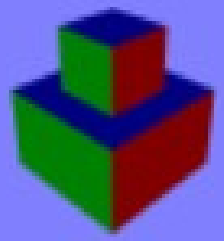
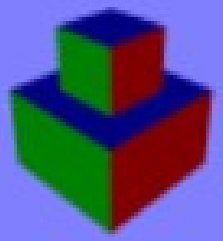
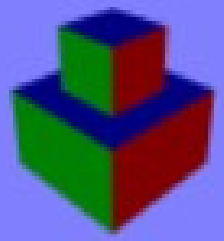
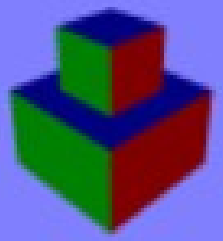




RCBASIC FOR BEGINNERS

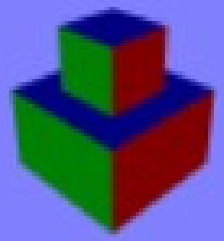


ABOUT

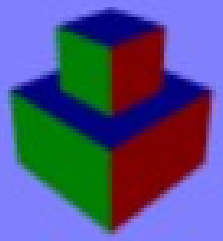


AUTHOR: Rodney Cunningham (AKA. N00b)

RCBasic is free open-source software distributed under the zlib license. More information and updates available at http://rcbasic.com or the forums at http://rcbasic.freeforums.net



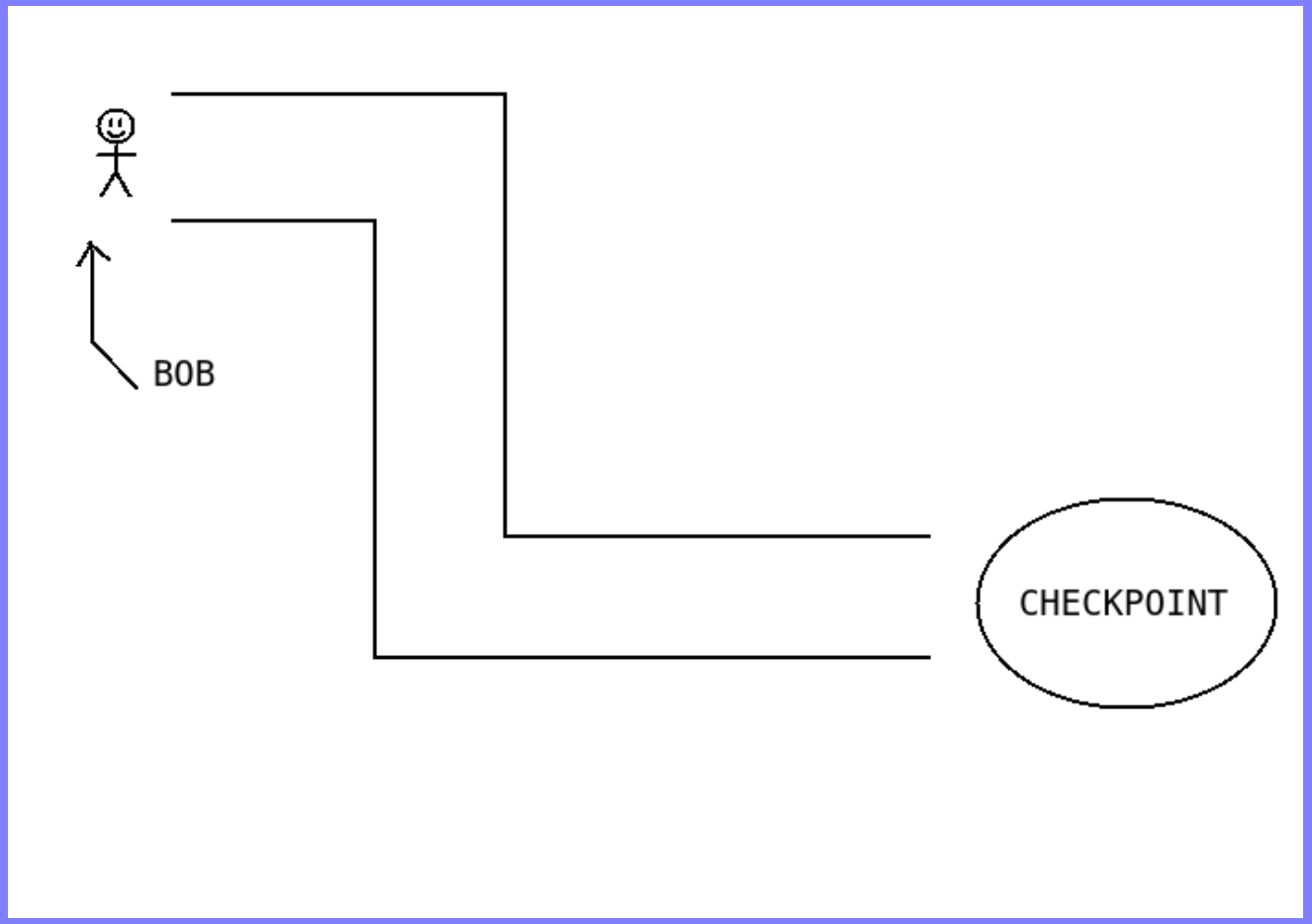
What is a Computer Program?



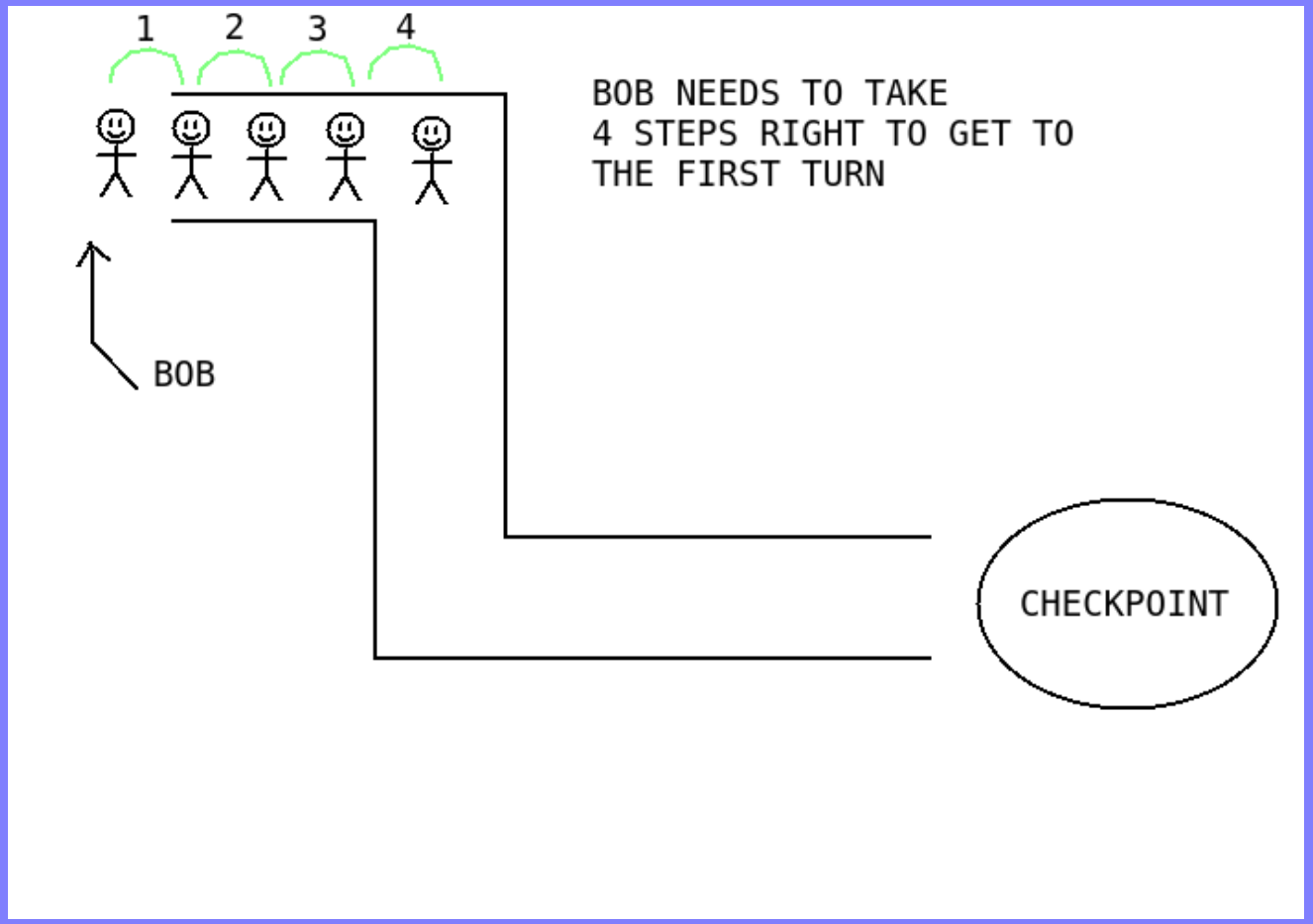
A **DETAILED** list of instructions on how to accomplish a specific task

