Java to Python Compiler

Group 3: Jinhan Bao, Yicheng Xu

Overview

- Motivations & Goals
- Compiler Components
- Features
- Optimizations & Plan
- Conclusions

Motivations & Goals

Motivation:

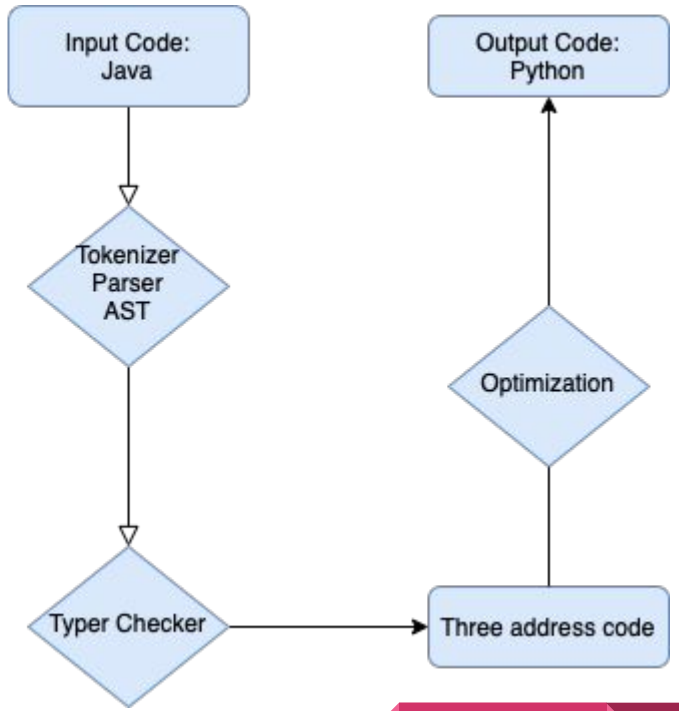
- Help new programmers learn Java when they already know Python

Goals:

- Translate different data types, classes, and functions in Java to corresponding Python codes
- Make the target code look nice and clean
- Translated codes should be executable with Python3

Compiler Components

1. Scanner converts Java to Tokens
2. Parser feeds these tokens into appropriate AST methods and AST constitutes a Tree with these tokens
3. Typechcker ensures values in the AST match our expectations and raise Error if necessary
4. IR generator takes typechecked AST and translates it into Three Address Code
5. Python code generator translates IR to Python code



Design & Difficulty

Design:

- Store every statements in input that use paranthesis as a dictionary
  - e.g: class & method declaration, loops, if/else, etc

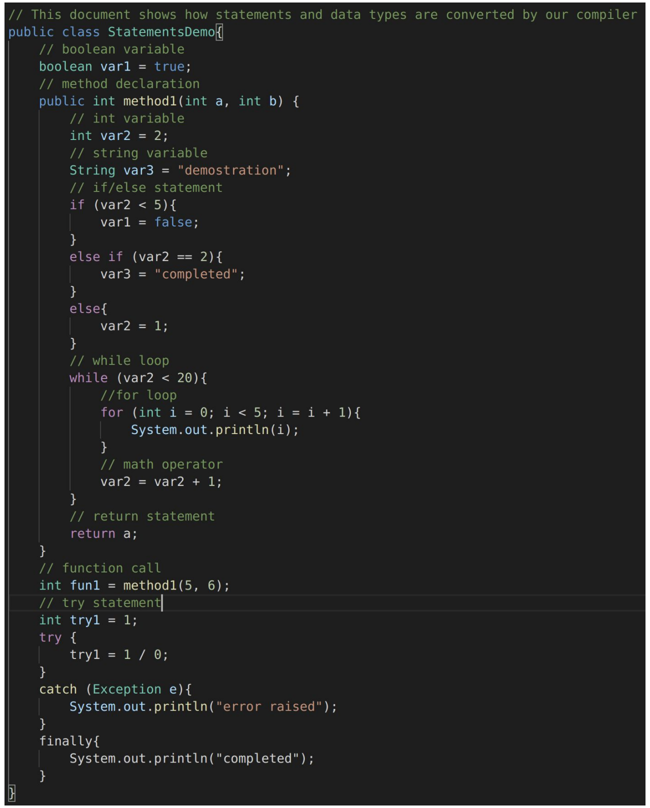
Major Difficulty:

