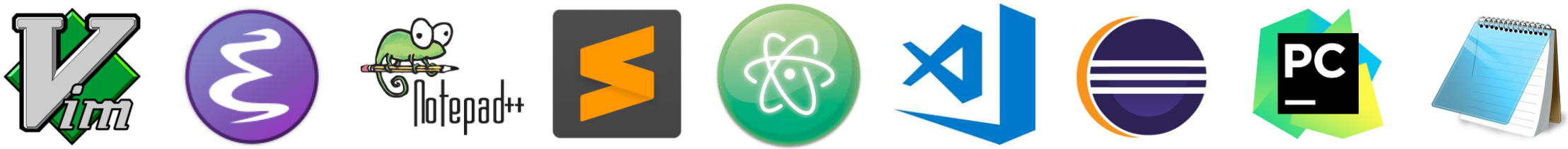
AST-based structural code editing for Python

Felix Kohlgrüber / 2018/07/10

Code Editors



… and so many more

Code Editor == Text Editor?

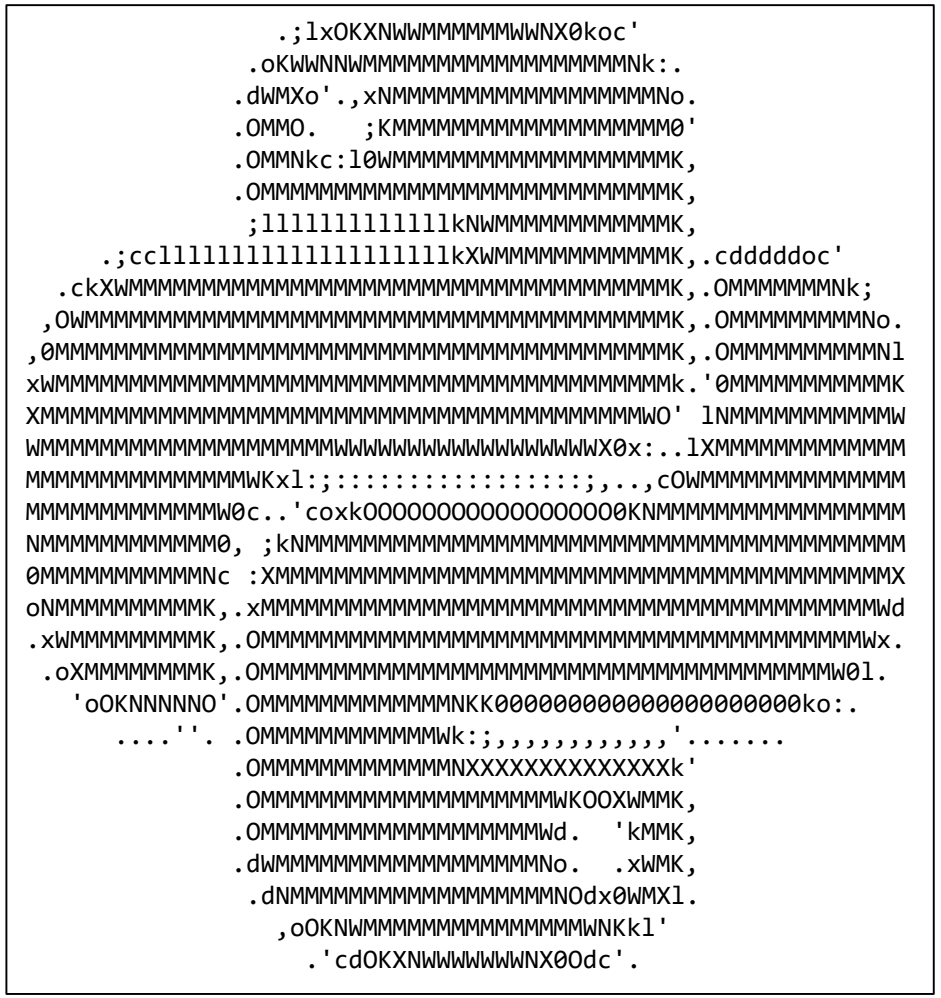
Text Editors

- General tool

```
def greet(s):
    print(f"Hello {s}")

if __name__ == '__main__':
    greet("World")
```

```
Lorem ipsum dolor sit amet, consectetur
adipisici elit, sed eiusmod tempor
incidunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris
nisi ut aliquid ex ea commodi
consequat.
```




