Websites

www.youtube.com – Learn Delphi channel

http://www.delphibasics.co.uk/ – intermediate

www.stackoverflow.com – advanced

http://teachitza.com/

http://www.delphibasics.co.uk/

Data/Variable types

integer – integer – used for whole numbers eg 99

real – decimal numbers – eg 54.56

string – any form of character, sentence, keyboard input or output – surround by single ‘my name’

char – a single character only, surround by single ‘c’

Boolean – evaluates as true (-1) or false (0) only

Const – constants are declared to hold a known value – they type is auto determined.

```
Const
    cvVat = 0.14;
    cvLanguage = ‘English’
```

Naming Conventions

Variables

All variables begin with a small letter indicating type, capitalise next word/s. No spaces, no symbols.

```
 iAnswer : integer;
rAnswer : real;
sAnswer : string;
cAnswer : char;
bAnswer : Boolean;
```

Screen components

Edit box	edt
memo	mem
Spin Edit	sed
Label	lbl
Panel	pnl
Form	frm
Button	btn
Shape	shp
LabelEdit	lbledt
Groupbox	gbx
RichEdit	red
Calendar	cal
Checkbox	chk
Radiogroup	rgp
PageControl	pgc → for tabbed pages
Masked Edit	med

Procedures that perform tasks

1. **Randomize**; scrambles random function – use in form onActivate
2. **Beep**; makes a noise
3. **Sleep(500);** Delays the program 0.5 secs
4. **isLeapYear**(iYear)→ returns true or false
5.

Maths operations

To do simple mathematical operations on numbers

1. + → plus; - →minus;
2. *→ multiply; / →divide
3. DIV → integer division 10 DIV 3 = 3
4. MOD → remainder   10 MOD 3 = 1
5. := → assign answer
6. PI → returns value of PI

String functions

Variable declaration: sName: String;

Assignment : sName: =’John’;

Access individual characters using a position – sName[1] returns ‘J’ : sName[3] returns ‘h’

1. iLength := **Length**(sName); returns number of characters in the string
2. iPos := **Pos**(‘h’,sName); returns the position of a character in a string
3. **Delete**(sName,iPos,1) – delete from string sName at position iPos, 1 character – all 3 parameters can be literal or variable
4. **Insert**(sSub,SString,iPos) – insert a substring to the string at position
5. sName := **Uppercase**(sName) – converts whole string to uppercase
6. sName[1] := **Upcase**(sName[1]) = converts the letter at position 1 to uppercase
7. sName := **LowerCase**(sName) – converts whole string to lowercase
8. sFirstThree := **Copy**(sName,1,3) – copy a substring (string,from pos, how many pos)

Math functions

1. **Round** → rNum := Round(8.125);
2. **inc**(iCount); → increment by 1
3. **dec**(iCount); → decrement by 1
4. **ceil**(3.1) → returns next biggest number = 4
5. iVal := **Random**(10) – returns integer between 0 and 9
6. iVal :- **Random**(10)+1 – returns a value from 1 to 10 inclusive
7. rVal := **Random**; returns a value from 0 inclusive to 1 exclusive
8. iVal := **RandomRange**(50, 100); → random numberin a range – be sure to add the Math class to your uses clause.
9. iNum := **Ord**(‘A’); return ascii value 65
10. cLetter := **Chr**(65); returns letter ‘A’

<u>Screen field retrieval and setting</u>

All screen components have methods for retrieving their contents or setting the contents

1. <u>Edit box– single line entry</u>
   String name := edtName.Text;
   edtName.Text := 'Shireen';
   edtName.Clear; clears the contents

2. <u>Memo – usually for output</u>
   memOutput.Caption:='Names'; set heading
   memOutput.Lines.Add('Shireen'); adds a line
   To concatenate onto one line:
   memOutput.Lines.Text :=
   memOutput.Lines.Text+' '+IntToStr(iCount);
   memOut.Clear; clears everything

