BriSopht BASIC Documentation

By Johnny Jarrell

March 16, 2021

Contents

Introduction

About

BriSopht BASIC, pronounced bree-soft, is a development environment I have created to make programming more accessible to the younger audience. BriSopht BASIC is a dialect of BASIC that I have put together that contains new commands that will be a little more intuitive to the young programmer. For example, instead of using a command such as CLS 1 to clear the screen with a green color they would instead type ERASE GREEN. The old commands are still there for us who grew up programming in BASIC.

Features:

- Free to use and contains no ads or data collection.
- Kid friendly interface for programming.
- Word suggestions to help those just starting to read and spell.
- Friendly commands for the young programmer.
- No file writing commands for security when running programs created by others.
- Animation and built-in graphics for use in your programs or games.

Why

This project began when I wanted to teach my kids, ages five and six, how to program in a way similar to the way I learned. I learned how to program in the early 80's using a TRS-80 Color Computer. I would type in the code, spend my time debugging and save it to a cassette tape.

When they were ages 4 and 5, I wanted to gauge their interest in programming so I let them play on my still working TRS-80 Color Computer 3. This computer is over 30 years old! I showed them how to erase the screen with a color, print some text and make some sounds. They really got a kick out of it. Teaching them how to program on this computer would be a very difficult and frustrating experience and I did not want to take that route. The commands are not very intuitive to the young programmer and showing how to edit mistakes would be a nightmare. The resolution on modern TVs is not very pretty either.

Most "programming" applications for their age are of the drag-n-drop variety like Scratch Jr. I wanted to teach them more than how to drag a code block around and change numbers. I wanted to teach them the actual typing of code and creating their own graphics. A modern programming language could do this but the learning curve would be a little steep for their age since the user interface and terminology is geared for learners well above their age. There was no way I would be able to explain libraries, dependencies, etc... for their age. Also, customizing commands or changing the syntax to make it easier for the young programmer would be very challenging for me.

Building an interpreter

So, I began my journey into creating a language interpreter. I created the user interface with a young programmer in mind. Big buttons, gentle colors, recognizable icons. Simplicity was my main goal. Watching how my daughters navigated the interface helped me tweak it. I started off one command at a time beginning with simple stuff like the shape commands and colors. Next came variables and loops. I stuck with simple conditional statements. I am still working with recursion. I used a couple of libraries to help with sound and solving recursive math equations.

Initial testing

After I got a working prototype up and running, I let my 6 year old start working with it. Very quickly I noticed a few things that needed to be added. She is at the age where she knows pretty much what letter a new word starts with but not necessarily how to spell the whole word. For example, she knew she needed to start with the ERASE command and wanted PURPLE. She knew ERASE started with an "e" and purple started with a "p" but struggled to spell the whole word. For this I added a word prediction box for a certain command set and the colors. I also added a help screen to show the colors, common shapes and how to spell them. The next hurdle I had was how to explain the coordinate system for placing graphics. I created a coordinate box that shows the position of the mouse so she could determine the numbers to use for coordinates. To start off, I am just working with them to create pictures with the shape commands and a little music. This is very visual and they love the creative aspect. I will gradually present them with new scenarios and commands to practice. Letting them type in simple programs from the Usborne books is also rewarding to them.

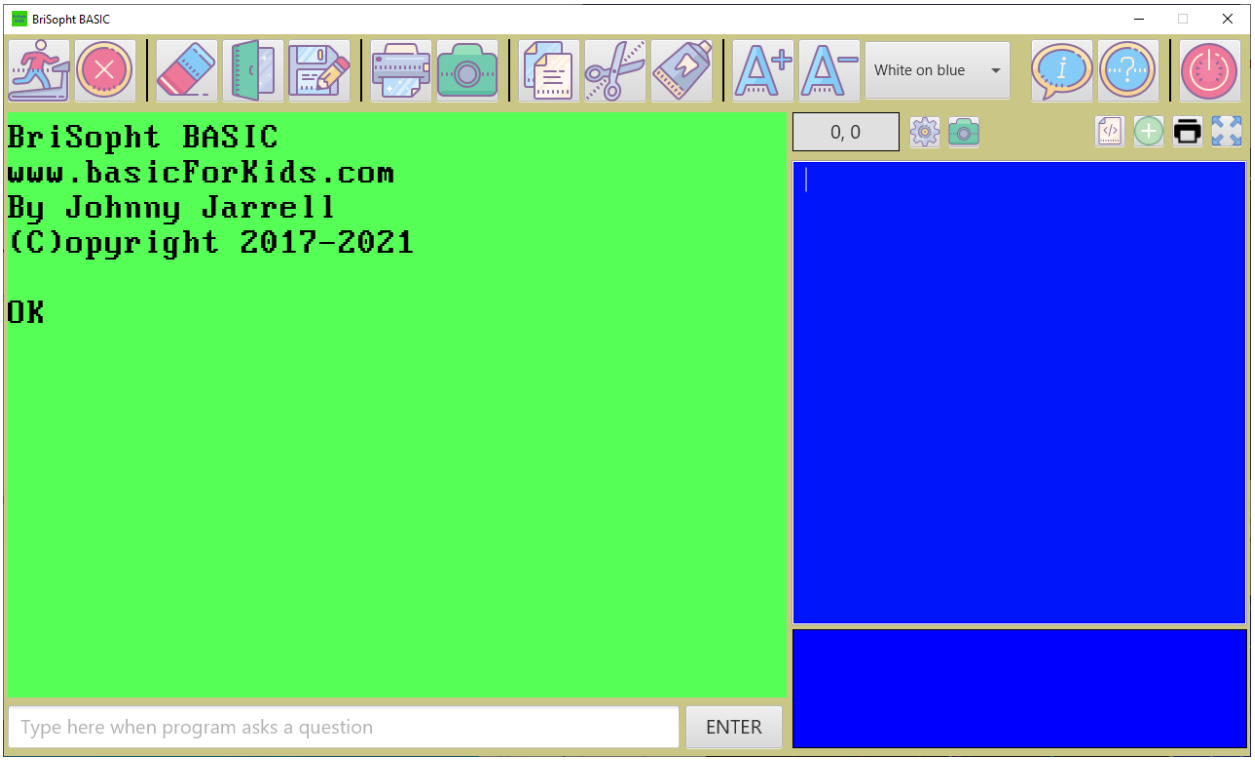
Skills utilized by the new young programmer:

