1. Delphi

1.1. Data types

1.1.1. Integers

```
var
    Int1 : Byte;     //                         0 to 255
    Int2 : ShortInt; //                      -127 to 127
    Int3 : Word;     //                         0 to 65,535
    Int4 : SmallInt; //                   -32,768 to 32,767
    Int5 : LongWord; //                         0 to 4,294,967,295
    Int6 : Cardinal; //                         0 to 4,294,967,295
    Int7 : LongInt;  //            -2,147,483,648 to 2,147,483,647
    Int8 : Integer;  //            -2,147,483,648 to 2,147,483,647
    Int9 : Int64;    // -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
```

1.1.2. Decimals

```
var
    Dec1 : Single;   //  7  significant digits, exponent   -38 to +38
    Dec2 : Currency; // 50+ significant digits, fixed 4 decimal places
    Dec3 : Double;   // 15  significant digits, exponent  -308 to +308
    Dec4 : Extended; // 19  significant digits, exponent -4932 to +4932
```

1.1.3. Texts

```
var
   Str1 : Char;        // Holds a single character, small alphabet
   Str2 : WideChar;    // Holds a single character, International alphabet
   Str3 : AnsiChar;    // Holds a single character, small alphabet
   Str4 : ShortString; // Holds a string of up to 255 Char's
   Str5 : String;      // Holds strings of Char's of any size desired
   Str6 : AnsiString;  // Holds strings of AnsiChar's any size desired
   Str7 : WideString;  // Holds strings of WideChar's of any size desired
```

1.1.4. Logical

```
var
   Log1 : Boolean;     // Can be 'True' or 'False'
```

1.1.5. Sets, enumerations and subtypes

```
type
   TSuit = (Hearts, Diamonds, Clubs, Spades);    // Defines the enumeration
 var
   suit : TSuit;                                 // An enumeration variable
```

Sets are often confused with enumerations.

An **enumeration variable** can have **only one of the enumerated values**.

A **set** can have **none**, **1**, **some**, or **all of the set values**.

```
type
   TWeek = Set of 1..7;      // Set comprising the days of the week, by number
 var
   week : TWeek;
 begin
   week := [1,2,3,4,5];      // Switch on the first 5 days of the week
 end;
```

1.1.5.1. Enumerations = `enum`

```
type
   TSuit = (Hearts, Diamonds, Clubs, Spades);   // Defines enumeration range
 var
   suit : TSuit;                                // Defines enumeration variable
 begin
```

```
  suit := Clubs;                                  // Set to one of the values
end;
```

The TSuit type definition creates a new Delphi data type that we can use as a type for any new variable in our program.

```
type
  TDay = (Mon=1, Tue, Wed, Thu, Fri, Sat, Sun);   // Enumeration values
var
  today   : TDay;
  weekend : Boolean;
begin
  today := Wed;           // Set today to be Wednesday

  if today > Fri          // Ask if it is a weekend day
  then weekend := true
  else weekend := false;
end;
```

`today` is set to `Wed` which has ordinal value = `3` `weekend` is set to `false` since `Wed` (`3`) `<=` `Fri` (`5`)

```
type
   TDay = (Mon=1, Tue, Wed, Thu, Fri, Sat, Sun);   // Enumeration values
 var
   day : TDay;           // Enumeration variable
 begin
   for day := Mon to Fri do
   begin
     // day has each of the values Mon to Fri ( 1 to 5) in 5 iterations
     // of this loop, allowing you to whatever you want.
   end;
 end;
```

> **Warning** : each of the values in an enumeration must be unique in a program. This restriction allows you to assign an enumeration value without having to qualify the type it is defined in.

1.1.5.2. SubRanges

```
type
   TSmallNum = 0..9;
 var
   smallNum : TSmallNum;
 begin
   smallNum := 5;    // Allowed
   smallNum := 10;   // Not allowed
```

```
    smallNum := -1;   // Not allowed
 end;
```

> Delphi will not compile code that has assignments outside of the given range.

