1 Running `ghci` locally

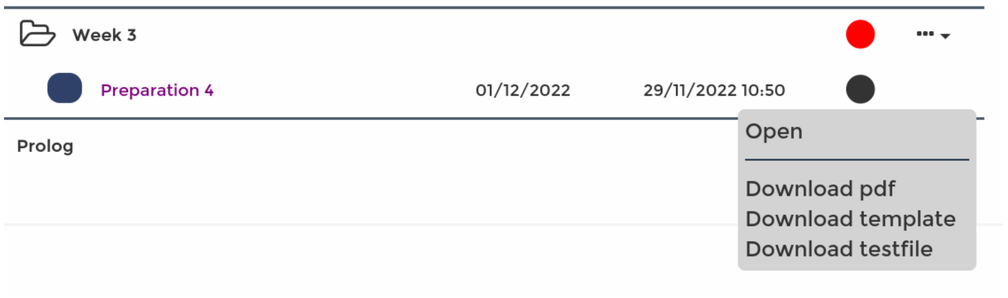
Haskell's default compiler (the Glasgow Haskell Compiler, or GHC) features a read–evaluate–print loop (REPL) command-line tool called `ghci`. While it is possible to develop IO functions in E-Systant, the experience using `ghci` directly may be better since it allows for interaction. Therefore, we recommend you download the files for this exercise, and run it locally.

1.1 Getting GHC and `ghci`

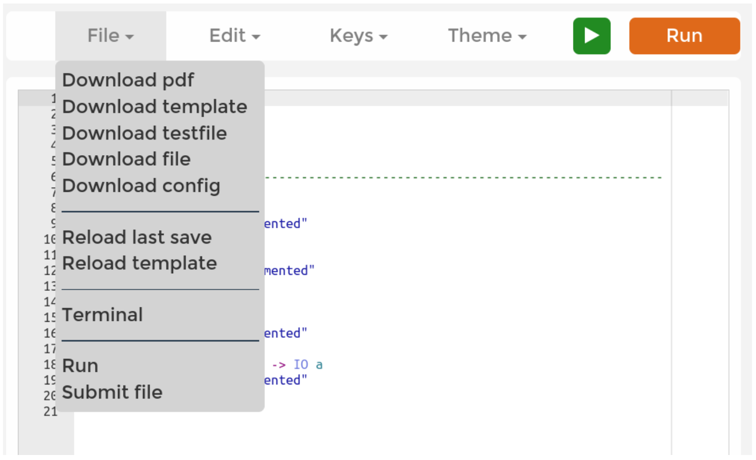
GHC and `ghci` are already installed on the lab computers. For your personal computer it is recommended to use GHCup: `https://www.haskell.org/ghcup/`. For Windows users, this uses WSL2. While not recommended, it may be possible to install GHC natively on windows, see `https://www.haskell.org/downloads/`.

1.2 Downloading the files

We recommend downloading the files from the assignments menu.



It is also possible to download them from the editor itself. However, due to a bug in E-Systant, downloads from this menu may be given the wrong extension. For example, a `.hs` file might be given a `.pdf` extension. Changing the extension allows you to use these files.



1.3 Loading the template

To load a file in `ghci`, execute `ghci Template.hs` in a terminal. If you later change the file, you can execute `:r` to reload the file.

1.4 Running the tests

The testfiles depend on another file called `Testing.hs`, you can download this file from Toledo!

For running the test, you could load up the testfile in `ghci` and execute `main`. However, GHC also comes with the `runhaskell` command, which executes the `main` function of a given `.hs` file. This works well for executing the tests: from outside of `ghci`, call `runhaskell TemplateTest.hs`.

1.5 Prettyprinting the testresults

By default the tests output JSON that E-Systant can read. In the bottom of the file containing the tests you can uncomment a different main function which prettyprints the results.

2 Guessing game

Exercise 1 : Write a number guessing game. The user thinks of a number and the game guesses it in a number of attempts. If the input is unrecognized, print the message like below and stop guessing.

```
Main> game
Think of a number between 1 and 100!
Is it 50? higher
Is it 75? lower
Is it 62? lower
Is it 56? yes
Great, I won!
```

```
Main> game j
Think of a number between 1 and 100!
Is it 50? higher
Is it 75? foo
Unrecognized input, please start over.
```

Exercise 2 : Invert the game: the program thinks of a number between 1 and 100, and the user guesses it. Write `game2 :: Int -> IO ()` that takes the number the user must guess, and lets the user guess.

```
Main> game2 75
What is your guess? 42
higher
...
What is your guess? 75
Congratulations, you have finally, after many attempts, guessed my number.
```