

# Syntax

## 1. Semicolon:

**NO** semicolon required at the end of the line.

## 2. Variables, Keywords:

Variables ~

No data types required before variable names to be defined.

Example:

a=10	declare integer
b="string"	declare a string name inside double inverted commas
c= 5.765	declare float
D='c'	declare character

Data Types ~

bool a = t	declare a as true
bool b = f	declare b as false
bool c = !a	c is false
show f"(a)"	output is 1
show f"(c)"	output is 0

Keywords ~

int, float, double, default, main, if, elif, else, for, while, do, def, break, continue  
are all keywords

NOTE: for type casting data types are required.

Example : int(5.77) gives 5 or (int)5.77 gives 5

## 3. Operators:

1.Arithmetic operator : + Addition

- Subtraction

\* multiplication

/ division

% modulo operator (remainder after division can only be  
used with integers )

2.Assignment operator:	=	a=b	a=b
	+=	a+=b	a=a+b
	-=	a-=b	a=a-b

	<code>*</code>	<code>a*=b</code>	<code>a=a*b</code>
	<code>/</code>	<code>a/=b</code>	<code>a=a/b</code>
	<code>%</code>	<code>a%=b</code>	<code>a=a%b</code>
3.Relational operators:	<code>==</code>	is equal to	
	<code>!=</code>	is not equal to	
	<code>&gt;</code>	Greater than	
	<code>&lt;</code>	Less than	
	<code>&gt;=</code>	Greater than or equal to	
	<code>&lt;=</code>	Less than or equal to	
4.Logical operators:	<code>&amp;&amp;</code>	logical AND	
	<code>  </code>	logical OR	
	<code>!</code>	logical NOT	
5.Bitwise operator:	<code>&amp;</code>	binary AND	
	<code> </code>	binary OR	
	<code>~</code>	binary one's complement	
	<code>&lt;&lt;</code>	binary left shift	
	<code>&gt;&gt;</code>	binary right shift	
6.Other operators:	<code>sizeof</code>	returns size of the data type	<code>sizeof(int)=4</code>
	<code>?:</code>	returns value based on condition	string result = (5>0)? "Even" : "odd"
	<code>\$</code>	stores memory address of The operator	<code>\$num</code>
	<code>.</code>	access the members of Structure or class	<code>s1.marks=92</code>
	<code>-&gt;</code>	used with pointers to access Class or struct members	<code>ptr-&gt; marks =92</code>
7.Increment and	<code>++</code>	increases by 1	<code>a=10</code>
Decrement operator	<code>--</code>	decreases by 1	<code>a++ gives 11</code> <code>a-- gives 9</code>

## 4.Scanning, Printing and Comments :

### 1.Scanning ~

`a = read ""` when a is any data type.  
Here this `read""` function will simply store the value in a

`a = read "Enter the value of a"` when a is a string.  
Here `read"something written here"`, will print the string inside " " and then  
Store the value in a .

### 2.Printing ~

show "Things to be printed"  
show "\n"

```
a=10
show f"Value of a is (a)\n"    output is: Value of a is 10
```

- 3. Single line comment ~ ^Comments
- Multi line comment ~ ^^Comments
- Comments^^

## 5.Arrays, Pointers, Structures :

### 1.Arrays ~

```
@array_name[5] = {1,2,3,4,5}          whether string or integer,
@array_name[ ] = {"name1", "name2", "name3"}  declaration style remains the
                                              same.
```

array\_name[ ]      [ ] used to access array elements with the help of the index no.

### 2.Pointers ~

```
int #ptr = &(var_name)
```

A pointer named ptr that points to an integer variable. # is used to make a pointer  
Dereferencing, increment, decrement can also be done with the help of #.

### 3.Structures~

```
struct (struct_name) -
    Indentation code
```

## 6.Control structures :

### 1.If-else-elseif ~

```
if(condition)
{
    Code
}
elif(condition)
{
    Code
}
else
{
    Code
}
```

### 2.Switch Case ~

```
switch(variable_name)-
    case 1-
        Indentation code
    default-
```

Indentation code

## 7.Loops :

1.While loop ~

```
while(condition)-  
    Indentation code
```

2.Do-While loop ~

```
do-  
    Indentation code  
while(condition)
```

Executed at least once, even though condition is satisfied or not.

3.For loop ~

```
for(i=0; i<n; i++) -  
    Indentation code  
break  
continue
```

Semicolons required to separate 3 parts.

'break' and 'continue' are used inside the conditional statements which are either separate or inside any of above 3 loops

## 8.Functions :

1. Main Function ~

```
def main()-  
    Indentation code
```

Main function contains objects of structure or class, and these objects with the help of Dot operator access the members of structure or class.

2.Any other function ~

```
def function_name (parameters)-  
    Indentation code
```

## 9.Space Sensitivity :

- 1.The language is not space sensitive, except for the indentation (4 spaces) used in control structures and loops.
2. The language is case sensitive.