Logging and Debugging

Collections of tools for logging and debugging Python code.

rich.inspect: Produce a Beautiful Report on any Python Object

```
!pip install rich
```

If you want to quickly see which attributes and methods of a Python object are available, use rich's `inspect` method.

rich's `inspect` method allows you to create a beautiful report for any Python object, including a string.

```
from rich import inspect

print(inspect('hello', methods=True))
```

```
──────────────────────────────────────── <class 'str'> ─────────────────────────────────────────
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

 ╭──────────────────────────────────────────────────────────────────────────────────────────────╮
 │ 'hello'                                                                                      │
 ╰──────────────────────────────────────────────────────────────────────────────────────────────╯

  capitalize = def capitalize(): Return a capitalized version of
    casefold = def casefold(): Return a version of the string sui
               comparisons.
      center = def center(width, fillchar=' ', /): Return a cente
               width.
       count = def count(...) S.count(sub[, start[, end]]) -> int
      encode = def encode(encoding='utf-8', errors='strict'): Enc
               codec registered for encoding.
    endswith = def endswith(...) S.endswith(suffix[, start[, end]
```

```
expandtabs = def expandtabs(tabsize=8): Return a copy where all
             expanded using spaces.
      find = def find(...) S.find(sub[, start[, end]]) -> int
    format = def format(...) S.format(*args, **kwargs) -> str
format_map = def format_map(...) S.format_map(mapping) -> str
     index = def index(...) S.index(sub[, start[, end]]) -> int
   isalnum = def isalnum(): Return True if the string is an alp
             otherwise.
   isalpha = def isalpha(): Return True if the string is an alp
             otherwise.
   isascii = def isascii(): Return True if all characters in th
             False otherwise.
 isdecimal = def isdecimal(): Return True if the string is a de
             otherwise.
   isdigit = def isdigit(): Return True if the string is a digi
             otherwise.
isidentifier = def isidentifier(): Return True if the string is a

             identifier, False otherwise.
   islower = def islower(): Return True if the string is a lowe
             otherwise.
 isnumeric = def isnumeric(): Return True if the string is a nu
             otherwise.
isprintable = def isprintable(): Return True if the string is pr
             otherwise.
   isspace = def isspace(): Return True if the string is a whit
             otherwise.
   istitle = def istitle(): Return True if the string is a titl
             otherwise.
   isupper = def isupper(): Return True if the string is an upp
             otherwise.
      join = def join(iterable, /): Concatenate any number of s
     ljust = def ljust(width, fillchar=' ', /): Return a left-j
             length width.
     lower = def lower(): Return a copy of the string converted
     lstrip = def lstrip(chars=None, /): Return a copy of the st
             whitespace removed.
 maketrans = def maketrans(...) Return a translation table usab
 partition = def partition(sep, /): Partition the string into t
             given separator.
   replace = def replace(old, new, count=-1, /): Return a copy
             substring old replaced by new.
     rfind = def rfind(...) S.rfind(sub[, start[, end]]) -> int
    rindex = def rindex(...) S.rindex(sub[, start[, end]]) -> i
     rjust = def rjust(width, fillchar=' ', /): Return a right-
             length width.
 rpartition = def rpartition(sep, /): Partition the string into
             given separator.
    rsplit = def rsplit(sep=None, maxsplit=-1): Return a list o
             string, using sep as the delimiter string.
    rstrip = def rstrip(chars=None, /): Return a copy of the st
             whitespace removed.
     split = def split(sep=None, maxsplit=-1): Return a list of
             string, using sep as the delimiter string.
splitlines = def splitlines(keepends=False): Return a list of t
             breaking at line boundaries.
```

```
                breaking at the boundaries.
 startswith = def startswith(...) S.startswith(prefix[, start[,
      strip = def strip(chars=None, /): Return a copy of the str
                trailing whitespace removed.
   swapcase = def swapcase(): Convert uppercase characters to lo
                characters to uppercase.
      title = def title(): Return a version of the string where
  translate = def translate(table, /): Replace each character in
                given translation table.
      upper = def upper(): Return a copy of the string converted
      zfill = def zfill(width, /): Pad a numeric string with zer
                a field of the given width.
```

Rich's Console: Debug your Python Function in One Line of Code

```
!pip install rich
```

