3' for 3%  
REM n is the number of periods. This must match the interest rate  
(e.g., either both are 'per year' or both are 'per month')
```

```
REM Good practice to run the TEST program to make sure the  
calculations are OK.
```

```
TEST()
```

```
PV = INPUT DEFAULT 900 PROMPT "Present value"  
n = INPUT DEFAULT 3 PROMPT "How far into the future (years)"  
r = INPUT DEFAULT 3 PROMPT "Interest rate (per year). Enter 3 for  
3%"  
r = r / 100  
FV = FutureValue(PV, n, r)  
Calculator.Message = "FV of " + PV + " at " + r*100 + "%"  
STOP FV
```

```
FUNCTION FutureValue(PV, n, r)  
FV = PV * ((1+r)**n)  
RETURN FV
```

```
FUNCTION TestOne(PV, n, r, expected)  
nerror = 0  
actual = FutureValue(PV, n, r)  
IF actual ≈ expected THEN GOTO 10  
nerror = nerror + 1  
PRINT "ERROR: FutureValue"  
PRINT "PV=";PV; " n="; n; " r=";r  
PRINT "expected="; expected  
PRINT "actual=";actual  
10 REM  
RETURN nerror
```

```
FUNCTION TEST()
```

```
nerror = 0  
nerror = nerror + TestOne (1000, 5, .1, 1610.51)  
RETURN nerror
```

### 37.9.5 Money Conversion

Simple program to convert from one currency to another. The program will always prompt for the conversion rate, but will remember the last conversion rate you used. This program does not go on-line to get the current set of conversion rates (it's not possible in BC BASIC)

```
REM
```

```
REM The defaults here are roughly the conversion rate from yen to  
REM australian dollars. 1 yen is about 0.011 australian dollar;  
REM 10000 yen is therefore about 110 australian dollars.
```

```
REM
```

```
rate = INPUT DEFAULT Memory.GetOrDefault ("ConversionRate",  
0.011) PROMPT "Conversation rate <from> to <to> [e.g., yen to  
australian dollars]"
```

```
Memory.ConversionRate = rate
```

```
amount = INPUT DEFAULT Memory.GetOrDefault
```

```
("ConversionAmount", 10000) PROMPT "Amount to convert [e.g.,  
amount in yen]"
```

```
Memory.ConversionAmount = amount
```

```
value = amount * rate
```

```
Calculator.Message = "Convert " + amount + " at a rate of " + rate +  
" is " + value
```

```
Calculator.Value = value
```

### 37.9.6 Present Value

Calculates the present value (PV) of a sum of money (the future value, FV) given an interest rate and the number of years in the future that the sum of money will be paid.

```
REM Future value uses the standard formula PV = FV / (1+r)**n  
REM where PV = present value (e.g., money to invest)
```

REM FV is the value of the investment in the future

REM r is the interest rate (3% is .03 in the formula, but this program lets the user enter '3' for 3%)

REM n is the number of periods. This must match the interest rate (e.g., either both are 'per year' or both are 'per month')

REM Good practice to run the TEST program to make sure the calculations are OK.

TEST()

FV = INPUT DEFAULT 900 PROMPT "Future value (amount of money in the future)"

n = INPUT DEFAULT 3 PROMPT "When will the money be paid"

r = INPUT DEFAULT 3 PROMPT "Interest rate (per year). Enter 3 for 3%"

REM Some people want to type 10 for 10%

r = r / 100

PV = PresentValue (FV, n, r)

Calculator.Message = "PV of " + FV + " at " + r\*100 + "%"

STOP PV

FUNCTION PresentValue(FV, n, r)

PV = FV / ((1+r)\*\*n)

RETURN PV

FUNCTION TestOne(FV, n, r, expected)

nerror = 0

actual = PresentValue(FV, n, r)

IF actual ≈ expected THEN GOTO 10

nerror = nerror + 1

PRINT "ERROR: PresentValue"

PRINT "FV=";FV; " n="; n; " r=";r

PRINT "expected="; expected

PRINT "actual=";actual

10 REM

RETURN nerror

```
FUNCTION TEST()
nerror = 0
nerror = nerror + TestOne (900, 3, .1, 676.18)
nerror = nerror + TestOne (570, 1, .1, 518.18)
nerror = nerror + TestOne (570, 3, .1, 428.25)
RETURN nerror
```

### 37.9.7 Return on Investment

Also called ROI, the return on investment shows the percentage return on an investment.

```
REM ROI
EARNINGS = INPUT DEFAULT 1100 PROMPT "Earnings on the
investment"
INITIAL = INPUT DEFAULT 1000 PROMPT "Initial investment"
ROI = (EARNINGS - INITIAL) / INITIAL
Calculator.Message = "ROI when earnings is " + EARNINGS + " on an
initial investment of " + INITIAL
STOP ROI
```

## 37.10 EX: REAL ESTATE

Convert square feet to acres and more

### 37.10.1 Acres to square feet

Converts acres from the calculator display into square feet

```
value = Calculator.Value
retval=value * 43560
Calculator.Message = "Converted " + value + " acres into square feet"
STOP retval
```

### 37.10.2 Debt to Income calculations

Given two numbers -- the borrower's yearly income and the bank's income limit (e.g., 31 for 31% allowed for housing), calculates the allowed amount per month for housing.