REMEMBER, WITH GREAT POWER
COMES GREAT RESPONSIBILITY
quickmeme.com

Problem: Syntax Errors

```
def greet(s):
    return f"Hello {s}"

if __name__ == '__main__':
    print greet("World")
```

```
def half_diag(a, b):
    retrun ((sqrt(a[i]**2 + b[i]**2)/2 for i in range(len(a)))
```

```
function multiply(a, b) {
  return a * b;
}

if __name__ == '__main__’:
    print(multiply(42, 20))
```

Problem: Tabs vs. Spaces

```
if __name__ == '__main__':
    a = 1000
    print(f"a = {a}")
```

```
C:\Users\felix\Desktop>python test.py
  File "test.py", line 3
    print(f"a = {a}")
                    ^
IndentationError: unindent does not match any outer
indentation level
```

Problem: Style Guide Violations

```
def calculate(a   ,b) :
    sum=a+  b
    return(sum/ 2)

if __name__ == '__main__':
    print (calculate(1,3),)
```

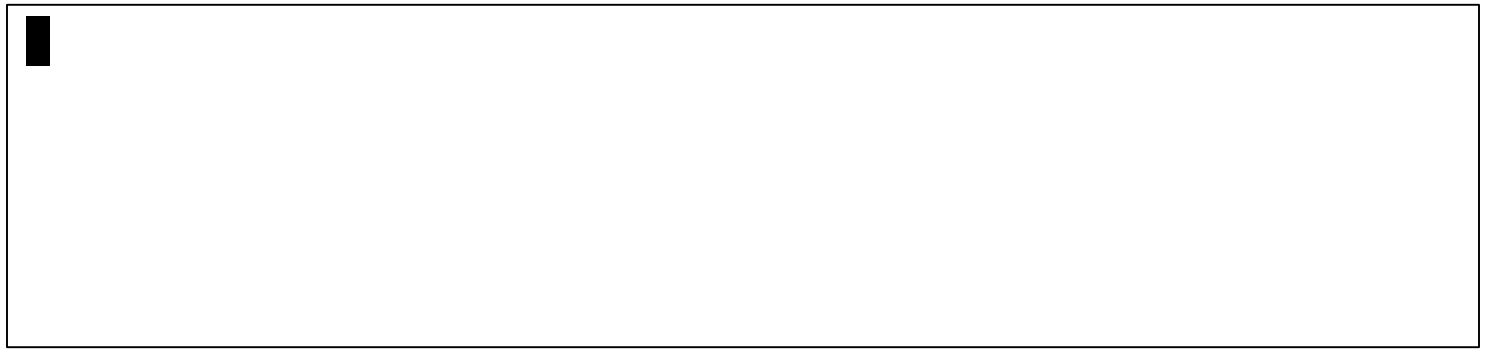
```
def calculate(a, b):
    sum = a + b
    return sum / 2

if __name__ == '__main__':
    print(calculate(1, 3), )
```

```
# Python 2
In [1]: def calculate(a   ,b) :
   ...:     sum=a+   b
   ...:     return(sum/ 2)
   ...:
   ...: if __name__ == '__main__':
   ...:     print (calculate(1,3),)
   ...:
(2,)
```

```
# Python 3
In [1]: def calculate(a   ,b) :
   ...:     sum=a+   b
   ...:     return(sum/ 2)
   ...:
   ...: if __name__ == '__main__':
   ...:     print (calculate(1,3),)
   ...:
2.0
```

Problem: Discoverability



Do we need all that power?

