```python
import asyncio


class AsyncOpen:
    def __init__(self, f, m):
        self.fp = open(f, m)

    async def __aenter__(self):
        return self

    async def read(self):
        return self.fp.read()

    async def __aexit__(self, *args, **kwargs):
        self.fp.close()

class ExampleClass:
    def __init__(self, name):
        self.name = name

    async def async_method(self):
        async with AsyncOpen('file.txt', 'r') as f:
            return await f.read()

# Constants, Lists, Tuples, Sets, Dicts
x = [1, 2, 3]  # ast.List
y = (1, 2, 3)  # ast.Tuple
z = {1, 2, 3}  # ast.Set
w = {'a': 1, 'b': 2}  # ast.Dict

# Starred, UnaryOp, BinOp, Compare
*rest, = x  # ast.Starred in unpacking
not_x = not x  # ast.UnaryOp
sum_xy = x[0] + y[1]  # ast.BinOp
comparison = x[0] < y[1]  # ast.Compare

# Call, IfExp, Attribute, NamedExpr, Subscript, Slice
result = len(x)  # ast.Call
max_value = x[0] if x[0] > y[0] else y[0]  # ast.IfExp
attribute_access = result.bit_length()  # ast.Attribute
if (n := len(x)) > 2:  # ast.NamedExpr
    print(f"List is longer than 2, length is {n}")
list_slice = x[1:2]  # ast.Subscript with ast.Slice

# SetComp, GeneratorExp, DictComp
set_comp = {i*2 for i in x}  # ast.SetComp
generator_exp = (i*2 for i in x)  # ast.GeneratorExp
dict_comp = {i: i*2 for i in x}  # ast.DictComp
```

```python

# Comprehension, Assign, AnnAssign, AugAssign
comprehension = [i for i in x if i > 1]  # ast.comprehension in ListComp
x[0] = 10  # ast.Assign
count: int = 0  # ast.AnnAssign
count += 1  # ast.AugAssign

# For, AsyncFor, With, AsyncWith, FunctionDef, Lambda, YieldFrom, Await, ClassDef
async def async_loop(items):  # ast.AsyncFor
    async for item in items:
        print(item)

async def async_read(file):  # ast.AsyncWith
    async with AsyncOpen(file, 'r') as f:
        return await f.read()  # ast.Await

def function_def(x, y):  # ast.FunctionDef
    return x + y

lambda_func = lambda x, y: x + y  # ast.Lambda

def generator_func():  # ast.YieldFrom
    yield from range(10)

# Using ExampleClass
example_instance = ExampleClass("Example")

# Call the function
function_result = function_def(5, 3)

# Use the lambda
lambda_result = lambda_func(2, 3)

# Instantiate the class and call a method
example_instance = ExampleClass("Example")

# Asynchronous operations require running an event loop
async def run_async_operations(file):
    # Call the async method
    async_result = await example_instance.async_method()

    async def items():
        yield 1
        yield 2
        yield 3

    # Call the async loop
    await async_loop(items())

    # Call async_read
    async_read_result = await async_read('file.txt')

# Multiplication with sequences
sequence_multiplication = [1, 2, 3] * 2  # Repeats the list

# Modulus for formatting strings
name = "World"
formatted_string = "Hello, %s!" % name  # Old-style string formatting

# ast.In for membership tests
item = 2
container = [1, 2, 3, 4, 5]
membership_test = item in container  # This will use ast.Compare with ast.In
print(f"Item in container: {membership_test}")

# A more complex example of kwargs with a function that calculates an operation
def calculate_operation(x, y, operation='add'):
    if operation == 'add':
        return x + y
    elif operation == 'subtract':
        return x - y
    elif operation == 'multiply':
        return x * y
    elif operation == 'divide':
        return x / y
    else:
        return "Unknown operation"

# Calling the function with kwargs
result_add = calculate_operation(x=5, y=3, operation='add')

result_multiply = calculate_operation(x=5, y=3, operation='multiply')

# Assigning to a slice of a list
numbers = [0, 0, 0, 0, 0]  # A list of five zeros

# Replace a slice of the list with new values
numbers[1:4] = [1, 2, 3]  # This modifies the list in place

# Unary operation for negation
negative_number = -10
```