intro to fig (presentation edition)

fig is based on these 7 concepts:

- variables
- input
- output
- basic math
- loops
- conditionals
- functions

the fig language:

- goes → left → to → right
- ignores(parentheses())
- ignores

    indentation
- IGNORES cASE

the main variable

a few keywords in fig are stand-alone (they get their own line,) but most are designed to share a line with **a "main variable" that begins each line** of code:

```
now "Hello, world!"
```

...sets variable **now** to string "Hello, world!"

commands that get their own line:

- **graphics** / **textmode**
- **for** / **forin**
- **while** / **break** / **pass**
- **function**
- **python** / **try** / **except** / **resume**
- **fig** / **next** / **nextin** / **wend**
- **iftrue** / **ifequal** / **ifmore** / **ifless** / **else**

from left to right:

now	"Hello world"	print
1. set **now** to 0	2. change **now** to the string defined here	3. display contents of **now** on the screen (or tty)

copying variables is like setting them

```
x     "hello there"
----------------------
```

- will set x to string

```
y     x
----------------------
```

- will set y to the contents of x

    (or if you prefer: copies x contents to y)

order of keywords counts

```
 1      2        3     4
now  "hello"  print  ucase
---------------------------
```

- **puts "hello" on screen**

```
now  "hello"  ucase  print
---------------------------
```

- **puts "HELLO" on screen**

all these forms are valid:

- `mainvariable 5.8`
- `mainvariable "string"`
- `mainvariable` *variable*
- `mainvariable` *command*

some commands read/write the main variable

some commands:

- ...only read the main variable, using it for some purpose
- ...only write to the main variable, without using it
- ...neither read nor alter the main variable
- ...both read and alter the main variable

`left` command always has 1 parameter

1	2	3	4	5
now	"hello"	print	left 2	print

1. set **now** to 0
2. change **now** to "hello"
3. display contents of **now**
4. change **now** to **2 left**most characters
5. display contents of **now**

output is:

```
hello
he
```

fig commands have a fixed parameter count

`left 2`

command parameter

- most commands have 0 parameters
- all commands have a fixed # of parameters...
- ...all except **`function`**
- **`function`** makes custom commands

almost zero syntax is required

- strings go **`"in quotes"`**
- hashes **#** start comments
- decimals work: **`5.2`** as you would expect

- these are decorative: **`( ) ; : , | =`**

variables as parameters

like most commands that have parameters, `left` doesnt require a "constant" value like 2, but can use a variable instead:

```
howmany 2
now  "hello"  print  left howmany  print
```

- using variables as parameters allows the program to take input and gain more effect over output, even before we get into conditionals.

four data types

- **string** type: data "in quotes" is **a string of characters**, which may include numbers but they will not be treated as numeric data. `"5" plus "5"` is `"55"`
- **integer** type: numeric, **whole-number** data. `5 plus 5` is `10`
- **float** type: numeric data **with decimals**. `5 plus 5.7` is `10.7`
- **array** type: instead of a variable holding one value at a time, a variable can be converted to an array **holding the same value, plus others** (of one or multiple types.) `5 plus "5" plus 5.0` is `[5, '5', 5.0]`

creating an array

```
now 5 arr # create an array holding 5
now 5 arr times 100 # array has 100 5s
```

normally the `times` command is used for basic multiplication, but when used on a string it sticks several copies of that string together into one:

```
p "hello" times 4 # hellohellohellohello
```

reusing main variables

when you start a line with a variable youve already used, it **resets the value** to zero:

```
p   print            # displays 0
p 5 print            # displays 5
p   print            # displays 0 again
p   plus 7  print    # displays 7
```

- **arrays are the exception** to this rule.

arrays dont reset automatically

since an array may hold a lot of information, it is not automatically reset when used as the main variable:

```
p 5  arr  print  # displays [5]
p         print  # displays [5]
p 0       print  # displays 0
```

other methods of array creation

- the **`split`** command can split a string into array elements.
- **`arropen`** and **`arrcurl`** load the lines of a text file or webpage into an array.
- **`arrstdin`** creates an array "piped" in from another program.
- **`command`** creates an array from parameters used while calling a fig program from the command line.
- inline **`python`** and **`arrshell`** can create arrays.