GROUP 7

Name of Language: ACE

Members:

Ayush Modi - 21110039

Kaushal Kothiya - 21110107

Anish Karnik - 21110098

Pratham Sagar - 21110165

Language Syntax: -

- Assignment and Let expressions.
    - **cook for let:**
        - Eg: cook <datatype> <variable name> = <value>;
    - **Assignment operator:** “=” (Equals sign)

- Number type and simple arithmetic operations (both unary and binary).
    - **num** - declaring a numeric value
    - Eg. cook num <variable name>=<val>/<variable of same datatype>;
    - **Binary Operations:**
        - (+) - Addition
        - (-) - Subtraction
        - (*) - Multiplication
        - (/) - Division
        - (#) - Power
        - (&) -BITWISE AND
        - (|) - BITWISE OR

        - (&&) - AND
        - (||) - OR
        - (^)- XOR
        - (%) - MODULO
    - **Unary Operations:**
        - (++) - ADDING ONE
        - (--) - DECREMENTING ONE
        - (!) - NOT

- Boolean type, comparisons, and if-else.
    - **flag** - declaring boolean value and value of a flag can be **'true'** or **'false'**
    - Eg. cook flag <variable name>= true / false/<variable of same datatype>;
    - **>** greater than
    - **<** less than
    - **>=** Greater than or equal
    - **<=** less than or equal
    - **==** equal
    - if/else/else if

- Strings with concatenation and slicing.
    - **str** - declaring a string
    - cook str <string name>= ".....";
    - '**+**' - **'str1 + str2'** will concatenate str1 and str2
    - **strname[x:y]** will give the string from index x to y
    - For accessing character at i^{th} index : <string name>[i]

- List/array type with basic operations like length, head, tail, cons.
    - ARRAY:(elements cannot have different data types)
        - Declaration :cook <datatype> <arrayname>[length of array] ;
        - Declaration +Initialization :cook <datatype> <arrayname> = {values comma separated } ;
        - To calculate length: len(<arrayname>)
        - For head: headof(<arrayname>)
        - For tail: tailof(<arrayname>)
        - For Cons: <array name> = <element> :: <array name>
        - For accessing element at i^{th} index : <arrayname>[i]
        - For modifying element at i^{th} index : <arrayname>[i]=<some value> ;

- TUPLE: (elements can have different data types)
  - Declaration +Initialization :cook <tuple name> = (values comma separated) ;
  - To calculate length: len(<tuplename>)
  - For accessing element at i^{th} index : <tuple name>[i]
- LIST:(elements can have different data types)
  - Declaration : cook <list name>=[] ;
  - Declaration +Initialization :cook <listname> = [values comma separated] ;
  - To calculate length: len(<list name>)
  - For accessing element at i^{th} index: <list name>[i]
  - For head: headof(<list name>)
  - For tail: tailof(<list name>)
  - For modifying element at i^{th} index: <list name>[i]=<val> ;

- An operation that prints values to the screen.
  - **echo(expression to be printed) ;**

- Branches and Loops
  - Branches : {.....}
    - **if/else if (boolean expression or integer ->{**false if 0 else true**}) {...}**
    - **else {...}**
  - Loops :
    - **floop**(declaration + initialization ; stopping condition ; iteration){.....}
    - **wloop**(stopping condition){.....}

- Functions
  - func <data type/void> <function name> (parameters with data type comma separated){..... return something }

- Closures
  - Curly Brackets : {}

- End of line is represented by semicolon
  - Eg. cook num var = 5 ;

- Exception-handling constructs (try, throw and catch)
  - try{.....}
  - catch(parameters){....}
  - throw (exception)

- Future Scopes:
  - +=, -= - Add, Subtract unary operation
  - >>, << - bit shift operation
  - push(<arrayname>,val), pop(<arrayname>) - push and pop from an array