- Keyboarding
- Spelling
- Colors
- Shapes
- Programming concepts
- Grid coordinates
- Problem solving (debugging)
- Ordering
- NumbersMath
- Word recognition

The future

Now I hope to learn from others how they use BriSopht BASIC and incorporate their feedback into improving BriSopht BASIC.

The BriSopht Interface



Execute (run) the current program.

Stop the program currently running.

Erases the current program and clears the display area so you can begin entering a new program.

Load (open) a program. Programs can be typed in other editors but must have the ***.bas*** extension.

Saves the current program.

Print the output screen.

Take a screen shot and save it in the BriSopht BASIC captures folder. The smaller version of this icon near the settings icon, opens up the folder containing the screen shots.

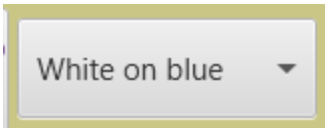
Copy selected section of code to the clipboard.

Paste the current clipboard.

Copy the selected code to the clipboard and delete it from the code window.

Increase the font size used in the editor.

Decrease the font size used in the editor.

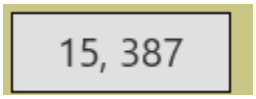


Change contrast of text colors in editor.

Display information about **BriSopht BASIC** along with any copyright and versioning information.

Show example graphics and colors and application path.

Turn off **BriSopht Basic**.



Shows the position of the mouse in the display area to help get coordinates for placing graphic objects or cursor positioning.

Set the resolution of the coordinate locations.

Expand the code window.

Collapse an expanded code window.

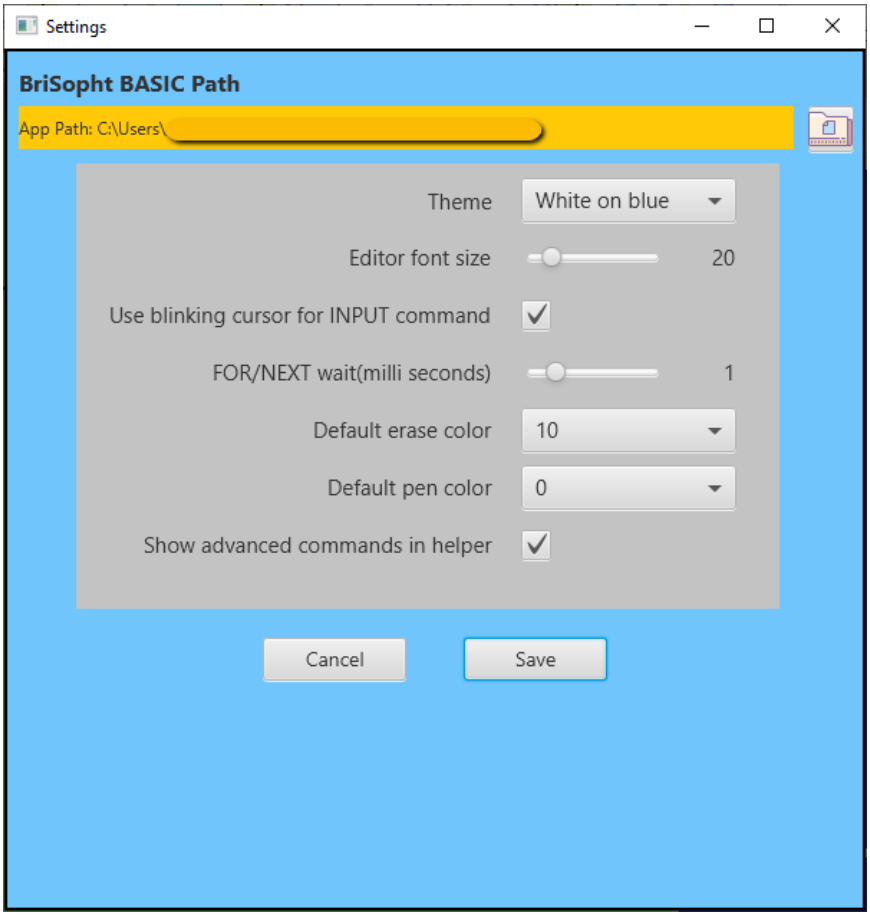
Print the currently loaded program code.