DETAILED INSTRUCTIONS

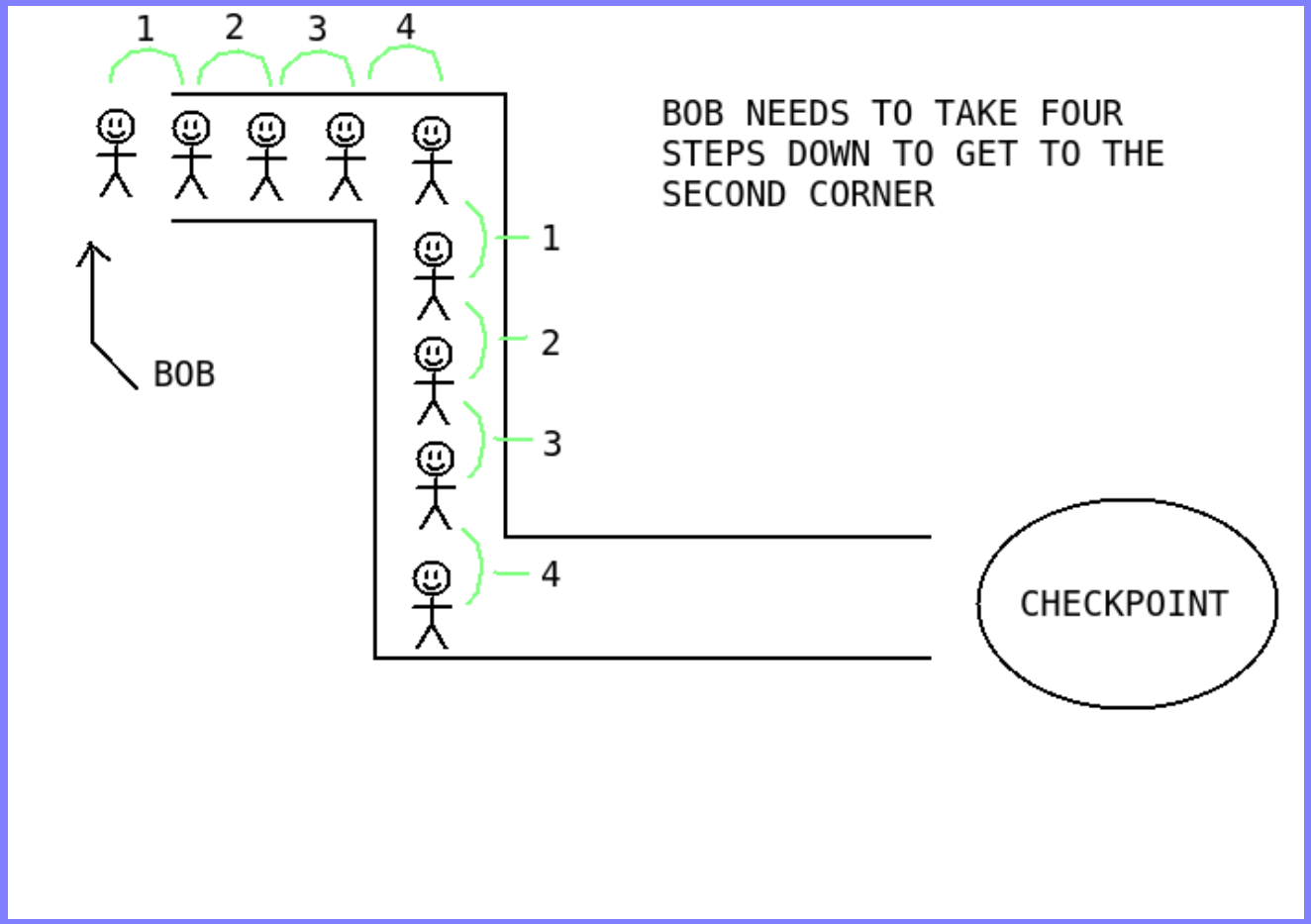
LETS GET BOB TO THE CHECKPOINT



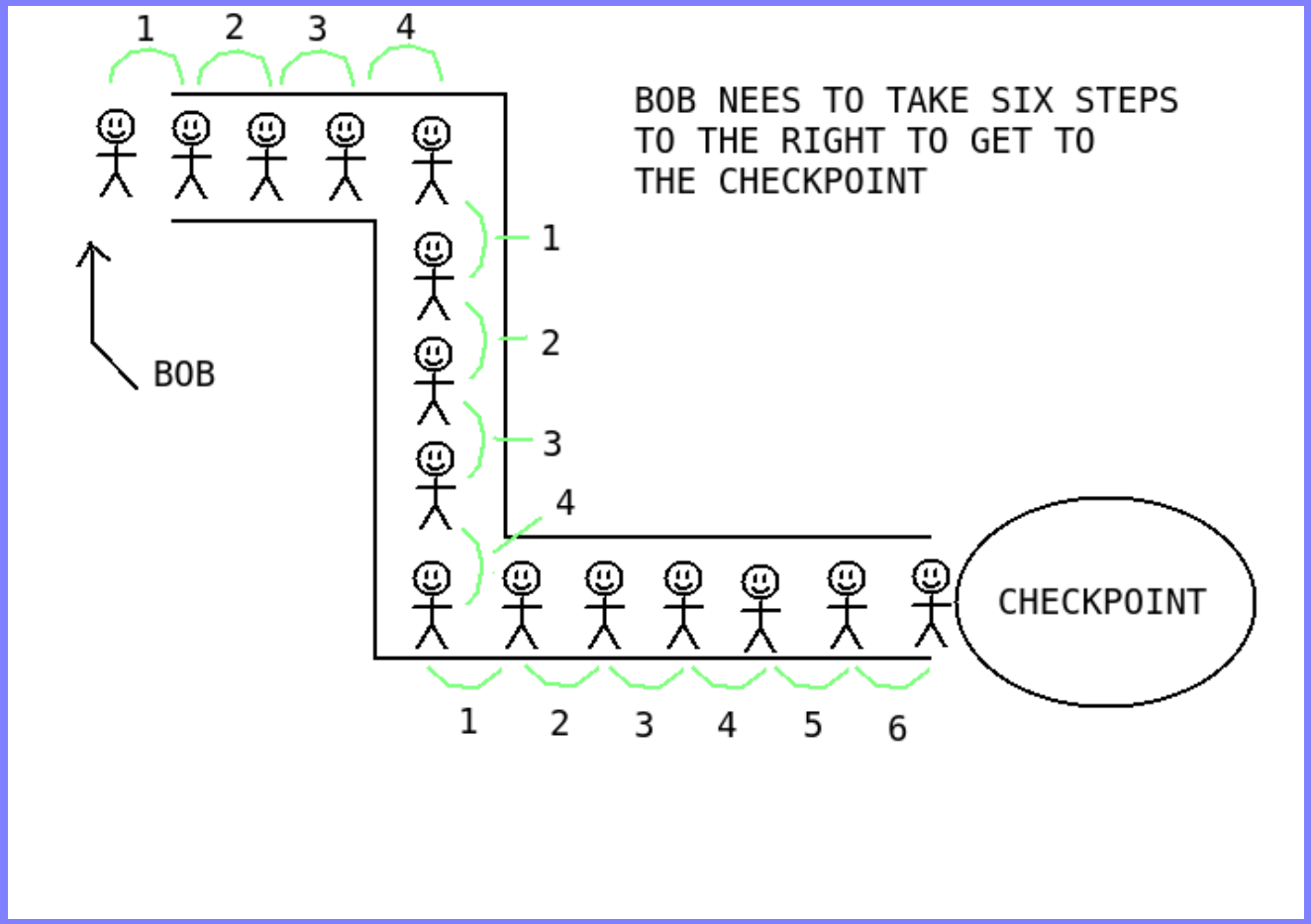
DETAILED INSTRUCTIONS

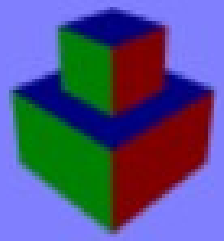


DETAILED INSTRUCTIONS

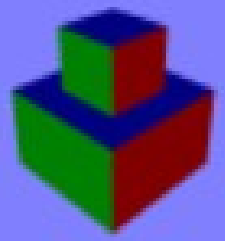


DETAILED INSTRUCTIONS





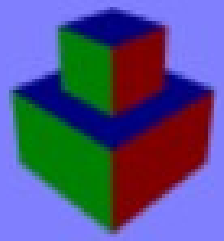
DETAILED INSTRUCTIONS



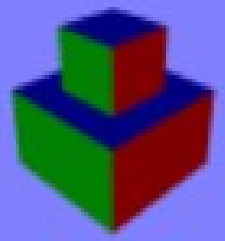
TO GET BOB TO THE CHECKPOINT WE CANT JUST TELL HIM TO GO THROUGH THE PATH. WE HAVE TO BE SPECIFIC IN WHAT INSTRUCTIONS WE GIVE HIM.

HERE IS THE LIST OF DETAILED INSTRUCTIONS TO GET BOB TO THE CHECKPOINT

- MOVE TO THE RIGHT 4 STEPS
- MOVE DOWN 4 STEPS
- MOVE TO THE RIGHT 6 STEPS



OUR FIRST PROGRAM

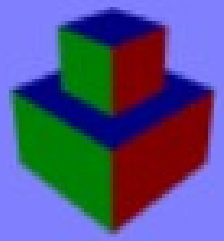


CODE:

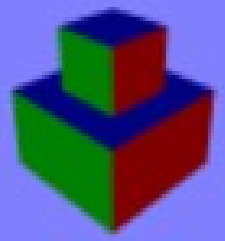
```
Print "Hello World"
```

OUTPUT:

```
Hello World
```



WHAT IS IN A INSTRUCTION?

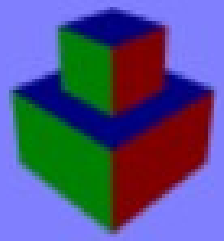


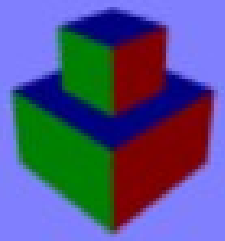
KEYWORDS: WORDS THAT ARE ASSOCIATED WITH PRE-DEFINED RULES OF A LANGUAGE

VARIABLES: A SINGLE CHARACTER OR COMBINATION OF CHARACTERS THAT A PROGRAMMER USES AS A SYMBOL TO REPRESENT DATA

FUNCTIONS: A SINGLE CHARACTER OR COMBINATION OF CHARACTERS THAT A PROGRAMMER USES TO RUN A SET BLOCK OF CODE

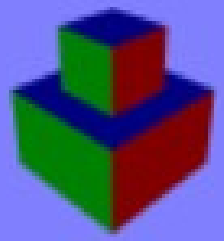
EXPRESSIONS: EITHER A SINGLE VALUE OR MULTIPLE VALUES THAT CAN BE MANIPULATED BY A NUMBER OF DIFFERENT OPERATORS, KEYWORDS, OR FUNCTIONS



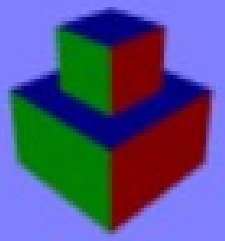


KEYWORDS

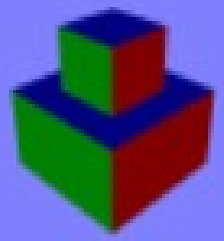
1. PRINT
2. AND
3. OR
4. XOR
5. NOT
6. MOD
7. SHL
8. SHR
9. IF
10. THEN
11. ELSEIF
12. ELSE
13. END
14. DIM
15. FUNCTION
16. SUB
17. RETURN
18. BYREF
19. SELECT
20. CASE
21. DEFAULT
22. INCLUDE
23. DELETE
24. REDIM
25. FOR
26. TO
27. STEP
28. NEXT
29. DO
30. LOOP
31. UNTIL
32. EXIT
33. WHILE
34. WEND



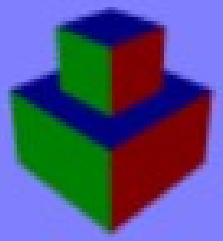
KEYWORDS



EVERY KEYWORD HAS ITS OWN SET OF RULES ON HOW, WHEN, AND WHERE IT CAN BE USED.



PRINT



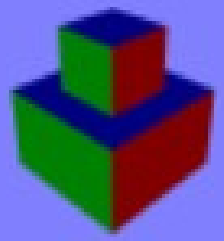
PRINT CAN OUTPUT TEXT AND NUMBERS

CODE:

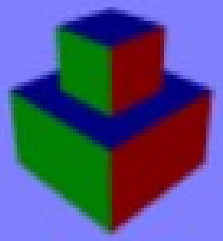
```
Print “The meaning of life is ”
Print 42
```

OUTPUT:

```
The meaning of life is
42
```



PRINT



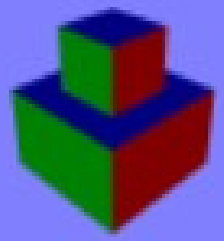
YOU CAN USE A SEMI-COLON TO OUTPUT MULTIPLE SETS OF DATA ON THE SAME LINE

CODE:

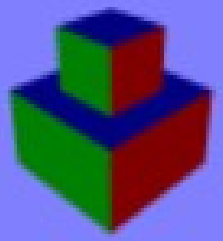
```
Print "The meaning of life is "; 42
Print "This is text. "; "This is more text."
```

OUTPUT:

```
The meaning of life is 42
This is text. This is more text.
```



PRINT



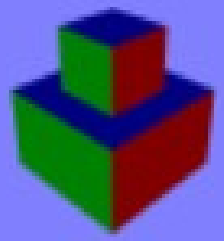
YOU CAN ALSO USE A SEMI-COLON AT THE END OF A PRINT STATEMENT TO STOP PRINT FROM AUTOMATICALLY GOING TO THE NEXT LINE

CODE:

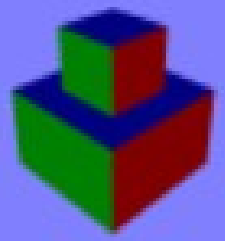
```
Print "The meaning of life is ";
Print 42
```

OUTPUT:

```
The meaning of life is 42
```



AND, OR, XOR

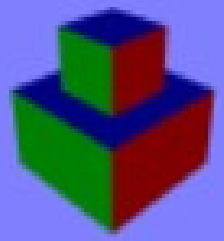


AND, OR, XOR – LOGICAL OPERATOR USED FOR COMPARISON

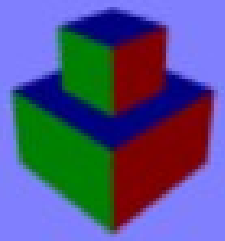
AND - [expression] AND [expression]

OR - [expression] OR [expression]

XOR - [expression] XOR [expression]



