| [Print](javascript:window.print()) | [Close](javascript:parent.window.close()) |

**Course Transcript**

Introduction to Software Program Design

**Program components**

| [1. Variables, Constants, and Data Types](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t3) |

| [2. Expressions and Statements](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t5) |

| [3. Building Expressions](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t7) |

**Arrays and Procedures**

| [1. Arrays](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t12) |

| [2. Procedures and Functions](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t14) |

| [3. Using Arrays and Procedures](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#t16) |

Variables, Constants, and Data Types

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how to create a variable*
* *recognize the conventions for naming a variable*
* *identify the types of data stored in variables*

**1. Creating variables**

At the simplest level, a program is a set of instructions that processes inputs and outputs. Input data can take many forms and can come from various sources. For instance, data can be captured from keyboard or mouse input.

Graphic

*The diagram of a program shows the input of data – from a keyboard or mouse, for example – that is then processed. The data can be saved to and retrieved from storage. The processing of the data produces output in the form of information. This output can be displayed on a screen or printed on paper.*

Input data is stored in a computer's memory and is accessible to a program via its memory address. However, memory addresses are represented in hexadecimal format and are very unwieldy for humans to use.

Variables solve this problem by assigning user-friendly names to data. A program then performs the hard work by keeping track of each variable in memory.

The trade-off for this convenience is reduced processing speed. Every time you want to read a variable's data from memory, a computer has to look up the memory address to access the data. However, some programming languages – like C++ and Assembler – allow you to access data directly using its memory address.

Programming languages also support the use of constants, which is data that is set and remains the same while a program runs. One example of a constant could be the mathematical number pi.

Creating a variable usually involves three steps:

**declaring the variable**

Naming a variable is known as *declaring* the variable. All variables have names that you use to access the data they store. The name of a variable should describe the data it will contain, but must also adhere to specific naming conventions.

**defining the data type, and**

A data type defines the kind of data a variable can contain, which can include text, special characters, or numbers. Most programming languages require that a variable be assigned a certain data type. This helps ensure that data is stored correctly and prevents incompatible types of data from being used together.

**initializing the variable**

The process of assigning data, or a value, to a variable so it can be stored in memory is known as *initializing* the variable. A variable can typically contain only one piece of data at a time. If you assign another value to it, the new value will replace the existing data. One way around this, however, is to use arrays, which are designed to store a number of variables together under the same name.

It's best practice to declare variables at the start of code. However, some programming languages do allow you to declare variables anywhere in code, as long as they're declared before they're used.

Each variable must be given a unique name to ensure that the correct data is stored and can be retrieved using that name.

Computer memory used to be relatively expensive, which meant that variables were typically given very short names to save on space. Simple variable names like X or Y were commonly used.  
  
This made it difficult to remember which variable names represented which values, and you had to go back and check your source code to figure out what a particular variable was used for.  
  
Modern variable names, however, can be longer and more descriptive.

Code

Dim markupPercentage As Integer = 10

When naming variables, it's important to adhere to three basic naming conventions:

* a variable name shouldn't consist of any keywords that are reserved in the programming language you're using
* a variable name can't begin with a number, and
* a variable name can't contain special characters that are disallowed by the programming language

Keywords have particular code meanings in programming languages and so may not be used as variable or constant names. For example, FOR is a keyword in BASIC and var is a keyword in PASCAL. Neither of these words could be used as constant or variable names in the respective languages.

Variable or constant names may not begin with digits or certain characters. For example, the variable name "2ndName" wouldn't be allowed because it begins with a digit. Instead you could name this variable "name2."

Certain languages don't allow the use of special characters. BASIC, for example, doesn't allow the use of characters such as the asterisk (\*), plus sign (+), or parentheses (()). Using these characters would result in errors.

To make variables easier to distinguish, you can name them using strings of words that describe the data they store.

For example, if your program needed to calculate sales totals using Value Added Tax, you could combine the words to name a variable "valueAddedTax."

Two casing styles commonly used in programming are

**camel case, and**

When you string together a name using camel casing, the first letter of the variable name is written in lowercase. The first letter of each subsequent word is written in uppercase. An example is valueAddedTax.

**Pascal case**

Pascal casing is similar to camel casing except that the first letter of each word, including the first word, is written in uppercase. An example is ValueAddedTax.

Some programming languages like C++ are case sensitive, so it's important to be consistent with capitalization throughout the program. For example in C++, you can use both ValueAddedTax and valueAddedTax as variable names. However, C++ will assume that you're referring to two different variables.

The maximum length of a variable name depends on the programming language being used. In PASCAL, for example, the maximum length is 32 characters.

Question

Which naming conventions apply to variable names?

**Options:**

1. They can't consist of keywords
2. They can't begin with a number
3. They can't contain special characters
4. They must be at least eight characters long
5. They have to start with a capital letter

Answer

***Option 1:****Correct. Variable names can't be keywords, which have particular code meanings in a programming language and are reserved for performing specific functions.*

***Option 2:****Correct. Variable names can't begin with numbers, although they can contain numbers.*

***Option 3:****Correct. Variable names can't contain special characters that are disallowed by the programming language you're using.*

***Option 4:****Incorrect. There is no minimum limit on the number of characters in a variable name. However, a maximum length restriction may be imposed by the programming language.*

***Option 5:****Incorrect. Variable names can start with uppercase or lowercase letters. In languages that are case-sensitive, you need always to use the same capitalization when naming and referring to a particular variable.*

**Correct answer(s):**

1. They can't consist of keywords  
2. They can't begin with a number  
3. They can't contain special characters

**2. Data types and values**

Variables can contain various types of data, ranging from character strings and numbers to combinations of strings and numbers.

Data can take the form of

**numbers**

A variable of the Integer data type can store whole numbers. You can use different sub-types depending on the size, or length, of the numeric data you need to store. For example, a variable of the Byte data type is 8 bits long and can store numbers from 0 to 255.  
  
Another sub-type is also called Integer and is usually declared using int. This stores a larger integer number range than the Byte data type, but less than the Long integer data type, which can store the largest range of integer numbers.  
  
The Real data type stores floating point numbers, which are numbers that have decimal values. There are two Real sub-types – Single Precision and Double Precision. A Single Precision variable requires 4 bytes of storage and a Double Precision variable uses 8 bytes.

**text, or**

The String data type is the main data type you'll use for text and can house a virtually unlimited number of characters at a time. It can store letters, special characters, and numbers.   
  
It's important to note than even though you can store a number in a String variable, you can't then use this number in a calculation. You'd have to convert the String variable to the Integer or Real data type before doing so.  
  
A variable of the Char data type can store only one character at a time.

**Boolean values**

The Boolean data type includes two values – such as 0 and 1, or true and false – to represent logical conditions. When you assign a value to a Boolean variable in programming, you typically use the term true or false.  
  
Note that if you enclose either value in quotation marks – using "true", for example – it will be interpreted as a literal string. If you do this for a Boolean variable, a program will return an error.

The sizes of the numerical data types and the range of numbers they can store depend on the programming language you use. It's advisable to research this before you start writing a program.

Some programming languages define subsets of the basic types of data. In PASCAL for example, the Long, Short, and Unsigned data types are variations of the basic Integer data type.  
  
Also, some languages use different names for the same types of data. For instance, numbers with a decimal point are of the Real type in PASCAL but are defined as of the Float type in C.

Both the Real and Integer data types can be positive or negative. Values without a minus sign are assumed to be positive values.

It's important to find the right data type for the data you want to store. This helps ensure that the data is correctly identified and used. It also enables you to be economical with storage space and memory.

If you want to store a person's age in a variable, for example, using the Integer sub-type would be unnecessarily large. In this case, the Byte data type would be more efficient.   
  
Similarly, if you use a variable of the Byte data type to store a phone number, only the first few digits of the number will be saved and the rest will be discarded. And if you use a variable of the Integer data type to store a real number, the decimal value of the real number will be discarded.

Question

Which data types can contain only numerical values?

**Options:**

1. Integer
2. Real
3. Char
4. String

Answer

***Option 1:****Correct. You can store only whole numbers in variables of the Integer data type.*

***Option 2:****Correct. You can store real and whole numbers in a variable of the Real data type, but a whole number will have two decimal digits added to its end.*

***Option 3:****Incorrect. You can store only ASCII characters in a variable of the Char data type.*

***Option 4:****Incorrect. You can store text, special characters, and numbers in a variable of the String data type.*

**Correct answer(s):**

1. Integer  
2. Real

Many programming languages perform an operation called *type-checking*, which validates data to ensure that the data is compatible with the associated variable's data type.

Type-checking returns an error if a program attempts to store data that doesn't match the variables data type, like trying to store a text string in a number variable. To prevent type-checking errors, the programmer writes code to validate input and thus filter out potentially troublesome data.