3. <u>Label – to name fields, can be for output</u>
   lblLanguage.Caption := 'English';
   lblName.Caption := ' '; clear the label text

4. <u>Combo Box – selecting from a list</u>
   String name := cmblist.text or
   cmbList.Items[cmbList.ItemIndex] – retrieve the selected item in the combo box.
   cmbList.ItemIndex := 0; → set to first item
   cmbList.ItemIndex := -1; → set to a blank

5. <u>Rich Edit found in Win32 palette</u>
   redOut.lines.clear; → clear all text

redOut.lines.add('This test will display'); → add text
redOut.lines.LoadFromFile('text.rtf'); → loads a file
redOut.Print('This will print');

<u>6. List Box</u>

String name := lstList.Items[lstList.ItemIndex] – retrieve the selected item in the list box.
lstList.ItemIndex := 0; → set to first item
lstList.ItemIndex := -1; → set to a blank

<u>7. Checkbox – evaluates true or false</u>

```
   if  chkCheese.checked = true then
     memOutput.lines.add('Cheese');
 chkCheese := false;  - set unticked
```

<u>8. Radiogroup – select an option from list of choices entered in the Items property.</u>

iNum := rgpPizzaSize.**ItemIndex** → returns the number of the choice in the list
rgpPizzaSize[0] := 'Small' →will set the first choice
lblSize.caption := rgpPizzaSize[0]; → sets the caption to the first item in the radio group list.
memPizza.Lines.Add(rgpSize.Items[rgpSize.ItemIndex]); → find the selected item
rgpPizzaSize.ItemIndex := 0; → set to first item
rgpPizzaSize.ItemIndex := -1; → set to a blank

9. <u>Spin Edit for values</u>
   iAge := sedAge.Value; retrieve the value
   sedAge.Value := 0; set a value

<u>Conversion functions</u>

Sometimes it is necessary to change from one data type to another – eg: keyboard input is always of type String and must be converted to be used as numbers

StrToInt → string to integer
StrToFloat → String to float
IntToStr → integer to string for output
FloatToStr → float to string for output
FloatToStrF → format the output using:

```
  lblCaption:= FloatToStrF(rValue,ffFixed,6,2);
              (Value,Format,Total Digits, Decimals)
```

ffCurrency – includes currency
ffFixed – as per specified decimals
ffExponent – in scientific notation
ffGeneral – only includes decimals if they exist
ffNumber – seperates thousands

BoolToStr → Outputs a Boolean variable as -1 for true and 0 for false

<u>InputBox input and ShowMessage output</u>

Ask user for input:

```
var
  sName : String;
begin
  sName := InputBox('Name','Enter your name and surname','Shireen');  //parameters are label, instruction, default – the default can be a blank string.
```

Conversion functions must be used if int or real

```
var
iAge : integer;
begin
  iAge := StrToInt(InputBox('Age,'Enter your age','');
```

Output a message to user:

```
 ShowMessage('Hello');
ShowMessage('The answer is '+ IntToStr(6*3);
```

<u>If Statement – decision making</u>

Use the templates: **ifb** – if with begin and end;
ifeb – if else with begins and ends

```
If iAge>=13 then
  begin
    lblOut.caption:='you are a teen';
  end;
if (iAge>=13) AND (gender='F') then
  begin
    lblOut.caption:='you are a female teen';
  end       //no semicolon before else
else
 begin
   lblOut.caption:='you are a male teen';
 end;
if (gender='M') OR (gender='F') then
  begin
    lblOut.caption:='valid gender';
  end       //no semicolon before else
else
 begin
   lblOut.caption:='invalid gender';
 end;
If NOT(gender='F') then
  begin
    lblOut.caption:='You are not a girl!';
  end;
//use sets to simplify statements instead of multiple
IF AND OR statements
if icount IN [1..10] then
begin
  Inc(iTimes);
end
if cletter IN ['A'..'Z','a'..'z'] then
begin
 showMessage('Part of the alphabet');
end;
```