```
income = INPUT DEFAULT 100000 PROMPT "What is the person's  
yearly income"  
monthlyIncome = income / 12  
  
housingPercent = INPUT DEFAULT 31 PROMPT "What is the allowed  
housing debt to income ratio?"  
maxPercent = INPUT DEFAULT 43 PROMPT "What is the allowed debt  
to income ratio? "  
  
allowedMonthlyHousing = INT (monthlyIncome *  
housingPercent/100)  
allowedMonthlyTotal = INT (monthlyIncome * maxPercent/100)  
  
CLS  
PRINT "Income per month="; INT(monthlyIncome)  
PRINT "Housing per month=";allowedMonthlyHousing  
PRINT "Total monthly debt=";allowedMonthlyTotal  
PRINT "Other debt=";allowedMonthlyTotal-allowedMonthlyHousing  
Calculator.Message = "For year income of " + income + " housing per  
month is " + allowedMonthlyHousing  
STOP allowedMonthlyHousing
```

### 37.10.3 Minimum and Maximum density

Demonstrates one way to calculate the minimum and maximum number of units that can be built on a lot given its size in acres. The rules roughly match those of Redmond, WA for residential neighborhoods (Redmond code 20C.30.25) as of 2015. You will need to supply the R type (e.g., 1 for R1, 4 for R4).

```
R = INPUT DEFAULT 6 PROMPT "What is the R type zoning district?  
Enter .2 for type RA-5"  
grossAcres = INPUT DEFAULT 2 PROMPT "What is the gross site area  
(acres)?"  
netAcres = INPUT DEFAULT grossAcres PROMPT "What is the net  
buildable area (acres)?"
```

REM Sample: R=6 grossAcres=2 netAcres=1.5 result in minimum 7 maximum 12 house

```
allowed = Math.Round (RtoAllowedDensity(R) * grossAcres)
minimum = Math.Round (RtoMinimumDensity(R) *
RtoAllowedDensity(R) * netAcres)
```

REM How much more land do you need to build 1 more house?

```
start = RtoAllowedDensity(R) * grossAcres
fraction = start - Math.Floor (start)
IF fraction >= .5 THEN next = Math.Ceiling(start) + .5
IF fraction < .5 THEN next = Math.Floor(start) + .5
delta = next - start
deltaAcres = delta / RtoAllowedDensity(R)
```

CLS

```
PRINT "You can build up to "; allowed; " houses"
PRINT "You must build at least "; minimum; " houses"
PRINT "You need "; deltaAcres; " more acres to build 1 more house"
STOP
```

REM The R number exactly matches the allowed density for all values

REM Except RA-5 which must be entered as .2

```
FUNCTION RtoAllowedDensity (R)
RETURN R
```

REM There are three minimum density sizes in Redmond

```
FUNCTION RtoMinimumDensity(R)
IF R < 8 THEN RETURN .8
IF R < 18 THEN RETURN .75
RETURN .65
```

### 37.10.4 Rectangle in feet to acres

Given a lot size in feet, calculates the lot size in acres

```
length = INPUT DEFAULT 80 PROMPT "Enter the first length in feet"  
width = INPUT DEFAULT 11 PROMPT "Enter the second length in feet"  
sqfeet = length * width  
acres=sqfeet / 43560  
Calculator.Message = "Lot " + length + "x" + width + " is " + acres +  
" acres"  
STOP retval
```

### 37.10.5 Square feet to acres

Converts the current values in the calculator from square feet to acres.

```
value = Calculator.Value  
retval=value / 43560  
Calculator.Message = "Converted " + value + " square feet into acres"  
STOP retval
```

## 37.11 EX: SPACE AND ASTRONOMY

Programs for astronomy and space

### 37.11.1 Arc Length

A COGO program to calculate an arc length or a circle given the radius and the angle (in degrees)

REM 3959 is the radius of the earth in miles

```
TEST()  
radius = INPUT DEFAULT 3959 PROMPT "Radius of the circle"  
degrees = INPUT DEFAULT 45 PROMPT "Angle in degrees"  
arc = ArcLength(degrees, radians)  
Calculator.Message = "Given radius=" + radius + " and angle=" +  
angle + " arc is " + arc  
STOP arc
```

```
FUNCTION ArcLength(degrees, radius)  
radians = Math.DtoR (degrees)
```

```
circum = Math.PI * 2 * radius  
arc = circum * radians / (2 *Math.PI)  
RETURN arc
```

```
FUNCTION TestOne (degrees, radius, expected)  
actual = ArcLength (degrees, radius)  
IF (actual ≈ expected) THEN RETURN 0  
PRINT "ERROR; ArcLength ";h  
PRINT "Expected "; expected  
PRINT "Actual "; actual  
PRINT "Difference "; actual-expected  
RETURN 1
```

