# Challenge 1

## Building a Code Profiler

The goal of this challenge is to write a cloud module that enables **code profiling**. A code profiler allows users to tell where their code spends the most time, and how many times a word is called.

**NOTE**: Only attempt Challenge 1 after you complete the Quizzes of Session 2.

John wants to profile his code. He wants to know:

(a) how many times one of his words is called,

(b) how much time is spent when in each word when they are used in a program.

**Quiz C.1.0**: What data structure ( sequence / tuple / hash ) is best suited to store this profile information? Defend your answer.

**Quiz C.1.1**: Define a new variable to store this information. Is it a local or global variable? Defend your answer. Let's call this variable `COUNT-INFO`

**Quiz C.1.2**: To accomplish (a), you need to increment a counter each time a word is called. You do this by re-defining `:` (COLON). Some hints:

i) The word `latestXT ( -- xt )` gives the XT of the last word that was defined.

ii) `name? ( xt -- "s" )` gives the name of an XT.

Use these to re-define the colon `:` and increment a counter on `COUNT-INFO`.

**Quiz C.1.3**: Write a word `word-counts ( -- # )` that returns the `COUNT-INFO`. Ensure it returns a copy of `COUNT-INFO` by using the word `clone ( x -- x )`.

**Quiz C.1.4**: Use your answer for `PP` in Session-1 to write a word `.word-counts ( -- )` that nicely prints the contents of `COUNT-INFO`.

**Quiz C.1.5**: The word `now ( -- n )` gives the current time in milliseconds. Use it to write a timer to calculate how long Smojo spends in a word.

i) You need to also re-define the SEMICOLON `;` word. This is tricky as SEMICOLON is an immediate word. You need to use `postpone` and `immediate`.

ii) What kind of data structure will you use to store timer information? Use a global variable called `TIMER-INFO` to store this information. This should be used to store two pieces of information for a word: the **start time** and **total elapsed time**. The start time is set to `NOW` each time a word is called and the elapsed time calculated when the word ends, either through `;` or `EXIT`.

iii) Amend COLON `:` to start the timer. Your amended SEMICOLON `;` should stop the timer and update the `TIMER-INFO`.

iv) Ensure your new SEMICOLON `;` is an immediate word.

**Quiz C.1.6**: Write a word `word-times ( -- # )` to return the `TIMER-INFO`. Also write a word `.WORD-TIMES ( -- )` to print out the times nicely.

**Quiz C.1.7**: Test out your profiler thoroughly! You should be able to print out the counts and times of new words.

**Quiz C.1.8**: Finally, package your profiler using a cloud module:

```
module *profiler
  \ .... Your code goes here ....
end-module
: profiler
    export: *profiler
;.
```

Once you run this code, it will publish your profiler as `username/profiler`, where `username` is your own username.

**Important**: Ensure that your profiler profiles both counts and word times simultaneously.

**Quiz C.1.9**: In practice, we rarely want to profile all words in a program, since this may slow down the program unacceptably, and also because some words might only take a small amount of time and so be rounded down to zero. A better solution is to selectively profile words.

To this end,

(i) create a word `(PROFILE) ( xt -- )` that causes the given XT to become profiled.

(ii) remove the profiling code from ; and :, since all profiling is done using `(PROFILE)`.

(iii) create a word `\PROFILE ( -- )` that profiles the latest XT. Define `\PROFILE` in terms of `(PROFILE)`.

For example:

```
: hello
   "Hello World" . cr
; \profile
```

will cause just `hello` to be profiled, for both counts and elapsed time.