For example, if you've declared a variable that stores a person's age, you should write commands that validate the age number input by the user. You would need to create a function that checks if the input number is a positive number and below or equal to the maximum age humans are generally expected to reach. Failure to program these input checks often lead to type-checking errors.

Variables are initialized when values are assigned to them. It's good practice to assign a value to a variable when you declare it, even if this is a zero for a numerical variable or "" for a string.

Note

*Some programming languages will automatically assign 0 – or "" for a string – if you don't assign a value when declaring a variable.*

The way that values are assigned to variables differs from one programming language to another. In C and C++, for example, you specify a data type, then the variable name, and then the variable's initial value in parenthesis.

Code

int i(0);

In Java and JavaScript, you use var followed by the variable name. You then include a colon and specify the data type. Finally you assign an initial value using an equals sign.

Code

var i:int = 0;

Note

*In C, C++, Java, and JavaScript, the end of any line is indicated by a semi-colon (;).*

In Visual Basic, you use Dim followed by the variable name. You follow this with As and the variables' data type, an equals sign, and then the value you want to assign to the variable.

Code

Dim i As Integer = 0

When you assign a string value to a variable, you enclose the text in quotation marks.

Code

String FirstName("");

Say you're using C++ to write a program and want to store the name John in a string variable named FirstName. You start by declaring the data type and the variable name, and follow it with the value, in quotation marks and inside parentheses.

Code

String FirstName("John");

Question

Which are valid variable declarations in C++, JavaScript, or Visual Basic?

**Options:**

1. Boolean status(true);
2. var status:Boolean = true;
3. Dim status As Boolean = true;
4. Boolean status = "true";
5. var status:Boolean = "true";

Answer

***Option 1:****Correct. In C++, you declare a variable by specifying its data type, then its name, and then its value in parentheses.*

***Option 2:****Correct. In JavaScript, you declare a variable by specifying the var keyword, followed by the name of the variable, a colon, the variable's data type, an equals sign, and finally the value you want to assign to the variable.*

***Option 3:****Correct. In Visual Basic, you declare a variable with Dim, followed by the name of the variable, the As operator, the required data type, and an equals sign followed by the initial value of the variable.*

***Option 4:****Incorrect. The Boolean data type can accept only true or false as its value. When either of these values is enclosed in quotation marks, it's identified as a string value and can't be stored in the Boolean data type.*

***Option 5:****Incorrect. If you use quotation marks while assigning a value, you indicate that the value is a string. However, the Boolean data type doesn't accept strings. It accepts only trueor false, without the quotation marks, as values.*

**Correct answer(s):**

1. Boolean status(true);  
2. var status:Boolean = true;  
3. Dim status As Boolean = true;

**3. Summary**

The data processed by programs is stored in variables or as constants. A constant is declared at the beginning of a program and is a value that remains unchanged, whereas the value of a variable can change during normal program execution. To create a variable, you declare it by naming it, define its data type, and initialize it by assigning it a value. You need to adhere to a number of naming conventions for variables to ensure correct program execution.  
  
Programming languages use a number of data types and sub-types to identify which data particular variables can store. It's important to set variables to the correct data types so that values are stored correctly, and for program efficiency. Many programming languages provide type-checking, which raises an error if the wrong data type is assigned to a particular variable. Properly written programs include code to validate input data thus preventing type-checking errors.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

Expressions and Statements

Learning Objectives

*After completing this topic, you should be able to*

* *identify the functions of mathematical and comparison operators*
* *recognize how operators are evaluated*

**1. Mathematical expressions and operators**

Expressions and statements form a large part of developed applications. An expression is a combination of operators and operands that determines a value. This example expression shows 22 divided by 7.

Graphic

*An example is 22 / 7.*

A statement is an instruction in a computer language that performs a task. Assignment operators are used to instruct the program to assign values to data. In the example, the variable X equals 3.14.

Graphic

*An example is X = 3.14.*

You can use an expression with a statement to calculate a value and store it in a variable. When you do this, the variable name is always placed on the left of the operator. The expression is written on the right of the operator. In the example X equals 22 divided by 7, X is the variable and 22 divided by 7 is the expression.

Graphic

*An example is X = 22 / 7.*

The simplest operator is the equals sign, or =. It's used to assign values. In this example, the variable variableOne is assigned the value 22. You can assign a numerical or string value, or the result of a mathematical expression, to a variable.

Graphic

*The example is variableOne = 22.*

The term operands refers to everything in a statement that isn't an operator. In this example, variableOne and 7 are operands, whereas = and / are operators.

Graphic

*The example is result = variableOne / 7.*

Assignment operators in a program function differently from operators in mathematics. In programming, the compiler returns an error because it will try to assign the value on the right of the operator to the expression on the left. In this example, X multiplied by 2 equals 10.

Graphic

*An example is X \* 2 = 10.*

To remedy this, the statement is written the other way around with the expression on the right of the assignment operator. In this case, the example is changed to X equals 5 multiplied by 2.

Graphic

*The example is changed to X = 5 \* 2.*

You can assign a new value to a variable by using it as an operand in the expression and performing an operation on it. In this example, X equals X multiplied by 2. If X has an initial value of 10, the new value of X will be 20 when the statement is executed.

Graphic

*The example is X = X \* 2.*

The operators you'll use most often are mathematical operators.

**()**

Parentheses are used to prioritize expressions within other expressions. The result of an expression in parentheses is used as an operand in the main expression. Operators and terms within parentheses are evaluated according to standard operator precedence. In the example p = (X + 2) / 7, the expression in parentheses, X + 2, is prioritized.  
*The example shown is p = (X + 2) / 7.*

**^**

The caret symbol is the exponentiation operator. It's used to multiply a number exponentially a specified number of times. In the example s = p \* r ^ 2, r is the base number and 2 is the exponent.  
*The example s = p \* r ^ 2 is shown.*

**\***

The asterisk is the multiplication operator, used to multiply one number or variable by another. As in the example, d equals r multiplied by 2.  
*The example d = r \* 2 is shown.*

**/**

The forward slash character is used for division operations using integers or floating point numbers. The result will be rounded to an integer if p is an integer data type, otherwise, the result will be a Real number with a quotient. In the example, p equals X divided by 7.  
*The example is: p = X / 7.*

**\**

The backslash character is used for integer division operations. Integer division returns a whole number and any decimal value will be discarded, even if the result is stored in a variable of the Real data type. In the example, q equals the integer value of 22 divided by 7, q has a value of 3 once the statement is executed.  
*The example q = 22 \ 7 is shown.*

**+**

The plus sign is used for addition operations. In this example the value of X is replaced by the result of the expression X plus 2. If you have an expression like X = 10 + -2, the result will be 8.  
*The statement X = X + 2 is shown.*

**-**

The minus sign is used to subtract one number or variable from another. If you use two -characters, like in Y = 22 -- 7, the operation will be changed to addition and Y will be 29.  
*The example is Y = 22 - 7.*

**%**

A modulo operation divides one number with another and returns the remainder. In the example Z equals Y % 2. if Y is 15, Z will have the value of 0.5 when the statement is executed. The percentage symbol represents a modulo operation in languages such as C#, Java, and Visual Basic. In other languages, the mod command is often used.  
*Two examples are shown – Z = Y % 2 and Z = Y mod 2.*

Programming languages use particular rules of precedence when there's more than one operator in an expression. Operations will be executed in this order:

* expressions in parentheses
* exponentiation
* multiplication and division
* integer division
* modula, and
* addition and subtraction

Operators that use two different operands to produce a result are called *binary operators*.

Graphic

*The example is:   
int a;  
int b;  
int c;  
  
a = 10;  
b = a++;  
c = --b;  
  
The three integer declarations are highlighted.*

In C++, you can use *unary operators*, which operate on a single operand. You can use either ++ to increment a value by one, or -- to decrement a value by one. The result of using these operators in statements depends on which side of the operand you place them on.

Graphic

*The highlighted example is a = 10.*

Note

*C++ was developed from C and gets its name from the ++ operator, which essentially means "one more."*

In the first example, the unary operator is on the right – so when this expression is executed, bequals 10 and a is increased to 11. This is because the value of b is assigned before the unary operation is performed.  
  
If the unary operator is written on the left of the operand, the unary operation is performed before the value is assigned. So in the second example, both b and c will be 9 because the value of b is decreased before it's assigned.

Graphic

*The highlighted examples are:   
b = a++;  
c = --b;*

Mathematical operators are used to change numbers but *comparison operators* are used to compare values. The result of a comparison is returned as a Boolean value – either true or false.

Graphic

*Example of comparison operators include = or == which means equal to, <> or != =which means not equal to, < which means less than, > which means greater than, <= which means less than or equal to, and >= which means greater than or equal to.*

Comparison expressions are often used to make decisions on which step a program should perform next.