AND, OR, XOR

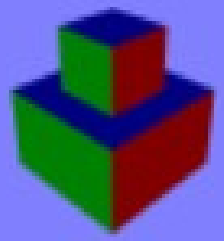


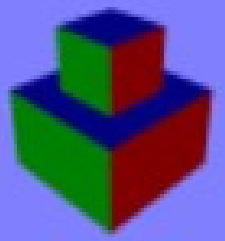
AND - IF LEFT VALUE IS **(NOT ZERO)** AND RIGHT VALUE IS **(NOT ZERO)**
THEN THE EXPRESSION IS TRUE

OR - IF LEFT VALUE IS **(NOT ZERO)** OR RIGHT VALUE IS **(NOT ZERO)**
THEN THE EXPRESSION IS TRUE

XOR - IF THE LEFT VALUE IS **(ZERO)** AND THE RIGHT VALUE IS **(NOT ZERO)**
THEN THE EXPRESSION IS TRUE

NOTE: XOR (EXCLUSIVE-OR) MEANS ONE OPERAND HAS TO BE ZERO AND
THE OPERAND HAS TO BE (NOT ZERO)



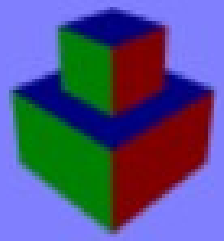


AND, OR, XOR

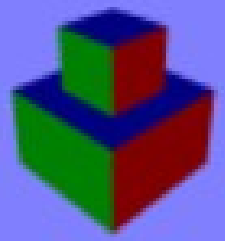
THIS TABLE DEMONSTRATES WHAT **AND**, **OR**, AND **XOR** WILL RETURN IF THEY ARE GIVEN **A** AS THE LEFT VALUE AND **B** AS THE RIGHT VALUE

TRUE = NOT ZERO (ANY VALUE OTHER THAN ZERO)
FALSE = ZERO

A	B	AND	OR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

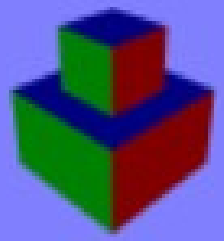


AND, OR, XOR

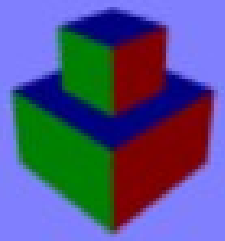


EXAMPLE:

```
PRINT  1  AND 0     '→ AND, OR, XOR EXPRESSIONS RETURN A NUMBER
                    '→ SO IT CAN BE USED ANYWHERE A NUMBER CAN

N = 1  OR  0   '→ THE VALUES OF THE EXPRESSIONS CAN BE STORED
               '→ CAN BE STORED IN A NUMBER VARIABLE


IF  ( 5 < 7 )  AND  ( 6 = 6 ) THEN   '→ THIS IS THE MOST COMMON
                                      '→ USE FOR AND, OR, XOR

   PRINT "TESTING AND"

END IF
```



IF-THEN BLOCK

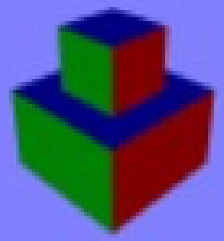


THE IF-THEN BLOCK IS A WAY OF HAVING SOMETHING IN YOUR PROGRAM HAPPEN IF A CERTAIN CONDITION IS MET

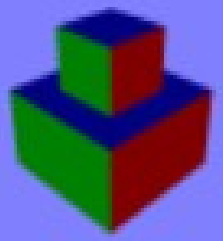
```
IF [expression] THEN

    ..CODE TO EXECUTE..

END IF
```

THE EXPRESSION MUST BE A MATH OR LOGICAL EXPRESSION.

*THE CODE INSIDE THE **IF-BLOCK** ONLY EXECUTES IF THE EXPRESSION EQUALS* ***(NOT ZERO)***



IF-THEN BLOCK



AN IF-BLOCK CAN TAKE ANY MATH OR LOGIC EXPRESSION.

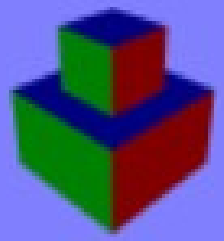
WE WILL GO OVER DIFFERENT EXPRESSIONS LATER.

CODE:

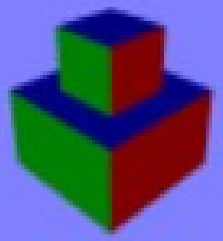
```
If 5 > 4 Then
   Print "The condition is true"
End If
```

OUTPUT:

```
The condition is true
```



IF-THEN BLOCK

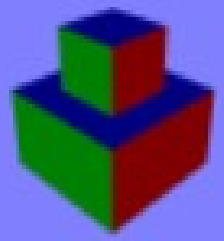


IF A LOGICAL EXPRESSION IS FALSE **(NOT ZERO)** THEN THE CODE INSIDE THE IF BLOCK IS NOT EXECUTED

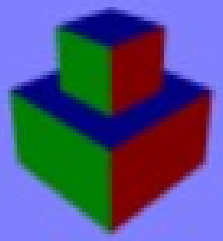
CODE:

```
If 5 < 4 Then
   Print "The condition is true"
End If
```

OUTPUT:

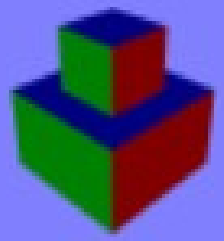


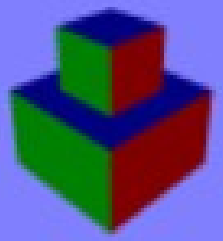
IF-THEN BLOCK



IF-BLOCKS CAN ALSO BE EXTENDED TO DO DIFFERENT THINGS IF DIFFERENT CONDITIONS ARE MET

TO CHECK FOR MULTIPLE CONDITIONS YOU WOULD USE THE **ELSEIF** AND **ELSE** KEYWORDS



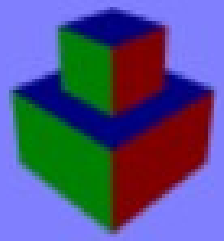


IF-THEN BLOCK

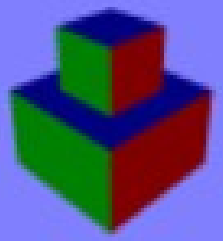
ELSEIF IS USED JUST LIKE **IF**. THE ONLY DIFFERENCE IS IT CAN ONLY BE USED IN A IF-BLOCK THAT WAS ALREADY STARTED WITH **IF**.

ELSE IS USED INSIDE OF AN ALREADY STARTED IF-BLOCK. ELSE CAN BE USED TO EXECUTE CODE IF THE **IF** AND ALL THE **ELSEIF**'S WERE FALSE.

NOTE: REMEMBER THAT **ELSEIF** AND **ELSE** ARE OPTIONAL. YOU CAN USE BOTH OR JUST ONE OR NOT USE ANY IF YOU DON'T NEED THEM.



IF-THEN BLOCK

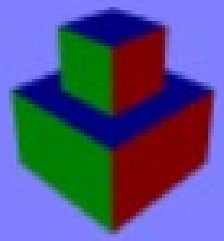


CODE:

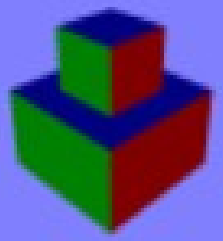
```
If 5 < 4 Then
   Print “FIRST CONDITION”
ElseIf 2+2 = 4 Then
   Print “SECOND CONDITION”
Else
   Print “THIRD CONDITION”
End If
```

OUTPUT:

```
SECOND CONDITION
```



IF-THEN BLOCK



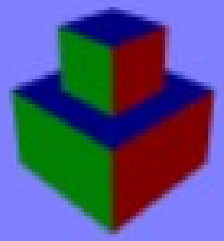
IN THIS EXAMPLE WE ADDED THE OR OPERATOR TO COMBINE MULTIPLE CONDITIONS INTO ONE CONDITION

CODE:

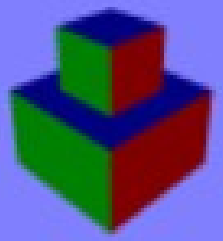
```
If 5 < 4 OR 2*2 = 4 Then
    Print "FIRST CONDITION"
ElseIf 2+2 = 4 Then
    Print "SECOND CONDITION"
Else
    Print "THIRD CONDITION"
End If
```

OUTPUT:

```
FIRST CONDITION
```



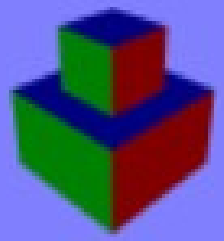
DATA TYPES AND EXPRESSIONS



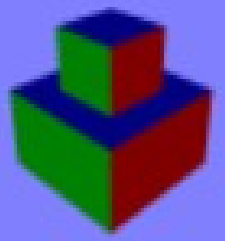
DATA TYPE – THE CATEGORY A VALUE IS CLASSIFIED AS

RCBASIC SUPPORTS TWO DATA TYPES

- NUMBERS
- STRINGS



DATA TYPES AND EXPRESSIONS

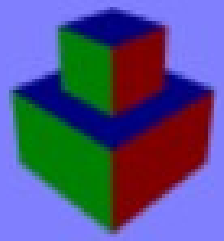


NUMBERS

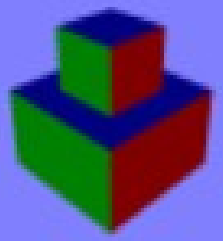
THIS SHOULD BE PAINFULLY OBVIOUS

EXAMPLE:

```
1
2.333
-5
-4.6
```



DATA TYPES AND EXPRESSIONS

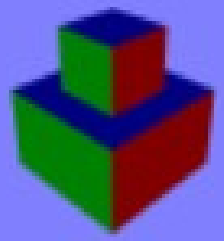


STRINGS

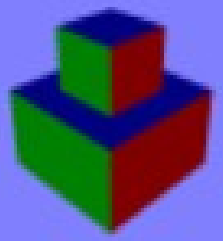
- A SET OF CHARACTERS
- **RAW STRING** DATA IS ENCLOSED IN “ “

EXAMPLE:

```
“HELLO WORLD”
“12”
“SFJLKD”
“\q”
```



DATA TYPES AND EXPRESSIONS

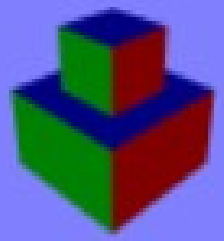


EXPRESSIONS

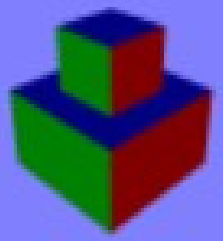
AN OPERATION THAT CAN COMBINE AND MANIPULATE DATA

ALL VALUES IN A EXPRESSION MUST BE THE SAME DATA TYPE

(BASICALLY YOU CAN ONLY DO MATH WITH NUMBERS AND YOU CAN ONLY ADD STRINGS TO STRINGS)



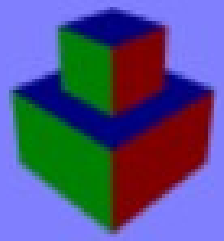
DATA TYPES AND EXPRESSIONS



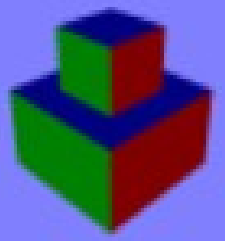
EXPRESSIONS CONSIST OF DATA AND OPERATORS

EXPRESSIONS RETURN EITHER A NUMBER OR A STRING

MULTIPLE EXPRESSIONS OF DIFFERENT TYPES CAN BE IN A SINGLE INSTRUCTION



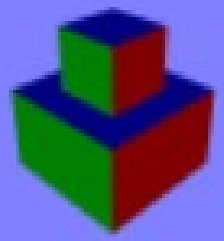
DATA TYPES AND EXPRESSIONS



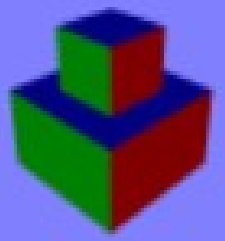
NUMBER EXPRESSIONS AKA **MATH**

OPERATORS:

