How Python* Works

Built-in data structures
Compiling and Interpreting Python

*CPython implementation; there are others

CPython is a program

CPython is written in C
Compiled into machine code
named “python” or “python3”

Every Python data type is
implemented by a C data
structure in CPython

Caveats ...

Some simplifications
Some guesswork
(there was more guesswork before I found Laurent Luce’s blog: http://www.laurentluce.com/)

Everything is an object

I've been oversimplifying
a little in drawing diagrams
likes this ...

GameBoard

```python
class GameBoard:
    def __init__(self):
        self.rows = 47
        self.cols = 52

x = 332
y
```
**Everything is an object**

I’ve been kind of oversimplifying a little in drawing diagrams ... because reality gets a little messy.

(I’m still omitting some details, and getting others wrong.)

![Diagram of object relationships](image)

**Data structures in CPython**

For every built-in object type in Python ...

*integer, string, dict, set, etc*

... there is a CPython data structure implemented in C
(and compiled to machine language)

**Integer**

Even integers are objects, with methods (!)

\[ x + y \] is actually a method call to

\[ x \_\_add\_\_ (y) \]

(Lots of cute tricks to make this reasonably fast)

\[ x = 5 \]

![Diagram of integer relationship](image)

**Strings in CPython**

\[ x = "foo" \]

![Diagram of string relationship](image)

String value refers to an array of characters, ending with a nul (zero byte), as in the C language.
Lists in CPython

```python
x = [3, 2, 4]
```

References an array longer than the current list length, so that `x.append(7)` will be fast. Reallocates an array when necessary.

Alternative List Data Structures

- Fast insert, delete from either end; slow to find `lis[99]` (like List in Java)
- Fast insert, delete only at end; fast to find or change `lis[99]` (like Vector in Java)

Dictionaries are “hash tables”

- `hash('horse')` == 12
- `hash('goat')` == 29

Pseudo-random but deterministic “scatter storage” based on a “hash function”

Ways to implement dictionaries

- Search tree: Like binary search, but "go left" or "go right" depending on comparison. Database files use a version of this. Complication: keeping the tree "balanced."

- Hash table: Fast on average, but potentially slow in the worst case because of "collisions" (equal hashes). Compilers use this for variable names. Complications: Handling collisions, expanding full tables.
**Python sets are also hash tables**

```
{ 'horse', 'goat', 'zebra' }
```

Hash codes are actually large numbers; position is remainder when divided by table size. The hash table is expanded (copied to a larger table) if it becomes 2/3 full.

Java library equivalents: hashmap, HashSet

---

**The Python Compiler/Interpreter** (simplified)

```
.x = 32
if x > 15 :
y = 32
```

Conventional compiler: Similar to Java, C, C++, etc.

---

**Python vs. C, Java, etc.**

Python compiler generates byte code
- vs: C compiler generates machine code (the interpreter is the computer)
- vs: standard Java compiler generates byte code (also for a stack machine)
- vs: Dalek Java compiler (Android) generates byte code for a virtual register machine

Python values are all objects in the heap
- vs: C values can be in the stack or the heap, untagged
- vs: Java “primitive” values are in the stack and untagged; objects are in the heap and tagged

Compiled? Interpreted? Both!