# cell

## Summary

**draft version: 0.3**
**rev date: 020521**

audio visual poetic language

## Description

cell is a stack-based two-dimensional esoteric programming language made to create short audio visual poems.

cell programs are written by placing letters on a 10x10 grid with each cell of the grid containing one letter/sound only. The grid is not displayed in the final output, it only defines display rules and operation space.

The cell programmer acts as an invisible entity called **atom**.

The atom has 4 stacks called **slots** used for the most fundamental operations :

- **X**: horizontal position
- **Y**: vertical position
- **CHAR**: unicode character decimal value
- **OSC**: audio oscillator shape

Once set, a slot value is kept in memory until it’s popped out of the stack.

There is an optional **VAR** slot used to create a unique variable and store its value. It is intended to perform calculations by retrieving values in an instruction sequence. A variable must be declared with the **SYM** operator to define its corresponding symbol **which must be different from cell reserved operators

cell uses a base 10 numbering system, counting starts from 0. Decimal values can also be calculated from basic arithmetic operations using postfix notation. It is also possible to do arithmetic operations on the current value of a slot.

The order of operators doesn't matter inside of the instruction.

One can write a letter to a specific cell by specifying its (x,y) coordinates using predetermined unicode symbols in place of numbers. This way, a letter can be selected by its corresponding decimal code.

Audio can also be generated by writing dedicated symbols on the same grid, each cell holding a single oscillator pulse with a specific frequency and ADSR envelope parameters.

The atom can write a sequence of steps to output a sound from a cell.

There is a silence of 1000 ms between each step by default.
The speed duration can be specified with the BPM operator.

A cell audio program can output sound by defining the note sequence of each cell. Blank cells output silence.

# OP

## Coordinates

| COORD | X | Y |
|---|---|---|
| OP | ⋯ | ⋮ |

## Numbers

| NUM | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| OP | ○ | ● | ∟ | ▶ | ⊹ | ▣ | ∈ | ◆ | ☍ | × |

## Arithmetic

| CMD | + | - | * | / |
|---|---|---|---|---|
| OP | ⊷ | ⊸ | ∞ | ⧖ |

/: floor integer division

## Commands

| CMD | CHAR | OSC | WRT | RMV | DOT | JMP |
|---|---|---|---|---|---|---|
| OP | : | ≃ | :: | ∸ | · | ↬ |

CHAR operator is used to define a unicode decimal value

OSC operator is used to define an audio oscillator shape

WRT operator is used to write a letter or oscillator shape into a cell based on current slot values

RMV operator is used to remove a letter or oscillator shape from a cell based on current slot values

DOT operator is used to mark the end of a decimal sequence

JMP operator can be used to move the atom from a cell to another without modifying slot values.

## Variables

| **CMD** | VAR | SYM |
|---|---|---|
| **OP** | ◉ | ≷ |

VAR operator is used to declare and define the value of the optional variable slot

SYM operator is used to define a custom symbol as a placeholder for the VAR slot value

## Oscillators

| **OSC** | sine | square | sawtooth | triangle |
|---|---|---|---|---|
| **OP** | ~ | ■ | ⥊ | ▲ |

## Sequencer

| **CMD** | BPM | FREQ | NOTE | GAIN |
|---|---|---|---|---|
| **OP** | ⁘ | ∴ | ⁖ | ⁞ |

BPM operator defines the duration between each steps in milliseconds

FREQ operator defines the audio frequency of a cell

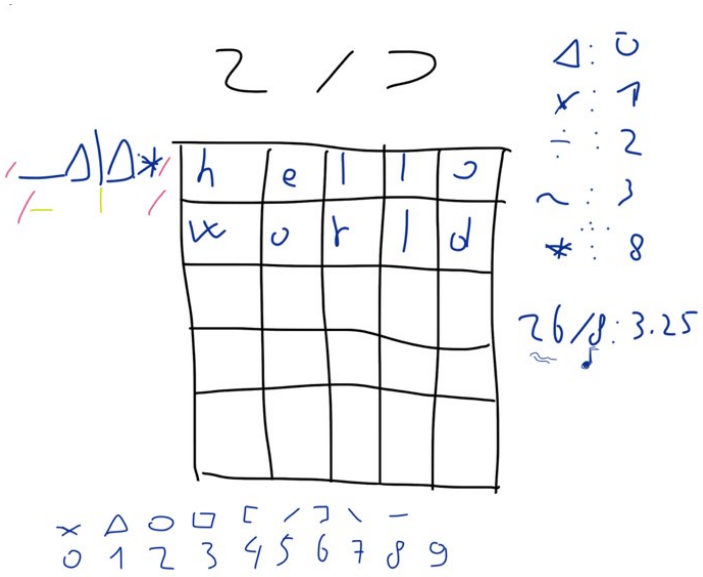
NOTE operator defines the note of a cell

