Fuzz Tests for CPython

These fuzz tests are designed to be included in Google's oss-fuzz project.

oss-fuzz works against a library exposing a function of the form `int LLVMFuzzerTestOneInput(const uint8_t* data, size_t length)`. We provide that library (`fuzzer.c`), and include a `_fuzz` module for testing with some toy values -- no fuzzing occurs in Python's test suite.

oss-fuzz will regularly pull from CPython, discover all the tests in `fuzz_tests.txt`, and run them -- so adding a new test here means it will automatically be run in oss-fuzz, while also being smoke-tested as part of CPython's test suite.

Adding a new fuzz test

Add the test name on a new line in `fuzz_tests.txt`.

In `fuzzer.c`, add a function to be run:

```
int $test_name (const char* data, size_t size) {
    ...
    return 0;
}
```

And invoke it from `LLVMFuzzerTestOneInput`:

```
#if _Py_FUZZ_YES(fuzz_builtin_float)
    rv |= _run_fuzz(data, size, fuzz_builtin_float);
#endif
```

`LLVMFuzzerTestOneInput` will run in oss-fuzz, with each test in `fuzz_tests.txt` run separately.

Seed data (corpus) for the test can be provided in a subfolder called `<test_name>_corpus` such as `fuzz_json_loads_corpus`. A wide variety of good input samples allows the fuzzer to more easily explore a diverse set of paths and provides a better base to find buggy input from.

Dictionaries of tokens (see oss-fuzz documentation for more details) can be placed in the `dictionaries` folder with the name of the test. For example, `dictionaries/fuzz_json_loads.dict` contains JSON tokens to guide the fuzzer.

What makes a good fuzz test

Libraries written in C that might handle untrusted data are worthwhile. The more complex the logic (e.g. parsing), the more likely this is to be a useful fuzz test. See the existing examples for reference, and refer to the oss-fuzz docs.