## 1. Points to Remember

1.1. Italic font has been used for bookmarks. You can click on them.

1.2. Examples from various programming languages are given for better explanation.

1.3. **Syntax** is a statement in a program. Syntax errors in programming are similar to grammatical errors in English(which is not acceptable). Every programming language has its own rules for writing syntax.

1.4. Meaning of Words to be used in this document:

1.4.1. **Application –** program, code

1.4.2.**Execution –** running the application

1.4.3.**Delimiter -** ***Syntax Terminator***

1.4.4.**Character –** any symbol on keypad. Example- numbers, alphabets, + - * / ; ‘ . etc

1.4.5.**Language –** programming language

1.5. Any statement preceded by ‘*’ in any example program is a comment for explanation. Comments are not part of program syntax.

1.6. Most of the Programming languages are case sensitive(just like your login passwords). Means it is sensitive to uppercase and lowercase letters. Example – hello and Hello are totally different in programming language.

1.7. **Compilation** – When you complete writing your program code, then you need to generate executable file which a computer can run. To generate this file you have to use software tool **compiler** in which your code gets compiled means it is getting converted to language that computer understands. Compiler also checks for errors in coding.

## 2. Programming Concepts

To learn any programming language, you need to know following concepts (at least). These concepts are applicable to mostly all popular programming languages.

### 2.1.Syntax Terminator

2.1.1.It is used to indicate end of a ***Syntax*** . Similar to ‘.’ (full-stop) in English.

2.1.2.Different programming languages have different ***Delimiter*** .

2.1.3.Example 1: C language ***Syntax*** :

```
x = 3;      * (;) i.e. semi colon is a delimiter in this syntax.
```

2.1.4.Some programming languages don’t use any character as ***Delimiter*** . Example 2 – python ***Syntax*** :

```
x = 3       * no delimiter used.
```

### 2.2.Variable

2.2.1.It is an entity to store data.

2.2.2.Every programming language has its set of rules that defines naming convention and use of a variable.

2.2.3.See ***Example 1***. * x is a variable.

### 2.3.Data Type

2.3.1.Data can be of following types – **numbers**: integer, decimal number, complex number, **Other -** ***Character*** , ***String***.

2.3.2.Different languages have different names for their data types. Examples:

| Data type name | data |
|---|---|
| int | integers |
| float | Decimal numbers |
| char | Single character |
| String | Group of characters |

2.3.3. In some programming languages you need to declare the variable's ***Data Type*** before storing any data in it. Example 3 - C language ***Syntax*** :

```
int x = 3;    * "int" is integer data type. 'x' is a variable.
```

2.3.4. In some programming languages you don't need to declare the variable's ***Data Type*** before storing any data in it. Example 4 - Php language ***Syntax*** :

```
$x = 3;       *  '$' before 'x' indicates that it is a variable. Here you don't need to declare data type of 'x'.
```

2.3.5. In English, we use group of characters to make words and sentences. Similarly, in programming languages, group of characters makes a **String** data type**.** Strings are generally enclosed within pair of inverted comma(""). Example 5 – C++ language ***Syntax*** :

```
String var = "Hello World";          *String is data type. 'var' is variable name. String data is enclosed within "
                                     *". Thus, data is  Hello World
```

2.3.6. Note – You perform mathematical operations only on those variables which has integer/decimal data stored in them. Example 6 – C++ language program to add 1 to a variable and store in another variable.

```
int x = 3, y;          * data type of x and y is int i.e. integer
char z = '3';          * data type of z is char. it is character data, not integer data.
y = x + 1;   * add 1 to x and store in y. This is correct. Value of y becomes 4.
y = z +1;    * This syntax will give incorrect result because z is of data type 'char' i.e. character. Here value of y
will not become 4.
```

2.3.7. **How to differentiate between numerical data and character data**? Numerical data is not written between inverted comma ("    ").

## 2.4.Function

2.4.1. 'Function' is the execution body of a program code. Whatever you write inside a function, it gets executed.

2.4.2. Generally, there are 2 type of functions: **Main function(or main thread), other functions.**

2.4.3. There could be unlimited number of functions in a program, but there can be only one main function.

2.4.4. A program starts and end execution from the main function. Other functions are called from the main function and other functions.

2.4.5. You can write all your program code inside the main function, but if there is any piece of code that you need to use repeatedly for various scenarios then instead of writing the same piece again and again, you can write a separate function for it.

2.4.6. These other functions can be created by you or could be already present in software's library.

2.4.7. **Function call** – A function executes only when a call is made to it from an already executing function.

2.4.8. **Function Arguments –** It is the data passed to a function that it may need. It is optional.

2.4.9. **Function Return –** a function can return some data (it depends on function type ) after completion. Return also indicates the end of function. It is optional.

2.4.10. After the 'called' function returns, the 'calling' function continues execution from the ***Syntax*** next to function call.

