Code From Nothing
Procedural Generation of Python Source Code

Kirill Borisov
Greetings!

• I’m Kirill Borisov
• 15+ years of programming experience
• Creator of pybetter & BlackConnect
• In love with everything “code”
About this talk

• We will talk about how code is written
• Cover a little bit of parsing
• Introduce Hypothesmith
• Dive deeper into how it works
print("Hello, world!")
Code... What is it, really?

• Code is our bread & butter
• Code is usually written “by hand"
• But who checks it?
• Whole cottage industry of “linters"
file_input:  (NEWLINE | stmt)* ENDMARKER
stmt:      simple_stmt | compound_stmt

simple_stmt:  small_stmt (';' small_stmt)* [';'']
NEWWLINE

small_stmt:  (expr_stmt | print_stmt | del_stmt |
            pass_stmt | flow_stmt |
            import_stmt | global_stmt |
            exec_stmt | assert_stmt)
print("Hello, world!")
Module
  body
    Expr
      value
        Call
          func
          args
            Name
              ctx
              id
                Load
                'print' 'Hello, world!'
            Str
              s
Linters and autoformatters

- They "read" your code
- Code style, security checks, complexity...
- Some can also modify it!
- pep8, pyflakes, black...
Checking the Checkers

• Use hand-crafted examples?
• You need a lot of permutations
• Limited by your imagination
• Real world will surprise you!
Random acts of code

• Random set of characters as input
• Take one which compiles!
• "Infinite monkey theorem", anyone?
• Highly impractical in terms of time
In computing, **procedural generation** is a method of creating data **algorithmically** as opposed to manually, typically through a combination of human-generated assets and algorithms coupled with computer-generated randomness and processing power....

(Source: Wikipedia)

https://en.wikipedia.org/wiki/Procedural_generation
Structure is the king

• *Rules* on how to arrange things
• *Patterns* for generating things
• Need to cover whole of the language
• Sounds like a *grammar*, isn’t it?
Grammar as a template

• It can be represented as a tree
• Rules (*non-terms*) as nodes
• Text (*terms*) as leaves
• Just do random walk through the tree
Enter Hypothesis

https://hypothesis.works/

• Property-based testing
• Generates wide range of input data
• Based on QuickCheck paper
• Can do “hill-climbing search”
```python
def valid_branch_names():
    return st.text(
        alphabet=letters, min_size=1, max_size=112).map(lambda t: t.lower()) | st.just("master")

from hypothesis import assume

given(branch_name=valid_branch_names())
def test_checkout_new_branch(self, branch_name):
    assume(branch_name != "master")
    tmpdir = FilePath(self.mktemp())
    tmpdir.makedirs()
    repo = Repository.initialize(tmpdir.path)
    repo.checkout(branch_name, create=True)
    self.assertEqual(branch_name, repo.get_active_branch())
```

Source: https://hypothesis.works/articles/incremental-property-based-testing/
Lark is a parsing toolkit for Python
• Parses language grammar into a tree
• Select subsets of nodes on each step
• Generates terminals from regexes

https://github.com/lark-parser/lark
Python is quirky

• *Indentation* to mark blocks of code
• Identifiers must be *UTF-8 encodable*
• Lot of AST post-processing
• New PEG parser in Python 3.10
Enter *Hypothesmith*

https://github.com/Zac-HD/hypothesmith

- Inspired by **CSmith**
- Strategy for generating Python code
- Works around mentioned quirks
- Has support for per-node generation
import hypotesmith
from hypothesis import given, settings, HealthCheck

settings.register_profile(
    "slow_example_generation",
    suppress_health_check=HealthCheck.all(),
    deadline=None,
)
settings.load_profile("slow_example_generation")

given(generated_source=hypothesmith.from_grammar())
@settings(max_examples=1000)
def test_no_crashes_on_valid_input(generated_source):
    print(generated_source.encode("utf-8"))
    print("--------")
```python
class A:
    def __init__(self):
        self.a = 0

    for a in A:
        if a:
            with a:
                # 0
```

```python
# 0
```
from ..... import*;global\xc2\xba;nonlocal\xc5\x9df\xcf0\x9e\xb4\xc3\x91\xf0\xa8\xa4\x81Z;\r\n
Targeted search

• Use metrics to find better examples
• Targets:
  - Bytecode instructions
  - Total number of AST nodes
  - Number of unique AST node types
• Longer and more complicated code
Bugs found with Hypothesmith

- **BP0-40661** - Python parser segfault
- **BP0-38953** - `tokenizer` bug
- **lib2to3** errors on `\r` in comment
- **black** fails on files ending in `\`
- Round-trip bugs in **LibCST**
Caveats

• Most generated code is *gibberish*
• It can only serve as a *smoke test*
• No support for AST postprocessing
• Can be quite slow
Further reading

• How Hypothesis Works

• Finding and Understanding Bugs in C Compilers

• QuickCheck – A Lightweight Tool for Random Testing of Haskell Programs

• Compilers: Principles, Techniques and Tools
Questions?
- Thank you!

🐦лечневол

✉️ lensvol@gmail.com