Engineering Computation

A Tutorial for Beginners in Computation for Applied Mechanics

Serhat Beyenir

14 August 2025

Contents

List of Figures

List of Tables

Preface

This book presents a collection of lecture notes on engineering computation, designed to provide learners with succinct yet essential insights into key topics covered in class. Each chapter is accompanied by a problem set to facilitate comprehension and reinforce understanding.

Chapter 2: The International System of Units (SI) is the globally accepted standard for measurement. Established to provide a consistent framework for scientific and technical measurements, SI units facilitate clear communication and data comparison across various fields and countries. The system is based on seven fundamental units: the meter for length, the kilogram for mass, the second for time, the ampere for electric current, the kelvin for temperature, the mole for substance, and the candela for luminous intensity.

Part I

Beginning

Chapter 1

Python Tutorial

This tutorial introduces Python programming, covering basic concepts with examples to illustrate key points. We will start by using Python as a calculator, then explore variables, functions, and control flow.

1.1 Requirements

To follow this tutorial, you must have Python (version 3.10 or later) installed on your computer. Python is available for Windows, macOS, and Linux. Additionally, ensure you have a text editor or an Integrated Development Environment (IDE) to write Python code. We recommend Positron, a user-friendly IDE with a built-in terminal for running Python scripts, though other editors like VS Code or PyCharm are also suitable.

1.2 Basic Syntax

Python uses indentation (typically four spaces) to define code blocks. A colon (`:`) introduces a block, and statements within the block must be indented consistently. Python is case-sensitive, so `Variable` and `variable` are distinct identifiers. Statements typically end with a newline, but you can use a backslash (\) to continue a statement across multiple lines.

```
total = 1 + 2 + 3 + \
        4 + 5
print(total)  # Output: 15
```

Basic syntax rules:

- Comments start with `#` and extend to the end of the line.

- Strings can be enclosed in single quotes (`'`), double quotes (`"`), or triple quotes (`'''` or `"""`) for multi-line strings.
- Python is case-sensitive, so `Variable` and `variable` are different identifiers.

1.3 The `print()` Function

The `print()` function displays output in Python.

```
name = "Rudolf Diesel"
year = 1858
print(f"{name} was born in {year}.")
```

Output: `Rudolf Diesel was born in 1858.`

1.4 Formatting in `print()`

The following table illustrates common f-string formatting options for the `print()` function:

Format	Code	Example	Output
Round to 2 decimals	`f"{x:.2f}"`	`print(f"{3.14159:.2f}")`	`3.14`
Round to whole number	`f"{x:.0f}"`	`print(f"{3.9:.0f}")`	`4`
Thousands separator	`f"{x:,.2f}"`	`print(f"{1234567.89:,.2f}")`	`1,234,567.89`
Percentage	`f"{x:.1%}"`	`print(f"{0.756:.1%}")`	`75.6%`
Currency style	`f"${x:,.2f}"`	`print(f"${1234.5:,.2f}")`	`$1,234.50`

Note: The currency symbol (e.g., `$`) can be modified for other currencies (e.g., `€`, `£`) based on the desired locale.

1.5 Variables and Data Types

Variables store data and are assigned values using the `=` operator.

```
x = 10
y = 3.14
name = "Rudolph"
```

Python has several built-in data types, including:

- Integers (`int`): Whole numbers, e.g., `10`, `-5`

- Floating-point numbers (`float`): Decimal numbers, e.g., `3.14`, `-0.001`
- Strings (`str`): Text, e.g., `"Hello"`, `'World'`
- Booleans (`bool`): `True` or `False`

1.5.1 Arithmetic Operations

```
a = 10
b = 3
print(a + b)  # Addition: 13
print(a - b)  # Subtraction: 7
print(a * b)  # Multiplication: 30
print(a / b)  # Division: 3.3333...
print(a // b) # Integer Division: 3
print(a ** b) # Exponentiation: 1000
```

1.5.2 String Operations

```
first_name = "Rudolph"
last_name = "Diesel"
full_name = first_name + " " + last_name  # Concatenation using +
print(full_name)  # Output: Rudolph Diesel
print(f"{first_name} {last_name}")  # Concatenation using f-string
print(full_name * 2)  # Repetition: Rudolph DieselRudolph Diesel
print(full_name.upper())  # Uppercase: RUDOLPH DIESEL
```

Note: String repetition (`*`) concatenates the string multiple times without spaces. For example, `full_name * 2` produces `Rudolph DieselRudolph Diesel`.