Hide advanced commands from code hints.

Show all code including code from “included” programs using some basic code highlighting.

Opens the folder where the screen captures are saved.

Opens the settings window.



The BriSopht Language Guide

Here are some guidelines when creating programs in **BriSopht BASIC**.

Commenting Lines Of Code

You can use **REM** or ‘ or // to comment a line. The commented line of code will not be executed. For bulk commenting use //* to start the block and *// to end the comment block. See the code example for **REM** to see how to use the different commenting commands.

Line Numbers

The use of line numbers is optional. Your program can use both line numbers and no line numbers for command lines.

Variables

Variables can be used to store strings(text) or numbers(doubles). Variable names can contain up to two letters. Numbers 0-9 can be used in place of the second letter. Variable names are not case sensitive. String variables are denoted with a $.

Examples:

```
LET A=10
LET AC=5
LET A5=34
LET A$=”HELLO”
LET A1$=”WORLD”
LET AB$=”AGAIN”
```

There are 32,000 slots, beginning with the value of 0 and ending with 31,999, to store either strings or variables using the **POKE** and **POKE$** commands. You can use the **PEEK** command to retrieve these values.

Arrays

Arrays use the same naming convention as variables.

```
LET A(2)=7, LET B$(20)=”Hello”
```

Multi-dimensional array dimensions are separated by commas.

```
LET A(10,5)=200
```

A **DIM** statement is not required and will be ignored.

Recursion

I am working on better recursion.

Nested **FOR/NEXT** statements should work as expected.

For example:

```
FOR X=1 TO 10
      FOR Y=1 TO 10
            DOT X,Y
      NEXT Y
NEXT X
```

All math recursive functions should work fine.

For example:

```
LE A=SIN(COS(45))*10
```

Recursive functions involving strings or string variables should be resolved individually.

For example:

```
LET A=LEN("HELLO")+10
SHOULD BE WRITTEN AS
LET A=LEN("HELLO")
LET A=A+10
```

Another example:

```
LET A$=LOWER(LEFT("HELLO",2))
SHOULD BE WRITTEN AS
LET A$=LEFT("HELLO",2)
LET A$=LOWER(A$)
```

Conditional Statements

You can use Boolean operators AND and OR for multiple tests.

You can use these operators to test.

<, >, =, !, <>, ><, =>, >=, =<, <=

Boolean groups are not yet available.

This will not work:

```
if a=5 and (b=5 or c=10) then
```

Complicated nested **IF/ELSEs** will most likely not work correctly. An example would be:

```
IF THEN
      IF THEN
      ELSE
      ENDIF
      IF THEN
      ELSE
      ENDIF
ELSE
      IF THEN
      ENDIF
ENDIF
```

Variable Conversion

You can convert number variables to string variables by using:

```
LET A=5

LET B$=A
```

Do not try to convert a number.

```
LET A$=5
```

Code it as

```
LET A$="5"
```

The BriSopht Command List

Math Functions

There are some really advanced math commands and concepts included that the young programmer would have no clue about. Do not worry about this. They are included for the advanced programmer.

Use * for multiplication and / for divide.

Example: PRINT (10-(2+2)*2)/2

Replace any instances of powers using ^ with the POWERS command

Example: Change PRINT 2^3 to PRINT POWERS(2,3)

ABIN()– Returns a decimal number from a binary string. **PRINT ABIN("1101")**

ABS() – Returns the absolute value of a value. **PRINT ABS(-4)**

ACOS() – Returns the inverse cosine of a value in radians. **PRINT ACOS(VALUE)**

ACOSD() – Returns the inverse cosine of a value in degrees. **PRINT ACOSD(VALUE)**

AHEX() – Returns a decimal number from a hexadecimal string. **PRINT AHEX("1F")**

AOCT() – Return a decimal number from a octal string. **PRINT AOCT("11")**

ASIN() – Returns the inverse sine of a value in radians. **PRINT ASIN(VALUE)**

ASIND() – Returns the inverse sine of a value in degrees. **PRINT ASIND(VALUE)**

ATAN() – Returns the inverse tangent of a value in radians. **PRINT ATAN(VALUE)**

ATAND() – Returns the inverse tangent of a value in degrees. **PRINT ATAND(VALUE)**

AVG() – Returns the average of a set. **PRINT AVG(10,20,20)**

BIN() – Returns a binary string of a decimal number. **PRINT BIN(23)**

CEILING() – Round up to the next highest integer. **PRINT CEILING(1.2)**

CINT() – Round .5 and greater up to the next integer else round down to the next. **PRINT CINT(1.3)**

COS() – Returns the COS of a value in radians. **PRINT COS(VALUE)**

COSD() – Returns the COS of a value in degrees. **PRINT COSD(VALUE)**

DICE() – Returns a random number representing the size of the die and the number to roll. **PRINT DICE(3,6)** to roll three six sided dice.

