

# Running the steps in sequence

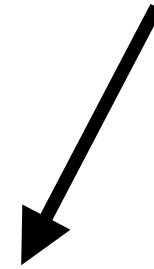
```
def interpret(source_code, verbose=False):
    try:
        lexemes = lexer.lex(source_code) ← Lex
        if verbose:
            print("Lexemes:")
            print(lexemes)
        try:
            ast = parser.parse(lexemes) ← Parse
            if verbose:
                print("AST:")
                print(ast)
            try:
                return process_ast_exp(ast) ← Interpret
            except Exception as e:
                if verbose:
                    print("Encountered interpreting error: {}".format(repr(e)))
                    print("Returning None")
                return None
        except Exception as e:
            if verbose:
                print("Encountered parsing error: {}".format(repr(e)))
                print("Returning None")
            return None
    except Exception as e:
        if verbose:
            print("Encountered lexing error: {}".format(repr(e)))
            print("Returning None")
        return None
```

**Error handling**

The diagram illustrates the flow of error handling in the provided Python code. On the right side, the text 'Error handling' is positioned. Three arrows originate from this text and point to specific exception blocks within the code: the top arrow points to the 'except Exception as e:' block inside the 'try:' block (handling interpreting errors), the middle arrow points to the 'except Exception as e:' block inside the 'try:' block (handling parsing errors), and the bottom arrow points to the 'except Exception as e:' block at the end of the function (handling lexing errors). This visualizes how the 'Error handling' concept is implemented across different stages of the interpreter's execution.

# Running from a file

Using argparse library to make it  
easy to run from the command line



```
with open(args.file, "r") as f:  
    source_code = f.read()
```

```
result = interpret(source_code)
```



All we need to do is call "interpret"  
on the source code string!