- delete earlier blocks of code in the optimization process

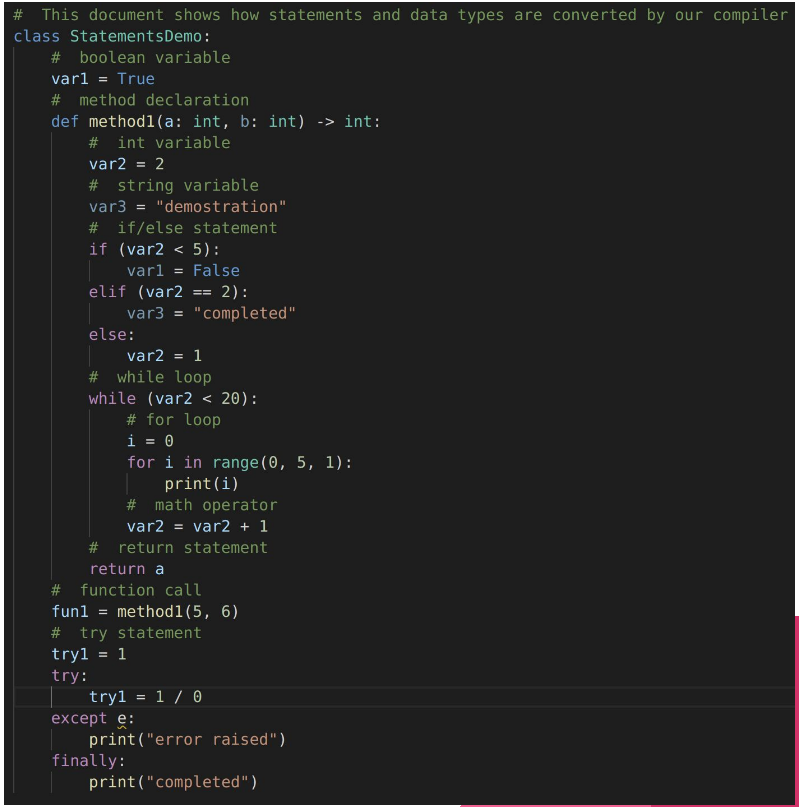
Features

- Data types (int, float, boolean, string, ArrayList)
- If/else
- For/While loops
- Try/catch
- Methods Declaration
- OOP
- Comment & Print