Structure Editors

- *"A **structure editor**, also **structured editor** or **projectional editor**, is any document editor that is cognizant of the document's underlying structure." (Wikipedia)*
- Edit parse tree / abstract syntax tree (AST) instead of sequence of characters

Abstract syntax trees

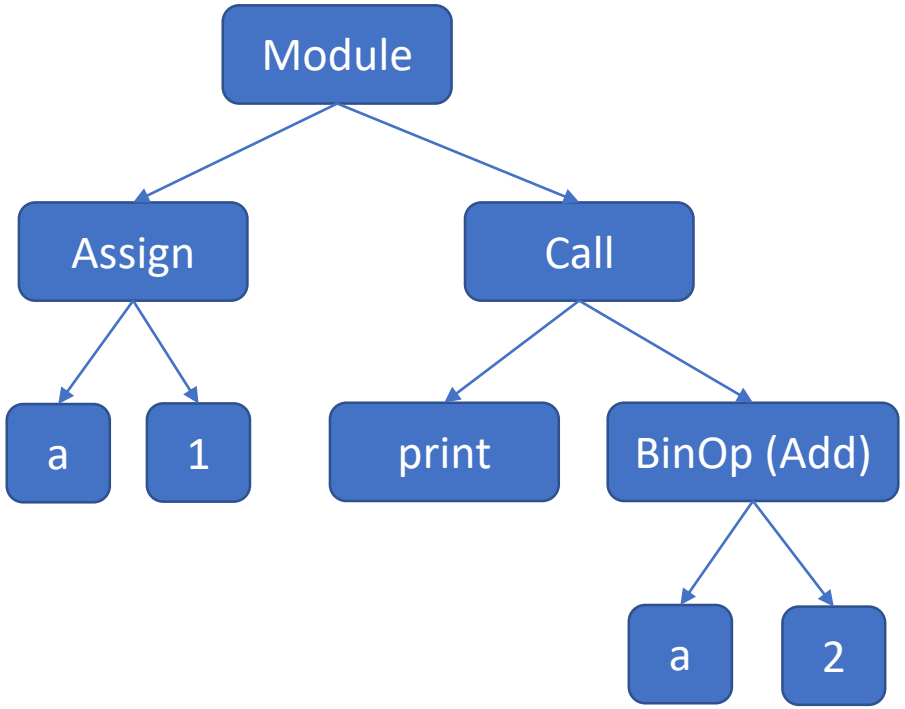
- Tree structure representing source code
- Example:

```
import ast

code = """
a = 1
print(a+2)
"""

a = ast.parse(code)
print(ast.dump(a))
```

```
# Simplified AST
Module(
   body=[
      Assign(targets=[Name(id='a')],
             value=Num(n=1)),
      Call(
        func=Name(id='print'),
        args=[
          BinOp(left=Name(id='a'),
                op=Add(),
                right=Num(n=2))
        ],
      )
   ]
)
```



Structure Editors

- *"A **structure editor**, also **structured editor** or **projectional editor**, is any document editor that is cognizant of the document's underlying structure." (Wikipedia)*
- Edit parse tree / abstract syntax tree (AST) instead of sequence of characters
- Active research topic since ~1985
- Example:

