Science & Technology Facilities Council | UK Research and Innovation | NERC SCIENCE OF THE ENVIRONMENT

Python

Logging and Debugging

Logging

Python has the "logging" module which allows you to log in many useful ways:

- To the terminal
- To file(s)
- To custom-handlers (e.g. e-mail)
- To system log files

Logging options

- You can configure:
  - The number of loggers
  - The format of log messages
  - The level of ferocity with which logging should happen, e.g.:
    - Log everything in "DEBUG" mode
    - Only log errors in "operational" mode

Logging – getting started

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO)
>>> log = logging.getLogger(__name__)
>>> log.info("The system is running")
INFO:root:The system is running.

>>> log.debug("Nothing said") # Not displayed
          because logging at lower than
          priority level
>>> log.error("Now it's serious!")
ERROR:<module_name>:Now it's serious!
```

Logging – more options

We do not have time to cover logging properly. There are lots you can do with it: https://docs.python.org/3/library/logging.html

```
>>> import sys, logging  # You'll need both these modules
>>> stream_handler = logging.StreamHandler(sys.stderr)
>>> stream_handler.formatter =
  logging.Formatter(logging.BASIC_FORMAT)



>>> log = logging.getLogger(__name__)  # Create logger
>>> log.addHandler(stream_handler)  # Add handler to display
>>> log.setLevel(logging.DEBUG)  # Set minimum logging level

>>> log.warning("Danger! Will Robinson! Danger!")
WARNING:<module_name>:Danger! Will Robinson! Danger!
```

What is the python debugger?

We all write code with bugs in...that is why it is important to write tests for our code.

The python debugger is a tool that allows you to:

- Run through your code interactively;
- Inspect/change the variables at run-time;
- Set "break points" in the code where you can step in and examine the state.

Best illustrated through an example...

A simple script

```
def double_it(x):
    double = 2 * x
    return double


# Now the main code
items = [34, 6.2, {"key": 34}]
for i in items:
    print(double_it(i))
```

Can you guess where python raises an error?

A simple script – with debugger

```
import pdb                          # Import the debugger

def double_it(x):
    pdb.set_trace()                 # Set a break point
    double = 2 * x
    return double

# Now the main code
items = [34, 6.2, {"key": 34}]
for i in items:
    print(double_it(i))
```

Debugger in action

```
$ python double.py
> /home/vagrant/double.py(5)double_it()
-> double = 2 * x
(Pdb) n
```

Run the next line of code

```
> /home/vagrant/double.py(6)double_it()
-> return double
(Pdb) print(double, x)
```

Display current values of double and x

```
68 34

(Pdb) n
```

Run the next line of code

```
--Return--
> /home/vagrant/double.py(6)double_it()->68
-> return double
```

Finding the error

