

A 1,500 (!! ) line switch statement  
powers your Python!



Allison Kaptur

[github.com/akaptur](https://github.com/akaptur)  
[@akaptur](mailto:akaptur@akaptur)



**Hacker** School

A 1,500 (!! ) line switch statement  
powers your\* Python!



Allison Kaptur

[github.com/akaptur](https://github.com/akaptur)  
[@akaptur](mailto:akaptur@akaptur)



**Hacker** School

Bytecode:  
the internal representation of a python  
program in the interpreter

# Bytecode: it's bytes!

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans
```

# Bytecode: it's bytes!

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod.func_code.co_code
```

Function

Code  
object

Bytecode

# Bytecode: it's bytes!

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod.func_code.co_code  
'\x00\x00|\x01\x00\x16}\x02\x00|\x02\x00S'
```

# Bytecode: it's bytes!

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod.func_code.co_code  
'\x00\x00|\x01\x00\x16}\x02\x00|\x02\x00S'  
>>> [ord(b) for b in mod.func_code.co_code]  
[124, 0, 0, 124, 1, 0, 22, 125, 2, 0, 124, 2, 0, 83]
```

# dis, a bytecode disassembler

```
>>> import dis
```

```
>>> dis.dis(mod)
```

2	0	LOAD_FAST	0 (a)
	3	LOAD_FAST	1 (b)
	6	BINARY_MODULO	
	7	STORE_FAST	2 (ans)
3	10	LOAD_FAST	2 (ans)
	13	RETURN_VALUE	



# dis, a bytecode disassembler

```
>>> import dis
```

```
>>> dis.dis(mod)
```

2	0	LOAD_FAST	0 (a)
	3	LOAD_FAST	1 (b)
	6	BINARY_MODULO	
	7	STORE_FAST	2 (ans)
3	10	LOAD_FAST	2 (ans)
	13	RETURN_VALUE	



Line  
Number



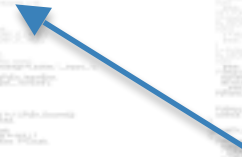
Index in  
bytecode



Instruction  
name, for  
humans



More bytes, the  
argument to each  
instruction

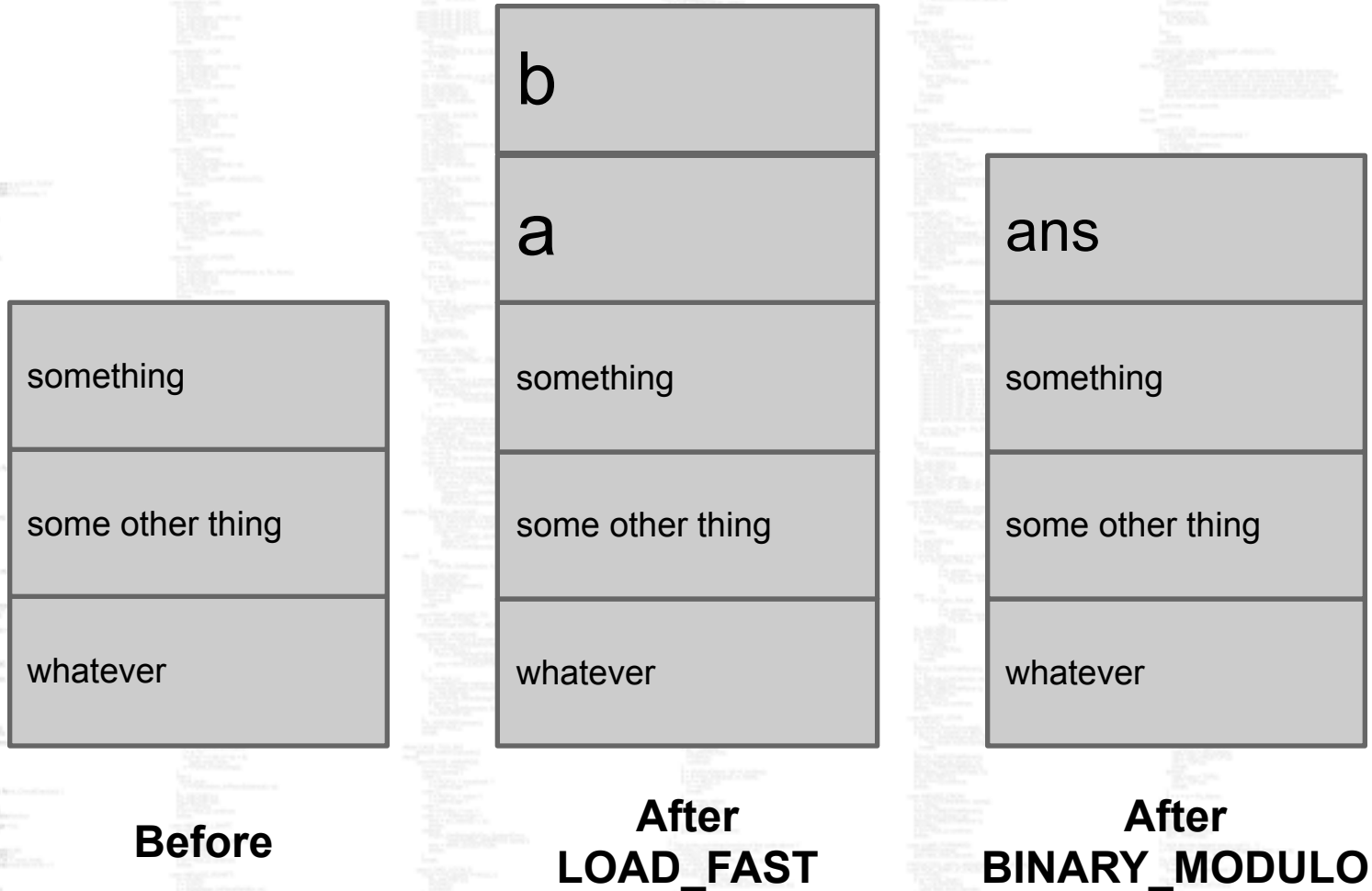


Hint about  
arguments

# The Python virtual machine:

## A bytecode interpreter

# Virtual Machine stack



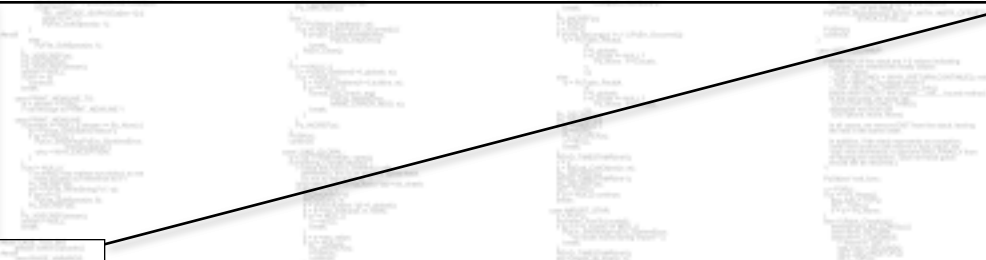
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80																				

```
/* Main switch on opcode */  
READ_TIMESTAMP(inst0);  
  
switch (opcode) {
```

```
} /*switch*/
```

```
/* Turn this on if your compiler chokes on the big switch: */  
/* #define CASE_TOO_BIG 1 */
```

```
#ifdef CASE_TOO_BIG  
    default: switch (opcode) {  
#endif
```



default:

# Back to that bytecode

```
>>> dis.dis(mod)
```

2	0	LOAD_FAST	0 (a)