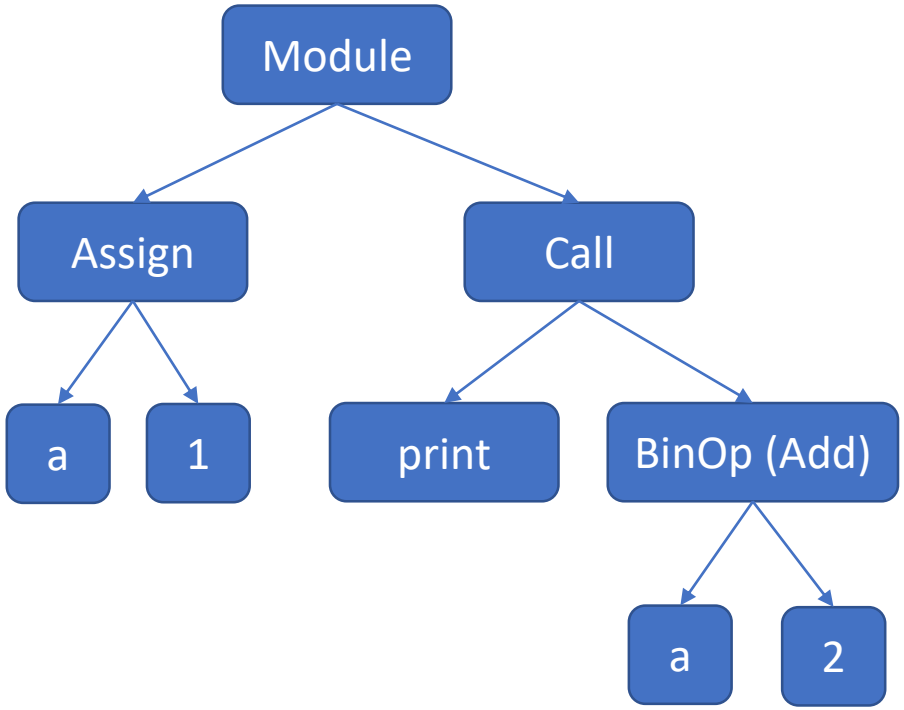
vrite

- Structured editor for Python
- Web-based (javascript, vue.js)
- Status: notevenclosetoalpha

DEMO

vrite – Navigation

- AST-nodes instead of character-by-character




```
a = 1
print(a+2)
```

vrite – Semantic commands

„Wrap with function call“

„Create function“

„Swap operands“

„Insert import statement“

„Move argument to the right“

…

- High-level interactions
- Availability depends on context

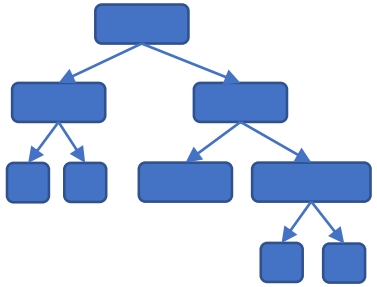
vrite – Deactivate code

- Alternative to „commenting out“-hack

```
x = my_func(a, b, input_sorted=True)
print(x * 5 / pi)
```

```
x = my_func(a, b, input_sorted=True)
print(x * 5 / pi)
```

vrite – Flexible representation



?

- Pep8/Black-formatted text
- Depending on available width
- User-configurable amount of indentation (in px)
- Parametric fonts

```
def fn(first_argument, second_arg):
    pass
```

```
def fn(
    first_argument,
    second_arg
):
    pass
```

```
def my_function(a, b):
    """This is the description of my Function"""
    return 2 * a + b
```

```
def my_function(a, b):
    """This is the description of my Function"""
    return 2 * a + b
```

vrite – Flexible representation (cont.)

- Use color to disambiguate tokens
- Use syntax that‘s easy to read/understand by humans
- *Possible: Braceless Javascript/Java/C++ (!!)*
- Use non-text elements