<u>Escape Sequences for formatting output</u>

#9 – tab spaces in output
#13#10 – new line in output

```
Eg: memOut.Lines.Add('Name'+#9+'Age'+#13#10);
    memOut.Lines.Add(sName+#9+iAge+#13#10);
Name        Age
John        14
```

<u>Loops</u>

<u>for...loop – unconditional, fixed iteration [number of times it runs]</u>

Shortcut → for [tab] - used for basic for loop
Start at 0, run 10 times, increment by 1;

```
for iCount := 1 to 10 do
  begin
    memOutput.Lines.Add(IntToStr(iCount);
  end;
```

All the numbers can be replaced by variables;
The counter can also decrement

```
for iCount := 1 downto 10 do
  begin
    memOutput.Lines.Add(IntToStr(iCount);
  end;
```

<u>While ...loop – conditional – terminates using a sentinel – remember the ITC principle</u>

```
iNumber := 0;   //Intialise
While iNumber < 10 do  //Test
   begin
    Inc(iNumber);   //Change
  end;
```

<u>Character functions</u>

<u>Case statement</u>

used to evaluate integers or characters only
can use lists as below or ranges ['A'..'Z']

```
case iMonth of
  1,6,9,11  : iDays := 30;
  1,3,5,7,8,10,12 : iDays := 31;
  2  : begin   //can have begin..end with lots of code
        iDays := 28;
        if isLeapYear(iYear) then
          iDays := 29;
      end  //no semicolon before an else
else
   ShowMessage('Invalid month')
end;  //end of case
```

<u>Text files</u>

to load directly to a memo or richedit for display purposes:
memDisplay.Lines.LoadFromFile('Names.txt);

to process line by line

```
var
   tNames : textfile; → declare a file variable
  sLine : String;
begin
  AssignFile(tNames,'Names.txt'); → assign the file
  Try
    Reset(tNames);
  Except
     ShowMessage('File does not exist');
     Exit;
   End;
   while not EOF(tNames) do
     begin
        Readln(tNames,sLine);  //load a line to a string
//use string processing to split the line up into the
various fields required – use Pos to find the
delimeters, the copy to get that field and then
delete the used portion of the string.

     end;
    CloseFile(tNames);
```

to write to a text file

```
var
Names : Textfile; → declare a file variable
begin
```

```
  AssignFile(Names,'Names.txt'); → assign the file
  Rewrite(Names); → open the file for writing
  Writeln(Names, 'Shireen Steytler'); -→ write to file
  CloseFile(Names);
end;
```

**Rewrite overwrites any existing data in the file
**Append(Names); will add to an existing file but the file must exist!

String formatting

Functions and Procedures
The purpose is to create reusable code segments.
Declare the procedure/function signature in the private section of the class
Procedure TestAnswer; //no parameters
Procedure showPerc(rMark:real;rTotal:real); //parameters
Procedure isValid(sCell:String;var bValid:boolean; var sMessage:String); //returns more than 1 value
Function calcAve:real; //no parameters
Function isValid(sID:string):boolean//parameters

CTRL-SHIFT-C to create the code structure

Procedure with no parameters
A procedure can be called as a stand alone stmt – it will execute a function and produce output in its code.
Procedure with parameters
Can be sent data to work with – the parameters are declared in the receiving brackets <u>except</u> arrays which must be declared as a global type in the class
Procedures that return values
A procedure can return more than 1 value through reference parameters – these are declared in the brackets using the **var** stmt before the variable.
Function with no parameters
Functions return a **result** of the type specified in the signature – only 1 result can be returned. Functions must be called in an output or assignment stmt.
Functions with parameters
Functions can receive data to work with by declaring parameters in the brackets.

