```
import ast
import tokenize
import io
from sentence_transformers import SentenceTransformer

# 10 example snippets covering varied patterns
snippets = [
    "def add(a, b):\n    return a + b",
    "class Greeter:\n    def greet(self, name):\n        print(f\"Hello, {name}!\")",
    "import math",
    "squares = [x * x for x in range(5)]",
    "d = {'one': 1, 'two': 2}",
    "unique = {x for x in [1, 2, 2, 3]}",
    "with open('file.txt') as f:\n    data = f.read()",
    "result = \"yes\" if True else \"no\"",
    "value = obj.attr",
    "match color:\n    case \"red\":\n        print(\"Color is red\")\n    case _:\n        print(\"Unknown color\")",
]

class CodeAnalyzer(ast.NodeVisitor):
    def __init__(self):
        self.functions = []
        self.classes = []
        self.imports = []
        self.patterns = {
            "comprehensions": 0,
            "with_statements": 0,
            "if_expressions": 0,
            "attribute_accesses": 0,
            "match_statements": 0,
        }

    def visit_FunctionDef(self, node):
        self.functions.append(node.name)
        self.generic_visit(node)

    def visit_ClassDef(self, node):
        self.classes.append(node.name)
        self.generic_visit(node)

    def visit_Import(self, node):
        for alias in node.names:
            self.imports.append(alias.name)
        self.generic_visit(node)

    def visit_ImportFrom(self, node):
        for alias in node.names:
            self.imports.append(alias.name)
        self.generic_visit(node)

    def visit_ListComp(self, node):
        self.patterns["comprehensions"] += 1
        self.generic_visit(node)

    def visit_SetComp(self, node):
        self.patterns["comprehensions"] += 1
        self.generic_visit(node)

    def visit_DictComp(self, node):
        self.patterns["comprehensions"] += 1
        self.generic_visit(node)

    def visit_With(self, node):
        self.patterns["with_statements"] += 1
        self.generic_visit(node)

    def visit_IfExp(self, node):
        self.patterns["if_expressions"] += 1
        self.generic_visit(node)

    def visit_Attribute(self, node):
        self.patterns["attribute_accesses"] += 1
        self.generic_visit(node)

    def visit_Match(self, node):
        self.patterns["match_statements"] += 1
        self.generic_visit(node)

def tokenize_code(code):
    tokens = []
    readline = io.BytesIO(code.encode('utf-8')).readline
```

```
    for toknum, tokval, _, _, _ in tokenize.tokenize(readline):
        tokens.append((toknum, tokval))
    return tokens

# Analyze snippets and print details
for i, code in enumerate(snippets, 1):
    print(f"Snippet {i}:")
    print(code)

    tree = ast.parse(code)
    analyzer = CodeAnalyzer()
    analyzer.visit(tree)
    tokens = tokenize_code(code)

    print("Extracted Functions:", analyzer.functions)
    print("Extracted Classes:", analyzer.classes)
    print("Extracted Imports:", analyzer.imports)
    print("Code Patterns:", analyzer.patterns)
    print("Tokens (first 10 shown):", tokens[:10])
    print("-" * 50)

# Now encode snippets using MPNet
model_mpnet = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')

# Encoding snippets for semantic embeddings
embeddings_mpnet = model_mpnet.encode(snippets, normalize_embeddings=True)

print(f"Encoded {len(snippets)} snippets into embeddings of shape {embeddings_mpnet.shape}")
print("First snippet embedding sample:", embeddings_mpnet[0][:5])
```

```
Snippet 1:
def add(a, b):
    return a + b
Extracted Functions: ['add']
Extracted Classes: []
Extracted Imports: []
Code Patterns: {'comprehensions': 0, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'def'), (1, 'add'), (55, '('), (1, 'a'), (55, ','), (1, 'b'), (55, ')'), (55, ':
--------------------------------------------------
Snippet 2:
class Greeter:
    def greet(self, name):
        print(f"Hello, {name}!")
