AN EMPIRICAL STUDY ON THE CLASSIFICATION
OF PYTHON LANGUAGE FEATURES USING
EYE-TRACKING

Committee:
Dr. Robert Dyer
Dr. Bonita Sharif
Dr. Rahul Purandare
Python is a multi-paradigm programming language and currently one of the most popular languages.

- **Imperative**: assignment, logical operations, ...
- **Procedural**: def, return, calls, ...
- **Object-Oriented**: class, inheritance, with, ...
- **Functional**: lambda, for loops, iterators, ...
• Peng et al. found that developers *choose* certain features more than others to perform a task with Python project (2021)[3]

• Floyd et al. found that if a developer learns imperative style of coding, then it is *harder to switch* to OO paradigm (1979)[5]

• Shrestha et al. found that *prior learning* of a previous language can hinder the grasping of a new language (2020)[4]

• Alexandru et al. studied how developers *lack an understanding* of Python definitions and use them over GitHub and StackOverflow (2018)[2]
• Dyer and Chauhan 2022, explored over 100K+ Python projects from GitHub and classified Python paradigms using a query. They found that functional paradigm was used significantly less than procedural and object-oriented paradigms\(^1\)

• Therefore, we were interested to investigate how Python developers classify Python paradigms.
Help the people in education sector by teaching and training developers with Python paradigms and features.

Training Python developers with new language paradigms and features, required to perform certain tasks in the industry.

Researchers trying to understand Program Comprehension. Also, the Python community that has been maintaining Python, can understand how developers use Python features.
RESEARCH QUESTIONS
Research Questions

1. How difficult is it for developers to classify the predominant Python paradigm?

2. How accurately do developers classify the predominant paradigm in Python code?

3. Do developers fixate their gaze on specific Python language features when classifying predominant paradigms?

4. Does the predominant paradigm affect how long developer’s take to debug logical errors?

5. Does the predominant paradigm affect a developer’s ability to debug logical errors?
We needed to investigate developer’s behavior with collecting surveys and analyzing time data using Python libraries.

We needed eye-tracking to understand developer’s behavior with respect to classification and bug localization Python paradigms.

We needed to interview Python developers to understand their approach and methodology towards Python paradigms.
STUDY DESIGN AND APPROACH
Boa to search for tasks
Eclipse IDE for viewing Python code
Tobii TX300 eye tracker (60 Hz)
iTrace plugin and toolkit
• 29+2 participants
  (removed 1 due to no Python experience and
   1 due to poor eye tracking calibration)
• More than 85% were CS majors
• All participants had at least 1 year of experience in Python
Task Categories

Task Category 1 – Classification of paradigms
• Small code (1-15 statements)
• Medium code (16-30 statements)
• Large code (31-45 statements)

Task Category 2 – Bug localization in different paradigms
• Cube of a number
• Factorial of a number
• Largest number
• Palindrome number
Data Collection Methods

Survey + Task questionnaires (Google forms)

Eye-tracking data (XML files and database by iTrace toolkit)

Audio only interview (audio files → transcribed text on index cards)
Study Flow

1. Pre-questionnaire
2. Training
3. Task Category 1 (Classification)
4. Interview
5. Task Category 2 (Bug Localization)
6. Interview
7. Post-questionnaire
Training Example for Classification (Task 1)

class MyNumbers:
    # func oo
    
x = 1
    # oo imp
    def m(self):
        # oo
        def m3():
            # oo
            return 1
            y = m3()  # oo proc
            return y  # oo
    
def __iter__(self):
        # func oo
        self.x = 1
        return self
        
def __next__(self):
        # func oo
        y = self.x
        self.x += 1
        return y
    
x = MyNumbers()  # oo

(code listing taken from [1])

Paradigms
func: functional
oo: object-oriented
imp: imperative
proc: procedural
RESULTS: Qualitative
How would you rate your programming in Python?
Pre-questionnaire Data

More than 50% use Python frequently!

How often do you program in Python?
How important is it for you to code in a specific programming paradigm? For example: Functional, Object oriented, Procedural

More than 60% think paradigms are important!
Task 1 Questionnaire

Self-identified approach used to classify the predominant paradigm
RESULTS: Quantitative
Results: Task Category 1 (Classification)
RQ1: How difficult is it for developers to classify the predominant Python paradigm?

Time taken to classify predominant paradigm for Task 1
RQ1: How difficult is it for developers to classify the predominant Python paradigm?

Time taken to classify predominant paradigm for Task 1 by size: small, medium, large
• We found that participants classify all paradigms in a similar time.
• We see no correlation between different length of the code and time taken to classify.
RQ2: How accurately do developers classify the predominant paradigm in Python code?

- Very confident: 47.2%
- Confident: 38.7%
- Moderately Confident: 11.3%
- Slightly Confident: 3.9%
- Not Confident: 1.2%
RQ2: How accurately do developers classify the predominant paradigm in Python code?

More than 85% were confident!
RQ2: How accurately do developers classify the predominant paradigm in Python code?