```
FUNCTION TEST ()  
nerror = 0  
nerror = nerror + TestOne (3959, 45, 3109.391)  
nerror = nerror + TestOne(10, 90, 15.707963)  
nerror = nerror + TestOne(0, 90, 0)  
nerror = nerror + TestOne(10, 0, 0)  
  
IF (nerror > 0) THEN PRINT "HORIZON NERROR=";nerror  
IF (nerror > 0) THEN PAPER RED  
RETURN nerror
```

### 37.11.2 AU to Meters

Converts a distance in AU (Astronomical units) to a distance in meters

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

```
from = Calculator.Value  
m = ConvertToMeters(from, "au")  
Calculator.Message = "Convert " + from + " au into " + m + "  
meters"  
STOP m
```

## 37.11.3 Conversion Library

A set of functions to convert between AU and kilometer and between Parsecs, Lightyears and Meters

```
FUNCTION ConvertToMeters(distance, units)
IF units = "au" THEN RETURN distance * 149597870700
IF units = "earthradius" THEN RETURN distance * 6371000
IF units = "lightsecond" THEN RETURN distance * 299792458
IF units = "lightyear" THEN RETURN distance * 9.4605284E15
IF units = "parsec" THEN RETURN distance * 3.08567758E16
CONSOLE "ERROR: Astronomy Conversion library: Unknown units " +
units
RETURN Math.NaN

CLS BLACK
TEST()

FUNCTION TestOne (distance, units, expected)
actual = ConvertToMeters (distance, units)
IF (actual ≈ expected) THEN RETURN 0
IF (Math.IsNaN(expected) AND Math.IsNaN(actual)) THEN RETURN 0
PRINT "ERROR; ConvertToMeter "; distance; " "; units
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1

FUNCTION TEST ()
nerror = 0
REM Test AU with data from Mercury and Neptune
nerror = nerror + TestOne (.387, "au", 5.7894E10)
nerror = nerror + TestOne (30.06, "au", 4.4969E12)

nerror = nerror + TestOne (1, "lightsecond", 299792.458E3)
REM Wikipedia says the distance Earth to Moon is 1.282
nerror = nerror + TestOne (1.28222, "lightsecond", 384400E3)
```

```
REM data from Bing.com
nerror = nerror + TestOne (1, "lightyear", 9.4605284E15)
nerror = nerror + TestOne (1, "earthradius", 6371E3)
nerror = nerror + TestOne (1, "parsec", 3.08567758E16)
nerror = nerror + TestOne (1, "NOSUCHUNIT", Math.NaN)

IF (nerror > 0) THEN PRINT "Astronomical Conversion
NERROR=";nerror
IF (nerror > 0) THEN PAPER RED
RETURN nerror
```

### 37.11.4 Distance to horizon

Calculates the distance to the horizon in miles given a height above the Earth in feet.

```
h = Calculator.Value
TEST ()
d = Distance (h)
Calculator.Message = "Given a height of " + h + " feet, the distance to
horizon in miles is " + d
STOP d

FUNCTION Distance (height)
d = 1.22 * SQR(height)
RETURN d

FUNCTION TestOne (h, expected)
actual = Distance(h)
IF (actual ≈ expected) THEN RETURN 0
PRINT "ERROR; DistanceToHorizon ";h
PRINT "Expected "; expected
PRINT "Actual "; actual
PRINT "Difference "; actual-expected
RETURN 1
```

```
FUNCTION TEST ()  
nerror = 0  
nerror = nerror + TestOne (100, 12.2)  
nerror = nerror + TestOne(22841, 184.382)  
PRINT "HORIZON NERROR=";nerror  
IF (nerror > 0) THEN PAPER RED  
RETURN nerror
```

### 37.11.5 Lightyears to Parsecs

Converts a distance in parsecs to a distance in light-years

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

```
from = Calculator.Value  
m = ConvertToMeters(from, "lightyear")  
inv = ConvertToMeters(1, "parsec")  
to = m / inv  
Calculator.Message = "Convert " + from + " into " + to + " parsecs"  
STOP to
```

### 37.11.6 Meters to AU

Converts a distance in meters to a distance in AU (Astronomical units)

```
IMPORT FUNCTIONS FROM "Conversion Library"
```

```
from = Calculator.Value  
inv = ConvertToMeters(1, "au")  
to = from / inv  
Calculator.Message = "Convert " + from + " into " + to + " au"  
STOP to
```

### 37.11.7 Parsecs to Lightyears

Converts a distance in parsecs to a distance in light-years

**IMPORT FUNCTIONS FROM "Conversion Library"**

```
from = Calculator.Value  
m = ConvertToMeters(from, "parsecIES!")  
m = ConvertToMeters(from, "parsec")  
inv = ConvertToMeters(1, "lightyear")  
to = m / inv  
Calculator.Message = "Convert " + from + " into " + to + " light  
years"  
STOP to
```

### **37.11.8      Rocket Equation**

The Tsiolkovsky rocket equation will tell you how much fuel you have to burn in order to achieve some change in velocity (delta-v). You have to provide the starting rocket weight (including fuel) and the rocket effective exhaust velocity. The Space Shuttle effective exhaust velocity is 4,400 m/s.

