An Exploratory Study on the Predominant Programming Paradigms in Python Code

Robert Dyer and Jigyasa Chauhan
Increasing Prevalence of Multi-Paradigm Languages
Increasing Prevalence of Multi-Paradigm Languages
Increasing Prevalence of Multi-Paradigm Languages
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])

class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

    def __init__(self):
        self.x = 1

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])

Object-oriented features
class MyCounter:
    x = 1

def val(self):
    def wrapper():
        return self.x
    return wrapper

def __iter__(self):
    self.x = 1
    return self

def __next__(self):
    if self.x > 50:
        raise StopIteration
    tmp = self.x
    self.x += 1
    return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```
RQ1  What is the distribution of predominant paradigms for Python projects on GitHub?

RQ2  What are the most and least used features for some programming paradigms?

RQ3  Are project size and predominant paradigm related?

RQ4  How does predominant paradigm use change over time?
Research Questions

RQ1  What is the distribution of predominant paradigms for Python projects on GitHub?

RQ2  What are the most and least used features for some programming paradigms?  

| Feature              | Count     |
|----------------------|-----------|
| method declarations  | 2,575,397 |
| class declarations   | 1,738,668 |

RQ3  Are project size and predominant paradigm related?

RQ4  How does predominant paradigm use change over time?
RQ1  What is the distribution of predominant paradigms for Python projects on GitHub?

RQ2  What are the most and least used features for some programming paradigms?  

- method declarations: 2,575,397
- class declarations: 1,738,668
- built-in functions (functools/itertools): 990,333
- array comprehensions: 729,309

RQ3  Are project size and predominant paradigm related?

RQ4  How does predominant paradigm use change over time?
Research Questions

RQ1  What is the distribution of predominant paradigms for Python projects on GitHub?

RQ2  What are the most and least used features for some programming paradigms?

| Feature                                               | Count       |
|-------------------------------------------------------|-------------|
| method declarations                                    | 2,575,397   |
| class declarations                                     | 1,738,668   |
| built-in functions (functools/itertools)               | 990,333     |
| array comprehensions                                   | 729,309     |

RQ3  Are project size and predominant paradigm related?

RQ4  How does predominant paradigm use change over time?

(it doesn’t)
| Metric                                    | Value               |
|------------------------------------------|---------------------|
| Projects                                 | 101,648             |
| Revision (with a Python file)            | 15,254,331          |
| Python Files (main branch only)          | 3,658,391           |
| Python File Snapshots                    | 68,787,597          |
| ASTs                                     | 105,907,774,611     |
Phase 1: Manual Classification

Phase 2: Automated Classification
Phase 1: Manual Classification

Phase 2: Automated Classification

Sample 102 files (out of 98,537)
Phase 1: Manual Classification

Phase 2: Automated Classification

Sample 102 files (out of 98,537)

3 raters: 0.759 kappa ("good" agreement)
Phase 1: Manual Classification

Phase 2: Automated Classification
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:  
    x = 1  

def val(self):  
    def wrapper():  
        return self.x  
    return wrapper  

def __iter__(self):  
    self.x = 1  
    return self  

def __next__(self):  
    if self.x > 50:  
        raise StopIteration  
    tmp = self.x  
    self.x += 1  
    return tmp  

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

def val(self):
    def wrapper():
        return self.x
    return wrapper

def __iter__(self):
    self.x = 1
    return self

def __next__(self):
    if self.x > 50:
        raise StopIteration
    tmp = self.x
    self.x += 1
    return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1
    
def val(self):  
        def wrapper(): 
            return self.x 
        return wrapper

    def __iter__(self):  
        self.x = 1  
        return self

    def __next__(self):  
        if self.x > 50:  
            raise StopIteration 
        tmp = self.x  
        self.x += 1  
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
class MyCounter:
    x = 1

