Lecture Notes

Computer Programming & Data Structures in Python (Theory & Lab)

MA26003 & MA26005

M.Sc. Mathematics and Scientific Computing
Department of Mathematics

Prepared by

Dr. Sivalingam S M

Assistant Professor
Department of Mathematics
National Institute of Technology Warangal

Academic Year 2026–2027

Preface

Python has emerged as one of the most widely used programming languages in scientific computing due to its simplicity, readability, extensive standard library, and rich ecosystem of scientific packages. It is now an indispensable tool in mathematics, statistics, data science, machine learning, numerical analysis, optimization, and scientific research.

These lecture notes have been prepared for the course **MA26003: Computer Programming & Data Structures in Python** offered by the Department of Mathematics, National Institute of Technology Warangal. The notes are intended primarily for M.Sc. Mathematics students with little or no prior programming experience.

Students are encouraged to practice every programming example independently and consult the prescribed textbooks for further reading.

Prepared by
Dr. Sivalingam S M
Assistant Professor
Department of Mathematics
National Institute of Technology Warangal

Course Information

Computer Programming & Data Structures in Python

MA26003 3-0-0 (3)

Course Outcomes

CO-1	Introduce the fundamental concepts of Python programming.
CO-2	Provide a foundation for using the basic building blocks of Python.
CO-3	Develop Python scripts for solving computational problems.
CO-4	Explore various exception-handling mechanisms in Python.
CO-5	Develop and utilize Python packages for scientific computing.

Syllabus

1. Introduction to Python		
Python variables	1	
Python basic operators	2	
Python data types	3	
Numeric data types: `int`, `float`, etc.	4	
Variable declaration and assignment	5	
Basic input and output operations	6	

2. Conditionals and Loops		
Boolean values	7	
Conditional statements (`if`, `else`, `elif`)	8	
Simple `for` loops	9	
Using `range()` in `for` loops	10	
Iteration over strings, lists and dictionaries	11	
The `while` loop	12	
Loop manipulation using `break`, `continue`, `pass` and `else`	13	

3. Strings		
Creating and assigning strings	14	
String manipulation	15	
String operators	16	
String formatting	17	
Triple quoted strings	18	
Raw strings	19	
Unicode strings	20	
Built-in string methods	21	

4. Lists, Tuples and Dictionaries		
Introduction to lists	22	
Accessing list elements	23	
List manipulation	24	
List operations	25	
Indexing and slicing	26	
Matrices using nested lists	27	
Tuple data type	28	
Dictionary data type	29	
Programming using built-in functions for strings, lists and dictionaries	30	

5. Functions		
Built-in functions	31	
User-defined functions	32	
Writing Python functions	33	
Returning values from functions	34	
Pass by value and pass by reference	35	
Function arguments and their types	36	
Recursive functions	37	

6. Python Packages		
Introduction to Python packages	38	
NumPy: Basic numerical computations	38	
Matplotlib: Basic plotting	39	
Pandas: Basic data handling	40	
Simple programs using built-in functions of NumPy, Matplotlib and Pandas	41-42	

Learning Resources

Text Books

1. William Mitchell, Pavel Solin, Martin Novak *et al.*, *Introduction to Python Programming*, NCLab Public Computing, 2012.

2. C. Jacob Fredslund, *Introduction to Python Programming*, 2007.

Reference Books

1. John C. Lusth, *An Introduction to Python*, The University of Alabama, 2011.

2. Dave Kuhlman, *Introduction to Python*, 2008.

Course Information

CPDS with Python Lab

MA26005 0-1-2 (2)

Course Outcomes

CO-1	Understand the basics of Python programming.
CO-2	Design and test programs to solve mathematical and scientific problems.
CO-3	Develop and test programs using control structures.
CO-4	Implement modular programs using functions.
CO-5	Develop Python packages.

Syllabus

Programming Exercises		
Conditional control constructs	1	
Loops	2	
Strings	3	
Lists	4	
User-defined functions and library functions	5	
Built-in functions of Matplotlib	6	
NumPy	7	
Pandas	8	

Learning Resources

Text Books

1. William Mitchell, Pavel Solin, Martin Novak *et al.*, *Introduction to Python Programming*, NCLab Public Computing, 2012.
2. C. Jacob Fredslund, *Introduction to Python Programming*, 2007.