```
REM From https://en.wikipedia.org/wiki/Tsiolkovsky_rocket_equation  
REM deltav = ve* ln (m0 / m1)  
REM m0=initial mass  
REM m1=final mass (after fuel is burnt)  
REM ve = effective exhaust velocity (about 4400 for Space Shuttle  
main engines)  
  
REM Solve the equation for m1 to return (m0-m1), the amount of  
propellant to burn  
REM deltav = ve* ln (m0 / m1)  
REM deltav / ve = ln(m0/m1)  
REM exp(deltav / ve) = m0/m1  
REM m0 / exp (deltav / ve) = m1  
REM fuelburned = m0 - (m0 / exp(deltav/ve))  
  
m0 = INPUT DEFAULT 727 PROMPT "Starting mass of the rocket (any  
units)"
```

```
REM Sample specific impulse is from the Centaur rocket
```

```
isp = INPUT DEFAULT 450.5 PROMPT "Specific Impulse (in seconds)"
```

```
REM delta-v of about 6,900,000 is (approximately) the amount  
needed to get to low earth orbit
```

```
deltav = INPUT DEFAULT 5000 PROMPT "What delta-v do you need?  
(in meters/second)"
```

```
CLS GREEN
```

```
ve = SpecificImpulseToEffectiveVelocity(isp)
```

```
fuelburned = FuelBurned (m0, deltv, ve)
```

```
m1 = m0-fuelburned
```

```
PRINT "Starting rocket mass="; m0
```

```
PRINT "Specific Impulse="; isp
```

```
PRINT "Exhaust Velocity="; ve
```

```
PRINT "Change in velocity=";deltav; "(m/s)"
```

```
PRINT "Fuel burned=";fuelburned
```

```
PRINT "Final rocket mass=";m1
```

```
IF (m1 > 0) THEN GOTO 10
```

```
PRINT "Ran out of fuel!"
```

```
PAPER RED
```

```
10 REM done
```

```
Calculator.Message = "You used " + fuelburned + "fuel"
```

```
STOP fuelburned
```

```
FUNCTION SpecificImpulseToEffectiveVelocity (lsp)
```

```
g0 = 9.81
```

```
Ve = g0 * lsp
```

```
RETURN Ve
```

```
FUNCTION FuelBurned (m0, deltv, ve)
```

```
ratio = 1 / EXP (deltv / ve)
```

```
m1 = m0 * ratio
```

```
fuelburned = m0 - m1
```

```
RETURN fuelburned
```

## 37.12 EX: STATISTICS

Sample programs for statistics

### 37.12.1 Finite Population Correction

Corrects the Margin of Error calcations when drawing from a finite instead of infinit population.

```
IMPORT FUNCTIONS FROM "Sample Size Library"
```

```
population = INPUT DEFAULT 1000000 PROMPT "Enter the actual population"
```

```
sample = INPUT DEFAULT 1000 PROMPT "Enter the number of samples"
```

```
fpc = FPC (sample, population)
```

```
Calculator.Message = "Finite Population Correction for population=" + population
```

```
STOP fpc
```

### 37.12.2 Margin of Error

Calculates the margin of error for samping an infinite population given a sample size

```
IMPORT FUNCTIONS FROM "Sample Size Library"
```

```
z = GetZ()
```

```
n = INPUT DEFAULT 1000 PROMPT "How many samples will you take?"
```

```
stddev = 0.5
```

```
REM 0.5 is a very conservative approach and applies a derate factor of .25
```

```
REM chosing .1 would result in a derate of 0.09 which is not wildly different
```

```
me = MarginOfError (z, n, stddev)
Calculator.Message = "Margin of error for sample size "+n
STOP me
```

### 37.12.3 Pfail

Returns the probability of failure by time t given an MTBF (mean time to failure) value.

REM Note: MTBF is 1/lambda

TEST()

```
MTBF = INPUT DEFAULT 10 PROMPT "MTBF (Mean Time Before Failure)"
```

```
T = INPUT DEFAULT 5 PROMPT "Time to calculate from"
```

```
P = Pfail (T, MTBF)
```

```
Calculator.Message = "Pfailure by time T given MTBF"
```

```
STOP P
```

FUNCTION Pfail (T, MTBF)

```
P = 1 - Math.E ** ( - (T / MTBF))
```

```
RETURN P
```

FUNCTION TestOne (T, MTBF, Expected)

```
Actual = Pfail (T, MTBF)
```

```
IF (Actual ≈ Expected) THEN RETURN 0
```

```
PRINT "Pfail: T=";T;" MTBF="; MTBF
```

```
PRINT "Actual=";Actual
```

```
PRINT "Expected=";Expected
```

```
RETURN 1
```

FUNCTION TEST()

```
nerror = 0
```

```
nerror = nerror + TestOne (43800, 250000, 0.16071)
```

```
RETURN nerror
```

### 37.12.4 Sample Size

Calculates the sample size required to meet a desired margin of error given a confidence limit.