Data Types & Statements

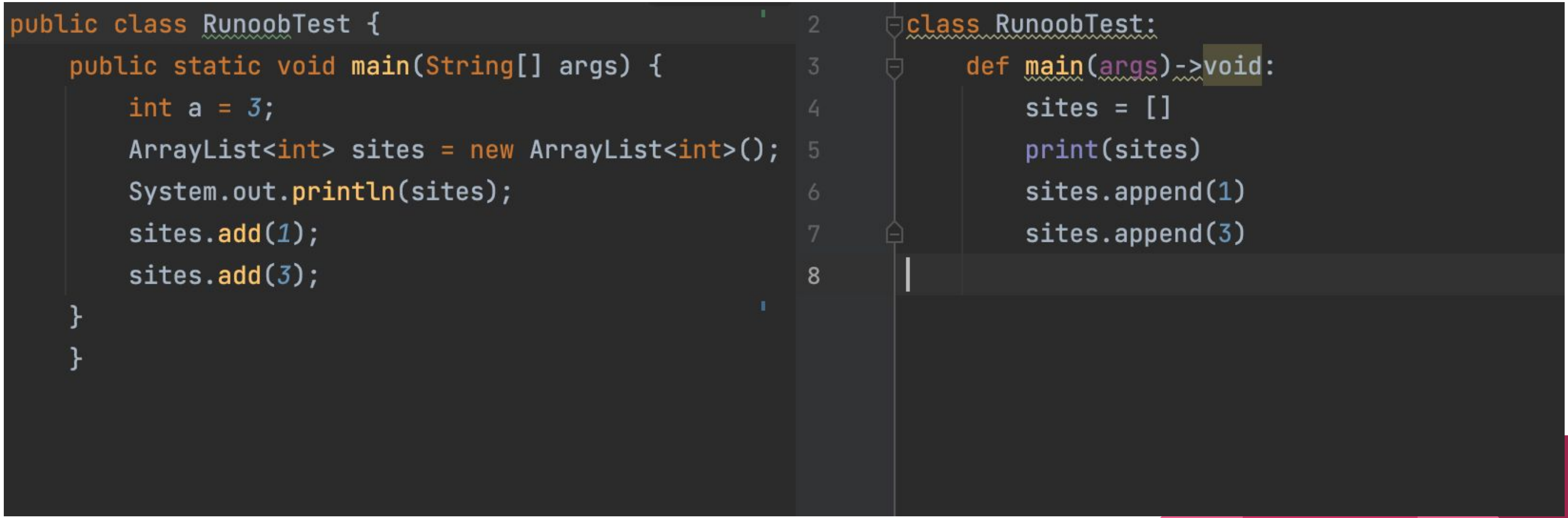


```java
// This document shows how statements and data types are converted by our compiler
public class StatementsDemo{
    // boolean variable
    boolean var1 = true;
    // method declaration
    public int method1(int a, int b) {
        // int variable
        int var2 = 2;
        // string variable
        String var3 = "demostration";
        // if/else statement
        if (var2 < 5){
            var1 = false;
        }
        else if (var2 == 2){
            var3 = "completed";
        }
        else{
            var2 = 1;
        }
        // while loop
        while (var2 < 20){
            //for loop
            for (int i = 0; i < 5; i = i + 1){
                System.out.println(i);
            }
            // math operator
            var2 = var2 + 1;
        }
        // return statement
        return a;
    }
    // function call
    int fun1 = method1(5, 6);
    // try statement
    int try1 = 1;
    try {
        try1 = 1 / 0;
    }
    catch (Exception e){
        System.out.println("error raised");
    }
    finally{
        System.out.println("completed");
    }
}
```



```python
#  This document shows how statements and data types are converted by our compiler
class StatementsDemo:
    #  boolean variable
    var1 = True
    #  method declaration
    def method1(a: int, b: int) -> int:
        #  int variable
        var2 = 2
        #  string variable
        var3 = "demostration"
        #  if/else statement
        if (var2 < 5):
            var1 = False
        elif (var2 == 2):
            var3 = "completed"
        else:
            var2 = 1
        #  while loop
        while (var2 < 20):
            # for loop
            i = 0
            for i in range(0, 5, 1):
                print(i)
            #  math operator
            var2 = var2 + 1
        #  return statement
        return a
    #  function call
    fun1 = method1(5, 6)
    #  try statement
    try1 = 1
    try:
        try1 = 1 / 0
    except e:
        print("error raised")
    finally:
        print("completed")
```

Array

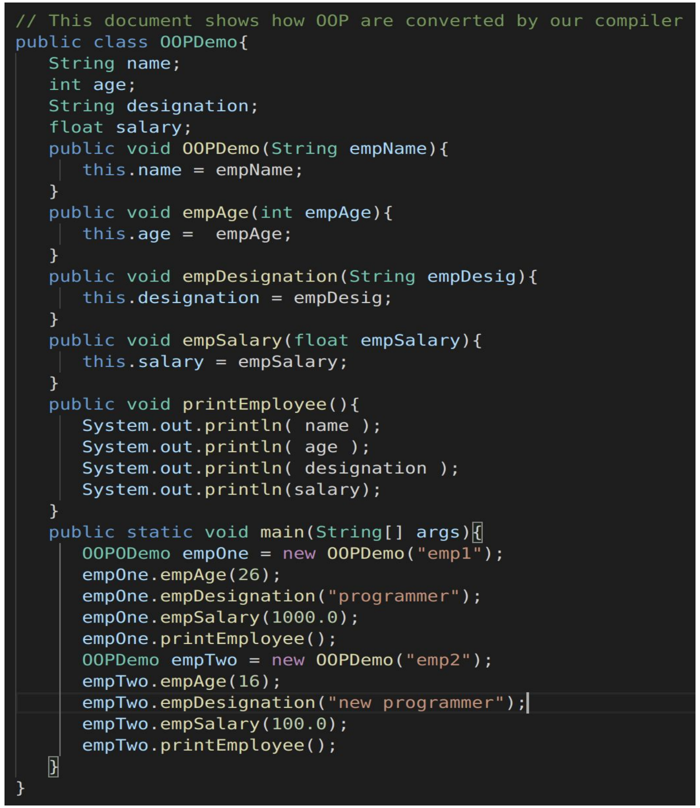


```java
public class RunoobTest {
    public static void main(String[] args) {
        int a = 3;
        ArrayList<int> sites = new ArrayList<int>();
        System.out.println(sites);
        sites.add(1);
        sites.add(3);
    }
    }
```

