# HIPLANG

In our proposed language, all the Keywords are written in UPPERCASE letters and Identifiers are allowed to hold `lowercase` only.

Statement ends at exclamations (!) as we use semicolons(;) in C, C++, Java.

A block of code is enclosed between **`<-`** and **`->.`**

## ➔ Main function

The keywords `HI` and `BYE` mark the beginning and end of the main function

```
HI
<-
      statements!
...
->
BYE
```

## ➔ Datatypes

Integer (int): NUMBER
Floating point (float): REALNUM
Character (char): CHARACTER

Eg: `int a;` - in our language is,

```
a IS NUMBER!
```

## ➔ Comments

Comments are enclosed within NEVERMIND and THIS keywords
eg: `NEVERMIND A comment THIS`

## ➔ Arrays

`a ARE 10 NUMBERS!`
defines a as an array of 10 numbers.

For accessing the ith element of an array, we use:

```
n OF array_name
```

Eg: `a[5] = 10;` -in our language is,

```
5 OF a = 10!
```

## ➔ Input and Output

For standard input, we use GIMME

```
GIMME var_name!
```

Eg: `scanf("%d",&a);` -in our language is,

```
GIMME a!
```

Similarly for standard output, we use SHOWME

```
SHOWME "Sup Bro?"!
```

## ➔ Conditional Statements

The analog for:

```
if( expression ) {
      statements;
}
else {
      statements;
}
```

in our language is,

```
IF expression THEN
<-
      statements!
->
NOPE?
<-
      statements!
->
```

Similarly for if else if ladder is written as:

```
IF expression THEN
<-
      statements!
      ...
->
NOPE? IF expression THEN
<-
      statements!
      ...
->
```

```
NOPE?
<-
      statements!
      ...
->
```

## ➔ Loops

The construct for looping is made using GOTTADO keyword and the condition for looping is given after the WHEN keyword

```
GOTTADO WHEN expression
<-
      statements!
      ...
->
```

## ➔ Exit controlled loop:

The do-while like exit controlled loops are written as:

```
GOTTADO
<-
      statements!
->WHEN expression!
```

## ➔ 'For' Loop

The 'for' loop is used to quickly write a loop controlled by a NUMBER that incrementally iterates between two values.

```
FOR var_name FROM val1 TO val2 KEEPDOING
<-
      statements!
->
```

## ➔ Functions

Function **prototypes** are defined by:

```
      YOUGOT fn_name WITH parameters!
```

Eg: `void square(int);` -in our language is,

```
      YOUGOT square WITH NUMBER!
```

Function **calls/invocations** are defined by:

```
HEY fn_name TAKE parameters!
```

Eg: `square(4);` -in our language is,

```
HEY square TAKE 4!
```

Function **definition** for functions with no return type is:

```
CALLME fnname WITH parameters TODO
```

Eg:
```
void square(int num) {
        statements;
  }
```

Is written as:

```
CALLME square WITH num NUMBER TODO
<-
     statements!
->
```

And the function definition with return types are defined as,

```
CALLME fn_name WITH parameter TOGET returntype
<-
     statements!
     THROW var!
->
```

Here `THROW` is analogous to `return` statement in C, C++ etc.

## SAMPLE CODE TO FIND SUM OF NUMBERS FROM 1 TO N

```
BRINGITON stdio.h

YOUGOT sumton WITH NUMBER! NEVERMIND function prototype THIS

HI
<-
      num IS NUMBER!
      sum IS NUMBER!
      SHOWME "Input a number: "!

      sum = HEY sumton TAKE 10! NEVERMIND function call THIS
      SHOWME "Sum is: "!
      SHOWME sum!
->
BYE

CALLME sumton WITH num NUMBER TOGET NUMBER
NEVERMIND function definition THIS
<-
      temp IS NUMBER!
      sum IS NUMBER!
      sum = 0!
      FOR temp FROM 1 TO num KEEPDOING
      <-
            sum = sum + temp!
      ->
      THROW sum!

->
```