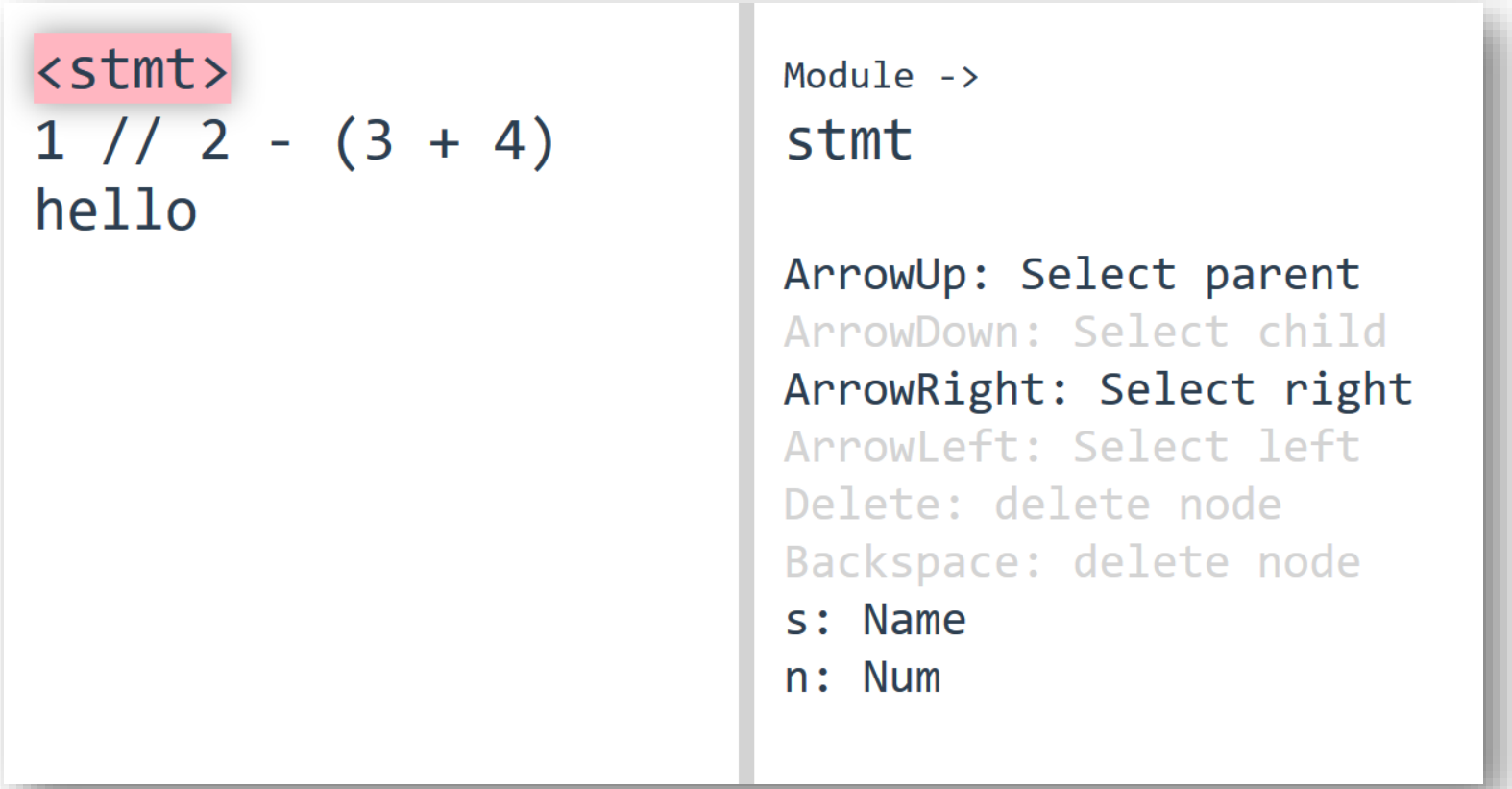
```
a = "1000"
b = 1000
c = "say \"hello\"\nsecond line"
d = map(lambda x: x+2, range(10))
```

```
a = 1000
b = 1000
c = say "hello"
    second line
c = say "hello"⏎second line
d = map(|x| x+2, ..10)
```

```
if a:
    if b:
        return 42
    else:
        return 23
else:
    if b:
        return 19
    else:
        return 0
```

return		b	not b
	a	42	23
	not a	19	0

vrite – Discoverability



Summary

```
File "<stdin>", line 1
    async = 123
          ^
SyntaxError: invalid syntax
```

```
def calculate(a   ,b) :
    sum=a+  b
    return(sum/ 2)

if __name__ == '__main__':
    print (calculate(1,3),)
```

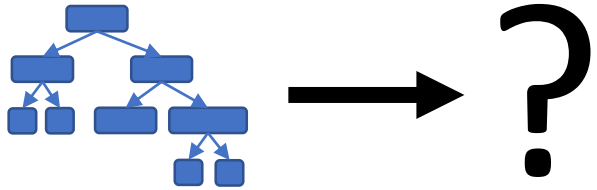
„Wrap with function call“

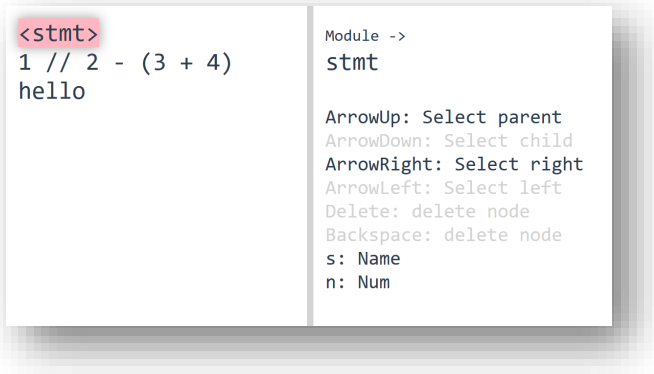
„Create function“

„Swap operands“

„Insert import statement“

„Move argument to the right“





Thank you!

github.com/fkohlgrueber/vrite

@fkohlgrueber

felix.kohlgrueber@gmail.com