```
REM Calculates a required Sample Size
```

```
REM You have to enter a Confidence (90, 95 or 99%)
```

```
REM You have to enter a margin of error (e.g., 3 to mean 3%)
```

```
REM
```

```
REM The code assume an infinite populat. Use the FPC (Finite Population Correction)
```

```
REM if the population size is smaller.
```

```
REM
```

```
REM The code assumes a population stddev of 0.5; this is the most
```

```
REM conservative assumption.
```

```
REM
```

```
IMPORT FUNCTIONS FROM "Sample Size Library"
```

```
PRINT "Sample Size"
```

```
TEST()
```

```
z = GetZ()
```

```
me = INPUT DEFAULT 5 PROMPT "What is your required margin of error in percent?"
```

```
me = me / 100
```

```
stddev = 0.5
```

REM 0.5 is a very conservative approach and applies a derate factor of .25

REM choosing .1 would result in a derate of 0.09 which is not wildly different

```
n = SampleSize(z, me, stddev)
```

```
STOP n
```

### 37.12.5      Sample Size Library

Useful functions for calculating sample size

REM A set of functions and tests for sample sizes

```
FUNCTION MarginOfError(z, n, stddev)
me = SQR ((z**2 * stddev * (1-stddev)) / n)
RETURN me
```

```
FUNCTION SampleSize(z, me, stddev)
n = z**2 * stddev * (1-stddev) / me**2
RETURN n
```

```
FUNCTION FPC(sample, population)
REM Finite Population Correction; corrects the Margin of Error
REM based on drawing from a finite (instead of infinite) population
```

```
ratio = (population - sample) / (population - 1)
fpc = SQR(ratio)
RETURN fpc
```

```
FUNCTION GetZ()
10 confidence = INPUT DEFAULT 95 PROMPT "Required confidence
level (one of 90, 95 or 99)"
z = Z(confidence)
IF (z = 0) THEN GOTO 10
RETURN z
```

```
FUNCTION Z(confidence)
IF (confidence = 90) THEN RETURN 1.645
IF (confidence = 95) THEN RETURN 1.96
IF (confidence = 99) THEN RETURN 2.576
RETURN 0
```

```
FUNCTION TestOne (confidence, n, me, stddev)
nerror = 0
z = Z(confidence)
nactual = SampleSize (z, me, stddev)
IF (nactual ≈ n) THEN GOTO 20
PRINT "ERROR: SampleSize (pt1)"
PRINT "z=";z
PRINT "me=";me
PRINT "expected n=";n
PRINT "actual n=";nactual

20 meactual = MarginOfError (z, n, stddev)
IF (meactual ≈ me) THEN GOTO 30
PRINT "ERROR: SampleSize (2)"
PRINT "z=";z
PRINT "n=";n
PRINT "expected me=";me
```

