**We take this example:**

# work

@Include(None) #no include, just for parsing

String acs = "TestACS"

Number num = 42

Fun AnotherFun(None) -> None

{

Print("Inside AnotherFun")

Return None

}

Fun ReturnFun(None) -> Number

{

Return 123

}

Fun ReturnWithParam(Number val) -> Number

{

Return val + 1

}

Fun Main(None) -> String

{

String test = "This is a test"

Print("Hello ACS %s %d %f %b", acs, num, 1.129420, True)

Print("Test variable in func: %s", test)

{

String nested = "Nested variable"

Print("Inside nested block: %s", nested)

}

if(true) {

Print("Inside if block")

}

if(false) {

Print("This won't print")

}

if(test == "This is a test") {

Print("Condition met: %s", test)

if (test != "a test") {

Print("No there is no hello")

}

}

for(i = 0 to 5; 1) {

Print("For loop iteration: %d", i)

}

Number j = 0

while(j < 3) {

Print("While loop iteration: %d", j)

j = j + 1

}

AnotherFun(None)

Number retNum = ReturnFun(None)

Print("ReturnFun returned: %d", retNum)

Number a = 10

a = a + 5

Print("a after addition: %d", a)

Float b = 5.5

b = b \* 2

Print("b after multiplication: %f", b)

String c = "Hello, "

c = c + "World!"

Print("c after concatenation: %s", c)

String multi = "Hello, "

multi = multi + "this is "

multi = multi + "a multi-part "

multi = multi + "string."

Print("multi after concatenation: %s", multi)

if(multi == "Hello, this is a multi-part string.") {

Print("Multi-part string condition met")

}

else {

Print("Multi-part string condition not met")

}

Number returnsParam = ReturnWithParam(10)

Print("ReturnWithParam returned: %d", returnsParam)

for(i = 0 to 5; 1) {

if(i == 1)

{

Print("i is one")

}

elseif(i == 2)

{

Print("i is two")

}

else

{

Print("i is something else: %d", i)

}

}

Return "Function complete"

}

**The A.C. Script format is a format that combines in a way Python and C, being executed without a VM. This format is inspired by a competitor’s scripting format, whose syntax is derived from batch.**

**At the moment this format only supports:**

1. **Function calls (with returntype or not)**
2. **Variable assignments (with functioncall or direct)**
3. **Printing**
4. **Simple types (Strings, Nunber(Integer), Floats, Boolean)**
5. **If cases**
6. **Attribution (ex j = j + 1)**
7. **Else and Elseif cases**
8. **System execute**

**Some things to do:**

1. **System management function (being used in a factory environment for stuff like BIOS update in single script, where we need to tell which mfg and mobo is supported and which func should be used for update)**
2. **Include parsing**

**Rules of Usage:**

1. **You cannot do classes, struct, or anything else other than functions, include, vars and calls. This format is intended to be as simple as possible**
2. **The if clauses must NOT be separated by space, in that case it must be** if() **and not** if ()
3. **You cannot do functions with indented braces, only non-indented block functions will do something**
4. **If you’re planning on using this format, please note this rule when modifying executer for functions and blocks:**

If/For/While → Internal function -> Return → Is comment or empty → Function call → Variable → Block

1. **For Boolean, do not use NUMBER values (0/1) for attribution, instead use False/True.**
2. **Do not use concat, sum or diff ops when attributing a value, another reason to make it as simple as possible**
3. **For string concatenations, please do not do it in a single line, if for example you have:**

String test = “abc”

test = test + “def” + “ghi”

**You’d do:**

String test = “abc”

test = test + “def”

test = test + “ghi”

1. **The function GetValue(<source>, <where>) can ONLY be used for variable creation or attributions.**