YR++

Contents

Chapter 1

Basic Information

§0: Introduction
YR++ is an esoteric high-level programming language created by Cornea Studios on 6th October 2025. The language has changed its name a few times. First it was Ahuinka → AhuinkaScript → R++ and finally YR++. At the current moment YR++ has 5 libraries to work with but the number will increase during the development. The supported file extensions are .rpp and .yrpp

§1: Syntax
YR++ has very specific syntax in comparison with other programming languages and at the current time is only interpreted language.

Each construction in YR++ has its own block/statement. Classes, functions, conditions and loops, everything has different blocks but about that we'll discuss in the next paragraphs. This is also a strong typing language and requires to have ';' (semi-colon) at the end of each line. There is also one instruction per line.

§2 Variables
There are the most common variable types which should be explicitly declared. Types: int, float, double, bool,string, char, list and null.

Below you can see the creation process of the variables for each type. It's very simple

---

```
1   //numeric
2   int a = 8;
3   float b = 5.0534;
4   double c = 9.98327864;
```

This is the creation of all numeric variables which can only handle numbers and no text, symbols and lists

---

The variable which handles text is only called as string and nothing anymore. As you see there are two options for a string creation.
Note: pay attention if a text is number. In this case you should write it with "...".

```
1   //text
2   string name = "Bob";
3   string name2 = Bob;
```

```
5   //symbols
6   char yes = 'y';
7   char no = n;
```

For one single symbol it is better to use char as a variable type. YR++ has the same story for creation of this variable type with '...'. It's better to use '...' than without.

Boolean or bool has only two possibilities. That is true or false as in all classic languages.

```
1   //boolean
2   bool t = true;
3   bool f = false;
```

How to create a list? Simple! It's kinda similar to Python but it requires you to indicate a variable type with 'list'. It's just one example of how to declare a list but you can also make a char, string, float, double list just by typing what you need between []. For char use '...', for string "...". Lists in list is also supported. The access to a value we'll discuss later.

```
1   //list
2   list nums = [1, 2, 3, 4, 5, 6];
```

§3 Access to list element

In this paragraph we'll discuss how to get a specific value from a list. There are a few options but this time I'll show you the most common way and we'll work with the same list as in example.

Case 1: direct into output

The list starts with the index of the first element as 0. It's very important. In this case we will get direct output of value. The access is done by typing the name of the list and the index between [] as you see in the example. General form: list_name[index]. So the output will be 3.

```
1   //list
2   list nums = [1, 2, 3, 4, 5, 6];
3   print(nums[2]);
```

Case 2: direct into output but complicated list

```
5   //complicated
6   list ingewikkeld = [1, 2, [3, 4, [5, 6], 7, 8], 9, 10];
7   print(ingewikkeld[2][2][1]);
```

Here we have a main list surrounded by yellow [] and contains the next values: 1, 2, 9 and 10. The second list is surrounded by pink [] and contains values: 3, 4, 7 and 8. And finally the third list is surrounded by blue [] and contains values 5 and 6.

list ingewikkeld = [1, 2, [3, 4, [5, 6], 7, 8], 9, 10];
in light-blue is the entire list. The smaller list (in light-green) corresponds to an index of 2. Important to know: in the smaller list each element starts counting from an index of 0. And the smallest list (in yellow) corresponds to an index of 2 in the green list. It's time to get indexes per list:

Light-blue:

Index	Value	
0	1	
1	2	
2	[3, 4, [5,6], 7, 8]	→ list!
3	9	
4	10	

Green:

Index	Value	
0	3	
1	4	
2	[5, 6]	→ list!
3	7	
4	8	

Yellow:

Index	Value
0	5
1	6

Access to list in a list

Now you know how the indexation in a list works, you can get elements per index in this way:
ingewikkeld[2][2][1]

where:
ingewikkeld = list name
2 (in light-green) = index of the first list ([3, 4, [5,6], 7, 8]) in main list
2 (in pink/red/light-pink/…) = index of the second list ([5, 6]) inside of the other list
1 (in blue) = corresponds to the value of 6 in the list ([5, 6]).

So the output of the program in example will be 6.

Case 3: Assignment of a list element to a variable

There are also options to assign an element to a variable in two ways. The first one is standard for usage (See example). It could be any variable type except char and bool.

```
1   //list and assignment
2   list nums = [1, 2, 3, 4, 5, 6];
3   int number = nums[3];
4   print(number);
```

```
1   //list and assignment
2   list nums = [1, 2, 3, 4, 5, 6];
3   a; //variable declaration
4   a --> nums[3]; //assignment
5   b;
6   b <-- nums[2];
7   print("a =", a, "and b =", b);
```

The second way is longer but without the requirement to declare a variable type at the same time. It could be done with operators → or ←.
The arrows are universal operators to assign a value to a variable of any type. You can also reassign the value of the already defined variable type.