```
PRINT "actual me=";meactual
```

```
30 REM all done
```

```
RETURN nerror
```

```
FUNCTION TestFPC(sample, population, fpc)
```

```
nerror = 0
```

```
actual = FPC (sample, population)
```

```
IF (actual  $\leq$  fpc) THEN RETURN 0
```

```
PRINT "ERROR: FPC"
```

```
PRINT "sample=";sample
```

```
PRINT "population=";population
```

```
PRINT "fpc=";fpc;" actual=";actual
```

```
RETURN nerror
```

```
FUNCTION QuickTest()
```

```
nerror = 0
```

```
nerror = nerror + TestFPC (1000, 1000000, 0.99950037)
```

```
nerror = nerror + TestFPC (20, 50, 0.78246)
```

```
nerror = nerror + TestOne (99, 1727.34, 0.030990321, 0.5)
```

```
nerror = nerror + TestOne (95, 1000, 0.030990321, 0.5)
```

```
nerror = nerror + TestOne (90, 704.4, 0.030990321, 0.5)
```

```
nerror = nerror + TestOne (95, 1000, 0.026838405, 0.25)
```

```
RETURN nerror
```

```
FUNCTION TEST()
```

```
nerror = QuickTest()
```

```
RETURN nerror
```

```
REM RUN the library in order to test it!
```

```
PRINT "Testing Sample Size Library"
```

```
TEST()
```

# INDEX TO BEST CALCULATOR MANUAL

---

- (subtract), 89
- (subtract, BASIC), 88
- (subtract, Calculator), 17
- !
- (factorial) key, 27
- M (store)
  - In Calculator Memory Page, 35
- M (To Memory) key, 19
- % (percent) key, 23
- & (and) key, 40
- ( ) (parentheses, Calculator), 18
- ( ) (parenthesis)
  - when calling a function, required, 87
- ( ) (parenthesis, BASIC)
  - in expressions, 86
- \*(multiply), 88
- \*\* (raise to the power), 88
- / (divide), 88
- [ ] (square brackets)
  - in expressions, 86
- ^ (xor) key, 40
- | (or) key, 40
- ~ (inverse) key, 40
- + (add), 89
- + (add, Calculator), 17
- < (less than), 89
- <= (less than or equal), 89
- <> (not equal), 89
- = (equal), 89
- = (equals, Calculator), 17
- > (greater than), 89
- >= (greater than or equal), 89
- ± (change sign) key, 21
- ≪ (shift left) key, 42
- ≪+○ (rotate left) key, 42
- ≫ (shift right) key, 42
- ≫+○ (rotate right) key, 42
- × (multiply, Calcualtor), 17
- ÷ (divide, Calculator), 17
- √ (squarae root), 88
- √ (square root) key, 21
- ³√ (cube root), 88
- ³√x key, 27
- ⁴√ (fourth root), 88
- ≈ (approximately equal), 89
- ⌫ (Delete) key, 18
- ¤ (word) key, 41
- ¤¤¤ (dword) key, 41
- ¤ (byte) key, 41
- 1/x (Inverse) key, 21
- ² (square), 88
- 2's complement key, 40
- 2541, 156
- ³ (cube), 88
- ⁴ (fourth power), 88
- ∞ (infinity) key, 34
- ABS (BASIC), 93
- Abs key, 29
- AccelerometerSetup, 156
- ACS (BASIC), 92
- AlarmSetting, 153
- Algebraic Entry, 17

Guide to Using Best Calculator  
Algebraic Entry System with Hierarchy, 18  
Algebraic Operating System, 18  
AND, 90  
And (&) key, 40  
array, 104  
As  
    Bluetooth, device.As(), 136  
ASCII Table, 56  
ASN (BASIC), 93  
*assignment*, 113  
ATN (BASIC), 93  
Average  
    Example BASIC program, 105  
B# (count bits) key, 40  
BarometerSetup, SensorTag  
    2541, 156  
beLight, 146  
bin key, 39  
Binary numbers, 39  
bind a program to a key, 71  
Bluetooth, 130  
Bluetooth object, 134  
Bluetooth.Devices, 135  
Boxplots, 50  
ButtonSetup, SensorTag 2541,  
    157  
byte (ﷺ) key, 41  
C (clear) key, 18  
c (speed of light) key, 34  
Calculator.Message, 118  
Calculator.Value, 118  
Calendar calculations, 37  
CALL, 101  
callbacks, Bluetooth, 141  
Carriage return, 100  
CC2540T, 146  
CE (clear entry) key, 18  
Ceil key, 29  
Chain calculations, 17  
ChangeMode, 147, 154  
characteristic, Bluetooth, 138  
CHR, 95  
Classical Statistics, 46  
CLS, 102  
    clearing the screen manually,  
        74  
CODE, 95  
Comment (using REM), 116  
Console, 80  
    CONSOLE command, 162  
    CONSOLE Command, 103  
    display or hide (toggle), 81  
    DUMP command, 162  
Constants (Calculator), 34  
Conversions  
    Between bases, 39  
    BTUs, 53  
    Bushels, 55  
    Calories, 53  
    Celsius, 54  
    Centimeters, 54  
    Cups, 55  
    degrees to radians, 24  
    Donuts, 53  
    Ergs, 53  
    Fahrenheit, 54  
    Feet, 54  
    Gallons, 55  
    Grains, 54  
    Grams, 54  
    Inches, 53  
    Joules, 53  
    Kelvin, 54  
    Kilograms, 54  
    Kilometers, 54  
    Kilowatt-Hours, 53  
    Liters, 55  
    Long tons, 54  
    Maund, 54