There are a number of comparison operators.

Graphic

*The example is a = 12, b = 10, and c=10.*

**= or ==**

The *equal* comparison operator is denoted by one or two = signs. You use it to check whether two values are equal. The single = sign is usually used to assign a value, but in some languages this is also a comparison operator. In Java, JavaScript, and C++, == is distinguished as the equal operator. In this example, the expression will be true.  
*The example is b == c. Both b and c equal 10.*

**<> or !=**

The *not equal to* operator is represented by the <> or != set of characters, depending on the programming language. You use it to check if one value is not equal to another. In this example the expression will be true.  
*The example is a != c. Variable a = 12 and variable c = 10.*

**<**

The *less than* operator is represented by the < sign. You use it to check if the operand on the left of the operator is less than the one on the right. In this example, the expression will be false.  
*The example is a < b. Variable b = 10 and variable a = 12.*

**>**

The *greater than* operator is represented by the > sign. You use it to check if the operand on the left of the operator is larger than the one on the right. In this example, the expression will be true.  
*The example is a > c. Variable a = 12 and variable c = 10.*

**<=**

The *less than or equal to* operator is represented by the <= set of characters. You use it to check if the operand on the left of the operator is less than or equal to the one on the right. In this example, the expression will be true.  
*The example is b <= c. Both b and c equal 10.*

**>=**

The *greater than or equal to* operator is represented by the >= set of characters. You use it to check if the operand on the left of the operator is greater than or equal to the one on the right. In this example, the expression will be true.  
*The example is a >= b. Variable a = 12 and variable b = 10.*

You can also use comparison operators to compare strings. Each character corresponds with a numerical value in the ASCII character set used by computers. When you compare two characters, the ASCII or Unicode numerical values of the characters are compared. In the example, "j" is greater than "J".

Graphic

*The example is "j" > "J".*

In this example, the ASCII values for the characters are shown. The expression 106 > 74 will return true.

Graphic

*The examples are "j" > "J" and 106 > 74.*

When comparing two strings, a program compares each character in one string with the character in the same position in the other string. If the first characters are the same, it moves through the strings one character at a time until a true or false result is achieved. The example 74, 75 > 74, 107 will return false when the expression is executed.

Graphic

*The examples are "JK" > "Jk" and 74, 75 > 74, 107.*

Programming languages also use *Boolean operators* to calculate Boolean values. Because the results of Boolean expressions are either true or false, they are often used with comparison expressions to return the results of the comparisons.

Most programming languages use four main Boolean operators.

**AND**

The AND operator compares two Boolean values and returns a single Boolean value. If both values are true, it returns true, but if one or both is false, it returns false. In this example, payment = 550, therefore both comparison expressions will return true, so the result of this Boolean expression will also be true.  
*The example shown on screen is:   
(payment < 650) AND (payment > 450)*

**OR**

The OR operator is the opposite of the AND operator and will return true if any of the compared values are true. For a false result, both Boolean values have to be false. In this example, payment = 550, therefore the second comparison expression is true, so the result will also be true.  
*The example is (payment < 450) OR (payment >= 500).*

**XOR**

The XOR operator, also known as the *exclusive or* operator, returns a true value if one but not both of the operands is true. So if you compare two true or two false values, the result will be false – but if one value is true, it returns true. In this example, payment = 500, therefore the first expression is false and the second true, so the result will be true.  
*The example is (payment != 500) XOR (payment == 500).*

**NOT**

The NOT operator changes a Boolean value to its opposite. In this example, payment = 500, therefore the result will be false because the comparison expression is true.  
*The example is NOT (payment < 650).*

An example in which Boolean operations are used is an automated road toll system where drivers buy credit before using a road. In this example, a car is allowed through with true expression results.

Graphic

*The example statements are  
restricted = false;  
registered = true;  
credit = false;  
VIP = true;*

When a driver approaches, the first evaluation determines whether the car is restricted and so not allowed to enter the toll road. If it's NOT restricted, the program passes to the next evaluation.

Graphic

*The statement is NOT(restricted).*

Then if a car is registered AND has available credit, it's allowed to pass. In addition, if credit is false but the car is marked as VIP, the car will be allowed through as result of the OR operation.

Graphic

*The statement is (registered AND credit) OR VIP.*

Along with basic mathematical operators, most programming languages have built-in math functions for performing more complex calculations. Functions like square root or logarithm are either incorporated natively or can be added using math libraries.

These functions work on the same principles as other programming functions and take numerical values as parameters.

Graphic

*An example is X = sqrt(360).*

Question

Match the mathematical and comparison operators with their descriptions.

**Options:**

1. ^
2. \
3. /
4. %
5. !=
6. <

**Targets:**

1. Multiplies a number exponentially
2. Divides a number into another and returns a whole number
3. Divides a number into another and returns a real number with a decimal value if the variable has been assigned as Real
4. Computes the remainder of a division operation
5. Compares two values and checks if they aren't equal
6. Compares two values and checks if the first is smaller than the second

Answer

*You use the caret symbol, or exponentiation operator, to multiply a number exponentially.*

*You use the backslash for integer division. It returns a whole number and discards decimal values.*

*You use the forward slash for division. It returns a real number with a decimal value if the variable has been assigned as Real.*

*You use either the percentage symbol or mod keyword to divide two numbers and return the remainder.*

*You use != or <> for not equal comparison operations. The result is returned as a Boolean value.*

*You use the less than symbol to check whether one value is smaller than another. The result is returned as a Boolean value.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Target 5 = Option E

Target 6 = Option F

**2. Operator precedence**

Operators are evaluated in a specific order according to operator precedence. This means that certain operators have precedence over others when written in an expression.

Operators between parentheses are given the highest precedence, and the results are used for further calculations. Terms within parentheses are evaluated according to standard operator precedence.

Two common mnemonics can help you remember the order of operator precedence:

**PEMDAS, and**

PEMDAS is an acronym for parenthesis, exponentiation, multiplication, division, addition, and subtraction.

**BEDMAS**

BEDMAS is an acronym for brackets, exponentiation, division, multiplication, addition, and subtraction.

Integer division and modulo are omitted from this list because these are more obscure operators and not often used in programming.

The words "brackets" and "parentheses" are interchangeable and have the same meaning. The order in which multiplication and division, or addition and subtraction occur is also irrelevant, because the results will be the same.

If operators in an expression have equal precedence, calculation occurs from left to right.

Say you want to buy something from a web store based in another country. You want to calculate the total cost in dollars before buying the item.

Graphic

*Two expressions are shown - unitCost equals unitPrice plus shipping, and dollarValue equals unitCost multiplied by exchangeRate. These are represented as:  
unitCost = unitPrice + shipping  
dollarValue = unitCost \* exchangeRate*

You can use two expressions to do this, but combining them into one expression would be more efficient. If the two expressions are simply combined, however, you won't get an accurate answer because operator precedence dictates that multiplication will occur before addition.

Graphic

*The expression dollarvalue equals unitPrice plus shipping multiplied by exchange rate. This is represented as dollarValue = unitPrice + shipping \* exchangeRate*

If you add parentheses to the first expression, the cost of shipping the unit will be added to the unit price before the exchange rate is multiplied. This will return the correct answer.

Graphic

*The expression dollarvalue equals open parenthesis unitPrice plus shipping closing parenthesis multiplied by exchange rate. This is represented as dollarValue = (unitPrice + shipping) \* exchangeRate*

You can construct complex statements as long as you factor in operator precedence.  
  
Say you want to find the exponential result of a base value using a variable exponent. Currently the value of the exponent is 2.

Graphic

*The expression is exp = 2.*

When the statement in this example is executed, value will be 100 because the ++ unary operation will occur only after the exponential operation is performed.

Graphic

*The expression is value = 10^exp++.*

However, if you include parentheses in the expression, value will be 1000 because the operation inside the parentheses will be performed before the exponentiation operation.

Graphic

*The expression is value = 10^(exp++).*

Strings play a major role in computer programming. As with numbers, you can perform addition and subtraction operations on strings.

Some programming languages also enable you to search a string for a character or set of characters, and replace or remove them.

More complex operations are usually performed using functions, but some operators will also work with strings.

Say you want to print a person's name on the screen.

Graphic

*Two variable values are shown – FirstName = "Emily" and LastName = "Novak".*

To do this, you add the two variables to form a new string variable, and print this to screen. It's important to remember to add a space between the variables, which you do by using a set of quotation marks surrounding a space.

Graphic

*The example statements are person = FirstName + " " + LastName and PrintToScreen person.*

Alternatively, you can combine the two statements into one print statement.

Graphic

*The example statement is PrintToScreen FirstName + " " + LastName.*

You can also add text to an existing string variable using the addition operator.

