The Editor

Your first steps with Smojo

Smojo is a new programming language for Data Science and Artificial Intelligence. It is fast, easy to learn and use.

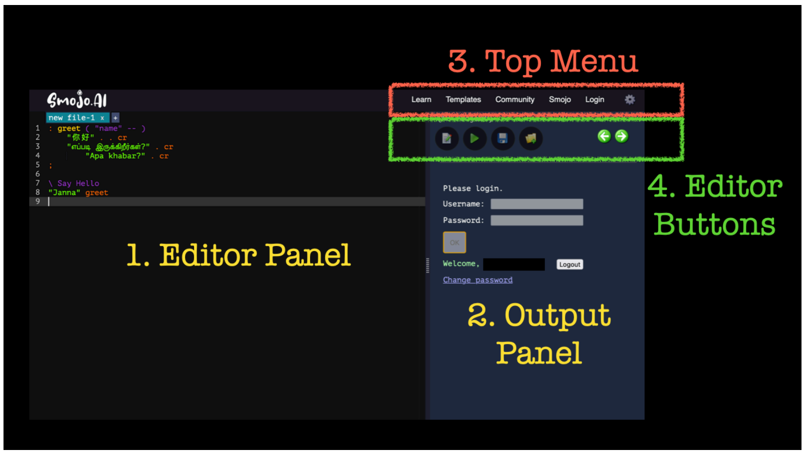
As you will see, Smojo is cloud-first and comes bundled with a community of users you can reach out to for help, share code with and get new ideas.

Using the Editor

To begin, point your web browser to: https://smojo.ai/editor . You should see a layout similar to Figure 1 below.

The layout on Figure 1 will differ for small screens on phones and tablets. In these, the Editor Panel is at the top, and the Editor Buttons and Output Panel at the bottom.

The Top Menu is hidden on small screens. You need to click on a blue arrow icon to reveal this menu.



```
: greet ( "name" -- )
   "你好" . . cr
   "எப்படி இருக்கிறீர்கள்?" . cr
      "Apa khabar?" . cr
;

\ Say Hello
"Janna" greet
```

```
Please login.
Username:
Password:
Welcome,
```

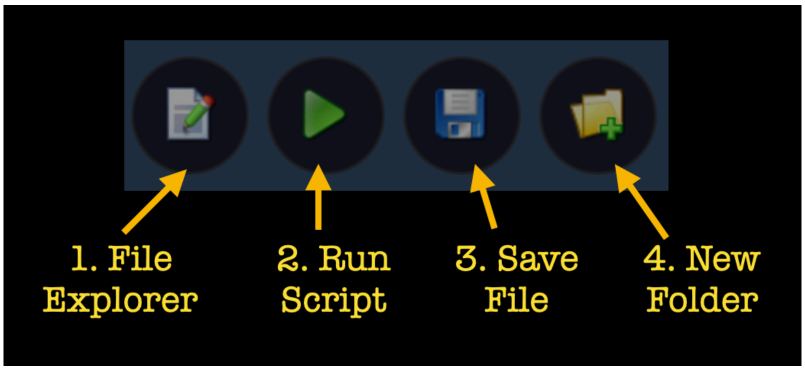
Figure 1 - the layout of the Smojo Editor.

There are 4 main sections:

1. The **Editor Panel** is where you do all your coding.
2. The **Output Panel** is a multi-purpose panel to list files, show documentation and display the output of your program.
3. The **Top Menu** shows several words (Learn, Template, Community, Smojo, Login and the gear icon), each of which will lead you to a new page of content. We'll come to these in due course.

Lastly, there are the **Editor Buttons**, which you will frequently use when programming.

Figure 2: The important Editor Buttons



File Explorer - when you click it, will display your files and folders in the right Output Panel.

Run Script - which you click on to run the code displayed on the Editor Panel.

Save File - when clicked, saves the code currently displayed on the Editor Panel.

New Folder - creates a new folder in which you may save files.

