**Summary**

|  |  |
| --- | --- |
| **Language Name** | Turtle |
| **Language Paradigm** | Declarative (Functional) |
| **Translation Type** | Interpreted (Mostly) |
| **Trans-piled To** | Python (3.x) |
| **Lexical Analyzer** | Flex (or a self-written in Python) |
| **Parser Generator** | GNU Bison |
| **Context-Free-Grammar Validator** | GNU Bison |

**Explanation**

**Why Functional language?**

As we know there are 3 major types of languages i.e. Imperative, Declarative (Functional), and Logical. The reason we decided to make Turtle a functional language is the simplicity of syntax as well as semantics. The tradeoff that we are willing to make, for now, is the efficient execution. We also aim to allow as less as possible side-effects with Pure Functions, which can occur by functions in other paradigms.

**Trans-piling to Python**

Python is a versatile language with enriched libraries and modules. Its syntax is simple and small. Hence, trans-piling some language to python is rather convenient. Furthermore, the community support of python can help in many underlying tasks to be performed.

**Turtle as An Interpreted language**

Turtle will be trans-piled to python which is mostly an interpreted language. To convert interpreted code another interpreted code is simpler. Moreover, we don’t aim to optimize code written in Turtle so compiler won’t be absolutely necessary.

**Flex as Lexical Analyzer**

Flex provides many functionalities of a lexical analyzer as built-in, so we don’t have to write those from scratch. Furthermore, if at any time we need to have a dedicated lexer, we will write it in Python.

**GNU Bison for Parser Generation**

Bison is a general purpose parser generator that converts a grammar description for a context-free grammar into a C program to parse that grammar. Again, it provides many built-in features, so we don’t have to write our own parser.

**Reserved Words**

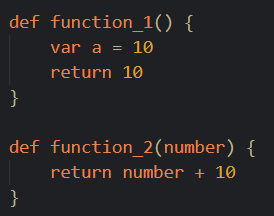
|  |  |
| --- | --- |
| def | break |
| if | continue |
| else | class |
| elif | true |
| or | false |
| and | null |
| var | for |
| const | while |
| import | return |
| print |  |

**Keywords**

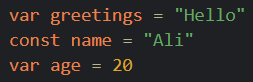
|  |  |
| --- | --- |
| def | const |
| if | var |
| else | return |
| print | null |
| elif | true |
| or | false |
| and |  |

**Syntax**

**Defining a Function:**



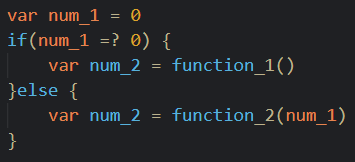
**Declaring and Initializing a Variable and a Constant:**

****

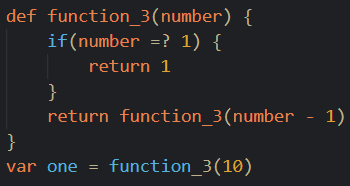
**Calling a Function:**

****

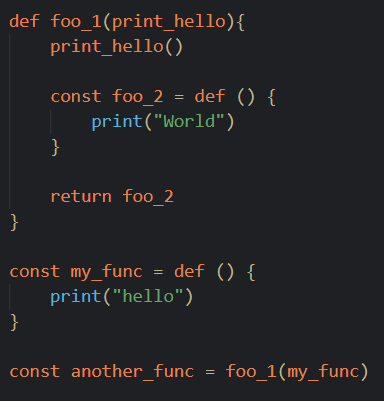
**Control Statements:**

****

**Recursion:**

****

**High-Order Functions:**

****