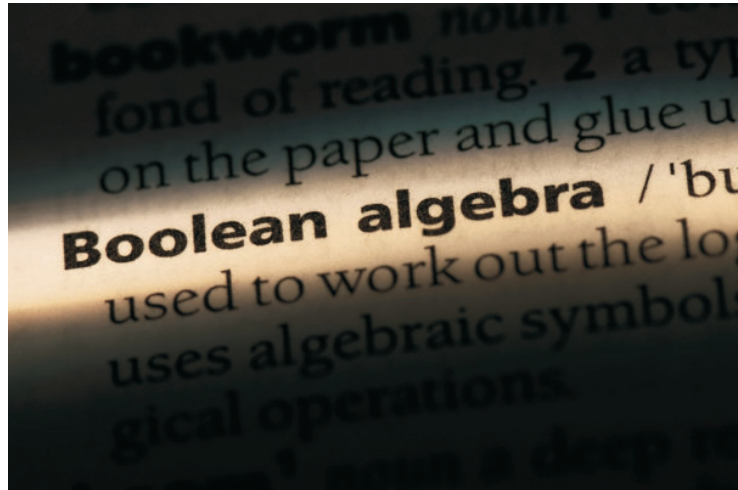


# Boolean Logic

**C**OMPUTER PROGRAMS contain large sections of code that only execute under specific circumstances. For example, a student is awarded an “A” grade only when their points equal or are greater than 90. Logic is implemented using decision structures with Boolean expressions. It is, therefore, important to understand Boolean algebra because it allows you to construct logically sound Boolean expressions.



## Objective:



Explain Boolean Expressions and understand Boolean Algebra.

## Key Terms:



ASCII Collating  
sequence  
Boolean Algebra

Boolean Expression  
Logic Gates  
Logical Operators

Relational Operators  
Truth Tables

## Boolean Logic

Boolean logic is based upon the work of George Boole, an English mathematician. He was born in 1815 and died in 1864 and created a system of logic based on the operators, “AND”, “NOT”, and “OR”. This system of logic is called “Boolean logic” and was named after him. Boolean logic lent itself to creation of computers that understand 2 binary states. Therefore, computer programming logic depends on understanding the principles of Boolean logic.

## EXPLAIN BOOLEAN EXPRESSIONS

Boolean expressions are used in computer programs to compare 2 values and to evaluate the result of the comparison operation. These values can be held in literals or in variables. Most programming languages require that the 2 entities being compared be of the same datatype. For example, a string may be compared to another string, but a string may not be compared to a number. Logic is implemented based upon the result of the comparison operation. This section discusses the parts of a Boolean expression in detail.

### Relational Operators

Two operands of the same datatype are compared to each other using **Relational Operators**. See FIGURE 1. It shows the list of relational operators commonly used in modern languages today. Some of the relational operators, such as the greater-than-equal-to operator, are composed of 2 symbols from the keyboard. This is because in the early days of programming, the keyboard used to key in programs was the same one used by typewriters, and it did not have the  $\neq$ ,  $\geq$ , or the  $\leq$  symbols on it. Therefore, a combination of symbols found on the keyboard was used to represent certain operations, such as the  $\geq$  operator. No space must be left between the 2 symbols and the order of the 2 symbols is significant. For example, the  $\geq$  operator cannot be replaced by the  $=>$  operator. The “!” and the “=” symbols are combined as the “not-equals” operator. The last 2 operators shown in FIGURE 1 are used in loosely typed languages, such as JavaScript and PHP. These operators check both the datatype and value of the 2 entities being compared.

Relational Operator	Description	Operator Used in Math
>	Greater than	>
<	Less than	<
>=	Greater than equal to	$\geq$
<=	Less than equal to	$\leq$
==	Equals	=
!=	Not equal to	$\neq$
===	Strict equals	
!==	String not equal to	

There is no space between the individual symbols.

FIGURE 1. Relational operators.

### Structure of a Boolean Expression

A simple **Boolean expression** consists of 2 operands of the same datatype connected by a relational operator. The 2 operands can be literals or variables.

### Boolean Expressions with Literals

See FIGURE 2 for examples of Boolean Expressions built around literals. Valid Boolean expressions contain operands of the same data type. Numeric literals are compared with one

another, and string literals are compared with one another. Invalid Boolean expressions compare operands of different data types. For example, 12 cannot be compared with “15”. “15” is treated as a string literal even though its contents are numeric. Comparing “Spring” with spring causes the system to treat spring as a variable since it is not surrounded by quotes. The system will attempt to locate a variable with the name spring and raise an error if it cannot be found in the program.

## Boolean Expressions with Variables and Literals

See FIGURE 3 for Boolean expressions with variables and literals. Valid statements are those where the data types of the 2 operands match. For example, the stu\_name variable can be

### Valid Boolean Expressions with Literals

Operand 1	Operand 2	Boolean Expression
12	15	12 > 15
12	15	12 < 15
Spring	spring	"SPRING" >= "spring"
12	12	12 <= 12
12	15	12 == 15
12	15	12 != 15

### Invalid Boolean Expressions with Literals

Operand 1	Operand 2	Boolean Expression
12	"15"	12 > "15"
12	"fifteen"	12 < "fifteen"
Spring	spring	"SPRING" >= spring

FIGURE 2. Relational operators.

