Debugging in Python 3.6: Better, Faster, Stronger

Elizaveta Shashkova
JetBrains

PyCon US 2017

Bio

- **Software developer of PyCharm IDE at JetBrains**
- **Debugger**
- **Saint Petersburg, Russia**

Debugging

- **Adding print statements**
- **Logging**

Debugging

- **Adding print statements**
- **Logging**
- **Special tools: debuggers**

Debugger's Performance

Debuggers are 30 times slower

Contents

- **Tracing debugger**
- **Python 3.6**
- **Frame evaluation debugger**
- **Results**

Contents

- **Tracing debugger**
- **Python 3.6**
- **Frame evaluation debugger**
- **Results**

Tracing Function

`sys.settrace(tracefunc)` - set the system tracing function

```
1 def tracefunc(frame, event, arg):
2     print(frame.f_lineno, event)
3     return tracefunc
4
5
6 sys.settrace(tracefunc)
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
2 line
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
```

Tracing Function

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
5 line
5 return
```

Build Python Debugger

- **Breakpoints**
- **Stepping**

Tracing Debugger

- **Suspend program if breakpoint's line equals `frame.f_lineno`**
- **Handle events for stepping**

Performance

```
1  def foo():
2      friends = ["Bob", "Tom"]
3      for f in friends:
4          print("Hi %s!” % f)
5      return len(friends)
6
7
8  sys.settrace(tracefunc)
9  foo()
```

```
1 call
2 line
3 line
4 line
Hi Bob!
3 line
4 line
Hi Tom!
5 line
5 return
```

Example 1

```
1  def calculate():
2      sum = 0
3      for i in range(10 ** 7):
4          sum += i
5      return sum
6
7
8
9
```

Example 1

```
1  def calculate():
2      sum = 0
3      for i in range(10 ** 7):
4          sum += i
5      return sum
6
7
8  def tracefunc(frame, event, arg):
9      return tracefunc
```

Performance

Performance

Performance

Performance

~ 25 times slower!

Problem

- **Tracing call on every line**

Contents

- **Tracing debugger**
- **Python 3.6**
- **Frame evaluation debugger**
- **Results**

Contents

- Tracing debugger
- **Python 3.6**
- Frame evaluation debugger
- Results

Python 3.6

Python 3.6

- **New frame evaluation API**
- **PEP 523**

PEP 523

- **Handle evaluation of frames**
- **Add a new field to code objects**

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7
8
9
```

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7
8
9
```

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7
8
9
```

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7
8
9
```

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7
8
9
```

Frame Evaluation

```
1  def frame_eval(frame, exc):
2      func_name = frame.f_code.co_name
3      line_number = frame.f_lineno
4      print(line_number, func_name)
5      return _PyEval_EvalFrameDefault(frame, exc)
6
7  def set_frame_eval():
8      state = PyThreadState_Get()
9      state.interp.eval_frame = frame_eval
```

Example

```
1  def first():
2      second()
3
4  def second():
5      third()
6
7  def third():
8      pass
9
10 set_frame_eval()
11 first()
```

Example

```
1  def first():
2      second()
3
4  def second():
5      third()
6
7  def third():
8      pass
9
10 set_frame_eval()
11 first()
```

```
1 first
4 second
7 third
```

Custom Frame Evaluation

- **It works!**
- **Executed while entering a frame**
- **Access to frame and code object**

Contents

- Tracing debugger
- **Python 3.6**
- Frame evaluation debugger
- Results

Problem

- **Tracing call on every line**

Problem

- **Tracing call on every line**
- **Remove the tracing function!**

Replace tracing function with frame evaluation function

Contents

- Tracing debugger
- Python 3.6
- **Frame evaluation debugger**
- Results

Build Python Debugger

- **Breakpoints**
- **Stepping**

Breakpoints

- **Access to the whole code object**

Breakpoints

- **Access to the whole code object**
- **Insert breakpoint's code into frame's code**