Graphic

*The expression Name equals FirstName plus Novak. This expression is represented as Name = FirstName + " Novak".*

Question

Using standard operator precedence, match each expression with the correct result.

**Options:**

1. X = 8 - 3 + 7
2. Y = (34 + 7) \* 89
3. Z = 7 + 7 \* 7
4. V = 882 / 3^2

**Targets:**

1. 12
2. 3,649
3. 56
4. 98

Answer

*In this statement, the - and + operators have equal precedence. So the expression is simply calculated from left to right.*

*In this statement, the expression between parentheses takes precedence. So the result is calculated as 41 multiplied by 89.*

*In this statement, the multiplication operator takes precedence. So the result is calculated as 7 plus 49, which equals 56.*

*In this statement, the exponentiation operation takes precedence. So 882 is divided by 9 to give the result of 98.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

**3. Summary**

Programs typically contain a large number of expressions and it's vital that they're accurate. A number of different types of operators are used to form expressions. Of these, mathematical operators are the most common. A number of mathematical functions are also generally available. Comparison operators are used to compare two values and return a Boolean value. In programming, Boolean operators are most often used for decision making.  
  
It's important to know the rules of operator precedence when creating expressions. Two common mnemonics for remembering these rules are PEMDAS and BEDMAS. In some languages, you can manipulate strings in ways similar to those you use to manipulate numerical values.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

Building Expressions

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how variables, operators, and values are used to construct expressions*
* *recognize how to build expressions*

**1. Exercise overview**

In this exercise, you're required to demonstrate a basic understanding of variables and expressions in programming.

This involves the following tasks:

* recognizing how to declare variables, and
* recognizing how to build expressions

**2. Declaring variables**

Question

Which term describes data that is preset and remains unchanged during processing?

**Options:**

1. Constant
2. Variable
3. Operator

Answer

***Option 1:****Correct. A constant is a value that is declared in source code and remains unchanged during processing.*

***Option 2:****Incorrect. A variable is a placeholder for data that is declared in the source code. The value it stores can change during program execution – the variable may receive a value from user input or as the result of an expression.*

***Option 3:****Incorrect. An operator is a character that represents a mathematical or logical operation to be performed on data.*

**Correct answer(s):**

1. Constant

Question

Which variable names follow the standard naming conventions?

**Options:**

1. secondNumber
2. SecondNumber
3. 2ndNumber
4. second()
5. \*Numbers

Answer

***Option 1:****Correct. It's acceptable to write a variable name with an initial word in all lowercase and subsequent words beginning with an initial uppercase letter. This is known as camel case. The name in this example also doesn't violate any other conventions for variable names.*

***Option 2:****Correct. It's acceptable to include an initial capital for each word in a variable name. This is known as Pascal case. The name in this example also doesn't violate any other conventions for variable names.*

***Option 3:****Incorrect. Variables may not start with a number. You can add the number after the name if required.*

***Option 4:****Incorrect. Parentheses, or brackets, aren't allowed in variable names.*

***Option 5:****Incorrect. The asterisk is a special character – it's used as an operator in programming languages and so can't be included in the names of variables.*

**Correct answer(s):**

1. secondNumber  
2. SecondNumber

Question

Match each data example with its data type.

**Options:**

1. true
2. 254
3. a
4. 17.2553

**Targets:**

1. Boolean
2. Byte
3. Char
4. Real

Answer

*The Boolean data type can represent true or false values. It stores 1 or 0 to reflect the value.*

*The Byte data type can store up to eight bits of data and a number in the range from zero to 255. You can also save the number 254 in a variable of the Integer type, but good programming conventions dictate that the size of the data type should be as closely matched to the value as possible.*

*The Char data type can store a single character at a time. You can also use the String data type for this value – but if the value of this variable will always be only one character, using Char is the better solution.*

*The Real data type can store real numbers that have decimal values. Whole numbers can also be stored, but will be written with a decimal value of .0.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Question

When you name a variable, what are you actually doing?

**Options:**

1. Declaring the variable
2. Initializing the variable
3. Validating the data type

Answer

***Option 1:****Correct. When you declare a variable, you give the variable a name and specify its data type.*

***Option 2:****Incorrect. A variable is initialized when it's given a value. Some programming languages will assign 0, or "" for strings, if you don't assign a value explicitly.*

***Option 3:****Incorrect. Data type validation is done automatically by a programming language when data is entered.*

**Correct answer(s):**

1. Declaring the variable

**3. Building expressions**

Question

Using standard operator precedence, match each expression with the correct result.

**Options:**

1. 2 + 3 \* 2
2. (2 + 3) \* 2
3. 2 + 3 ^ 2
4. (2 + 3) / 2
5. (2 + 3) \ 2
6. 2 + 3 / 2

**Targets:**

1. 8
2. 10
3. 11
4. 2.5
5. 2
6. 3.5

Answer

*In this statement, the multiplication operator takes precedence. So the result is calculated as 2 plus 6.*

*In this statement, the expression between the parentheses takes precedence. So the result is calculated as 5 multiplied by 2.*

*In this statement, the exponentiation operator takes precedence. So the result is calculated as 2 plus 9.*

*In this statement, the expression between the parentheses takes precedence. So the result is calculated as 5 divided by 2.*

*In this statement, the expression between the parentheses takes precedence. The backslash character is used for integer division operations and returns the integer value of 5 divided by 2, which equals 2.*

*In this statement, the division operator takes precedence. So the result is calculated as 2 plus 1.5.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Target 5 = Option E

Target 6 = Option F

Question

Identify the order in which the components of expressions are evaluated, from first to last.

**Options:**

1. Expressions in parentheses
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction

Answer

**Correct answer(s):**

**Expressions in parentheses is ranked**

Parentheses take precedence over all other operators, so the expressions they contain will always be evaluated first.

**Exponentiation is ranked**

Operations involving exponentiation are performed after any expressions in parentheses have been evaluated, but before operations involving multiplication or division.

**Multiplication and division is ranked**

Multiplication and division are performed after exponentiation but before addition and subtraction.

**Addition and subtraction is ranked**

Addition and subtraction are at the lowest level of precedence and are performed last.

Data types and naming conventions for variables have been detailed and mathematical operators have been used to build expressions.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

Arrays

Learning Objectives

*After completing this topic, you should be able to*

* *recognize how to use arrays in programs*
* *recognize how to create arrays in BASIC, C++, and Revolution*

**1. Using arrays**

All computer programs need to store data and variables are the most common way to do this. However, variables are limited to storing only one piece of data at a time. For example, if you want to store an employee's first name, last name, and telephone number, you'd have to create three variables – which isn't very efficient.

Structures and arrays were created as two ways to store several pieces of related data in one place.

Structures provide a way to group several variables, and can be described as variables that can contain several other variables.

Arrays offer a better solution. They can be described as "super" variables to which you can add as many pieces of related data as you want.

For example, if you want a program to store the names of all 20 employees in a company, you can create one array to store them.

When you create an array, you need to define

**a variable name**

When you create an array, you have to give it a name. It's best to give arrays descriptive names so you know what kind of data they store.

**the array size, and**

The advantage of arrays is that they can each hold many pieces of data. Each piece is stored in an element of the array. When you create an array, you have to specify how many elements it will contain. This is known as the size of the array.

**the type of data you want to store**

All the elements in an array must have the same data type. You specify the data type when you create the array, although there are commands that let you convert to a different data type later.  
  
If an array is for storing names, for example, you should specify the data type as String. Other data types you can set are Integer and Boolean. Storing numbers is, however, not limited to the Integer data type. For example, if you want to store names and telephone numbers in one array, you can set the data type as String and store the numbers as strings.

It's important to plan before you start creating an array so you can specify its size. You set the size by specifying the bounds of an array. The lower bound defines the number of the first array element and the upper bound defines the number of the last array element.  
  
Depending on which program language you're using, the default lower bound can be either zero or one. It's important to know what the default lower bound is so that you can specify enough elements for your data.

Programs such as BASIC, Java, and C – and any language derived from the C language – always define the lower bound of an array as zero. These are called zero-based arrays. To specify the number of an element, you have to start counting at zero.

Other programs use one-based arrays. To set the size of a one-based array, you have to start counting at one.

Some programs, such as Pascal, enable you to define both the lower and upper bounds of an array. The advantage of this is that the lower bound doesn't have to start at zero or one, but can start at any number you want.

This is useful when you want to link each element to a meaningful number. For example, if your company uses employee numbers and you're creating an array to store employee names, you can set the lower bound to correspond to the first employee number.

Most arrays are one-dimensional and store data in the form of a list. So they are defined only in terms of their length.

You can, however, create multi-dimensional arrays. These are more flexible and enable you to store more complex data.  
  