1.6 Python as a Calculator in Interactive Mode

Python's interactive mode allows you to enter commands and see results immediately, ideal for quick calculations. To start, open a terminal (on macOS, Linux, or Windows) and type:

```
python3  # Use 'python' on Windows if 'python3' is not recognized
```

You should see the Python prompt:

```
>>>
```

Enter expressions and press **Enter** to see results:

```
2 + 3  # Output: 5
7 - 4  # Output: 3
6 * 9  # Output: 54
8 / 2  # Output: 4.0
```

```
8 // 2 # Output: 4
2 ** 3 # Output: 8
```

1.6.1 Parentheses for Grouping

```
(2 + 3) * 4  # Output: 20
2 + (3 * 4)  # Output: 14
```

1.6.2 Variables

```
x = 10
y = 3
x / y  # Output: 3.3333333333333335
```

1.6.3 Exiting Interactive Mode

To exit, type:

```
exit()
```

Alternatively, use: - **Ctrl+D** (macOS/Linux) - **Ctrl+Z** then Enter (Windows)

1.7 Control Flow

Control flow statements direct the execution of code based on conditions.

1.7.1 Conditional Statements

Conditional statements allow you to execute different code blocks based on specific conditions. Python provides three keywords for this purpose:

- `if`: Evaluates a condition and executes its code block if the condition is `True`.
- `elif`: Short for "else if," it checks an additional condition if the preceding `if` or `elif` conditions are `False`. You can use multiple `elif` statements to test multiple conditions sequentially, and Python will execute the first `True` condition's block, skipping the rest.
- `else`: Executes a code block if none of the preceding `if` or `elif` conditions are `True`. It serves as a fallback and does not require a condition.

The following example uses age to categorize a person as a Minor, Adult, or Senior, demonstrating how `if`, `elif`, and `else` work together.

```
# Categorize a person based on their age
age = 19
if age < 18:
```

```
    print("Minor")
elif age <= 64:
    print("Adult")
else:
    print("Senior")
```

Output: `Adult`

1.7.2 For Loop

A `for` loop iterates over a sequence (e.g., list or string).

```
components = ["piston", "liner", "connecting rod"]
for component in components:
    print(component)
```

Output:

```
piston
liner
connecting rod
```

1.7.3 While Loop

A `while` loop executes as long as a condition is true. Ensure the condition eventually becomes false to avoid infinite loops.

```
count = 0
while count <= 5:
    print(count)
    count += 1
```

Output:

```
0
1
2
3
4
5
```

1.8 Functions

1.8.1 The `def` Keyword

Functions are reusable code blocks defined using the `def` keyword. They can include default parameters for optional arguments.

```
def add(a, b=0):
    return a + b
print(add(5))      # Output: 5
print(add(5, 3))   # Output: 8

def multiply(*args):
    result = 1
    for num in args:
        result *= num
    return result
print(multiply(2, 3, 4))  # Output: 24
```

1.8.2 The `lambda` Keyword

The `lambda` keyword creates anonymous functions for short, one-off operations, often used in functional programming.

```
celsius_to_fahrenheit = lambda c: (c * 9 / 5) + 32
print(celsius_to_fahrenheit(25))  # Output: 77.0
```

1.9 The `math` Module

The `math` module provides mathematical functions and constants.

```
import math
print(math.sqrt(16))  # Output: 4.0
print(math.pi)        # Output: 3.141592653589793
```

```
import math
angle = math.pi / 4  # 45 degrees in radians
print(math.sin(angle))  # Output: 0.7071067811865475 (approximately √2/2)
print(math.cos(angle))  # Output: 0.7071067811865476 (approximately √2/2)
print(math.tan(angle))  # Output: 1.0
```

Note: Floating-point arithmetic may result in small precision differences, as seen in the `sin` and `cos` outputs.

```
import math
print(math.log(10))       # Natural logarithm of 10: 2.302585092994046
print(math.log(100, 10))  # Logarithm of 100 with base 10: 2.0
```

1.9.1 Converting Between Radians and Degrees

The `math` module provides `math.radians()` to convert degrees to radians and `math.degrees()` to convert radians to degrees, which is useful for trigonometric calculations.

```
import math
degrees = 180
radians = math.radians(degrees)
print(f"{degrees} degrees is {radians:.3f} radians")  # Output: 180 degrees is 3.142 radians

radians = math.pi / 2
degrees = math.degrees(radians)
print(f"{radians:.3f} radians is {degrees:.1f} degrees")  # Output: 1.571 radians is 90.0 degrees
```

