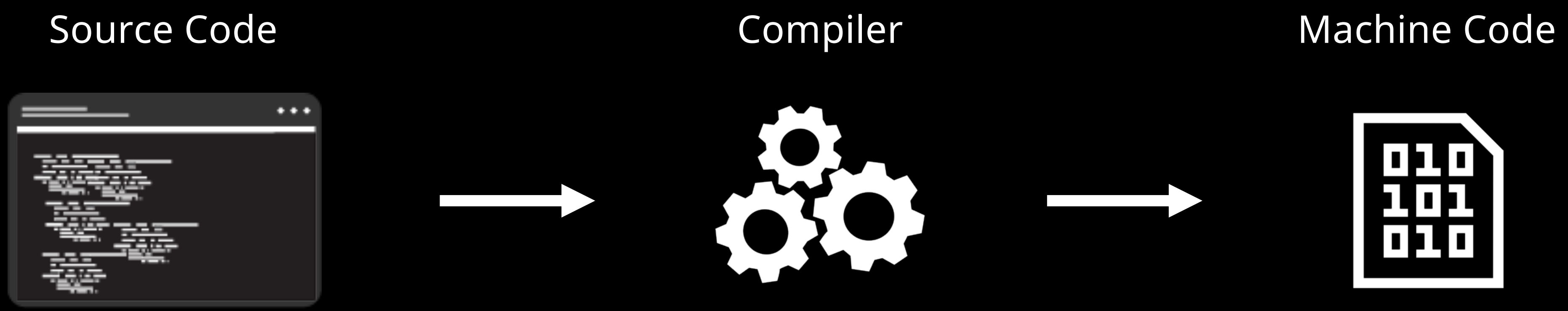
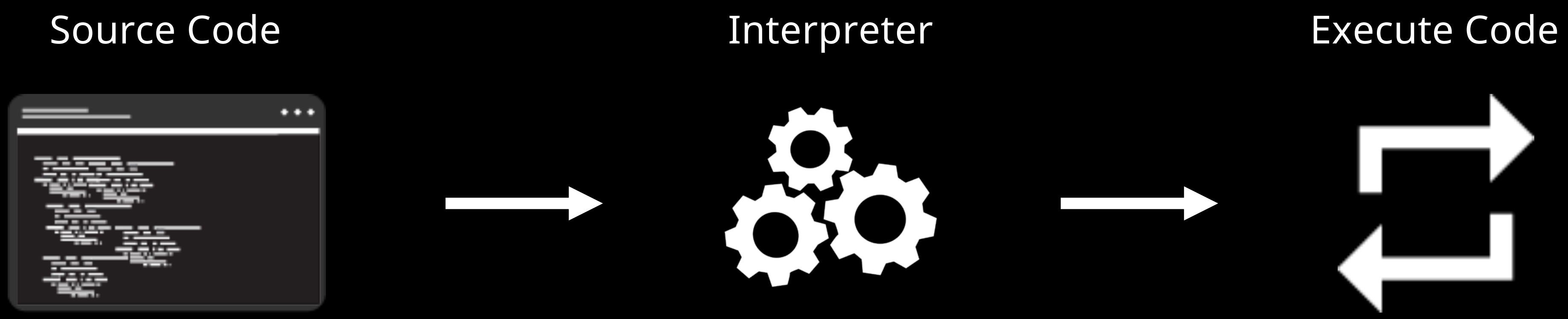
Fundamental Concepts Navigation

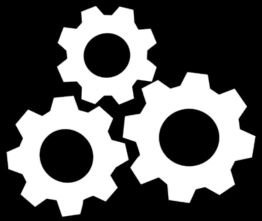

Source Code
Compiler
Machine Code
010
101
010

## Compiler

1. Lexical Analysis (Lexing)
2. Syntax Analysis (Parsing)
3. Semantic Analysis
4. Optimization
5. Code Generation


Source Code
Interpreter
Execute Code

## Interpreter



## Main Interpreter Strategies

A. Parse source code and execute directly
B. Compile to intermediate form (bytecode) and then execute:
   - Python does this