EXP() – Return the natural log of a number. **PRINT EXP(30)**

FIX() – Truncates value at the decimal point. **PRINT FLOOR(1.2)**

FLOOR() – Truncates value at the decimal point. **PRINT FLOOR(1.2)**

HEX() – Return a hexadecimal string of a decimal number. **PRINT HEX(23)**

INT() – Returns a value as an integer. **PRINT INT(1.7)**

LOG() – Return the log of a number. **PRINT LOG(30)**

MAX() – Returns the highest value from a set. **PRINT MAX(2,3,1)**

MEDIAN() – Returns the median of a set. **PRINT MEDIAN(1,2,2,3,4,4,5)**

MIN() – Returns the lowest value from a list. **PRINT MIN(3,2,5)**

MOD() – Returns the remainder. **PRINT MOD(10,3)**

POWER() – Returns a number raised to a power. **PRINT POWER(2,3)**

RANDOM() – Generate a random number between 1 and the number provided. **PRINT RANDOM(10)**

ROUND() – Round .5 and greater up to the next integer else round down to the next. **PRINT ROUND(1.3)**

RND() – Generate a random number between 1 and the number given. **PRINT RND(10)**

RND2() – Generate a random number between 0 and 1 multiplied by the number given. **PRINT RND2(10)**

SGN() – Returns 1 for positive number, 0 for zero and -1 for a negative number. **PRINT SGN(-20)**

SIN() – Returns the sine of an angle in radians. **PRINT SIN(VALUE)**

SIND() – Returns the sine of an angle in degrees. **PRINT SIND(VALUE)**

SOLVE() – Solve a string of text or data in a string variable. **PRINT SOLVE("3+4")**

SPC() – Generate a string with a number of spaces. **PRINT SPC(5)**

SQR() – Return the square root of a value. **PRINT SQR(4)**

STRING() – Generate a string with a number of characters or patterns. **PRINT STRING(5,"*")**

TAN() – Returns the tangent of an angle in radians. **PRINT TAN(VALUE)**

TAND() – Returns the tangent of an angle in degrees. **PRINT TAND(VALUE)**

Date Functions

YEAR(0) – Returns the current year as a number. **PRINT YEAR(0)**

MONTH(0) – Returns the current month as 1-12. **PRINT MONTH(0)**

DAY(0) -Returns the current day of the month as a number. **PRINT DAY(0)**

DATE$** – Returns the date as a string formatted as MM-DD-YYYY. **PRINT DATE$

HOUR(0) – Returns the current hour of the day. **PRINT HOUR(0)**

MINUTE(0) – Returns the current minute of the hour. **PRINT MINUTE(0)**

SECOND(0) – Returns the current second of the minute. **PRINT SECOND(0)**

Program Commands

ANIMATION OFF – Turn your animation on and off. **ANIMATION OFF**

ANIMATION ON – Turn your animation on and off. **ANIMATION ON**

ANIMATIONPOSITION – This is where your animation scene will be drawn. **ANIMATIONPOSITION 100,100,320,320**

ANIMATIONSPEED– Set the speed of your animation. Speed is in milliseconds. A speed of 1000 is equal to 1 second. **ANIMATIONSPEED 1000**

ARC – Draw an arc at coordinate x,y with a radius, starting angle and angle of arc. It can be filled or empty. **ARC 200,200,100,0,90**

ASC() – Return the ASCII value of a character or first character of a string. **PRINT ASC("A")**

AUTOREFRESH ON - Turn on the autoscreen refresh after NEXT and LOOP statements. **AUTOREFRESH ON**

AUTOREFRESH OFF - Turn off the autoscreen refresh after NEXT and LOOP statements. **AUTOREFRESH OFF**

BEEP – Play a short beep sound. **BEEP**

BOX – Draw a box starting at coordinate x, y with a given width and height. It can be filled or empty. **BOX 10,10,100,100**

CHR() – Return the character of an ASCII value. **PRINT CHR(65)**

CIRCLE – Draw a circle at a x, y coordinate with a designated radius. Has optional parameters color, height ratio (1 is a perfect circle), start point (0-1) and end point (0-1). It can be filled or empty. **CIRCLE 100,100,40**

CIRCLE2 – Draw a circle at a x, y coordinate with a designated height and width and an optional color component. It can be filled or empty. **CIRCLE2 200,150,100,50,blue**

CLIPSET – Send text to the clipboard. **CLIPSET**

CLS – Clear the screen with the background color or color as an argument. This is the same as the erase command. **CLS(0)**

COLOR – Set the foreground and background color. **COLOR BLUE,BLACK**

CSRLIN – Return the column of the cursor. **PRINT CSRLIN**

CURSOR – Set the x, y coordinate (in pixels) where text will start when using the print command. **CURSOR 40,50**

DATA – Strings or variables that can be read. The data pointer increases as each data value is read. **DATA "CAR",50**

DATACOUNT(0) – Returns the number of entries located in the **DATA** statement. **PRINT DATACOUNT(0)**

DOT – Draw a dot at coordinate x, y. Size is controlled by the thickness command. **DOT 100,100**

DRAW – Draw a picture with directional commands beginning at coordinate x,y. Will use the ZX Spectrum draw command, which draws a line from the last plot, pset, line coordinate, if no quote or $ present.