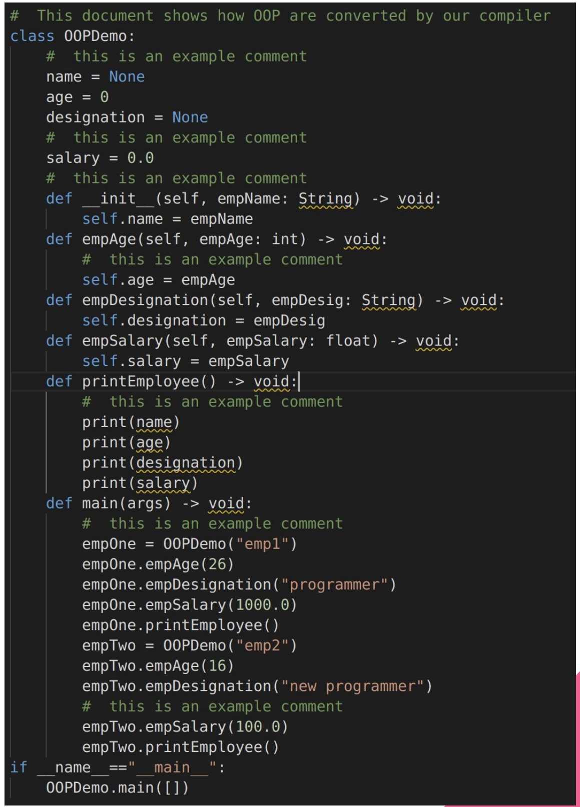
```python
class RunoobTest:
    def main(args)->void:
        sites = []
        print(sites)
        sites.append(1)
        sites.append(3)

```

OOP



```java
// This document shows how OOP are converted by our compiler
public class OOPDemo{
    String name;
    int age;
    String designation;
    float salary;
    public void OOPDemo(String empName){
        this.name = empName;
    }
    public void empAge(int empAge){
        this.age =  empAge;
    }
    public void empDesignation(String empDesig){
        this.designation = empDesig;
    }
    public void empSalary(float empSalary){
        this.salary = empSalary;
    }
    public void printEmployee(){
        System.out.println( name );
        System.out.println( age );
        System.out.println( designation );
        System.out.println(salary);
    }
    public static void main(String[] args){
        OOPODemo empOne = new OOPDemo("emp1");
        empOne.empAge(26);
        empOne.empDesignation("programmer");
        empOne.empSalary(1000.0);
        empOne.printEmployee();
        OOPDemo empTwo = new OOPDemo("emp2");
        empTwo.empAge(16);
        empTwo.empDesignation("new programmer");
        empTwo.empSalary(100.0);
        empTwo.printEmployee();
    }
}
```



