SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

A SIMPLE JAVA-PYTHON CONVERTER

PREPARED BY:

EMAN JAMMAA COCO

HADEEL ALI OSMAN

SUPERVIOR: Dr. NIEMAH IZZELDIN

OCTOBER 2017

A THESIS SUBMITTED AS A PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF B.Sc. (HONORS) IN COMPUTER SYSTEMS AND NETWORKS

بسم الله الرحمن الرحيم

SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY

FACULTY OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

DEPARPMENT OF COMPUTER SYSTEMS AND NETWORK

A SIMPLE JAVA-PYTHON CONVERTER

PREPARED BY:

EMAN JAMMAA COCO

HADEEL ALI OSMAN

SUPERVIOR: Dr. NIEMAH IZZELDIN

A THESIS SUBMITTED AS A PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF B.Sc. (HONORS) IN COMPUTER SYSTEMS AND NETWORKS

SIGNITURE OF SUPERVISOR: …………　　　　DATE: …... OCTOBER 2017

الحمد لله

الحمد لله في سري وفي علني والحمد لله في حزني وفي سعدي الحمد لله على إحسانه وله الشكر على توفيقه و إمتنانه الحمد لله على ما أراد بنا من عاجل الخير و آجله

الحمد لله الذي علم بالقلم علم الإنسان ما لم يعلم وأشهد أن لا إله إلا الله وحده لا شريك له و أشهد أن محمد عبده ورسوله

الآية

(وَقُلِ اعْمَلُوا فَسَيَرَى اللَّهُ عَمَلَكُمْ وَرَسُولُهُ وَالْمُؤْمِنُونَ ۖ وَسَتُرَدُّونَ إِلَىٰ عَالِمِ الْغَيْبِ وَالشَّهَادَةِ فَيُنَبِّئُكُمْ بِمَا كُنْتُمْ تَعْمَلُونَ) سورة التوبة - الآية 105

(وآتَاكُم مِّن كُلِّ مَا سَأَلْتُمُوهُ ۚ وَإِن تَعُدُّوا نِعْمَتَ اللَّهِ لَا تُحْصُوهَا ۗ إِنَّ الْإِنسَانَ لَظَلُومٌ كَفَّارٌ)

سورة ابراهيم - الآية 34

(قَالَا رَبَّنَا إِنَّنَا نَخَافُ أَنْ يَفْرُطَ عَلَيْنَا أَوْ أَنْ يَطْغَى (45) قَالَ لَا تَخَافَا ۖ إِنَّنِي مَعَكُمَا أَسْمَعُ وَأَرَىٰ)

سورة طه - الآية 46

الإهداء

إلي من لا يمكن للكلمات أن توفي حقهما إلي من لا يمكن للأرقام أن تحصي فضلهما إلي والدي أدامهما الله تاجا علي رأسنا.

إلي إخوننا و أخواتنا الذين كانوا خير عونا لنا في هذه الحياة.

إلي كل معلم و معلمة ساهموا في تعليمنا طوال هذه السنوات الدراسية من أول خطواتنا الى مرحلة التخرج بعون الله.

إلي صديقاتنا وأخواتنا اللاتي كن لنا خير عون طوال هذه السنوات الدراسية.

إلي دفعتنا الغالية الدفعة العاشرة قسم نظم الحاسوب والشبكات التي كانت خير عائلة لنا في هذه السنوات الدراسية الجامعية.

إلي كل من سقط القلم عن ذكرهم.

الشكر والعرفان

نتوجه بجزيل الشكر والإمتنان إلي كل من ساعدنا من قريب أو بعيد علي إنجاز هذا العمل وفي تذليل ما واجهناه من صعاب. ونخص بالشكر الإنسانة الفاضلة الدكتورة نعمة عز الدين التي لم تبخل علينا بتوجيهاتها ونصائحها القيمة التي كانت عوناً لنا في إكمال هذا البحث بفضل الله تعالي.

المستخلص

تعتبر لغة الجافا ولغة البايثون من لغات البرمجة الأكثر شعبية وقوة في الوقت الحاضروكلاهما من لغات البرمجة الموجهة للكائنات. وتتمتع كلتا اللغتين بمزايا فريدة للمطورين والمستخدمين.

منذ سنوات عديدة تحولت الكثير من المؤسسات التعليمية من إستخدام لغات C و ++C (أو باسكال، الخ) إلى لغة الجافا لتدريس مقررات البرمجة. وبالنظر إلي مزايا لغة بايثون وإرتباطها بالإتجاهات الجديدة في علوم الحاسوب مثل إنترنت الأشياء، يتوقع أن تحل بايثون محل لغة جافا في تعليم مقررات البرمجة في المستقبل القريب.

في حال أن رغب المبرمج أو الدارس تحويل برامجه من لغة جافا إلى بايثون لكسب ميزاتها، سيكون عليه إعادة كتابة البرنامج بأكمله من البداية، والذي يعد إضاعة للجهد والوقت.

الحل المقترح لهذه المشكلة هو محول يقوم بتحويل لغة جافا إلي لغة بايثون وهو عبارة عن تطبيق لسطح المكتب يأخذ كود جافا كمدخل ويقوم بتحويله إلى كود بايثون كمخرج دون الإضطرار لإعادة كتابة برنامج بايثون بأكمله.

ABSTRACT

Java and Python are two of the most popular and powerful programming languages of present time. Both of them are object-oriented programming languages. Both languages have their unique advantages for developers and end users.

Many years ago a lot of educational institutions switched from C and C++ or Pascal to Java for introducing programming courses. Given the features of Python and how it is related to emerging fields in computer science such as an Internet of things, it is expected that Python will replace Java in teaching programming languages in the near future.

If a programmer or student wants to convert their software programs from Java to Python to gain its features, he will have to rewrite the whole program from start, which is considered a waste of time and effort.

The suggested solution to solve this problem is using a Simple Java-Python converter, which is a desktop application that takes Java code as an input and converts it to Python code as an output without having to rewrite the whole program from start.

List of Abbreviations:

Abbreviation	Expression
AST	Abstract Syntax Tree
BCX	The BASIC to C translator
BNF	Backus–Naur form
DLL	Dynamic Link Libraries
DOM	Document Object Model
GUI	Graphical User Interface
IDLE	Python's Integrated Development and Learning Environment
JIT	Just-In-Time
JLCA	Java Language Conversion Assistant
JVM	Java Virtual Machine
Jython	Python for the Java Platform
OOP	Object Oriented Programming
Parse::RecDescent	Recursive descent parser
Perthon	Python-to-Perl Source Translator
RMI	Remote Method Invocation
VB	Visual Basic
XML	eXtensible Markup Language

Index of Figures:

Index of Tables:

Table of Contents

Chapter 1 INTRODUCTION

1.1 Overview

Java and Python are two of the most popular and powerful programming languages of present time. Both of them are object-oriented programming languages. Both languages have their unique advantages for developers and end users.

Many years ago a lot of educational institutions switched from C and C++ (or Pascal, etc) to Java for introducing programming courses. Given the features of Python and how it is related to emerging fields in computer science such as Internet of things, it is expected that Python will replace Java in teaching programming languages in the near future.

Python is an easier language for novice programmers to learn. One can progress faster if learning Python as a first language instead of Java, because Java is restrictive and more complex compared to Python.

Python is more user-friendly, easier to read and understand than Java, as it has a more intuitive coding style and uses whitespaces to convey the beginning and end of blocks of code.

Python is a more productive language than Java because it is a dynamically typed programming language whereas java is statically typed.

1.2 Problem Statement

If programmers want to convert their software programs from Java to Python to gain its features, they will have to rewrite the whole program from start which consumes time and increases cost.

1.3 Proposed Solution

To reduce effort, time and cost that is consumed to convert programs from Java to Python, we need a mechanism that converts them automatically.

The conversion process has been placed among the top 10 challenges in the programming world [13]. Achieving the maximum efficiency of the conversion without compromising the quality of the converted program is the programmer's target.

A Simple Java-Python Converter is a software that takes a Java file code as input and converts it to Python file code as output as shown in Figure 1.1.

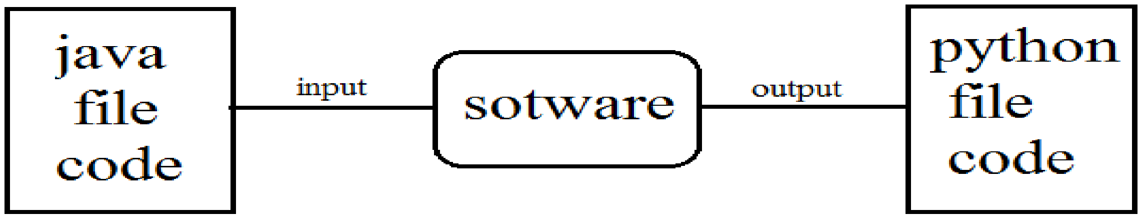


Figure 1.1 A Simple Java-Python Converter

1.4 Objectives:

The objectives of this project are to:

- Prevent the loss of programs that are written in java.
- Use old programs to produce the newer.
- Reduce the software evolution cost.
- Help Java programmers to learn Python.
- Help in switching from Java to Python.

1.5 Scope:

The Simple Java-Python Converter converts a program written in Java to a Python program. The Java program must be free of syntax.

The software covers the Java principles:

- Class declaration.
- Method declaration.

- Comments.
- Declaring and initializing primitive, floating point and boolean variables.
- Selection statements:
    - If statement.
    - If…else statement.
    - Nested if statement.
- Switch statement.
- Iteration Statements:
    - While statement.
    - Do…while statement.
    - For statement.

1.6 Methodology:

The Simple Java-Python Converter reads Java statements from the Java program, converts them to XML (eXtensible Markup Language) tags as an intermediate code and writes them in an .xml file. It then reads XML tags, converts them to Python statements and writes them in a Python file.

1.7 Thesis Organization:

This thesis contains six chapters. Chapter two explains Java and Python programming languages and compares the two languages. Chapter three explains language processors (compiler and interpreter) and reviews the literature review. The system design is implemented in Chapter four. Chapter five demonstrates results and provides execution examples. Finally, Chapter six is conclusions and recommendations.

Chapter 2 PROGRAMMING LANGUAGES

2.1 Introduction

This chapter explains in details Java and Python programming languages including structure, data types and statements. It also compares the two languages.

2.2 Programming Language

Java programming language is used to develop the Simple Java-Python Converter desktop application. It is used to develop both phases of the conversion process.

Java programming language is a popular programming language and it is used to develop all kinds of applications. Such as mobile, client-server, web and desktop applications.

Java is a highly portable language as it must be executed through a cross-platform compatible Java Virtual Machine (JVM).

Java is a statically-typed language, which means the code will have to be checked for errors before it can be built into an application.

As a statically typed language, Java is faster than dynamically typed languages because things are more clearly defined.

Performance can be optimized in real time to help a Java program run faster. This is very helpful as some applications grow larger or need to handle more processes.

2.3 Java

2.3.1 Java history

Like the successful computer languages that came before, Java is a blend of the best elements of its rich heritage combined with the innovative concepts required by its unique mission. Computer language innovation and development occurs for two fundamental reasons:

- To adapt to changing environments and uses.
- To implement refinements and improvements in the art of programming.

The development of Java was driven by both these elements in nearly equal measure [1].

Java is related to C++, which is a direct descendant of C. From C, Java derives its syntax. Many of Java's object-oriented features were influenced by C++. Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to develop the first working version. This language was initially called "Oak," but was renamed "Java" in 1995[1].

The primary motivation was the need for a platform-independent (that is, architecture-neutral) language that could be used to create software to be embedded in various consumer electronic devices, such as microwave ovens and remote controls. With the emergence of the World Wide Web, Java was propelled to the forefront of computer language design, because the Web, too, demanded portable programs.

2.3.2 Java Virtual Machine (JVM)

In other language the compiler coverts source code into executable code. However, in java the output of a Java compiler is not executable code, it is bytecode. Bytecode is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM) (an interpreter for bytecode) [1]. Although the JVM differs from platform to another, all of them understand the same bytecode and this supports portability features.

Because the JVM is in control, it can contain the program and prevent it from generating side effects outside of the system [1].

In general executing code in two stages (bytecode, executable code) is slower than executing it in one stage (executable code only). But Java is not slow as expected because the bytecode is highly optimized which thus JVM executes it in a rapid manor.

Sun supply’s its technology HotSpot, which provides a Just-In-Time (JIT) compiler for bytecode.

The JIT is a part of the JVM which takes a part of the bytecode depending on the demand and executes it.

It is not suitable to execute source code all at once, because there are some checks done during run time only.

JIT compiler compiles only the part of code that will benefit from compilation and interprets the rest.

2.3.3 Characteristic of Java

2.3.3.1 Object-oriented

Object-oriented programming (OOP) is at the core of Java. One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism [1].

