When running a complicated pipeline

- might evolve like living beings (working but not optimal solutions are not revisited (:thumbsdown:)
- once need to apply to a project, want it to be wrapt and easy to apply :question: -> enter!

The Almighty Wrapper

```python
def the_almighty_wrapper(dicom_path, **kwargs):
    """ Meant to run a very complicated processing pipeline

    Parameters:
    -----------
    dicom_path: str,
        Path to the folder containing DICOM files of a multi-echo 3D GRE

```

(with `the_path` in the argument)

OK, we give up on the input

- then you apply it to a bit of a different data and it quickly becomes to much of a pain to maintain meaningfull import wrapped, -> so enter!

```python
def the_almighty_wrapper(myobject, **kwargs):
    """ Meant to run a very complicated processing pipeline

    Parameters:
    -----------
    myobject: myclass,
        Your input data preloaded in our custom way. The pipeline used to start
        with just the path to the data, but importing routine turned out
        different so many times, we finally decided to separate I/O
        (just kidding, only input) into a separate function.

```

Figure 1: title-v1

. . . because there has to be something unified for the rest of the pipeline to build on top :question: **IMHO:** much like in Hawkings' someone-elses-words each formula halves the number of readers. . . $E = mc^2$ Each custom class incorporated in the core of the algorithms reduces the number of people contributing to the code (we all love numpy or reimplementing algos from matlab's scratch). Plus, you pay the penalty of calling API of your custom objects in algorithms.

If `myobject`-abstraction leaks. . .

..you want to add a hook!.. :thumbsdown:

```
wrappers.py
        with just the path to the data, but importing routine turned out
        different so many times, we finally decided to separate I/O
        (just kidding, only input) into a separate function.

    # Following are the options to control the workflow and special cases

    echo_time: iterable, float, optional
        (necessary for the processing) If given overwrites the echo time
        specified in the input ``myclass`` instance

```

Figure 2: echoes-1

(to control corrupted object [the world we live in is cruel, some data come inconsistent])

```
wrappers.py
    if echo_time is None:
        # if not given, use the one from the I/O
        if not myobject.echo_time is None:
            echo_time = myobject.echo_time
        else:
            # no good if you are here, let's see if the data came
            # from the other route:
            try:
                echo_time = myobject.header['echo_time']
            except KeyError:
                print('Echo time not present')
                AttributeError('echo_time not set')

```

Figure 3: echoes-2

(But then if you add the hook, it must have priority over the object, so the flow control starts to build up, and in some other dataset ther might be other source of metainformation) **IMHO:** separate preprocessing run to normalize and unify the data