


Chapter 7: Decisions, Decisions, Decisions.

The computer is also a whiz at comparing things. In this chapter we will explore how to compare two expressions, how to work with complex comparisons, and how to optionally execute statements depending on the results of our comparisons. We will also look at how to generate random numbers.

True and False:

The BASIC-256 language has one more special type of data, it is the Boolean data type. Boolean values are either true or false and are usually the result of comparisons and logical operations. Also to make them easier to work with there are two Boolean constants that you can use in expressions, they are: *true* and *false*.


 <p>New Concept</p>	<p><i>true</i> <i>false</i></p> <p>The two Boolean constants <i>true</i> and <i>false</i> can be used in any numeric or logical expression but are usually the result of a comparison or logical operator. Actually, the constant <i>true</i> is stored as the number one (1) and <i>false</i> is stored as the number zero (0).</p>
--	--

Comparison Operators:

Previously we have discussed the basic arithmetic operators, it is now time to look at some additional operators. We often need to compare two values in a program to help us decide what to do. A comparison operator works with two values and returns *true* or *false* based on the result of the comparison.

Operator	Operation
<	Less Than expression1 < expression2 Expression is <i>true</i> (1) if expression1 is less than expression2, otherwise it is <i>false</i> (0).
<=	Less Than or Equal expression1 <= expression2 Expression is <i>true</i> (1) if expression1 is less than or equal to expression2, otherwise it is <i>false</i> (0).
>	Greater Than expression1 > expression2 Expression is <i>true</i> (1) if expression1 is greater than expression2, otherwise it is <i>false</i> (0).
>=	Greater Than or Equal expression1 >= expression2 Expression is <i>true</i> (1) if expression1 is greater than or equal to expression2, otherwise it is <i>false</i> (0).
=	Equal expression1 = expression2 Expression is <i>true</i> (1) if expression1 is equal to expression2, otherwise it is <i>false</i> (0).
<>	Not Equal Expression1 <> expression2 Expression is <i>true</i> (1) if expression1 is not equal to expression2, otherwise it is <i>false</i> (0).

Table 7: Comparison Operators

 <p>New Concept</p>	<p>< <= > >= = <></p> <p>The six comparison operations are: less than (<), less than or equal (<=), greater than (>), greater than or equal (>=), equal (=), and not equal (<>). They are used to compare numbers and strings.</p> <p>Strings are compared alphabetically left to right.</p>
---	--

Making Simple Decisions – The If Statement:

The *if* statement can use the result of a comparison to optionally execute a statement or block of statements. This first program (Program 33) uses three *if* statements to display whether your friend is older, the same age, or younger.

```

1 # compareages.kbs
2 # compare two ages
3
4 inputinteger "how old are you?", yourage
5 inputinteger "how old is your friend?", friendage
6
7 print "You are ";
8 if yourage < friendage then print "younger than";
9 if yourage = friendage then print "the same age as";
10 if yourage > friendage then print "older than";
11 print " your friend"

```

Program 33: Compare Two Ages

```

how old are you?13
how old is your friend?12
You are older than your friend

```

Sample Output 33: Compare Two Ages

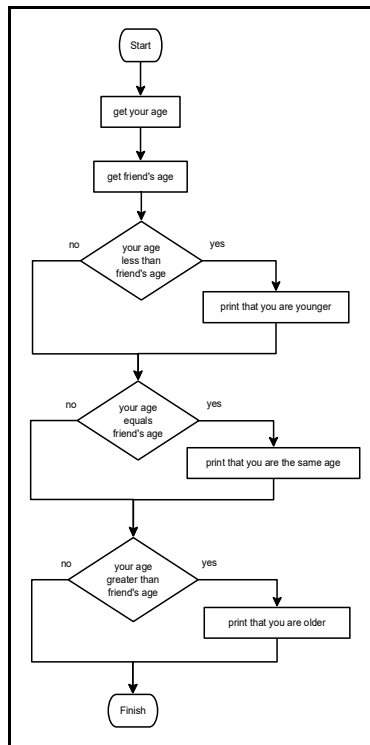


Illustration 16: Compare Two Ages - Flowchart




if condition then statement

If the condition evaluates to *true* then execute the statement following the *then* clause.