2.3.3.2 Robust

Java helps in finding mistakes in the java program by determining the area of the error, java checks code in compile time and run time.

2.2.3.3 Multithreaded

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows writing programs that run simultaneously [1].

2.3.3.4 Architecture-neutral

In general, the lifetime of the program is not guaranteed because the operating system and the processor are updated continually.

Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was “write once; run anywhere, anytime, forever” [1].

2.3.3.5 Distributed

Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. In fact, accessing a resource using a URL is not much different from accessing a file. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network [1].

2.3.4 Java structure

2.3.4.1 Java class

```
public class Welcome{
 public static void main(String[] args) {
                  System.out.println(“Welcome to java”);  }
}
```

Public: is a keyword indicating that the class can be seen or manipulated by all other classes.

Class: is a keyword to declare that a new class is being defined. Every java class must begin with it.

Welcome: is the class name, and must be the same as the file name (Welcome.java).

{}: the content of the java class must be between the {} braces, the { brace is the start of the java class and } brace I the end of the java class.

void: is a keyword indicating that the main method does not return a value. If the method returns a value, instead of void, the data type of the returned value is written.

main(): the execution of a java program begins from the main method. Without the main method the program can be compiled but cannot be executed. After the method name the () braces must be included. If the method receives parameters they are declared inside the () braces.

String[] args: the main method receives an array of string named args.

System.out.println(): System is a class in java that provides access to the system, out is the output stream that is connected to the console, and println is the method that prints the data that is passed to it followed by a new line.

2.3.4.2 Comments in Java

There are three types of comments in java:

- **Line comment:**

```
//this is a line comment
```

- **Paragraph comment**

```
/*	This is paragraph comment */
```

- **javadoc comment**

```
/**	This is javadoc comment
*/
```

2.3.4.3 Declaring variables in Java

Variables are used to store data in a program. Declaring a variable tells the compiler to allocate appropriate memory space for the variable based on its data type.

Syntax: datatype variable_Name= value;

2.3.4.4 Primitive data types

Java provides eight (8) primitive data types as shown in Table 2.1.

Table 2.1 Primitive Data Types in Java

Type	**Name**	**Size**
Numeric	Byte	8-bit signed
	Short	16-bit signed

	Int	32-bit signed
	Long	64-bit signed
Floating point	Float	32-bit
	Double	64-bit
Characters	Char	16-bit
Boolean	Boolean	true or false

2.3.4.5 Selection statements

- **If statement:**

```
if(condition) {
    Statements;
}
```

- **If else statement**

```
if(condition) {
Statements;
} else{
Statements; }
```

- **Nested if statement**

```
if(condition) {
    Statements;
}//end outer if
else {
        if{
                Statements;
              } //end inner if
        else{
                Statements;
        }//end inner else
}//end outer else
```

2.3.4.6 Switch statement

```
switch (expression)

{
case value1: statements;
                 break;
case value2: statements;
                 break;
default: statements;
}//end switch
```

The switch expression must hold a value of: char, byte, short, or int type.

2.3.4.7 Iteration statements

- **While statement:**

  ```
  while(condition) {
          // body of loop
  }
  ```

- **do-while statement**

  ```
  do {// body of loop
  } while (condition);
  ```

- **for statement**

  ```
  for(initialization; condition; iteration) {
  // body
  }
  ```

2.4 Python

2.4.1 Introduction to Python

Python programming language was created by Guido van Rossum in the late 1980s [2]. Python is a higher-level programming language. It is considered to be a higher-level language than C, C++, Java, and C#.

2.4.2 IDLE

Integrated Development and Learning Environment (IDLE) is a simple Python integrated development environment available for Windows, Linux, and MacOS X [2].

Python code can be written directly in IDLE shell to execute but will not be saved.

To save the python code it must be written in IDLE editor, for example (welcome.py).

The name of the python program is irrelevant but describes the nature of this program.

print("Welcome to python"): this statement prints Welcome to python on the screen.

2.4.3 Python structure

2.4.3.1 Operators

- Arithmetic Operators (+, -, *, /, **, //, %).
- Bitwise Operators (<<, >>, &, |, ~, ^).
- Relational Operators (<, >, <=, >=, ==,! =, <>).
- Logical Operators (and, or, not).

2.4.3.2 Declaring functions in Python

To declare a function in python the def keyword is used.

```
def name(arg1, arg2,... argN):
        #block
        return value
```

2.4.3.3 Comments in Python

Python uses the (#) symbol to indicate that this line is a comment and uses (''') in case of a multiple line comment, the compiler ignores it.

```
#  this line is a comment
'''
        this is a multiple line comment
'''
```

2.4.3.4 Values and variables

- To declare a variable in python, it doesn’t require determining its data type.
- Python by default determines the data type of the variable depending on its value.
- To know the data type of a variable the type (variable_Name) function is used.
- Syntax:
  - variable_Name = value

2.4.3.5 Boolean in Python

- Boolean take only two values True or False.
- In python 0 and "" indicate False and all other values indicate True.

2.4.3.6 Types of numbers

- In python version 2 there are four types of numbers: int, long, float, complex:
  - Int:-2147483648➔2147483647
  - Long:> 2147483647 and < -2147483648 (End with L symbol)
  - Float: contain floating point ex (2.5).
  - Complex: Imaginary numbers ex (2+5j).
- In Python version three the int and long types are merged in int type.

2.4.3.7 Declaration of a string

There are three ways to declare a string in Python:

1. Double quote "string in Python".
2. Single quote ' string in Python'.
3. Triple quote """string in Python contain quote""".

2.4.3.8 Selection statements

- **If statement**

```
    If condition:
    Statements
```

- **If else statement**

```
If condition:
    Statements
else:
    Statements
```

- **Nested if statement**

```
    If condition:
        Statements
    else:
           if condition:
              Statements
        else:
              Statements
```

2.4.3.9 Iteration statements

- **While statement:**

```
while condition:
    #block
```

- **for statement:**

```
for n in range(begin, end, step):
        # block
```

2.5 Java vs. Python

Table 2.2 compares Java and Python in terms of comments, variables, data types and statements.

Table 2.2 Java vs. Python

	Java	Python
Print statement	System.out.println("Welcome to java");	print("Welcome to python")
Comments	// line comment /* paragraph comment */ /** javadoc comment */	#comment
Declaring Variables	Datatype variable_Name = value;	variable_Name = value
Primitive Data Types	byte, short, int, long, float, double, char, boolean.	int, long, float, complex, boolean.

If statement	if(condition) {
	Statement;
	}
		If condition:
		Statements

If else statement	if(condition)
	{ Statement;
	} else {
	Statement;
	}
		If condition:
		Statements
		else:
		Statements

Nested if	if(condition)	If condition:

statement	{ Statement;
	}//end outer if
	else
	{
	if{
	Statement;
	} //end inner if
	else{
	Statement;
	}//end inner else
	}//end our else
		Statements
		else:
		if condition:
		Statements
		else:
		Statements

Switch statement	switch(expression)
	{
	case value1: statements;
	break;
	case value2: statements;
	break;
	default: statements;
	}//end switch
		Doesn’t have a switch statement

While statement	while(condition)
	{
	// body of loop
	}
		while condition:
		# block

for Statement	for(initialization; condition; iteration) {
	// body
	}
		for n in range (begin, end, step):
		# block

2.6 Summary

This chapter overviewed Java and Python programming languages in details including introduction to each language, structure and statements, and compared the two languages. The next chapter explains in details language processors including compilers and interpreters. It also reviews the literature review.

Chapter 3 LANGUAGE PROCESSORS AND LITERATURE REVIEW

3.1 Introduction

This chapter explains in details language processors (compilers and interpreter), compare the two language processors, Java and Python language processors and finally reviews the literature review.

All software used in computers is written in some programming language, but before executing the program it must be converted to another form that can be understood by the machine, as described in Figure 3.1.

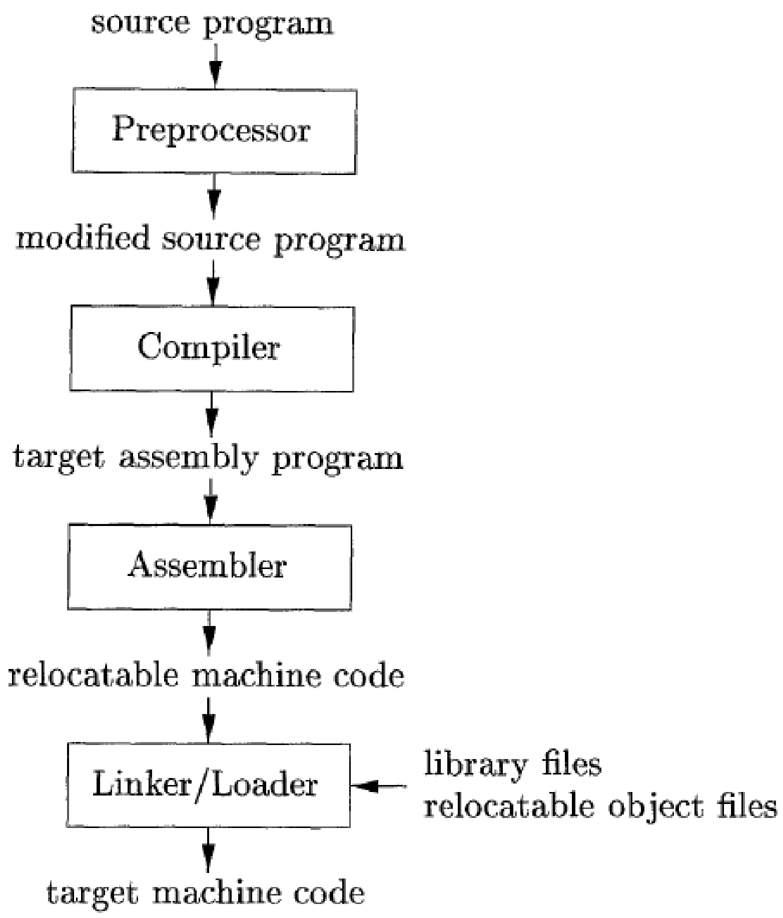


Figure 3.1 Language Processing

Language processing is achieved by four steps. Following is an explanation of each step.

Preprocessor: is concerned with the collection of the source program (which may be divided into separates files) and expand shorthand into source code statements.

Compiler: takes the modified program from the preprocessor as an input and then converts it to an assembly code (because it is easy to produce and easy to debug) as an output.

Assembler: takes the assembly program as an input and then converts it to a machine code.

Linker/Loader: the linker resolves external memory addresses, where the code in one file may refer to a location in another file. The loader then puts together all of the executable object files into memory for execution [3].

3.2 Compiler

A compiler is a program that can read a program in one language (the source language) and translate it into an equivalent program in another language (the target language) as can be seen in Figure 3.2 [3].

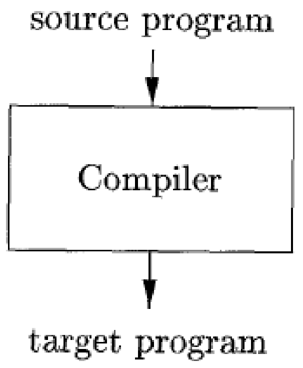


Figure 3.2 A Compiler

3.2.1 Structure of a compiler

The compiler consists of two parts: analysis and synthesis. The Analysis divides the source code into pieces, applies the grammatical structure to them and generates an intermediate representation of the source code. If the syntax of the source code is ill, it generates an informative message to the user. It also collects information about the source code and stores them on the symbol table. The synthesis part uses the symbol table and intermediate representation to generate the target program. Figure 3.3 illustrates the structure of a compiler [3].

