BHASVIC Pseudocode Guide for Computer Science/ Computing

In the final exams you MUST write all of the algorithms and any code-based answers in pseudocode and not a specific programming language.

Pseudocode literally means “a type of notation resembling a simplified programming language”.

OCR state that “Learners are not expected to memorise the syntax of any specific pseudocode and when asked may provide answers in any style of pseudocode they choose providing its meaning could be reasonably inferred by a competent programmer”.

**** We expect that all of your planning in class and homework submissions involving code will be written in pseudocode NOT in C, Python, Java, Ruby or any other particular language. ****

Rules for writing pseudo-code

Try to use this approach to make sure that your code is easy to follow:

- Each step of your algorithm should be as simple and broken down as possible
- Make the key commands UPPERCASE
- Indent anything inside selection/ iteration blocks so it is clear what belongs to that section of code

Variables

Assignment is shown using the single = operator e.g.

```
x=3
name=“Ben”
```

Scope - variables declared inside a function or procedure are local to that subroutine.
Variables in the main program can be shown as global by adding the keyword global e.g.

```
global userid = 123
```

Declaring variables must happen before they are used, usually at the start of the subroutine, where they should also be initialised to a start value e.g.

```
int age = 0
string name = “Ruth”
```

Casting - variables can be typecast (Changed from one type to be used as another) using the int, str and float functions e.g.

`str(3)`	returns “3”
`int(“3”)`	returns 3
`float(“3.14”)`	returns 3.14

Output of data

To display data on screen you can use the PRINT or OUTPUT keywords e.g.

```
PRINT (string)

OUTPUT (“hello”)

PRINT (“The result is:” + sum)     //this concatenates the string with the variable as one output
```

Getting Input from user

An input should always be assigned to a variable and you should put a text prompt within the brackets e.g.

```
Name = INPUT(“Please enter your name”)
```

Logical operators

Used to connect simple conditions as part of a larger condition and produce a Boolean true/ false result e.g. AND OR NOT

```
WHILE ( x<=5 AND flag==false)

IF (passcode != 0110 OR tries > 3)
```

Relational/ comparison operators

Used to compare two items of data to each other and return a Boolean true or false value.

==	Equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Arithmetic operators

+	Addition e.g. x=6+5 gives 11
-	Subtraction e.g. x=6-5 gives 1
*	Multiplication e.g. x=12*2 gives 24
/	Division e.g. x=12/2 gives 6
MOD	Modulus e.g. 12MOD5 gives 2
DIV	Quotient e.g. 17DIV5 gives 3
^	Exponentiation e.g. 3^4 gives 81

MOD and DIV are really key functions which you will use quite a lot e.g. if you need to find out if something is divisible by another number you would use the MOD function.

Selection

Used to choose change the flow of a program by selecting different paths through the code with either `if/else` or `switch/case` statements.

IF/ ELSE

```
IF ( entry == ”a” ) THEN
      PRINT (“You selected A”)
ELSE IF ( entry == ”b”) THEN
      PRINT (“You selected B”)
ELSE
      PRINT (“Unrecognised selection”)
END IF
```

SWITCH/ CASE

```
SWITCH (entry)
      case “A”: print(“You selected A”)
      case “B”: print(“You selected B”)
      default: print(“Unrecognised selection”)
END SWITCH
```

Iteration (loops)

Count Controlled

FOR loops repeat a set number of times and are used when the number or iterations is known at the start. They often start at 0 e.g.

```
FOR i=0 to 11
      print( i +” x 1 = ” +i )
NEXT i
```

This loop will print:

0 x 1 = 0

1 x 1 = 1

2 x 1 = 2

3 x 1 = 3 ……..etc

….11 x 1 = 11

Condition Controlled

WHILE and DO loops run while a certain condition is not met / until a certain condition is true e.g.

```
WHILE (answer != ”computer” )
      answer = INPUT(“What is the password?”)
ENDWHILE
```

DO and WHILE loops often use a counter variable to track how many times they have run e.g.

```
DO
      answer= INPUT(“What is the password?”)
      i = i+1
UNTIL (answer == ”computer” OR i>3)
```

This is also sometimes called a **REPEAT UNTIL** loop.

Working with strings

Concatenation of two or more strings/ joining an output string with variables on the same line:

```
PRINT (“You scored: “ + score + “points”)    //use + sign to join
```

To get the **length of a string :**

```
value = len(FirstName)
```

or

```
IF (FirstName.length <3) THEN…
```

To **get a substring** (a smaller part of a string of text):

```
stringname.subString(startPosition, numberOfCharacters)
```

NB The string will start with the 0th character. e.g.

```
someText = ”Computer Science”
PRINT (someText.length + someText.substring(3,3))
```

Will display: `16 put`

Strings are commonly stored as arrays of characters and you can also work through them with loops e.g.

```
someText = ”Computer Science”
FOR i = 0 to someText.length-1
      PRINT (someText[i])        //would output each letter of the string
NEXT i
```

Arrays

Arrays (like a list) will usually be 0 based e.g. the first element is stored in array[0].

They are usually declared in the same way as variables but have the array bounds (how many elements there are) in SQUARE brackets after the array name.

In some exam Qs they may declared with the keyword array e.g either of the versions in the example below may be used.