+	ADDITION	5 + 3 = 8
-	SUBTRACTION	5 – 3 = 2
*	MULTIPLICATION	5 * 3 = 15
/	DIVISION	5 / 3 = 1.66667
^	RAISE TO POWER	5 ^ 3 = 125
MOD	MODULOUS (REMAINDER)	5 MOD 3 = 2
SHL	SHIFT LEFT	1 SHL 3 = 8
SHR	SHIFT RIGHT	8 SHR 3 = 1



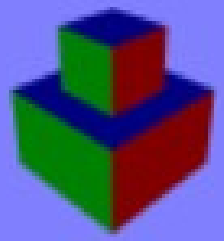
DATA TYPES AND EXPRESSIONS



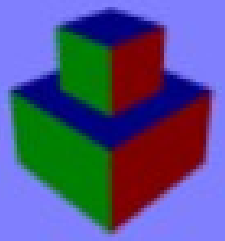
NUMBER EXPRESSIONS AKA **MATH**

OPERATORS:

AND	LOGICAL AND	5 AND 3 = 1 (TRUE)
OR	LOGICAL OR	5 OR 0 = 1 (TRUE)
XOR	LOGICAL XOR	5 XOR 3 = 0 (FALSE)
NOT	NOT	NOT 0 = 1 (TRUE)



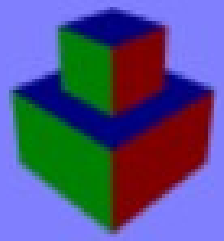
DATA TYPES AND EXPRESSIONS



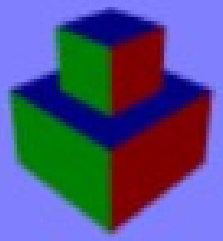
NUMBER EXPRESSIONS AKA **MATH**

OPERATORS:

=	EQUAL	5 = 3 = 0 (FALSE)
<>	NOT EQUAL	5 <> 0 = 1 (TRUE)
>	GREATER THAN	5 > 3 = 1 (TRUE)
>=	GREATER OR EQUAL	3 >= 3 = 1 (TRUE)
<	LESS THAN	5 < 5 = 0 (FALSE)
<=	LESS OR EQUAL	5 <= 5 = 1 (TRUE)



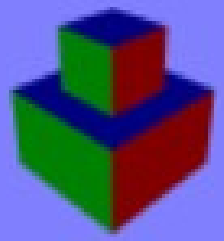
DATA TYPES AND EXPRESSIONS



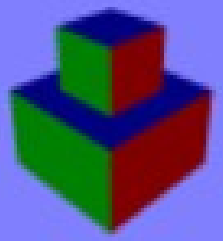
NUMBER EXPRESSIONS AKA **MATH**

ORDER OF OPERATIONS:

1. PARENTHESIS ( )
2. FUNCTIONS / SUB ROUTINES
3. NOT
4. EXPONENTS
5. MULTIPLICATION / DIVISION
6. ADDITION / SUBTRACTION
7. BIT SHIFT
8. COMPARISON (>, >=, <, <= , =, <>)
9. AND / OR / XOR



DATA TYPES AND EXPRESSIONS



NUMBER EXPRESSIONS AKA **MATH**

EXAMPLE:

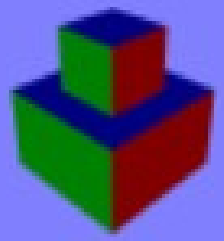
EXPRESSION 1: 5 **+** 3

EXPRESSION 2: (33.4 **–** 0.1) ***** 3

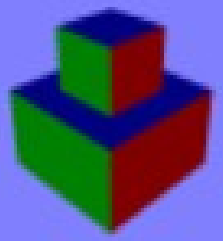
EXPRESSION 3: 4**^**2

EXPRESSION 4: **NOT** 3 **+** 2

EXPRESSION 5: 5 **<** 2



DATA TYPES AND EXPRESSIONS



NUMBER EXPRESSIONS AKA **MATH**

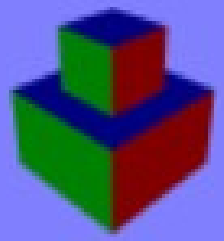
TO SEE THE VALUE OF THE EXAMPLE EXPRESSIONS WE CAN JUST OUTPUT THEM TO THE SCREEN WITH **PRINT**

CODE:

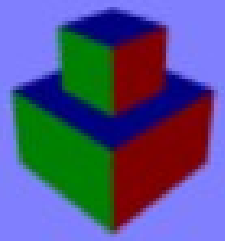
```
PRINT 4 < (2 * 3)
```

OUTPUT:

```
1
```



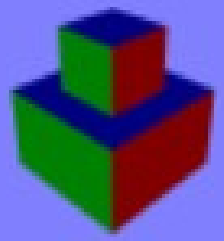
DATA TYPES AND EXPRESSIONS



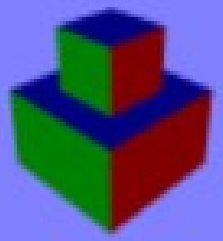
STRING EXPRESSIONS

OPERATORS:

+	APPEND	“12” + “34” = “1234”
=	EQUAL	“AB” = “AB” = 1 (TRUE)
<>	NOT EQUAL	“AB” <> “AA” = 1 (TRUE)



DATA TYPES AND EXPRESSIONS



STRING EXPRESSIONS

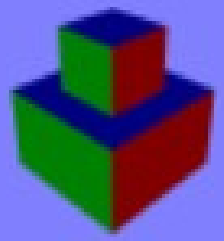
EXAMPLE:

EXPRESSION 1: “HELLO” **+** “WORLD”

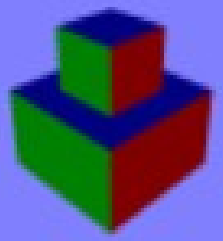
EXPRESSION 2: “THIS” = “THAT”

EXPRESSION 3: “THIS” <> “THAT”

EXPRESSION 4: “THIS” + “ IS A “ + “ STRING”



DATA TYPES AND EXPRESSIONS



STRING EXPRESSIONS

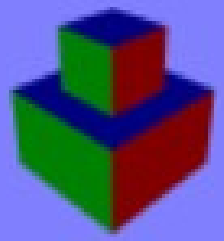
TO SEE THE VALUE OF THE EXAMPLE EXPRESSIONS WE CAN JUST OUTPUT THEM TO THE SCREEN WITH **PRINT**

CODE:

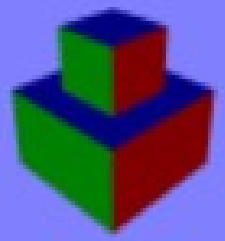
```
PRINT "HELLO" + "WORLD"
PRINT "HELLO " + "WORLD"
PRINT "HELLO" = "WORLD"     '---> THIS IS A LOGIC EXPRESSION
                            '---> LOGIC EXPRESSIONS ARE 0 IF FALSE
                            '---> AND 1 IF TRUE
```

OUTPUT:

```
HELLOWORLD
HELLO WORLD
0
```



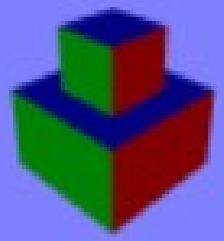
LOGIC EXPRESSION AND FLOW CONTROL



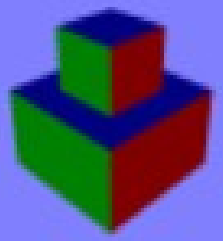
WHAT CAN WE DO WITH LOGIC EXPRESSIONS?

LOGIC EXPRESSIONS CAN BE USED AS CONDITIONAL TRIGGERS FOR FLOW CONTROL STATEMENTS AND LOOPS

(THIS WILL MAKE MORE SINCE AS WE USE THEM)

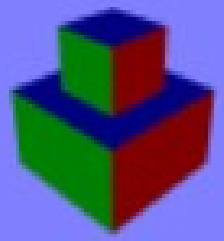


LOGIC EXPRESSION AND FLOW CONTROL

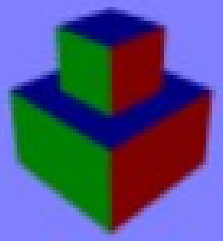


EXAMPLE:

```
IF "THIS" <> "THAT" THEN
    PRINT "THIS IS NOT EQUAL TO THAT"
END IF

IF 3 + 4 > 5 THEN
    PRINT "SEVEN GREATER THAN FIVE"
END IF
```

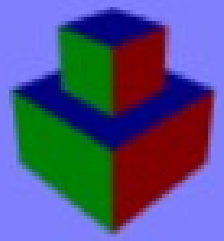


LOGIC EXPRESSION AND FLOW CONTROL

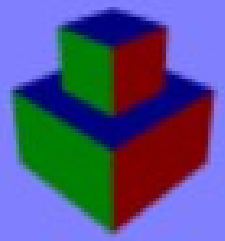


<u>THE SELECT BLOCK:</u>

SELECT IS A WAY OF DOING A COMPARISON AND TRIGGERING A BLOCK OF CODE DEPENDING ON THE VALUE BEING COMPARED



LOGIC EXPRESSION AND FLOW CONTROL



SELECT CASE [EXPRESSION MAIN]

CASE [EXPRESSION A]

...CODE TO EXECUTE..

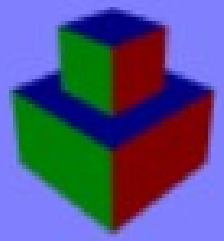
CASE [EXPRESSION ETC.]

...CODE TO EXECUTE..

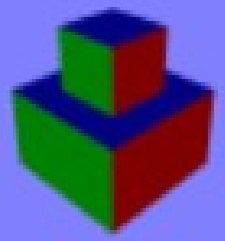
DEFAULT

...CODE TO EXECUTE

END SELECT



LOGIC EXPRESSION AND FLOW CONTROL

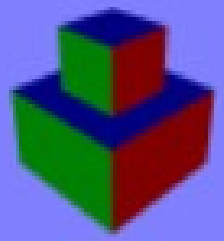


CODE:

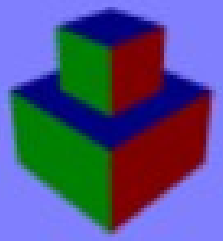
```
SELECT CASE 5     '--> SINCE MAIN CASE IS 5 IT WILL EXECUTE THE
                  '--> THE CODE FOR CASE 5
CASE 2
    PRINT "MAIN CASE IS 2"
CASE 7
    PRINT "MAIN CASE IS 7"
CASE 5
    PRINT "THIS IS WHAT WILL PRINT: "; 2+2
END SELECT
```

OUTPUT:

```
THIS IS WHAT WILL PRINT: 4
```



LOGIC EXPRESSION AND FLOW CONTROL

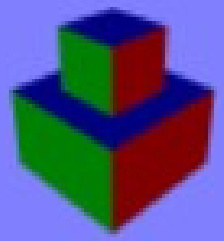


CODE:

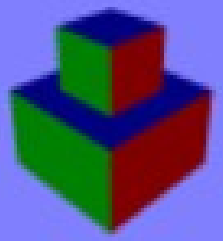
```
SELECT CASE 5.2    '--> SINCE MAIN CASE IS 5 IT WILL EXECUTE THE
                   '--> THE CODE FOR CASE 5
CASE 2
    PRINT "MAIN CASE IS 2"
CASE 7
    PRINT "MAIN CASE IS 7"
CASE 5
    PRINT "THIS IS WHAT WILL PRINT: "; 2+2
DEFAULT
    PRINT "THIS PRINTS IF ALL OTHER CASES ARE FALSE"
END SELECT
```

OUTPUT:

```
THIS PRINTS IF ALL OTHER CASES ARE FALSE
```



VARIABLES



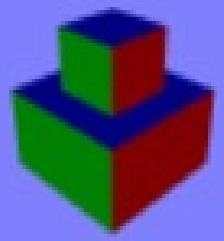
VARIABLES ARE SYMBOLS THAT WE CAN MAKE TO REPRESENT DATA

VARIABLES MUST START WITH A LETTER

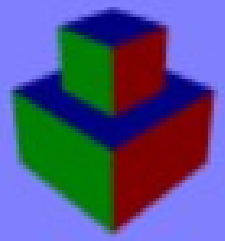
VARIABLES ARE NOT CASE SENSITIVE

VARIABLES CANNOT BE NAMED ANY KEYWORD OR FUNCTION THAT IS ALREADY IN RCBASIC

VARIABLES CAN BE EITHER A NUMBER OR A STRING



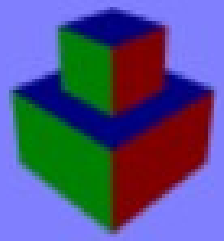
VARIABLES

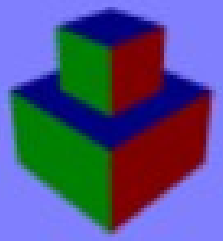


WHY DO WE NEED VARIABLES?