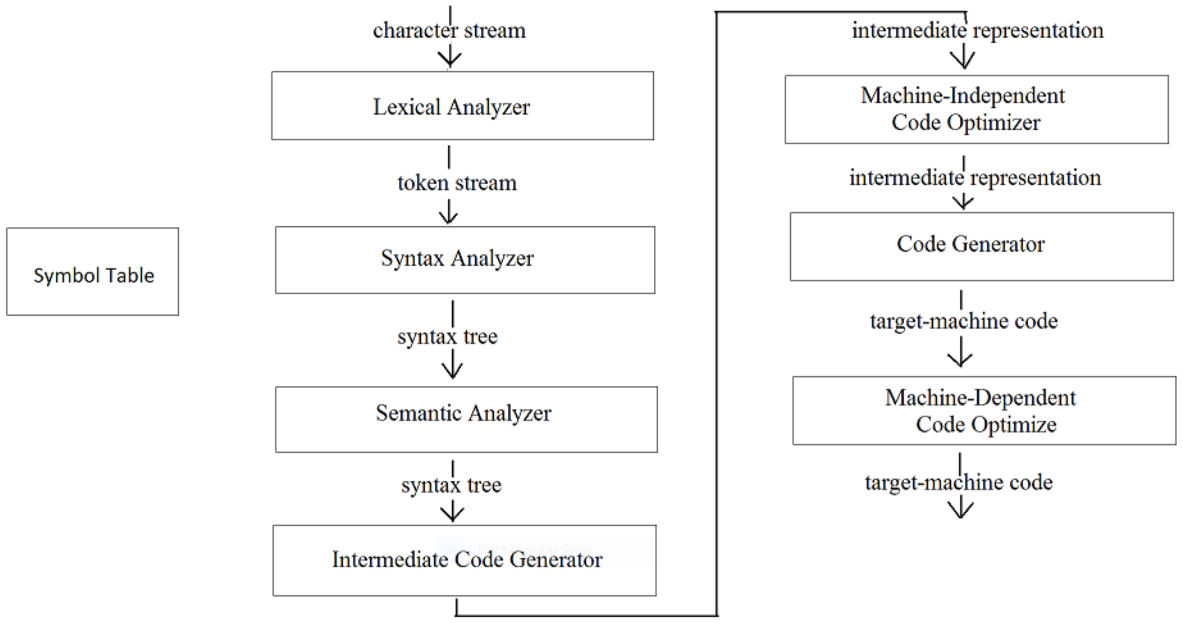


Figure 3.3 Structure of a Compiler

We will use a simple example to understand the job of each phase, using the following statement:

x = a * b/2

3.2.1.1 Lexical analyzer (scanning)

The lexical analyzer reads a stream of characters and groups them into a meaningful sequence called lexemes and produces tokens as an output, its syntax:

(Token name, attribute value)

Token name is an abstract symbol used during syntax analysis, and attribute value points to the token as in Table 3.1.

In our example the output tokens are:

(id, 1) (=) (id, 2) (*) (id, 3) (/) (2)

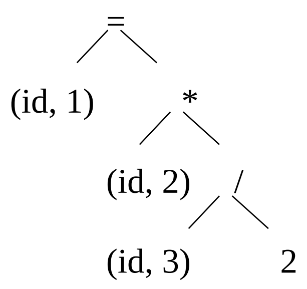
In case of operators and values we use its symbol as its token name.

Table 3.1 Symbol Table

1	X	
2	A	
3	B	

3.2.1.2 Syntax analyzer (parsing)

In this phase, the syntax tree is created from the tokens.



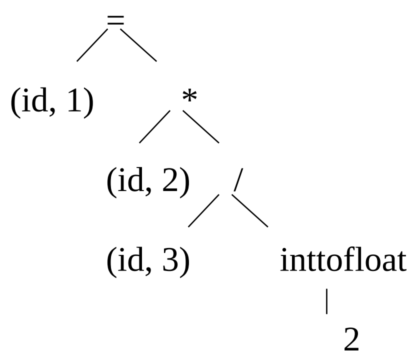
3.2.1.3 Semantic analyzer

The semantic analyzer uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition [3].

It also:

- Gathers type information

- Type checking: checks that each operator has matching operands
- Reports errors
- Type conversion: suppose that x, a and b are floating point and (2) is an integer then the 2 is converted to floating point number:



3.2.1.4 Intermediate code generator

Which generates an intermediate representation (program for an abstract machine), that must be easy to produce and easy to translate.

The output of this phase is three-address codes which consist of a sequence of assembly like instructions with three operands per instruction. Each operand can act as a register [3]. With respect to three-address code:

- Each instruction has at most one operator on the right side.
- The compiler must generate a temporary name to hold the result of three-address instruction.

```
t1 = inttofloat(2)
t2 = id3 / t1
t3 = id2 * t2
id1 = t3
```

3.2.1.5 Code optimization

This phase is optional, and is used to improve the intermediate code to be:

- Faster
- Shorter
- Consume less power

```
t1 = id3 / 2.0
id1 = id2 * t1
```

3.2.1.6 Code generation

Maps the intermediate representation to the target language. In this phase registers or memory locations are selected.

```
LDF   R2, id3
DIVF  R2,    R2,    #2.0
LDF   R1,    id2
MULF         R1,    R1,    R2
STF   id1,   R1
```

3.2.1.7 Symbol table

Is a data structure which contains records for source program variables and provide information about them such as:

- Storage allocated
- Type
- Variable scope

In case of procedures it stores information about its name, the type of its arguments, the method of passing and type returned.

3.3 Interpreters

An interpreter is another common kind of language processors that directly executes the source program on user inputs as shown in Figure 3.4. The task of an interpreter is more or less the same as of a compiler but the interpreter works in a different fashion. The interpreter takes a single line of code as input at a time and executes that line. It will terminate the execution of the code as soon as it finds an error. Memory requirement is less because no object code is created [3].

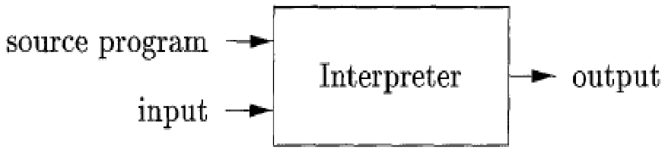


Figure 3.4 An Interpreter

The machine language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

3.4 Compilers vs. Interpreters:

Table 3.2 compares compilers and interpreters in input, intermediate code, memory requirement, error checks, and time consumption.

Table 3.2 Compilers vs. Interpreters

	Compiler	**Interpreter**
1	Compiler takes entire program as input.	Interpreter takes a single instruction as input.
2	Intermediate object code is generated.	No intermediate object code is generated
3	Conditional control statements are executed faster.	Conditional control statements are executed slower.
4	Memory requirement more; since object code is generated.	Memory requirement is less.
5	Program need not be compiled every time.	Every time the higher level program is converted into lower level program.
6	Errors are displayed after the entire program is checked	Errors are displayed for every instruction interpreted (if any).
7	Compiled languages are more efficient but difficult to debug.	Interpreted languages are less efficient but easier to debug because interpreter stops and reports errors as it encounters them.
8	Compiler does not allow a program to run until it is completely error-free.	Interpreter runs the program from first line and stops execution only if it encounters an error.
9	It takes less amount of time to	It takes large amount of time to analyze the

	analyze the source code but the overall execution time is slower.	source code but the overall execution time is comparatively faster.
10	Example : C compiler	Example : Python

3.5 Java and Python Languages Processors

3.5.1 Java

Java is both a compiled and interpreted language. When a Java program is written, the javac compiler converts the program into bytecode. Bytecode compiled by javac, is entered into the JVM memory where it is interpreted by another program called java. This java program interprets bytecode line-by-line and converts it into machine code to be run by the JVM [4].

3.5.2 Python

There are four steps that python takes when the return key is pressed: lexing, parsing, compiling, and interpreting. Lexing is breaking the line of code into tokens. The parser takes those tokens and generates a structure that shows their relationship to each other in this case, an Abstract Syntax Tree (AST). The compiler then takes the AST and turns it into one (or more) code objects. Finally, the interpreter takes each code object and executes the code it represents.

3.6 Literature Review

3.6.1 java2python

A simple but effective tool to translate Java source code into Python source code.

Created by Troy Melhase, java2python is licensed under the GNU General Public License 2.0. To translate code, java2python requires python 2.5, ANTLR and PyANTLR [5].

The java2python package can translate any syntactically valid Java source code file. The generated Python code is not guaranteed to run, nor is guaranteed to be syntactically valid Python [6].

Even though some manual checking is required in the output code; however, large amounts of time from hand converting would be saved.

3.6.2 Programming Language Inter-conversion

This paper presents a new approach of programming languages inter-conversion which can be applied to all types of programming languages [7].

The idea is an implementation of an intermediate language for inter-conversion. This language can be used to store the logic of the program in an algorithmic format without disturbing the structure of the original program. Separate convertors to and from the intermediate language have to be created for each language.

This is not an easy task, but if correctly implemented, it would greatly change the future of software development. It would simplify the process of developing programs, maintaining them and hence bring down costs tremendously.

3.6.3 VARYCODE

Varycode [8] is an online all-in-one programming code converter between C#, Visual Basic .Net, Java, C++, Ruby, Python and Boo, supporting 21 directions of conversion.

Varycode comes in different plans and pricing, and customers are provided with the payment option according to the use packages.

Varycode does not completely claim that the codes generated after conversion are a 100% accurate so it needs some manual check.

The home page of Varycode is “https://varycode.com”.

3.6.4 Tangible Software Solutions

Tangible Software Solutions Inc. [9] is a privately held corporation founded in 1997. It is a software development and consulting firm specializing in source code conversion tools and source code conversion projects.

It offers the most accurate and reliable source code conversion utilities on the market today: Instant C#, Instant VB, C++ to C# Converter, C++ to VB Converter, C++ to Java Converter, C# to Java Converter, VB to Java Converter, Java to C# Converter, Java to VB Converter, C# to C++ Converter, VB to C++ Converter, and Java to C++ Converter.

The corporation produces desktop applications and provides free editions for folder conversions up to 1000 lines at a time and code snippet conversions up to 100 lines at a time, but the standard editions can only be purchased.

3.6.5 JLCA

The Microsoft Java Language Conversion Assistant (JLCA) [10] is a tool that automatically converts existing Java-language source code to C# for developers who want to move their existing applications to the Microsoft .NET Framework.

The Java Language Conversion Assistant provides integration with Visual Studio .NET or Visual Studio 2005 IDE. It also allows to deliver full C# implementation by using the power of the .NET Framework and the component oriented programming features of C#.

Applications and services converted with the JLCA run only on the .NET Framework. They do not run on any Java Virtual Machine. Microsoft developed the JLCA independently. It is neither endorsed nor approved by Sun Microsystems, Inc.

3.6.6 BCX

BCX [7] [11] was originally started by Kevin Diggins. It is now completely open source and developed by a group.

BCX is a small command line tool that inputs a BCX BASIC source code file and outputs a 'C' source code file which can be compiled with many C or C++ compilers.

Using BCX and a C compiler enables the programmers to produce powerful 32-bit native code Windows console mode programs, windows GUI applications, and Dynamic Link Libraries (DLL's) without having to incur the costs of an expensive commercial BASIC compiler.

The only drawback is that Hardware Voices Controls are disabled.

3.6.7 PERTHON

Perthon converts Python source code to human-readable Perl 5.x source code[7] [12]. It makes use of Damian Conway's Parse::RecDescent for parsing, and aims to re-implement the Python language as specified in the Python Reference Manual and Backus-Naur Form (BNF) grammar. Perthon is similar to Jython, which re-implements Python on the JVM, except that Perthon works at the source code (not byte code) level.

Perthon does not yet support 'use', 'BEGIN', 'END', etc. This is due to how Perl handles these expressions: they get executed while parsing. It also does not handle 'bless', 'packages', etc. The references may or may not be resolved correctly. The prefix/postfix operators are not resolved as well.

3.7 Comparison between Converters

Table 3.3 discusses the major differences between previous studies and the simple Java Python converter.

Table 3.3 Comparison between Converters

Case Study	**Category**	**Source language**	**Target language**	**Written in**	**Intermediate language**	**Manual checking**	**Availability**
java2python	Tool	Java	Python	-	ANTLR	Required	Open source

VARYCODE	Online web based service	C#, Visual Basic .Net, Java, C++, Ruby, Python and Boo	C#, Visual Basic .Net, Java, C++, Ruby, Python and Boo	-	-	Required	Free to some 2048 to 4096 characters
Tangible Software Solutions	Desktop application	C++, C#, VB, Java	C++, C#, VB, Java	-	-	Required	Free up to 1000 line for folder and 100 line for snippet
JLCA	Tool	Java	C#	-	-	Required	Free
BCX	Tool	BCX BASIC	C	BCX BASIC	-	Required	Open source
Perthon	Software	Python	Perl 5.x	Perl	-	Required	Free
A simple Java-Python converter	Desktop application	Java	Python	Java	XML DOM	Required	Free

3.8 Summary

This chapter explained in details language processors (compilers and interpreters), and compared them. It also described Java and Python language processors and finally reviewed the literature review. The next chapter illustrated the system design and explains the conversion process in details.

Chapter 4 SYSTEM DESIGN

4.1 Introduction

This chapter explains the description of the system, the software environment, the programming language used to develop the project and the conversion process.

4.2 System Description

The simple Java Python Converter converts the Java input file to XML (eXtensible Markup Language) as an intermediate code and then converts the XML file to Python.

It reads Java statements from the Java file, converts them to XML tags and writes them in a scripting file (.xml). Then it reads XML tags, converts them to Python statements and writes them in a Python file as shown in Figure 4.1.