1.10 Writing Python Scripts

Write Python code in a `.py` file and run it as a script. Create a file named `script.py`:

```
# script.py
import math
print("Square root of 16 is:", math.sqrt(16))
print("Value of pi is:", math.pi)
print("Sine of 90 degrees is:", math.sin(math.pi / 2))
print("Natural logarithm of 10 is:", math.log(10))
print("Logarithm of 100 with base 10 is:", math.log(100, 10))
```

To run the script, open a terminal, navigate to the directory containing `script.py` using the `cd` command (e.g., `cd /path/to/directory`), and type:

```
python3 script.py  # or python script.py on Windows
```

Output:

```
Square root of 16 is: 4.0
Value of pi is: 3.141592653589793
Sine of 90 degrees is: 1.0
Natural logarithm of 10 is: 2.302585092994046
Logarithm of 100 with base 10 is: 2.0
```

1.11 Summary

This tutorial covered Python basics, including syntax, variables, data types, operations, control flow, and functions. Python's rich ecosystem includes libraries like:

- **NumPy**: For numerical computations and array manipulations.
- **Matplotlib**: For data visualization and plotting.
- **Pandas**: For data manipulation and analysis with tabular data structures.
- **Pint**: For handling physical quantities and performing unit conversions.

You can explore these libraries to enhance your Python programming skills further. For example installing them can be done using `pip`:

```
pip install numpy matplotlib pandas pint
```

`pip` is Python's package manager for installing and managing additional libraries.

Chapter 2

International System of Units

2.1 SI Units

The International System of Units (SI) is the globally accepted standard for measurement. Established to provide a consistent framework for scientific and technical measurements, SI units facilitate clear communication and data comparison across various fields and countries. The system is based on seven fundamental units: the meter for length, the kilogram for mass, the second for time, the ampere for electric current, the kelvin for temperature, the mole for substance, and the candela for luminous intensity.

Table 2.1: Base SI units.

Physical Quantity	SI Base Unit	Symbol
Length	Meter	m
Mass	Kilogram	kg
Time	Second	s
Electric Current	Ampere	A
Temperature	Kelvin	K
Amount of Substance	Mole	mol
Luminous Intensity	Candela	cd

Table 2.2: Derived SI units.

Physical Quantity	Derived SI Unit	Symbol
Area	Square meter	m^2
Volume	Cubic meter	m^3
Speed	Meter per second	m/s
Acceleration	Meter per second squared	m/s^2
Force	Newton	N
Pressure	Pascal	Pa
Energy	Joule	J
Power	Watt	W
Electric Charge	Coulomb	C
Electric Potential	Volt	V
Resistance	Ohm	Ω
Capacitance	Farad	F
Frequency	Hertz	Hz
Luminous Flux	Lumen	lm
Illuminance	Lux	lx
Specific Energy	Joule per kilogram	J/kg
Specific Heat Capacity	Joule per kilogram Kelvin	$J/(kg \cdot K)$

Table 2.3: Common multiples and submultiples for SI units.

Factor	Prefix	Symbol
10^{9}	giga	G
10^{6}	mega	M
10^{3}	kilo	k
10^{2}	hecto	h
10^{1}	deca	da
10^{-1}	deci	d
10^{-2}	centi	c
10^{-3}	milli	m
10^{-6}	micro	µ

2.2 SI System Rules and Common Mistakes

Using the SI system correctly is crucial for clear communication in science and engineering. Below are common mistakes in using the SI system, examples of incorrect usage, and how to correct them.

Table 2.4: SI system rules and common mistakes

Concept	**Mistake**	**Correct Usage**	**Notes**
Use of SI Unit Symbols	`m./s`	`m/s`	Use the correct format without additional punctuation.
Spacing Between Value & Unit	`10kg`	`10 kg`	Always leave a space between the number and the unit symbol.
Incorrect Unit Symbols	`sec`, `hrs`, `°K`	`s`, `h`, `K`	Use the proper SI symbols; symbols are case-sensitive.
Abbreviations for Units	`5 kilograms (kgs)`	`5 kilograms (kg)`	Avoid informal abbreviations like "kgs"; adhere to standard symbols.
Multiple Units in Expressions	`5 m/s/s`, `5 kg/meter²`	`5 m/s²`, `5 kg/m²`	Use compact, standardized formats for derived units.