IN ANY PROGRAM DATA HAS TO BE STORED SO IT CAN BE USED LATER

- **IN A GAME YOU WOULD NEED TO KEEP TRACK OF SCORE, SPEED, POSITION, ETC.**
- **IN AN ADDRESS BOOK YOU WOULD NEED TO STORE NAMES, ADDRESSES, ETC.**
- **IN A SIMPLE QUIZ, YOU WOULD WANT TO STORE QUESTIONS AND ANSWERS**





VARIABLES

HOW DO WE MAKE A VARIABLE?

NUMBER VARIALBE:

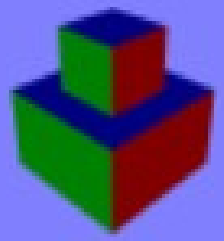
SCORE = 0

STRING VARIABLE:

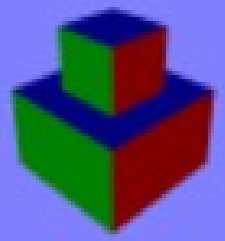
NAME$ = “BOB” **‘→ WHEN MAKING A STRING VARIABLE IN**

‘→ RCBASIC YOU MUST PUT A “$” AT THE END

‘→ WHEN YOU FIRST DECLARE YOUR VARIABLE



VARIABLES



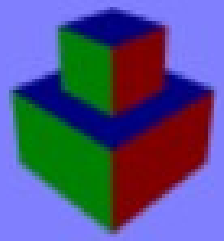
HOW DO WE USE A VARIABLE?

"=" ASSIGNS A VALUE TO A VARIABLE

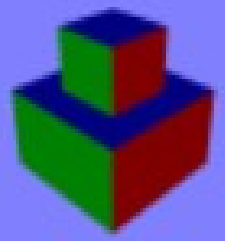
VALUE CAN BE ANY EXPRESSION FOR THE VARIABLES DATA TYPE

EXAMPLE:

```
SCORE = 10 + 5 * 2     'SCORE IS NOW 20 (DO THE MATH)
SCORE = SCORE + 1  'SCORE IS NOW 21

A$ = "HELLO "    'A IS NOW "HELLO "
A = A + "WORLD"   'A IS NOW "HELLO WORLD"
A = "TREE"  'A IS NOW "TREE"  SINCE WE DID NOT ADD TO A WE REPLACED
                  'ITS VALUE WITH "TREE"
```



VARIABLES



VARIABLES CAN BE USES ANYWHERE THAT THE DATA IN THEM CAN?

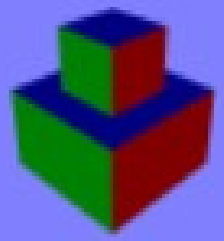
EXAMPLE:

INSTEAD OF

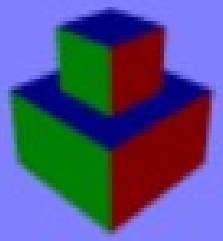
```
PRINT "HELLO WORLD"
```

WE CAN DO THIS

```
HW$ = "HELLO WORLD"
PRINT HW
```



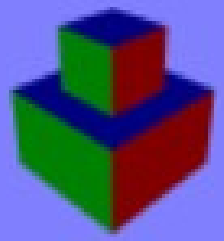
VARIABLES



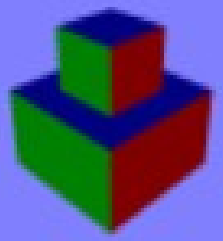
VARIABLES CAN BE USES ANYWHERE THAT THE DATA IN THEM CAN?

EXAMPLE:

```
X = 5
IF X > 2 THEN
    PRINT "VARIABLE HAS A VALUE OF ";X
END IF
```

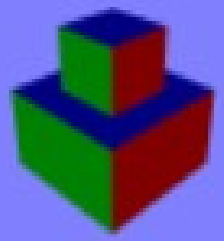


FUNCTIONS AND INPUT

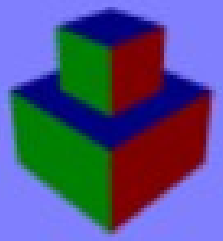


FOR A PROGRAM TO TRULY BE USEFUL WE NEED TO BE ABLE TO GET INPUT FROM THE USER AND DO STUFF BASED ON THAT INPUT

RCBASIC ALLOWS YOU TO GET INPUT TEXT INPUT FROM THE USER WITH THE **INPUT$( )** FUNCTION



FUNCTIONS AND INPUT

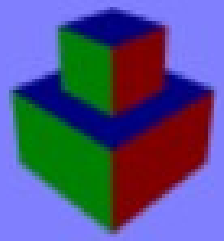


HOW THE INPUT FUNCTION WORKS

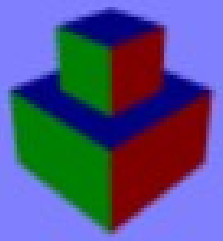
INPUT$ (PROMPT$) - PROMPT IS TEXT THAT THE USER SEES BEFORE THEY START ENTERING INPUT

EXAMPLE:

```
Input$ ("ENTER YOUR NAME:")
```

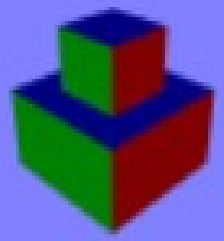


FUNCTIONS AND INPUT

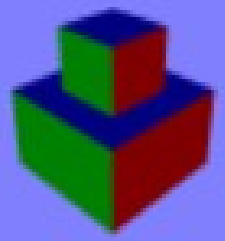


TO BE ABLE TO USE THE INPUT FROM THE USER LATER ON IN OUR PROGRAM WE HAVE TO SAVE THE INPUT TO A VARIABLE

BEFORE WE CAN DO THIS WE HAVE TO UNDERSTAND A LITTLE BIT MORE ABOUT FUNCTIONS



FUNCTIONS AND INPUT



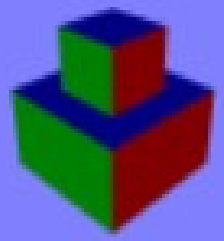
WHAT IS A FUNCTION?

A FUNCTION IS A SYMBOL THAT CAN BE USED TO EXECUTE A BLOCK OF CODE

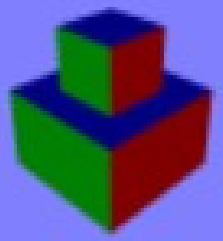
NAMIMG RULES FOR FUNCTIONS ARE THE SAME AS VARIABLES

FUNCTIONS CAN RETURN NUMBERS OR STRINGS JUST LIKE VARIABLES

FUNCTIONS CAN TAKE PARAMETERS THAT CAN EFFECT THE EXECUTION OF THE FUNCTION



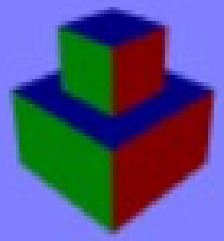
FUNCTIONS AND INPUT



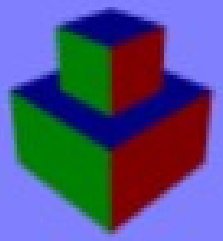
LETS EXAMINE THE **INPUT$**( ) FUNCTION

IN THE MANUAL YOU WILL NOTICE THAT **INPUT$** ( ) HAS A “**$**” ADDED TO ITS NAME

THE “**$**” TELLS US THAT IT RETURNS A STRING



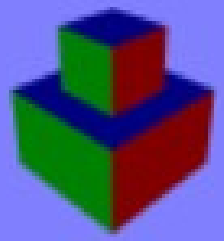
FUNCTIONS AND INPUT



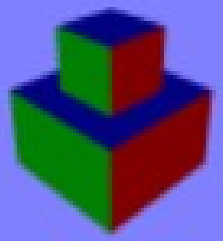
SINCE INPUT$ ( ) RETURNS A STRING VALUE WE MUST STORE THE RESULT IN A STRING VARIABLE

EXAMPLE:

```
name$ = Input ("Enter your name: ")
```



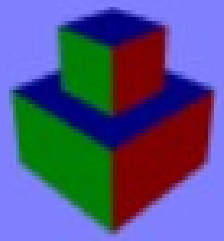
FUNCTIONS AND INPUT



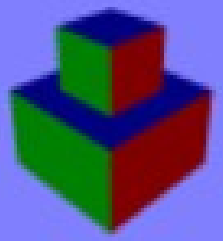
SINCE INPUT$ ( ) RETURNS A STRING VALUE WE MUST STORE THE RESULT IN A STRING VARIABLE

EXAMPLE:

```
name$ = Input ("Enter your name: ")
```



FUNCTIONS AND INPUT

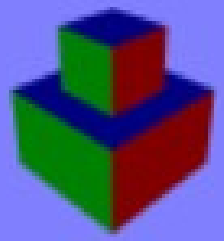


CODE:

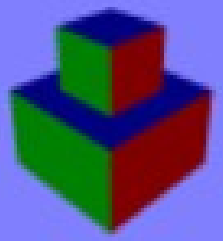
```
Name$ = Input ("Enter your name: ")

Print "Hello "; Name$; ", Welcome to RCBasic"
```

OUTPUT: **RED characters are user input**

```
Enter your name: n00b
Hello n00b, Welcome to RCBasic
```



FUNCTIONS AND INPUT

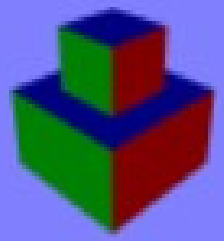


CODE:

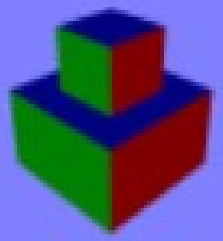
```
Name$ = Input ("Enter your name: ")

Print "Hello "; Name$; ", Welcome to RCBasic"
```

OUTPUT: **RED characters are user input**

```
Enter your name: n00b
Hello n00b, Welcome to RCBasic
```



FUNCTIONS AND INPUT

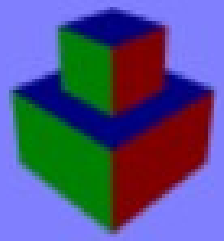


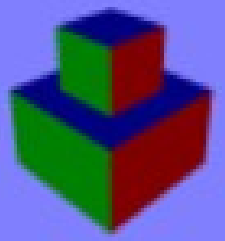
IF INPUT$ ( ) ONLY RETURNS A STRING HOW CAN WE GET A NUMBER FROM THE USER?