ELSE – Used with **IF** statements if test is false.

END – Stop the program. **END**

ENDIF– Mark the end of an **IF** statement.

ERASE – Clear the screen with the background color or color as an argument. **ERASE BLUE**

ERASEHEX – Clear the screen with a hex color. **ERASEHEX "#FFFFFF"**

ERASERGB – Clear the screen with a rgb color. **ERASERGB(100,20,80)**

FILELINES – Return the number of lines from an opened file. **PRINT FILELINES**

FILL OFF – Turn off fill for shapes. The shape will be filled with the current pen color. The fill state will remain until it is switched on. **FILL OFF**

FILL ON - Turn on fill for shapes. The shape will be filled with the current pen color. The fill state will remain until it is switched off. **FILL ON**

FONTSET – Change the font. 0 for TRS-80 font set, 1 for DOS font set and 2 for a Courier like font set. The default font set is set to 1. **FONTSET 1**

FOR – Do a set of commands a certain amount of times at a specific interval using **STEP**. If **STEP** is omitted a value of 1 will be used. Assigns a count value to a variable. Returns back to the **FOR** with **NEXT**. **FOR I=1 TO 20 STEP 2**

GET x, y, width, height, which screen – Grab a section of a screen to use with the **PUT** command. **GET 10,10,100,100,0**

GOSUB – Jump to a label or line number. Use the **RETURN** command to return to the line following **GOSUB**. **GOSUB NEWGAME**

GOTO – Jump to a label or line number. **GOTO NEWGAME**

HIDE – Hide the code window until the program finishes running. **HIDE**

HOME – Same as the **ERASE** command. **HOME BLACK**

IF – Compare two statements for equal, =, greater than, >, less than, <, or not,!.

INCLUDE – Insert another program at this point in the code. **INCLUDE "PROGRAM2.BAS"**

INPUT– Ask the user for input to be stored in a variable.

INSTR - Returns the position of a search string from a string. Will return 0 if the search string is not found. **PRINT INSTR("hello world", "world")**

LABEL – A label is a location in the program used by **GOTO** and **GOSUB** to jump to in the program flow.

LEFT – Create a string from the left most number of characters. **PRINT LEFT("HELLO",2)**

LET – Set a variable to a value. **LET A=10**

LINE – Draw a line from coordinate x, y to x1 ,y1. The stroke size is controlled by the thickness command. **LINE 10,10,100,100**

LOADIMAGE – Loads an image file and places it on the screen.

LOADIMAGE2 – Loads an image file and places it on the screen.

LOADIMAGE3 – Loads an image file and places it on the screen.

LOCATE – Move the cursor to an x, y coordinate based on text size. **LOCATE 10,10**

LOOP – Go to a label that is located above the loop X times. **LOOP MYLABEL,10**

LOWER – Change a string to lower case. **PRINT LOWER("HELLO")**

LTRIM – Remove white space on the left side of a string. **PRINT LTRIM(" hello ")**

MEDIALOAD – Open a media file (video or music) and display it at coordinates x,y with a width and height and a control bar. The media will not auto play. Use the mediaplay command to load and auto play. You can set the width and height to 0 for sound files. **MEDIALOAD "fish.mp4",100,100320,240**

MEDIAPAUSE– Pause the currently playing media. **MEDIAPAUSE**

MEDIAPLAY – Open a media file (video or music) and display it at coordinates x,y with a width and height and a control bar. The media will auto play. Use the mediaload command to load without auto playing. You can set the width and height to 0 for sound files. **MEDIAPLAY "fish.mp4",100,100320,240**

MID – Create a string starting at a position and number of characters. **PRINT MID("HELLO",2,2)**

MIDIPLAY – Play a midi file. The midi file must be located in your application folder or application sub-folder. You can press the help button to find this folder. **MIDIPLAY "MIDIS/LAUNCH.MID"**

MIDISTOP – Stop playing the current midi file. **MIDISTOP**

MOUSEBUTTONMIDDLE(0) – Get the state of the middle button. Returns 0 for not pressed and 1 for pressed. **PRINT MOUSEBUTTONMIDDLE(0)**

MOUSEBUTTONPRIMARY(0) – Get the stat of the primary button. Returns 0 for not pressed and 1 for pressed. **PRINT MOUSEBUTTONPRIMARY(0)**

MOUSEBUTTONSECONDARY(0) – Get the state of the secondary button. Returns 0 for not pressed and 1 for pressed. **PRINT MOUSEBUTTONSECONDARY(0)**

MOUSEX – Get the x coordinate of the mouse. **PRINT MOUSEX**

MOUSEY – Get the y coordinate of the mouse. **PRINT MOUSEY**

NEXT – Return to the designated **FOR** line to continue loop.