Concept	Mistake	Correct Usage	Notes
Incorrect Use of Prefixes	`0.0001 km`	`100 mm`	Choose prefixes to keep numbers in the range (0.1 x < 1000).
Misplaced Unit Symbols	`5/s`, `kg10`	`5 s ⁻¹`, `10 kg`	Symbols must follow numerical values, not precede them.
Degrees Celsius vs. Kelvin	`300°K`	`300 K`	Kelvin is written without "degree"
Singular vs. Plural Units	`5 kgs`, `1 meters`	`5 kg`, `1 meter`	Symbols do not pluralize; full unit names follow grammar rules.
Capitalization of Symbols	`Kg`, `S`, `Km`, `MA`	`kg`, `s`, `km`, `mA`	Symbols are case-sensitive; use uppercase only where specified (e.g., `N`, `Pa`).
Capitalization of Unit Names	`Newton`, `Pascal`, `Watt`	`newton`, `pascal`, `watt`	Unit names are lowercase, even if derived from a person's name, unless starting a sentence.
Prefix Capitalization	`MilliMeter`, `MegaWatt`	`millimeter`, `megawatt`	Prefixes are lowercase for (10^{-1}) to (10^{-9}), uppercase for (10^6) and larger (except `k` for kilo).
Formatting in Reports	`5`, `Temperature: 300`	`5 kg`, `Temperature: 300 K`	Always specify units explicitly.

2.3 Unity Fraction

The **unity fraction** method, or **unit conversion using unity fractions**, is a systematic way to convert one unit of measurement into another. This method

relies on multiplying by fractions that are equal to one, where the numerator and the denominator represent the same quantity in different units. Since any number multiplied by one remains the same, unity fractions allow for seamless conversion without changing the value.

The principle of unity fractions is based on:

1. **Setting up equal values**: Write a fraction where the numerator and denominator are equivalent values in different units, so the fraction equals one. For example, $\frac{1km}{1000m}$ is a unity fraction because 1 km equals 1000 m.
2. **Multiplying by unity fractions**: Multiply the initial quantity by the unity fraction(s) so that the undesired units cancel out, leaving only the desired units.

2.4 Classwork

Example 2.1. Suppose we want to convert 5 kilometers to meters.

1. Start with 5 kilometers:
$$5\,\text{km}$$
2. Multiply by a unity fraction that cancels kilometers and introduces meters. We use $(\frac{1000\,\text{m}}{1\,\text{km}}), since\ 1\,\text{km} = 1000\,\text{m}$:
$$5\,\text{km} \times \frac{1000\,\text{m}}{1\,\text{km}} = 5000\,\text{m}$$
3. The kilometers km cancel out, leaving us with meters m:
$$5\,\text{km} = 5000\,\text{m}$$

This step-by-step approach illustrates how the unity fraction cancels the undesired units and achieves the correct result in meters.

Unity fractions can be extended by using multiple conversion steps. For example, converting hours to seconds would require two unity fractions: one to convert hours to minutes and another to convert minutes to seconds. This approach ensures accuracy and is widely used in science, engineering, and other fields that require precise unit conversions.

Example 2.2. Convert 15 m/s to km/h.

1. Start with 15 m/s.
2. To convert meters to kilometers, multiply by $\frac{1\,\text{km}}{1000\,\text{m}}$.
3. To convert seconds to hours, multiply by $\frac{3600\,\text{s}}{1\,\text{h}}$.

$$15\,\text{m/s} \times \frac{1\,\text{km}}{1000\,\text{m}} \times \frac{3600\,\text{s}}{1\,\text{h}} = 54\,\text{km/h}$$

The meters and seconds cancel out, leaving kilometers per hour: $54\,\text{km/h}$.

2.5 Problem Set

Instructions:

1. Use unity fraction to convert between derived SI units.
2. Show each step of your work to ensure accuracy.
3. Simplify your answers and include correct units.

---

1. **Speed**
   Convert $72\,\text{km/h}$ to m/s.
2. **Force**
   Convert $980\,\text{N}$ (newtons) to $\text{kg} \cdot \text{m/s}^2$.
3. **Energy**
   Convert $2500\,\text{J}$ (joules) to kJ.
4. **Power**
   Convert $1500\,\text{W}$ (watts) to kW.
5. **Pressure**
   Convert $101325\,\text{Pa}$ (pascals) to kPa.
6. **Volume Flow Rate**
   Convert $3\,\text{m}^3/\text{min}$ to L/s.
7. **Density**
   Convert $1000\,\text{kg/m}^3$ to g/cm^3.
8. **Acceleration**
   Convert $9.8\,\text{m/s}^2$ to cm/s^2.
9. **Torque**
   Convert $50\,\text{N} \cdot \text{m}$ to $\text{kN} \cdot \text{cm}$.
10. **Frequency**
    Convert $500\,\text{Hz}$ (hertz) to kHz.
11. **Work to Energy Conversion**
    A force of $20\,\text{N}$ moves an object $500\,\text{cm}$. Convert the work done to joules.
12. **Kinetic Energy Conversion**
    Calculate the kinetic energy in kilojoules of a $1500\,\text{kg}$ car moving at $72\,\text{km/h}$.