Extracted Functions: ['greet']
Extracted Classes: ['Greeter']
Extracted Imports: []
Code Patterns: {'comprehensions': 0, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'class'), (1, 'Greeter'), (55, ':'), (4, '\n'), (5, '    '), (1, 'def'), (1, 'gr
--------------------------------------------------
Snippet 3:
import math
Extracted Functions: []
Extracted Classes: []
Extracted Imports: ['math']
Code Patterns: {'comprehensions': 0, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'import'), (1, 'math'), (4, ''), (0, '')]
--------------------------------------------------
Snippet 4:
squares = [x * x for x in range(5)]
Extracted Functions: []
Extracted Classes: []
Extracted Imports: []
Code Patterns: {'comprehensions': 1, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'squares'), (55, '='), (55, '['), (1, 'x'), (55, '*'), (1, 'x'), (1, 'for'), (1,
--------------------------------------------------
Snippet 5:
d = {'one': 1, 'two': 2}
Extracted Functions: []
Extracted Classes: []
Extracted Imports: []
Code Patterns: {'comprehensions': 0, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'd'), (55, '='), (55, '{'), (3, "'one'"), (55, ':'), (2, '1'), (55, ','), (3, "'
--------------------------------------------------
Snippet 6:
unique = {x for x in [1, 2, 2, 3]}
Extracted Functions: []
Extracted Classes: []
Extracted Imports: []
Code Patterns: {'comprehensions': 1, 'with_statements': 0, 'if_expressions': 0, 'attribute_accesses': 0, 'match_statements':
Tokens (first 10 shown): [(67, 'utf-8'), (1, 'unique'), (55, '='), (55, '{'), (1, 'x'), (1, 'for'), (1, 'x'), (1, 'in'), (55,
--------------------------------------------------
Snippet 7:
with open('file.txt') as f:
    data = f.read()
Extracted Functions: []
Extracted Classes: []
```

```
Extracted Imports: []
```

```
from transformers import AutoTokenizer

# Initialize DistilRoBERTa tokenizer
tokenizer = AutoTokenizer.from_pretrained("distilroberta-base")

# Example snippets (code as strings)
snippets = [
    "def add(a, b):\n    return a + b",
    "class Greeter:\n    def greet(self, name):\n        print(f\"Hello, {name}!\")",
    "import math",
    "squares = [x * x for x in range(5)]",
    "d = {'one': 1, 'two': 2}",
    "unique = {x for x in [1, 2, 2, 3]}",
    "with open('file.txt') as f:\n    data = f.read()",
    "result = \"yes\" if True else \"no\"",
    "value = obj.attr",
    "match color:\n    case \"red\":\n        print(\"Color is red\")\n    case _:\n        print(\"Unknown color\")",
]

# Tokenize and encode with padding to the longest sequence
encoded_inputs = tokenizer(
    snippets,
    padding=True,         # Pad to the longest snippet
    truncation=True,      # Truncate if too long
    return_tensors="pt"   # Return PyTorch tensors
)

# Print the keys and shapes to confirm
print("Keys in encoded inputs:", encoded_inputs.keys())
print("Input IDs shape:", encoded_inputs['input_ids'].shape)
print("Attention mask shape:", encoded_inputs['attention_mask'].shape)

# Example: print token ids of first snippet (truncated/padded)
print("Token IDs snippet 1:", encoded_inputs['input_ids'][0])
```

```
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 1.09kB/s]
config.json: 100%  480/480 [00:00<00:00, 8.15kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 16.2MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 4.20MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 15.8MB/s]
Keys in encoded inputs: KeysView({'input_ids': tensor([[    0,  9232,  1606,  1640,   102,     6,   741,  3256, 50118,  1437,
          1437,  1437,   671,    10,  2055,   741,     2,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0,  4684,  6879,  5906,    35, 50118,  1437,  1437,  1437,  3816,
         17395,  1640, 13367,     6,   766,  3256, 50118,  1437,  1437,  1437,
          1437,  1437,  1437,  1437,  5780,  1640,   506,   113, 31414,     6,