OOP – class
Objects are declared with attributes and methods
Attributes are private**;**
Methods can be private or public.
Constructor method creates the object – it can be default or parametised.
Accessor methods are functions and return the value of the attribute.
Mutator methods are procedures and receive a parameter to set the local attribute.
toString is a function that returns the values of the object as a formatted string.

```
TYPE
  TStudent = class(TObject)
   private
     fName          : String;
     fGender        : char;
     fQuestionnaires : integer;
     fHours         : real;
   public
     function  GetName : String; //accessor
     function  GetGender : char;
     function  GetQuestionnaires  : integer;
     function  GetHours : real;
     procedure SetHours(rHours : real); //mutator
     function calcAvg:real;
     function toString : string;
     constructor create(sName:String;
cGender:char;iQuestions:integer;rhours:real);
  end;  //parametised constructor – receives all the attributes as values from main program
```

OOP – main
in the GUI program:
The class must be included in the uses clause.
objStudent : TStudent; //Declare a global object of the class in the global variables
The object is instantiated with values from the user or text file processing
objStudent:=TStudent.create(*send the parameters here*);
All methods of the object are then called using the object
objStudent.toString
objStudent.getName
Methods that return values must be called in an output stmt or assigned to a variable

Dynamic Instantiation
Create a new label

Create a button with event

Create a text field

Getting the current date and time
sTime := TimeToStr(Time); → converts system time to a string
sDate := DateToStr(Date); → converts system date to a string
where format of system date is dd/mm/yyyy
sYear := Copy(sDate,7,4);
sMonth := copy(sDate,4,2);

sDay := Copy(sDate,1,2);

You can format the output:
lblMsg.caption:=
formatDateTime('hh:mm',time); displays 16:55
formatDateTime('d/m/yy',date); 9/12/17
formatDateTime('dd/mm/yyyy',date); 09/12/2017
formatDateTime('dd mmm yyyy',date); 09 Dec 2017
formatDateTime('dd mmmm yyyy',date); 9 December 2017

Add DateUtils to uses clause to use:
bLeapYear := isLeapYear(2009); → returns true or false for a leap year
bIsValid := isValidDate(2014,2,29); → true or false if valid date in the month and year
NumDays:=DaysInAMonth(2013,2);
lblMsg.caption:=dateToStr(Today);
lblMsg.caption:=dateToStr(Tomorrow);
lblMsg.caption:=dateToStr(Yesterday);
sMonth := LongMonthNames[12] ; =December

Isolate as an integer
iYear:= YearOf(date);
iMonth:=MonthOf(Date);
iDay:=DayOf(Date);

Manipulate a date – add or subtract days
// Set up our date just before the end of the year 2000
  myDate := EncodeDate(2000, 12, 30);
  ShowMessage('myDate = '+DateToStr(myDate));

Output → myDate = 30/12/2000

// Add 10 days to this date
  myDate := IncDay(myDate, 10);
  ShowMessage('myDate + 10 days = '+DateToStr(myDate));
Output → myDate + 10 days = 09/01/2001

// Subtract 12 days from this date
  myDate := IncDay(myDate, -12);
  ShowMessage('myDate - 12 days = '+DateToStr(myDate));

Output → myDate - 12 days = 29/12/2000

Arrays
declaring arrays:
const arrDays:array[1..7] of String = ('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'); →*array of constant values*
var arrNames : array [1..10] of String; → *array to hold 10 strings index 1 to 10*
var arrMarks:array[5..25] of integer; → *array to hold 20 integers index 5 to 25*
var arrLetters:array[A..Z] of char; → *array to hold 26 characters index A to Z*
const iMaxNum=100;
var arrLearners: array[1..iMaxNum] of String; -→ *array to hold a fixed number of strings defined by a constant value declared globally – used for text file processing or when the number of entries is unknown upfront. Make the array bigger than necessary.*

accessing the elements of an array
arrLearners[1] := 'Shireen'; →assign a name to position 1 of the array
memOut.Lines.Add(arrNames[1]);→ output position 1 of the array