### Valid Boolean Expressions with Variables

Declare stu\_name as String  
 Declare number\_of\_items as Integer  
 Set stu\_name = "Jane"  
 Set number\_of\_items = 18

Pseudocode to declare variables and assign values to them

Operand 1	Operand 2	Boolean Expression
stu_name	"John"	stu_name < "John"
stu_name	"Jane"	stu_name == "Jane"
number_of_items	0	number_of_items >= 0
number_of_items	100	number_of_items < 100

### Invalid Boolean Expressions with Variables

Declare stu\_name as String  
 Declare number\_of\_items as Integer  
 Set stu\_name = "Jane"  
 Set number\_of\_items = 18

Pseudocode to declare variables and assign values to them

Operand 1	Operand 2	Boolean Expression
stu_name	12	stu_name < 12
stu_name	number_of_items	stu_name == number_of_items
number_of_items	"0"	number_of_items >= "0"

FIGURE 3. Boolean expressions with variables and literals.

compared with the string literal “John” since the `stu_name` variable has been declared as a String variable. Similarly, variables that are declared as numeric variables may be compared with numeric literals. Invalid Boolean expressions are those where datatypes are dissimilar. The `stu_name` String variable cannot be compared with the numeric literal 12, and the Integer variable `number_of_items` cannot be compared with a string literal (even when the contents of the String literal is a number). This principle is True when comparing 2 variables. A String variable, such as `stu_name`, cannot be compared with a numeric variable, `number_of_items`.

### Suggested Format of a Boolean Expression

See FIGURE 4. When developing Boolean expressions with variables and literals in a computer program, it is customary to code the variable as the first operand and the literal as the second operand. There are no conventions when both the operands are variables.

Valid Boolean Expressions with Variables		
Declare <code>stu_name</code> as String Declare <code>number_of_items</code> as Integer Set <code>stu_name</code> = "Jane" Set <code>number_of_items</code> = 18		
		Pseudocode to declare variables and assign values to them
Format 1	Format 2	Suggested Format
<code>stu_name &lt; "John"</code>	<code>"John" &gt; stu_name</code>	<code>stu_name &lt; "John"</code>
<code>stu_name == "Jane"</code>	<code>"Jane" == stu_name</code>	<code>stu_name == "Jane"</code>
<code>number_of_items &gt;= 0</code>	<code>0 &lt; number_of_items</code>	<code>number_of_items &gt;= 0</code>
<code>number_of_items &lt; 100</code>	<code>100 &gt; number_of_items</code>	<code>number_of_items &lt; 100</code>

FIGURE 4. Suggested format of Boolean expressions.

### Comparing Non-Integer Values

When integer values are compared, their values are used in the Boolean expression. String entities and float values are evaluated in a different manner and are discussed here in detail.

### Comparison of 2 String Values

Computers do not natively understand any human language, including English. Therefore, in order to process alphabets in a computer device, they must be converted into numbers. The **ASCII Collating sequence** is used on PCs, Macs, and Unix/Linux machines, and it provides numeric values for all characters that can be keyed in via a keyboard. See FIGURE 5 for the ASCII collating sequence characters and their numeric representations. For example, the