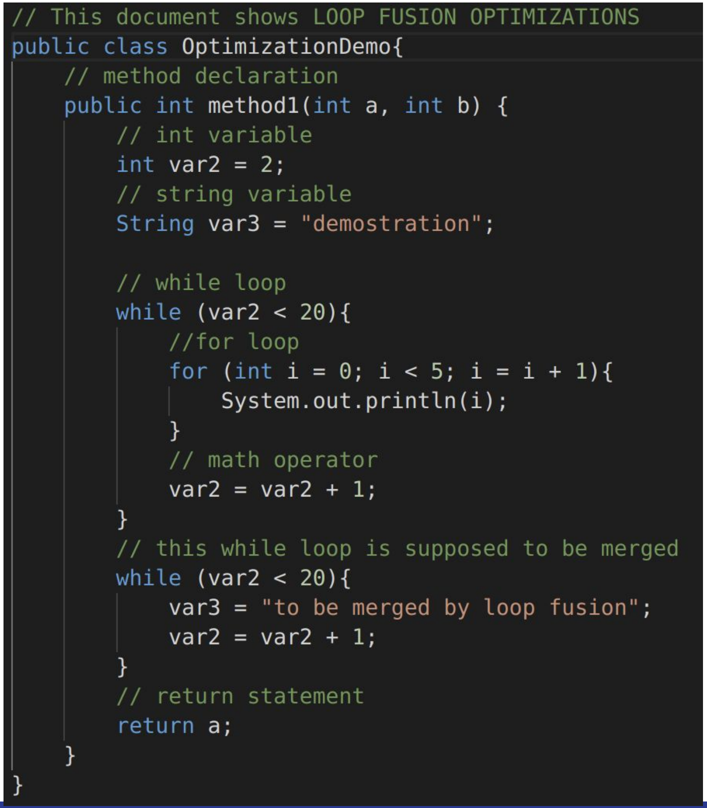
```python
#  This document shows how OOP are converted by our compiler
class OOPDemo:
    #  this is an example comment
    name = None
    age = 0
    designation = None
    #  this is an example comment
    salary = 0.0
    #  this is an example comment
    def __init__(self, empName: String) -> void:
        self.name = empName
    def empAge(self, empAge: int) -> void:
        #  this is an example comment
        self.age = empAge
    def empDesignation(self, empDesig: String) -> void:
        self.designation = empDesig
    def empSalary(self, empSalary: float) -> void:
        self.salary = empSalary
    def printEmployee() -> void:
        #  this is an example comment
        print(name)
        print(age)
        print(designation)
        print(salary)
    def main(args) -> void:
        #  this is an example comment
        empOne = OOPDemo("emp1")
        empOne.empAge(26)
        empOne.empDesignation("programmer")
        empOne.empSalary(1000.0)
        empOne.printEmployee()
        empTwo = OOPDemo("emp2")
        empTwo.empAge(16)
        empTwo.empDesignation("new programmer")
        #  this is an example comment
        empTwo.empSalary(100.0)
        empTwo.printEmployee()
if __name__=="__main__":
    OOPDemo.main([])
```

Optimizations

completed so far:

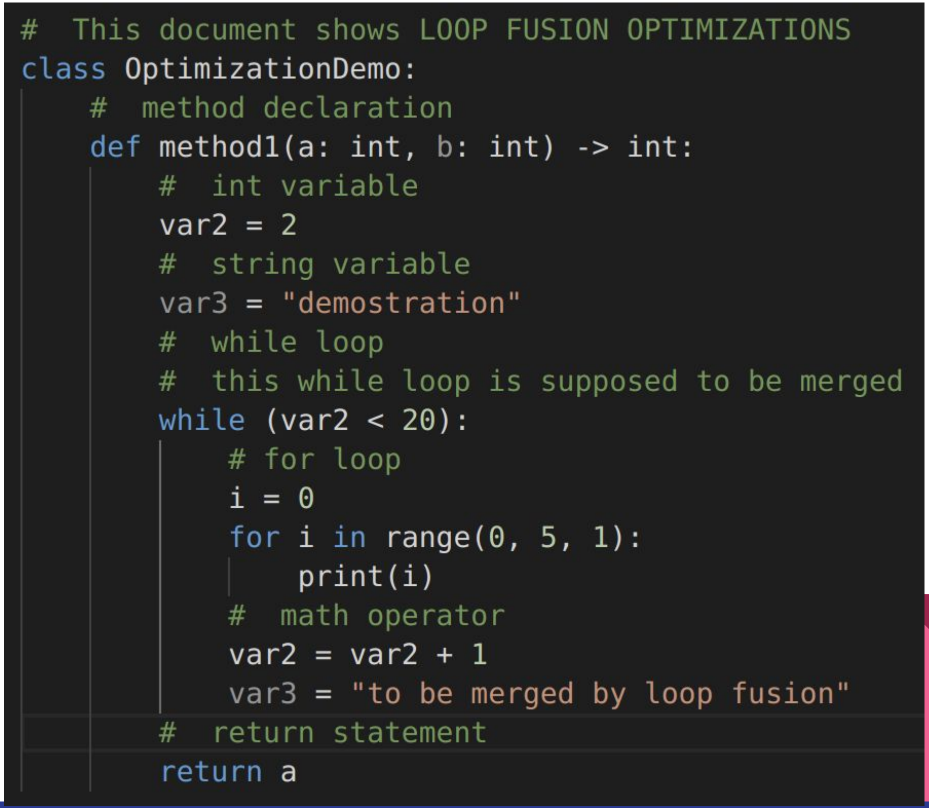
- Loop Fusion (merge while loops with same condition)
- Dead Code Elimination (delete defined but not used variables)