13. **Power to Energy Conversion**
    A machine operates at 2 kW for 3 hours. Convert the energy used to megajoules.

14. **Pressure to Force Conversion**
    Convert a pressure of 200 kPa applied to an area of $0.5\,\mathrm{m}^2$ to force in newtons.

15. **Density to Mass Conversion**
    Convert $0.8\,\mathrm{g/cm}^3$ for an object with a volume of $250\,\mathrm{cm}^3$ to mass in grams.

2.5.1 Answer Key

1. $72\,\mathrm{km/h} = 20\,\mathrm{m/s}$
2. $980\,\mathrm{N} = 980\,\mathrm{kg \cdot m/s}^2$
3. $2500\,\mathrm{J} = 2.5\,\mathrm{kJ}$
4. $1500\,\mathrm{W} = 1.5\,\mathrm{kW}$
5. $101325\,\mathrm{Pa} = 101.325\,\mathrm{kPa}$
6. $3\,\mathrm{m}^3/\mathrm{min} = 50\,\mathrm{L/s}$
7. $1000\,\mathrm{kg/m}^3 = 1\,\mathrm{g/cm}^3$
8. $9.8\,\mathrm{m/s}^2 = 980\,\mathrm{cm/s}^2$
9. $50\,\mathrm{N \cdot m} = 5\,\mathrm{kN \cdot cm}$
10. $500\,\mathrm{Hz} = 0.5\,\mathrm{kHz}$
11. $20\,\mathrm{N} \times 5\,\mathrm{m} = 100\,\mathrm{J}$
12. Kinetic energy $= 1500\,\mathrm{kg} \times (20\,\mathrm{m/s})^2/2 = 300\,\mathrm{kJ}$
13. $2\,\mathrm{kW} \times 3\,\mathrm{hours} = 21.6\,\mathrm{MJ}$
14. $200\,\mathrm{kPa} \times 0.5\,\mathrm{m}^2 = 100,000\,\mathrm{N}$
15. $0.8\,\mathrm{g/cm}^3 \times 250\,\mathrm{cm}^3 = 200\,\mathrm{g}$

2.6 Further Reading

Introduction in Russell et al. (2021) and SI units in Bolton (2021) for additional information.

Chapter 3

Advanced Python Tutorial

This tutorial builds upon the foundational concepts introduced in the basic Python tutorial, focusing on more advanced topics. It covers virtual environments for project isolation, installing external libraries using `pip`, and applying these skills to build a unit conversion script with the Pint library. Examples are provided to demonstrate practical implementation, including conversions for speed and pressure units.

3.1 Requirements

To follow this tutorial, ensure you have Python (version 3.10 or later) installed on your computer, as detailed in the basic tutorial. You will also need access to a terminal or command prompt for creating virtual environments and installing libraries. No additional IDE is required beyond what was recommended previously, though Positron or VS Code remains suitable.

3.2 Virtual Environments (.venv)

Virtual environments in Python allow you to create isolated spaces for projects, ensuring that dependencies (libraries and versions) do not conflict across different projects. This is particularly useful when working on multiple applications that require different library versions.

3.2.1 Why Use Virtual Environments?

- **Isolation**: Each project can have its own set of installed packages without affecting the global Python installation.
- **Reproducibility**: Share your project's dependencies easily via a `requirements.txt` file.

- **Cleanliness**: Avoid cluttering your system Python with project-specific libraries.

3.2.2 Creating a Virtual Environment

To create a virtual environment named `.venv` in your project directory, open a terminal and navigate to the desired folder, then run:

```
python -m venv .venv
```

This command generates a `.venv` directory containing an isolated Python interpreter and `pip`.

3.2.3 Activating the Virtual Environment

Activation makes the virtual environment's Python and `pip` the default for your terminal session.

- On macOS/Linux:

  ```
  source .venv/bin/activate
  ```

- On Windows:

  ```
  .venv\Scripts\activate
  ```

Once activated, your terminal prompt will typically show `(.venv)` to indicate the active environment.