§4 Variable reassignment

Arrows could change as value as well the type of a variable depends on the new value.
Example:
before the reassignment:

```
//before reassignment
int a = 78;
print("begin =", a);
```

→ Output:

```
begin = 78
```

It's without any try of reassignment

---

after the reassignment:
in this case we've reassigned the value of a number (int) into a text (string) by using ← (may also through →).

```
5   //after reassignment
6   a <-- "Bob";
7   pt("after =", a);
```

→ Output:

```
after = Bob
```

With reassignment

---

§5 Operators

YR++ has a bunch of operators. They're divided into 2 different categories:

Arithmetics	Comparison
+, -, *, /,	>, >>, >=, <, <<, <=
**, //, %	==, !=
+=, -=, *=, /=, %=	

Arithmetic operators in the first 2 lines are the same as Python.
Comparisons have two specific operators >> and << but they are similar to > and <.

Chapter 2

Statements and Loops

§6 If, elif and else statements
These are basic language constructions which have combined different syntax from C++/C#/C, Python and own YR++ parts.

First we're going to learn how to make an if-statement.

If

This is the creation process of if-statement where if is a type of statement, condition between braces and this ends with ':' at the end. In order to open an if-statement you should use the most common in YR++ block {{ }}.

```
//conditions
if (condition):
{{
    //your code
}}
```

If you're missing one you'll get an error:
YRPP0002: **'{{' expected in order to open statement" / '}}' expected in order to close statement**. Everything depends on which part of {{ }} is missing.

Elif

```
//conditions
elif (condition):
{{
    //your code
}}
```

Elif comes from Python but reworked into YR++ style. It should be used in combination with an if-statement, but not before the if.

It has literally the same function as else if in C/C++/C#/JS/…

Executed only if the if-statement is equal to false and isn't executed/fulfilled.

Else

Else comes after an if-statement or an elif-statement and also as in all standard languages contains no condition field. Block is also {{ }} and requires ':' at the end of the keyword (else).

Executed only if the next statements are false for execution:
if and/or elif.

```
//conditions
else:
{{
    //your code
}}
```

---

§7 Loops

At the current moment we have only 2 loops: for and while. Each with different syntax.

For

Increasing for

YR++ is completely different in comparison with other languages because of its syntax.

```
for (i in (1 --> 5); i++; target=6):
{{
    //your code
}}
```

Here we have to discuss each moment between the main braces of the loop.
At first the variable i. We declare i in the loop and it becomes a local variable for this loop only.
Between nested braces we choose the range of numbers/parameters from where to where this loop will go.
About i++: i++ is the standard step format per 1. So the loop will go like this: 1, 2, 3, 4, 5. This is also possible to place variables in the range. If you want to change it, then you need to put in place of the second '+' a number (IMPORTANT: number of the int type).
The parameter target defines where the loop ends. We recommend using 1 more than your maximum number in range, otherwise 5 in this case will be excluded.
Note: this is an example of increasing loop

Descending for

There is also a descending for loop. It is a bit different from increasing.

```
for (i in (5 <-- 1); i--; target=0):
{{
    //your code
}}
```

Everything that you need is:

1) changing the 5 and 1 of place
2) change the arrow (→) to this one (←) so you're saying this loop to go from 5 to 1
3) change the i++ to i– or i-m where m is a number (step)
4) we recommend to use target as 0 for this type

It is important because the output will be: 5 4 3 2 1

While

While-loop works until the condition won't be true.
If the condition is still false, the loop will continue.

```
while (condition):
{{
    //your code
}}
```

Chapter 3

Functions and Classes

§8 Classes

Classes are classic structures for information. They could contain: functions, variables, conditions, loops etc.

```
class Person
{[
    //your code
]}
```

This is the creation process. The name of a class could be also written in lower case.

We call functions from one class in another one by writing:
className.functionName();

---

§9 Functions

YR++ contains 3 function definitions which return void as a basic type. Let's see how we make a function first.

Function with function type: func (basic)

```
func main(p1, p2, ..., pn)
{{
    //your code
}}

main();
```

An interesting thing about functions and YR++ itself. There are no entry points required to write an executable code.

Function names could be also in big as in small letters.

Between braces you give the parameters which your function needs (optional), they should be without type.
Once you've declared a function you should "activate" it by writing its name under the function (Note: not above).

Function with function type: void

Void is more classic type of function for people who used to any standard language like Java, C# etc.
YR++ has the exact same way for this function to declare it, but it may also return a value.

```
void main(p1, p2, ..., pn)
{
    //your code
}

main();
```

Function with function type: def

```
1    def main(p1, p2, ..., pn):
2    <_
3        //your code
4    _>
5
6    main();
```

Def is a more specific type, because of its connection with Python but not as much as you may think.

Difference: declaration and usage + specific blocks <_ _> like it was with func.

§10 Function's return value

Any function type does return nothing. But by using: return smth; function returns a value/message.