Loop Fusion



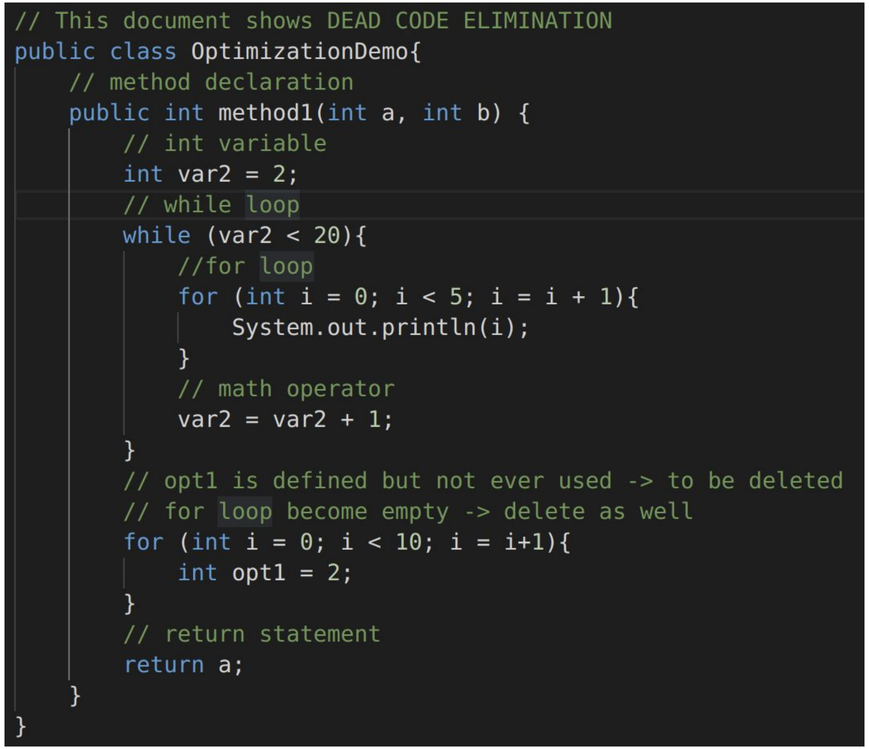
```java
// This document shows LOOP FUSION OPTIMIZATIONS
public class OptimizationDemo{
    // method declaration
    public int method1(int a, int b) {
        // int variable
        int var2 = 2;
        // string variable
        String var3 = "demostration";

        // while loop
        while (var2 < 20){
            //for loop
            for (int i = 0; i < 5; i = i + 1){
                System.out.println(i);
            }
            // math operator
            var2 = var2 + 1;
        }
        // this while loop is supposed to be merged
        while (var2 < 20){
            var3 = "to be merged by loop fusion";
            var2 = var2 + 1;
        }
        // return statement
        return a;
    }
}
```