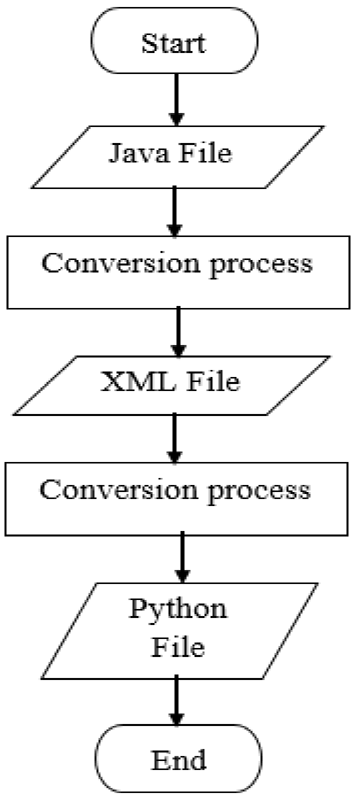


Figure 4.1 Conversion Stages

4.2.1 Reasons for using an intermediate language

The main reason of using an intermediate language is to facilitate the process of conversion by extracting the basic components of each statement on which programming languages depend on to build their own statement.

Also it can be used after some modifications associated with the structure of the language to which the code is converted to facilitate the process of conversion to other programming language besides Python without the need to start from the beginning and repeat the conversion process from Java.

4.2.2 XML

Stands for eXtensible Markup Language, Some of XML features:

- Was designed to carry data.
- Was designed to be both human- and machine-readable.
- XML does not do anything.
- XML tags are not predefined.

4.2.3 Selecting XML as an intermediate language

- One of the most time-consuming challenges for developers is to exchange data between incompatible applications. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
- XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
- With XML, data can be available to all kinds of "reading machines" (Handheld computers, voice machines, news feeds, etc.).

4.3 System Environment

The Java Python Converter is a desktop application, and runs on desktop devices (PCs) that use windows as its operating system.

The extension of the application is (.exe) so it can run on windows even if the Java Virtual Machine (JVM) has not been installed.

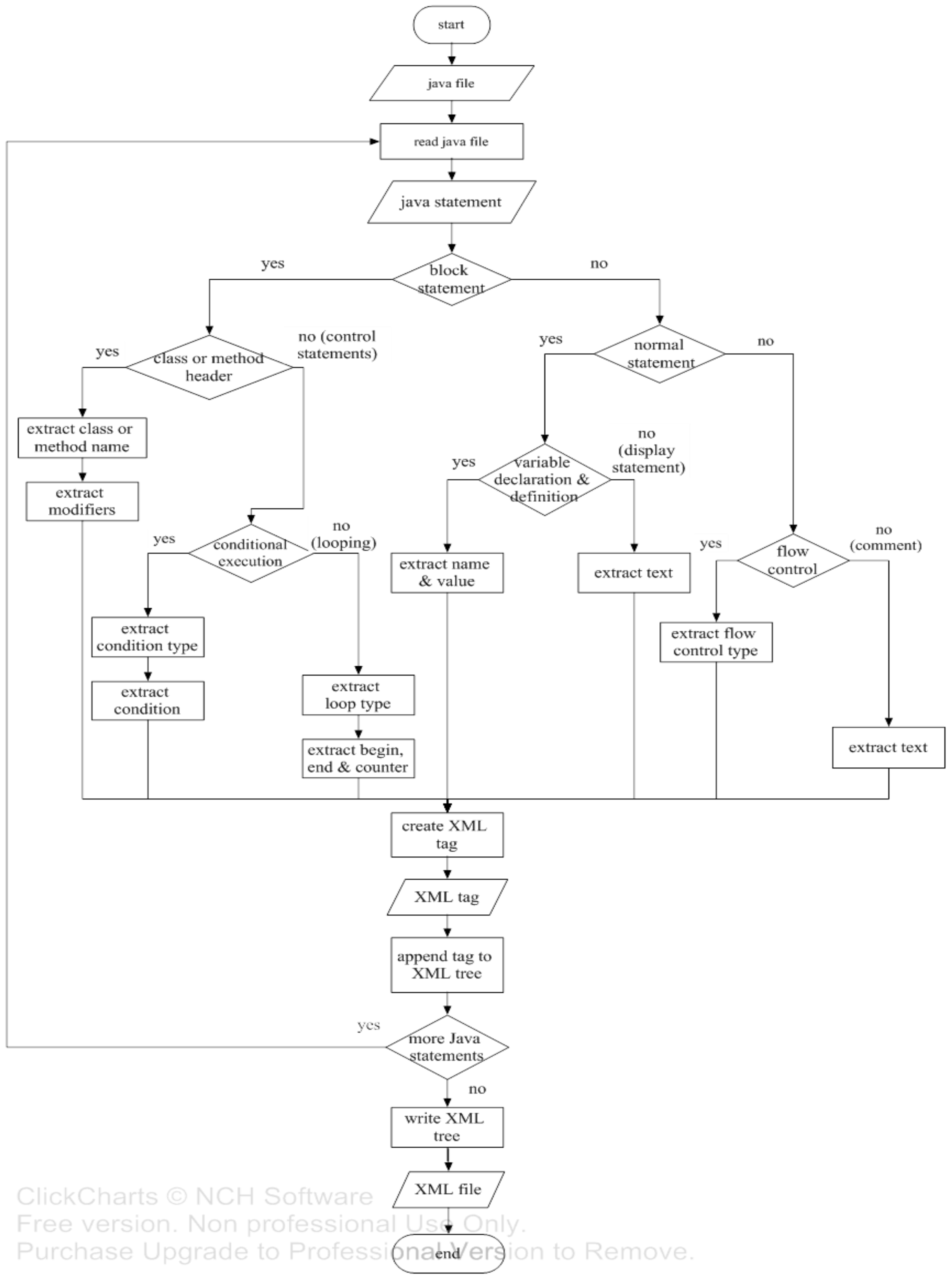
4.4 Conversion Process

The conversion process goes through two phases. In the first phase the Java file is converted to an XML file and in the second phase the XML file is converted to a Python file.

4.4.1 Phase one

The converter reads the Java file character by character to extract the type and components of the Java statements. Each Java statement is converted to an appropriate XML tag; the tag name is determined based on the type of Java statement. The components of the Java statement are stored in the tag attributes.

When an XML tag is created the Converter appends it to the XML Document Object Model (DOM) tree. The conversion process will continue until the end of the Java file. Then the Converter transforms the DOM tree to an XML file. As shown in Figure 4.2.



ClickCharts © NCH Software
Free version. Non professional Use Only.
Purchase Upgrade to Professional Version to Remove.

Figure 4.2 Java to XML Conversion Process

Table 4.1 describes XML tags names, attributes assigned to each one and if the tag has a value or body. The tag name determines the type of the Java statement and the tag attributes determine the components of the Java statement.

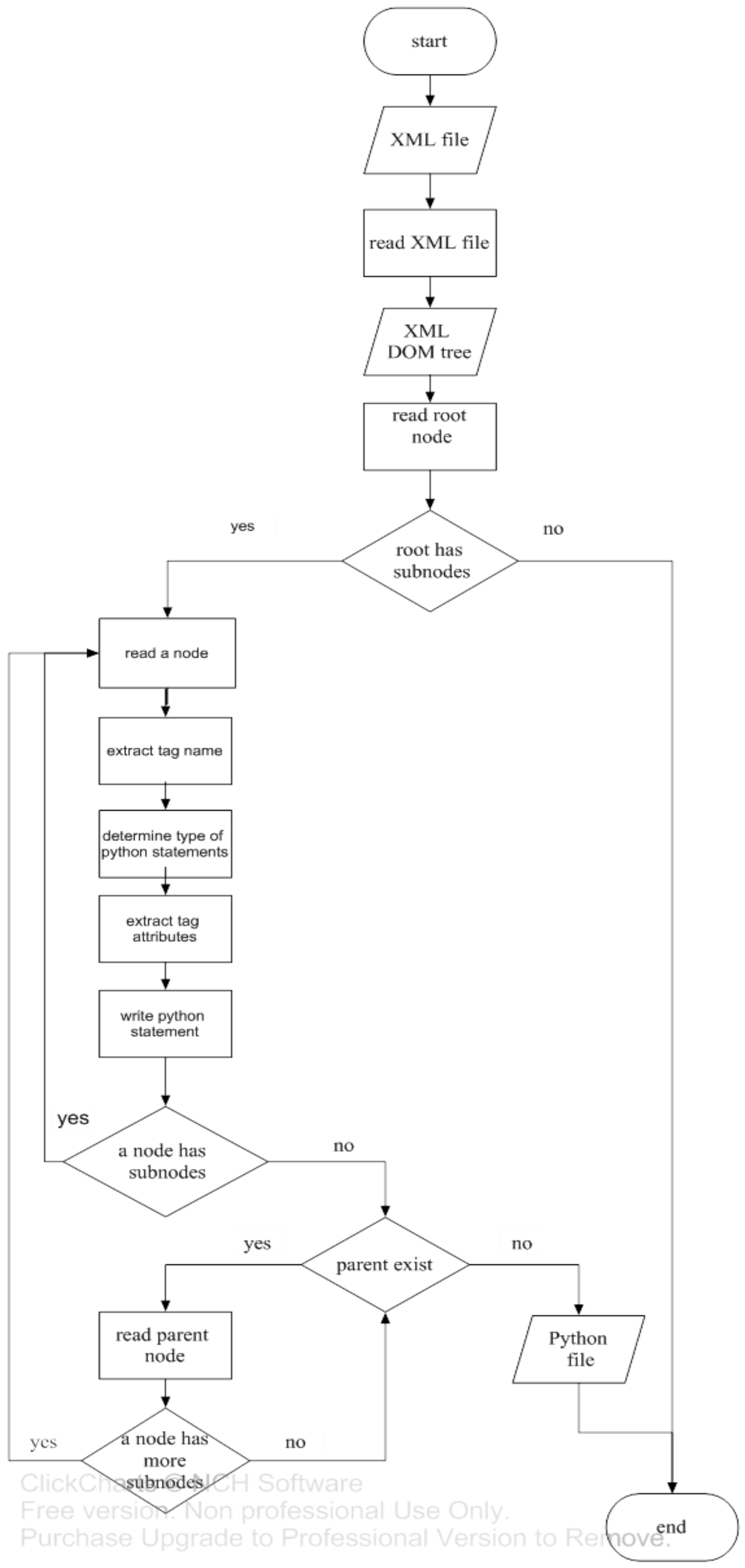
Table 4.1 XML Tags Components

Tag Name	Tag Attributes									Value	Body
	Access	Identifier	static	Type	condition	nl	begin	end	counter		
Class	✓	✓									✓
Comment				✓						✓	
Method (method header)	✓	✓	✓	✓							✓
param (method parameter)		✓		✓							
call (method call)										✓	
return (method return)										✓	
If					✓						✓
Else											✓
For							✓	✓	✓		✓
while (also do…while)					✓						✓
Continue	Doesn’t have tag attributes, value and body										
Break	Doesn’t have tag attributes, value and body										
var (variables declaration and definition)	✓	✓	✓	✓						✓	
Display						✓				✓	

4.4.2 Phase two

The converter reads the XML file to extract the DOM tree. Then it reads the tree nodes which are XML tags. Each XML tag is converted to an appropriate Python statement; the type of Python statement is determined based on the tag name. The components of the Python statement are extracted from the tag attributes.

The converter writes the Python statement in a Python file. The conversion process will continue until the end of the DOM tree. As shown in Figure 4.3.

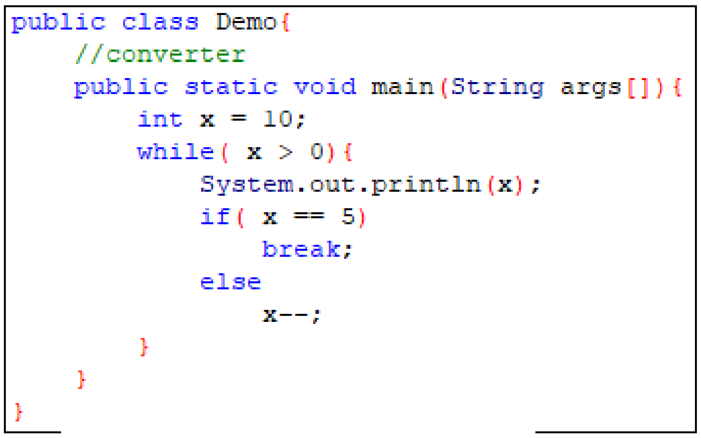


ClickCharts © NCH Software
Free version. Non professional Use Only.
Purchase Upgrade to Professional Version to Remove.