Breakpoints

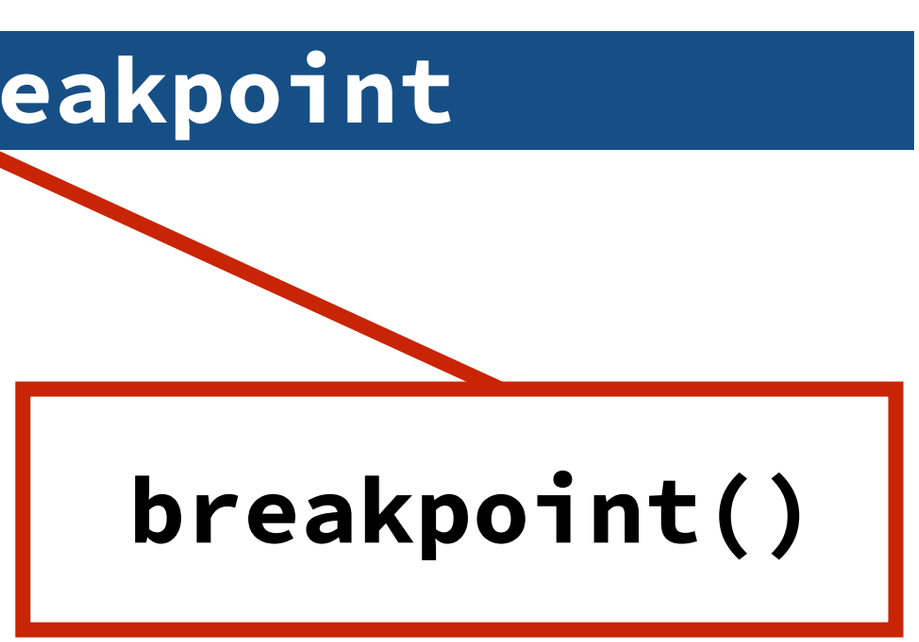
```
1 def maximum(a, b):
2     if a > b:
3         return a
4     else:
5         return b
6
7
8
```

Breakpoints

```
1 def maximum(a, b):
2     if a > b:
3         return a  # breakpoint
4     else:
5         return b
6
7
8
```

Breakpoints

```
1 def maximum(a, b):
2     if a > b:
3         return a  # breakpoint
4     else:
5         return b
6
7
8
```



Breakpoints

```
1 def maximum(a, b):
2     if a > b:
3         breakpoint()
4         return a  # breakpoint
5     else:
6         return b
7
8
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
1  def maximum(a, b):
2      if a > b:
3          return a
4      else:
5          return b
6
7  import dis
8  dis.dis(maximum)
```

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Python Bytecode

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

Bytecode Modification

```
2       0 LOAD_FAST                0 (a)
        2 LOAD_FAST                1 (b)
        4 COMPARE_OP               4 (>)
        6 POP_JUMP_IF_FALSE       12

3       8 LOAD_FAST                0 (a)
       10 RETURN_VALUE

5   >>  12 LOAD_FAST               1 (b)
       14 RETURN_VALUE
       16 LOAD_CONST               0 (None)
       18 RETURN_VALUE
```

```
breakpoint()
```

Bytecode Modification

- **Insert breakpoint's code**
- **Update arguments and offsets**

Bytecode Modification

- **Insert breakpoint's code**
- **Update arguments and offsets**
- **200 lines in Python**

Bytecode Modification

```
2           0 LOAD_FAST                0 (a)
            2 LOAD_FAST                1 (b)
            4 COMPARE_OP               4 (>)
            6 POP_JUMP_IF_FALSE       12

3           8 LOAD_FAST                0 (a)
           10 RETURN_VALUE

5     >>   12 LOAD_FAST                1 (b)
           14 RETURN_VALUE
           16 LOAD_CONST               0 (None)
           18 RETURN_VALUE
```

```
breakpoint()
```

?!

Breakpoint Bytecode

```
1 def _stop_at_break():
2     # a lot of code here
3
4 def breakpoint():
5     _stop_at_break()
6
7
8
```

```
 0 LOAD_GLOBAL    0
 2 CALL_FUNCTION  0
 4 POP_TOP
 6 LOAD_CONST     0
 8 RETURN_VALUE
```

Build Python Debugger

- **Breakpoints**
- **Stepping**

Stepping

- **Inserting temporary breakpoint on every line**
- **Use old tracing function**

Frame evaluation debugger is ready!