Meters, 54  
Miles, 54  
MMT, 54  
Ounces, 54  
Pecks, 55  
Pints, 55  
Pounds, 54  
Quarts, 55  
radians to degrees, 24  
Rankine, 54  
Sèr, 54  
Short tons, 54  
Therms, 53  
Tolä, 54  
Tonnes, 54  
Troy Ounces, 54  
Troy Pounds, 54  
Yards, 54  
Correlation coefficient, 48  
COS (BASIC), 93  
Cos key, 24  
Count, 46  
count bits (B#) key, 40  
d→r key, 24  
data boxes, 45  
Date Calculations, 37  
dec key, 39  
Decimal numbers, 39  
degree and radians  
    In Calculator, 24  
degrees key, 24  
degrees of freedom, 49  
Desktop Shortcut, 58  
df, 49  
DIM, 104  
Direct Algebraic Logic, 18  
DOTTI, 147  
double  
    numbers stored as doubles, 83  
DUMP, 105  
dword (四四) key, 41  
e key, 34  
EE (scientific notation) key, 22  
ELSE, 111  
END, 117  
Escape (stop program) key, 79  
EXP (BASIC), 93  
exponential notation (BASIC), 83  
exponential notation (Calculator), 22  
expressions, 86  
F5 (run program) key, 79  
Factorial (Calculator), 27  
fff0, 149  
fff0, NOTTI, 155  
fff3, 149  
Fixed character size screen, 80  
floats, 83  
Floor key, 29  
Flux light, 152  
Font size  
    indicator, 80  
    larger, 80  
    smaller, 80  
FOR..NEXT loops, 106  
Formatting (Calculator), 31  
Frac key, 29  
FUNCTION, 109  
Get  
    Bluetooth.Devices, 135  
GetAccelerometer, Hexiwear, 151  
GetCalories, Hexiwear, 151  
GetFirmwareRevision, Hexiwear, 150  
GetGyroscope, Hexiwear, 151  
GetHeart, Hexiwear, 151  
GetHumidity, Hexiwear, 151

Guide to Using Best Calculator  
GetLight, Hexiwear, 151  
GetMagnetometer, Hexiwear, 151  
GetManufacturerName, Hexiwear, 150  
GetMode, 151  
GetPower, Hexiwear, 150  
GetPressure, Hexiwear, 151  
GetSteps, Hexiwear, 151  
GetTemperature, Hexiwear, 151  
GLOBAL, 109  
 $g_n$  (gravitational constant) key, 34  
GOSUB, 110  
GOTO, 110  
Gregorian Calendar, 37  
GUID, 138  
GyroscopeSetup, SensorTag 2541, 157  
Hebrew Calendar, 37  
Herschel notation, 25  
hex key, 39  
Hex numbers, 39  
Hexiwear, 150  
Hijiri Calendar, 37  
History of BASIC, 60  
HumiditySetup, SensorTag 2541, 157  
IF - THEN, 111  
IMPORT FUNCTIONS, 112  
import package, 70  
infinite loop (stopping), 79  
Init  
    Bluetooth, 135  
INPUT (operator, details), 92  
INPUT (operator, preferred), 90  
INPUT (statement, not preferred), 112  
INT (BASIC), 93

Page | 242  
Integer key, 29  
Integers, 83  
Intercept, 48  
Inverse Cos (calculator), 25  
Inverse Sin (calculator), 25  
Inverse Tan (calculator), 25  
IRSetup, SensorTag 2541, 157  
IsError, 86  
IsNaN, 86  
IsNumber, 86  
IsObject, 86  
IsString, 86  
Julian Calendar, 37  
Keyboard calculator button, 59  
LEDBlue, 152  
LEFT, 94  
LEN, 95  
LET, 113  
library  
    delete package, 73  
    export package, 73  
    how to display properties, 69  
    made up of packages, 68  
library  
    add package, 164  
Light, 146  
Line ending  
    \n, 82  
    \r, 82  
    \v, 82  
Linear Regression, 48, 51  
Linear Regression Chart, 48  
LN (BASIC), 93  
In key, 26  
LoadScreenFromMemory, 148  
log key, 26  
 $\log_2$  key, 26  
Logarithms (Calculator), 26  
Looping (FOR..NEXT loops), 106  
M- (memory subtract) key, 20

M→ (recall)  
In Calculator Memory Page, 35  
M→ (Recall) key, 19  
M-(memory subtract)  
Calculator Memory Page, 35  
M+ (memory add)  
In Calculator Memory Page, 35  
key, 20  
MagicLight, 152  
MagnetometerSetup,  
SensorTag 2541, 158  
Math.Abs, 123  
Math.Acos, 123  
Math.Asin, 123  
Math.Atan, 123  
Math.Atan2, 123  
Math.Ceiling, 124  
Math.Cos, 123  
Math.Cosh, 123  
Math.DtoR, 123  
Math.E, 127  
Math.Exp, 126  
Math.Factorial, 127  
Math.Floor, 124  
Math.Frac, 124  
Math.Log, 126  
Math.Log10, 126  
Math.Log2, 126  
Math.Max, 124  
Math.Min, 124  
Math.Mod, 124  
Math.NaN, 127  
Math.Pi, 127  
Math.Pow, 126  
Math.Round, 124  
Math.RtoD, 123  
Math.Sign, 125  
Math.Sin, 123  
Math.Sinh, 123  
Math.Sqrt, 126  
Math.Tan, 123  
Math.Tanh, 123  
Math.Truncate, 125  
Mean, 46  
median, 47  
Memory  
M+ (memory add) key, 20  
Memory Recall (M→) key, 19  
Memory Store (→M) key, 19  
Memory Subtract (M-) key, 20  
Naming memory, 35  
Memory Page, 19, 35, 37  
Memory.<constant\_name>, 120  
Memory.GetOrDefault, 120  
Memory.IsSet, 120  
Memory[<expression>], 119  