Sometimes, you might want to know which elements in the function created a certain output. Instead of printing every variable in the function, you can simply use Rich's `Console` object to print both the output and all the variables in the function.

```
from rich import console
from rich.console import Console
import pandas as pd

console = Console()

data = pd.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6]})

def edit_data(data):
    var_1 = 45
    var_2 = 30
    var_3 = var_1 + var_2
    data['a'] = [var_1, var_2, var_3]
    console.log(data, log_locals=True)

edit_data(data)
```

```
[08:12:24]     a  b
           0  45  4
           1  30  5
           2  75  6
                locals
  data =     a  b
          0  45  4
          1  30  5
          2  75  6
  var_1 = 45
  var_2 = 30
  var_3 = 75
```

Link to my article about rich.

Link to rich.

loguru: Print Readable Traceback in Python

```
!pip install loguru
```

Sometimes, it is difficult to understand the traceback and to know which inputs cause the error. Is there a way that you can print a more readable traceback?

That is when loguru comes in handy. By adding decorator `logger.catch` to a function, loguru logger will print a more readable trackback and save the traceback to a separate file like below

```
from sklearn.metrics import mean_squared_error
import numpy as np
from loguru import logger

logger.add("file_{time}.log", format="{time} {level}
{message}")

@logger.catch
def evaluate_result(y_true: np.array, y_pred: np.array):
    mean_square_err = mean_squared_error(y_true, y_pred)
    root_mean_square_err = mean_square_err ** 0.5

y_true = np.array([1, 2, 3])
y_pred = np.array([1.5, 2.2])
evaluate_result(y_true, y_pred)
```

```
> File "/tmp/ipykernel_174022/1865479429.py", line 14, in
<module>
    evaluate_result(y_true, y_pred)
    |               |       └ array([1.5, 2.2])
    |               └ array([1, 2, 3])
    └ <function evaluate_result at 0x7f279588f430>
```

```
  File "/tmp/ipykernel_174022/1865479429.py", line 9, in
evaluate_result
    mean_square_err = mean_squared_error(y_true, y_pred)
                      |                  |       └ array([1.5,
2.2])
                      |                  └ array([1, 2, 3])
                      └ <function mean_squared_error at
0x7f27958bfca0>

  File "/home/khuyen/book/venv/lib/python3.8/site-
packages/sklearn/utils/validation.py", line 63, in inner_f
    return f(*args, **kwargs)
           |  |       └ {}
           |  └ (array([1, 2, 3]), array([1.5, 2.2]))
           └ <function mean_squared_error at 0x7f27958bfb80>
  File "/home/khuyen/book/venv/lib/python3.8/site-
packages/sklearn/metrics/_regression.py", line 335, in
mean_squared_error
    y_type, y_true, y_pred, multioutput = _check_reg_targets(
            |       |                     └ <function
_check_reg_targets at 0x7f27958b7af0>
            |       └ array([1.5, 2.2])
            └ array([1, 2, 3])
  File "/home/khuyen/book/venv/lib/python3.8/site-
packages/sklearn/metrics/_regression.py", line 88, in
_check_reg_targets
    check_consistent_length(y_true, y_pred)
    |                       |       └ array([1.5, 2.2])
    |                       └ array([1, 2, 3])
    └ <function check_consistent_length at 0x7f279676e040>
  File "/home/khuyen/book/venv/lib/python3.8/site-
packages/sklearn/utils/validation.py", line 319, in
check_consistent_length
    raise ValueError("Found input variables with inconsistent
numbers of"

ValueError: Found input variables with inconsistent numbers of
samples: [3, 2]
```

Link to loguru.

Icrecream: Never use print() to debug again

```
!pip install icecream
```

If you use print or log to debug your code, you might be confused about which line of code creates the output, especially when there are many outputs.

You might insert text to make it less confusing, but it is time-consuming.

```
from icecream import ic

def plus_one(num):
    return num + 1

print('output of plus_on with num = 1:', plus_one(1))
print('output of plus_on with num = 2:', plus_one(2))
```

```
output of plus_on with num = 1: 2
output of plus_on with num = 2: 3
```

Try icecream instead. Icrecream inspects itself and prints both its own arguments and the values of those arguments like below.

```
ic(plus_one(1))
ic(plus_one(2))
```

```
ic| plus_one(1): 2
ic| plus_one(2): 3

3
```

Output:

```
ic| plus_one(1): 2
ic| plus_one(2): 3
```

Link to icecream

Link to my article about icecream

heartrate — Visualize the Execution of a Python Program in Real-Time

```
!pip install heartrate
```

If you want to visualize which lines are executed and how many times they are executed, try heartrate.