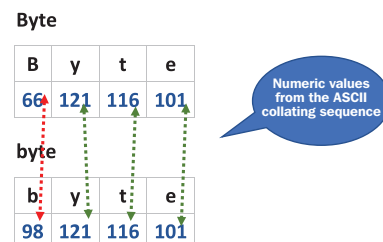
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	#32; Space		64	40	100	#64; @		96	60	140	#96; `	
1	1	001	SOH (start of heading)	33	21	041	#33; !		65	41	101	#65; A		97	61	141	#97; a	
2	2	002	STX (start of text)	34	22	042	#34; "		66	42	102	#66; B		98	62	142	#98; b	
3	3	003	ETX (end of text)	35	23	043	#35; #		67	43	103	#67; C		99	63	143	#99; c	
4	4	004	EOT (end of transmission)	36	24	044	#36; \$		68	44	104	#68; D		100	64	144	#100; d	
5	5	005	ENQ (enquiry)	37	25	045	#37; %		69	45	105	#69; E		101	65	145	#101; e	
6	6	006	ACK (acknowledge)	38	26	046	#38; &		70	46	106	#70; F		102	66	146	#102; f	
7	7	007	BEL (bell)	39	27	047	#39; '		71	47	107	#71; G		103	67	147	#103; g	
8	8	010	BS (backspace)	40	28	050	#40; (		72	48	110	#72; H		104	68	150	#104; h	
9	9	011	TAB (horizontal tab)	41	29	051	#41; )		73	49	111	#73; I		105	69	151	#105; i	
10	A	012	LF (NL line feed, new line)	42	2A	052	#42; *		74	4A	112	#74; J		106	6A	152	#106; j	
11	B	013	VT (vertical tab)	43	2B	053	#43; +		75	4B	113	#75; K		107	6B	153	#107; k	
12	C	014	FF (NP form feed, new page)	44	2C	054	#44; ,		76	4C	114	#76; L		108	6C	154	#108; l	
13	D	015	CR (carriage return)	45	2D	055	#45; -		77	4D	115	#77; M		109	6D	155	#109; m	
14	E	016	SO (shift out)	46	2E	056	#46; .		78	4E	116	#78; N		110	6E	156	#110; n	
15	F	017	SI (shift in)	47	2F	057	#47; /		79	4F	117	#79; O		111	6F	157	#111; o	
16	10	020	DLE (data link escape)	48	30	060	#48; 0		80	50	120	#80; P		112	70	160	#112; p	
17	11	021	DC1 (device control 1)	49	31	061	#49; 1		81	51	121	#81; Q		113	71	161	#113; q	
18	12	022	DC2 (device control 2)	50	32	062	#50; 2		82	52	122	#82; R		114	72	162	#114; r	
19	13	023	DC3 (device control 3)	51	33	063	#51; 3		83	53	123	#83; S		115	73	163	#115; s	
20	14	024	DC4 (device control 4)	52	34	064	#52; 4		84	54	124	#84; T		116	74	164	#116; t	
21	15	025	NAK (negative acknowledge)	53	35	065	#53; 5		85	55	125	#85; U		117	75	165	#117; u	
22	16	026	SYN (synchronous idle)	54	36	066	#54; 6		86	56	126	#86; V		118	76	166	#118; v	
23	17	027	ETB (end of trans. block)	55	37	067	#55; 7		87	57	127	#87; W		119	77	167	#119; w	
24	18	030	CAN (cancel)	56	38	070	#56; 8		88	58	130	#88; X		120	78	170	#120; x	
25	19	031	EM (end of medium)	57	39	071	#57; 9		89	59	131	#89; Y		121	79	171	#121; y	
26	1A	032	SUB (substitute)	58	3A	072	#58; :		90	5A	132	#90; Z		122	7A	172	#122; z	
27	1B	033	ESC (escape)	59	3B	073	#59; ;		91	5B	133	#91; [		123	7B	173	#123; {	
28	1C	034	FS (file separator)	60	3C	074	#60; <		92	5C	134	#92; \		124	7C	174	#124;	
29	1D	035	GS (group separator)	61	3D	075	#61; =		93	5D	135	#93; ]		125	7D	175	#125; }	
30	1E	036	RS (record separator)	62	3E	076	#62; >		94	5E	136	#94; ^		126	7E	176	#126; ~	
31	1F	037	US (unit separator)	63	3F	077	#63; ?		95	5F	137	#95; _		127	7F	177	#127; DEL	

Taken from <http://www.asciitable.com/>

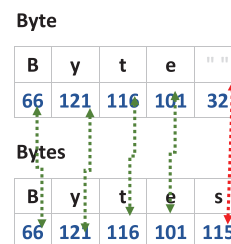
FIGURE 5. The ASCII collating sequence.

alphabet “M” has a numeric value of 77, and the alphabet “m” has a numeric value of 109. The ASCII collating sequence plays an important when strings are compared. When 2 string entities are compared, they are viewed as a sequence of numbers from the ASCII collating sequence. The numbers for each individual character in the strings are compared. See FIGURE 6. Example 1 shows the 2 Strings along with the numeric values for each alphabet in the 2 strings. When 2 String entities are compared, the system starts the comparison process from the left to the right, one character at a time. In this example, the numeric values of “B” and “b” are compared, and the “b” is deemed to be higher. The comparison process stops at this point with the determination that “byte” is larger than “Byte”. Example 2 compares 2 strings of unequal lengths. The shorter string is padded with a blank in order to make the 2 string values of the same length. A space, too, has an ASCII number associated with it. Since the numeric value of a space is smaller than that of “s”, the string “Bytes” is considered larger than “Byte”. As shown in Example 3, lengths of strings do not affect string comparison. For example, the string “Mat” is considered larger than “Able” since the numeric value of the first character “M” is larger than that of “A”.

Example 1 : "Byte" and "byte"



Example 2: "Byte" and "Bytes"



Example 3: "Mat" and "Able"

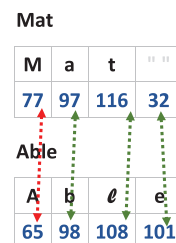


FIGURE 6. Comparing strings.



## Comparison of 2 Float Values

Comparing float values poses a challenge to computer programs. See FIGURE 7. In Example 1, 2 float values are compared. They differ in digits maintained at the 11th, 12th, 13th and 14th place after the decimal point. When they are compared, the computer will conclude that they are not equal. In Example 2, the difference between the 2 number is compared with 0.001. The difference is smaller than 0.001. This is the technique used to compare 2 float numbers. The difference of the 2 numbers is converted into an absolute value which makes it a positive number. This difference is compared to the difference threshold of the program. For example, when computing weights of 2 people, it is possible to write code that states that the weight of 2 people is the same if their weights differs in the 4th place after the decimal point.

### Example 1

Compare 10.333333333333333 and 10.3333333333

### Example 2

Compare (10.333333333333333 - 10.3333333333) and 0.0001

Compare 0.00000000003333 and 0.001

FIGURE 7. Comparing float values.

