

PythonTools

A collection of tools for Python that I created for my personal use.

GitHub repo: <https://github.com/KasparJohannesSchneider/PythonTools>

Table of Contents

1. File Structure	1
2. Import	2
3. Debug Tools	2
3.1. Debug Wrapper <code>@debug</code>	2
3.2. Timer Wrapper <code>@timer</code>	2
3.3. Run Function and get STDOUT <code>run_fct_get_stdout(fct: callable, *args) → str:</code>	3
4. Math Tools	3
4.1. Sum from 1 to n <code>sum_1_n(n: int) → int</code>	3
4.2. Lower Triangular Number <code>ltm(n: int) → int</code>	4
4.3. Is Triangular? <code>is_triangular(n: int) → bool</code>	4
5. Web Tools	4
5.1. Is Page up <code>is_page_up(url: str) → bool</code>	4

1. File Structure

```
| .gitignore
| LICENSE
| README.adoc
| README.pdf
|
|-----|.github                                # GitHub CI
|         |-----|workflows
|         |         |dependencies.txt          # Dependencies for unit tests
|         |         |doc.yml                  # Convert README to PDF
|         |         |test.yml                 # Run unit tests and coverage
|
|-----|python_tools                          # Python package
|         |debug_tools.py
|         |math_tools.py
|         |__init__.py
|
|-----|test                                # Package containing unit tests
|         |test_debug_tools.py
|         |test_math_tools.py
|         |__init__.py
```

2. Import

```
import python_tools as pt
```

3. Debug Tools

Tools that can be used for debugging.

3.1. Debug Wrapper `@debug`

A debug wrapper that prints some useful information about a function call.

3.1.1. How to use

This wrapper can be used by placing the corresponding decorator above the declaration of the function.

```
import python_tools as pt

@pt.debug
def a_function(x, y, z):
    pass
```

3.1.2. Expected output

```
--debug--debug--debug--debug--debug--debug--debug--debug--debug--debug--
-- Function: a_function(x, y, z)
-- Arguments: (1, 2, 3)
-- Returned: None
-- Time elapsed [s]: 0.0
--debug--debug--debug--debug--debug--debug--debug--debug--debug--
```

3.2. Timer Wrapper `@timer`

3.2.1. How to use

This wrapper can be used by placing the corresponding decorator above the declaration of the function.

```
import python_tools as pt

@pt.timer
def a_function(x, y, z):
    pass
```

3.2.2. Expected output

```
--timer--timer--timer--timer--timer--timer--timer--timer--timer--timer--
-- Function: a_function(x, y, z)
-- Time elapsed [s]: 0.0
--timer--timer--timer--timer--timer--timer--timer--timer--timer--timer--
```

3.3. Run Function and get STDOUT

run_fct_get_stdout(fct: callable, *args) → str:

Runs a function and collects stdout during the execution of said function and returns the collected stdout as a string.

3.3.1. How to use

```
>>> import python_tools as pt

>>> pt.run_fct_get_stdout(print, 'Hello world!')
'Hello world!\n'
```

4. Math Tools

Tools related to mathematics

4.1. Sum from 1 to n **sum_1_n(n: int) → int**

This function calculates the sum of all numbers from 1 to *n*.

[Wikipedia](#): $1 + 2 + 3 + 4 + \dots$

4.1.1. Example

```
>>> import python_tools as pt

>>> pt.sum_1_n(10)
55
```

4.2. Lower Triangular Number `ltm(n: int) → int`

This function returns `n` if it is a triangular number, or the next lower triangular number.

[Wikipedia: Triangular number](#)

4.2.1. Example

```
>>> import python_tools as pt

>>> pt.ltm(16)
15
```

4.3. Is Triangular? `is_triangular(n: int) → bool`

This function checks if a number is triangular.

[Wikipedia: Triangular number](#)

4.3.1. Example

```
>>> import python_tools as pt

>>> pt.is_triangular(15)
True
>>> pt.is_triangular(16)
False
>>> pt.is_triangular(21)
True
```

5. Web Tools

Tools related to the internet and webpages.

5.1. Is Page up `is_page_up(url: str) → bool`

Tests if a webpage is up (returns 200).

5.1.1. Example

```
>>> import python_tools as pt

# Test an existing url
>>> pt.is_page_up('https://www.twitter.com/')
True

# Test an url that doesn't exist
>>> pt.is_page_up('https://www.a1s2d3e5f2c5e4d2f5r1e23c5e1.com/')
False
```