Logging In

Your first step to using Smojo is to **login**:

- In the Top Menu, click Login.
- A form with two input fields and an OK button should appear in the Output Panel.
- Enter your username in the field provided. **Use only lowercase letters**.
- Enter your password in the second field
- Click **OK**

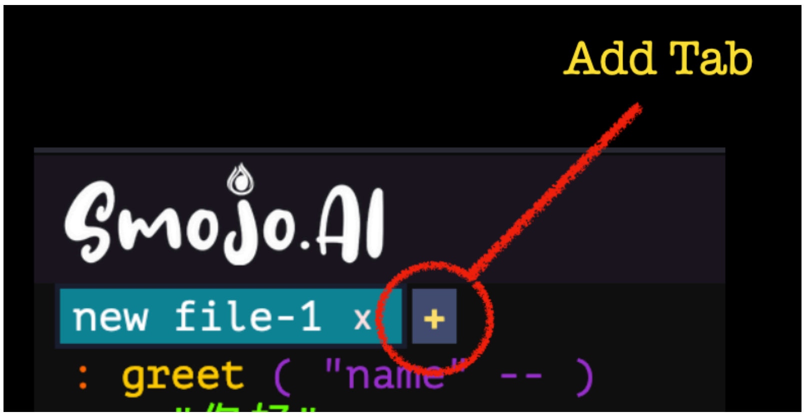
If you entered your credentials correctly, you should be logged in and ready to go!

If you're using a smartphone or tablet, the Login menu may not appear, you have to click the blue "down arrow" button (shown on the left) which will reveal the Top Menu for you to select from.

Your First Program

Click the **Add Tab** icon on the Editor Panel as in Figure 3 below. This will open a fresh tab where you can begin to code.

Figure 3: The add tab icon



```
: greet ( "name" -- )
```

In this tab on the Editor Panel, type it in the following code:

```
“Hello World” .
```

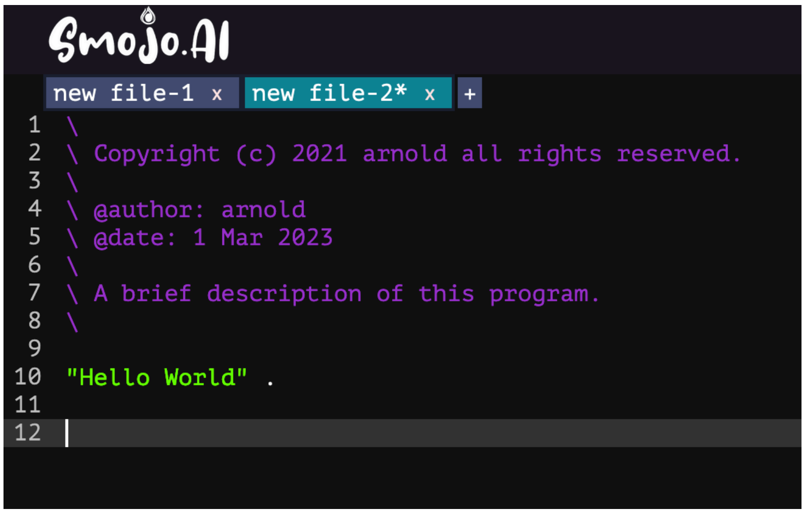
There are several points to pay attention to in this short program:

- Throughout this book, we will use the convention of using `red letters` for code.
- If you’re reading this book on a PDF, do not copy-paste from the PDF, because it frequently uses the wrong symbol for double-quote. Manually type in the program instead.
- In the program, note the space between the last double quote and the period `.` Spaces are important in Smojo.

If all is well, you should see something similar to Figure 4 below:

The top sections in purple are called **comments**, and these are important notes to programmers reading your code, but are ignored by Smojo itself.

Figure 4: The Hello World program



```
 1  \
 2  \ Copyright (c) 2021 arnold all rights reserved.
 3  \
 4  \ @author: arnold
 5  \ @date: 1 Mar 2023
 6  \
 7  \ A brief description of this program.
 8  \
 9
10  "Hello World" .
11
12
```

In Smojo, comments are created by a preceding backslash `\` with a space. This renders all anything to the right as a comment.