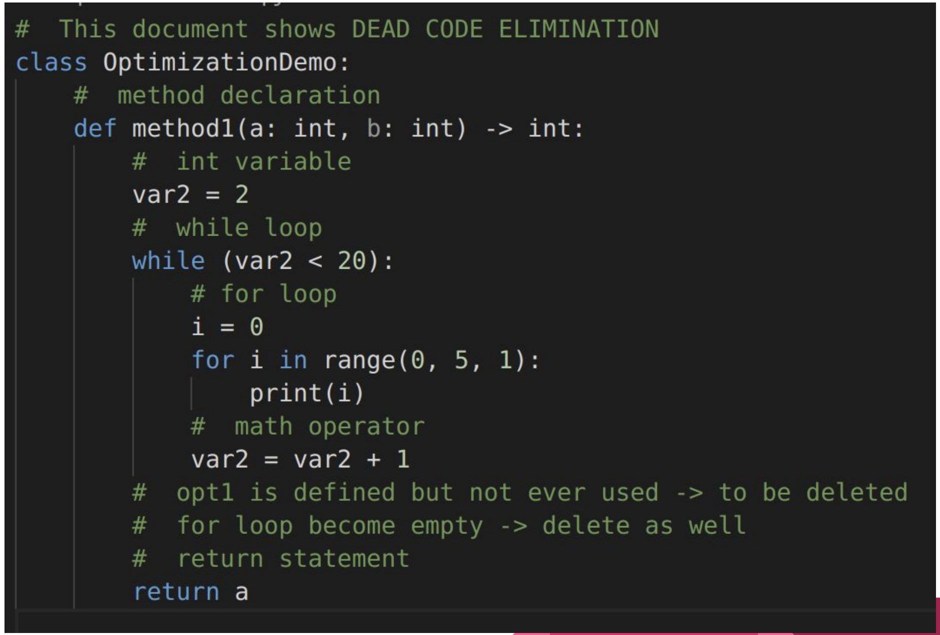


```python
#  This document shows LOOP FUSION OPTIMIZATIONS
class OptimizationDemo:
    #  method declaration
    def method1(a: int, b: int) -> int:
        #  int variable
        var2 = 2
        #  string variable
        var3 = "demostration"
        #  while loop
        #  this while loop is supposed to be merged
        while (var2 < 20):
            # for loop
            i = 0
            for i in range(0, 5, 1):
                print(i)
            #  math operator
            var2 = var2 + 1
            var3 = "to be merged by loop fusion"
        #  return statement
        return a
```

Dead Code Elimination



```java
// This document shows DEAD CODE ELIMINATION
public class OptimizationDemo{
    // method declaration
    public int method1(int a, int b) {
        // int variable
        int var2 = 2;
        // while loop
        while (var2 < 20){
            //for loop
            for (int i = 0; i < 5; i = i + 1){
                System.out.println(i);
            }
            // math operator
            var2 = var2 + 1;
        }
        // opt1 is defined but not ever used -> to be deleted
        // for loop become empty -> delete as well
        for (int i = 0; i < 10; i = i+1){
            int opt1 = 2;
        }
        // return statement
        return a;
    }
}
```



```python
#  This document shows DEAD CODE ELIMINATION
class OptimizationDemo:
    #  method declaration
    def method1(a: int, b: int) -> int:
        #  int variable
        var2 = 2
        #  while loop
        while (var2 < 20):
            # for loop
            i = 0
            for i in range(0, 5, 1):
                print(i)
            #  math operator
            var2 = var2 + 1
        #  opt1 is defined but not ever used -> to be deleted
        #  for loop become empty -> delete as well
        #  return statement
        return a
```

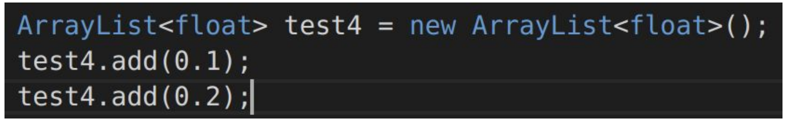
Optimization Demo

Plans & Conclusions

Optimization:

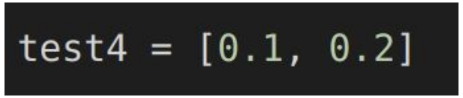
- ArrayList representation in Python

Input:



```
ArrayList<float> test4 = new ArrayList<float>();
test4.add(0.1);
test4.add(0.2);
```

Output:



```
test4 = [0.1, 0.2]
```

Features Planned:

- Develop Hashmap (Java) -> Dictionary (Python)
- Hashmap methods extensions(put, get, remove, clear)

Conclusions:

Overall, major developments are done and we will focus on optimizations and debugging before the deadline. We are providing users as much necessary features as possible to give them a better experience.

Thanks for Watching