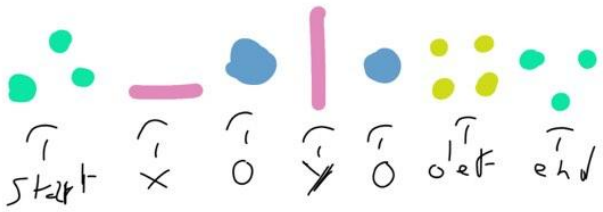
GAIN operator defines the audio volume of a cell

## Examples

### ver. 0.1

*letter are selected by alphabetical order instead of unicode




∴_○|○::◆∵
∴_●|○::▶∵
∴_⌓|○::●⌓∵
∴_⊹|○::●ω→∵
∴_▣|○::●▶∵

∴_○|●::⌓⊹⁚
∴_●|●::●▶⁚⁚
∴_⌓|●::●◆⁚
∴_⊹|●::●⌓⁚
∴_▣|●::▣⁚⁚

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| h | e | l | l | o | | | | | |
| w | o | r | l | d | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

ver. 0.2

···▶⁝⊹:××::···⊹⁝⊹:●○●::···▣⁝⊹:●○ ☍ ::···∈⁝⊹:▣⊹•L∞::···◆⁝⊹: ☍ ◆▣×::

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | c | e | l | l | :: | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |


···▶⁝⊹:××::

···⊹⁝⊹:●○●::

···▣⁝⊹:●○ ☍ ::

···∈⁝⊹:▣⊹•L∞::

···◆⁝⊹: ☍ ◆▣×::

```
X3Y4DEF99EOL
X4Y4DEF101EOL
X5Y4DEF108EOL
X6Y4DEF54DOT2*EOL
X7Y4DEF8759EOL
```

## ver. 0.3

user act as an invisible entity called **atom**

The atom has 4 main stacks called slots:

- **X**: horizontal position
- **Y**: vertical position
- **CHAR**: unicode character decimal value
- **OSC**: audio oscillator shape

There is an optional **VAR** stack used to create a unique variable and store its value. It is intended to perform calculations by retrieving its value in an instruction sequence.

Once set, a slot value is kept in memory until it’s popped out of the stack.

For example, the following prints a text vertically by defining the X slot only once at the beginning of the instruction:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | t | | | | | |
| | | | | o | | | | | |
| | | | | w | | | | | |
| | | | | e | | | | | |
| | | | | r | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

⋯⊹
⁝▶:●●∈⁚⁚
⁝⊹:●●●⁚⁚
⁝▣:●●×⁚⁚
⁝∈:●○●⁚⁚
⁝◆:●●⊹⁚⁚

```
X4
Y3CHAR116WRT
Y4CHAR111WRT
Y5CHAR119WRT
Y6CHAR101WRT
Y7CHAR114WRT
```

It is also possible to do arithmetic operations on the current value of a slot, to evaluate another unicode decimal value for example:

⋯⊹
⁝▶:●●∈⁚⁚
⁝⊹:▣→⁚⁚
⁝▣:☍↣⁚⁚
⁝●↣:●○·●○·●ω↣⁚⁚
⁝◆:●▶↣⁚⁚

```
X4
Y3CHAR116WRT
Y4CHAR5-WRT
Y5CHAR8+WRT
Y1+CHAR10DOT10DOT1*+WRT
Y7CHAR13+WRT
```

Same exemple, using the VAR slot:

⋯⊹
◉●●○≷∴
⁝▶:∴∈↣⁚⁚
⁝⊹:▣→⁚⁚
⁝▣:☍↣⁚⁚
⁝●↣:∴×→⁚⁚
⁝◆:●▶↣⁚⁚

```
X4
VAR110SYM10292
Y3CHAR1106+WRT
Y4CHAR5-WRT
Y5CHAR8+WRT
Y1+CHAR102929-WRT
Y7CHAR13+WRT
```

Audio sequencer exemple

⋯●
⋮●
:~

```

```

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | ~ | | | | | | | | |
| | | | | | | | | | |
| | | | | ~ | | | | | |
| | ■ | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | ■ | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

## Ideas

- add color operator followed by 3 chars of color hex code

- possible to create variables of types int char ascii hex

- possible to create variables that acts as placeholder for sub instruction using a custom symbol for the whole subinstruction

- audio and text can be used as modulation sources both ways e.g. to change a group of letters style from gain or to transpose a note by “patching” a letter from another cell.