This example declares a string array, assigns values to each element of the array and then loops to print each item of data in each element to screen:

```
string names [5] / array names[5]
names[0]=”Elijah”
names[1]=”Ben”   ……etc
names[4]=”Kate”
FOR x = 0 to names.size      //this finds the number of items in array
     PRINT (names[x])
NEXT x
```

A **2 dimensional array** (like a table of data) would be declared as:

```
int board[8,8]
```

…. where the first number represents the row, and the second number the column.

Subroutines

A separate section of code to carry out a specific task.

These subroutines can be written as a function that returns a single value or as a procedure, which does not have a return statement and updates global variables instead.

All subroutines can be called with one, several or no parameter values.

Functions

Called from main program as part of an expression – because the return value has to be stored in a variable e.g.

`result = triple(7)`	Calling the function triple and passing it
`....`	the parameter value of 7.
`function triple(number)`	Number takes on the value 7.
`    return number*3`	The function passes 7*3 back to result.
`endfunction`	Result stores the return value of 21

Procedures

Called from the main program as an individual statement e.g.

```
square(131)

procedure square(num)
      PRINT ("The square of "+num = " +num*num )
endprocedure
```

By Value / By Reference

Unless stated in a question, values passed to subroutines can be assumed to be passed by value - the actual data is passed into the subroutine in the parameter and is used as a new local variable. This means the original value isn't changed.

If a parameter is passed by reference, this means that a pointer to the memory location where the actual data is stored is passed into the subroutine and if the variable is used, it is updated throughout the program (a bit like making your parameter global).

If this is relevant to the question byVal and byRef will be used e.g. in the example below x is passed by value and y is passed by reference. Whatever happens to x will only change inside the foobar procedure, whereas y updates will affect all parts of the program.

```
procedure foobar(x:byVal, y:byRef)
      … …
endprocedure
```

Reading from and writing to files

To **open a file to read from** openRead is used and readLine to return a line of text from the file.

The following program makes x the first line of sample.txt

```
myFile = openRead("data.txt")
x = myFile.readLine()
PRINT ("Data = " +x)
myFile.close()
```

`endOfFile()` is used to determine the end of a file e.g. the following program would use a loop to print out the contents of sample.txt:

```
myFile = openRead("sample.txt")
WHILE NOT myFile.endOfFile()
      print(myFile.readLine())
ENDWHILE
myFile.close()
```

To **open a file to write to** openWrite is used and writeLine to add a line of text to the file e.g. this program adds hello world to sample.txt . Writing to a file overwrites/ erases any previous contents.

```
myFile = openWrite("sample.txt")
myFile.writeLine("Hello World")
myFile.close()
```

To **open a file to add data to it** openAppend is used and appendLine to add a line of text to the file e.g. this program adds hello world to any existing data in sample.txt.

```
myFile = openAppend("sample.txt")
myFile.appendLine("Hello World")
myFile.close()
```

Comments

Comments are shown with // e.g.

```
print("Hello World") //This is a comment
```

Structured English vs Pseudocode

This isn't acceptable in an exam and you would be awarded 0 marks, even if it is logically correct.

```
Get user input
If input isn't divisible by 10
  Output error message
Else
  Return amount to main program
```

Whereas this does show, in a non-specific way exactly HOW the code should work.

```
amount=INPUT("Enter amount")
IF (amount MOD 10 != 0)
      PRINT("Error")
ELSE
      return amount
```

Object Oriented Pseudocode

Methods and Attributes:

Methods (the functions a class uses) and attributes (the variables a class has) can be assumed to be public unless otherwise stated in the exam question.

Where the access level is relevant to the question it will always be explicit in the code denoted by the keywords **public** and **private.**

```
private attempts = 3
public procedure setAttempts(number)
      attempts= number
end procedure

private function getAttempts()
      return attempts
end function
```

Calling Methods:

The methods of a class are its subroutines e.g. functions and procedures.

Any method that is used to read the data stored in an object's attributes or write data to the attributes tend to be referred to as getters/ setters.

To call a class method to get/ set data you reference the methods of the class using the dot notation e.g.

//uses the setter method above to update the data in the attempts attribute
```
player.setAttempts(5)
```

//uses the getter method above to access & display data from the attempts attribute
```
print(player.getAttempts())
```

Constructors

A constructor is the method of a class that defines its objects and sets the attributes with start values when an object of that class is instantiated.

OCR (in their wisdom) have decided they will always use the procedure name `New` for constructors e.g.

In a vet's admin system, each time a new instance of the Pet class is instantiated, the name and data of birth of the pet are passed into the constructor and the number of treatments is set to 1.

```
class Pet
      private name
      public procedure new(givenName, givenDoB)
            name=givenName
            DoB= givenDoB
            NumTreatments = 1
      end procedure
end class
```

Instances

To create a new instance of an object the following format is used

`objectName` `= new` *`className`*`(`*`parameters`*`)`

e.g. `myPet = new Pet("Fido", #03/08/17#)`

Inheritance

Inheritance is shown by the `inherits` keyword, superclass methods will be called with the keyword super. i.e. `super.methodName(`*parameters*`)` in the case of the constructor this would be `super.new()`

```
class Dog inherits Pet
      private breed
      public procedure new(givenName, givenBreed)
            super.new(givenName)
            breed=givenBreed
      end procedure
end class
```