```
...
(Pdb) n
TypeError: unsupported operand type(s) for *:
  'int' and 'dict'
> /home/vagrant/double.py(5)double_it()
-> double = 2 * x
(Pdb) print(x)
{'key': 34}
(Pdb) type(x)
<class 'dict'>
```

Step through until we hit the error

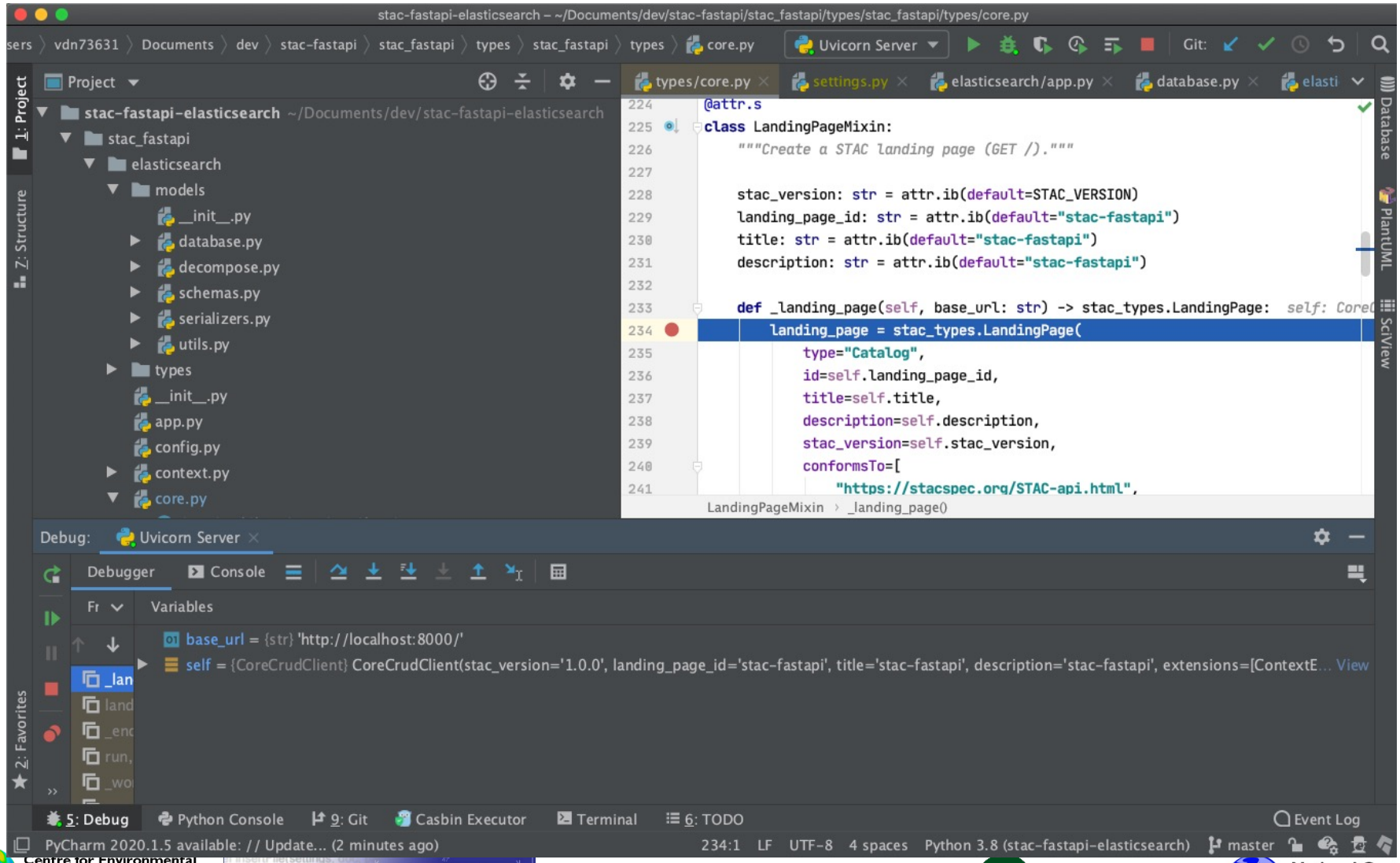
The line where the error occurred