## WHAT IS THE BOOLEAN ALGEBRA?

Boolean algebra is a formalized study of Boolean expressions and the rules that they obey. The basic unit of Boolean algebra is a Boolean expression. Simple Boolean expressions may be combined to create complex Boolean expressions and their result is explained by Boolean algebra.

### Value of a Boolean Expression

When 2 values are compared, the result of the comparison operation is either a yes or a no. For example, when a person's age is checked to see if it is greater than or equal to 21, the answer is either a yes or a no. In Boolean terms, the answer is either a True or a False. Therefore, a Boolean Expression has a value of True or False. Programming languages maintain Boolean values in different formats as discussed below:

### C and C++

The C programming language maintains Boolean values (True and False) in an integer. False has a value of 0 and any non-zero value is True. When assigning a value of True to a variable, 1 is typically used to represent True. C++, which is an enhanced version of C, contains a



## FURTHER EXPLORATION...

### ONLINE CONNECTION: Other Collating Sequences

The ASCII Collating sequence was introduced in this lesson. It provides numeric values for each key found in a keyboard, such as letters A-Z, a-z, 0-9, and other special characters. However, it is not the only collating sequence used to store data on a computer. Mainframes use the EBCDIC (Extended Binary Coded Decimal Interchange Code) collating sequence and Unicode is gaining popularity with its ability to store emoji characters.

To learn more about ASCII, EBCDIC and Unicode, go to <https://www.youtube.com/watch?v=3kXLHLUhV5Q>.

bool datatype to hold Boolean values of True and False. However, C++ honors C's representation of Boolean values as integers and understands 0 as False and non-zero values as True.

### Java

Java contains a Boolean datatype that can hold only one of 2 values, True and False. Placing Integers into a Boolean variable will raise an error.

### Other Languages

Languages, such as JavaScript, have a more expansive view of True and False. In JavaScript, a variable with no value in it is treated as False. An empty string is also considered False.

## Boolean Logical Operators

Many real-world computer-based operations require that a value be compared with more than one other value. For example, there may be a requirement to ensure that a test score fall within a range. In such a case, the test score must be compared with the lower boundary value and the upper boundary value. Computers, however, can only compare two values at a time. When a value, such as a test score, must be checked to see if it is within a range, multiple Boolean expressions must be coded and tested. Logical operators are used to combine simple Boolean expressions to create a complex Boolean expression. See FIGURE 8 for the Logical operators used to combine Boolean expressions. Some of the original symbols used by Boolean algebra are not found on a keyboard. Therefore, computer programs like C, C++ and Java, use the "&&" and "||" to represent the "and" and the "or" operators respectively. The "|" symbol is referred to as the piping symbol, and its location on the keyboard is shown. Some computer programming languages (such as Python) use "or", "and", and "not" for their logical operators. This lesson uses "AND", "OR" and "NOT" instead of Boolean and programming operators. See FIGURE 9 for examples of combined Boolean expressions where one simple Boolean expression is combined with another simple Boolean expression using Logical operators.