Reference Books

1. John C. Lusth, *An Introduction to Python*, The University of Alabama, 2011.
2. Dave Kuhlman, *Introduction to Python*, 2008.

Contents

1. Introduction to Python

1.1 What is Python?

Python is a high-level, interpreted, general-purpose programming language that emphasizes code readability, simplicity, and programmer productivity. It was designed to enable programmers to write clear and logical code using a syntax that closely resembles the English language. Python supports multiple programming paradigms, including procedural programming, object-oriented programming, and functional programming, making it one of the most versatile programming languages available today.

Unlike low-level programming languages, Python allows programmers to focus on problem-solving rather than the complexities of memory management or hardware details. Consequently, it is widely used in education, research, software development, scientific computing, artificial intelligence, data science, automation, web development, and many other fields.

Python is an interpreted language, meaning that Python programs are executed line by line by the Python interpreter without requiring a separate compilation step. This feature makes program development, testing, and debugging easier and faster than many compiled programming languages.

Another important characteristic of Python is its extensive standard library, which provides built-in modules for performing mathematical computations, handling files, networking, graphical user interfaces, database management, statistical analysis, and many other tasks. In addition, thousands of external libraries developed by the open-source community extend Python's capabilities for specialized applications (pip.com).

Today, Python is one of the most popular programming languages in the world. It is extensively used by universities, research laboratories, industries, and technology companies such as Google, Microsoft, NASA, Meta, IBM, Netflix, and many others for developing scientific applications, machine learning models, data analysis tools, and large-scale software systems.

In recent years, Python has also become one of the primary programming languages for robotics, autonomous vehicles, unmanned aerial vehicles (UAVs), and intelligent control systems. Engineers use Python to process sensor data, simulate dynamical systems, develop control algorithms, visualize flight trajectories, and analyse the performance of autonomous systems.

Example: The following Python program prints a few messages that might appear when a quadrotor flight monitoring system is started.

Listing 1: Initializing a Quadrotor Flight Monitoring System

```
1
2 print("Quadrotor Flight Monitoring System")
3
4 print("Initializing Sensors...")
5
6 print("Checking Battery Status...")
7
8 print("System Ready")
```

Output

```
Quadrotor Flight Monitoring System

Initializing Sensors...

Checking Battery Status...

System Ready
```

The `print()` function is a built-in Python function used to display text, numbers, variables, and other objects on the screen. In this simple example, the program displays the status of a hypothetical quadrotor before flight. Although the program performs only simple output operations, it illustrates the basic syntax of Python. As we progress through this book, we shall extend this program by introducing variables, mathematical computations, conditional statements, loops, functions, and scientific libraries to create increasingly realistic engineering applications.

Python in Modern Engineering Modern engineering systems generate enormous amounts of data. For example, a quadrotor may record hundreds of measurements every second, including altitude, velocity, acceleration, battery voltage, GPS coordinates, orientation (roll, pitch, and yaw), and motor speeds. Python provides powerful libraries for storing, analysing, and visualizing such information. Later in these notes, we shall use

- `NumPy` to perform numerical computations,
- `Matplotlib` to visualize flight data,
- `Pandas` to analyse recorded flight logs.

Thus, the examples presented throughout this notes closely resemble the types of computational tasks encountered in scientific research and engineering practice.

Why Learn Python? Python has become the preferred programming language for scientists, mathematicians, engineers, and data analysts because it combines simplicity with powerful computational capabilities. With libraries such as `NumPy`, `SciPy`, `Matplotlib`, and `Pandas`, Python provides an excellent environment for solving mathematical problems, performing numerical computations, visualizing data, and developing scientific applications. For example, an engineer developing a quadrotor can use Python to

- compute the vehicle's trajectory,
- estimate kinetic and potential energy,
- analyse sensor measurements,
- monitor battery performance,
- visualize altitude and velocity profiles,
- develop autonomous flight algorithms.

1.2 History of Python

Python was initially developed during the late 1980s by the Dutch programmer **Guido van Rossum** at the Centrum Wiskunde & Informatica (CWI), Netherlands. The implementation of Python began in December 1989 as a hobby project during the Christmas holidays. Guido aimed to develop a programming language that was easy to learn, powerful, and capable of supporting structured and object-oriented programming.