Figure 4.3 XML to Python Conversion Process

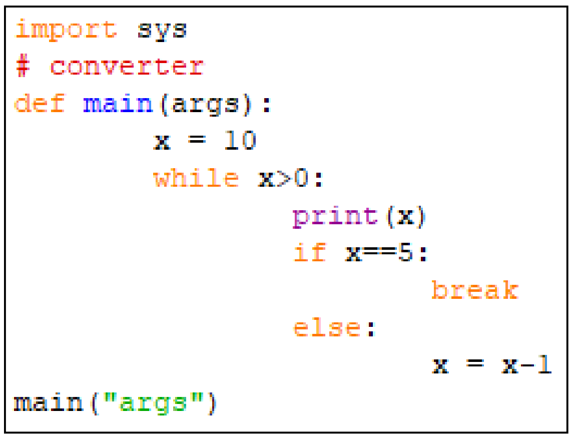
4.4.3 Simple example

Figure 4.4 (a) is an input Java code to the converter, (b) illustrates intermediate XML code and (c) is an output python code from the converter.



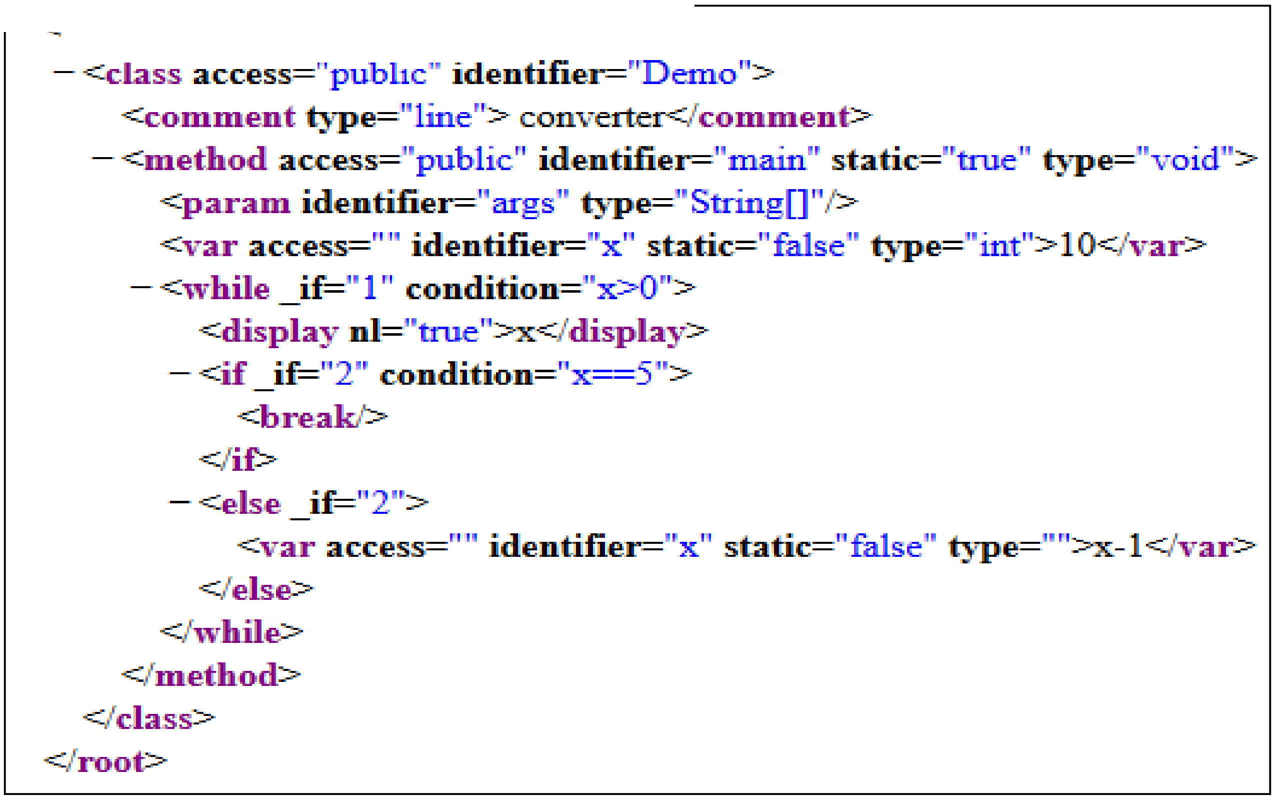
```java
public class Demo{
    //converter
    public static void main(String args[]){
        int x = 10;
        while( x > 0){
            System.out.println(x);
            if( x == 5)
                break;
            else
                x--;
        }
    }
}
```

(a)



```python
import sys
# converter
def main(args):
        x = 10
        while x>0:
                print(x)
                if x==5:
                        break
                else:
                        x = x-1
main("args")
```

(c)



```xml
  - <class access="public" identifier="Demo">
      <comment type="line"> converter</comment>
    - <method access="public" identifier="main" static="true" type="void">
        <param identifier="args" type="String[]"/>
        <var access="" identifier="x" static="false" type="int">10</var>
      - <while _if="1" condition="x>0">
          <display nl="true">x</display>
        - <if _if="2" condition="x==5">
            <break/>
          </if>
        - <else _if="2">
            <var access="" identifier="x" static="false" type="">x-1</var>
          </else>
        </while>
      </method>
    </class>
  </root>
```

(b)

Figure 4.4 simple input & output from the converter: (a) the Java Input, (b) the Intermediate XML Code and (c) the Python Output

4.5 Summary

In this chapter we explained the converter system design, the software environment, the programming language used to develop the project, the conversion process and a simple example for Java input code and its output in Python and XML. In the next chapter we explain the implementation of the converter and more code examples in details and how to run the converter.

Chapter 5 IMPLEMENTATION

5.1 Introduction:

In this chapter the implementation of the converter is demonstrated. The converter screens are displayed and some examples of input and output of the converter are described. This chapter also discusses the challenges facing the converter to get the correct output as much as possible.

5.2 Running the Converter:

The converter runs by clicking the icon shown in Figure 5.1, which is a .exe file. The GUI depicted in Figure 5.2 appears.

Figure 5.1 Converter Application Icon

The user clicks the browse button to specify the java file path, and the CONVERT button to perform the conversion process. After the conversion process is complete, the XML file and Python file are saved in the same path of the java file.

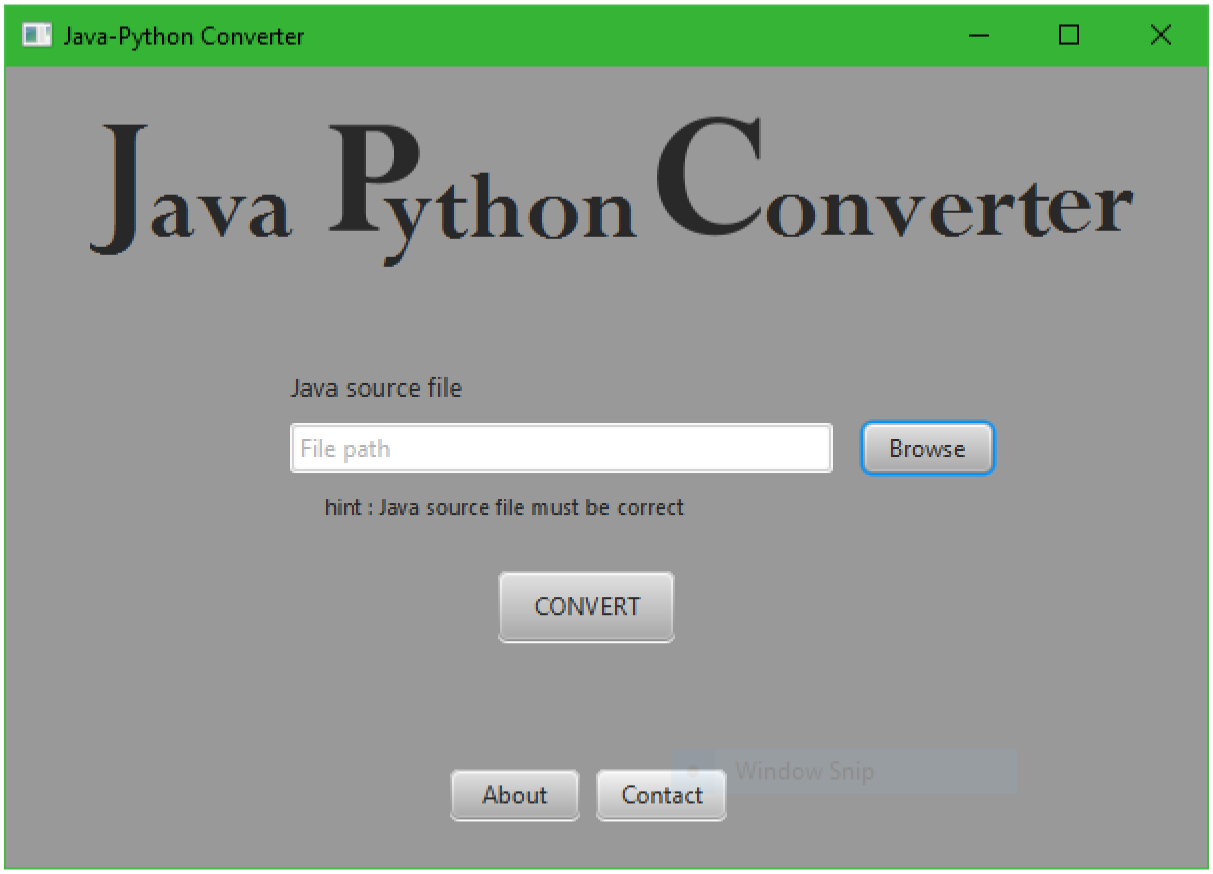


Figure 5.2 Converter GUI

5.3 Simple Examples:

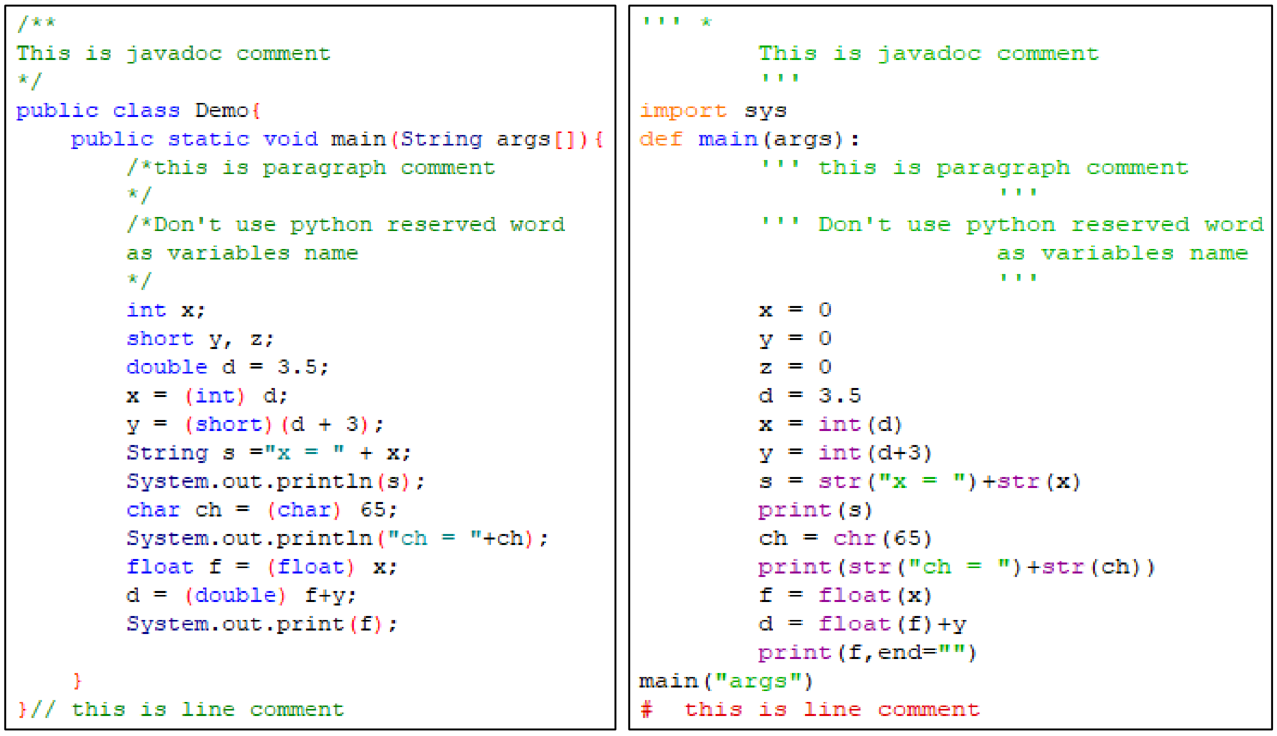
Figure 5.3 (a) is an input Java code to the converter and Figure 5.3 (b) is an output Python code from the converter.