Let's look at x when the error occurred

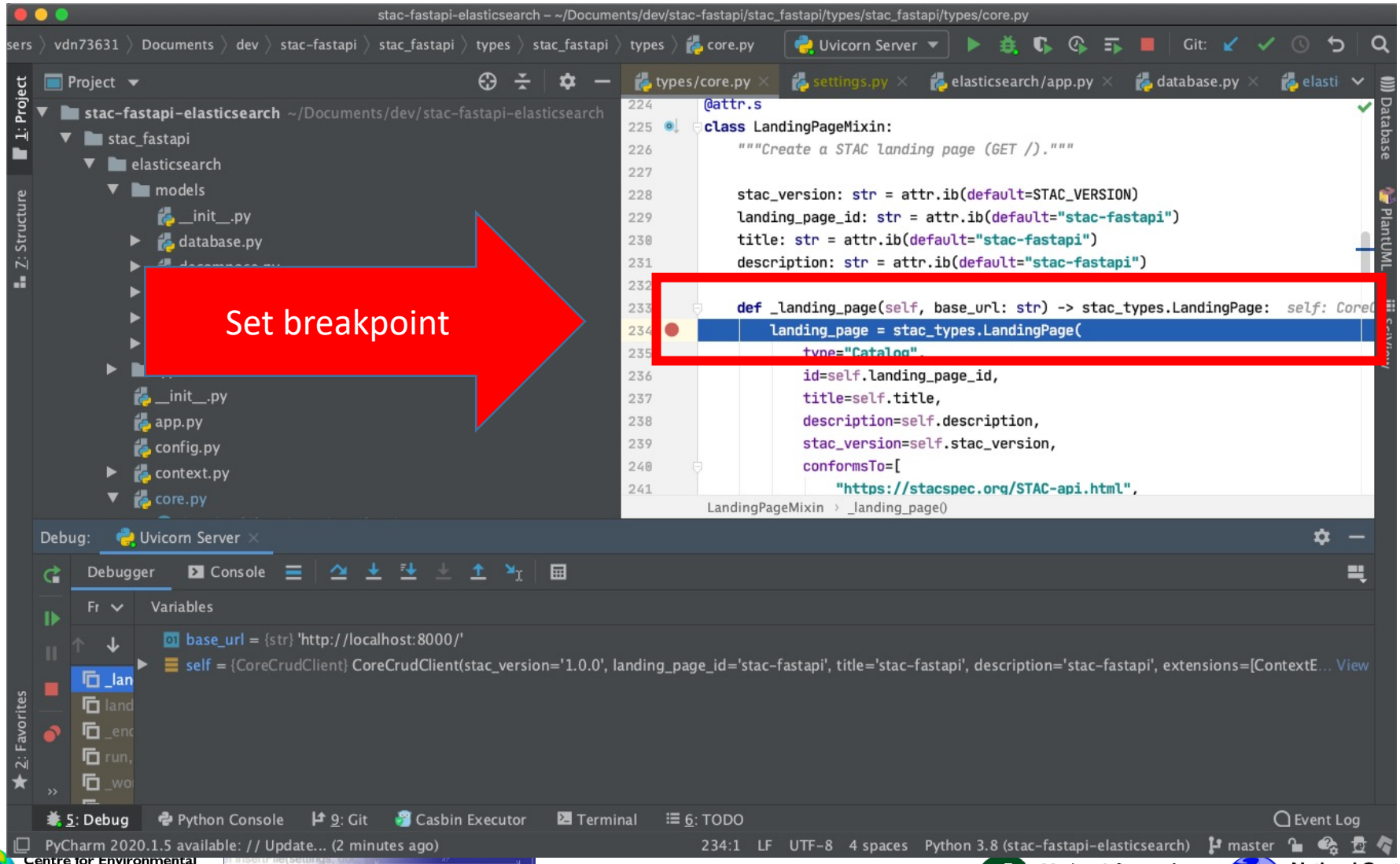
It failed because we can't double a dictionary!

https://docs.python.org/3/library/pdb.html#debugger-commands

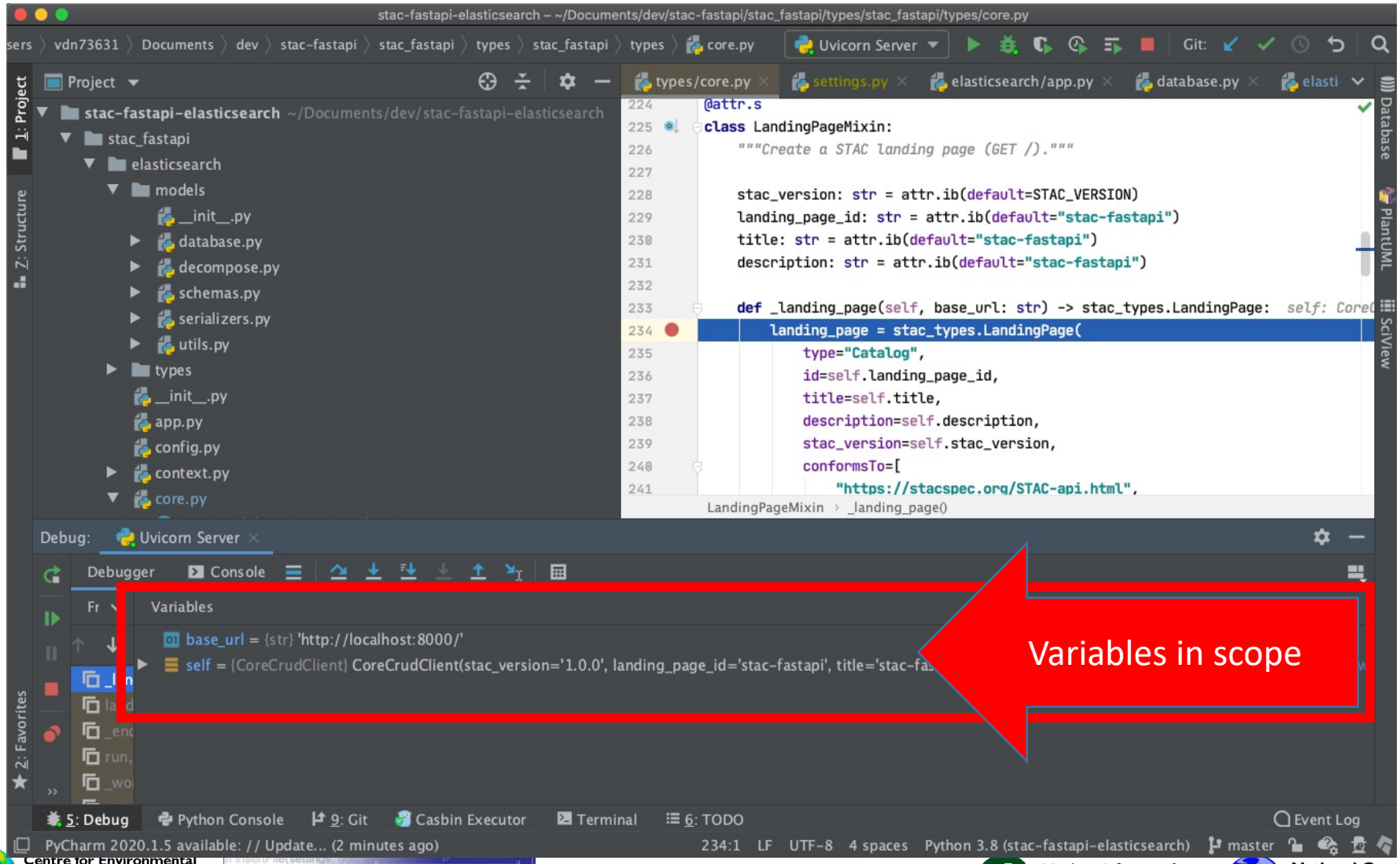
IDEs Wrap this up nicely



IDEs Wrap this up nicely



IDEs Wrap this up nicely



IDEs Wrap this up nicely