# Language Interpreters

How does your computer run Python?

# C/C++

C++

```cpp
#include <iostream>
int main()
{
 std::cout << "Hello World!";
}
```

64-bit x86 Linux

```asm
...
.LFB975:
                    .cfi_startproc
                    pushq                   %rbp
                    .cfi_def_cfa_offset 16
                    .cfi_offset 6, -16
                    movq                    %rsp, %rbp
                    .cfi_def_cfa_register 6
                    subq                    $16, %rsp
                    movl                    %edi, -4(%rbp)
                    movl                    %esi, -8(%rbp)
                    cmpl                    $1, -4(%rbp)
                    jne                     .L3
                    cmpl                    $65535, -8(%rbp)
                    jne                     .L3
                    movl                    $_ZStL8__ioinit, %edi
                    call                    _ZNSt8ios_base4InitC1Ev
                    movl                    $__dso_handle, %edx
                    movl                    $_ZStL8__ioinit, %esi
                    movl                    $_ZNSt8ios_base4InitD1Ev, %edi
                    call                    __cxa_atexit
.L3:
                    leave
                    .cfi_def_cfa 7, 8
                    ret
                    .cfi_endproc
...
```

```
g++ hello.cpp -S
```

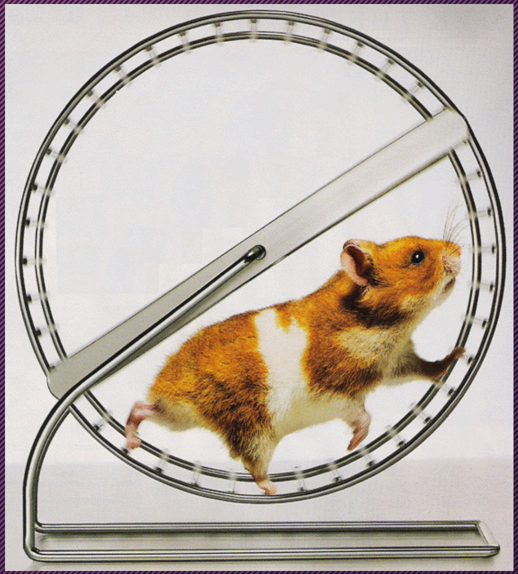
# Python?

Python

```
print “Hello World”
```

?

Something runs



# Interpreters

- No “compilation”
- Then how?
- How can Python run on nearly any computer?
- Python makes its own virtual computer

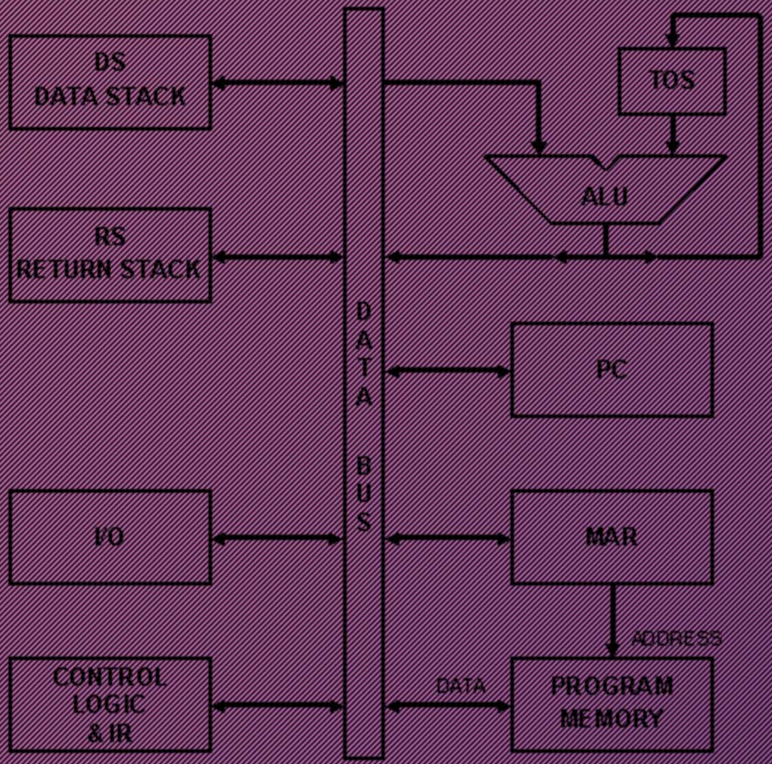
# CPython

- Reference Python Implementation
- Likely the one you have
- Written in C
    - Fun aside—check out PyPy, which is written in Python