Python was influenced by the ABC programming language, which was designed for teaching programming. Although ABC had several useful features, it lacked extensibility and was not suitable for large-scale software development. Guido adopted many of ABC's strengths while eliminating its limitations, resulting in the creation of Python.

The first official version, **Python 0.9.0**, was released in February 1991. This version already supported many features that remain fundamental to Python today, including classes, exception handling, functions, and core data types such as lists, dictionaries, and strings. Subsequent versions introduced significant improvements:

- **Python 1.0 (1994):** Added functional programming features such as `lambda`, `map()`, `filter()`, and `reduce()`.
- **Python 2.0 (2000):** Introduced garbage collection, Unicode support, list comprehensions, and many performance improvements.
- **Python 3.0 (2008):** A major redesign of the language that removed outdated features and improved consistency. Python 3 is now the standard version used worldwide.

Today, Python is developed and maintained by the **Python Software Foundation (PSF)**, an international non-profit organization dedicated to the development and promotion of Python. Python has become one of the most popular programming languages due to its simplicity, versatility, and extensive ecosystem of libraries.

Year	Event
1989	Guido van Rossum started developing Python.
1991	Python 0.9.0 released.
1994	Python 1.0 released.
2000	Python 2.0 released.
2008	Python 3.0 released.

Python and Modern Engineering The evolution of Python from a simple educational programming language to one of the world's leading scientific computing platforms has been driven largely by its growing ecosystem of numerical and machine learning libraries. Consequently, Python is now extensively used in robotics, autonomous vehicles, unmanned aerial vehicles (UAVs), digital twins, computational fluid dynamics, control engineering, and scientific machine learning. Throughout this book, we shall illustrate Python programming concepts using examples related to the monitoring and analysis of a quadrotor, thereby demonstrating how Python can be applied to solve real-world engineering problems.

1.3 Features of Python

Some of the important features of Python are described below.

1. **Simple and Easy to Learn** Python has a clean and readable syntax that resembles the English language. Programs written in Python are generally shorter and easier to understand than those written in many other programming languages.
2. **High-Level Language** Python allows programmers to concentrate on solving problems without worrying about low-level details such as memory allocation and hardware management.
3. **Interpreted Language** Python programs are executed line by line by the Python interpreter. This makes testing, debugging, and program development faster.
4. **Object-Oriented** Python supports object-oriented programming concepts such as classes, inheritance, encapsulation, and polymorphism. It also supports procedural and functional programming paradigms.
5. **Portable** Python programs can run on different operating systems such as Windows, Linux, and macOS with little or no modification.
6. **Dynamic Typing** Variables in Python do not require explicit type declarations. The data type is determined automatically during program execution.
7. **Free and Open Source** Python is freely available under an open-source license, allowing users to download, modify, and distribute it without licensing fees.
8. **Large Standard Library** Python provides a rich collection of built-in modules for mathematics, file handling, networking, databases, graphics, and many other applications.
9. **Extensive Third-Party Packages** Thousands of external packages such as NumPy, Pandas, SciPy, TensorFlow, and Matplotlib extend Python's capabilities for scientific computing, machine learning, and data analysis.
10. **Interactive Programming** Python provides an interactive shell where commands can be executed immediately, making experimentation and debugging easier.

Python has a clean and readable syntax that resembles the English language. For example, displaying the status of a quadrotor requires only a single statement.

```
1 print("Quadrotor Ready for Takeoff")
```

1.4 Applications of Python

Python is a versatile programming language used in a wide variety of academic, scientific, industrial, and commercial applications. Its simplicity and extensive library support have made it one of the most popular programming languages worldwide. Some important application areas of Python are listed below.

1. **Scientific Computing** such as NumPy, SciPy, and SymPy.

2. **Data Science and Data Analytics**
3. **Artificial Intelligence and Machine Learning**
4. **Web Development**
5. **Automation and Scripting**
6. **Software Development**
7. **Internet of Things (IoT)**
8. **Research and Engineering**
9. **Robotics and Autonomous Systems**

   Throughout this book, we shall develop programs that analyse various aspects of a quadrotor flight, including altitude, velocity, battery level, and trajectory.