MID, 94  
mikroElektronika, 150  
Mod (modulo) key, 27  
N (count), 46  
N<sub>a</sub> (Avogadro's number) key, 34  
NaN key, 34  
NEXT, 106  
NOT, 90  
NOTI, 153  
oct key, 39  
Octal numbers, 39  
OR, 90  
Or (|) key, 40  
Output screen, 80  
close, 81  
CLS, 102  
larger, 81  
PRINT, 114  
smaller, 81  
P10 (10% percentile), 47

Guide to Using Best Calculator  
P90 (90% percentile), 47  
package  
    about, 72  
    delete, 73  
    how to add, 164  
    how to display properties, 69  
    how to export, 73  
    how to import, 70  
    made up of programs, 74  
Parentheses (Calculator), 17  
PAUSE, 114  
Percent (%) key (%), 23  
Percent Discount, 23  
Percent Formatting (Calculator),  
    33  
PI, 86  
PickDevicesName, 136  
Population Standard Deviation,  
    46  
PRINT, 114  
program  
    about, 76  
    add, 74  
    delete, 77  
    edit, 75  
    edit dialog, 78  
    edit from the About dialog,  
        76  
    how to add, 165  
    how to bind to a program  
        key, 168  
    how to run, 167  
    properties, 75  
    run, 75  
    running from edit dialog, 78  
    saving while editing, 78  
program list  
    how to display, 69  
Programmer's calculator, 39  
p-value, 49  
Q1 (1<sup>st</sup> quartile), 47  
Q2 (2<sup>nd</sup> quartile), 47  
Q3 (3<sup>rd</sup> quartile), 47  
r→d key, 24  
radians and degrees  
    (Calculator), 24  
radians key, 24  
RAND (set random seed), 115  
Random Numbers (BASIC), 115  
Random Numbers (Calculator),  
    30  
Read, Bluetooth device, 139  
ReadCachedByte, 139  
ReadCachedBytes, 140  
ReadRawByte, 139  
ReadRawBytes, 140  
Recall (M→), 19  
Regression, 48  
Relative Standard Deviation, 46  
REM (comment), 116  
RequestActive, 128  
RequestRelease, 128  
RIGHT, 94  
RND, 86, 115  
rnd key, 30  
rnd N key, 30  
Robust Statistics, 47  
rotate left (<<+○) key, 42  
rotate right (>>+○) key, 42  
Round key, 29  
Rounding (Calculator), 29  
RSD, 46  
Sales Tax, 23  
Sample Standard Deviation, 46  
save (export) package, 73  
SaveScreenToMemory, 148  
Scatterplots, 51  
Scientific (exponential) notation  
    (BASIC), 83

Guide to Using Best Calculator  
Scientific (exponential)  
    Notation (Calculator), 22  
screen clear key (BASIC), 74  
Screen.ClearLine, 128  
Screen.ClearLines, 128  
Screen.H, 128  
Screen.RequestActive, 128  
Screen.RequestRelease, 128  
Screen.W, 128  
SensorTag 2541 (original), 156  
SensorTag\*, 156  
service, Bluetooth, 138  
SetAlarmTime, 154  
SetColor, beLight, 146  
SetColor, MagicLight, 152  
SetColor, NOTTI, 154  
SetColorCustom, NOTTI, 154  
SetColumn, 148  
SetName, 148  
SetName, NOTTI, 154  
SetNameArbitrary, 148  
SetNameArbitrary, NOTTI, 154  
SetOff, MagicLight, 152  
SetOn, MagicLight, 152  
SetPanel, 148  
SetPixel, 148  
SetRow, 148  
SGN (BASIC), 93  
shift left (<<) key, 42  
shift right (>>) key, 42  
SIN (BASIC), 93  
Sin key, 24  
Slope, 48  
Smart quotes, 100  
    and strings, 84  
specializations, Bluetooth, 144  
SQR (BASIC), 93  
Standard Deviation, 46  
statement, 97  
statement terminator (not in BASIC), 82  
Statistical calculator, 44  
Statistics  
    Entering data, 45  
StdErr Line, 48  
StdErr Slope, 48  
STOP, 117  
Store ( $\rightarrow M$ ) key, 19  
String constants, 84  
Student's t-test, 49  
Sum, 46  
SWAB (swap bytes) key, 41  
SyncTime, 148  
SyncTime, NOTTI, 154  
t statistic, 49  
TAN (BASIC), 93  
Tan key, 24  
Texas Instruments, 156  
Texas Instruments, beLight  
    CC2540T, 146  
trigonometry, 24  
T-Tests, 49  
Tukey boxplots, 44  
Unicode minus signs, 84  
Unicode table (BASIC), 100  
Unicode table (Calculator), 57  
VAL, 96  
Variables, 84  
    ending in \$ (dollar sign), 85  
Visually Perfect Algebraic  
    Method, 18  
Welch's t-test, 49  
Witti Design, 147, 153  
word ( $\bowtie$ ) key, 41  
WriteCallbackDescriptor,  
    Bluetooth, 141  
WriteClientCharacteristicConfig  
    urationDescriptorAsync, 141  
 $\bar{x}$  (mean), 46

$x^2$  (Square) key, 21

$\pi$  key, 34

$x^3$  key, 27

$s$  (sample standard deviation),

Xor (^) key, 40

46

$x^y$  key, 28

$\Sigma$  (sum), 46

XY Scatterplots, 51

$\sigma_n$  (population standard

$y\sqrt{x}$  key, 28

deviation), 46