The most common multi-dimensional array is a two-dimensional array, which can be represented as a grid with either two columns or two rows.

You can create arrays that are two-, four-, or even nine-dimensional. But the more dimensions arrays have, the more complex they are. With too many dimensions, working with arrays can be rather confusing.

Question

Identify what you need to specify when creating an array.

**Options:**

1. A name for the array
2. The size of the array
3. The type of data the array will store
4. How the data will be entered
5. The language used for data in the array

Answer

***Option 1:****Correct. When you create an array, you need to give it a name – preferably a name that describes the data it will store.*

***Option 2:****Correct. When you create an array, you have to specify how many elements it should contain.*

***Option 3:****Correct. All the elements in an array must have the same data type, which you need to specify when you create the array.*

***Option 4:****Incorrect. You don't have to consider how the data will be entered when creating an array. You need to give an array its name and specify its size and data type.*

***Option 5:****Incorrect. You don't need to specify language settings when creating an array, although you do need to consider what type of data the array will contain so you can specify the right data type.*

**Correct answer(s):**

1. A name for the array   
2. The size of the array  
3. The type of data the array will store

**2. BASIC, C++, and Revolution**

Although the general method for creating an array is the same no matter which programming language you use, some of the details vary.

To create arrays in LibertyBASIC, you use the DIM keyword.

Code

DIM

You specify a name for the array, without using spaces or special characters.

Code

DIM EmployeeNames

In LibertyBASIC, an array can accept either strings or numbers. To define the data type as String, you need to add a dollar sign – $ – at the end of the array name. If you don't add the dollar sign, the data type will be defined as Integer by default.

Code

DIM EmployeeNames$

To define the size of the array, you provide the upper bound, which defines the number of the last array element in brackets.  
  
Arrays in LibertyBASIC are zero-based so you start counting from zero to number the elements. In the example, the number four indicates that the array has five elements. The first element is zero, the second is one, and so on.

Code

DIM EmployeeNames$(4)

To create an array in REALbasic, you also use the DIM keyword.

Code

DIM

You specify a name for the array and specify the size of the array by providing the number for the last element in brackets after the name. Arrays are zero-based in REALbasic.

Code

DIM EmployeeNames(4)

To specify the data type of the array, you use a declaration with the DIM command. After the name and size, you simply state As and then provide the data type, such as Integer or String.

Code

DIM EmployeeNames(4) As String

To create an array in C++, you start by specifying the data type. You provide the name of the data type, such as String or Integer.

Code

String

Next you provide the array name.

Code

String EmployeeNames

You then specify the array size. C++ is different from other languages in that you specify the number of elements in the array rather than the upper bound. Also, in C++ you specify the size in square brackets instead of in parentheses. In the example, the number five indicates that the array should contain five elements.

Code

String EmployeeNames[5]

Note

*Arrays in C++ are zero-based so the index numbers start from zero.*

The way you create arrays in Revolution is different from in the other programming languages.  
  
You don't have to create an array in advance – by declaring its name, size, and data type – before you can start adding data. Revolution enables you to add data as you create arrays using the putcommand.

Code

put

Another difference is that items in a Revolution array aren't identified by an index number. Instead Revolution identifies array items by a key, which can be any string or unique number.  
  
In the example, the array named EmployeeNames is created. It stores the name Jonathan Gold and associates it with the key Sales.

Code

put Jonathan Gold into EmployeeNames["Sales"]

Creating two-dimensional arrays in LibertyBASIC and REALbasic is similar to creating one-dimensional arrays. You also use the DIM keyword, specifying the array name and data type.

Code

DIM EmployeeNames$()  
  
DIM EmployeeNames() As String

Specifying the size is different. For a two-dimensional array, you use two integers to indicate the array's size. The first specifies the upper bound of the elements in the horizontal dimension – row – of the grid and the second specifies the upper bound of the elements in the vertical dimension – column.  
  
Because LibertyBASIC and REALbasic are zero-based arrays, you start counting from zero.

Code

DIM EmployeeNames$(2,1)  
  
DIM EmployeeNames(2,1) As String

The first example here is in LibertyBASIC, and the second is in REALbasic. Both create a two-dimensional array named EmployeeNames that's designed to hold six string elements. This array could be represented as a grid with three rows and two columns.

Code

DIM EmployeeNames$(2,1)  
  
DIM EmployeeNames(2,1) As String

Creating two-dimensional arrays in C++ is similar to creating two-dimensional arrays in LibertyBASIC and REALbasic. However, you need to supply two values to specify the number of elements in both dimensions of the grid. Each of these values is specified in its own set of square brackets.   
  
This example shows the code for creating a two-dimensional array, which has three rows and two columns when represented as a grid.

Code

String EmployeeNames[3][2]

Using Revolution, you can enter data as you create two-dimensional arrays, just as you can when creating one-dimensional arrays.

Code

put "Jonathan Gold" into EmployeeNames[0,0]  
put "Alison Baker" into NameArray[0,1]  
put "Eleanor Davis" into NameArray[1,0]  
put "Luke Foster" into NameArray[1,1]  
put "Werner Horowitz" into NameArray[2,0]  
put "Maggie Chung" into NameArray[2,1]

The difference is that where elements in one-dimensional arrays are identified with a key, you have to define each element's place on the grid when creating two-dimensional arrays.

Revolution is a zero-based language. This means, for example, that the first element in both the horizontal and vertical dimensions of the grid will be numbered (0,0). This example shows a two-dimensional array called EmployeeNames that contains six name elements.

Question

Match the programming languages to corresponding examples of the statements used to create an array. One of the languages doesn't match to a statement.

**Options:**

1. LibertyBASIC
2. REALbasic
3. C++
4. Revolution

**Targets:**

1. DIM ClassArray$(10)
2. string ClassArray[11]
3. put Pilates into ClassArray("P")

Answer

*Using LibertyBASIC, you create an array of the String data type by specifying the DIMkeyword, an array name, a dollar sign – which identifies the String data type, and the size of the array in parentheses.*

*Using C++, you start by specifying the data type and then specify the number of elements in square brackets.*

*In Revolution, you can add data as you create arrays, and you identify array items by keys rather than index numbers.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option C

Target 3 = Option D

**3. Implementing arrays**

You assign values to the elements of an array in the same way that you assign values to variables.

In ANSI C, you can initialize the contents of an array when you declare the array. ANSI – short for the American National Standards Institute – published a standard for use with C to make C programs more portable.  
  
The declaration in the example is used to initialize an array of vowels.

Code

char vowel [5] = {'a', 'e', 'i', 'o', 'u' };

When you want to initialize the array of vowels in BASIC or Pascal, you have to assign a value to each element separately.

Code

vowel[1]="a"..vowel[5]="u"

To insert an element into an array that is already filled, you can copy and move all the elements forward one place to the right of the insert position. So the very last element in the array falls away.  
  
If you move all the elements back one place, to the left of the insert position, the very first element in the array falls away.

This may seem easy if you have a small array with only a few elements. But for a large array with thousands of elements, this could be an extremely cumbersome job, especially if you have to do it regularly.

Another way to insert an element in an array is to overwrite the current array element with the new element.  
  
Consider an array of four integers – {19,20,21,22} – called year. You want to insert 0 between 20 and 21 – at position 2, assuming the index numbers in the array start at zero.

The C code inserts 0 at position 2 in the array. The program starts at the end of the array and copies the element to the left of the current position onto the element at the current position. This process repeats until the insert position – which is 2 – is reached. At this point, the array contains the values {19,20,20,21}.

Code

for (index = 3; index >= 2; index--) {     
  year [index] = year [index - 1];  
}

The last statement in the code assigns the value 0 to the third element in the array. Now the array contains the integers {19,20,0,21}.

Code

year [2] = 0;

Now you want to delete 20 from the array {19,20,0,21}.

You can do so by overwriting 20 with a null value.

After the code has been executed, the array contains the four values {19, ,0,21}.

Code

year [2] = 0;

To empty an array in BASIC, you can use the command ERASE. So this code replaces all the integers in the array called "year" with null values.

Code

ERASE year

In C and Pascal, you use a loop to empty an array.

Code

for (index = 0; index <= 4; index++) {  
  vowel [index] = "";  
}

Question

Identify the correct methods for inserting an element into an array that is already filled.

**Options:**

1. Move all the elements up or down one place
2. Re-create the array with the new element
3. Overwrite a current array element with a new element
4. Have the array determine which item to replace

Answer

***Option 1:****Correct. If you move the elements up or down one place, the first or last element falls away to make space for the new element.*

***Option 2:****Incorrect. It's inefficient to re-create an array simply to replace an element. You can either move the elements up or down, or overwrite the element you want to replace.*

***Option 3:****Correct. You can insert an element in an array by overwriting the current array element with the new element.*

***Option 4:****Incorrect. An array can't determine which item to replace. You have to overwrite a specified location in the array with a new value.*

**Correct answer(s):**

1. Move all the elements up or down one place  
3. Overwrite a current array element with a new element

**4. Summary**

Variables are the most common way for programs to store data but they are limited to storing only one piece of data at a time. An array can store multiple pieces of related data in one place. When you create an array, you give it a name and specify its size and data type.  
  