1.5 Installing Python

Python can be installed free of cost from the official Python website: `https://www.python.org`. The latest stable version is recommended for new users. The basic installation procedure is as follows.

1. Download the latest Python installer for the appropriate operating system.
2. Execute the installer.
3. Select the option **Add Python to PATH**.
4. Click **Install Now**.
5. Verify the installation by opening the command prompt or terminal and typing

```
python --version
```

   or

```
python3 --version
```

   The installed Python version will be displayed if the installation is successful.

1.6 Python Interpreter

The Python interpreter is a software program that reads, translates, and executes Python statements. Since Python is an interpreted language, the source code is executed one statement at a time without requiring a separate compilation process. Python programs can be executed in two different modes.

1. **Interactive Mode:** Commands are entered individually and are executed immediately. This mode is useful for testing small pieces of code and performing simple calculations.
2. **Script Mode:** Python statements are written in a file with the extension `.py`. The complete program is then executed by the interpreter.

1.7 Integrated Development Environments (IDEs)

An Integrated Development Environment (IDE) is a software application that provides a convenient environment for writing, executing, debugging, and managing Python programs. Most IDEs include a code editor, syntax highlighting, auto-completion, debugging tools, and an integrated terminal. Some commonly used Python IDEs are listed below.

IDE	**Application**
IDLE	Default IDE bundled with Python.
Visual Studio Code (VS Code)	Lightweight editor with powerful extensions.
PyCharm	Professional IDE for Python development.
Jupyter Notebook	Interactive computing and scientific programming.
Google Colab	Cloud-based Python programming environment.
Spyder	Scientific computing and numerical analysis.

1.8 Writing and Executing Your First Python Program

After installing Python, the first step is to write and execute a simple Python program. Traditionally, the first program written in any programming language is the **Hello World** program. It demonstrates the basic syntax of the language and verifies that the Python installation is functioning correctly.

A Python program consists of one or more statements that are executed sequentially by the Python interpreter.

Example 1: Some basics

Listing 2: quadrotor

```
1
2  print("Quadrotor Flight Monitoring System")
3
4  print("Initializing Sensors...")
5
6  print("Checking Battery Status...")
7
8  print("System Ready")
9
10 print("Ready for Takeoff")
```

Output

```
Quadrotor Flight Monitoring System

Initializing Sensors...

Checking Battery Status...
```

```
System Ready

Ready for Takeoff
```

The `print()` function is a built-in Python function used to display text, numbers, variables, and other objects on the screen. In this example, the program simply prints a sequence of status messages similar to those that might appear while initializing a quadrotor before flight. Although the program performs only simple output operations, it illustrates the basic syntax of a Python program.

Writing a Python Program Python programs can be written using any text editor or Integrated Development Environment (IDE).

1. Open an IDE such as VS Code, IDLE, or PyCharm.
2. Create a new Python file.
3. Write the Python program.
4. Save the file with the extension `.py`.
5. Execute the program.

For example, save the previous program as

`quadrotor_startup.py`

The program may be executed from the command prompt using

```
python quadrotor_startup.py
```

or

```
python3 quadrotor_startup.py
```

depending on the operating system.

Example 2: Displaying Flight Parameters The `print()` function can also display numerical quantities.

Listing 3: Printing Flight Parameters

```
print("Altitude =",12.5,"m")

print("Velocity =",4.8,"m/s")

print("Battery =",95,"%")
```

Output

```
Altitude = 12.5 m

Velocity = 4.8 m/s

Battery = 95 %
```

The `print()` function can display integers, floating-point numbers, strings, Boolean values, variables, and mathematical expressions. Throughout this book, we shall frequently use `print()` to display intermediate results obtained during numerical computations, simulations, and engineering applications.

1.9 Structure of a Python Program

A Python program is a sequence of statements executed one after another. Unlike many programming languages, Python does not use braces ({}) to define program blocks. Instead, it uses **indentation**, which improves the readability of the program.

A simple Python program generally consists of the following components.

1. Comments
2. Import statements (optional)
3. Variable declarations
4. Processing statements
5. Input statements (optional)
6. Output statements

The general structure of a Python program is illustrated below.

