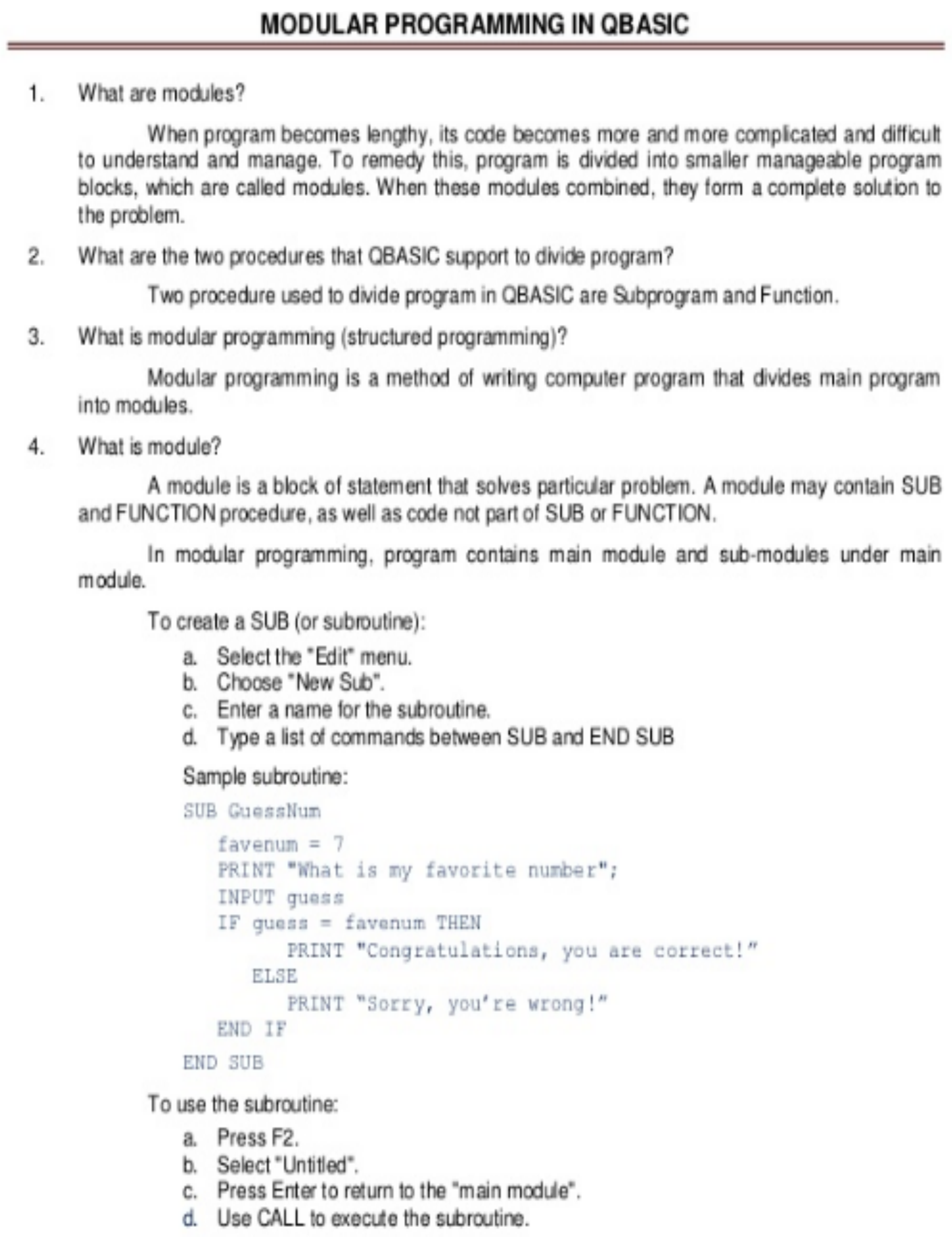
Unit 1 - Activity 7 - Modular Programming Reading

Modular programming is the process of subdividing a computer program into separate sub-programs.

A module is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.



MODULAR PROGRAMMING IN QBASIC

1. What are modules?

   When program becomes lengthy, its code becomes more and more complicated and difficult to understand and manage. To remedy this, program is divided into smaller manageable program blocks, which are called modules. When these modules combined, they form a complete solution to the problem.

2. What are the two procedures that QBASIC support to divide program?

   Two procedure used to divide program in QBASIC are Subprogram and Function.