# Stack Machine

- Counterpoint to register machine
- Uses stacks a primary memory, not addressable registers

# Diagram



- Similar components
- LIFO Stacks
- Top two elements of data stack are always the inputs to the ALU

# Performance Comparison

$x + y = z$

MIPS Register Machine

```
add $z, $x, $y
```

Pseudo-MIPS Stack Machine

```
push x_addr
push y_addr
add
pop z_addr
```

$x + y * z + u$

Pseudo-MIPS Stack Machine

```
push x
push y
push z
mult
add
push u
add
```

# Function/Subroutine Calls

- Register Machine
  - Save/restore every register
- Stack Machine
  - Just add new stuff to the stack

# ceval.c

- Main interpreter loop lives here
- Increments a pointer to the next instruction
- Giant switch statement

```
switch(opcode) {
           OPCODE1 {
                  /* how to do opcode 1 */
    }
           OPCODE2 {
                  /* how to do opcode 1 */
    }
    ...
}
```

# Binary AND

```
TARGET_NOARG(BINARY_AND)
{
    w = POP();
    v = TOP();
    x = PyNumber_And(v, w);
    Py_DECREF(v);
    Py_DECREF(w);
    SET_TOP(x);
    if (x != NULL) DISPATCH();
    break;
}
```

# Instruction Set

- Python bytecodes
- .pyc files
- Machine agnostic
- Meant only for interpreter

```
import dis
dis.opmap
```

```
{’CALL_FUNCTION’: 131,
 ’DUP_TOP’: 4,
 ’INPLACE_FLOOR_DIVIDE’: 28,
 ’MAP_ADD’: 147,
 ’BINARY_XOR’: 65,
 ’END_FINALLY’: 88,
 ’RETURN_VALUE’: 83,
 ’POP_BLOCK’: 87,
 ’SETUP_LOOP’: 120,
 ’BUILD_SET’: 104,
 ’POP_TOP’: 1,
 ’EXTENDED_ARG’: 145,
 ’SETUP_FINALLY’: 122,
 ’INPLACE_TRUE_DIVIDE’: 29,
 ’CALL_FUNCTION_KW’: 141,
 ’INPLACE_AND’: 77,
 ’SETUP_EXCEPT’: 121,
 ’STORE_NAME’: 90,
 ’IMPORT_NAME’: 108,
 ’LOAD_GLOBAL’: 116,
 ’LOAD_NAME’: 101,
 ’FOR_ITER’: 93,
 ’EXEC_STMT’: 85,
 ’DELETE_NAME’: 91,
 ’BUILD_LIST’: 103,
 ’COMPARE_OP’: 107,
 ’BINARY_OR’: 66,
 ’INPLACE_MULTIPLY’: 57,
 ’STORE_FAST’: 125,
 ’CALL_FUNCTION_VAR’: 140,
 ’SET_ADD’: 146,
 ’LOAD_LOCALS’: 82,
 ’CONTINUE_LOOP’: 119,
 ’PRINT_EXPR’: 70,
 ’DELETE_GLOBAL’: 98,
 ’GET_ITER’: 68,
 ’STOP_CODE’: 0,
 ’UNARY_NOT’: 12,
 ’BINARY_LSHIFT’: 62,
 ’LOAD_CLOSURE’: 135,
 ’IMPORT_STAR’: 84,
 ’INPLACE_OR’: 79,
 ’BINARY_SUBTRACT’: 24,
 ’STORE_MAP’: 54,
 ’INPLACE_ADD’: 55,
 ’INPLACE_LSHIFT’: 75,
 ’INPLACE_MODULO’: 59,
 ’STORE_ATTR’: 95,
 ’BUILD_MAP’: 105,
 ’SETUP_WITH’: 143,
 ’BINARY_DIVIDE’: 21,
 ’INPLACE_RSHIFT’: 76,
 ’PRINT_ITEM_TO’: 73,
 ’UNPACK_SEQUENCE’: 92,
 ’BINARY_MULTIPLY’: 20,
 ’PRINT_NEWLINE_TO’: 74,
 ’NOP’: 9,
 ’LIST_APPEND’: 94,
 ’INPLACE_XOR’: 78,
 ’STORE_GLOBAL’: 97,
 ’INPLACE_SUBTRACT’: 56,
 ’INPLACE_POWER’: 67,
 ’ROT_FOUR’: 5,
 ’DELETE_SUBSCR’: 61,
 ’BINARY_AND’: 64,
 ’BREAK_LOOP’: 80,
 ’MAKE_FUNCTION’: 132,
 ’DELETE_SLICE+1’: 51,
 ’DELETE_SLICE+0’: 50,
 ’DUP_TOPX’: 99,
 ’CALL_FUNCTION_VAR_KW’: 142,
 ’LOAD_ATTR’: 106,
 ’BINARY_TRUE_DIVIDE’: 27,
 ’ROT_TWO’: 2,
 ’IMPORT_FROM’: 109,
 ’DELETE_FAST’: 126,
 ’BINARY_ADD’: 23,
 ’LOAD_CONST’: 100,
 ’STORE_DEREF’: 137,
 ’UNARY_NEGATIVE’: 11,
 ’UNARY_POSITIVE’: 10,
 ’STORE_SUBSCR’: 60,
 ’BUILD_TUPLE’: 102,
 ’BINARY_POWER’: 19,
 ’BUILD_CLASS’: 89,
 ’UNARY_CONVERT’: 13,
 ’BINARY_MODULO’: 22,
 ’DELETE_SLICE+3’: 53,
 ’DELETE_SLICE+2’: 52,
 ’WITH_CLEANUP’: 81,
 ’DELETE_ATTR’: 96,
 ’POP_JUMP_IF_TRUE’: 115,
 ’JUMP_IF_FALSE_OR_POP’: 111,
 ’PRINT_ITEM’: 71,
 ’RAISE_VARARGS’: 130,
 ’SLICE+0’: 30,
 ’SLICE+1’: 31,
 ’SLICE+2’: 32,
 ’SLICE+3’: 33,
 ’POP_JUMP_IF_FALSE’: 114,
 ’LOAD_DEREF’: 136,
 ’LOAD_FAST’: 124,
 ’JUMP_IF_TRUE_OR_POP’: 112,
 ’BINARY_FLOOR_DIVIDE’: 26,
 ’BINARY_RSHIFT’: 63,
 ’BINARY_SUBSCR’: 25,
 ’YIELD_VALUE’: 86,
 ’ROT_THREE’: 3,
 ’STORE_SLICE+0’: 40,
 ’STORE_SLICE+1’: 41,
 ’STORE_SLICE+2’: 42,
 ’STORE_SLICE+3’: 43,
 ’UNARY_INVERT’: 15,
 ’PRINT_NEWLINE’: 72,
 ’INPLACE_DIVIDE’: 58,
 ’BUILD_SLICE’: 133,
 ’JUMP_ABSOLUTE’: 113,
 ’MAKE_CLOSURE’: 134,
 ’JUMP_FORWARD’: 110}
```