ON…GOSUB – Jump to a subroutine based on a variable. **ON A GOSUB 100,200,300**

ON…GOTO – Jump to a label based on a variable. **ON A GOTO 100,200,300**

OPENFILE – Open a text file for reading. The file must be located in your application folder or application sub-folder. You can press the help button to find this folder. If you installed as a local user only, then you may have to enable viewing of hidden files on the View tab in the file explorer. **OPENFILE "MYFILE.TXT"**

OVAL – Draw an oval at coordinate x, y with a designated height and width. **OVAL 50,50,100,50**

PEEK – Get the number or string that was **POKE**d. The locations are numbered 0-31999. **PEEK(35)**

PEN – Set the foreground or drawing color. **PEN BLUE**

PENHEX - Change the pen/drawing/foreground color using a HEX value. **PENHEX "FFFFFF"**

PENRGB – Set the foreground or drawing color to a rgb color. **PENRGB 100,50,10**

PLAY – Play notes and music from a string of characters. You will need to add a **WAIT** command to allow the screen to refresh. Visit Creating Music to learn more.

PLOT – Same as the **PSET** command. **PLOT(200,250)**

POLYGON – Draw a line between each set of points. The polygon doesn't automatically close so you will need to make the end point the starting point.

POKE - Set a number in a special reserved area to call later with **PEEK**. The locations are numbered 0-31999. **POKE 3,200**

POKE$ – Set a number or string in a special reserved area to call later with **PEEK**. The locations are numbered 0-31999. **POKE$ "HELLO",34**

PRESET – Will set a pixel at coordinate x, y to the background color. **PRESET(10,30)**

PRINT – Print text and variables. **PRINT "HELLO"**

PRINT@ – Start printing at a screen location. **PRINT@230,"HELLO"**

PRINTWIDTH – Set the screen wrap for **PRINT@** command. **PRINTWIDTH 40**

PSET – Will draw a pixel at coordinate x, y to with an optional color. Default color is foreground color. **PSET(10,30),3**

PUT – Place an image that was grabbed with the GET command on a particular screen number. **PUT 100,100,0**

RANDOM – Generate a random number between 1 and the number given. **RANDOM 10**

READ – Read data into a variable. Moves the data pointer forward. **READ A$**

READCOUNTER – Get or set the counter when reading an open file. **READCOUNTER 10:PRINT READCOUNTER(0)**

READLINE – Read a line from an open file. Counter will automatically increment. **LET A$=READLINE:PRINT A$**

RECTANGLE – Draw a rectangle starting at coordinate x, y and ending at coordinate x1, y1. It can be filled or empty. **RECTANGLE 10,10,100,100**

REM – You can use **REM** or ' or **//** to comment a line of code. This line of code will not be executed. For bulk commenting use **//*** to start the block and ***//** to end the comment block. **REM IGNORE THIS LINE**

RESTORE – Moves the data pointer back to the beginning. **RESTORE**

RETURN – Return to the line following the last used **GOSUB**.

RIGHT – Create a string from the right most number of characters. **PRINT RIGHT("HELLO",2)**

RND() – Generate a random number between 1 and the number given. **PRINT RND(10)**

RND2() – Generate a random number between 0 and 1 and multiply it by the number given. **PRINT RND2(10)**

RTRIM – Remove white space on the rightt side of a string. **PRINT RTRIM(" hello ")**

SIZE – Set the size of the text. **SIZE 24**

SLEEP – Pause a number of seconds before the screen updates. **SLEEP 5**

SQUARE – Draw a square starting at coordinate x, y with a given length of a side. It can be filled or empty. **SQUARE 10,10,50**

STEP – Interval to count in a **FOR** loop. Use a negative number to **STEP** backwards.

STOP – Stop the program. **STOP**

THICKNESS – set the thickness of dots and lines. **THICKNESS 4**

TRIANGLE – Create a triangle by providing three points. **TRIANGLE 10,10,100,100,100,200**

TRIM – Remove white space on the both sides of a string. **PRINT TRIM(" hello ")**

UNPLOT – Same as the **PRESET** command. **UNPLOT(20,30)**

UPPER– Change a string to upper case. **PRINT UPPER("boat")**

WAIT – Pause a number of seconds before the screen updates. **WAIT .5**

WEND - Return to the previous **WHILE** statement and check and see if conditions are met. **WEND**

WHILE - Checks if conditions are met. If not, the program jumps down to the next **WEND** statement. **WHILE A<10**