         25522, 13650, 24303,  2901,    43,     2,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 41975, 10638,     2,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 30919,  5347,  5457,   646,  1178,  1009,  3023,    13,  3023,
            11,  1186,  1640,   245, 46077,     2,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0,   417,  5457, 25522,   108,  1264, 13373,   112,     6,   128,
          7109, 13373,   132, 24303,     2,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 34755,  5457, 25522,  1178,    13,  3023,    11,   646,   134,
             6,   132,     6,   132,     6,   155, 49750,     2,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0,  5632,   490, 45803, 21710,     4, 46795, 27645,    25,   856,
            35, 50118,  1437,  1437,  1437,   414,  5457,   856,     4, 12745,
         43048,     2,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 43155,  5457,    22, 10932,   113,   114,  7447,  1493,    22,
          2362,   113,     2,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 19434,  5457, 26907,     4, 44156,     2,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1,     1,     1,     1,
             1,     1,     1,     1,     1,     1,     1],
        [    0, 10565,  3195,    35, 50118,  1437,  1437,  1437,   403,    22,
          2050,  7862, 50118,  1437,  1437,  1437,  1437,  1437,  1437,  1437,
          5780, 46469, 44287,    16,  1275,  8070, 50118,  1437,  1437,  1437,
           403, 18134,    35, 50118,  1437,  1437,  1437,  1437,  1437,  1437
```

```
from sentence_transformers import SentenceTransformer

# Load pretrained embedding models
model_minilm = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
model_distilroberta = SentenceTransformer('sentence-transformers/msmarco-distilroberta-base-v2')
model_mpnet = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')

# The code snippets (as text, not tokens) - example subset
snippets = [
    "def add(a, b):\n    return a + b",
    "class Greeter:\n    def greet(self, name):\n        print(f\"Hello, {name}!\")",
    "import math",
    "squares = [x * x for x in range(5)]",
    "unique = {x for x in [1, 2, 2, 3]}",
    "with open('file.txt') as f:\n    data = f.read()",
    "match color:\n    case \"red\":\n        print(\"Color is red\")\n    case _:\n        print(\"Unknown color\")",
]

# Encode each snippet with all models
embeddings_minilm = model_minilm.encode(snippets, normalize_embeddings=True)
embeddings_distilroberta = model_distilroberta.encode(snippets, normalize_embeddings=True)
embeddings_mpnet = model_mpnet.encode(snippets, normalize_embeddings=True)

# Output dimensions and sample embeddings for first snippet for verification
print("MiniLM embedding shape:", embeddings_minilm.shape)
print("MiniLM embedding for snippet 1:", embeddings_minilm[0][:5])  # first 5 values only

print("DistilRoBERTa embedding shape:", embeddings_distilroberta.shape)
```

```
print("DistilRoBERTa embedding for snippet 1:", embeddings_distilroberta[0][:5])

print("MPNet embedding shape:", embeddings_mpnet.shape)
print("MPNet embedding for snippet 1:", embeddings_mpnet[0][:5])
```

```
modules.json: 100% 349/349 [00:00<00:00, 4.60kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 1.55kB/s]
README.md: 10.5k/? [00:00<00:00, 279kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 1.19kB/s]
config.json: 100% 612/612 [00:00<00:00, 10.4kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 62.3MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 11.1kB/s]
vocab.txt: 232k/? [00:00<00:00, 2.12MB/s]
tokenizer.json: 466k/? [00:00<00:00, 9.56MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 2.86kB/s]
config.json: 100% 190/190 [00:00<00:00, 11.4kB/s]
modules.json: 100% 229/229 [00:00<00:00, 13.2kB/s]
config_sentence_transformers.json: 100% 122/122 [00:00<00:00, 7.54kB/s]
README.md: 3.54k/? [00:00<00:00, 195kB/s]
```