The basic method of creating an array is common to all programming languages, although the details may differ. Some of the common programming languages are BASIC, C++, and Revolution.  
  
Once you've created an array, you need to implement it by assigning values to it. You assign values to the elements of an array in the same way that you assign values to variables.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

Procedures and Functions

Learning Objectives

*After completing this topic, you should be able to*

* *recognize called procedures in code examples*
* *recognize the difference between local and global variables*

**1. Using procedures in programs**

Most modern programs are long and complex. So it's hard to organize all the different tasks a program must perform. To make this process a little easier, it's a good idea to divide a program into major tasks and subtasks.

Code

Enter\_competitors()  
    WHILE more records  
        GET current\_date  
        age = current\_date - date\_of\_birth  
        SET valid\_age to true  
        IF age <18 or age >50 THEN  
            PRINT "You must be between 18 and 50 to enter"  
            valid\_age = false  
        ENDIF  
        IF valid\_age = true THEN  
            GET entrant\_name  
            ADD entrant\_name to entrant\_list  
            PRINT receipt  
        ENDIF  
    ENDWHILE  
END

Top-down design – or functional programming – involves identifying the tasks and subtasks in a program. In this process, each program is broken down into modules, which are separate elements of functionality.

Each of these modules, or procedures, should be used to perform a single, specific task. And all the modules should work together to achieve the desired program output.

Procedures can be regarded as programs within programs. They are also known as functions or subroutines. They can be part of a main program file or part of a separate file.

Functions and subroutines normally work in exactly the same way, with a few exceptions. In some programming languages – like Visual Basic – functions need to return a value but subroutines don't.

The programmer decides where it's logical to create procedures in software. However, it's very important that the overall program structure, as set out in the planning phase of the software development process, is maintained.

When creating a procedure or function, you begin by deciding what task it should perform. You also need to decide whether the procedure must return a value to the main program.

Then you can give the procedure or function a meaningful name that reflects the task it performs. It's considered good form to begin with a verb followed by an object.  
  
For example, a procedure that prints labels could be named "Print\_labels." Procedure names are followed by parentheses, so the full procedure would be Print\_labels().

Consider writing a program to calculate the total cost of your grocery bill and to print the average cost of each item.   
  
You could write all the code in one programming block, as in this example. However, as a program gets larger, it becomes more difficult to work with.

Code

Process\_grocery\_bill()  
    FOR I=1 TO no\_of\_items   
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
    Average = total/no\_of\_items  
    PRINT average   
END

A more efficient approach is to create procedures for each of the tasks that has to be performed. In this case, the first procedure accepts the cost of each item and calculates the total cost of the bill. The second procedure calculates the average cost of the items you've purchased.

Graphic

*The name of the first procedure is Get\_amounts() and the name of the second procedure is   
Calculate\_and\_print\_average().*

Code

Get\_amounts()  
    FOR I=1 TO no\_of\_items   
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
END   
  
Calculate\_and\_print\_average()  
    Average = total/no\_of\_items  
    PRINT average   
END

The main body of the program now simply includes calls to the two procedures.

Code

Process\_grocery\_bill()  
    Get\_amounts()  
    Calculate\_and\_print\_average()  
END

The key benefits of creating procedures are that it enables you to

**keep a main program small and nimble**

If you use procedures to perform the main functions of a program, the main body of the program will usually consist largely of function calls. It will be small and so run efficiently.

**create a modular program**

Each procedure is responsible for a specific task. It contains only the statements that allow it to perform that task.  
  
This makes debugging and maintenance of a program easier because it's easier to isolate a problem in a single module than in a large block of code.

**re-use components, and**

Procedures can be reused in other programs or applications. You can store procedures in a repository and retrieve them as they're needed.

**make a program easier to understand**

Using procedures makes a program much easier to read and understand, because the program can then consist mostly of calls to these procedures – which should have meaningful names that describe the tasks they perform.  
  
It's also much easier to locate a procedure for a program than to look for particular lines of code within all the source code.  
  
Because the code for each procedure is separate from other procedures, it's also easier to maintain.

When you use procedures in your programming, you should aim to ensure that the different procedures are as looselycoupledas possible.

*Coupling* refers to the dependencies among the procedures in a program.   
  
With tight coupling, many procedures are dependent on others to function well. A loose coupling indicates that each procedure can perform well on its own, without relying heavily on other procedures.

When procedures are very tightly coupled, a failure in one procedure could cause an entire program to fail. When procedures are loosely coupled, this is unlikely to occur.

Question

What are some benefits of using procedures in programming?

**Options:**

1. It lets you create modular programs that are easier to maintain
2. It keeps a main program small and nimble
3. It makes a program easier to understand
4. It enables you to create reusable components
5. It ensures that all code is included in the main body of a program
6. It ensures that values are always returned to a main program

Answer

***Option 1:****Correct. Procedures break the program up into smaller sections, or modules. Each task in a program can then be easily isolated and fixed if there are errors.*

***Option 2:****Correct. The main program becomes simpler to maintain because it doesn't contain large chunks of code. Instead it will consist mainly of calls to procedures.*

***Option 3:****Correct. If you give procedures meaningful names, anyone who looks at the main program will have a clear picture of what the program does.*

***Option 4:****Correct. You can store procedures you've created for use in other programs.*

***Option 5:****Incorrect. The main goal of using procedures in programming is to create a simpler main program, which will generally include only calls to procedures – rather than large chunks of code.*

***Option 6:****Incorrect. Procedures don't always need to return values to a main program.*

**Correct answer(s):**

1. It lets you create modular programs that are easier to maintain  
2. It keeps a main program small and nimble  
3. It makes a program easier to understand  
4. It enables you to create reusable components

**2. Called procedures**

A program executes each procedure as it's required.   
  
A procedure can be called multiple times from a single program. It can also be called from multiple locations in a program, and even from within other procedures.

Code

Enter\_competitors()  
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF valid\_age = true THEN  
            Add\_entrant()  
        ENDIF   
    ENDWHILE  
END

The flow of execution, or controlling logic, in a program is called the mainline, or main program loop.

Code

Enter\_competitors()  
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF valid\_age = true THEN  
            Add\_entrant()  
        ENDIF   
    ENDWHILE  
END

To call a procedure, the programmer uses the procedure name as a command. In this example, the highlighted lines of code will cause the specified procedures to be executed.

Graphic

*The first highlighted line is Get\_amounts() and the second line is Calculate\_and\_print\_average().*

Code

Process\_grocery\_bill()  
    Get\_amounts()  
    Calculate\_and\_print\_average()  
END Get\_amounts()

The logic used depends on the type of programming, which can be either

**procedural programming, or**

In procedural programming, programs have a mainline procedure that ties other procedures together and provides master control over them.  
  
Applications are structured in terms of the actions the program must carry out and the procedures used to implement those actions.

**object-oriented programming**

In object-oriented programming, or OOP, the mainline is the starting point for a program. Examples of OOP languages are Java and C++.  
  
OOP applications comprise numerous interacting objects, each with their own data and functions. The data in an object is encapsulated.  
  
In Java, the main method is always executed first. Similarly, the main function is always executed first in C++.

Consider writing a program that adds users' names to a list of competition entrants and prints a receipt for each entrant.  
  
The mainline of the program is Enter\_competitors().

Code

Enter\_competitors()  
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF valid\_age = true THEN  
            Add\_entrant()  
        ENDIF   
    ENDWHILE  
END

When the program runs, it calls three procedures.

Graphic

*The subprocedures listed are Calculate\_age ( age ), Validate\_age ( age, valid\_age ), and Add\_entrant ().*

Code

Enter\_competitors()  
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF valid\_age = true THEN  
            Add\_entrant()  
        ENDIF   
    ENDWHILE  
END

The Calculate\_age() procedure is called first. So control of the program passes to this procedure. That means that the main program halts at that point and jumps to the procedure to execute its instructions.

Code

Calculate\_age (entrant\_name, date\_of\_birth)  
    GET current\_date  
    age = current\_date - date\_of\_birth  
END

When the Calculate\_age() procedure is completed, control is transferred back to the main program. The next program instruction is then executed, In this case, it's a call to the Validate\_age() procedure.

Code

Validate\_age(entrant\_name.age)  
    SET valid\_age to true  
    IF age <18 or age >50 THEN  
        PRINT "You must be between 18 and 50 to enter"  
        valid\_age = false  
    ENDIF  
END

After this procedure has executed, control returns to the mainline. A condition is tested and, if it evaluates to true, the Add\_entrant() procedure is called.