Example 1

```
1 def calculate():
2     sum = 0
3     for i in range(10 ** 7):
4         sum += i
5     return sum
```

Example 1

Example 1

Example 2

```
1  def foo():
2      pass
3
4  def calculate():
5      sum = 0
6      for i in range(10 ** 7):
7          foo()
8          sum += i
9      return sum
```

Example 2

PEP 523

- **Handle evaluation of frames**
- **Add a new field to code objects**

PEP 523

- Expand `PyCodeObject` struct
- `co_extra` - “scratch space” for the code object
- Mark frames without breakpoints

Mark Frames

```
1  def frame_eval(frame, exc):
2      flag = _PyCode_GetExtra(frame.f_code, index)
3      if flag == NO_BREAKS_IN_FRAME:
4          return _PyEval_EvalFrameDefault(frame, exc)
5
6      # check for breakpoints
7          ...
8
9
```

Example 2

PEP 523

- **Handle evaluation of frames**
- **Add a new field to code objects**

Contents

- Tracing debugger
- Python 3.6
- **Frame evaluation debugger**
- Results

Contents

- Tracing debugger
- Python 3.6
- Frame evaluation debugger
- **Results**

Real Life Example

Real Life Example

- **Included into PyCharm 2017.1**
- **Works in production**

PyCharm

Tracing w/o Cython

11,59 sec

Tracing with Cython

5,66 sec

Frame evaluation

0,28 sec

Frame evaluation rocks!

Disadvantages

- **More complicated**
- **Only with CPython**
- **Only with Python 3.6**

Frame Evaluation

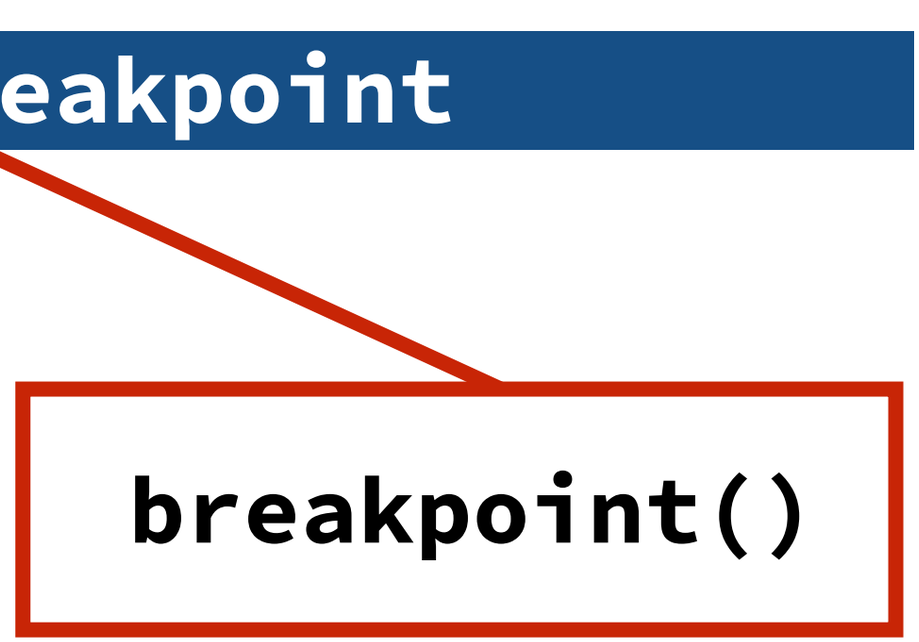
- **Let's move to Python 3.6!**

Frame Evaluation

- **Let's move to Python 3.6!**
- **Let's find another use cases!**

Use cases

```
1 def maximum(a, b):
2     if a > b:
3         return a  # breakpoint
4     else:
5         return b
6
7
8
```



PEP 523

- **Pyjion project**
- **JIT for Python**

Frame Evaluation

- **Let's move to Python 3.6!**
- **Let's find another use cases!**

Links

- **Prototype: https://github.com/Elizaveta239/frame-eval**
- **PyCharm Community Edition source code**

Questions?

- **Prototype: https://github.com/Elizaveta239/frame-eval**
- **PyCharm Community Edition source code**

@lisa_shashkova