Random Numbers:

When we are developing games and simulations it may become necessary for us to simulate dice rolls, spinners, and other random happenings. BASIC-256 has a built in random number generator to do these things for us.


 <p>New Concept</p>	<pre>rand rand()</pre>
	<p>A random number is returned when rand is used in an expression. The returned number ranges from zero to one, but will never be one ($0 \leq n < 1.0$).</p> <p>Often you will want to generate an integer from 1 to r, the following statement can be used $n = \text{int}(\text{rand} * r) + 1$</p>

```
1 # coinflip.kbs
2
3 coin = rand
4 if coin < .5 then print "Heads."
5 if coin >= .5 then print "Tails."
```

Program 34: Coin Flip

Tails.

Sample Output 34: Coin Flip


 <p>Warning</p>	<p>In program 34 you may have been tempted to use the <i>rand</i> expression twice, once in each if statement. This would have created what we call a "Logical Error".</p> <p>Remember, each time the <i>rand</i> expression is executed it returns a different random number.</p>
---	--

Logical Operators:

Sometimes it is necessary to join simple comparisons together. This can be done with the four logical operators: *and*, *or*, *xor*, and *not*. The logical operators work very similarly to the way conjunctions work in the English language, except that "or" is used as one or the other or both.


Operator	Operation																
<p>AND</p>	<p>Logical And expression1 AND expression2 If both expression1 and experssion2 are true then return a true value, else return false.</p> <table border="1" data-bbox="365 1077 1039 1301"> <thead> <tr> <th colspan="2" data-bbox="365 1077 711 1134">AND</th> <th colspan="2" data-bbox="711 1077 1039 1134">expression1</th> </tr> <tr> <td colspan="2" data-bbox="365 1134 711 1190"></td> <th data-bbox="711 1134 875 1190">TRUE</th> <th data-bbox="875 1134 1039 1190">FALSE</th> </tr> </thead> <tbody> <tr> <th data-bbox="365 1190 582 1247">expression2</th> <th data-bbox="582 1190 711 1247">TRUE</th> <td data-bbox="711 1190 875 1247">TRUE</td> <td data-bbox="875 1190 1039 1247">FALSE</td> </tr> <tr> <td colspan="2" data-bbox="365 1247 711 1301"></td> <th data-bbox="711 1247 875 1301">FALSE</th> <th data-bbox="875 1247 1039 1301">FALSE</th> </tr> </tbody> </table>	AND		expression1				TRUE	FALSE	expression2	TRUE	TRUE	FALSE			FALSE	FALSE
AND		expression1															
		TRUE	FALSE														
expression2	TRUE	TRUE	FALSE														
		FALSE	FALSE														