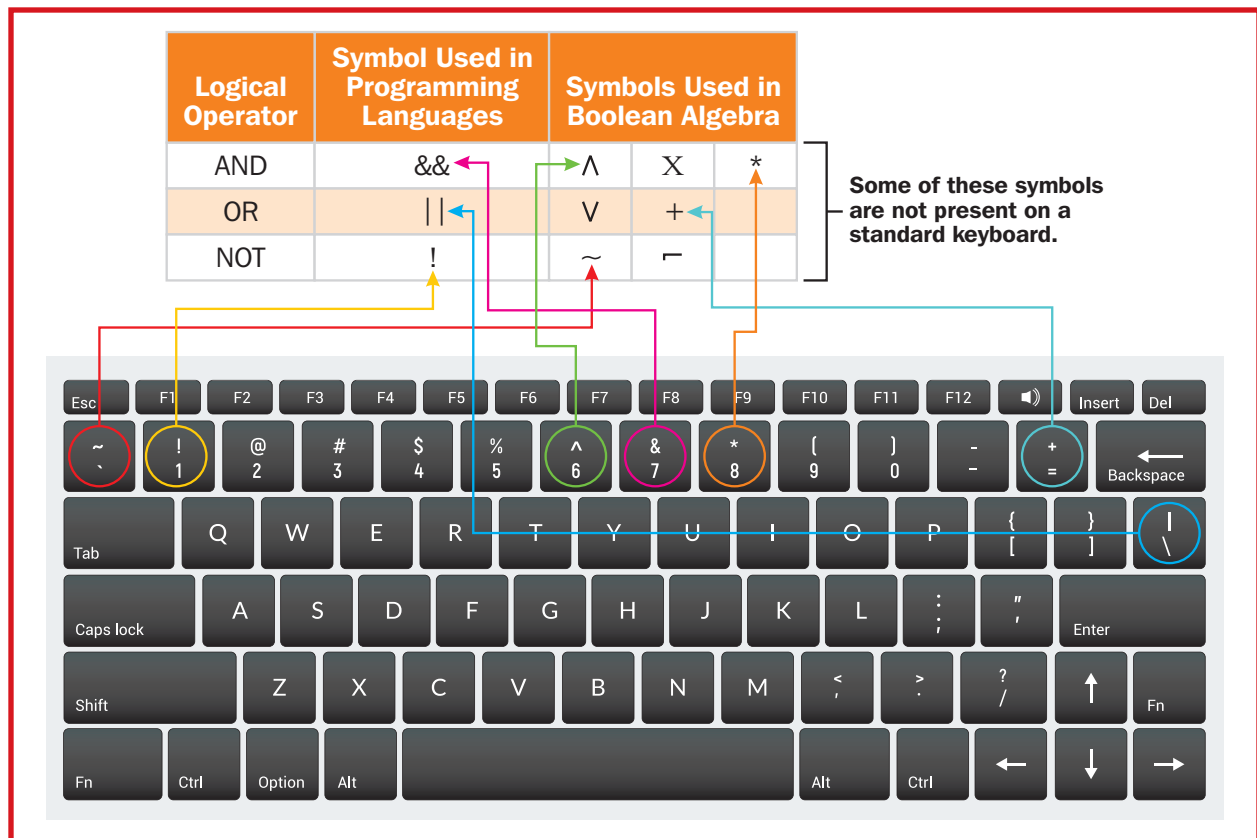


FIGURE 8. Logical operators.

### Variables used in Boolean Expressions

```

Declare quantity as Integer
Declare discount as float
Print "How many units did you purchase?"
Input quantity
Print "What is the discount %?"
Input discount

```

### Boolean Expressions

Boolean Expression 1	Boolean Expression 2
quantity > 100	discount == 25

### Complex Boolean Expression

Expression	Explanation
quantity > 100 AND discount == 0.25	User purchased more than 100 units and has a 25% discount coupon
quantity > 100 OR discount == 0.25	Either User purchased more than 100 units or has a 25% discount

FIGURE 9. Complex Boolean expressions.



## Basic Boolean Algebra

Boolean Algebra studies Boolean expressions and their behavior when they are grouped into larger and more complex expressions. In Boolean algebra, each operand is a Boolean expression which can have only 1 of 2 values, True or False. Boolean algebra uses 0 to represent False and 1 to represent True. This is in sharp contrast to “regular” algebra where operands can have any numeric value. See FIGURE 10 for examples of regular algebra and Boolean algebra. The variables used in regular algebra can have any numeric value. In Boolean algebra, each operand is a Boolean expression, as shown in the blue and orange callout boxes. Each Boolean expression evaluates to True or False.

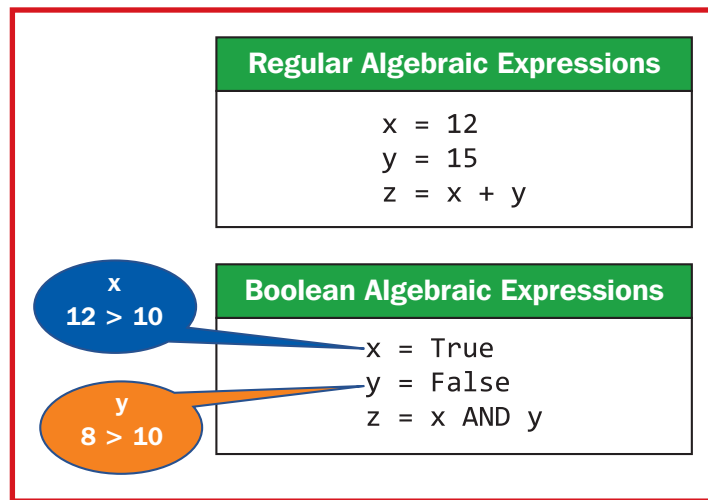


FIGURE 10. “Regular” Algebra and Boolean Algebra.

## The AND Logical Operator

Boolean expressions can be combined using the AND logical operator. When 2 expressions are combined using AND, there are 4 possible permutations possible. The table that contains all possible permutations of 2 or more Boolean expressions along with the result of the combination is called a **Truth Table**. The Truth Table for the AND logical operator is shown in FIGURE 11. The problem being addressed is whether a person can go out for a walk or not based upon the answer to 2 questions. These are whether it is sunny or not and whether the temperature is more than 70 degrees or not. Each of these questions is a Boolean expression that evaluates to True or False. There are 4 permutations of the values of these 2

