What is Python?
Python is an object-oriented, interpreted, dynamic programming language. It features a powerful and permissive, high-level syntax; and ranks 2nd in popularity on GitHub.

Semantics?
What? A mathematical description of the meaning of Python operators. Why? To relate static analyses with the actual program behavior.

Runtime Errors
```
y = int(input())
try:
x = 1 / y
except DivisionByZeroError:
    print("errors are catchable")
```

Dynamic Typing
```
Variable Scope
a = 2
def f():
    a = 1 + z
f()
# f raises an UnboundLocalError
```

Object Mutability
```
class A:
    def __init__(self):
        self.val = 0
    def update(self, x):
        self.val = x

x = A()
c = x.val
y = 1
y.update('a')
z = x.val
# z = 'a'
```

Goal
Detect potential run-time errors without executing programs.
Automatic analysis: no expert knowledge needed.
Sound analysis: no error found means no runtime error.
We use the Abstract Interpretation framework[1].

Motivation
Static analyses are widespread for statically-typed programming languages, and successfully used in critical software certification.
Dynamic programming languages leave less information in the syntax.
Thus, semantic static analyses would be most valuable in this setting.

Implementation & Benchmarks
Implementation into MOPSA[5], whose goal is to provide modular analyses.
Type analysis: 2500 lines
Container abstraction: 2100 lines
Python’s Semantics: 5500 lines
We are able to analyze some official Python benchmarks[6]!

Future work
Stable, easily maintainable and checkable concrete semantics.
Handle libraries through automatic code generation and multilingual analysis.
We can analyze real-world programs and frameworks (Django, SageMath,...)

Specificities of Python

### Static Type Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>LOC</th>
<th>Time (inlining)</th>
<th>Time (fun. cache)</th>
<th># Alarms</th>
<th>False Alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>fannkuch.py</td>
<td>59</td>
<td>0.07s</td>
<td>0.09s</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>float.py</td>
<td>63</td>
<td>0.10s</td>
<td>0.06s</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>spectral_n.py</td>
<td>74</td>
<td>3.9s</td>
<td>0.33s</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>nbody.py</td>
<td>157</td>
<td>2.6s</td>
<td>1.5s</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>chaos.py</td>
<td>324</td>
<td>19s</td>
<td>5.9s</td>
<td>1[7]</td>
<td>0</td>
</tr>
<tr>
<td>unpack_seq.py</td>
<td>458</td>
<td>5.6s</td>
<td>5.4s</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hexiom.py</td>
<td>674</td>
<td>61.7m</td>
<td>2.2m</td>
<td>0</td>
<td>52</td>
</tr>
</tbody>
</table>

### Semantics Example: Computing $e^1 + e^2$

Python’s efficient and concise syntax entails the semantics to be as accommodating as possible rather than raise exceptions, creating many cases for operators as simple as the addition.

Our current semantics is an input-output semantics which is defined on paper, and whose abstract version is implemented in our static analyzer, updated from [2].

### Guess the result – My favorite Python game

```
[] and 'a'
127 is 127
128 is 128
"a" is "a"
"a",1" is "a,1"
```

1 = list(range(10))
d = {0: 'a'}
for x in 1:
d.pop(1)
print(1)
print(1)

### Semantics Challenges

Uncovering the semantics By reading the documentation and the implementation.
Checking the semantics is correct
- By writing tests and comparing the results with the interpreter;
- By checking that our analysis passes the interpreter’s unit tests.

Other approaches Coq[d], K framework[3] are attractive tools (to extract a concrete interpreter, or to be able to write proofs), but their use would be time-consuming.


The top-right drawing is from XKCD (https://xkcd.com/353/), under licence CC-BY-NC 2.5.