2.4.11. Example 7 – C language program that have a function for adding 2 numbers.

```
void main()           *main function of type 'void' means it will not return any data after completion
{                     *beginning of function's body
   int a = 3, b = 2, result;   *3 variables declared
   result = Add(a,b);          * function call to Add. At this point, program jumps to function 'Add' and Return
                               *the data after completion of the function. The returned data is stored in 'result':
                               *a,b are the Arguments to this function.
}                     * end of function's body
```

```
int Add(int x, int y)       *body of 'Add' function. It is of type 'int', means it will Return data of type 'int' i.e.
                            *integer. 'x' & 'y' are the arguments which have values corresponding to passed
                            *Arguments
{
    int res;            * variable res of type 'int'
    res = x + y;        * Add the two numbers and store in another variable 'res'
    return res;         * Return 'res'
}
```

2.4.12. Functions are also called as '**methods**'.

## 2.5.Loop

2.5.1.It is a feature you can use when you want a piece of code to execute continuously for n number of times.

2.5.2.Every language has in built functionality to implement loop.

2.5.3.In many languages, following are the names of **loop** technique – **for, while, do while**

2.5.4.Each language has its own way of writing code for **loop**, but working algorithm is same in all languages.

2.5.5.Basically, to run a loop, you need a ***Variable*** that works as 'counter'.

2.5.6.First, at beginning of loop you give a **starting value**(a number) to the counter from where it should start counting.

2.5.7.Second, you should also specify the **last value** your counter should reach before coming out of the loop. It is the n number of times that you want that piece of code to execute.

2.5.8.Third, you should **update** your counter variable each time after execution of code(inside loop). If you don't, then counter will not reach the last value you specified and that code inside loop will execute infinitely (or till your program crashes).

2.5.9.Example 8: C language program to explain 'while' loop:

```
int count = 0;      *start value for counter variable 'count'
int n = 5;          *last value that counter should reach
while(count < n)    *'while' loop beginning. Program checks this condition(i.e. 'count' is less than 'n') each time
                    *before execution of loop's body. If this condition is true it executes the code inside loop's
                    *body, else it comes out of  the loop.
{                   * start of loop's body
      *write your code inside { …. }
      count++;      * counter value increased by one (update). Thus, after executing this code 5 times, value
                    *of 'count' will be 5. The condition mentioned at the beginning will become false. Hence,
                    *program will come out of the loop.
}                   * end of loop's body. Program goes back to beginning of loop after reaching this end.
```

2.5.10. **Reminder** – for any piece of code to execute, it should be inside ***Function*** body. Therefore, **loop** must also be written inside a ***Function***.

## 2.6.IF-ELSE Condition

2.6.1.Conditions are important in programming because many times you make a program as follows: "if this happens then do this or else do that"

2.6.2.You saw use of a condition in ***Example 8***

2.6.3.Each condition can either evaluate to 'true' or 'false'.

2.6.4.When you need to make a program for various scenarios then you should use **if-else** structure.

2.6.5.If-else structure is available in most of the popular programming languages.

2.6.6.Example 9 : C- program to add 1 to an integer if it is greater than 10, add 2 if it is less than 10.

```
void main()
{
    int num = 3;
    if(num > 10 )      * 'if' condition checks if 'num' is greater than 10
  {                    * start of 'if'
  num = num + 1;       * add 1 to 'num'
  }                    *end of 'if'
  else if(num < 10 )   *if above condition is false then check another condition using 'else if' - 'num' is less than
                       *10
  {                    *start of 'else if'
    num = num + 2;     * add 2 to 'num'
  }                    *end of 'else if'
```

## 2.7.Data Structure

2.7.1.Data Structure is a collection/group of data in a single variable. This variable is generally referred as **Object**.

2.7.2.Data inside an **object** can be accessed by name of that object. Each data is considered as **member** of this object.

2.7.3.Try to understand concept of **Object** with this analogy: There is class **A** of 100 students. Here **A** is the **Object** whereas **students** are its **members**. Each student has a unique Roll No. Thus, identity of each student becomes - Roll No 'x' from Class A. Similarly, there is another Class **B** which also has 100 students with same Roll No as in **A.** Now, if any outsider wants to contact any particular student then he must have information about the Class name and Roll No of the student. Or, if any one wants to address all students from a class then he just needs Class name. **Principal: Class A, please don't make noise.**

2.7.4.Members(***Variable***) inside an **Object** can be of any ***Data Type***. **Note** – There are different types of Data Structure in a programming language. Some structures support that all members of an object must be of same ***Data Type*** whereas some structures offer more flexibility to its members.

2.7.5.Generally, we group that data which is similar or related to each other. Suppose, you need to buy some items from shop **A** and some items from another shop **B**. So, in this case you make a **list**(group) of items to buy from Shop **A** and another list for shop **B.**

2.7.6.Different languages have different types of data structure. Names of some popular data structure: **struct, class, list, array, tuple**.

**Coding examples for any language are available in bulk on internet. See how these concepts are implemented(written or coded) in a language.**

**Remember : A programming language may have more things other than those mentioned above. This document contains knowledge of basic concepts that are implemented in most of the popular programming languages.**