# 1 Example

```
1 checking = Checking(environment=detect_environment())
2 checking.add_steps(
3     compile=CompileCpp('solution.cpp'),
4     run_solution=RunSolution(stdout='out'),
5     diff=Diff(),
6 )
7 status_code, detailed_result = checking.run()
```

We can distinguish three key parts:

- Initialization (line 1)

  The only argument to `Checking` constructor is the environment responsible for running the subsequent tasks. `detect_environment()` is an utility function which automatically detects the environment based on the command line arguments. For more information refer to the Environments section.

- Configuration (lines 2–6)

  This section is responsible for defining the judge pipeline. The usage of keyword over regular arguments is purely optional, designed to provide an option to specify custom step identifiers.

- Launching the checker (line 7)

  `run()` method runs the configured pipeline, and returns a 2-tuple, consisting of the final judge verdict code, and a detailed log of the steps ran.

# 2 Environments

There are two environments, `LocalComputer` and `KolejkaObserver`. The first one is supposed to provide a minimal support for running the judge without installing and launching kolejka-observer. It doesn't support any limits, and should be used solely for debugging purposes. `KolejkaObserver` should be used in the real situations instead. The utility function `detect_environment()` can be used to automatically select the environment, based on the command line arguments.

## ExecutionEnvironment

This is the base class, defining common methods for the environments.

`set_limits(self, **kwargs)`

Filters out the unrecognized limits passed as the arguments, based on the `self.recognized_limits` variable. Warning about the unknown limits is then printed on the `stderr`. Finally the identified limits are saved to be used during the following `run_command()` calls, until the next `set_limits()` invocation.

`run_command(self, command, stdin, stdout, stderr, env)`

Abstract method. Responsible for running the specified command within the appropriate launch configuration, consisting of standard input/output files (handles opened from `stdin`, `stdout`, `stderr` arguments, all of type `pathlib.Path`), environment variables (`env`), and limits (from previous `set_limits()` call). Returns the execution statistics object, details of which are left up to the particular environment - some attributes are, however, required for all of the environments (see TODO `CompletedProcess`).

`run_step(self, step, name)`

Responsible for running the step, which consists of the following parts:

- verifying that step is configured correctly
- verifying the prerequisites are met
- setting the limits requested by the step (see `CommandBase.get_limits()`)
- evaluating the `DependentExpr` expressions
- calling the `run_command()` method
- checking the postconditions
- restoring the old limits

### LocalComputer

```
recognized_limits = []
```

```
run_command(self, command, stdin, stdout, stderr, env)
```

Runs the command using the `/usr/bin/time` tool to measure the time and memory used. Returns the TODO `LocalComputer.LocalStats` object containing execution statistics.

### KolejkaObserver

```
recognized_limits = ['cpus', 'cpus_offset', 'pids', 'memory']
```

### detect_environment()

Returns the environment based on the command line arguments.

`--local` (*default*) – LocalComputer

`--kolejka` – KolejkaObserver

## 3 Steps