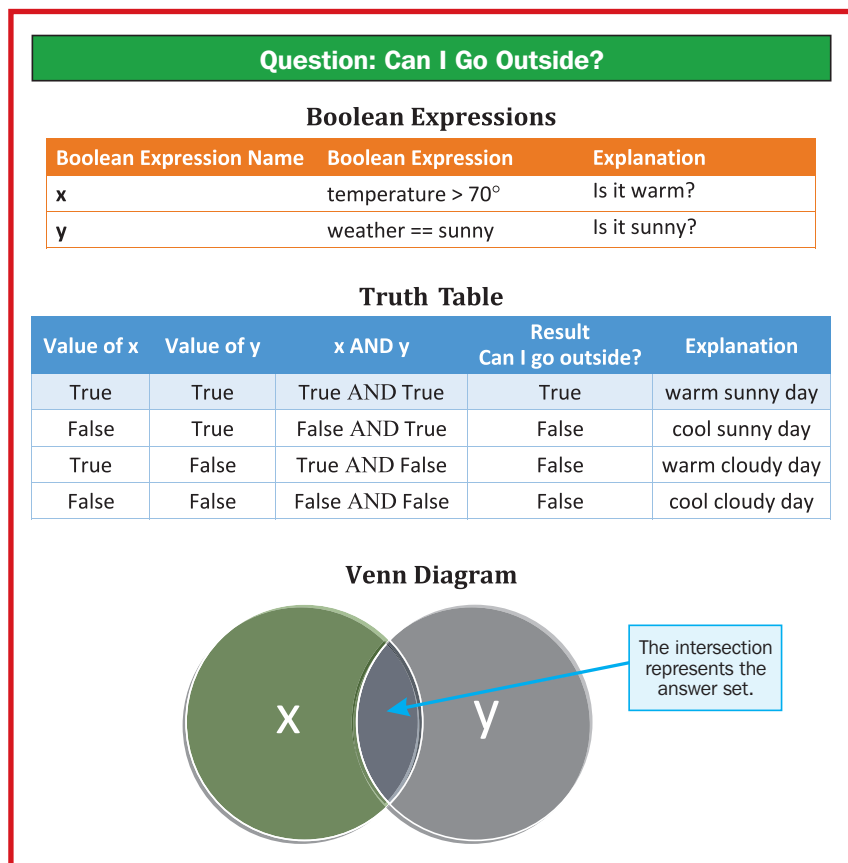


FIGURE 11. Truth Table for the AND logical operator.

Boolean expressions. When the “AND” logical operator is used, each individual Boolean expression must evaluate to True for the combined expression to return a True. In the example shown in FIGURE 11, the user can only walk on a warm, sunny day, which is possible only when both Boolean expressions return True.

## The OR Logical Operator

Boolean expressions can also be combined using the OR logical operator. The Truth Table for the OR logical operator is shown in FIGURE 12. The problem being addressed is whether to buy detergent or not based upon the answer to 2 questions. These are whether it is on sale and whether the user is running low on detergent. These questions are based upon 2 Boolean expressions that evaluate to True or False. There are, once again, 4 permutations of the values of these 2 Boolean expressions. When the “OR” logical operator is used, the result is False only when both Boolean expressions evaluate to False. When either one of them is true, or when they are both true, the result of the complex Boolean Expression is True.