fill an array using a loop:
for iCnt := 1 to 12 do
  arrMonths[iCnt]:=InputBox('Month','Enter month name',''); → *fill a 12 space array with user input. The index of the for loop (iCnt) is used as the position index of the array.*

output an array using a loop:
for iCnt := 1 to 10 do
  redOut.Lines.Add(arrNames[iCnt]); →*output the contents of arrNames using position iCnt – will loop 10 times and output arrNames positions 1 to 10.*

fill an array from a text file
var
   tNames : textfile; → *declare a file variable*
  sLine : String;
  arrNames:array[1..25]of String;
iCnt : integer; → *counter to track number of entries*
begin
  AssignFile(tNames,'Names.txt'); → *assign the file*
  Try
    Reset(tNames);
  Except
     ShowMessage('File does not exist');
     Exit;
   End;
  iCnt := 0; → *initialise your counter*
   while not EOF(tNames) do
     begin
         Inc(iCnt); → increment your counter
         Readln(tNames,sLine)→ *reads a line from the text file into the sLine variable*
         arrNames[iCnt] := sLine; → *assign the string to the array at position iCnt*

     end; //*end while*
    CloseFile(tNames);

Sorting arrays

Always use a double for loop to sort an array.
Descending **>** and Ascending **<**
<u>Remember the 6 steps</u>
For, For, If, Temp←I, I←J,J←Temp

```
var
  I, J, tempBooks: integer;
  tempName: String;
  tempGrade: char;
begin
  for I := 1 to iCount do
  begin
    for J := 1 to iCount do
    begin
      if arrNames[I] < arrNames[J] then
      begin
        tempName := arrNames[I];
        arrNames[I] := arrNames[J];
        arrNames[J] := tempName;
      end;  //endif
    end;  //endforJ
  end;  //endforI
```

Output as usual.

<u>Sorting linked arrays</u>
Always use a double for loop to sort an array.
Swop all linked arrays at the same time.

```
var
  I, J, tempBooks: integer;
  tempName: String;
  tempGrade: char;
begin
  for I := 1 to iCount do
  begin
    for J := 1 to iCount do
    begin
      if arrNames[I] < arrNames[J] then
      begin
        tempName := arrNames[I];
        arrNames[I] := arrNames[J];
        arrNames[J] := tempName;
        tempGrade := arrGrade[I];
        arrGrade[I] := arrGrade[J];
        arrGrade[J] := tempGrade;
        tempBooks := arrBooks[I];
        arrBooks[I] := arrBooks[J];
        arrBooks[J] := tempBooks;
      end;  //endif
    end;  //endforI
  end;   //endforJ
```

Searching an array
for loop and if check to find the index of whatever you are looking for. Then you can output the details, probably from linked arrays.

```
var
  sName: String;
  iPos, I: integer;
  bFound: boolean;
begin
  iPos := 0;
  bFound := false;
  sName := InputBox('Learner name', 'Enter Name',
'');
  for I := 1 to iCount do
  begin
    if arrNames[I] = sName then
    begin
      iPos := I;
      bFound := true;
    end;
  end;
  redOutput.Lines.Add('Search for learner ' +
sName);
  if Not bFound then
    ShowMessage('Not found')
  else
    redOutput.Lines.Add(arrNames[iPos] + #9 +
arrGrade[iPos] + #9 + IntTostr
      (arrBooks[iPos]));
end;
```