WIDTH – Will set the number of text columns. The default is 40. **WIDTH 80**

BriSopht Tiles

Tiles are integrated graphics you can use with your BriSopht programs. Just use the name of the tile to refer to it in your program. Each tile is 32 pixels by 32 pixels.

```
TILE "GRASS.PNG",320,320
LET A$="HILLS.PNG"
TILE A$,320,352
FOR I=1 TO 5
     TILE "GRASS.PNG",I*32,I*32
NEXT I
```


airship.png

airship2.png

airship3.png

anvil.png

axeman.png

axeman2.png

barrel.png

barrel2.png

bat.png

bat2.png

bedhorzblue.png

bedhorzpurple.png

bedhorzred.png

bedvertblue.png

bedvertpurple.png

bedvertred.png

beegiant.png

blockblack.png

blockbrown.png

blockgray.png

blockltblue.png

blockpink.png

blockwhite.png

boat.png

boat2.png

boat3.png

books.png

bordergray.png

boy1.png

boy2.png

brickblack.png

brickblack2.png

bridgebottom.png

bridgesingle.png

bridgetop.png

bridgevertall.png

bridgevertleft.png

bridgevertright.png

bushes.png

bushesflowers.png

cabinet.png

camp.png

camp2.png

campfire.png

campfire2.png

candelabra1.png

candelabra2.png

candelabra3.png

candle1.png

candle2.png

candle3.png

cave.png

cavemountain.png

chicken.png

chicken2.png

columnblue.png

columnred.png

columnwhite.png

columnwood.png

coral1.png

cow.png

cow2.png

crate.png

crate2.png

desk.png

doorwood.png

doorwoodbars.png

fireplacegray1.png

fireplacegray2.png

fireplacegray3.png

fireplacered1.png

fireplacered2.png

fireplacered3.png

floordirt.png

fountain1.png

fountain2.png

fountain3.png

girl1.png

girl2.png

grass.png

halbardman.png

halbardman2.png

hills.png

horse.png

horse2.png

icongloveleather.png

iconglovemail.png

iconglovepadded.png

icongloveplate.png

iconhalbard.png

iconhammer.png

iconhelmleather.png

iconhelmmail.png

iconhelmpadded.png

iconhelmplate.png

iconinstrument.png

iconkey.png

iconleather.png

iconmace.png

iconmail.png

iconmetalshield.png

iconmorningstar.png

iconpadded.png

iconplate.png

iconring.png

iconsack.png

iconspear.png

iconstaff.png

iconsword.png

iconswordshort.png

icontorch.png

iconwoodshield.png

jester1.png

jester2.png

king1.png

king2.png

ladderdown.png

ladderup.png

ladyblue.png

ladyblue2.png

ladygreen.png

ladygreen2.png

ladypink.png

ladypink2.png

ladyred.png

ladyred2.png

ladywhite.png

ladywhite2.png

lancer.png

lancer2.png

lantern.png

lava.png

lava2.png

manBlue.png

manBlue2.png

manGreen.png

manGreen2.png

manRed.png

manRed2.png

manWhite.png

manWhite2.png

mine.png

monument.png

moon1.png

moon2.png

moon3.png

moon4.png

moon5.png

moon6.png

moon7.png

moon8.png

moonFull.png

mountains.png

orb.png

palm.png

pirate.png

pirate2.png

pit.png

podium.png

post.png

post1.png

post2.png

post3.png

post4.png

post5.png

pyramid.png

queen1.png

queen2.png

raft.png

rat2.png

ratgiant.png

rocks.png

rubble.png

ruins.png

ruinstower.png

runea.png

runeb.png

runec.png

runed.png

runee.png

runeea.png

runeee.png

runef.png

runeg.png

runeh.png

runei.png

runej.png

runek.png

runel.png

runem.png

runen.png

runeng.png

runeo.png

runep.png

runer.png

runes.png

runest.png

runet.png

runeth.png

runeuv.png

runew.png

runex.png

runey.png

runez.png

sack.png

ship.png

ship2.png

shipback.png

shipfront.png

sign.png

sink.png

snake.png

snake2.png

spidergiant.png

stairsdown.png

stairsup.png

statue.png

swamp.png

swirl.png

swirl2.png

swordsman.png

swordsman2.png

table.png

teleporter1.png

teleporter2.png

teleporter3.png

tentcircus.png

texta.png

textb.png

textblock.png

textc.png

textcapleft.png

textcapright.png

textd.png

texte.png

textf.png

textg.png

texth.png

texti.png

textj.png

textk.png

textl.png

textm.png

textn.png

texto.png

textp.png

textq.png

textr.png

texts.png

textspace.png

textt.png

textu.png

textv.png

textw.png

textx.png

texty.png

textz.png

tileblack.png

tilegray.png

tilered.png

toilet.png

torch1.png

torch2.png

tower.png

treedead.png

treegreat.png

treeoak.png

treepine.png

void.png

walldirt.png

walledcity.png

wallgray.png

wallgreen.png

wallred.png

waterdeep.png

waterdeep2.png

waterdiag1.png

waterdiag2.png

waterdiag3.png

waterdiag4.png

watershallow.png

watershallow2.png

well.png

wood.png

Making Music in BriSopht

Music demo program

```
PRINT "START PLAYING"
WAIT .5
LET A$="C D E F G A B C6"
PLAY A$
PRINT "EXAMPLE 2"
WAIT .5
PLAY "V0 I[Piano] Eq Ch. | Eq Ch. | Dq Eq Dq Cw V1 I[Flute] Rw | Rw |
GmajQQQ Cmajw"
PRINT "NOW FOR SOME RANDOM NOTES"
LET B$=""
FOR I=1 TO 10
LET R=RANDOM(127)
LET B$=B$;" ";R
NEXT I
PRINT B$
WAIT .5
PLAY B$
PRINT "FINISHED PLAYING"
```

Use spaces to separate each command.