3. What is modular programming (structured programming)?

   Modular programming is a method of writing computer program that divides main program into modules.

4. What is module?

   A module is a block of statement that solves particular problem. A module may contain SUB and FUNCTION procedure, as well as code not part of SUB or FUNCTION.

   In modular programming, program contains main module and sub-modules under main module.

   To create a SUB (or subroutine):

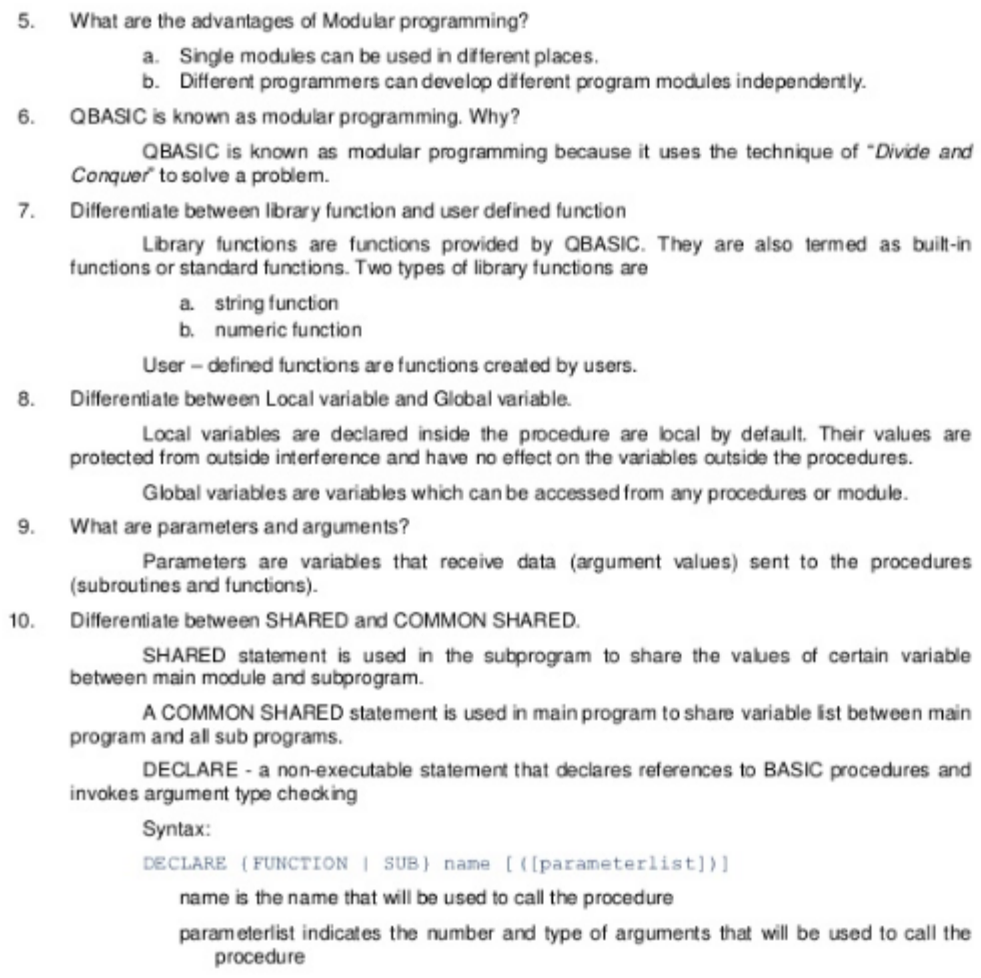
   a. Select the "Edit" menu.
   b. Choose "New Sub".
   c. Enter a name for the subroutine.
   d. Type a list of commands between SUB and END SUB

   Sample subroutine:

```
SUB GuessNum
   favenum = 7
   PRINT "What is my favorite number";
   INPUT guess
   IF guess = favenum THEN
         PRINT "Congratulations, you are correct!"
      ELSE
         PRINT "Sorry, you're wrong!"
   END IF
END SUB
```

   To use the subroutine:

   a. Press F2.
   b. Select "Untitled".
   c. Press Enter to return to the "main module".
   d. Use CALL to execute the subroutine.



5. What are the advantages of Modular programming?
   a. Single modules can be used in different places.
   b. Different programmers can develop different program modules independently.
6. QBASIC is known as modular programming. Why?