Code

Enter\_competitors()   
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF valid\_age = true THEN  
            Add\_entrant()  
        ENDIF   
    ENDWHILE  
END

The Add\_entrant() procedure completes the entry process and prints the receipt for the competitor.

Code

Add\_entrant (entrant\_name, date\_of\_birth)  
    GET entrant\_name  
    ADD entrant\_name to entrant\_list  
    PRINT receipt  
END

Question

You're writing a program that checks the availability of theater tickets requested by clients. If the tickets are available, you want the program to calculate their cost and print the results.   
  
Which are the called procedures in this program?

**Code**  
Process\_ticket\_request()   
    SET more\_records to yes   
    DOWHILE more\_records   
        get\_details(no\_of\_tickets, section)  
        check\_availability(no\_of\_tickets, section,   
            available)  
        IF available = true THEN   
            calculate\_cost(no\_of\_tickets, section,   
                price)  
            print\_ticket(no\_of\_tickets, section, price)  
        ENDIF   
   ENDDO

**Options:**

1. get\_details
2. check\_availability
3. calculate\_cost
4. print\_ticket
5. Process\_ticket\_request
6. more\_records

Answer

***Option 1:****Correct. get\_details is a called procedure for checking how many tickets a person requires.*

***Option 2:****Correct. check\_availability is a called procedure for checking if tickets are still available.*

***Option 3:****Correct. calculate\_cost is a called procedure for determining the cost of the tickets.*

***Option 4:****Correct. print\_ticket is a called procedure for printing the tickets that have been booked.*

***Option 5:****Incorrect. This is the mainline of the program, rather than a called procedure.*

***Option 6:****Incorrect. more\_records is a variable rather than a procedure.*

**Correct answer(s):**

1. get\_details  
2. check\_availability  
3. calculate\_cost   
4. print\_ticket

**3. Local and global variables**

Constants in a program are values that are specified and remain the same for the entire time that the program is running.

Variables hold values that can change while the program is running. This can be due to user input or results generated during processing.

Each variable used in a program has a given scope, which controls when and where the variable can be used.

There are two types of variables:

Code

Process\_grocery\_bill()  
    PROMPT FOR store\_name   
    GET store\_name    
    Get\_amounts()  
END  
  
Get\_amounts()  
    FOR I=1 TO no\_of\_items   
        PRINT store\_name  
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
END

**global, and**

Global variables are available to procedures throughout a program, and so have a global scope. They can be used more than once, and they can be created at any point in a main program. In this case, store\_name is a global variable.

**local**

Local variables are used only within a procedure and have local scope only. They aren't visible to or usable by procedures in the rest of the program.  
  
If you try to access a local variable outside of its specific function, you'll get an error. In this example, total is a local variable. It can be used only within the Get\_amounts()procedure.

A disadvantage of global variables is the potential problem of *side effects*. Side effects occur when one procedure alters a global variable or makes other changes to its environment. If the altered global variable is used by another procedure, the result returned by that procedure will change. This is undesirable when you want it always to return the same result.

Code

Process\_grocery\_bill()  
    PROMPT FOR store\_name   
    GET store\_name    
    Get\_amounts()  
END  
  
Get\_amounts()  
    FOR I=1 TO no\_of\_items   
        PRINT store\_name  
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
END

Side effects can cause confusion because they make it hard to identify which procedures are changing variables. This can make maintaining and debugging code very difficult.

Code

Process\_grocery\_bill()  
    PROMPT FOR store\_name   
    GET store\_name    
    Get\_amounts()  
END  
  
Get\_amounts()  
    FOR I=1 TO no\_of\_items   
        PRINT store\_name  
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
END

One way of dealing with side effects is to use local variables whenever possible in procedures.

Code

Process\_grocery\_bill()  
    PROMPT FOR store\_name   
    GET store\_name    
    Get\_amounts()  
END  
  
Get\_amounts()  
    FOR I=1 TO no\_of\_items   
        PRINT store\_name  
        PROMPT FOR item\_cost   
        GET item\_cost   
        total = total + item\_cost   
    END FOR   
END

So the advantage of using local variables is that you won't have the problem of incorrect values being used in different sections of a program.  
  
Consider this example. The loop inside the function is supposed to loop ten times using the variable i. In this instance, i is a local variable. When the loop executes, it will start at zero and end when iis equal to 9.

Code

display()  
    i=0  
    WHILE (i < 10)  
        println ("i = ", i)  
        i=i+1  
    ENDWHILE  
END

Here the code has been modified slightly – i is now a global variable, which has a value of 5. When i is referenced in the procedure, it will have a value of 5 rather than zero.

Code

Program\_main()  
    i=5  
    display(i)  
END  
  
    display(i)  
        WHILE (i < 10)  
            println ("i = ", i)  
            i=i+1  
        ENDWHILE  
    END

When the loop executes, the value of i will change according to the processing of the loop and may cause the rest of the program to use a new value for i.

Once data has been processed in a procedure, it may need to be passed to another part of the program.

A parameter is any variable or constant that is passed between parts of a program. In many programming languages, including C and Java, parameter names are placed in parentheses after a procedure name. The set of parameters for a procedure is referred to as a parameter list.  
  
In this example, the Java println method prints a string that is passed to it as a parameter.

Code

println(String x)

It's very important that the parameters in a parameter list match in both a main program that calls a procedure and in the procedure itself.   
  
The parameters in each case must be of the same type. The number of parameters has to be the same, and they have to be in the same order – based on the order in which they're needed for processing.

Code

Process\_ticket\_request()  
    SET more\_records to yes  
        DOWHILE more\_records  
            get\_details(no\_of\_tickets, section)  
            check\_availability(no\_of\_tickets, section, available)

In this code sample, the procedure is being called with three parameters, no\_of\_tickets, section, and available.

Code

Process\_ticket\_request()  
    SET more\_records to yes  
        DOWHILE more\_records  
            get\_details(no\_of\_tickets, section)  
            check\_availability(no\_of\_tickets, section, available)

The actual procedure uses the same three parameters in the same order.

Code

Check\_availability(no\_of\_tickets, section, available)  
    IF no\_tickets <= section\_seats\_left THEN  
        available = true  
        section\_seats\_left = section\_seats\_left -  
        no\_of\_tickets  
    ELSE  
        available = false  
    ENDIF  
END

Parameters don't process and change data. Instead they provide a way to control procedures. The procedures themselves access the specified data and modify it according to the instructions it contains.

Code

Process\_ticket\_request()  
    SET more\_records to yes  
    DOWHILE more\_records  
        get\_details(no\_of\_tickets, section)  
        check\_availability(no\_of\_tickets, section, available)  
        IF available = true THEN  
            calculate\_cost(no\_of\_tickets, section, price)  
            print\_ticket(no\_of\_tickets, section, price)  
        ENDIF  
    ENDDO  
END

Passing parameters to a procedure ensures that the same procedure can be used many times with different values. The values of the parameters passed to the procedure can simply be changed. This is a better method than creating a new procedure each time it's necessary to process a new value.

Graphic

*The parameters that will be passed to the get\_details() procedure are no\_of\_tickets and section.*

Code

Process\_ticket\_request()  
    SET more\_records to yes  
    DOWHILE more\_records  
        get\_details(no\_of\_tickets, section)  
        check\_availability(no\_of\_tickets, section, available)  
        IF available = true THEN  
            calculate\_cost(no\_of\_tickets, section, price)  
            print\_ticket(no\_of\_tickets, section, price)  
        ENDIF  
    ENDDO  
END

Parameters are passed from a calling procedure to a called procedure, and returned from a called procedure to a calling procedure. In this example, Process\_ticket\_request() is the calling procedure and get\_details() is the called procedure.

Code

Process\_ticket\_request()  
    SET more\_records to yes  
    DOWHILE more\_records  
        get\_details(no\_of\_tickets, section)  
        check\_availability(no\_of\_tickets, section, available)  
        IF available = true THEN  
            calculate\_cost(no\_of\_tickets, section, price)  
            print\_ticket(no\_of\_tickets, section, price)  
        ENDIF  
    ENDDO  
END

Question

Match each variable type to its characteristics.  
  
Each type may match to more than one characteristic.

