

Who's Afraid of the Big Bad ceval Loop?

Me

Moshe Zadka – <https://cobordism.com>

2020

What is not the eval loop?

► Parser

What is not the eval loop?

- ▶ Parser
- ▶ Byte compiler

What is not the eval loop?

- ▶ Parser
- ▶ Byte compiler
- ▶ Object semantics

What is the eval loop?

```
for (;;) {  
    /* ... */  
    /* This is a lie */  
    opcode = (*next_instr) & 0xff;  
    oparg = (*next_instr) >> 8;  
    next_instr++;  
    /* ... */  
    switch (opcode) {  
        /* ... */  
    }  
}
```

On lying

Everybody lies.

<https://cobordism.com>

Code objects

```
>>> (lambda x:None).__code__  
<code object <lambda> at 0x7fb7b67725d0, file "<std
```

Python bytecode

```
>>> (lambda a, b: a + b
... ).__code__.co_code
b'|\x00|\x01\x17\x00S\x00'
>>> dis.dis(_)
```

0	LOAD_FAST	0 (0)
2	LOAD_FAST	1 (1)
4	BINARY_ADD	
6	RETURN_VALUE	

Frame

Where the eval happens

<https://cobordism.com>

Frame contents

Code and context

<https://cobordism.com>

Frame inspection

```
>>> def foo(): bar()
...
>>> def bar():
...     global frm; frm = sys._getframe()
...
>>> frm.f_code.co_name
'bar'
>>> frm.f_back.f_code.co_name
'foo'
>>> frm.f_back.f_back.f_code.co_name
'<module>'
```

Stack machine

```
PyObject **stack_pointer;  
/* ... */  
#define BASIC_PUSH(v)      (*stack_pointer++ = (v))  
#define BASIC_POP()        (*--stack_pointer)
```

Stack operations

```
#define TOP()                (stack_pointer[-1])  
#define PEEK(n)             (stack_pointer[-(n)])  
#define SET_TOP(v)          (stack_pointer[-1] = (v))
```

The Loop arguments

```
PyObject *_PyEval_EvalFrameDefault(  
    PyFrameObject *f,  
    int throwflag)
```

The Loop

```
/* ... */
next_instr = f->f_code->co_code + f->f_lasti;
for (;;) {
    /* ... */
    fast_next_opcode:
        opcode = (*next_instr) & 0xff;
        oparg = (*next_instr) >> 8;
        next_instr++;
        switch (opcode) {
            /* ... */
        }
    }
    error:
```

```
/* ... */
https://cobordism.com
```

The Switch

```
switch (opcode) {  
    /* ... */  
    case TARGET(LOAD_FAST): {  
        PyObject *value = GETLOCAL(oparg);  
        if (value == NULL) {  
            /* ... */  
            goto error;  
        }  
        /* ... */  
        goto fast_next_opcode;  
    }  
    /* ... */  
    case TARGET(UNARY_NEGATIVE): {  
        /* ... */  
        continue;  
    }  
    /* ... */  
}
```

<https://cobordism.com>

Eval Breaker

```
ceval.eval_breaker =  
    ceval.gil_drop_request |  
    ceval.signals_pending |  
    ceval.pending.calls_to_do) |  
    ceval.pending.async_exc;
```

Eval Breaker

```
if (eval_breaker) {
    if (ceval.signals_pending)
        handle_signals(runtime);
    if (ceval.pending_calls_to_do)
        make_pending_calls(runtime);
    if (ceval.gil_drop_request) {
        drop_gil(ceval, tstate);
        /* Other threads may run now */
        take_gil(ceval, tstate);
    }
    if (tstate->async_exc != NULL) {
        _PyErr_SetNone(tstate, tstate->async_exc);
        goto error;
    }
}
```

<https://cobordism.com>

Signal handling

```
void trip_signal(int sig_num) {  
    /* ... */  
    PyRuntimeState *runtime = &_amp;PyRuntime;  
    PyThreadState *tstate = _PyRuntimeState_GetThre  
    runtime->ceval.signals_pending = 1  
    runtime->ceval.eval_breaker = 1  
    /* ... */  
}
```

Global Interpreter Lock

- ▶ Python bytecode evaluation

GIL: Acquisition

```
MUTEX_LOCK( gil->mutex );  
while ( gil.locked )  
    COND_TIMED_WAIT( gil->cond, gil->mutex, interval  
        ceval.gil_drop_request = 1;  
        ceval.eval_breaker = 1;  
    )  
MUTEX_UNLOCK( gil->mutex );  
MUTEX_LOCK( gil->switch_mutex );  
gil.locked = 1  
MUTEX_UNLOCK( gil->switch_mutex );
```

GIL: Release

```
MUTEX_LOCK( gil->mutex );  
gil.locked = 0;  
COND_SIGNAL( gil->cond );  
MUTEX_UNLOCK( gil->mutex );
```

Takeaways

- ▶ Python is not magic

Takeaways

- ▶ Python is not magic
- ▶ ...although it is definitely sufficiently advanced.