Subranges of characters:

```
type
   TUpper = 'A'..'Z';
   TLower = 'a'..'z';
   TDigit = '0'..'9';
 var
   upper : TUpper;
   lower : TLower;
   digit : TDigit;
 begin
   upper := 'G';   // Allowed
   lower := 'g';   // Allowed
   digit := '7';   // Allowed

   upper := 'g';   // Not allowed
   lower := '7';   // Not allowed
   digit := 4;     // Not allowed
 end;
```

Subrange of enumerations:

```
type
    TDay = (Mon=1, Tue, Wed, Thu, Fri, Sat, Sun);   // Enumeration values
    TWeekDays = Mon..Fri;                           // Enumeration subranges
    TWeekend  = Sat..Sun;
```

1.1.5.3. Sets

```
type
   TDigits = set of '1'..'9';       // Set of numeric digit characters
 var
   digits : TDigits;                // Set variable
   myChar : char;
 begin
   // At the start, digits has all set values switched off
   // So let us switch some on. Notice how we can switch on single
   // values, and ranges, all in the one assignment:
   digits := ['2', '4'..'7'];

   // Now we can test to see what we have set on:
   for myChar := '1' to '9' do
```

```
    if myChar In digits
    then ShowMessageFmt('''%s'' is in digits',[myChar])
    else ShowMessageFmt('''%s'' is not in digits',[myChar])
end;
```

The `In` operator tests to see if a set contains a value.

The data shown is as follows:

```
'1' is not in digits
'2' is in digits
'3' is not in digits
'4' is in digits
'5' is in digits
'6' is in digits
'7' is in digits
'8' is not in digits
'9' is not in digits
```

1.1.5.3.1. Including and excluding set values

`Include` *(switch on)* or `exclude` *(switch off)* individual values without affecting other values.

```
type
   // We define a set by type - bytes have the range : 0 to 255
   TNums = set of Byte;
var
   nums : TNums;
begin
   nums := [20..50];     // Switch on a range of 31 values
   Include(nums, 12);    // Switch on an additional value : 12
   Exclude(nums, 35);    // Switch off a value : 35
end;
```

`nums` now has the following values set : `12 , 20..34 , 36..50`

1.1.5.3.2. Set operators

operator	description
+	union of two sets
*	intersection of two sets
-	difference of two sets
=	tests for identical sets

operator	description
<>	tests for non-identical sets
>=	is one set a subset of another
<=	is one set a superset of another

```
type
   TNums = set of 1..9;
 var
   nums1, nums2, nums3, nums4, nums5, nums6 : TNums;
 begin
   nums1 := [1,2,3];
   nums2 := [1,2,4];
   nums3 := [1,2,3,4,5,6,7,8,9];

   nums4 := nums1 + nums2;    // nums4 now [1,2,3,4]
   nums5 := nums1 * nums2;    // nums5 now [1,2]
   nums6 := nums1 - nums2;    // nums6 now [3]

   // Test for equality
   if nums1 = nums2
   then ShowMessage('nums1 =  nums2')
   else ShowMessage('nums1 <> nums2');

   // Test for inequality
   if nums1 <> nums3
   then ShowMessage('nums1 <> nums3')
   else ShowMessage('nums1 =  nums3');

   // Is nums1 a subset of nums3?
   if nums1 <= nums3
   then ShowMessage('nums1 is a subset of nums3')
   else ShowMessage('nums1 is not a subset of nums3');

   // Is nums1 a superset of nums3?
   if nums1 >= nums3
   then ShowMessage('nums1 is a superset of nums3')
   else ShowMessage('nums1 is not a superset of nums3');
 end;
```

Output:

```
nums1 <> nums2
nums1 <> nums3
nums1 is a subset of nums3
nums1 is not a superset of nums3
```

1.1.6. Compound data types

1.1.6.1. Arrays

```
var
   Suits : array[1..4] of String;    // A list of 4 playing card suit names

 begin
   Suits[1] := 'Hearts';    // Assigning to array index 1
   Suits[2] := 'Diamonds';  // Assigning to array index 2
   Suits[3] := 'Clubs';     // Assigning to array index 3
   Suits[4] := 'Spades';    // Assigning to array index 4
 end;
```

The array defined above has indexes 1 to 4 (1..4). The two dots indicate a range.

1.1.6.2. Records

Like a `class` or a `struct`, but not because there are other data types for object oriented things

```
type
   TCustomer Record
     firstName : string[20];
     lastName  : string[20];
     age       : byte;
   end;
```

```
var
  customer : TCustomer;            // Our customer variable
begin
  customer.firstName := 'Fred';    // Assigning to the customer record
  customer.lastName  := 'Bloggs';
  customer.age       := 55;
end;
```

Output:

`customer.firstName` is now set to `'Fred'`

`customer.lastName` is now set to `'Bloggs'`

`customer.age` is now set to `55`

1.2. Programming logic