**Options:**

1. Local
2. Global

**Targets:**

1. Used within a procedure
2. Not visible to or usable by other procedures
3. Available throughout a program
4. Accessible to a variety of procedures
5. Occur in a number of places in a program

Answer

*A local variable is used only within the procedure in which it was created. You need to use global variables to allow access from any procedure.*

*A local variable can't be seen or accessed from any procedure except the procedure in which you create it.*

*A global variable is created in the main program and can be used at any point.*

*A global variable can be re-used by different procedures. This makes it unnecessary to create the variable each time it's needed.*

*Global variables can be declared at any point in a main program and referenced in the main program or in any procedure.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option A

Target 3 = Option B

Target 4 = Option B

Target 5 = Option B

**4. Summary**

Top-down design divides a program into major tasks and subtasks. In this process, each program is broken down into modules, or separate, workable sections. Modules are further divided into procedures, each of which addresses a separate task.  
  
In many programming languages, the flow of execution of a program is controlled by a procedure called the mainline, or main loop. The mainline controls execution by calling procedures as they're required. Once a called procedure has executed, control passes back to the mainline.  
  
Variables used in a program have a given scope – either local or global. Side effects occur when a procedure alters a global variable or other parts of its environment. Parameters are variables or constants passed between parts of a program. Passing parameters to a procedure ensures that the same procedure can be used many times with different values.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

Using Arrays and Procedures

Learning Objectives

*After completing this topic, you should be able to*

* *use arrays in programs*
* *use procedures in programs*

**1. Exercise overview**

In this exercise, you're required to recognize how to use arrays and procedures in programming.

This involves the following tasks:

* using arrays in programs, and
* using procedures in programs

**2. Using arrays in programs**

Question

You're using LibertyBASIC and want to create an array that contains 21 elements. The data you want to store is test scores, so you have to give the array the Integer data type. You decide to name the array "TestScores."  
  
Identify the correct array declaration.

**Options:**

1. DIM TestScores$(20)
2. DIM TestScores(20)
3. DIM TestScores(21)
4. DIM TestScores(20) As Integer

Answer

***Option 1:****Incorrect. The dollar sign specifies the data type as String. The correct data type is Integer.*

***Option 2:****Correct. In LibertyBASIC, you create an array by specifying the DIM keyword, the array name, and the array size in parentheses. To specify the size, you identify the required upper bound – which is 20 in this case because LibertyBASIC uses zero-based arrays.*

***Option 3:****Incorrect. LibertyBASIC is a zero-based array so the index number of the twenty-first element should be identified as 20.*

***Option 4:****Incorrect. You use a declaration with the DIM keyword to specify the data type of the array in REALbasic rather than in LibertyBASIC .*

**Correct answer(s):**

2. DIM TestScores(20)

Question

You're using C++ and want to create an array that contains 21 elements. The data you want to store is the names of test subjects, so the array must be of the String data type. You decide to name the array "SubjectNames."  
  
Identify the correct array declaration.

**Options:**

1. String SubjectNames[21]
2. SubjectNames$[20]
3. String SubjectNames[20]
4. String SubjectNames(21)

Answer

***Option 1:****Correct. To create an array in C++, you specify the data type, then the array name, and finally the size in square brackets. To indicate the size, you supply the number of elements.*

***Option 2:****Incorrect. In C++, you specify the data type by providing the name of the data type. You use a dollar sign to specify the String data type in LibertyBASIC. Also, array size in C++ isn't indicated by the highest index number but by the number of elements.*

***Option 3:****Incorrect. The array size in C++ isn't indicated by the highest index number but by the number of elements.*

***Option 4:****Incorrect. In C++, the size of the array is indicated in square brackets rather than parentheses.*

**Correct answer(s):**

1. String SubjectNames[21]

Question

You're using REALbasic and want to create a two-dimensional array that contains eight elements, in a grid with four rows and two columns. The required data type is String, and you've decided to name the array "Doctors."  
  
Identify the correct array declaration.

**Options:**

1. Dim Doctors(1,3) as string
2. Dim Doctors(3,1) as string
3. Dim Doctors(2,4) as string
4. Dim Doctors[1][3] as string

Answer

***Option 1:****Incorrect. You use the DIM keyword with a declaration at the end to specify the data type. You specify the array name and provide two numbers for the size. The first is the horizontal upper bound, for rows, and the second is the vertical upper bound, for columns.*

***Option 2:****Correct. When you specify the size of a two-dimensional array, the first number indicates the horizontal upper bound – for rows – and the second number indicates the vertical upper bound, for columns.*

***Option 3:****Incorrect. REALbasic uses zero-based arrays so you provide index numbers to specify the array size.*

***Option 4:****Incorrect. You'd provide the size of each dimension separately in square brackets in C++, rather than in REALbasic.*

**Correct answer(s):**

2. Dim Doctors(3,1) as string

**3. Using procedures in programs**

Question

What happens when a program calls a procedure?

**Options:**

1. Control passes to the called procedure, which executes and then returns control to the calling procedure
2. Control passes to the called procedure, which loops repeatedly until the main program signals that it should halt
3. Control passes to the called procedure after all the calling procedure's instructions are executed

Answer

***Option 1:****Correct. When a procedure is called, the calling procedure transfers control to the called procedure. The called procedure executes and then transfers control back to the calling procedure. The called procedure may or may not return a value to the calling procedure.*

***Option 2:****Incorrect. When a procedure is called, the calling procedure passes control to the called procedure, which completes its execution. Then control is passed back to the calling procedure.*

***Option 3:****Incorrect. When a procedure is called, the calling procedure immediately jumps to the called procedure. Control is passed back to the calling procedure once the called procedure completes its execution.*

**Correct answer(s):**

1. Control passes to the called procedure, which executes and then returns control to the calling procedure

Question

This is an example of the main body of a program.  
Which are the procedures being called in this example?

**Code**  
Enter\_competitors()   
    WHILE more records  
        Calculate\_age(age)  
        Validate\_age(age, valid\_age)  
        IF Valid\_age = true THEN  
            Add\_entrant()  
            Print\_tags()    
            PRINT "Age is:", Valid\_age  
        ENDIF   
    ENDWHILE  
END

**Options:**

1. Enter\_competitors
2. Validate\_age
3. Calculate\_age
4. Add\_entrant
5. Print\_tags
6. Valid\_age

Answer

***Option 1:****Incorrect. Enter\_competitors() is the mainline of the program, rather than a called procedure.*

***Option 2:****Correct. Validate\_age() is a called procedure.*

***Option 3:****Correct. Calculate\_age() is a called procedure.*

***Option 4:****Correct. Add\_entrant() is a called procedure.*

***Option 5:****Correct. Print\_tags() is a called procedure.*

***Option 6:****Incorrect. Valid\_age is a variable name, rather than a called procedure.*

**Correct answer(s):**

2. Validate\_age  
3. Calculate\_age  
4. Add\_entrant  
5. Print\_tags

Question

In the main body of the program, you need to insert a procedure call to the Print\_ticket()procedure.  
  
Which procedure call do you use?

*The procedure to be called is:  
  
Print\_ticket (no\_of\_tickets, section , price)  
 PRINT "This is your ticket to enter the theater"  
 PRINT no\_of\_seats, section , price  
END*

**Code**  
Process\_ticket\_request()  
    SET more\_records to yes  
    DOWHILE more\_records  
        Get\_details(no\_of\_tickets, section)  
        Check\_availability(no\_of\_tickets, section,   
            available)  
        IF available = true THEN  
            Calculate\_cost(no\_of\_tickets, section, price)  
            INSERT THE MISSING CODE  
        ENDIF   
    ENDDO   
END  
  
Print\_ticket(no\_of\_tickets, section , price)  
    PRINT "This is your ticket to enter the theater"  
    PRINT no\_of\_seats, section , price  
END

**Options:**

1. Print\_ticket(no\_of\_tickets, section, price)
2. Print\_ticket( )
3. Print\_ticket(section, price, no\_of\_tickets)
4. Print\_ticket(no\_of\_tickets, section)

Answer

***Option 1:****Correct. You need to specify the parameters being used in the correct sequence when you call a procedure.*

***Option 2:****Incorrect. You need to specify the parameters within the parentheses of the procedure call.*

***Option 3:****Incorrect. You need to specify the parameters in your procedure call in the same order as in the procedure itself.*

***Option 4:****Incorrect. You need to specify the same number of parameters in your procedure call as the procedure is expecting.*

**Correct answer(s):**

1. Print\_ticket(no\_of\_tickets, section, price)

Question

You want to access a value at multiple points in your program. Which program element should you use?

**Options:**

1. Local variable
2. Global variable
3. Procedure

Answer

***Option 1:****Incorrect. A local variable can be accessed only in the procedure in which it's created.*

***Option 2:****Correct. A global variable can be accessed at any point in a program. It's created in the main body of the program so that all procedures can access it.*

***Option 3:****Incorrect. A procedure is used to perform a specific task. It's not used to pass values.*

**Correct answer(s):**

2. Global variable

You have answered questions about using arrays and procedures in programming.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_spff_a02_it_enus/output/html/course_transcript.html#top)

© 2016 Skillsoft Ireland Limited