The Hidden Power of the Python Runtime

Elizaveta Shashkova
EuroPython 2020
Online
About Me

- Software Developer at JetBrains, PyCharm IDE
- St Petersburg, Russia
- @lisa_shashkova
- Discord: #talk-python-runtime
The hidden power?
The hidden power?

You already use it every day
Test Frameworks

- unittest
- pytest
AssertionError

• unittest

Traceback (most recent call last):
  File "file.py", line 8, in test_value
    assert a == 2, "Wrong value!"
AssertionError: Wrong value!
AssertionError

- unittest

Traceback (most recent call last):
  File "file.py", line 8, in test_value
    assert a == 2, "Wrong value!"
AssertionError: Wrong value!

- pytest

    AssertionError: Wrong value!
    assert 1 == 2
Contents

• Python Runtime
• Getting Runtime Information
• Development Tools
Contents

• Python Runtime

• Getting Runtime Information

• Development Tools
Python Objects

• Created explicitly
  • variables, functions, classes, modules
Python Objects

- Created explicitly
  - variables, functions, classes, modules
- Created implicitly
  - stack frame, code object
Stack Frame

- Represents a program scope
- Information about execution state:
  - Corresponding code object
  - Local and global variables
  - Other data
```python
def foo(n):
    n = n - 1
    return n

def bar(k):
    res = foo(k)
    return res * 2

print(bar(1))
```
```python
def foo(n):
    n = n - 1
    return n

def bar(k):
    res = foo(k)
    return res * 2

print(bar(1))
```
```python
def foo(n):
    n = n - 1
    return n

def bar(k):
    res = foo(k)
    return res * 2

print(bar(1))
```
```python
1  def foo(n):
2    n = n - 1
3    return n
4
5  def bar(k):
6    res = foo(k)
7    return res * 2
8
9  print(bar(1))
```
```python
1  def foo(n):
2      n = n - 1
3      return n
4
5
def bar(k):
6      res = foo(k)
7      return res * 2
8
9
10
11 print(bar(1))
```
Python Power

- Available out of the box
Python Power

- Available out of the box
- Stack frame objects access
Contents

• Python Runtime

• Getting Runtime Information

• Development Tools
Python Stack Frame

- `sys._getframe([depth])`
- `depth` - number of calls below the top
- `0` - current frame
Frame Object: Variables

- Local variables
  - `frame.f_locals`
Frame Object: Variables

- Local variables
  - `frame.f_locals`
- Global variables
  - `frame.f_globals`
Frame Object: Variables

- Local variables
  - `frame.f_locals`
- Global variables
  - `frame.f_globals`
- `locals()` & `globals()`
Code Object

- `frame.f_code` - frame’s attribute
Code Object

• Represents a chunk of executable code
Code Object

- Represents a chunk of executable code

```python
>>> c = compile('a + b', 'a.py', 'eval')
<code object <module> at 0x104f8fc90, file "a.py", line 1>
```
Code Object

- Represents a chunk of executable code

```python
>>> c = compile('a + b', 'a.py', 'eval')
<code object <module> at 0x104f8fc90, file "a.py", line 1>

>>> eval(c, {'a': 1, 'b': 2})
3
```
Code Object

- `code.co_filename` - filename where it was created
- `code.co_name` - name of function or module
- `code.co_varnames` - names of variables
Code Object

- `code.co_filename` - filename where it was created
- `code.co_name` - name of function or module
- `code.co_varnames` - names of variables
- `code.co_code` - compiled bytecode,

  disassemble with `dis.dis()`
Frame Object

- `frame.f_lineno` - current line number
- `frame.f_trace` - tracing function
- `frame.f_back` - previous frame
module

bar()

foo()
Traceback (most recent call last):
  File "file.py", line 12, in <module>
    print(bar(1))
  File "file.py", line 8, in bar
    res = foo(k)
  File "file.py", line 2, in foo
    raise ValueError("Wrong value!")
ValueError: Wrong value!
Frame Object

- `inspect` module
Frame Object

- `inspect` module

- Handle frame variable carefully!

```python
def handle_stackframe_without_leak():
    frame = inspect.currentframe()
    try:
        # do something with the frame
    finally:
        del frame
```
Contents

