sheetwhat Documentation

Release 0.5.3

DataCamp

Oct 15, 2019

Glossary

For an introduction to SCTs and how to use sheetwhat, visit the README.

This documentation features:

- A glossary with typical use-cases and corresponding SCT constructs.
- Reference documentation of all actively maintained sheetwhat functions.
- Some articles that gradually expose of sheetwhat's functionality and best practices.

If you are new to writing SCTs for Spreadsheats exercises, start with the tutorial. The glossary is good to get a quick overview of how all functions play together after you have a basic understanding. The reference docs become useful when you grasp all concepts and want to look up details on how to call certain functions and specify custom feedback messages.

CHAPTER 1

Glossary

This article lists some example solutions. For each of these solutions, an SCT is included, as well as some example student submissions that would pass and fail. In all of these, a submission that is identical to the solution will evidently pass.

Note: These SCT examples are not golden bullets that are perfect for your situation. Depending on the exercise, you may want to focus on certain parts of a statement, or be more accepting for different alternative answers.

All these examples come from live Spreadsheets courses. You can have a look at the GitHub source of these courses as well.

CHAPTER 2

Checks

`has_code`(*state*, *pattern*, *fixed=False*, *incorrect_msg=None*, *normalize=<function <lambda>>*)

`check_function`(*state*, *name*, *index=0*, *missing_msg=None*)

`check_operator`(*state*, *operator*, *missing_msg=None*)

`has_equal_value`(*state*, *incorrect_msg=None*, *ndigits=4*)

`has_equal_formula`(*state*, *incorrect_msg=None*, *ndigits=4*)

CHAPTER 3

Logic

classmethod Ex.**__call__**(*state=None*)

Returns the current code state as a Chain instance.

This allows SCTs to be run without including their 1st argument, `state`.

When writing SCTs on DataCamp, no State argument to `Ex` is necessary. The exercise State is built for you.

Parameters **state** – a State instance, which contains the student/solution code and results.

Example code

```
# How to write SCT on DataCamp.com
Ex().has_code(text="SELECT id")

# Experiment locally - chain off of Ex(), created from state
state = SomeStateProducingFunction()
Ex(state).has_code(text="SELECT id")

# Experiment locally - no Ex(), create and pass state explicitly
state = SomeStateProducingFunction()
has_code(state, text="SELECT id")
```

multi(*state*, **tests*)

Run multiple subtests. Return original state (for chaining).

This function could be thought as an AND statement, since all tests it runs must pass

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- **tests** – one or more sub-SCTs to run.

Example The SCT below checks two has_code cases..

```
Ex().multi(has_code('SELECT'), has_code('WHERE'))
```

The SCT below uses `multi` to 'branch out' to check that the SELECT statement has both a WHERE and LIMIT clause..

```
Ex().check_node('SelectStmt', 0).multi(
    check_edge('where_clause'),
    check_edge('limit_clause')
)
```

check_not (*state*, **tests*, *msg*)

Run multiple subtests that should fail. If all subtests fail, returns original state (for chaining)

- This function is currently only tested in working with `has_code()` in the subtests.
- This function can be thought as a `NOT(x OR y OR ...)` statement, since all tests it runs must fail
- This function can be considered a direct counterpart of multi.

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- ***tests** – one or more sub-SCTs to run
- **msg** – feedback message that is shown in case not all tests specified in `*tests` fail.

Example Thh SCT below runs two has_code cases..

```
Ex().check_not(
    has_code('INNER'),
    has_code('OUTER'),
    incorrect_msg="Don't use `INNER` or `OUTER`!"
)
```

If students use `INNER (JOIN)` or `OUTER (JOIN)` in their code, this test will fail.

check_or (*state*, **tests*)

Test whether at least one SCT passes.

If all of the tests fail, the feedback of the first test will be presented to the student.

Parameters

- **state** – State instance describing student and solution code, can be omitted if used with Ex()
- **tests** – one or more sub-SCTs to run

Example The SCT below tests that the student typed either 'SELECT' or 'WHERE' (or both)..

```
Ex().check_or(
    has_code('SELECT'),
    has_code('WHERE')
)
```

The SCT below checks that a SELECT statement has at least a WHERE c or LIMIT clause..

```
Ex().check_node('SelectStmt', 0).check_or(
    check_edge('where_clause'),
    check_edge('limit_clause')
)
```

check_correct (*state*, *check*, *diagnose*)

Allows feedback from a diagnostic SCT, only if a check SCT fails.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **check** – An sct chain that must succeed.
- **diagnose** – An sct chain to run if the check fails.

Example The SCT below tests whether students query result is correct, before running diagnostic SCTs..

```
Ex().check_correct(
    check_result(),
    check_node('SelectStmt')
)
```

disable_highlighting (*state*)

Disable highlighting in the remainder of the SCT chain.

Include this function if you want to avoid that pythonwhat marks which part of the student submission is incorrect.

fail (*state*, *msg='fail'*)

Always fails the SCT, with an optional msg.

This function takes a single argument, `msg`, that is the feedback given to the student. Note that this would be a terrible idea for grading submissions, but may be useful while writing SCTs. For example, failing a test will highlight the code as if the previous test/check had failed.

CHAPTER 4

Electives

has_chosen (*state*, *correct*, *msgs*)

Verify exercises of the type MultipleChoiceExercise

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **correct** – index of correct option, where 1 is the first option.
- **msgs** – list of feedback messages corresponding to each option.

Example The following SCT is for a multiple choice exercise with 2 options, the first of which is correct.:

```
Ex().has_chosen(1, ['Correct!', 'Incorrect. Try again!'])
```

success_msg (*state*, *msg*)

Changes the success message to display if submission passes.

Parameters

- **state** – State instance describing student and solution code. Can be omitted if used with Ex().
- **msg** – feedback message if student and solution ASTs don't match

Example The following SCT changes the success message:

```
Ex().success_msg("You did it!")
```

CHAPTER 5

Tutorial

sheetwhat uses the `.` to 'chain together' SCT functions. Every chain starts with the `Ex()` function call, which holds the exercise state. This exercise state contains all the information that is required to check if an exercise is correct, which are:

- the student's spreadsheet (both the formulas and the resulting values),
- the solution's spreadsheet (both the formulas and the resulting values),
- the range for which the SCT is specified.

As SCT functions are chained together with `.`, the `Ex()` exercise state is copied and adapted into 'sub states' to zoom in on particular parts of the state. Before this theory blows your brains out, some examples will be included in this tutorial soon.

For details, questions and suggestions, contact us.

Python Module Index

Index