Our actual program in Figure 4 is on line 10. The actual line number may be different for you. Notice how the text is coloured differently from the period. That's a helpful and important distinction:

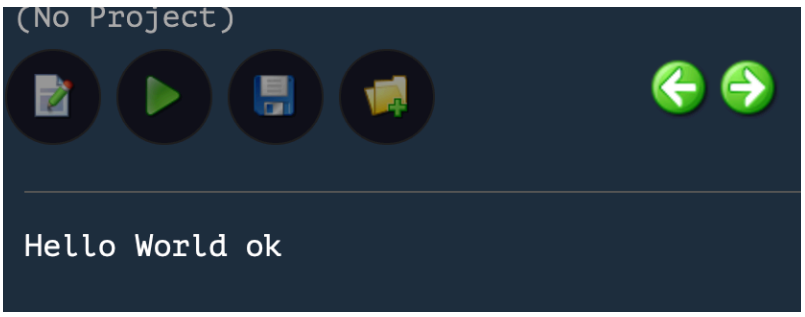
- The text **`“Hello World”`** is printed in green on the Editor Panel. It is considered a **literal** , meaning its value is what it says. For example, **`56`** is a number and it is also considered a literal.
- The period **`.`** is considered a **word** or action. Its value is the result of it performing an action.
- In Smojo, spaces are important, so put a space between the literal **`“Hello World”`** and the period **`.`**

To execute this program, click the **Run Script** button. You should see the program's output appear on the right Output Panel as in Figure 5:

So, Figure 5 says that the result of running the program is the **`Hello World`** displayed to you, followed by **`ok`**.

The **`ok`** tells you that your program ended normally. All successful programs in Smojo end in **`ok`**.

Figure 5: Output of the Hello World program



```
Hello World ok
```

Quiz 1.0

1.0.0: What happens if you delete the space between the last double quote and the period?

1.0.1: What happened if the literal `“Hello World”` and the period word `.` are on different lines?

Experiment with these changes!

Left to Right

Smojo code is executed from **top to bottom,** and **left to right** sequentially.

Words act on whatever precedes them on the left.

For example, in our Hello World program, to the left of the period `.` is the literal `“Hello World”` , which acts as the input to the period.

The period doesn't create any output on its right since its action is to print to screen. However most other words do have an output, which is implicitly placed on the right.

Take the following program:

```
1 2 +
```

This means that `1` and `2` act as inputs to `+`, since they are on the left of it.

Quiz 1.1

1.1.0: Copy this program into a new tab and try running it. What output do you observe? What output did you expect?

1.1.1: Try adding a period `.` to the right of `+`, making sure to insert a space between them. Run the program. What output do you observe? Does this match your expectations?

The left-to-right rule is universal in Smojo. This means that we must write

```
1 2 + \ works!
```

instead of the more natural

```
1 + 2 \ won't work
```

which will not work. Recall, the items in `purple` mean comments, which are ignored by Smojo.

In exchange for this inconvenience however, Smojo offers us surprising power in writing concise programs, which I hope you will see in later chapters.

Importance of Spaces

One immediate benefit of the left-to-right rule is that there is no need for parentheses in Smojo. Instead, **spaces** are important in Smojo. For example, this code won't work:

```
1 2+ . \ won't work
```

because Smojo needs a space between the literal `2` and the word `+`

```
1 2 + . \ works!
```

In the last example above, `1 2 + .`

- the input of the `+` is to the left of it, ie, `1` and `2`.

- the `+` outputs `3`, which is put on its right.
- the input of the period `.` is to the left of it, ie, the output of `+` which is `3`

A Longer Program

So far, we've written tiny programs - each just one line long! Let's try something longer and a little more challenging.

How would you find the average of two numbers?

1. First add them
2. The divide by 2

Don't confuse the backslash `\` which starts a comment with the slash `/` which is an integer divide.