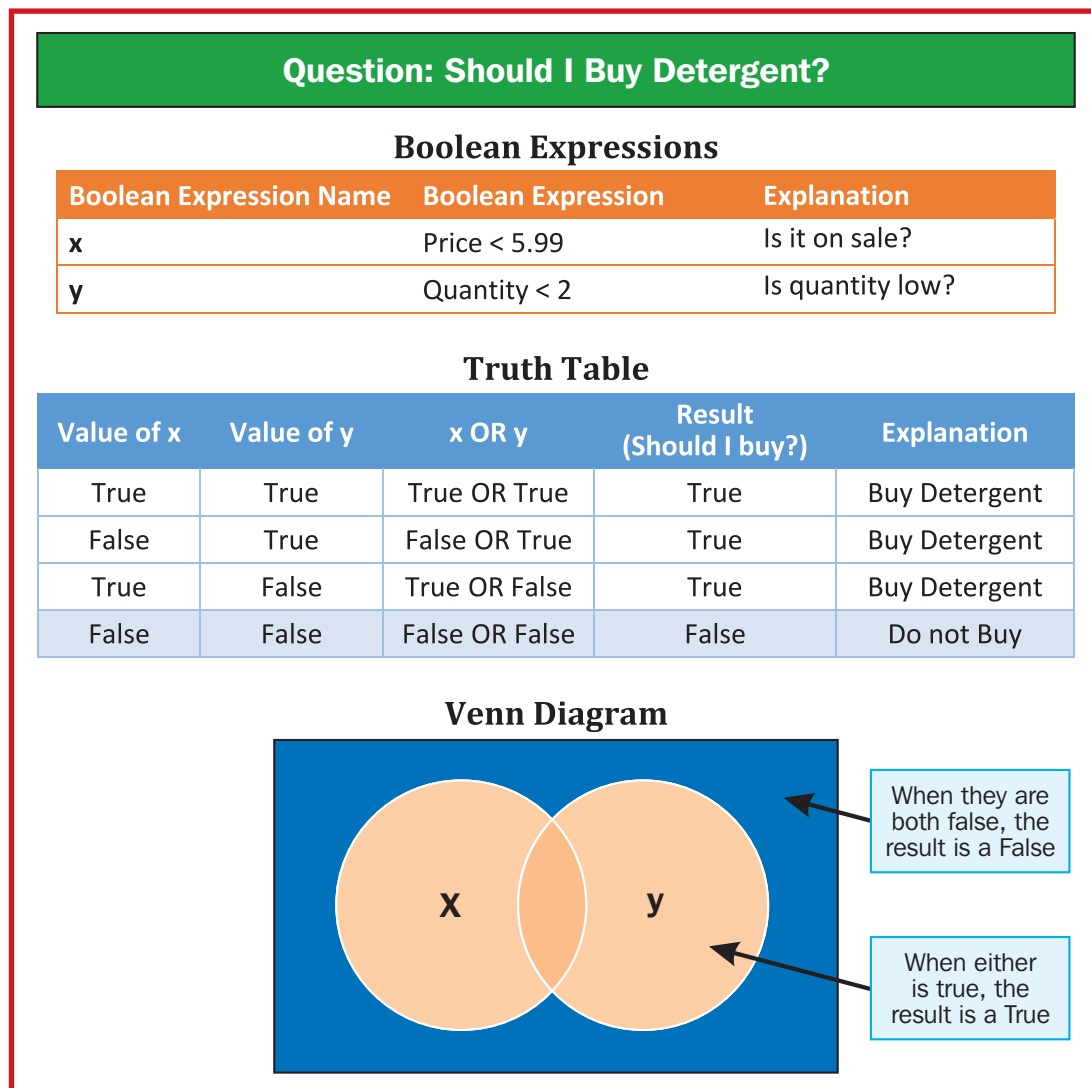


FIGURE 12. Truth Table for the OR logical operator.

## The NOT Logical Operator

See FIGURE 13. It shows the Truth Table for evaluating the “NOT” logical operator. Only 1 Boolean expression is used in the example, and the “NOT” operator inverts or negates the value of the Boolean expression. The Boolean expression checks the rating of a game and returns True or False depending upon whether it has a “Mature” rating or not. The “NOT” operator inverts the answer returned by the Boolean expression. If the Boolean expression evaluates to True, the value of the expression after the usage of “NOT” is False.

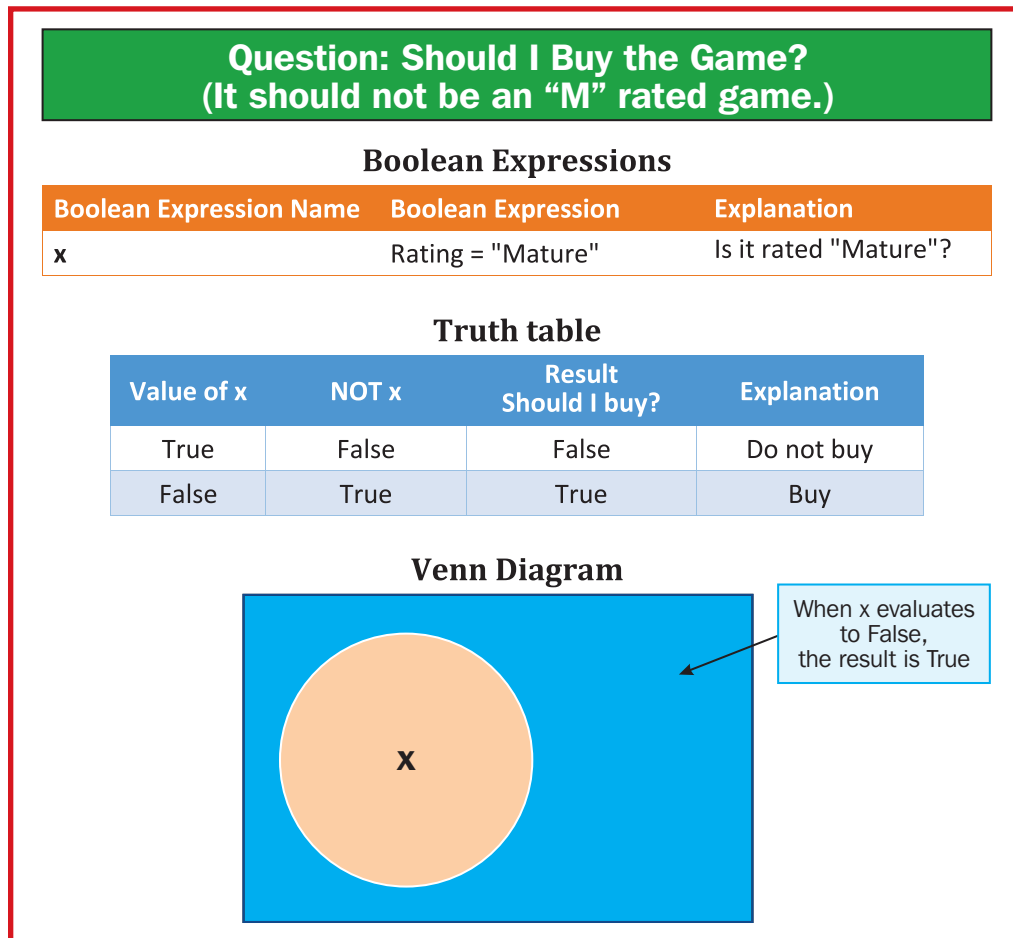




FIGURE 13. Truth Table for the NOT logical operator.