   QBASIC is known as modular programming because it uses the technique of "*Divide and Conquer*" to solve a problem.
7. Differentiate between library function and user defined function

   Library functions are functions provided by QBASIC. They are also termed as built-in functions or standard functions. Two types of library functions are
   a. string function
   b. numeric function

   User – defined functions are functions created by users.
8. Differentiate between Local variable and Global variable.

   Local variables are declared inside the procedure are local by default. Their values are protected from outside interference and have no effect on the variables outside the procedures.

   Global variables are variables which can be accessed from any procedures or module.
9. What are parameters and arguments?

   Parameters are variables that receive data (argument values) sent to the procedures (subroutines and functions).
10. Differentiate between SHARED and COMMON SHARED.

   SHARED statement is used in the subprogram to share the values of certain variable between main module and subprogram.

   A COMMON SHARED statement is used in main program to share variable list between main program and all sub programs.

   DECLARE - a non-executable statement that declares references to BASIC procedures and invokes argument type checking

   Syntax:

```
DECLARE {FUNCTION | SUB} name [([parameterlist])]
```

   name is the name that will be used to call the procedure

   parameterlist indicates the number and type of arguments that will be used to call the procedure

Linear programming:

When a program becomes complex and unmanageable, this type of programming structure is called linear programming. **Linear programming** (**LP**) (also called **linear optimization**) is a method to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. Linear programming is a special case of mathematical programming (mathematical optimization).

Encapsulation

In programming, the process of combining elements to create a new entity. For example, a procedure is a type of encapsulation because it combines a series of computer instructions. Likewise, a complex data type, such as a record or class, relies on encapsulation. Object-oriented programming languages rely heavily on encapsulation to create high-level objects. Encapsulation is closely related to abstraction and information hiding.

Polymorphism

In object-oriented programming, *polymorphism* refers to a programming language's ability to process objects differently depending on their data type or class. More specifically, it is the ability to redefine *methods* for *derived classes.* For example, given a base class *shape,* polymorphism enables the programmer to define different *area* methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the *area* method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language (OOPL).

Give an example of method of overloading or method of overriding by using modular design concept that support reusable code.

Following concepts demonstrate different types of polymorphism in java.
1) **Method Overloading**
2) **Method Overriding**

Method Definition:
A method is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the method's name.

1) **Method Overloading:**
   In Java, it is possible to define two or more methods of same name in a class, provided that there argument list or parameters are different. This concept is known as Method Overloading. One of the most **popular examples of method overloading** is System.out.println() method which is overloaded to accept all kinds of data types in Java.

Java Method Overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

2) Method Overriding

Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

User-defined classes

A *user-defined class* is like a structure or a record type in other languages. When you define a class, you specify its name, its direct super classes, and its *slots*. A slot has a name and a type. Normally, each instance stores its own value for the slot. A class inherits the slots defined by its superclasses, and it can define more slots if it needs them.

The <time-of-day> class

We start by defining a class to represent the concept of a time of day, such as 21:30. The definition of the <time-of-day> class is as follows:

```
// A specific time of day from 00:00 (midnight) to below 24:00 (tomorrow)
define class <time-of-day> (<object>) // 1
  slot total-seconds :: <integer>; // 2
```

```
end class <time-of-day>; // 3
```

The top line is a *comment*. The // characters begin a comment, which continues to the end of the line. We also provide comments that number the lines of code after the first comment. The line numbers are useful only for discussing the code examples in the book, and would not be used in source files. You can also have multi-line comments that start with /* and end with */.

On line 1, the words define class start the class definition. The name of the class is <time-of-day>. The list following the name of the class is a list of the direct superclasses of this class. The <time-of-day> class has one direct superclass, which is the class <object>. Each user-defined class must have at least one direct superclass. If no other class is appropriate, the class must have <object> as its superclass.

Line 2 contains the only slot definition of this class. This class has one slot, named total-seconds. The slot's type constraint is <integer>. The double colon, ::, specifies the type constraint of a slot, just as it specifies the type constraint of a module variable or of a method's parameter.

Line 3 is the end of the class definition. The text after the word end and before the semicolon is an optional part of the definition; it documents which definition is ending. Any text appearing after the end must match the definition ending, such as end class <time-of-day>, or end class. You do not need to put any text after the end — however, such text is useful for long or complex definitions, where it can be difficult to see which language construct is ending.