We already know that `+` adds 2 numbers on the left. `2 /` divides the leftmost number with the number `2`. So, putting them together, the fragment

```
+ 2 /
```

acts like an **average** operation, since it adds any two numbers on its leftmost side and outputs the average on the rightmost side. For example, run this 3-line program:s

```
2 4

+ 2 /

.
```

You should see **3 ok** being displayed in the Output Panel. Let's break it down line by line:

- Recall that Smojo does things in sequence, top to bottom, left to right. This mean that in this 3-line program, the first line `2 4` is run first. Since both `2` and `4` are literals, ie their values are themselves, then the result is also two values, `2 4` as input to whatever follows.
- On the second line, `+` adds `2` and `4` since they precede it, and outputs the result, `6` to the right.

- `2` acts as its own value, so the values on the left are now `6` followed by `2`.
- `/` takes the `6` and `2` , performing the division, resulting in `3` on the right of it.
- finally, `.` takes the `3` on its left and prints `3` to screen. It produces no output.
- there are no longer any operations so Smojo prints `ok` and comes to a halt.

Take time to understand each step carefully. Don't move on until you have thoroughly understood what is happening.

It may be that you may be wondering if anything interesting or useful can be done this way. Shelve these concerns for now and focus on how Smojo runs a program. The benefit of Smojo's way of doing things will be very clear later on. But you have to grasp the details first.

New Words

What if we want to take the average two pairs of numbers, then subtract them? Say `2 4` and `6 8` ? How would we program this with what we've learnt thus far?

Here's one possibility:

```
2 4 + 2 /
6 8 + 2 /
- .
```

This is program is easy to write but **hard to read.** Easy to write since we already know `+ 2 /` is the average, but hard to read since we need to figure that our ourselves.

It would be much better if we could somehow name the fragment `+ 2 /` as `average`. Then, we might be able to do:

```
2 4 average
6 8 average
- .
```

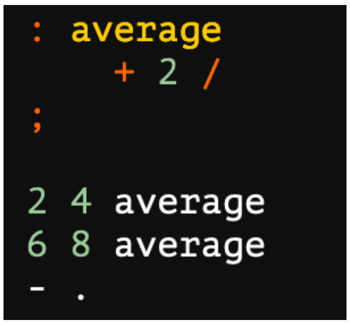
This code is easy to both write and read. Easy to read since we know what the English word average means.

We can "package" code fragments into new words with the help of the words `:` called "colon" and `;` called "semi-colon" like so:

```
: average
     + 2 /
;
```

This means we can now use `average` like any other Smojo word. Figure 6 shows the full program.

Figure 6: Defining a new word average



```
: average
    + 2 /
;

2 4 average
6 8 average
- .
```

The words `:` and `;` are not punctuation (remember, in Smojo, there is only Words or Literals). The word `:` begins the definition of a new word while the word `;` completes it.

Quiz 1.2

1.2.0: Write and run the program in Figure 6. Do you get the expected answer?

1.2.1: Amend your program by replacing `4 2 average` with `2 3 average`. What do you expect the answer to be now? Run the program with the changed values. Do you get the expected answer? Now try replacing `/` with `/.` and `–` with `–.` then rerun again. Now do you get the expected answer now? `– + / *` are integer subtraction, addition, division and multiplication. `-.` `+.` `/.` and `*.` are their real-number counterparts.

*__1.2.2:__ How would you write a new word `diff-average` that finds the difference of the average of two pairs of numbers? Eg:

```
4 3 2 1 diff-average .
```

should display `2.0 ok` Hint: reuse the `average` word. Write out your code and test.

Learning Points

- You've briefly learned about the different parts of the Smojo Editor.
- You understand that Smojo processes its data from left to right (and top to bottom).
- You are able to create new words using `:` and `;`
- Most Smojo words are often just 2 - 3 lines of code, very rarely more than 5 lines long. This rule makes Smojo easier to debug.
- Always **thoroughly** test your code as you program. Don't be tempted to code a lot all at once. Code a bit then test. This will save you a lot of time and make you a productive Smojo programmer!
- Refer to the Appendix on Math Words (below) to see what words are available to you for basic mathematical and logical operations.

Answers to Quizzes

Quiz 1.0

1.0.0: The program runs normally. This is a special case and only works for text within double quotes.

1.0.1: The program runs correctly and shows Hello World. So, it doesn't matter if things are on a different line.