## Other Boolean Operators

Programming languages build most logic structures using the three basic Boolean operators discussed above. However, there are other Boolean operators, and these are shown in FIGURE 14. These are the XOR, NAND and NOR operators, and the figure shows the Truth tables for each of operators. The XOR operator is used in some programming languages to manipulate individual bits in a byte and belongs to a class of operators called bit-wise operators. The other Boolean operators are used in basic electronic classes to discuss hardware gates.

## Truth Tables

Truth	False
	

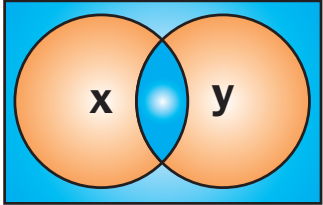
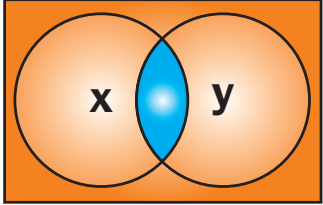
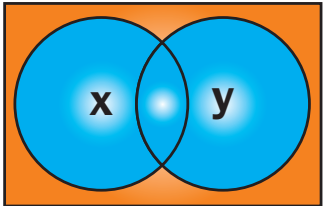
Operator	Truth Table				Venn Diagram
<b>XOR</b> (Exclusive OR)	<b>Value of x</b>	<b>Value of y</b>	<b>x XOR y</b>	<b>Result</b>	
	True	True	True XOR True	False	
	True	False	False XOR True	True	
	False	True	True XOR False	True	
	False	False	False XOR False	False	
<b>NAND</b> (NOT AND)	<b>Value of x</b>	<b>Value of y</b>	<b>x NAND y</b>	<b>Result</b>	
	True	True	True NAND True	False	
	False	True	False NAND True	True	
	True	False	True NAND False	True	
	False	False	False NAND False	True	
<b>NOR</b> (NOT OR)	<b>Value of x</b>	<b>Value of y</b>	<b>x NOR y</b>	<b>Result</b>	
	True	True	True NOR True	False	
	False	True	False NOR True	False	
	True	False	True NOR False	False	
	False	False	False NOR False	True	

FIGURE 14. Other Boolean operators.

## Boolean Algebra and Logic Gates

Boolean algebra lends itself to the creation of electronic devices. A **Logic Gate** is a hypothetical electronics device that implements Boolean logic by controlling the flow of electricity into a device. Logic gates can be implemented using transistors or vacuum tubes, and, theoretically, a computer can be considered an aggregation of various Logic gates. This section uses logic gates to graphically demonstrate Boolean expressions. See FIGURE 15 for Boolean logical operators and the symbols used to depict them as logic



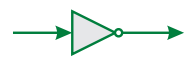
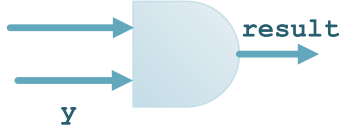
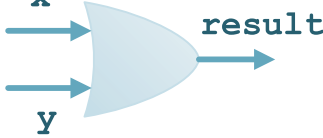
Boolean Logical Operator	Logic Gate
AND	
OR	
NOT	

FIGURE 15. Logic gates.

### Truth Table for **AND** Logical operator using Logic Gates

Value of x	Value of y	x AND y	Result	Logic Gate representation
True	True	True AND True	True	
False	True	False AND True	False	
True	False	True AND False	False	
False	False	False AND False	False	

### Truth Table for **OR** Logical operator using Logic Gates

Value of x	Value of y	x OR y	Result	Logic Gate representation
True	True	True OR True	True	
False	True	False OR True	True	
True	False	True OR False	True	
False	False	False OR False	False	

### Truth Table for **NOT** Logical operator using Logic Gates

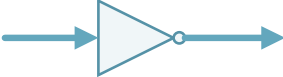
Value of x	NOT x	Result	Logic Gate representation
True	False	False	
False	True	True	

FIGURE 16. Logic gate representations.

gates. See FIGURE 16 for examples of “AND”, “NOT”, and “OR” operators shown graphically using Logic gates.

## Summary:



Computer programs are coded with many different complex conditions that make use of simple and complex Boolean expressions. These require a thorough knowledge of relational and logical operators. When complex Boolean expressions are created, truth tables must be developed to understand all outcomes of the expression. This lesson helps develop an understanding of Boolean expressions, relational and logical operators.

## Checking Your Knowledge:



1. What is a Boolean expression?
2. How are string entities compared with one another?

3. Describe the symbols used in programming languages to compare 2 values with one another.
4. What operators are used to combine multiple Boolean expressions?
5. What is the purpose of the “NOT” operator?

## Expanding Your Knowledge:

---



Reflect upon an App that you use often. Create a PowerPoint presentation with 4-5 Boolean expressions used in the App. Share your PowerPoint presentation with the rest of the class.

## Web Links:

---



### Logic Gates Basics

<https://youtu.be/Xi18hI1LqAA>

### This Simple Math Concept Went Nowhere For A Century And Then — BOOM — Computers

<https://www.businessinsider.com/boolean-algebra-computers-2014-7>

### Understanding ASCII and Unicode (GCSE

<https://youtu.be/5aJKKgSEUnY>