Listing 4: General Structure of a Python Program

```
import math

mass = 2.5          # kg
altitude = 12.5     # m
velocity = 4.8      # m/s

potential_energy = mass * 9.81 * altitude
kinetic_energy = 0.5 * mass * velocity**2

print("Potential Energy =", potential_energy)

print("Kinetic Energy =", kinetic_energy)
```

The execution of a Python program begins with the first statement and continues sequentially until the last statement unless the normal execution flow is modified by conditional statements, loops, or function calls.

The above program demonstrates the basic organization of a Python program. First, the required modules are imported, followed by the declaration of variables. The processing section performs the required computations, and finally the results are displayed using the `print()` function.

Importance of Indentation Indentation refers to the spaces placed before a statement. Python uses indentation to identify blocks of code. Unlike many programming languages, indentation is not merely a formatting convention; it is a fundamental part of the Python syntax. The following example illustrates correct indentation.

Listing 5: Correct Indentation

```
battery = 18

if battery < 20:
    print("Low Battery Warning")
    print("Initiate Landing Procedure")

print("Flight Monitoring Continues")
```

Here, both `print()` statements inside the `if` block are indented by the same number of spaces, indicating that they belong to the conditional statement. If indentation is incorrect, Python generates an **IndentationError**.

Listing 6: Incorrect Indentation

```
battery = 18

if battery < 20:
print("Low Battery Warning")
```

Output

```
IndentationError:
expected an indented block after 'if' statement
```

Hence, proper indentation is mandatory in every Python program. Good indentation not only prevents syntax errors but also improves the readability, maintainability, and reliability of scientific and engineering software.

1.10 Comments and Documentation

Comments are explanatory statements that improve the readability, maintainability, and documentation of a program. They are ignored by the Python interpreter during execution and therefore do not affect the program's output. Comments are useful for explaining the purpose of the program, describing variables, and documenting algorithms. Python supports two common forms of documentation.

Single-Line Comments A single-line comment begins with the symbol #.

Listing 7: Single-Line Comments

```
# Quadrotor Flight Parameters

mass = 2.5          # kg
altitude = 12.5     # m
velocity = 4.8      # m/s

# Compute potential energy
potential_energy = mass * 9.81 * altitude

print("Potential Energy =", potential_energy)
```

Everything following the symbol # is treated as a comment and is ignored by the Python interpreter during execution.

Multi-Line Documentation (Docstrings) Python commonly uses triple quotation marks to write documentation strings, known as **docstrings**. Docstrings are primarily used to document modules, functions, classes, and large programs.

Listing 8: Documentation String

```
"""
Program:
Quadrotor Flight Parameters

Author:
Dr. Sivalingam S M

Description:
Stores the values of quadrotors parameters such as mass, altitude, velocity
"""
```

Unlike ordinary comments, docstrings provide structured documentation and are widely used in professional software development.

Good Programming Practice When writing Python programs, the following practices should be followed.

- Write comments that explain *why* something is done rather than simply describing what the code already shows.
- Avoid excessive comments and Update comments.
- Use docstrings to document modules, functions, classes, and large programs.
- Use meaningful variable names together with appropriate comments.

Example The following program illustrates the use of comments in a simple quadrotor flight analysis.

Listing 9: Program with Comments

```
# Quadrotor Parameters

mass = 2.5          # kg
velocity = 6.2      # m/s

# Compute kinetic energy
kinetic_energy = 0.5 * mass * velocity**2

# Display the result
print("Kinetic Energy =", kinetic_energy)
```

Output

```
Kinetic Energy = 48.05
```

The comments improve the readability of the program without affecting its execution. In large scientific computing applications, good documentation is essential for collaboration, debugging, and long-term software maintenance.

1.11 Python Keywords

Keywords are predefined words that have special meanings in the Python programming language. They are reserved by the Python interpreter to perform specific tasks such as defining functions, creating loops, making decisions, handling exceptions, and importing modules. Since keywords have predefined meanings, they **cannot be used as identifiers** (variable names, function names, class names, etc.).

Example: Incorrect Use of Keywords The following statements are invalid because the words `if` and `for` are reserved Python keywords.

Listing 10: Incorrect Use of Keywords

```
if = "Takeoff"

for = "Landing"
```

The above program generates a **SyntaxError** because Python keywords cannot be redefined or used as variable names.

Displaying Python Keywords Python provides a built-in module named `keyword` that can be used to display all reserved keywords.