Quiz 1.1

1.1.0: The result is just `ok`. You might have expected `3 ok` to be shown instead. But the output `3` needs to be explicitly displayed using period `.`

1.1.1: `3 ok` is shown. The period `.` is needed to display the output.

Quiz 1.2

1.2.0: The program displays `-4 ok`. Yes, this is expected because the average of 2 and 4 is 3 and the average of 6 and 8 is 7. `3 7 -` is the same as subtracting 7 from 3, which results in -4.

1.2.1: The program shows `-5 ok`. This is unexpected, because the average of 2 and 3 is 2.5 and 2.5 - 7 = -4.5 not -5.

After making the change of `-` to `-.` and `/` to `/.` the resulting code is:

```
: average
  + 2 /.
;

2 3 average
6 8 average
-. .
```

If this new program is run, the result is `-4.5 ok` as expected.

So, there is a big difference between integer arithmetic using `+ - * /` and real arithmetic using `+. -. *.` and `/.` Integer arithmetic will cause intermediate answers like 1.5 to be rounded to become integers (2 in this case)

1.2.2: Here's a solution:

```
: average ( n n – n )
  + 2 /.
;

: diff-average ( n n n n – n )
   average { b }
   average { a }
   a b -.
;

4 3 2 1 diff-average .
```

When this program is run, it will show `2.0 ok`. The program needs some explanation:

- The items in purple eg `( n n – n )` are special comments called **stack comments**. Like ordinary comments, they are also ignored by Smojo, but they are very helpful to us: the `n n` on the left indicates to us that `average` needs 2 inputs (n means number) on the left. the single `n` after the dash `–` means that the output on the right is a single number.
- The items in braces eg `{ b }` and `{ a }` means that we are giving the intermediate results a temporary name. So, `average { a }` means calculate the average of the 2 numbers on the left then name the result `a`. We do that for the second average also calling it `b`.
- The last step is to recall the named results `a` and `b` then subtract them.

Appendix: Math Words

Word	Action
`+ - * / =`	Used to operate between integers only. If a real number is encountered, it is *truncated* -- fractional part is discarded. For example, `1 2 +` will give `3` as expected, but `1 2.8 +` will also give `3`, since `2.8` is truncated. The equal sign is used to test if two integers on the stack are equal.
`+. -. *. /. =.`	As above, but these can operate on real numbers. No truncation is performed.
`> >= < <=`	Inequality tests. These work on both integers and reals. The result is a boolean (`true` or `false`) on the stack. `1 2.2 < .` displays `true ok`
`FLOOR`	Displays the floor of a number, which is the closest integer lower than the given number. For example, `3.5 FLOOR` is 3.0
`CEIL`	Displays the ceiling of a number, which is the closest integer higher than the given number. For example, `4.1 CEIL` is 5.0
`ROUND`	Rounds off to the nearest integer. For example, `4.3 ROUND` is 4 and `4.5 ROUND` is 5
`MIN` `MAX`	Minimum and Maximum of two numbers. For example, `0 3 min` is 0.0 and `4 5.1 max` is 5.1

Word	Action
`^`	Exponentiation. `2 3 ^` becomes `8.0`

`TAN`
`SIN`
`COS`
	Usual trigonometric functions. The inputs are expected to be in **radians**.

`ATAN2`	The special arctangent, very useful for computing directions (in radians) of 2D-vectors. For example, `1 1 atan2 to-deg .` will display 45.0 , meaning a vector (1,1) has angle 45 degrees to the horizontal. `to-deg` is used to convert the answer in radians into degrees.
`#e` `#pi`	Constant values of e and π respectively.
`EXP` `LN`	Natural exponentiation and natural logarithms.
`LOG10`	Logarithms base 10.
`TO-DEG` `TO-RADIANS`	Converts angles to degrees or radians.
`MOD` `FMOD`	Integer modulus and real modulus. Use `FMOD` if either operand is a real number.
`ABS`	The absolute value.
`TRUE` `FALSE`	Puts the logical values `true` and `false` on the stack.
`OR`	Logical OR. For example, `true false OR` is `true`.
`AND`	Logical AND. For example, `true false AND` is `false`.
`NOT`	Logical NOT. For example, `true NOT` is `false`