def val(self):
    def wrapper():
        return self.x
    return wrapper

def __iter__(self):
    self.x = 1
    return self

def __next__(self):
    if self.x > 50:
        raise StopIteration
    tmp = self.x
    self.x += 1
    return tmp

print([y for y in MyCounter() if y % 2 == 0])
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```
```python
class MyCounter:
    x = 1

    def val(self):
        def wrapper():
            return self.x
        return wrapper

    def __iter__(self):
        self.x = 1
        return self

    def __next__(self):
        if self.x > 50:
            raise StopIteration
        tmp = self.x
        self.x += 1
        return tmp

print([y for y in MyCounter() if y % 2 == 0])
```

| Type               | Count | Percent |
|--------------------|-------|---------|
| Statements         | 16    | 100.00% |
| Functional         | 5     | 31.25%  |
| Object-Oriented    | 16    | 100.00% |
| Procedural         | 3     | 18.75%  |
| Imperative         | 1     | 6.25%   |

Automated Classification

```
|                | Human Judgements | Machine Judgements |
|----------------|------------------|--------------------|
| Imperative     | 5                | 8                  |
| Mixed          | 16               | 10                 |
| Object-Oriented| 53               | 55                 |
| Procedural     | 28               | 29                 |
RQ1: Python files are small
RQ1: Python files are small
RQ1: Python files are small
RQ1: Python is Truly Multi-Paradigm
RQ3: Correlating # Statements with Predominant Paradigm

Files

OO
Procedural
Imperative
Mixed
RQ3: Correlating # Statements with Predominant Paradigm

![Bar chart showing the number of files for different programming paradigms: OO, Procedural, Imperative, and Mixed. The OO paradigm has the highest number of files, followed by Procedural, Imperative, and Mixed.](chart.png)
RQ3: Correlating # Statements with Predominant Paradigm
| Description                          | Value     |
|-------------------------------------|-----------|
| Projects                            | 101,648   |
| Revision (with a Python file)       | 15,254,331|
| Python Files                        | 7,758,882 |
| (main branch only)                  | 3,658,391 |
| Python File Snapshots               | 68,787,597|
| ASTs                                | 105,907,774,611 |
| Category                              | Value     |
|--------------------------------------|-----------|
| Projects                             | 101,648   |
| Revision (with a Python file)        | 15,254,331|
| Python Files                         | 7,758,882 |
| (main branch only)                   | 3,658,391 |
| Python File Snapshots                | 68,787,597|
| ASTs                                 | 105,907,774,611|

Phase 1: Manual Classification

Phase 2: Automated Classification

COLLEGE OF ENGINEERING
| Category                              | Value         |
|--------------------------------------|---------------|
| Projects                             | 101,648       |
| Revision (with a Python file)        | 15,254,331    |
| Python Files                         | 7,758,882     |
| (main branch only)                   | 3,658,391     |
| Python File Snapshots                | 68,787,597    |
| ASTs                                 | 105,907,774,611 |

**Phase 1: Manual Classification**

**Phase 2: Automated Classification**
| Category                        | Value                  |
|--------------------------------|------------------------|
| Projects                       | 101,648                |
| Revision (with a Python file)  | 15,254,331             |
| Python Files                   | 7,758,882              |
| (main branch only)             | 3,658,391              |
| Python File Snapshots          | 68,787,597             |
| ASTs                           | 105,907,774,611        |

**Phase 1: Manual Classification**

**Phase 2: Automated Classification**
| Category                          | Count            |
|----------------------------------|------------------|
| Projects                         | 101,648          |
| Revision (with a Python file)    | 15,254,331       |
| Python Files                     | 7,758,882        |
| (main branch only)               | 3,658,391        |
| Python File Snapshots            | 68,787,597       |
| ASTs                             | 105,907,774,611  |

**Phase 1: Manual Classification**

**Phase 2: Automated Classification**

---

![Graphs showing file statements distributions across different file structures and programming paradigms.](image)