PEP 701: Full F-String Syntax Support in Black

Black needs to support the new Python 3.12 f-string syntax formalized in PEP 701, which lifts the arbitrary restrictions that existed in the original f-string implementation and provides a proper grammar for f-strings.

The Problem

Before Python 3.12, f-strings had several unintuitive restrictions that served no purpose from a language user perspective but existed only due to implementation limitations in CPython's parser:

Quote reuse was forbidden:

```python
# This failed in Python 3.11
f"Magic wand: {bag['wand']}"  # SyntaxError

# The workaround was ugly
f'Magic wand: {bag["wand"]}'
```

Backslashes were forbidden in expressions:

```python
# This failed in Python 3.11
names = ["Alice", "Bob"]
f"Names: {'\n'.join(names)}"  # SyntaxError

# You had to use a workaround
newline = '\n'
f"Names: {newline.join(names)}"
```

Comments were forbidden:

```python
```

```python
# This failed in Python 3.11
f"""Result: {
    bag['items']  # Get the items
}"""  # SyntaxError
```

Limited nesting depth: The deepest possible nesting was only 4 levels due to quote exhaustion:

```python
f"""{f"{f{f"{1+1}"}}"}"""  # Maximum in Python 3.11
```

Python 3.12 is introducing PEP 701 which fundamentally changes how f-strings work by formalizing their syntax. This enables several new f-string variations that our current parser can't handle at all. All of these are valid in Python 3.12 but Black chokes on them. For each of these, parsing simply fails and Black is not able to reformat the file.

Expected Behavior

Black should support the full range of PEP 701 syntax. For example, this should work:

1. Quote Reuse Within F-Strings

You can now use the same quote type inside an f-string expression as the outer f-string:

```python
# Single quotes inside single-quoted f-string
f'Magic wand: {bag["wand"]}'
f'{source.removesuffix(".py")}.c'

# Double quotes inside double-quoted f-string
f"Magic wand: {bag['wand']}"
f"Path: {source.removesuffix('.py')}.c"

# Triple quotes with nested strings
f"Complex: {"""nested"""}"
f"""Another: {"nested"}"""
```

This is the biggest change from PEP 701. Previously, the outer quotes "claimed" that quote type for the entire f-string, but now each level of string nesting is independent.

2. Arbitrary F-String Nesting

F-strings can now be nested inside other f-strings to arbitrary depth:

```python
# Simple nesting with quote reuse
f"{f"{f"{'nested'}"}"}"

# Nested f-strings with format specifiers
f"{f"{value:.{precision}f}"}"

# Deeply nested f-strings (at least 5 levels must work)
f"{f"{f"{f"{f"{'value'}"}"}"}"}"
```

3. Backslashes in Expression Parts

Backslashes and escape sequences are now allowed in f-string expressions:

```python
# Newline in join
names = ["Alice", "Bob", "Charlie"]
f"Names:\n{'\n'.join(names)}"

# Tab characters
f"Columns: {'\t'.join(columns)}"

# Escaped quotes in nested strings
f"Quote: {\"He said \\\"hello\\\"\"}"

# Raw strings with backslashes
f"{r'C:\path\to\file'.replace('\\', '/')}"
```

When strings are nested within f-strings, escape sequences are expanded when the innermost string is evaluated.

Expressions can now span multiple lines:

```python
# Simple multi-line expression
f"Result: {
    x + y
}"

# With nested structures
f"Items: {
    [item for item in items
     if item.active]
}"

# Complex expressions
f"""
    Sum: {
        sum(
            value
            for value in values
        )
    }
"""
```

5. Comments in Multi-Line F-Strings

Comments using the (#) character are now allowed:

```python
```

```python
# Comment in expression part
f"""Result: {
    bag['items']  # Get the items
}"""

# Multiple comments
f"""
    Data: {
        process(
            data  # Raw input
        )  # Processed result
    }
"""
```

The closing brace `}` must be on a different line from the comment, or it will be treated as part of the comment.

6. Format Specifiers With Expressions

Format specifiers can contain expressions, including nested f-strings:

```python
# Expression in format spec
width = 10
f"{value:>{width}}"

# Nested f-string in format spec
f"{value:{f'{width}'}}"

# Dynamic precision
precision = 2
f"{pi:.{precision}f}"

# Complex format spec with multiple levels (at least 2 levels of nesting)
f"{':*^{1:{1}}}"
```

7. Debug Expressions (f-string = operator)

The f-string debug syntax with (=) must continue to work and preserve whitespace:

```python
# Basic debug expression
f"{x + y=}"  # 'x + y=42'

# With spacing preserved
f"{ x + y = }"  # ' x + y = 42'

# Debug with format spec
f"{value=:.2f}"  # 'value=3.14'

# Nested debug expressions
f"{f"{x=}"=}"
```

8. Special Cases and Edge Cases

Lambda and colon expressions still need parentheses at top level:

```python
# This still requires parentheses
f"Lambda: {(lambda x: x*2)}"
f"Dict: {({'key': 'value'})}"

# But works fine when nested or not at top level
items = [f"{(lambda: x)()}" for x in range(5)]
```

Double braces for literal braces still work:

```python
f"{{literal braces}}"  # '{literal braces}'
f"{{nested: {value}}}"  # '{nested: 42}'
```

Multiline f-strings with nested multiline strings:

```python
# Multiline f-string containing another multiline string
f"""
   Content: {'''
      nested
      multiline
   '''}
"""
```

Assignment expressions in f-strings:

```python
f"{(x := 10)}"  # Assignment needs parentheses
f"Value is {x} after {(x := x + 1)}"
```

9. Tokenization Edge Cases

Self-documenting expressions with method calls:

```python
# Method calls with quoted arguments
f"{source.removesuffix('.py')}.c"
f"{path.replace('\\', '/')}"

# Complex method chains
f"Result: {obj.method('arg').property}"
```

Mixed quote types in nested context:

```python
# Triple-quoted with nested strings
f"Text {f"""{f'{value}'}"""}'"
f"""Text {f"{f'{value}'}"'}"""
```

Empty expressions and edge whitespace:

```python
# Whitespace handling
f"{value }"
f"{ value}"
f"{ value }"
```

10. Regression Cases

These specific patterns caused crashes or parsing failures and must be handled:

Newlines in nested f-strings (pre-PEP 701 valid code):

```python
# This was valid in Python 3.9+ but crashed Black initially
f"""foo {
    f"bar"
}"""
```

Escaped quotes within f-string expressions:

```python
f"Quote: {\"He said \\\"hello\\\"\"}"
f'{\'Single quote: \\\'test\\\'\'}'
```

Complex nested f-strings with varying quote types:

```python
f"outer {f'inner {f"deep"}'}  more"
```

Format specs with special characters:

```python
# Certain format specifications that caused parser issues
f"This is a long string, {2+2:d} with format"
```

Escaped braces `\{` in f-strings:

```
python

# Although the PEP leaves escaped braces for future work,
# the tokenizer must not crash on them
# Note: These produce literal backslash + brace, not literal brace
```

Performance Considerations: The new tokenizer should not cause significant performance degradation on files with complex f-strings or long string literals.

Testing

To verify the implementation works correctly, Black should:

1. Successfully parse all valid PEP 701 f-string syntax
2. Produce stable output (formatting is idempotent)
3. Generate valid Python code

You can run the code in `/workdir/black` with `uv`. There is a `/workdir/black/.venv` configured already (you can do `source .venv/bin/activate` to activate the env). You can run the tests with pytest.