You only need to add two lines of code to use heartrate.

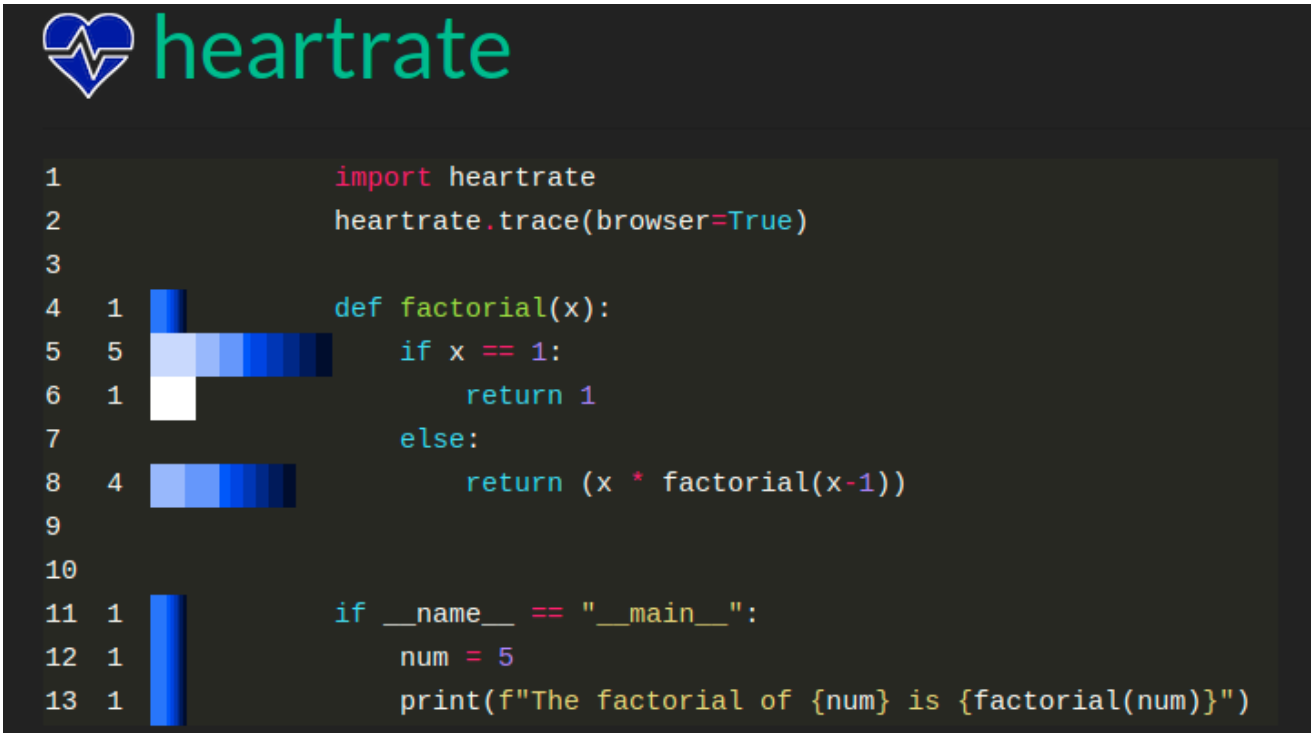
```
import heartrate
heartrate.trace(browser=True)

def factorial(x):
    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


if __name__ == "__main__":
    num = 5
    print(f"The factorial of {num} is {factorial(num)}")
```

```
 * Serving Flask app 'heartrate.core' (lazy loading)
 * Environment: production
•[31m   WARNING: This is a development server. Do not use it
in a production deployment.•[0m
•[2m   Use a production WSGI server instead.•[0m
 * Debug mode: off
The factorial of 5 is 120
Opening in existing browser session.
```

You should see something similar to the below when opening the browser:



Link to heartrate.