• Python Runtime

• Getting Runtime Information

• Development Tools
Contents

• Python Runtime

• Getting Runtime Information

• Development Tools
Development Tools

• AssertionError in pytest
Exception Object

- `tb = e.__traceback__` - traceback object
- `tb.tb_frame` - frame object
def vars_in_assert(e):
    tb = e.__traceback__
    frame = tb.tb_frame
    code = frame.f_code
    line = tb.tb_lineno - code.co_firstlineno + 1
    source = inspect.getsource(code)
def get_vars_names(source, line):
    # get variables names with `ast` module

def vars_in_assert(e):
    tb = e.__traceback__
    frame = tb.tb_frame
    code = frame.f_code
    line = tb.tb_lineno - code.co_firstlineno + 1
    source = inspect.getsource(code)
    for name in get_vars_names(source, line):
        pass
def get_vars_names(source, line):
    # get variables names with `ast` module

def vars_in_assert(e):
    tb = e.__traceback__
    frame = tb.tb_frame
    code = frame.f_code
    line = tb.tb_lineno - code.co_firstlineno + 1
    source = inspect.getsource(code)
    for name in get_vars_names(source, line):
        if name in frame.f_locals:
            var = frame.f_locals[name]
            print(f"{name} = {var}")
```python
>>> try:
    assert a + b < 1
except AssertionError as e:
    vars_in_assert(e)
```
```python
>>> try:
    assert a + b < 1
except AssertionError as e:
    vars_in_assert(e)

File "~/file.py", line 10, in foo
    assert a + b < 1

Variables Values:
a = 1
b = 2
```
Development Tools

- AssertionError in pytest
Development Tools

- AssertionError in **pytest**
- Debugger
Debugger

```python
before_set = set()  before_set: <type 'set'>: set([])
after_set = set()  after_set: <type 'set'>: set([])
pad = 4  pad: 4
for dx in xrange(-pad, pad + 1):  dx: -4
  for dy in [0]:  xrange(-pad, pad + 1): dy: 0
  for dz in xrange(-pad, pad + 1): dz: -4
    if dx ** 2 + dy ** 2 + dz ** 2 > (pad + 1) ** 2:
      continue
    if before:
      x, y, z = before
      before_set.add((x + dx, y + dy, z + dz))
```
Python Debugger

• Tracing function

• Frame evaluation function
Tracing Function

- `tracefunc(frame, event, arg)`
- `sys.settrace(tracefunc)` - set to current frame
- Stored in a frame: `frame.f_trace`
- Debugger analyses events
Frame Evaluation

- `frame_eval(frame, exc)`

- Debugger inserts breakpoint’s code into code object
More About Debuggers

• PyCon US 2017

• “Debugging with Python 3.6: Better, Faster, Stronger”
Access to Frame

- `tracefunc(frame, event, arg)`
- `frame_eval(frame, exc)`
Debugger: Location

- `frame.f_code.co_filename` and `frame.f_lineno`
Debugger: Variables

- `frame.f_locals`

```
Variables

01 a = {int} 123
01 answer = {int} 42
1/3 my_list = {list: 3} [1, 2, 3]
01 text = {str} 'Hello World!'
```

Special Variables
Debugger: Frames

- `frame.f_back`

![Frames]

- `MainThread`
- `foo, file.py:3`
- `bar, file.py:11`
- `<module>, file.py:15`
Development Tools

- AssertionError in pytest
- Debugger
Development Tools

- AssertionError in pytest
- Debugger
- Code coverage
Code Coverage

- Shows which lines were executed

```python
def foo(n):
    n = n - 1
    if n > 1:
        print(n)
        return n
    else:
        return n + 1
```
coverage.py

• The most popular code coverage library
coverage.py

- `tracefunc(frame, event, arg)`
- `frame.f_code.co_file` and `frame.f_lineno`
Development Tools

- AssertionError in pytest
- Debugger
- Code coverage
Development Tools

- AssertionError in pytest
- Debugger
- Code coverage
- Runtime typing tools
Typing Tools