Deleting a record from an array
First find the record – same as searching, then overwrite with next records and decrease the count.

```
var
  sName: String;
  iPos, I: integer;
  bFound: boolean;
begin
  iPos := 0;
  bFound := false;
  sName := InputBox('Learner name', 'Enter Name',
'');
  for I := 1 to iCount do
  begin
    if arrNames[I] = sName then
    begin
      iPos := I;
      bFound := true;
    end;
  end;
  redOutput.Lines.Add('Search for learner ' +
sName);
  if Not bFound then
    ShowMessage('Not found')
  else
  begin
    for I := iPos to iCount do
    begin
//move the next record backwards
  //to overwrite the deleted record
      arrNames[I] := arrNames[I + 1];
```

```
     arrGrade[I] := arrGrade[I + 1];
     arrBooks[I] := arrBooks[I + 1];
    end;
    Dec(iCount);
//decrease the total number of records in array
//output as usual
    redOutput.Lines.Add('After delete');
    for I := 1 to iCount do
    begin
     redOutput.Lines.Add(arrNames[I] + #9 +
arrGrade[I] + #9 + IntTostr
        (arrBooks[I]));
    end;
  end;
end;
```

Removing duplicates
Sort first, loop, if the record at [I+1] NOT equals the record at [I] then move the record to a new array else skip the record by doing nothing.

Find Top 5
Sort first in descending order then output 1 to 5

__2D Arrays__

Declare a 2D array – can use numbers only as indices or number and letters or letters only
Arrays can be of type integer, string, real, char – but only 1 of the types at a time.

```
arrClasses:array[8..12,'A'..'F'] of integer;
arrClasses:array[8..12,1..6] of integer;
arrClasses:array['S'..'Z','A'..'F'] of integer;
```

To fill a 2D array: use a double for loop.

```
for r := 8 to 12 do
     begin
     for c := 'A' to 'F' do
       begin
          arrClasses[r,c]:= StrToInt(InputBox('Pupils
present in grade '+inttostr(r),'Enter pupils in class
'+c,''));
       end;
```

To access any cell you need row and column
arrClasses[r,c] - variable access normally using for loop index or inputted positions
arrClasses[8,A] = specifically access row 8 and col A

To output a 2D array: use a double for loop

```
for r := 8 to 12 do
     begin
     for c := 'A' to 'F' do
       begin
         redOut.Lines.Add('Pupils in grade'+inttostr(r)
+ ' and class ' + inttostr(c) +  arrClasses[r,c];
       end;
      end;
```

To sum the rows in a 2D array

```
for r := 8 to 12 do
     begin
     iSum := 0;
     for c := 'A' to 'F' do
       begin
         iSum := iSum + arrClasses[r,c];
       end;
      redOut.lines.Add('Sum for Row '+inttostr(r)+ ' '
+inttostr(iSum));
    end
```

To sum the columns in a 2D array

```
for c :=  'A' to 'F' do
     begin
     iSum := 0;
     for r := 8 to 12 do
       begin
         iSum := iSum + arrClasses[r,c];
       end;
      redOut.lines.Add('Sum for Col '+inttostr(c)+ ' '
+inttostr(iSum));
     end
```

To sum a left diagonal of a square 2D array

```
sout := 'Diagonal1: ';
 itotal := 0;
 for r := 1 to 4 do
   begin
    itotal := itotal + arrnum[r,r];
    sout := sout + IntToStr(arrnum[r,r])+' + ';
   end;
  Delete(sout,length(sout)-1,1);
  sout  := sout + ' = ' + IntToStr(itotal);
  redOutput.Lines.Add(sout);
```

To sum a right diagonal of a square 2D array

```
sout := 'Diagonal2: ';
 itotal := 0;
 for r := 4 downto 1 do
   begin
    itotal := itotal + arrnum[r,5-r];
    sout := sout + IntToStr(arrnum[r,5-r])+' + ';
   end;
  Delete(sout,length(sout)-1,1);
  sout  := sout + ' = ' + IntToStr(itotal);
  redOutput.Lines.Add(sout);
```

To work out average in a 2D array

To get the highest number in a 2D array

Note: The same algorithms apply for finding the lowest number, summing etc – just use a double for

loop instead of a single for loop – always row then column except if you are summing columns.
Note: A 2D array can be filled through input from the keyboard or a text file – in the fill algorithm replace the “0” with what input you are getting.