RCBASIC HAS LOTS OF BUILT-IN FUNCTIONS THAT PERFORMS LOTS OF DIFFERENT TASK

TO CONVERT THE USER INPUT TO A NUMBER WE CAN USE ANOTHER BUILT-IN FUNCTION

VAL ( )





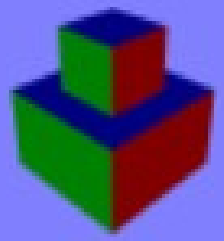
FUNCTIONS AND INPUT

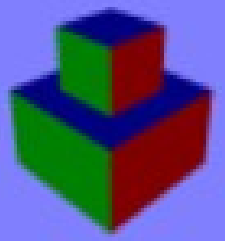
HOW THE VAL ( ) FUNCTION WORKS

VAL (STRING_TO_CONVERT$)

EXAMPLE:

```
x = Val ( “5” )
Print x + 5
```





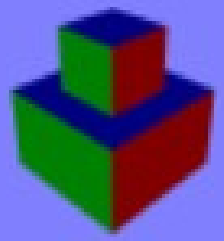
FUNCTIONS AND INPUT

HOW THE VAL ( ) FUNCTION WORKS

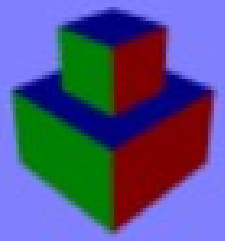
VAL (STRING_TO_CONVERT$)

EXAMPLE:

```
x = Val ( “5” )
Print x + 5
```



FUNCTIONS AND INPUT

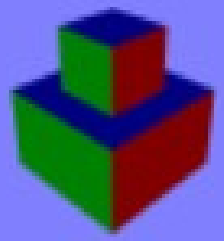


CODE:

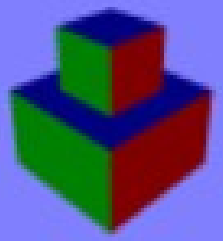
```
Age_string$ = Input ("Enter your age: ")

Age = Val ( Age_string )

Print "In 5 years you will be "; Age + 5
```

OUTPUT: **RED characters are user input**

```
Enter your age: 30
In 5 years you will be 35
```



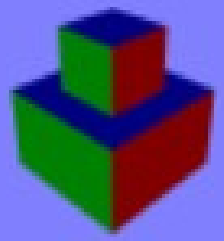
FUNCTIONS AND INPUT



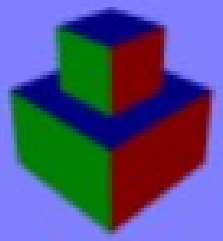
RCBASIC HAS SEVERAL BUILT-IN FUNCTIONS WHICH CAN HELP YOU WITH MANY DIFFERENT TASK YOU WILL HAVE TO PERFORM IN YOUR PROGRAM

BUT SOMETIMES YOU NEED A FUNCTION TO PERFORM A TASK THAT IS NOT BUILT-IN TO RCBASIC

IN THESE CASES YOU CAN BUILD YOUR OWN FUNCTIONS



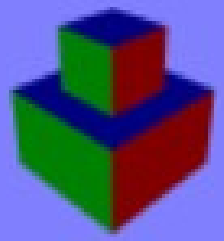
FUNCTIONS AND INPUT



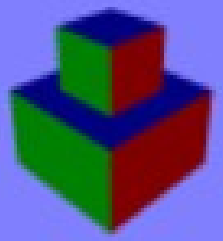
HOW TO MAKE OUR OWN FUNCTION

TO MAKE YOUR OWN FUNCTION YOU WILL USE THE **FUNCTION** KEYWORD

```
FUNCTION  FUNC_NAME ( PARAMETERS )
   ...CODE TO EXECUTE…
   RETURN  VALUE
END FUNCTION
```



FUNCTIONS AND INPUT

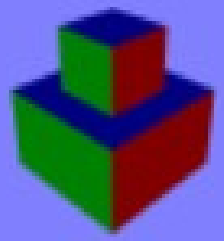


NUMBER FUNCTION EXAMPLE:

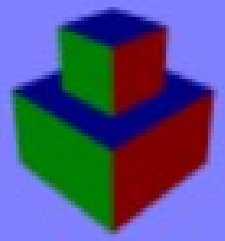
```
Function  MyFirstFunction ( )
      return 99   '→Must return a number
End Function
```

STRING FUNCTION EXPAMPLE:

```
Function  MySecondFunction$ ( )  '→Must have "$"
      return "this is a string"  '→Must return string
End Function
```



FUNCTIONS AND INPUT

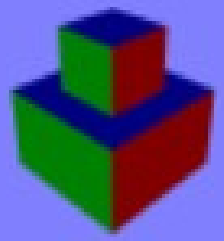


NUMBER FUNCTION EXAMPLE:

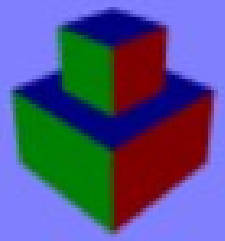
```
Function  MyFirstFunction ( num1, num2 )
      num3 = num1 + num2
      return num3   '→Must return a number
End Function
```

STRING FUNCTION EXPAMPLE:

```
Function  MySecondFunction$ ( t$ )  '→Must have "$"
      r$ = "t is " + t
      return r '→Must return string
End Function
```



FUNCTIONS AND INPUT

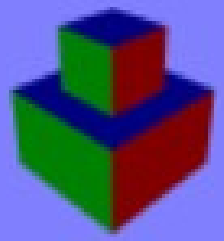


CODE:

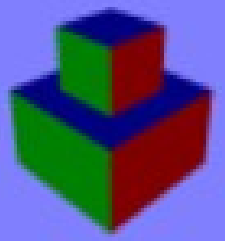
```
Function AddNumbers ( a, b )
     Return a + b
End Function

Print AddNumbers ( 4, 3 )
```

OUTPUT:

```
7
```

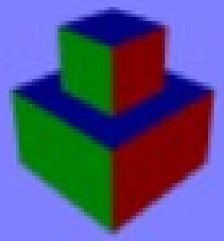


LOOPS

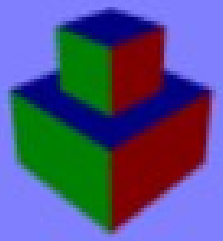


WHAT IS A LOOP?

A BLOCK OF CODE THAT REPEATS



LOOPS

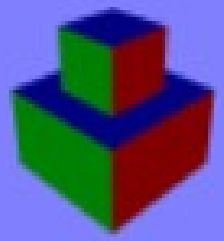


TYPES OF LOOPS

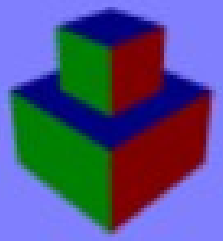
WHILE – EXECUTES A BLOCK OF CODE
WHILE A CONDITION IS TRUE

FOR – EXECUTES A LOOP UNTIL A COUNTER
REACHES A SPECIFIED VALUE

DO – LIKE A WHILE LOOP BUT IT WILL ALWAYS EXECUTE
ATLEAST ONCE



LOOPS

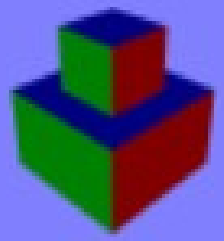


WHILE LOOP:

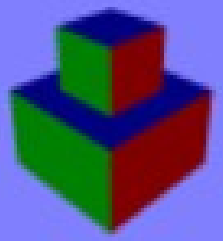
WHILE [expression]

...CODE TO EXECUTE…

WEND



LOOPS

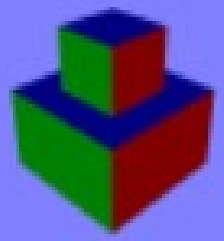


EXAMPLE:

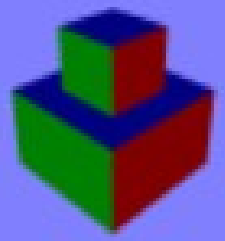
```
X = 5
WHILE  X  >  0   '→ runs all code between this line
                 '→ and wend while x is greater than 0
     PRINT  X
     X  =  X - 1   '→ set x to x's current value minus 1
WEND   '→ goes back to start of loop
```

OUTPUT:

```
5
4
3
2
1
```



LOOPS



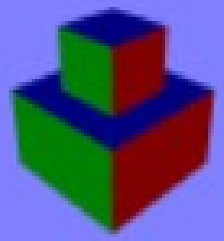
FOR LOOP:

START VALUE END VALUE

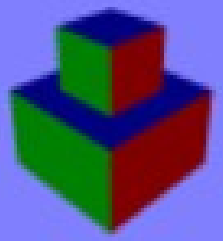
FOR *variable* = [expression] **TO** [expression]

...CODE TO EXECUTE…

NEXT



LOOPS

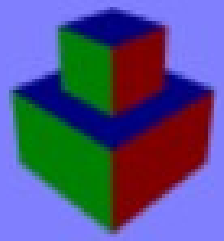


EXAMPLE:

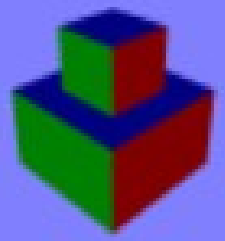
```
FOR I = 0 TO 4  '→ I will increase by 1 until its 4
      PRINT I
NEXT   '→ Goes back to start of FOR loop
```

OUTPUT:

```
0
1
2
3
4
```



LOOPS



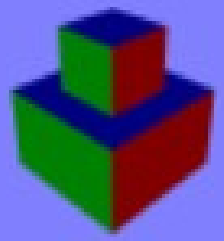
FOR LOOP:

START VALUE END VALUE

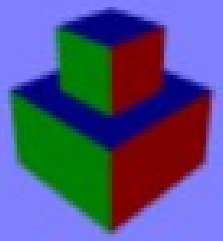
FOR *variable* = [expression] **TO** [expression] **STEP** [expression]

...CODE TO EXECUTE…

NEXT



LOOPS

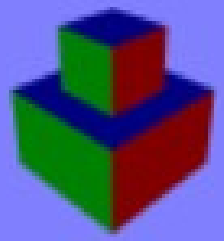


EXAMPLE:

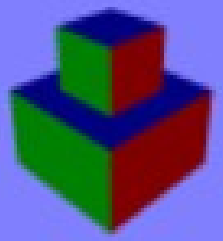
```
FOR I = 0 TO 8 STEP 2 '→ I will increase by 2 until its 8
      PRINT I
NEXT   '→ Goes back to start of FOR loop
```

OUTPUT:

```
0
2
4
6
8
```



LOOPS

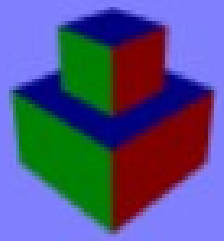


DO LOOP:

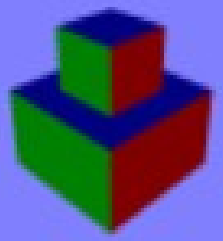
DO

...CODE TO EXECUTE…

LOOP



LOOPS

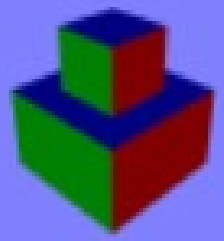


EXAMPLE:

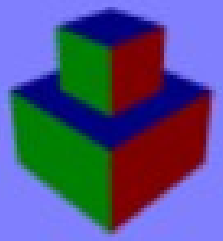
```
DO
    PRINT "THIS WILL PRINT INFINITELY"
LOOP   '→ GOES BACK TO THE START OF THE DO LOOP
```

OUTPUT:

```
THIS WILL PRINT INFINITELY
THIS WILL PRINT INFINITELY
THIS WILL PRINT INFINITELY
THIS WILL PRINT INFINITELY
...
```



LOOPS

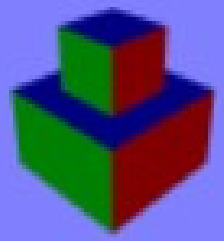


DO LOOP:

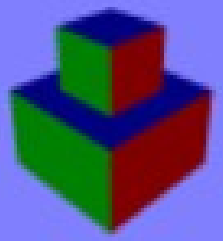
DO

...CODE TO EXECUTE…

LOOP WHILE [expression]