This code covers comments, variables declaration and definition, type casting and display on screen. Following are some notes:

- A Javadoc comment is converted to a multiple line comment because Python doesn't have doc comments.
- Reserved words in Python should not be used as variable names in the Java code because this causes a problem with the Python code.
- Python does not allow the declaration of variables without assigning values to them. So if a variable is declared in Java with no initial value, the converter assigns zero to the variable in Python.

- In the case of declaration or definition of more than one variable in the same statement in Java, the converter processes each variable separately and each variable has its own line in Python because it is not possible to define more than one variable in the same statement.
- When concatenating a string with another data type, by default Java considers it as a string, but Python produces an error, so the converter castes the other data type to string.
- Casting String to integer or string to float is out of the scope of the converter because Java uses specific classes for casting: Integer.ParseInt("string") or Double.ParseDouble("string ").

- When casting to short or long the converter converts to integer and converts double to float; because Python does not contain these types.
- Python display statement print() prints a new line by default at the end of the statement, so when System.out.print() is used in the Java code, the converter adds (end="") to the end of the print() Python statement.



```
/**
This is javadoc comment
*/
public class Demo{
    public static void main(String args[]){
        /*this is paragraph comment
        */
        /*Don't use python reserved word
        as variables name
        */
        int x;
        short y, z;
        double d = 3.5;
        x = (int) d;
        y = (short)(d + 3);
        String s ="x = " + x;
        System.out.println(s);
        char ch = (char) 65;
        System.out.println("ch = "+ch);
        float f = (float) x;
        d = (double) f+y;
        System.out.print(f);

    }
}// this is line comment
```

```
''' *
        This is javadoc comment
        '''
import sys
def main(args):
        ''' this is paragraph comment
                '''
        ''' Don't use python reserved word
                as variables name
                '''
        x = 0
        y = 0
        z = 0
        d = 3.5
        x = int(d)
        y = int(d+3)
        s = str("x = ")+str(x)
        print(s)
        ch = chr(65)
        print(str("ch = ")+str(ch))
        f = float(x)
        d = float(f)+y
        print(f,end="")
main("args")
#  this is line comment
```

(a) (b)

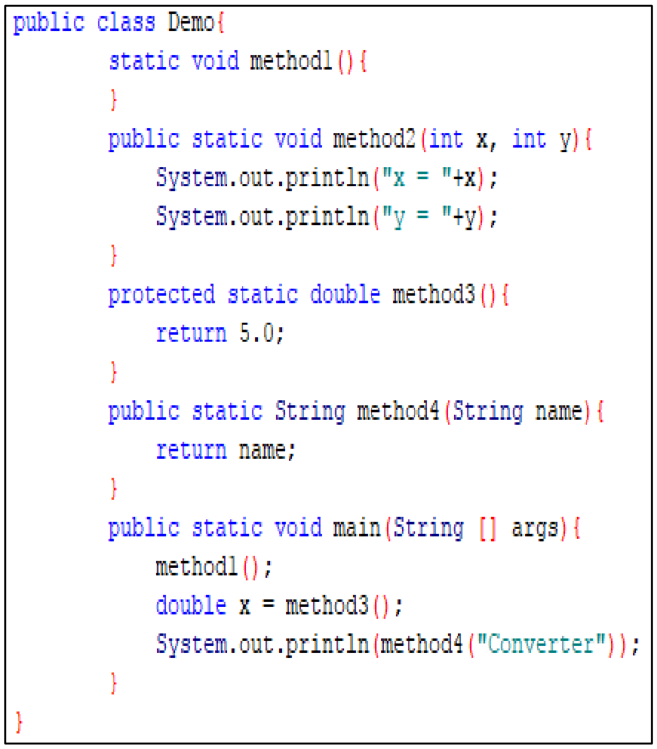
Figure 5.3 Simple Example: (a) Java Input Code and (b) Python Output Code

5.3.1 Methods:

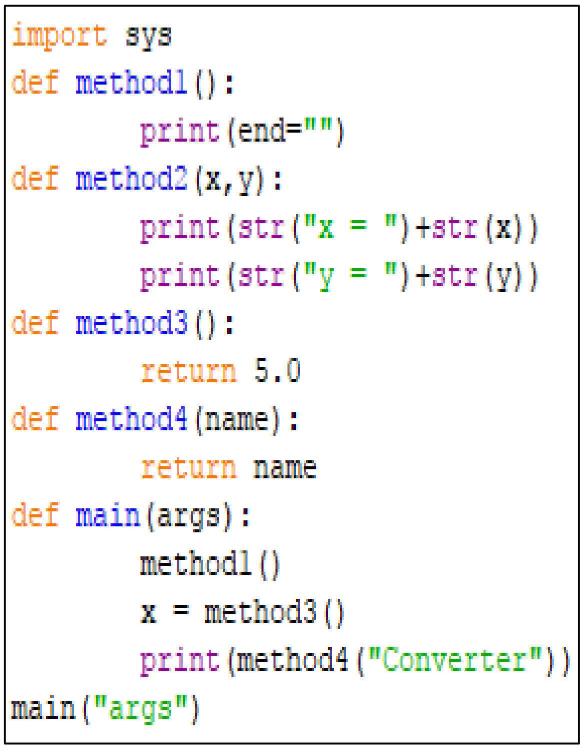
Figure 5.4 shows an example of method conversion where (a) is an input Java code to the converter and (b) is an output Python code from the converter.

By default, the execution of the program in Java starts from the main method. Therefore, the converter invokes the main method in the Python code if found in the Java code, because Python does not require the existence of the main method for execution.

Unlike Java, Python doesn't accept an empty block. So in case of an empty method body the converter writes print(end="") to act as the method body.



```java
public class Demo{
        static void method1(){
        }
        public static void method2(int x, int y){
            System.out.println("x = "+x);
            System.out.println("y = "+y);
        }
        protected static double method3(){
            return 5.0;
        }
        public static String method4(String name){
            return name;
        }
        public static void main(String [] args){
            method1();
            double x = method3();
            System.out.println(method4("Converter"));
        }
}
```



```python
import sys
def method1():
        print(end="")
def method2(x,y):
        print(str("x = ")+str(x))
        print(str("y = ")+str(y))
def method3():
        return 5.0
def method4(name):
        return name
def main(args):
        method1()
        x = method3()
        print(method4("Converter"))
main("args")
```

(a) (b)

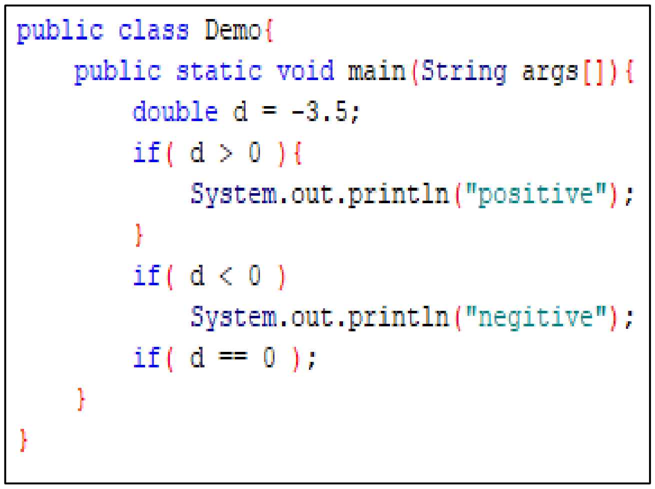
Figure 5.4 Method Example: (a) Java Input Code and (b) Python Output Code

5.3.2 Selection statements:

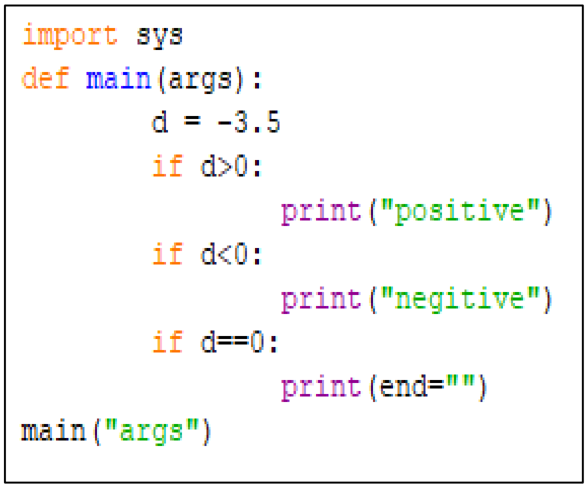
5.3.2.1 The if statement:

Figure 5.5 is an example of if statement where (a) is an input Java code to the converter and (b) is an output Python code from the converter.

When the converter finds if or else statements without a body it writes print(end="") in Python to act as the body.



```
public class Demo{
    public static void main(String args[]){
        double d = -3.5;
        if( d > 0 ){
            System.out.println("positive");
        }
        if( d < 0 )
            System.out.println("negitive");
        if( d == 0 );
    }
}
```



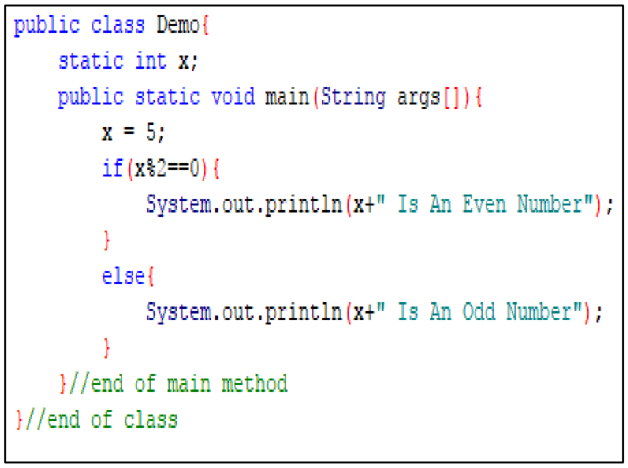
```
import sys
def main(args):
        d = -3.5
        if d>0:
                print("positive")
        if d<0:
                print("negitive")
        if d==0:
                print(end="")
main("args")
```

(a) (b)

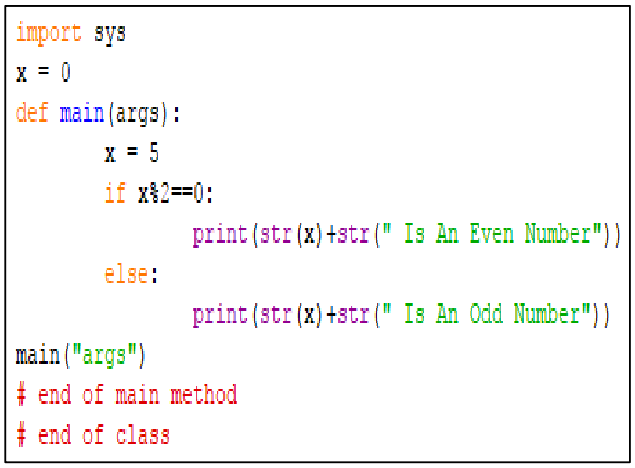
Figure 5.5 if Example: (a) Java Input Code and (b) Python Output Code

5.3.2.2 The if...else statement:

Figure 5.6 shows the if...else statement where (a) is an input Java code to the converter and (b) is an output Python code from the converter.



```java
public class Demo{
    static int x;
    public static void main(String args[]){
        x = 5;
        if(x%2==0){
            System.out.println(x+" Is An Even Number");
        }
        else{
            System.out.println(x+" Is An Odd Number");
        }
    }//end of main method
}//end of class
```



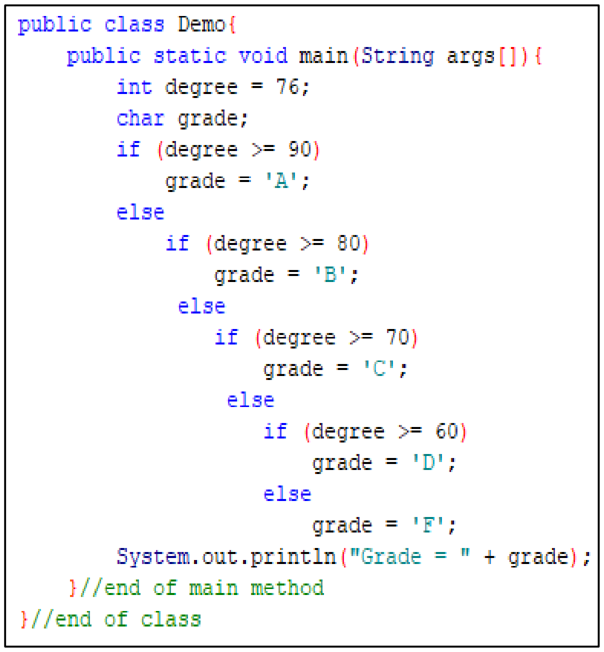
```python
import sys
x = 0
def main(args):
        x = 5
        if x%2==0:
                print(str(x)+str(" Is An Even Number"))
        else:
                print(str(x)+str(" Is An Odd Number"))
main("args")
# end of main method
# end of class
```