3.2.4 Deactivating the Virtual Environment

To exit the virtual environment and return to the global Python, simply run:

```
deactivate
```

3.3 Installing Libraries with pip

`pip` is Python's package installer, used to download and install libraries from the Python Package Index (PyPI). Within an activated virtual environment, installations are confined to that environment.

3.3.1 Installing Pint

Pint is a library for handling physical quantities and unit conversions, ensuring dimensional consistency in calculations.

To install Pint, activate your virtual environment (as described above) and run:

```
pip install pint
```

This command downloads and installs Pint and its dependencies. To verify the installation, start Python in interactive mode (e.g., `python`) and import Pint:

```
import pint
```

If no errors occur, the installation is successful.

3.3.2 Managing Dependencies

To save your project's dependencies (e.g., for sharing), generate a `requirements.txt` file:

```
pip freeze > requirements.txt
```

Others can recreate the environment by installing from this file:

```
pip install -r requirements.txt
```

3.3.3 Sample Requirements File

A `requirements.txt` file lists the libraries and their versions required for a project. Below is an example for a project using Pint:

```
pint>=0.23
```

Save this content in a file named `requirements.txt` in your project directory. You can install these dependencies in a new virtual environment using `pip install -r requirements.txt`. This ensures consistent library versions across different setups.

3.4 Building a Unit Conversion Script with Pint

Pint simplifies unit conversions by associating units with numerical values, automatically handling conversions and ensuring compatibility (e.g., preventing addition of length and mass).

3.4.1 Basic Usage of Pint

First, import Pint and create a `UnitRegistry` to manage units:

```
from pint import UnitRegistry

ureg = UnitRegistry()

# Define a quantity with units
length = 2.5 * ureg.meter

# Convert to another unit
```

```
length_in_feet = length.to(ureg.foot)
print(length_in_feet)  # Output: 8.202099737532808 foot
```

Pint supports a wide range of units, including length, mass, temperature, speed, and pressure. For temperature conversions, use the `.to()` method carefully, as some require delta considerations for differences versus absolute values.

3.4.2 Example: Temperature Conversion

```
from pint import UnitRegistry

ureg = UnitRegistry()

# Absolute temperature conversion
temp_c = 100 * ureg.degC
temp_f = temp_c.to(ureg.degF)
print(temp_f)  # Output: 212.0 degF

# Delta temperature (for differences)
delta_c = 10 * ureg.delta_degC
delta_f = delta_c.to(ureg.delta_degF)
print(delta_f)  # Output: 18.0 delta_degF
```

3.4.3 Example: Additional Unit Conversions

Pint also supports conversions for speed and pressure units, such as knots to kilometers per hour (km/h), pounds per square inch (psi) to kilopascals (kPa), and bar to psi, which are common in aviation, engineering, and industrial applications.

```
from pint import UnitRegistry

ureg = UnitRegistry()

# Speed: knots to km/h
speed = 100 * ureg.knot
speed_kmh = speed.to(ureg.km_per_hour)
print(speed_kmh)  # Output: 185.2 kilometer / hour

# Pressure: psi to kPa
pressure_psi = 50 * ureg.psi
pressure_kpa = pressure_psi.to(ureg.kPa)
print(pressure_kpa)  # Output: 344.737864655216 kilopascal

# Pressure: bar to psi
```

```
pressure_bar = 2 * ureg.bar
pressure_psi = pressure_bar.to(ureg.psi)
print(pressure_psi)  # Output: 29.0075477738527 psi
```

3.4.4 Building the Script

Create a file named **unit_converter.py** in your project directory. The following script provides a command-line interface for converting various units (e.g., length, temperature, speed, pressure) using Pint. Activate your virtual environment, ensure Pint is installed, and add the code below:

```
# unit_converter.py
from pint import UnitRegistry, UndefinedUnitError, DimensionalityError
import sys

def main():
    ureg = UnitRegistry()

    if len(sys.argv) != 4:
        print("Usage: python unit_converter.py <value> <from_unit> <to_unit>")
        print("Examples:")
        print("  python unit_converter.py 2.5 meter foot")
        print("  python unit_converter.py 100 degC degF")
        print("  python unit_converter.py 100 knot km_per_hour")
        print("  python unit_converter.py 50 psi kPa")
        print("  python unit_converter.py 2 bar psi")
        sys.exit(1)

    try:
        value = float(sys.argv[1])
        from_unit = sys.argv[2]
        to_unit = sys.argv[3]

        quantity = value * ureg(from_unit)
        converted = quantity.to(ureg(to_unit))

        print(f"{value} {from_unit} is {converted.magnitude} {converted.units}")

    except (UndefinedUnitError, DimensionalityError) as e:
        print(f"Error: {e}")
        print("Ensure units are valid and compatible (e.g., length to length, pressure to pressur
    except ValueError:
        print("Error: The value must be a number.")

if __name__ == "__main__":
```

```
    main()
```