# Questions?

# References

[1] "java - Compiled vs. Interpreted Languages - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/3265357/compiled-vs-interpreted-languages. [Accessed: 15-Dec-2015].

[2] "Stack Machine." [Online]. Available: http://www.cp.eng.chula.ac.th/˜piak/teaching/ca/stack.htm. [Accessed: 14-Dec-2015].

[3] "Stack machine - Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Stack_machine. [Accessed: 15-Dec-2015].

[4] Y. Shi, K. Casey, M. A. Ertl, and D. Gregg, "Virtual machine showdown: Stack versus registers," *Acm Trans. Archit. Code Optim.*, vol. 4, no. 4, p. 21, 2007.

[5] "Stack Computers: 6.2 ARCHITECTURAL DIFFERENCES FROM CONVENTIONAL MACHINES." [Online]. Available: https://users.ece.cmu.edu/˜ koopman/stack_computers/sec6_2.html. [Accessed: 14-Dec-2015].

[6] *Python 2.7.11*. Python Software Foundation, 2015.

[7] "python - What is the purpose of Py_DECREF and PY_INCREF? - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/24444667/what-is-the-purpose-of-py-decref-and-py-incref. [Accessed: 15-Dec-2015].

[8] "32.12. dis — Disassembler for Python bytecode — Python 2.7.11 documentation." [Online]. Available: https://docs.python.org/2/library/dis.html. [Accessed: 15-Dec-2015].

http://austincomputerlabs.com/images/custom/stack.jpg