<p>OR</p>	<p>Logical Or expression1 OR expression2 If either expression1 or experssion2 are true then return a true value, else return false.</p> <table border="1" data-bbox="415 373 1089 597"> <thead> <tr> <th colspan="2" rowspan="2">OR</th> <th colspan="2">expression1</th> </tr> <tr> <th>TRUE</th> <th>FALSE</th> </tr> </thead> <tbody> <tr> <th rowspan="2">expression2</th> <th>TRUE</th> <td>TRUE</td> <td>TRUE</td> </tr> <tr> <th>FALSE</th> <td>TRUE</td> <td>FALSE</td> </tr> </tbody> </table>	OR		expression1		TRUE	FALSE	expression2	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE
OR				expression1										
		TRUE	FALSE											
expression2	TRUE	TRUE	TRUE											
	FALSE	TRUE	FALSE											
<p>XOR</p>	<p>Logical Exclusive Or expression1 XOR expression2 If only one of the two expressions is true then return a true value, else return false. The XOR operator works like "or" often does in the English language - "You can have your cake xor you can eat it".</p> <table border="1" data-bbox="415 885 1089 1109"> <thead> <tr> <th colspan="2" rowspan="2">OR</th> <th colspan="2">expression1</th> </tr> <tr> <th>TRUE</th> <th>FALSE</th> </tr> </thead> <tbody> <tr> <th rowspan="2">expression2</th> <th>TRUE</th> <td>FALSE</td> <td>TRUE</td> </tr> <tr> <th>FALSE</th> <td>TRUE</td> <td>FALSE</td> </tr> </tbody> </table>	OR		expression1		TRUE	FALSE	expression2	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
OR				expression1										
		TRUE	FALSE											
expression2	TRUE	FALSE	TRUE											
	FALSE	TRUE	FALSE											
<p>NOT</p>	<p>Logical Negation (Not) NOT expression1 Return the opposite of expression1. If expression 1 was true then return false. If experssion1 was false then return a true.</p> <table border="1" data-bbox="415 1358 899 1525"> <thead> <tr> <th colspan="2">NOT</th> <th></th> </tr> </thead> <tbody> <tr> <th rowspan="2">expression 1</th> <th>TRUE</th> <td>FALSE</td> </tr> <tr> <th>FALSE</th> <td>TRUE</td> </tr> </tbody> </table>	NOT			expression 1	TRUE	FALSE	FALSE	TRUE					
NOT														
expression 1	TRUE	FALSE												
	FALSE	TRUE												

 <p>New Concept</p>	<pre>and or xor not</pre> <p>The four logical operations: logical and, logical or, logical exclusive or, and logical negation (not) join or modify comparisons.</p> <p>You may also use parenthesis to group operations together.</p>
---	---

Making Decisions with Complex Results – If/End If:

When we are writing programs it sometimes becomes necessary to do multiple statements when a condition is *true*. This is done with the alternate format of the **if** statement. With this statement you do not place a statement on the same line as the **if**, but you place multiple (one or more) statements on lines following the **if** statement and then close the block of statements with the **end if** statement.

 <p>New Concept</p>	<pre>if condition then statement(s) to execute when true end if</pre> <p>The if/end if statements allow you to create a block of programming code to execute when a condition is true. It is customary to indent the statements within the if/end if statements so they are not confusing to read.</p>
---	--


In the following example you will see **if** statements nested inside another **if** statement. It is important that you remember that the inner **ifs** will only be tested when the outer **if** is true.


```
1 # dice.kbs - roll 2 6-sided dice
2
3 die1 = int(rand * 6) + 1
4 die2 = int(rand * 6) + 1
5 total = die1 + die2
6
7 print "die 1 = " + die1
8 print "die 2 = " + die2
9 message = "You rolled " + total + "."
10
11 if die1 = die2 then
12     message += " Doubles."
13     if total = 2 then
14         message += " Snake eyes."
15     end if
16     if total = 12 then
17         message += " Box Cars."
18     end if
19 end if
20
21 print message
```

Program 35: Rolling Dice


```
die 1 = 1
die 2 = 1
You rolled 2. Doubles. Snake eyes.
```

Sample Output 35: Rolling Dice

 <p>New Concept</p>	<p>"Edit" then "Beautify" on the menu</p> <p>The "Beautify" option on the "Edit" menu will clean up the format of your program to make it easier to read. It will remove extra spaces from the beginning and ending of lines and will indent blocks of code (like in the if/end if statements).</p>
---	--

Deciding Both Ways – If/Else/End If:

The third and last form of the **if** statement is the **if/else/end if**. This extends the **if/end if** statements by allowing you to create a block of code to execute if the condition is *true* and another block to execute when the condition is *false*.

 <p>New Concept</p>	<pre>if condition then statement(s) to execute when true else statement(s) to execute when false end if</pre> <p>The if, else, and end if statements allow you to define two blocks of programming code. The first block, after the then clause, executes if the condition is <i>true</i> and the second block, after the else clause, will execute when the condition is <i>false</i>.</p>
--	--

Program 36 re-writes Program 34 using the *else* statement.

```
1 # coinflip2.kbs
2 # coin flip with else
3
4 coin = rand
```

```

5  if coin < .5 then
6      print "Heads."
7      say "Heads."
8  else
9      print "Tails."
10     say "Tails."
11 end if

```

Program 36: Coin Flip – With Else

Heads.