LOOPS

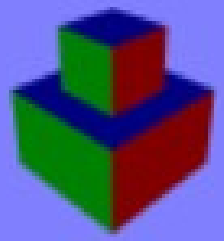


EXAMPLE:

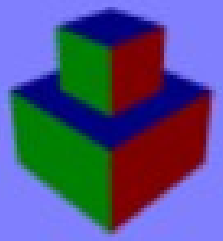
```
I = 5
DO
   PRINT I
   I = I - 1
LOOP WHILE I > 0   '→ GOES BACK TO THE START OF THE DO LOOP
                   '→ IF I IS GREATER THAN 0
```

OUTPUT:

```
5
4
3
2
1
```



LOOPS

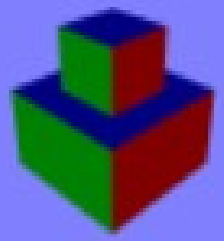


DO LOOP:

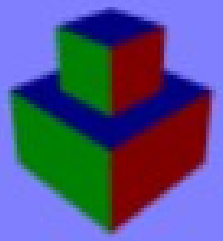
DO

...CODE TO EXECUTE…

LOOP UNTIL [expression]



LOOPS

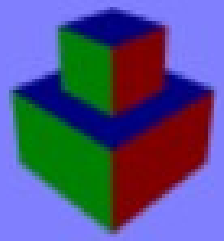


EXAMPLE:

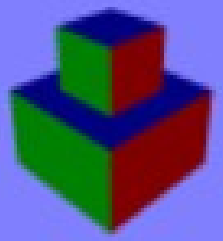
```
I = 0
DO
   PRINT I
   I = I + 1
LOOP UNTIL I = 5   '→ GOES BACK TO THE START OF THE DO LOOP
                   '→ IF I IS NOT EQUAL TO 0
```

OUTPUT:

```
0
1
2
3
4
```



LOOPS

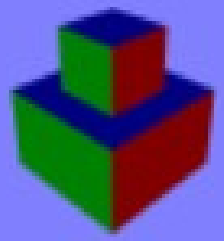


EXAMPLE:

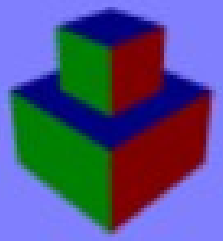
```
I = 0
DO
   PRINT I
   I = I + 1
LOOP UNTIL I = 5   '→ GOES BACK TO THE START OF THE DO LOOP
                   '→ IF I IS NOT EQUAL TO 0
```

OUTPUT:

```
0
1
2
3
4
```



LOOPS

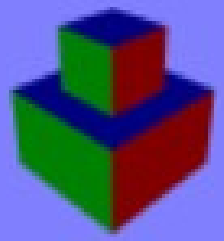


HOW TO EXIT FROM A LOOP BEFORE ITS CONDITION IS MET

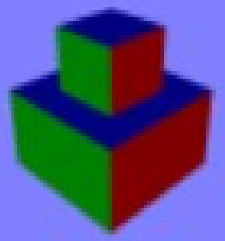
EXIT FOR

EXIT WHILE

EXIT DO



LOOPS

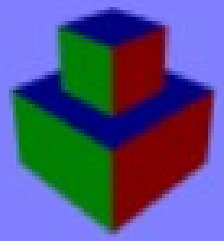


EXAMPLE:

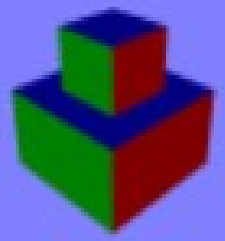
```
I = 5
DO
   PRINT I
   I = I - 1
   IF I = 3 THEN
      EXIT DO    '→ This will end the DO loop and start executing on the first
                 '→ line after the end of the loop block
   END IF
LOOP UNTIL I <> 0
```

OUTPUT:

```
5
4
```

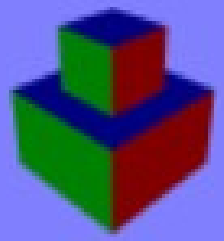


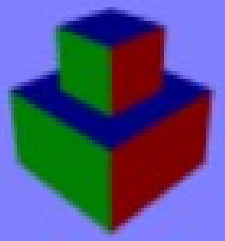
ARRAYS



ARRAYS ARE VARIABLES THAT CAN STORE MORE THAN ONE VALUE

RCBASIC SUPPORTS 1 DIMENSIONAL UP TO 3 DIMENSIONAL ARRAYS

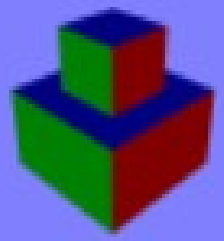




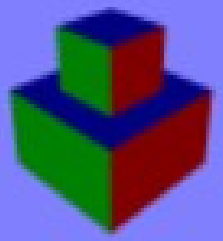
ARRAYS

YOU CAN THINK OF THE LAYOUT OF AN ARRAY AS A TABLE

ARRAYS CAN BE EITHER STRINGS OR NUMBERS JUST LIKE REGULAR VARIABLES



ARRAYS



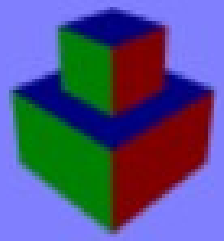
WHY WOULD WE NEED TO USE ARRAYS?

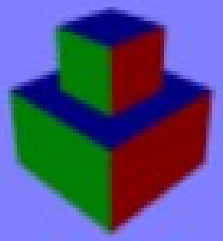
ARRAYS ARE OFTEN USED TO STORE MULTIPLE VALUES OF RELATED DATA

EXAMPLES:

- STORING MULIPLE NAMES, SENTENCES, QUESTIONS IN A STRING ARRAY

- STORING SIZES, POSITIONS, SCORES, ETC IN A NUMBER ARRAY



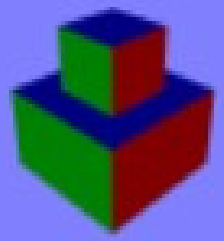


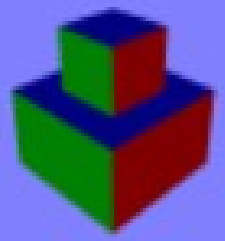
ARRAYS

HOW TO MAKE AN ARRAY

TO MAKE AN ARRAY YOU WILL USE THE DIM KEYWORD

YOU WILL SPECIFY THE DIMENSIONS OF THE ARRAY IN SQUARE BRACKETS





ARRAYS

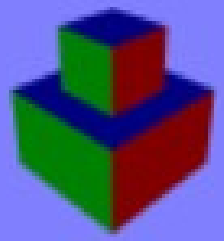
HOW TO MAKE AN ARRAY

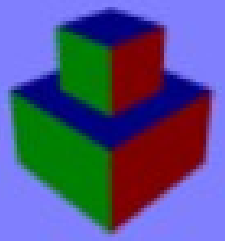
1D ARRAY - **DIM** *variable* [SIZE1]

2D ARRAY - **DIM** *variable* [SIZE1, SIZE2]

3D ARRAY - **DIM** *variable* [SIZE1, SIZE2, SIZE3]

NOTE: WE WILL NOT BE GOING OVER 3D ARRAYS IN THIS DOCUMENT



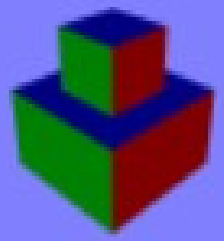


ARRAYS

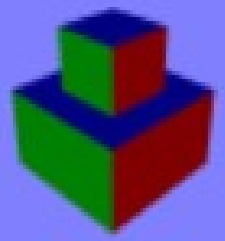
1D ARRAY:

```
DIM  X[ 5 ]
X [ 0 ] = 3
X [ 1 ] = 22
X [ 2 ] = 4
X [ 3 ] = 69
X [ 4 ] = 77
```

	[0]	[1]	[2]	[3]	[4]
X	3	22	4	69	77



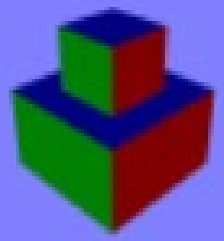
ARRAYS



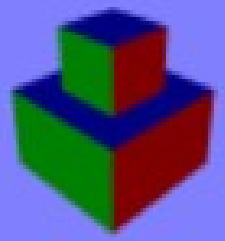
2D ARRAY:

```
DIM  X$[ 4, 3 ]
X [ 0, 2 ] = “A STRING”
X [ 1, 2 ] = “TESTING”
X [ 2, 1 ] = “42”
X [ 3, 0 ] = “HELLO WORLD”
```

X	[0]	[1]	[2]	[3]
[0]				HELLO WORLD
[1]			42	
[2]	A STRING	TESTING		



ARRAYS

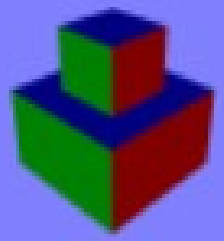


EXAMPLE:

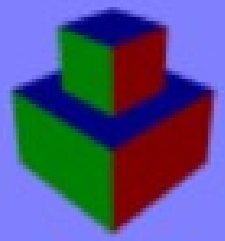
```
Dim x [ 5 ]    '→ this will create an array called x which can hold 5
               '→ values.  The first value will be x[0] and the last
               '→ value will be x[4].  Note: 0 to 4 is 5 different values

x [ 0 ] = 11
x [ 1 ] = 19
x [ 2 ] = 33
x [ 3 ] = 4
x [ 4 ] = 0


Print x [ 2 ]    '→ this will output the value stored in x[2]
```

OUTPUT:

```
33
```



ARRAYS

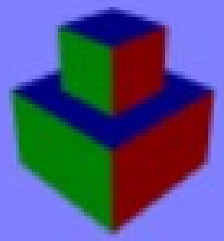


EXAMPLE:

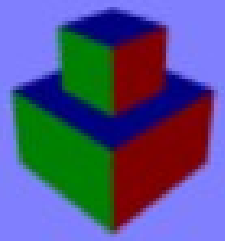
```
Dim x$ [ 5, 4 ]    '→ this will create a 2d string array called x
x [ 0, 0 ] = "this"
x [ 1, 3 ] = "is"
x [ 2, 3 ] = "a"
x [ 3, 1 ] = "string"
x [ 4, 2 ] = "array"


Print x [ 2, 3 ]    '→ this will output the value stored in x[2,3]
```

OUTPUT:

```
a
```



ARRAYS

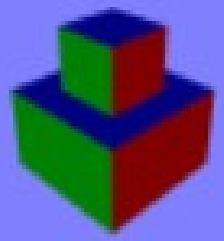


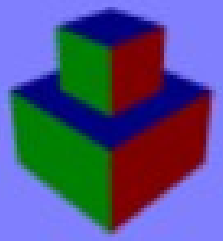
EXAMPLE:

```
Dim x [ 4, 3 ]    '→ this will create a 2d array called x

x [ 0, 2 ] = 5

x [ 3, 1 ] = 88

x [ 2, 0 ] = -4


Print x [ 2, 0 ]    '→ this will output the value stored in x[2,0]
```

OUTPUT:

```
-4
```

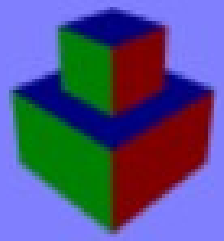




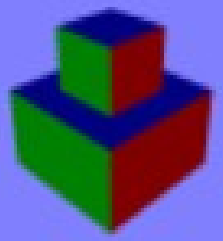
SCOPE

SCOPE REFERS TO WHERE VARIABLES AND ARRAYS CAN BE ACCESSED

VARIABLES THAT ARE CREATED INSIDE OF A FUNCTION, LOOP, IF, OR SELECT CASE ONLY EXISTS INSIDE THOSE STRUCTURES

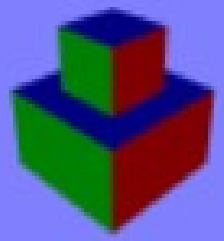


SCOPE

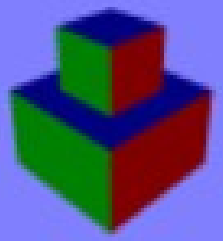


EXAMPLE:

```
IF TRUE THEN   '→THIS LINE IS USING THE BUILT-IN
               '→CONSTANT TRUE WHICH IS EQUAL TO 1
   X = 5       '→ Here we are creating a variable
               '→ called x and setting it to 5
END IF
PRINT X        '→ This will cause a compile error because
               '→ X was created inside the IF block
               '→ it does not exists outside the IF block
```

