Stuff - the language

Accepted characters

In the range of `0x21` to `0x7E`

Variables

Variables are `!`, `"`, `#`, ... all the way to `~`.

Initializing/Setting a variable

The variables are initially undefined, to use them, you must initialize them. The example program sets the variable `a` to `Hi`.

```
!aS02Hi
```

The assignment opcode is `!`. Then we have the variable name, `a`. Then we have the type, `S` (a string).

Opcode	Definition	Example	Description	Variable value
`S`	String	`!aS02Hi`	Sets the variable `a` to a String of length `2`	`Hi`
`n`	Unsigned number	`!an3101`	Sets the variable `a` to a uNumber of length `3`	`101`
`N`	Signed number	`!aN3-101`	Sets the variable `a` to a sNumber of length `3`	`-101`
`s`	Charecter	`!as[`	Sets the variable `a` to a Char	`[`
`u`	Undefined	`!au`	Undefines/Clears the variable `a`	
`B`	Boolean	`!aBt`	Sets the variable `a` to a Bool	`true`

Note: with a `sNumber`, for a positive number, write `!aN3+101` (sets `a` to `101`)
Note: with a `Bool`, for `false`, write `!aBf` (sets `a` to `false`)

Copying variables

You can copy variables to other ones. For example if you want to copy the variable `Z` to the variable `@`:

```
"Z@
```

Addition

If you want to add `@` to `#` and store the result in `@` then you could do:

```
#@#
```

Where `#` is the operation and `@` and `#` are the two operands.

Subtraction

If you want to subtract `@` with `#` and store the result in `@` then you could do:

```
$@#
```

Where `$` is the operation and `@` and `#` are the two operands.

Multiplication

If you want to multiply `@` with `#` and store the result in `@` then you could do:

```
%@#
```

Where `%` is the operation and `@` and `#` are the two operands.

Division

If you want to divide `@` by `#` and store the result in `@` then you could do:

```
&@#
```

Where `%` is the operation and `@` and `#` are the two operands.

Exponents

If you want to exponentiate `@` with `#` and store the result in `@` then you could do:

```
'@#
```

Where `&` is the operation and `@` and `#` are the two operands.

Whitespace

Spaces, tabs, and newlines only **between operation groups** are allowed.

For example, the following is valid:

```
!aS12Hello World!

    *a
```

But the following is **not**:

```
!  a S 12Hello World!
   *   a
```

Comments

Comments are defined by `~`, followed by their length.

Ex.:

```
~07 Hello!
~02Hi
~10 very long
```

If you would like a string *longer* than 99 charecters, just do multiple comments

```
~99veryveryveryveryvery (etc) very~20veryverylong string!
```

Output

To print the value of the variable `$`:

```
*$
```

Str-to-int

To convert a string (in `@`) to an integer (written to `@`):

```
(@
```

Int-to-str

To convert a integer (in `@`) to an string (written to `@`):

```
)@
```

Output without trailing newline

To print the value of the variable `$` (without the trailing newline):

```
+$
```

Input

To write the user input to the variable `$`:

```
,$
```

Delay

To delay the program 1 second (1000ms):

```
-00001000
```

Loops

To print the content of the variable `$` 10 times:

```
.010<*$>
```

If statements

If you want to print the content of the variable `!` if the variable `R` is equal to the variable `J`, you can do:

```
/R=J<*!>
```

Opcode	Definition	Example
`=`	Equal to	`/R=J<*!>`
`>`	Greater than	`/R>J<*!>`
`<`	Smaller than	`/R<J<*!>`

Opcode	Definition	Example
`g`	Greater or equal than	`/RgJ<*!>`
`l`	Smaller or equal than	`/RlJ<*!>`
`!`	Not equal to	`/R!J<*!>`

The rest

Reserved for expansion (returns a FutureWarning)

```
WARNING: FutureWarning: opcode 'I' not defined
```

Warnings

`FutureWarning` : undefined opcode that may be used in the future
`DepricatedWarning`: opcode that is not in use anymore

Errors

`InvalidOpcodeError` : invalid opcode outside of the `0x21` - `0x7E` range
`InvalidArgsError` : invalid arguments for an opcode
`InvalidTypeError` : invalid type for operation, e.g. you can't add a string to a boolean

Planned expansions

- input
- functions
- modules (import, export)
- file i/o
- classes
- gui (tkinter?)