Sample Output 36: Coin Flip – With Else



**Big
Program**

This chapter's big program is a program to roll a single 6-sided die and then draw on the graphics display the number of dots.

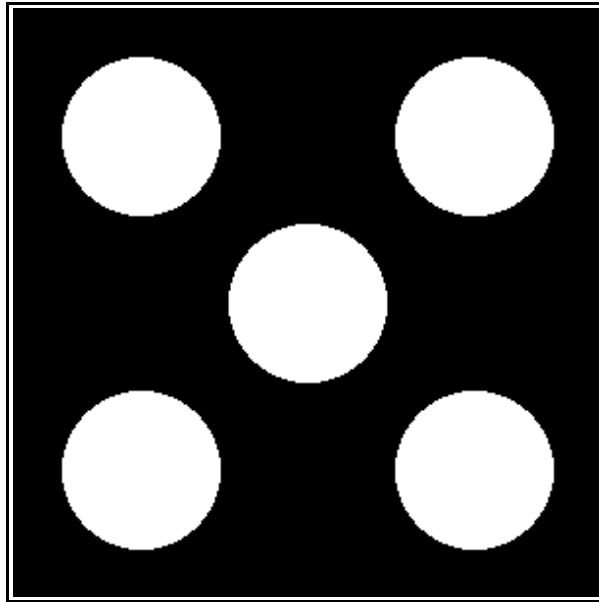
```

1  # dieroll.kbs - roll a 6-sided die on the screen
2
3  # radius of the dots
4  r = 40
5  # z1, z2, and z3 contain the center if the dots in
   each row and column
6  z1 = 65
7  z2 = 150
8  z3 = 235
9
10 # get roll
11 roll = int(rand * 6) + 1
12

```


```
13  clg black
14
15  color white
16  # top row
17  if roll <> 1 then circle z1,z1,r
18  if roll = 6 then circle z2,z1,r
19  if roll >= 4 and roll <= 6 then circle z3,z1,r
20  # middle row
21  if roll = 1 or roll = 3 or roll = 5 then circle
    z2,z2,r
22  # bottom row
23  if roll >= 4 and roll <= 6 then circle z1,z3,r
24  if roll = 6 then circle z2,z3,r
25  if roll <> 1 then circle z3,z3,r
26
27  message = "You rolled a " + roll + "."
28  print message
29  say message
```

Program 37: Big Program - Roll a Die and Draw It



Sample Output 37: Big Program - Roll a Die and Draw It

Exercises:

 <p>Word Search</p>	<pre> b t t h e n m r n s i o r w l f o r z e e d o u l d d o d s r n r l e w n t j l a a e u e t a a r e p n t l n a r r o o m o a a d s n e p l o t e u i h l p t e c i r q f f s o h s w f g e e s l a f s </pre> <p>and, boolean, compare, else, endif, equal, false, greater, if, less, not, operator, or, random, then, true</p>
---	--



Problems

1. Write a program that will toss a coin and tell you if your guess was correct. Assign a variable with a random number. Ask the user to enter the letter 'h' or 't' (for heads or tails). If the number is less than .5 and the user entered 'h' or the number was greater than or equal .5 and the user chose 't' then tell them they won the toss.
2. Modify program #1 in this chapter to also tell the user that they did not win the toss.
3. Write a simple program to draw a round of rock, paper, scissors. Use two numeric variables and assign a draw (random number) to each one. If a variable is less than 1/3 then it will be rock, greater than or equal to 1/3 and less than 2/3 it will be paper, and 2/3 or greater it will be scissors. Display what the two draws are.
4. Take the simple rock, paper, scissors draw program from #3 in this chapter and add rules to say who won. Remember "paper covers rock", "rock smashes scissors", and "scissors cut paper". If both players draw the same thing then declare the round a "draw".
5. Take the rock paper scissors game from #4 and add graphics and sound. Draw paper as a white rectangle, rock as a darkorange circle, and scissors as a red X. Have the computer announce the winner.