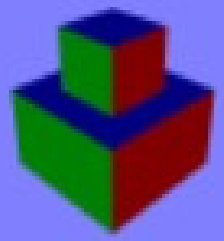


SCOPE

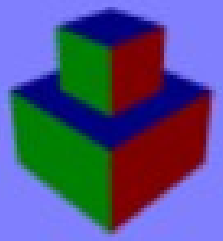


EXAMPLE:

```
X = 0   '→ We are creating X outside of the IF block
        '→ so we will be able to use it both in and out of
        '→ IF block
IF TRUE THEN   '→THIS LINE IS USING THE BUILT-IN
               '→CONSTANT TRUE WHICH IS EQUAL TO 1
   X = 5    '→ Since X was created outside of this IF block
            '→ we can access it inside this if statement as well
            '→ any block started after x was created
END IF
PRINT X  '→ X was created outside of the IF block so it will be
         '->accessable here
```

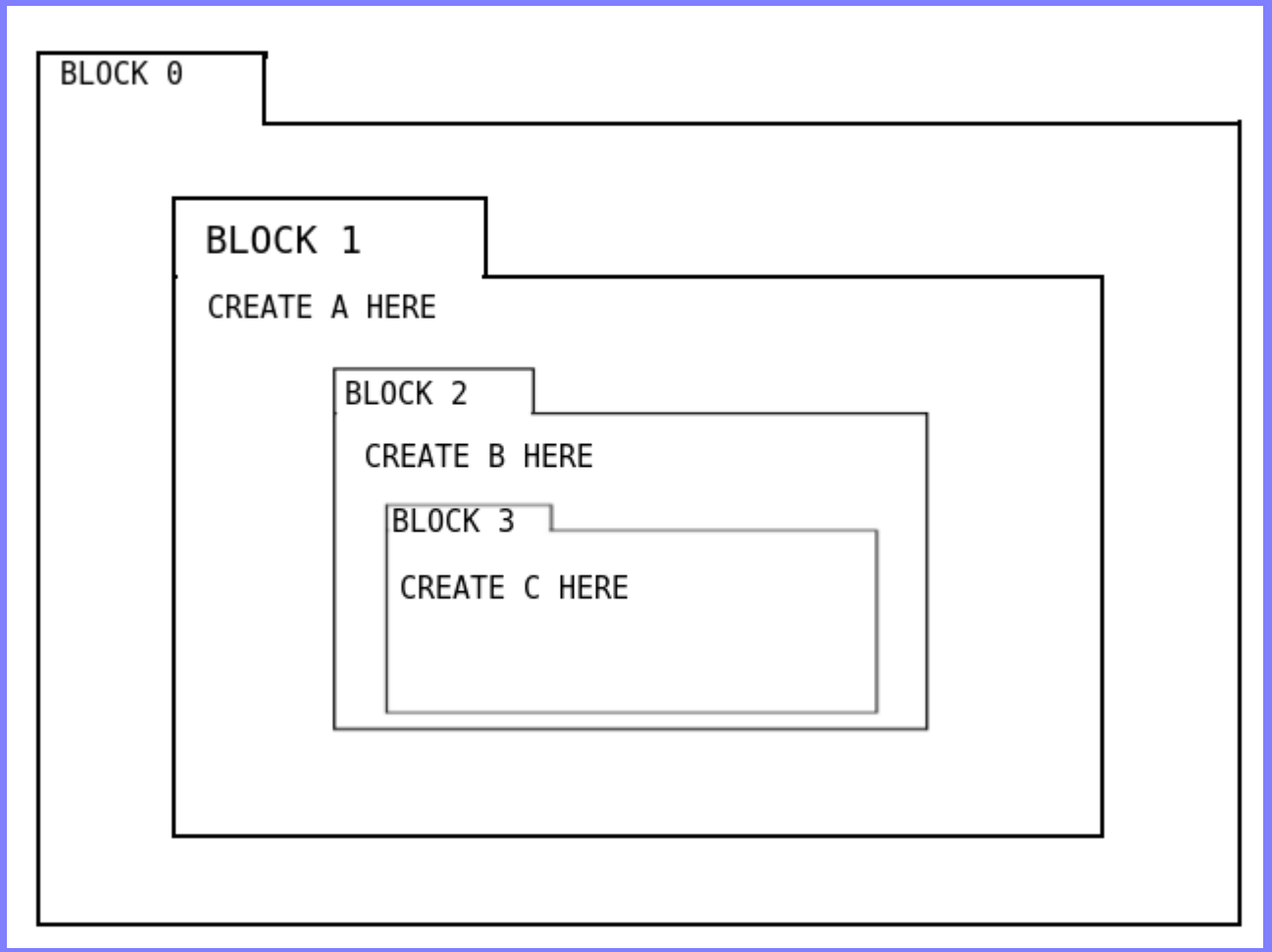


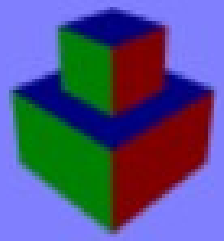
SCOPE



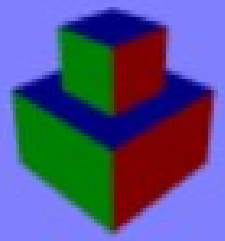
THIS VISUAL EXAMPLE DEMONSTRATES HOW VARIABLES CREATED IN DIFFERENT SCOPES CAN BE ACCESSED

VARIABLE	SCOPE
A	1, 2, 3
B	2, 3
C	3





PRACTICE



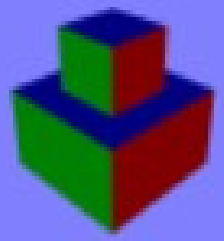
LETS PUT SOME OF WHAT WE WENT OVER IN THIS GUIDE TO PRACTICE WITH A FEW SIMPLE PROGRAMS

1. A PROGRAM THAT USES A **LOOP** TO OUTPUT THE SQUARES OF THE NUMBERS 1 THROUGH 5

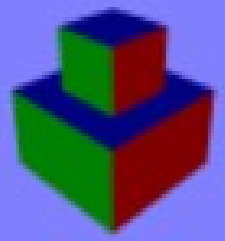
2. CREATE A **FUNCTION CALLED MYFUNC** THAT TAKES **A STRING PARAMETER CALLED S$** AND **A NUMBER PARAMETER CALLED N** AND OUTPUTS THE STRING **N** TIMES

3. A PROGRAM WHICH TAKES 5 NUMBER FROM THE USER AND STORES THEM INTO **A NUMBER ARRAY.** IT WILL THEN ADD ALL THE NUMBERS IN THE ARRAY AND OUTPUTS THE RESULT.

4. A PROGRAM THAT WILL KEEP TAKE USER INPUT UNTIL THE USER TYPES QUIT



PRACTICE #1

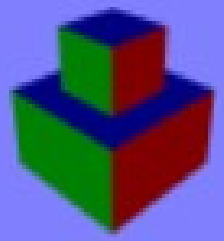


CODE:

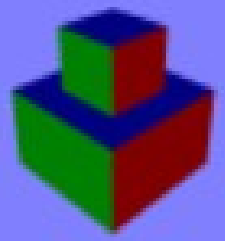
```
For i = 1 to 5     '→ This will will set the variable i to 1 and
                   '→ loop until i is equal to 5

   Print i^2       '→ this will print I squared each time through
                   '→ the loop

Next               '→ this will go back to the start of the for loop if I is
                   '→ less than 5
```

OUTPUT: **RED characters are user input**

```
1
4
9
16
25
```



PRACTICE #2

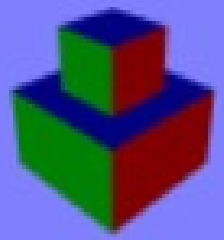


CODE:

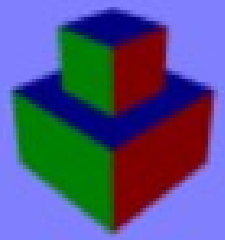
```
Function MyFunc ( s$, n )    '→ Create a function called MyFunc with a string parameter
                             '→ called s and a number parameter called n
   For count = 1 To n    '→ this will set count to 1 and loop until count is equal to n

      Print s    'Print s each time through the loop

   Next    '→ go to the start of the loop if count < n

   Return n * 2   '→ Sets the functions return value to n times 2

End Function   '→ Ends the function

x = MyFunc ( "test", 4 )   '→ This line will run MyFunc with s$ being set to "test" and
                           '→ n being set to 4;  The return value of the function is
                           '→ being stored in the variable x

Print "x = "; x   '→ This line is outputting x to the screen
```

OUTPUT: **RED characters are user input**

```
test
test
test
test
8
```



PRACTICE #3

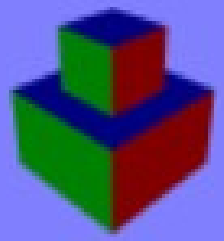


CODE:

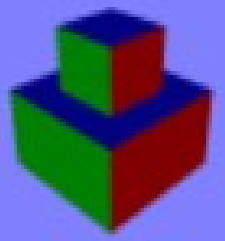
```
Dim user_number [ 3 ]    '→ Create an array of 3 numbers called user_number

For i = 0 To 2    '→ this will set i to 0 and will increase i each time through the loop
                  '→ until i is equal to 2
                  '→ NOTE: This loop goes to 2 because ARRAYS start at 0 so 2 is the
                  '→ third number in the ARRAY

   s$ = Input$ ("Enter a number: ")   '→ this line will get input from the user and
                                      '→ store the input in a variable called s$
                                      '→ We need a string variable because INPUT$()
                                      '→ returns a string

   user_number [ i ] = Val ( s$ )   '→ This line uses the VAL() function convert s$ to a number
                                    '→ and stores it in position i in our number array

Next   '→ if i is less than 2 then this line will jump back to the start of the loop

'→ The line below will create a variable called sum and store the sum of all the
'→ numbers in it
Sum = user_number [ 0 ] + user_number [ 1 ] + user_number [ 3 ]

Print "The sum is "; Sum    '→ Output the sum to the screen
```

OUTPUT: **RED characters are user input**

```
Enter a number: 5
Enter a number: 7
Enter a number: 3
The sum is 15
```



PRACTICE #4

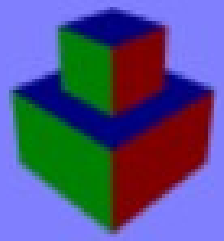


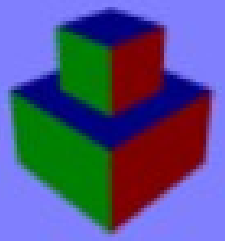
CODE:

```
While TRUE    '→ sets up an infinite loop

   user_input$ = Input("Enter something: ")   '→ takes input from the user and
                                              '→ stores it in the user_input variable

   If user_input = "quit" Then    '→ checks if user_input is equal to "quit"
                                  '→ NOTE: This is case-sensitive

      Exit While   '→ Exits the while loop

   End If   '→ Ends the IF control block

Wend   '→ jumps back to the start of the while loop
```

OUTPUT: **RED characters are user input**

```
Enter something: bob
Enter something: test
Enter something: quit
```





FINAL NOTES

THIS DOCUMENT IS JUST DESIGNED TO INTRODUCE BEGINNERS TO PROGRAMMING AND SOME OF THE BASIC FEATURES OF RCBASIC

FOR MORE INFORMATION YOU CAN VISIT

http://rcbasic.com

http://rcbasic.freeforums.net