Listing 11: Displaying Python Keywords

```
import keyword

print(keyword.kwlist)
```

A part of the output is shown below.

```
['False', 'None', 'True', 'and', 'as', 'assert',
'async', 'await', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

Many of these keywords will be introduced gradually throughout this notes. For example,

- `if` will be used to decide whether the quadrotor should continue its mission or perform an emergency landing.
- `for` and `while` will be used to continuously monitor sensor measurements during flight.
- `def` will be used to create reusable functions for computing flight parameters.
- `return` will be used to return computed quantities such as velocity, energy, and battery status from functions.

Some of the frequently used Python keywords are listed in Table 1.

Table 1: Common Python Keywords

Keyword	**Purpose**
if	Decision making
elif	Additional condition
else	Alternative block
for	Iteration over sequences
while	Conditional loop
break	Terminates a loop
continue	Skips the current iteration
pass	Null statement
def	Defines a function
return	Returns a value from a function
import	Imports a module
class	Defines a class
try	Begins exception handling
except	Handles exceptions

finally	Executes regardless of exceptions
True	Boolean true value
False	Boolean false value
None	Represents no value

- Python keywords are case-sensitive.
- Keywords cannot be used as identifiers.
- The list of keywords may change slightly between different Python versions.

1.12 Identifiers and Naming Conventions

An **identifier** is a name assigned to a variable, function, class, module, or any other user-defined object in Python. Identifiers enable the programmer to refer to objects during program execution.

Choosing meaningful identifiers is one of the most important aspects of good programming practice. Well-chosen names make programs easier to read, understand, debug, and maintain. Throughout this book, we shall use descriptive identifiers related to a quadrotor flight monitoring system so that the purpose of each variable is immediately clear. For example,

Listing 12: Examples of Identifiers

```
altitude = 12.5

velocity = 4.8

kinetic_energy = 0.5 * 2.5 * velocity**2

print(kinetic_energy)
```

Here,

- `altitude` is an identifier.
- `velocity` is an identifier.
- `kinetic_energy` is an identifier.
- `print` is a built-in Python function.

Rules for Naming Identifiers A valid Python identifier must satisfy the following rules.

1. An identifier may contain letters (`A-Z`, `a-z`), digits (`0-9`), and the underscore (_).
2. The first character must be either a letter or an underscore.
3. An identifier cannot begin with a digit.
4. Spaces are not permitted.

5. Special characters such as

   `@ # $ % ! &`

   cannot be used.

6. Keywords cannot be used as identifiers.

7. Python identifiers are case-sensitive.

Identifier	**Valid?**	**Reason**
altitude	Yes	Starts with a letter
flight_velocity	Yes	Uses underscore
battery1	Yes	Ends with a digit
_sensor	Yes	Starts with underscore
2motor	No	Starts with a digit
motor-speed	No	Contains '-'
battery level	No	Contains space
for	No	Python keyword

Case Sensitivity Python distinguishes between uppercase and lowercase letters.

Listing 13: Case Sensitivity

```
Altitude = 15

altitude = 20

print(Altitude)

print(altitude)
```

Output

```
15
20
```

Thus, `Altitude` $\neq$ `altitude`. These are treated as two completely different identifiers by the Python interpreter.

Naming Conventions Although Python allows many naming styles, programmers follow certain conventions to improve code readability and maintain consistency.

Object	Convention
Variables	lower_case
Functions	lower_case
Constants	UPPER_CASE
Classes	PascalCase
Modules	lowercase
Packages	lowercase

Examples Meaningful identifiers clearly describe the quantity or object they represent. This makes the program easier to understand and maintain.

Listing 14: Meaningful Identifiers

```
quadrotor_altitude = 15.2

flight_velocity = 6.5

battery_percentage = 94

motor_speed = 5200

flight_time = 120

gps_coordinates = (17.978, 79.594)
```

From the variable names alone, a reader can immediately understand what each quantity represents.

Bad Examples

Listing 15: Poor Naming Practices

```
a = 15.2

b = 6.5

x1 = 94

abc = 5200

temp = 120
```

Although these identifiers are syntactically correct, they provide little or no information about the quantities they represent. Such naming practices make programs difficult to understand, debug, and maintain, especially in large scientific computing projects.