3.4.5 Running the Script

Navigate to your project directory in the terminal, activate the virtual environment, and run:

```
python unit_converter.py 2.5 meter foot
```

Output: `2.5 meter is 8.202099737532808 foot`

For temperature:

```
python unit_converter.py 100 degC degF
```

Output: `100 degC is 212.0 degF`

For speed:

```
python unit_converter.py 100 knot km_per_hour
```

Output: `100 knot is 185.2 kilometer / hour`

For pressure:

```
python unit_converter.py 50 psi kPa
```

Output: `50 psi is 344.737864655216 kilopascal`

```
python unit_converter.py 2 bar psi
```

Output: `2 bar is 29.0075477738527 psi`

This script handles errors for invalid units, incompatible conversions (e.g., meters to kilograms), and non-numeric inputs.

3.5 Summary

This advanced tutorial explored virtual environments for project isolation, installing libraries like Pint using `pip`, and constructing a versatile unit conversion script. The script supports conversions for length, temperature, speed (e.g., knots to km/h), and pressure (e.g., psi to kPa, bar to psi), making it useful for scientific and engineering applications. For further exploration, consult the official Pint documentation or experiment with additional units and quantities.

Part II

Making progress

Chapter 4

Python for Applied Mechanics: A Follow-Up Tutorial

This tutorial extends the foundational and advanced Python concepts from previous tutorials, tailoring them to applied mechanics in engineering. It focuses on using NumPy for numerical computations involving vectors and matrices (e.g., forces, stresses), Matplotlib for visualizing mechanics data (e.g., stress-strain curves, motion plots), and integrates these with Pint for unit-aware calculations. Examples are drawn from statics, dynamics, and mechanics of materials. Assume Pint, NumPy, and Matplotlib are installed in your virtual environment (as covered in the advanced tutorial).

4.1 Requirements

Build on the advanced tutorial: Activate your virtual environment and ensure the following libraries are installed via `pip install numpy matplotlib scipy pint`. SciPy is included for numerical methods like solving differential equations.

4.2 NumPy for Vectors and Matrices in Mechanics

NumPy is essential for handling arrays and matrices in applied mechanics, such as representing force vectors, displacement arrays, or stiffness matrices.

4.2.1 Vector Operations

Vectors are used for forces, velocities, and moments. NumPy arrays enable efficient operations like addition (resultant forces) and dot/cross products (work or torque).

```
import numpy as np

# Force vectors in 3D (e.g., in Newtons)
force1 = np.array([10, 20, 0])
force2 = np.array([5, -10, 15])

# Resultant force
resultant = force1 + force2
print(resultant)  # Output: [15 10 15]

# Dot product: Work done by force along a direction (e.g., force1 · force2)
work = np.dot(force1, force2)
print(work)  # Output: -150

# Cross product: Torque (moment) vector
torque = np.cross(force1, force2)
print(torque)  # Output: [ 300 -150 -200]
```

These operations are crucial in statics for equilibrium analysis or in dynamics for momentum calculations.

4.2.2 Matrix Operations

Matrices represent systems like truss structures or stress tensors. NumPy supports linear algebra for solving equations (e.g., Ax = b for displacements).

```
import numpy as np

# Stiffness matrix K and force vector F for a simple system
K = np.array([[2, -1], [-1, 2]])  # e.g., for a two-spring system
F = np.array([0, 5])  # Applied forces

# Solve for displacements: x = K^{-1} F
displacements = np.linalg.solve(K, F)
print(displacements)  # Output: [2.5 5. ]
```

This example solves for nodal displacements in a basic finite element or truss problem.

4.3 Matplotlib for Visualizing Mechanics Data

Matplotlib allows plotting of mechanics results, such as stress-strain curves in materials testing or position-time graphs in kinematics.

4.3.1 Basic Plotting

Plot force vs. displacement for a linear spring (Hooke's law: F = kx).

```
import numpy as np
import matplotlib.pyplot as plt

# Displacement array (m)
x = np.linspace(0, 0.1, 50)
k = 100  # Spring constant (N/m)
F = k * x  # Force (N)

plt.plot(x, F, label='Force vs. Displacement')
plt.xlabel('Displacement (m)')
plt.ylabel('Force (N)')
plt.title('Hooke\'s Law for a Spring')
plt.legend()
plt.grid(True)
plt.show()
```

This generates a line plot showing a linear relationship, useful for visualizing elastic behavior.