(a) (b)

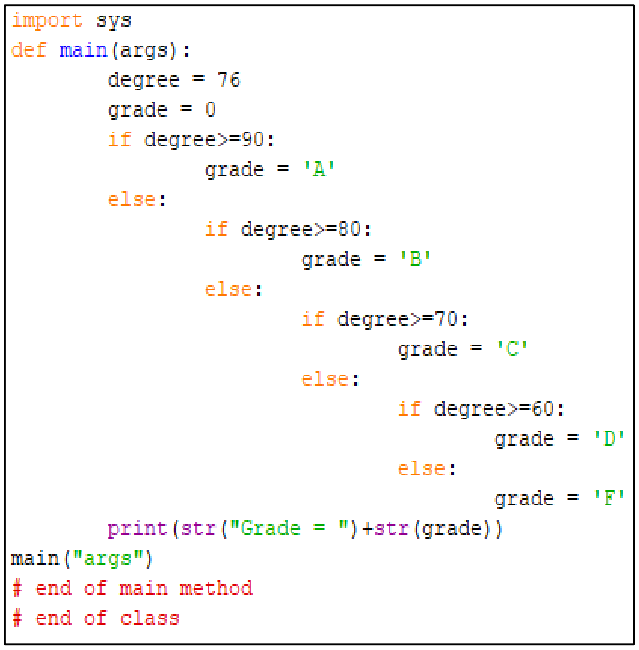
Figure 5.6 if...else Example: (a) Java Input Code and (b) Python Output Code

5.3.2.3 Nested if statement:

Figure 5.7 (a) is an input Java code to the converter and (b) is an output Python code from the converter. This code illustrates nested if statements in Java and Python.



```java
public class Demo{
    public static void main(String args[]){
        int degree = 76;
        char grade;
        if (degree >= 90)
            grade = 'A';
        else
            if (degree >= 80)
                grade = 'B';
             else
                if (degree >= 70)
                    grade = 'C';
                 else
                    if (degree >= 60)
                        grade = 'D';
                    else
                        grade = 'F';
        System.out.println("Grade = " + grade);
    }//end of main method
}//end of class
```



```python
import sys
def main(args):
        degree = 76
        grade = 0
        if degree>=90:
                grade = 'A'
        else:
                if degree>=80:
                        grade = 'B'
                else:
                        if degree>=70:
                                grade = 'C'
                        else:
                                if degree>=60:
                                        grade = 'D'
                                else:
                                        grade = 'F'
        print(str("Grade = ")+str(grade))
main("args")
# end of main method
# end of class
```

(a) (b)

Figure 5.7 Nested if Example: (a) Java Input Code and (b) Python Output Code

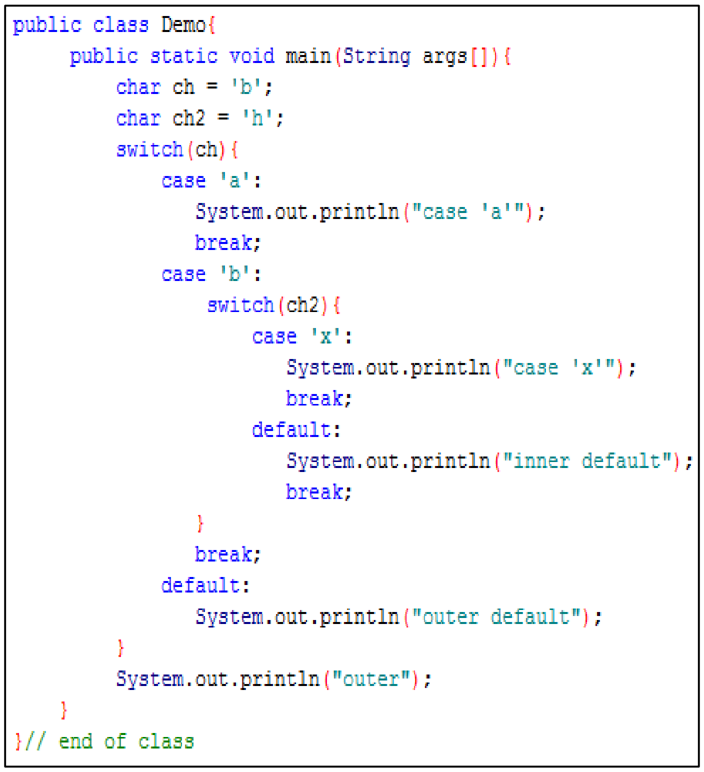
5.3.3 The switch statement:

Figure 5.8 (a) is an input Java code to the converter and (b) is an output Python code from the converter. This code illustrates the switch statement in Java and the corresponding if statement in Python.

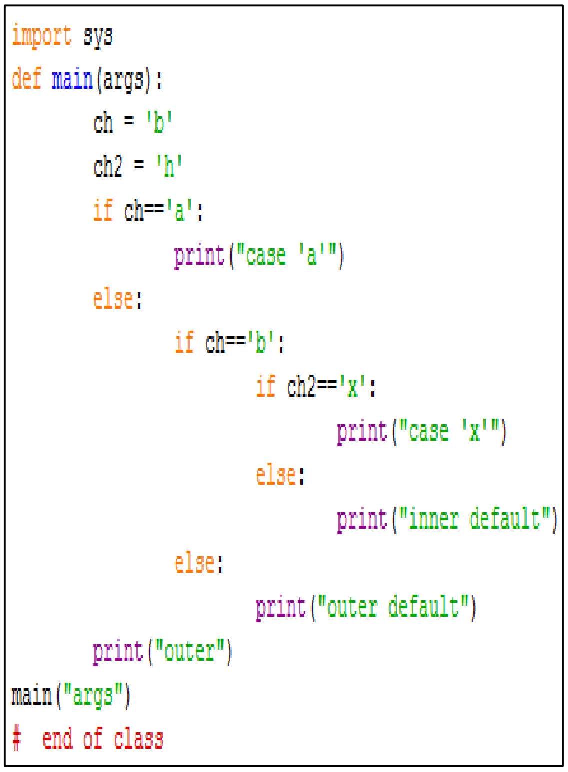
It is known that Python does not contain the switch statement in its structure as Java. Even though there are many ways to represent the Java switch statement in Python code, it does not cover all cases.

The converter converts the Java switch statement to nested…if statements in Python, because it is clear, simple and easy to understand.

It is worth noting that even with no break statement found in the switch statement the conversion process is done assuming its existence.



```java
public class Demo{
    public static void main(String args[]){
        char ch = 'b';
        char ch2 = 'h';
        switch(ch){
            case 'a':
               System.out.println("case 'a'");
               break;
            case 'b':
                switch(ch2){
                    case 'x':
                       System.out.println("case 'x'");
                       break;
                    default:
                       System.out.println("inner default");
                       break;
                }
               break;
            default:
               System.out.println("outer default");
        }
        System.out.println("outer");
    }
}// end of class
```



```python
import sys
def main(args):
        ch = 'b'
        ch2 = 'h'
        if ch=='a':
                print("case 'a'")
        else:
                if ch=='b':
                        if ch2=='x':
                                print("case 'x'")
                        else:
                                print("inner default")
                else:
                        print("outer default")
        print("outer")
main("args")
# end of class
```

(a) (b)

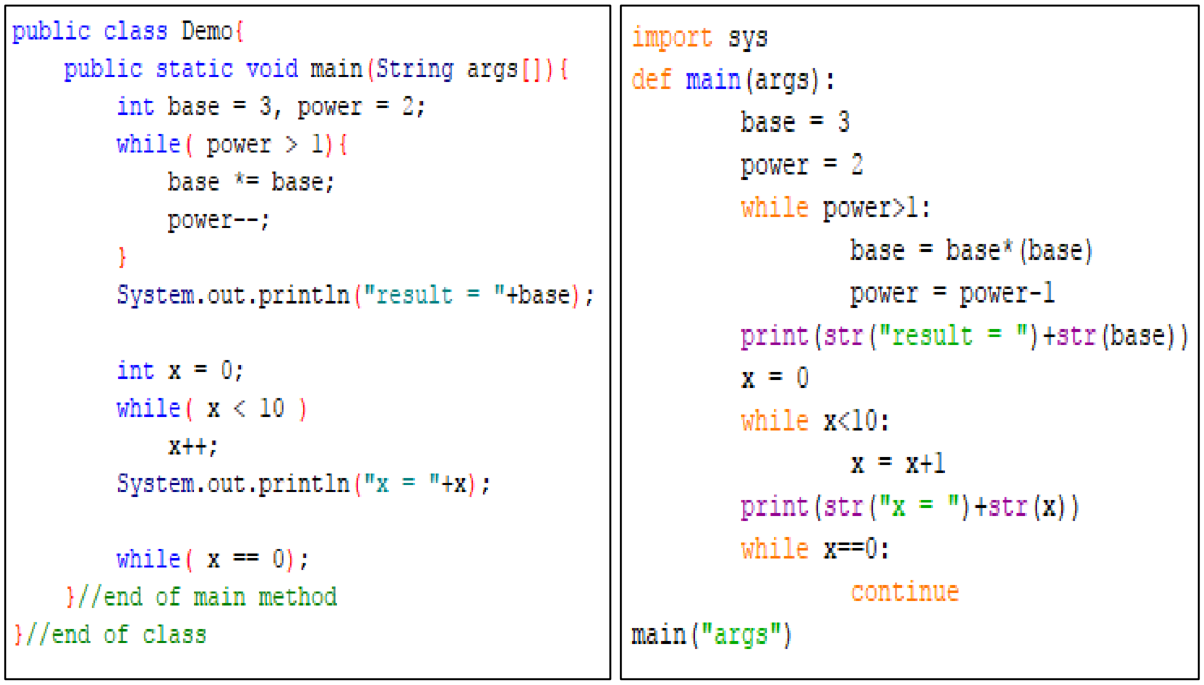
Figure 5.8 switch Example: (a) Java Input Code and (b) Python Output Code

5.3.4 Iteration statements:

5.3.4.1 The while statement:

Figure 5.9 (a) is an input Java code to the Converter and (b) is an output python code from the Converter. This code covers while statement in Java and Python.

When the converter finds a while statements without a body it writes a continue statement to act as a body. This is shown in **Error! Reference source not found.** (a) and (b).



```java
public class Demo{
    public static void main(String args[]){
        int base = 3, power = 2;
        while( power > 1){
            base *= base;
            power--;
        }
        System.out.println("result = "+base);

        int x = 0;
        while( x < 10 )
            x++;
        System.out.println("x = "+x);

        while( x == 0);
    }//end of main method
}//end of class
```

```python
import sys
def main(args):
        base = 3
        power = 2
        while power>1:
                base = base*(base)
                power = power-1
        print(str("result = ")+str(base))
        x = 0
        while x<10:
                x = x+1
        print(str("x = ")+str(x))
        while x==0:
                continue
main("args")
```

(a) (b)

Figure 5.9 while Example: (a) Java Input Code and (b) Python Output Code

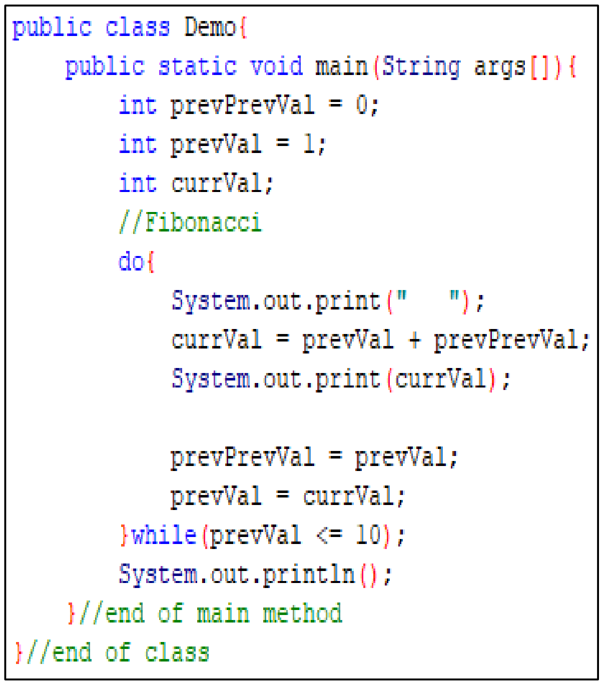
5.3.4.2 The do…while statement:

The input and output codes illustrated in Figure 5.10 (a) and (b) are examples of the do…while statement.

Python does not have a do…while loop, therefore the converter converts it to a while loop with the following steps:

- The converter sets the condition of the while loop to True.