Use a, b, c, d, e, f, g for notes. Notes can also be played with a value from 0 to 127.

Use r for rest. **Example:** rw for a whole rest.

Place a # for sharp and lowercase b for flat after the notes. **Example:** c# or eb for c sharp and e flat.

Use o for octave ranging from 0-10 with a default of 5. **Example:** c6 plays the note c on octave 6.

Use w, h, q, i, s, t, x or o, for whole, half, quarter, eighth, sixteenth, thirty-second, sixty-forth or 128th notes respectively. Example: cw for note c played as a whole note.

Add a . after duration for dotted note. **Example:** h. for dotted half note.

You can mash durations such as **whq** to tie the durations together.

Use v for voice 0-15. **Example:** v1 See the code example for more details on how to use the voice command.

Use i for instruments 0-127. **Example:** i20

Use t for tempo. **Example:** t60

Here are some examples of sound effects using the instrument numbers from the Midi Instrument List.

```
REM APPLAUSE
PLAY "I126 cw"
REM SEASHORE
PLAY "I22 cww"
```

Midi Instrument List

Piano:

0 Acoustic Grand Piano

1 Bright Acoustic Piano

2 Electric Grand Piano

3 Honky-tonk Piano

4 Electric Piano 1

5 Electric Piano 2

6 Harpsichord

7 Clavinet

Chromatic Percussion:

8 Celesta

9 Glockenspiel

10 Music Box

11 Vibraphone

12 Marimba

13 Xylophone

14 Tubular Bells

15 Dulcimer

Organ:

16 Drawbar Organ

17 Percussive Organ

18 Rock Organ

19 Church Organ

20 Reed Organ

21 Accordion

22 Harmonica

23 Tango Accordion

Guitar:

24 Acoustic Guitar (nylon)

25 Acoustic Guitar (steel)

26 Electric Guitar (jazz)

27 Electric Guitar (clean)

28 Electric Guitar (muted)

29 Overdriven Guitar

30 Distortion Guitar

31 Guitar harmonics

Bass:

32 Acoustic Bass

33 Electric Bass (finger)

34 Electric Bass (pick)

35 Fretless Bass

36 Slap Bass 1

37 Slap Bass 2

38 Synth Bass 1

39 Synth Bass 2

Strings:

40 Violin

41 Viola

42 Cello

43 Contrabass

44 Tremolo Strings

45 Pizzicato Strings

46 Orchestral Harp

47 Timpani

48 String Ensemble 1

49 String Ensemble 2

50 Synth Strings 1

51 Synth Strings 2

52 Choir Aahs

53 Voice Oohs

54 Synth Voice

55 Orchestra Hit

Brass:

56 Trumpet

57 Trombone

58 Tuba

59 Muted Trumpet

60 French Horn

61 Brass Section

62 Synth Brass 1

63 Synth Brass 2

Reed:

64 Soprano Sax

65 Alto Sax

66 Tenor Sax

67 Baritone Sax

68 Oboe

69 English Horn

70 Bassoon

71 Clarinet

Pipe:

72 Piccolo

73 Flute

74 Recorder

75 Pan Flute

76 Blown Bottle

77 Shakuhachi

78 Whistle

79 Ocarina

Synth Lead:

80 Lead 1 (square)

81 Lead 2 (sawtooth)

82 Lead 3 (calliope)

83 Lead 4 (chiff)

84 Lead 5 (charang)

85 Lead 6 (voice)

86 Lead 7 (fifths)

87 Lead 8 (bass + lead)

Synth Pad:

88 Pad 1 (new age)

89 Pad 2 (warm)

90 Pad 3 (polysynth)

91 Pad 4 (choir)

92 Pad 5 (bowed)

93 Pad 6 (metallic)

94 Pad 7 (halo)

95 Pad 8 (sweep)

Synth Effects:

96 FX 1 (rain)

97 FX 2 (soundtrack)

98 FX 3 (crystal)

99 FX 4 (atmosphere)

100 FX 5 (brightness)

101 FX 6 (goblins)

102 FX 7 (echoes)

103 FX 8 (sci-fi)

Ethnic:

104 Sitar

105 Banjo

106 Shamisen

107 Koto

108 Kalimba

109 Bag pipe

110 Fiddle

111 Shanai

Percussive:

112 Tinkle Bell

113 Agogo

114 Steel Drums

115 Woodblock

116 Taiko Drum

117 Melodic Tom

118 Synth Drum

Sound effects:

119 Reverse Cymbal

120 Guitar Fret Noise

121 Breath Noise

122 Seashore

123 Bird Tweet

124 Telephone Ring

125 Helicopter

126 Applause

127 Gunshot