	3	LOAD_FAST	1 (b)
	6	BINARY_MODULO	
	7	STORE_FAST	2 (ans)
3	10	LOAD_FAST	2 (ans)
	13	RETURN_VALUE	

```
case LOAD_FAST:
    x = GETLOCAL(oparg);
    if (x != NULL) {
        Py_INCREF(x);
        PUSH(x);
        goto fast_next_opcode;
    }
    format_exc_check_arg(PyExc_UnboundLocalError,
        UNBOUNDLOCAL_ERROR_MSG,
        PyTuple_GetItem(co->co_varnames, oparg));
    break;
```



```
case BINARY_MODULO:
```

```
    w = POP();
```

```
    v = TOP();
```

```
    if (PyString_CheckExact(v))
```

```
        x = PyString_Format(v, w);
```

```
    else
```

```
        x = PyNumber_Remainder(v, w);
```

```
    Py_DECREF(v);
```

```
    Py_DECREF(w);
```

```
    SET_TOP(x);
```

```
    if (x != NULL) continue;
```

```
    break;
```

It's “dynamic”

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod(15, 4)  
3
```

# “Dynamic”

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod(15, 4)  
3  
>>> mod("%s%s", ("!!", "Con"))
```

# “Dynamic”

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod(15, 4)  
3  
>>> mod(“%s%s”, (“!!”, “Con”))  
‘!!Con’
```

# “Dynamic”

```
>>> def mod(a, b):  
...     ans = a % b  
...     return ans  
>>> mod(15, 4)  
3  
>>> mod(“%s%s” (“!!”, “Con”))  
‘!!Con’  
>>> print “%s%s” % (“!!”, “Con”)  
!!Con
```

```
case BINARY_MODULO:
```

```
    w = POP();
```

```
    v = TOP();
```

```
    if (PyString_CheckExact(v))
```

```
        x = PyString_Format(v, w);
```

```
    else
```

```
        x = PyNumber_Remainder(v, w);
```

```
    Py_DECREF(v);
```

```
    Py_DECREF(w);
```

```
    SET_TOP(x);
```

```
    if (x != NULL) continue;
```

```
    break;
```

```
>>> class Surprising(object):  
...     def __mod__(self, other):  
...         print "Surprise!"
```

```
>>> s = Surprising()  
>>> t = Surprising()  
>>> s % t  
Surprise!
```

“In the general absence of type information, almost every instruction must be treated as `INVOKE_ARBITRARY_METHOD`.”

- Russell Power and Alex Rubinsteyn, “How Fast Can We Make Interpreted Python?”



# More!

An awesome blog!

<http://tech.blog.aknin.name/category/my-projects/python-innards/>

A Python interpreter in pure Python! (Contribute!)

<https://github.com/nedbat/byterun>

A plug for my blog!

[akaptur.github.io](http://akaptur.github.io)