PALS
====

Python Autonomous Lafon specificity Scripts.

What are these scripts?
=======================

These scripts are here to help you compute Lafon's specificity with only the
terminal. They are pure python scripts without any dependency that needs to be
installed and anyone can launch them. These script can be configured to try and
emulate some existing tools that provide ways to compute Lafon specificity,
namely `TXM <https://txm.gitpages.huma-num.fr/textometrie/>`_ and
`itrameur <http://www.tal.univ-paris3.fr/trameur/iTrameur/>`_. The emulation of
these tools are mostly guess work, do not expect to have the exact same results.

Why use these scripts?

There are some uses for these script, including but not limited to:

- no install (except for python on Windows), no dependencies, command line only
- minimal data preprocessing required (one token per line)
- no indexing, this saves time on big input
- command line options for quick and versatile configuration

This is not a drop-in replacement of existing tools. I just made these scripts
to have some way to quickly try stuff and just share them.

License

There is no licence at the moment because I am looking for the right one. See
what this means according to the `GitHub documentation <https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository#choosing-the-right-license>`_

How to launch these scripts?
============================

Input format

The provided scripts do not handle tokenization. You will have to tokenize your
data beforehand. The excpected tokenization format is a minimalistic CoNLL:

- one token per line
- two sentences are separated by at least one blank line

Tool emulation

By default, these scripts will provide their own results that will most likely
differ from off-the-shelf tools. I tried to emulate what they would output, but
do not expect to have the exact same results, there is some guess work which may
be completely wrong.

The tools you may try to emulate are:

- `TXM <https://txm.gitpages.huma-num.fr/textometrie/>`_
- `itrameur <http://www.tal.univ-paris3.fr/trameur/iTrameur/>`_

If you want to have these tools actual results, use them.

partition.py

`partition.py` wants to have a corpus split into parts. There are two ways to
provide the parts that make up a corpus. If you want some help on how the
command works, you can provide the `--help` or `-h` option:

.. code-block:: bash

        python3 ./partition.py -h

The first way is to repeatedly provide data by using the `--input` or `-i`
option. Say you have two files `a.txt` and `b.txt` and you want to compare them
against each other:

.. code-block:: bash

        python3 ./partition.py -i a.txt -i b.txt

You could also compare two folders `f1` and `f2`:

.. code-block:: bash

        python3 ./partition.py -i f1/* -i f2/*

There can also be times you simply want to compare all files at once and just
want to provide a list of files as the partition of your corpus. You can do that
by using the `--inputs` (with an `s`) switch. This switch is incompatible with
`-i` or `--input`:

.. code-block:: bash

        python3 ./partition.py --inputs folder/*

You can also provide an experimental emulation of existing tools like TXM and
itrameur with the option `--tool-emulation` or `-t`:

.. code-block:: bash

        python3 ./partition.py -i f1/* -i f2/* --tool-emulation TXM
        python3 ./partition.py -i f1/* -i f2/* -t itrameur

cooccurrents.py

`cooccurrents.py` takes a corpus that can be multiple files. Since there is no
partitioning required, you simply provide the corpus as multiple arguments
with no options. It also takes a target token to gather its cooccurrents. If you
want some help on how the command works, you can provide the `--help` or `-h`
option:

.. code-block:: bash

        python3 ./cooccurrents.py -h

Since it would be error prone to simply provide the target token as a simple
argument (it could be mistaken with a file of the corpus), the target token has
to be provided with the required option `--target`:

.. code-block:: bash

        python3 ./cooccurrents.py a.txt b.txt --target foo

By default, only tokens that are stricly equal to target will be considered. If
you want to provide your target as a regular expression (for handling declension
for example), you can tell the script to match using regular expressions instead
with the option `--match-mode`.

.. code-block:: bash

        python3 ./cooccurrents.py a.txt b.txt --target "[Ff]oo" --match-mode regex

This regular expression has to match the whole token, be sure to use jokers `.*`
when necessary. For example, the first command will match any token that starts
with `a` and the second one any token that ends with `a` and the third one any
token than contains `a`:

.. code-block:: bash

        python3 ./cooccurrents.py a.txt b.txt --target "^a.*" --match-mode regex
        python3 ./cooccurrents.py a.txt b.txt --target ".*a$" --match-mode regex
        python3 ./cooccurrents.py a.txt b.txt --target ".*a.*" --match-mode regex

You can also provide an experimental emulation of existing tools like TXM and
itrameur with the option `--tool-emulation` or `-t`:

.. code-block:: bash

        python3 ./cooccurrents.py a.txt b.txt --target foo --tool-emulation TXM
        python3 ./cooccurrents.py a.txt b.txt --target foo -t itrameur