Task 1 – Judgements vs Confidence Levels

Not Confident: 0    Slightly Confident: 1    Moderately Confident: 2    Confident: 3    Very Confident: 4
RQ2: How accurately do developers classify the predominant paradigm in Python code?

Task 1 – Judgements vs Confidence Levels

Not Confident: 0  Slightly Confident: 1  Moderately Confident: 2  Confident: 3  Very Confident: 4
RQ2: How accurately do developers classify the predominant paradigm in Python code?

Task 1 – Judgements vs Confidence Levels

Not Confident: 0  Slightly Confident: 1  Moderately Confident: 2  Confident: 3  Very Confident: 4
RQ3: Do developers fixate their gaze on specific Python language features when classifying predominant paradigms?

Fixations of four participants for **functional** task classification
RQ3: Do developers fixate their gaze on specific Python language features when classifying predominant paradigms?

Gazes on all **mixed** task token types by all participants
Discussion

Comparing fixations for Procedural paradigm (Task 1)
Comparing fixations for Procedural paradigm (Task 1)

Incorrect judgement

```
def_encrypt(string):
    a = string
    new_string = ''
    for x in a:
        new_string = new_string + chr(int(x))
    return new_string
```

Correct judgement

```
def_encrypt(string):
    a = string
    new_string = ''
    for x in a:
        new_string = new_string + chr(int(x))
    return new_string
```

“... def() I think I chose to be functional...”

“.. I saw def is a function so procedural. But also, for loops which are more functional, so I saw a lot of functional going on inside also”
Results: Task Category 2 (Bug Localization)
RQ4: Does the predominant paradigm affect how long developer’s take to debug logical errors?

Time taken for Bug Localization (Task Category 2)
RQ4: Does the predominant paradigm affect how long developer’s take to debug logical errors?

"I went through the for loop if it is calculating properly because we have to get the factorial of but here factorial is initialized to zero which is going to give zero for all iterations."

"The functional paradigms are harder for me to understand when I’m debugging, because functional paradigms quickly change objects so fast."

Time taken for Bug Localization (Task Category 2)
RQ4: Does the predominant paradigm affect how long developer’s take to debug logical errors?

Confidence Levels for Logical Debugging (Task 2)

More than 50% were confident with debugging!
RQ5: Does the predominant paradigm affect a developer’s ability to debug logical errors?

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-Oriented</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>Procedural</td>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>Functional</td>
<td>23</td>
<td>6</td>
</tr>
<tr>
<td>Mixed</td>
<td>19</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>Factorial</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>Largest</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>Palindrome</td>
<td>25</td>
<td>4</td>
</tr>
</tbody>
</table>

Effect of paradigm on correctness and debugging
Discussion

Comparing fixations for **Factorial mixed** paradigm (Task Category 2)

Correct Judgment

```python
1 # Find a logical bug in the code below
2 # The following code provides a factorial of a number
3 import sys
4 n = int(sys.argv[1])
5 factorial = 1
6 for i in range(1, n + 1):
7     factorial = factorial * i
8 print(f'The factorial of {n} is {factorial}')
```

Incorrect Judgment

```python
1 # Find a logical bug in the code below
2 # The following code provides a factorial of a number
3 import sys
4 n = int(sys.argv[1])
5 factorial = 0
6 for i in range(1, n + 1):
7     factorial = factorial * i
8 print(f'The factorial of {n} is {factorial}')
```
CONTRIBUTIONS
AND
CONCLUSION
CONTRIBUTIONS AND CONCLUSION
CONTRIBUTIONS AND CONCLUSION
CONTRIBUTIONS AND CONCLUSION

```python
1 import numpy as np
2 tansig = lambda n: 2 / (1 + np.exp(-2 * n)) - 1
3 sigmoid = lambda n: 1 / (1 + np.exp(-n))
4 hardlim = lambda n: 1 if n >= 0 else 0
5 purelin = lambda n: n
6 relu = lambda n: np.fmax(0, n)
7 square_error = lambda x, y: np.sum(0.5 * (x - y)**2)
8 sig_prime = lambda z: sigmoid(z) * (1 - sigmoid(z))
9 relu_prime = lambda z: relu(z) * (1 - relu(z))
10 softmax = lambda n: np.exp(n) / np.sum(np.exp(n))
11 softmax_prime = lambda n: softmax(n) * (1 - softmax(n))
12 cross_entropy = lambda x, y: -np.dot(x, np.log(y))
```
import numpy as np

tanh = lambda n: 2 / (1 + np.exp(-2 * n)) - 1
sigmoid = lambda n: 1 / (1 + np.exp(-n))
hardlim = lambda n: 1 if n >= 0 else 0
purelin = lambda n: n
relu = lambda n: np.fmax(0, n)
square_error = lambda x, y: np.sum(0.5 * (x - y)**2)
sig_prime = lambda z: sigmoid(z) * (1 - sigmoid(z))
relu_prime = lambda z: relu(z) * (1 - relu(z))
softmax = lambda n: np.exp(n)/np.sum(np.exp(n))
softmax_prime = lambda n: softmax(n) * (1 - softmax(n))
cross_entropy = lambda x, y: -np.dot(x, np.log(y))