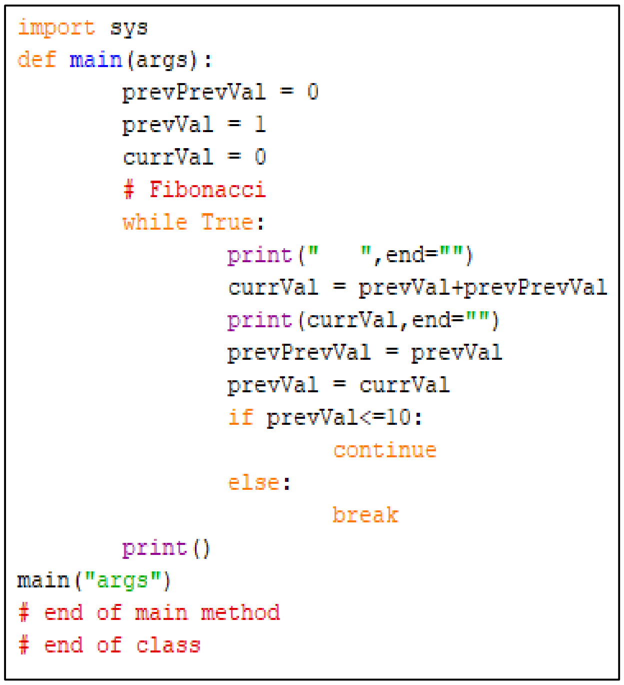
- It then appends an if…else statement to the body of the do…while loop with a continue statement in the if body and a break statement in the else body. if condition represents a do…while condition.



```
public class Demo{
    public static void main(String args[]){
        int prevPrevVal = 0;
        int prevVal = 1;
        int currVal;
        //Fibonacci
        do{
            System.out.print("   ");
            currVal = prevVal + prevPrevVal;
            System.out.print(currVal);

            prevPrevVal = prevVal;
            prevVal = currVal;
        }while(prevVal <= 10);
        System.out.println();
    }//end of main method
}//end of class
```

(a)



```
import sys
def main(args):
        prevPrevVal = 0
        prevVal = 1
        currVal = 0
        # Fibonacci
        while True:
                print("   ",end="")
                currVal = prevVal+prevPrevVal
                print(currVal,end="")
                prevPrevVal = prevVal
                prevVal = currVal
                if prevVal<=10:
                        continue
                else:
                        break
        print()
main("args")
# end of main method
# end of class
```

(b)

Figure 5.10 do...while Example: (a) Java Input Code and (b) Python Output Code

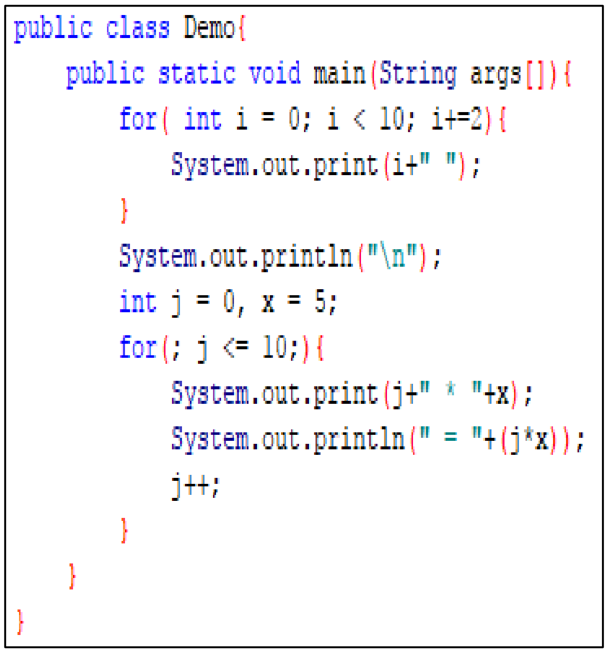
5.3.4.3 The for statement:

The code shown in Figure 5.11 covers the for statement, where (a) is the Java input code and (b) is the Python output code.

There are many differences in syntax and logic between Java and Python for the for statement. Therefore, the converter makes important changes to eliminate these differences. These differences include:

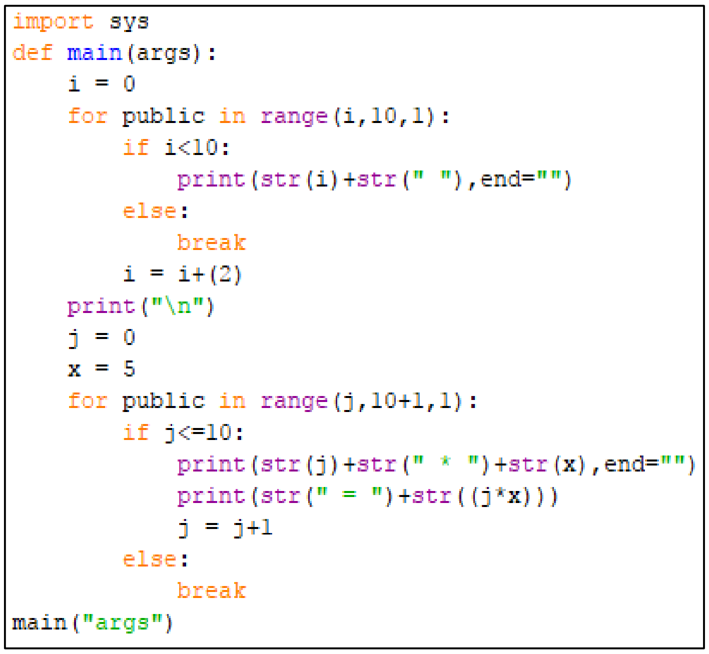
- Python does not allow defining variables on the header of the for statement as Java.
- The index of the for statement in Java holds the start value, while in Python a variable separate from the index variable holds the start value.

- Start and end values should be explicitly mentioned in the Python for statement.
- The for loop in Java repeats until the end value but in Python until the end value minus one. For example: if a for loop starts at 0 and ends at 5, in Java this for loop repeats 6 times (0,1,2,3,4,5) while in Python it repeats 5 times (0,1,2,3,4).
- Therefore, when the start value is assigned to the index of the Java for statement on the header, the converter defines it before the Python for statement.
- The converter adds a public variable to reserve the index space and use the variable which holds the start value as an index. It increases (or decreases) the index at the end of the for statement body and adds an if…else statement to break the loop if the condition is false. The body of the for statement becomes the body of the if statement.



```
public class Demo{
    public static void main(String args[]){
        for( int i = 0; i < 10; i+=2){
            System.out.print(i+" ");
        }
        System.out.println("\n");
        int j = 0, x = 5;
        for(; j <= 10;){
            System.out.print(j+" * "+x);
            System.out.println(" = "+(j*x));
            j++;
        }
    }
}
```

(a)

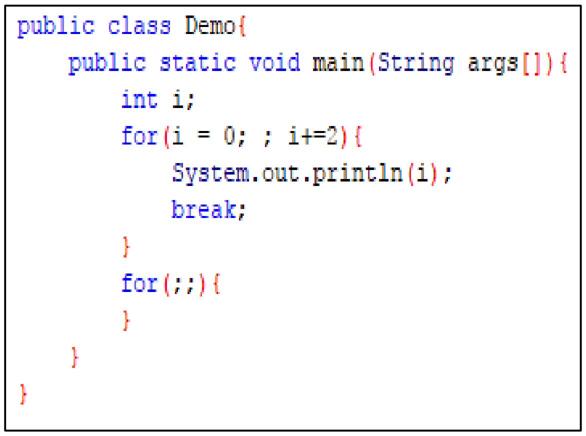


```
import sys
def main(args):
    i = 0
    for public in range(i,10,1):
        if i<10:
            print(str(i)+str(" "),end="")
        else:
            break
        i = i+(2)
    print("\n")
    j = 0
    x = 5
    for public in range(j,10+1,1):
        if j<=10:
            print(str(j)+str(" * ")+str(x),end="")
            print(str(" = ")+str((j*x)))
            j = j+1
        else:
            break
main("args")
```

(b)

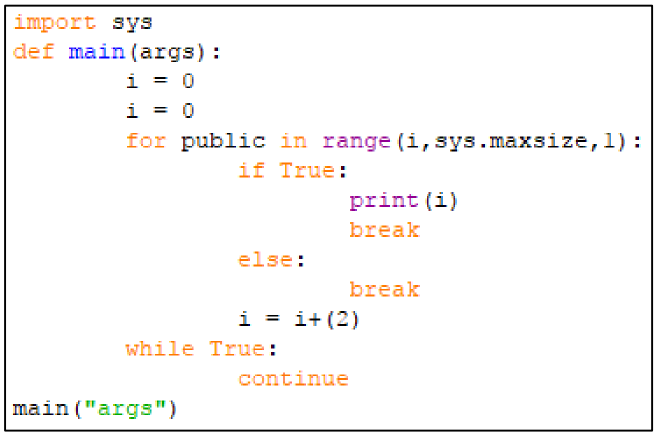
Figure 5.11 for Example: (a) Java Input Code and (b) Python Output Code

- When the repetition condition is not clarified in the Java for statement, the converter considers the condition to be True and the end value is the maximum value of the integer type to prevent errors. See Figure 5.12.
- When the header of the Java for loop is empty the converter represents it as a while True statement. See Figure 5.12.



```java
public class Demo{
    public static void main(String args[]){
        int i;
        for(i = 0; ; i+=2){
            System.out.println(i);
            break;
        }
        for(;;){
        }
    }
}
```

(a)



```python
import sys
def main(args):
        i = 0
        i = 0
        for public in range(i,sys.maxsize,1):
                if True:
                        print(i)
                        break
                else:
                        break
                i = i+(2)
        while True:
                continue
main("args")
```

(b)

Figure 5.12 Empty for Example: (a) Java Input Code and (b) Python Output Code

5.4 Summary:

This chapter demonstrated how to run the converter, and provided examples of how the converter manipulates Java input file to gain the correct Python output. The next chapter is conclusions and recommendations for future work.

Chapter 6 CONCLUSIONS AND RECOMMENDATIONS

This chapter explains the final result of this project, recommendations for future work to improve the converter and the conclusion of this thesis.

6.1 Results:

After finishing the design, implementation and testing of the converter, we found that it can convert the syntax of Java programs to Python without having to rewrite Python program from start. It is better to check the Python program to make sure that the logic is correct.
The converter covers the following Java principles:

- Class declaration.
- Method declaration.
- Comments.
- Declaring and initializing primitive, floating points and boolean variables.
- Selection statements:
    - If statement.
    - If…else statement.
    - Nested if statement.
- Switch statement.
- Iteration Statements:
    - While statement.
    - Do…while statement.
    - For statement.

6.2 Recommendations:

To improve the converter, we recommend the following:

- Extending the converter to convert arrays in Java to the corresponding type in Python.
- Include Object Oriented Programming (OOP) principles.

- Use our first conversion phase (Java to XML) to convert from Java to other programming languages besides Python.

6.3 Conclusions:

The conversion process has been placed among the top 10 challenges in the programming world. Achieving the maximum efficiency of the conversion without compromising the quality of the converted program is the programmer’s target.

This research has been completed, and serves the programmers in the transition from Java programming language to Python programming language in business and academia to gain time and effort.

Finally, we thank Allah, who helped us complete this project.

References:

[1] Naughton, P., &Schildt, H. (1996). *Java: the complete reference*. Osborne/McGraw-Hill.

[2] Halterman, R. L. (2011). Learning to program with python. *Retrieved January*, *14*, 2016.

[3] Aho, A. V., Sethi, R., & Ullman, J. D. (2007). Compilers: principles, techniques, and tools (Vol. 2). Reading: Addison-wesley.

[4] Brown, P. J. (1979). Writing interactive compilers and interpreters. Wiley Series in Computing, Chichester: Wiley, 1979.

[5] java2python, [online], available at: http://www.indicthreads.com/1154/translate-java-code-into-python-source-code-using-java2python-tool/, date accessed: 15/4/2017.

[6] java2python, [online], available at: https://github.com/natural/java2python, date accessed: 15/4/2017.

[7] George, D., Girase, P., Gupta, M., Gupta, P., & Sharma, A. (2010). Programming Language Inter-conversion. syntax, 1(20).

[8] Varycode, [online], available at: http://www.thetaranights.com/varycode/, date accessed: 3/10/2017.

[9] Tangible Software Solutions, [online], available at: https://www.tangiblesoftwaresolutions.com/index.html, date accessed: 6/10/2017.

[10] JLCA, [online], available at: http://support.microsoft.com/kb/819018, date accessed: 9/10/2017.

[11] BCX, [online], available at: http://bcx-basic.sourceforge.net/, date accessed: 3/10/2017.

[12] perthon, [online], available at: http://freshmeat.net/projects/perthon, date accessed: 5/10/2017.

[13] Terekhov, A. A., & Verhoef, C. (2000). The realities of language conversions. IEEE Software, 17(6), 111-124.