Lexical Analysis is the process of breaking up source code into identified tokens. Here's an example with a line of code converting from Celsius to Fahrenheit:

```
f = (c * 9 / 5) + 32
```

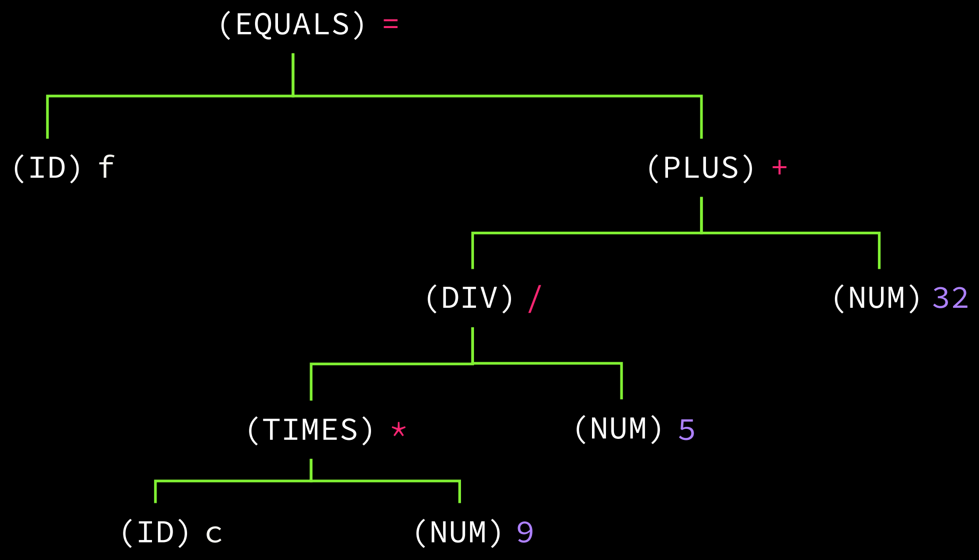
Lexical Analysis is the process of breaking up source code into identified tokens. Here's an example with a line of code converting from Celsius to Fahrenheit:

```
(ID) f , (EQUALS) = , (RPAREN) ( , (ID) c , (TIMES) * , (NUM) 9 , (DIV) / , (NUM) 5 ,

(LPAREN) ) , (PLUS) + , (NUM) 32
```

Syntax Analysis (or parsing) is the process of constructing a parse tree using the tokens provided by the lexer. Parsing provides more context about how the code should run.

```
(ID) f , (EQUALS) = , (RPAREN) ( , (ID) c , (TIMES) * , (NUM) 9 , (DIV) / , (NUM) 5 ,

(LPAREN) ) , (PLUS) + , (NUM) 32
```

Syntax Analysis (or parsing) is the process of constructing a parse tree using the tokens provided by the lexer. Parsing provides more context about how the code should run.



Semantic Analysis is the process of applying the languages rules to the parse tree.
This includes raising errors when rules are broken.
Example errors caught during semantic analysis:

- Type mismatch
- Undeclared variable
- Parameter mismatch

Keywords are reserved words that can't be used as identifiers in our code.
Python has a relatively small list of keywords.

```
False       finally     while
await       is          assert
else        return      del
import      and         global
pass        continue    not
None        for         with
break       lambda      async
except      try         elif
in          as          if
raise       def         or
True        from        yield
class       nonlocal
```

Bytecode instructions are the various commands that the Python virtual machine knows how to run.
Python is an interpreted programming language, but source code is compiled to bytecode before being run.

## Bytecode for our example

```
$ python3.7 -c "import dis; dis.dis('f = (c * 9 / 5) + 32')"
1             0 LOAD_NAME               0 (c)
              2 LOAD_CONST              0 (9)
              4 BINARY_MULTIPLY
              6 LOAD_CONST              1 (5)
              8 BINARY_TRUE_DIVIDE
             10 LOAD_CONST              2 (32)
             12 BINARY_ADD
             14 STORE_NAME              1 (f)
             16 LOAD_CONST              3 (None)
             18 RETURN_VALUE
```