- PyAnnotate by Dropbox
- MonkeyType by Instagram
- “Collect Runtime information” in PyCharm
Typing Tools

- PyAnnotate by Dropbox
- MonkeyType by Instagram
- “Collect Runtime information” in PyCharm
Typing Tools

- PyAnnotate by Dropbox
- MonkeyType by Instagram
- “Collect Runtime information” in PyCharm
Typing Tools

• PyAnnotate by Dropbox

• MonkeyType by Instagram

• “Collect Runtime information” in PyCharm
Typing Tools

- PyAnnotate by Dropbox
- MonkeyType by Instagram
- “Collect Runtime information” in PyCharm
Acces To Frame

- PyAnnotate, MonkeyType:
  - `sys.setprofile(profilefunc)`
  - `profilefunc(frame, event, arg)`
Acces To Frame

- “Collect Runtime information” in PyCharm
  - Integrated with Debugger
  - Access to a frame object
def arg_names(co):
    nargs = co.co_argcount
    names = co.co_varnames
    return list(names[:nargs])

names = arg_names(frame.f_code)
def arg_names(co):
    nargs = co.co_argcount
    names = co.co_varnames
    return list(names[:nargs])

names = arg_names(frame.f_code)
locs = frame.f_locals
objects = [locs[n] for n in names]
Typing Tools

• PyAnnotate by Dropbox

• MonkeyType by Instagram

• “Collect Runtime information” in PyCharm
Development Tools

- AssertionError in **pytest**
- Debugger
- Code coverage
- Runtime typing tools
- ?
Concurrent Execution

• Threads
• Async Tasks
import threading

def fun():
    print("Hello!")

t = threading.Thread(target=fun)
t.start()
t.join()
Synchronisation

- Lock - fundamental synchronisation object

```python
lock = threading.Lock()

lock.acquire()
# only one thread here
lock.release()

with lock:
    # equivalent
```
Deadlock

- Waiting for resources which can’t be released
def run1():
    with lock1:
        with lock2:
            # do sth

def run2():
    with lock2:
        with lock1:
            # do sth else

Thread(target=run1).start()
Thread(target=run2).start()
Deadlock

• Waiting for resources which can’t be released

• Hard to detect in big projects
Thread States

- `sys._getframe()`: frame object for current thread
- `sys._current_frames()`: topmost stack frame for each thread
Thread Handler

- Print tracebacks for threads with interval
- Help to find deadlock location
Fault Handler

• `faulthandler.dump_traceback(file)`
  
  Dumps the tracebacks of all threads info file

• Implemented natively
Concurrent Execution

- Threads
- Async Tasks
**Async Locks**

```python
asyncio.Lock()

async with alock:
    # equivalent
```

Example code:

```python
alock = asyncio.Lock()

alock.acquire()
# only one task here
alock.release()

async with alock:
    # equivalent
```
Async Fault Handler

- `asyncio.all_tasks(loop)` - all the running tasks

- `Task.get_stack()` - list of stack frames for this Task
Async Fault Handler

• In a separate thread:

```python
def dump_traceback_later(timeout, loop):
    while True:
        sleep(timeout)
        dump_traceback(loop, timeout)
```
Async Fault Handler

- In a separate thread:

```python
def dump_traceback(loop):
    for task in asyncio.all_tasks(loop):
        task.print_stack()

def dump_traceback_later(timeout, loop):
    while True:
        sleep(timeout)
        dump_traceback(loop, timeout)
```
Development Tools

- AssertionError in pytest
- Debugger
- Code coverage
- Runtime typing tools
- (Async) Fault Handler
The Hidden Power of Runtime

- Python Runtime is very powerful
- Easy access to stack frame and code objects
- Development Tools:
  - pytest, Debugger, Code Coverage, Typing Information, Fault Handler
Inspiration

• Use existing Runtime Development Tools (more often)

• Create something new!
Links

- https://github.com/Elizaveta239/PyRuntimeTricks
- https://elizaveta239.github.io/the-hidden-power-part1/
- elizaveta.shashkova@jetbrains.com
- Discord: #talk-python-runtime