4.3.2 Advanced Plot: Stress-Strain Curve

For mechanics of materials, plot a typical stress-strain curve for steel.

```
import numpy as np
import matplotlib.pyplot as plt

# Strain (dimensionless)
strain = np.linspace(0, 0.015, 100)
# Stress (MPa): Linear elastic up to yield, then plastic
yield_strain = 0.002
E = 200000  # Young's modulus (MPa)
yield_stress = E * yield_strain
stress = np.where(strain <= yield_strain, E * strain, yield_stress + 10000 * (strain - yield_stra

plt.plot(strain, stress, color='blue')
plt.xlabel('Strain')
plt.ylabel('Stress (MPa)')
plt.title('Stress-Strain Curve for Steel')
```

```
plt.axvline(x=yield_strain, color='red', linestyle='--', label='Yield Point')
plt.legend()
plt.grid(True)
plt.show()
```

This plot illustrates elastic and plastic regions, with a dashed line at the yield point.

4.4 Numerical Methods with SciPy for Dynamics

SciPy provides tools for numerical integration and solving differential equations, key in dynamics for motion simulation.

4.4.1 Solving ODEs: Simple Harmonic Motion

Model a mass-spring system: $d^2x/dt^2 + {}^2x = 0$.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def shm(y, t, omega):
    # y = [position, velocity]
    return [y[1], -omega**2 * y[0]]

# Initial conditions: position=1 m, velocity=0 m/s
y0 = [1, 0]
t = np.linspace(0, 10, 100)  # Time array (s)
omega = 1  # Angular frequency (rad/s)

# Solve ODE
sol = odeint(shm, y0, t, args=(omega,))

# Plot position vs. time
plt.plot(t, sol[:, 0], label='Position (m)')
plt.xlabel('Time (s)')
plt.ylabel('Position (m)')
plt.title('Simple Harmonic Motion')
plt.legend()
plt.grid(True)
plt.show()
```

The plot shows oscillatory motion, decaying to equilibrium if damping were added.

4.5 Advanced Unit Handling with Pint for Mechanics

Extend Pint usage to mechanics-specific units, ensuring calculations account for dimensions.

4.5.1 Force and Torque Conversions

```
from pint import UnitRegistry

ureg = UnitRegistry()

# Force: Newtons to pounds-force
force_n = 100 * ureg.newton
force_lbf = force_n.to(ureg.lbf)
print(force_lbf)  # Output: 22.480894309999998 pound_force

# Torque: Newton-meters to foot-pounds
torque_nm = 50 * ureg.newton * ureg.meter
torque_ftlb = torque_nm.to(ureg.foot * ureg.pound_force)
print(torque_ftlb)  # Output: 36.878681655 foot * pound_force
```

These conversions are vital in international engineering projects or when using mixed unit systems (SI vs. Imperial).

4.6 Integration Example: Projectile Motion

Combine NumPy, SciPy, and Matplotlib for a dynamics problem: Simulate projectile trajectory under gravity.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

def projectile(y, t, g, theta, v0):
    # y = [x, vx, y, vy]
    return [y[1], 0, y[3], -g]

# Initial conditions: Launch at angle theta=45 deg, v0=20 m/s
theta = np.radians(45)
v0 = 20
y0 = [0, v0 * np.cos(theta), 0, v0 * np.sin(theta)]
t = np.linspace(0, 4, 100)  # Time (s)
g = 9.81  # Gravity (m/s²)
```

```
# Solve ODE
sol = odeint(projectile, y0, t, args=(g, theta, v0))

# Plot trajectory (x vs. y)
plt.plot(sol[:, 0], sol[:, 2])
plt.xlabel('Horizontal Distance (m)')
plt.ylabel('Vertical Distance (m)')
plt.title('Projectile Trajectory')
plt.grid(True)
plt.show()
```

This simulates and plots the parabolic path, ignoring air resistance.

4.7 Summary

This tutorial applies Python tools to applied mechanics, using NumPy for computations, Matplotlib for visualization, SciPy for numerical methods, and Pint for unit handling. These techniques support analysis in statics, dynamics, and materials. Experiment with parameters or integrate with real data for class projects. For advanced topics, explore SymPy for symbolic mechanics or finite element libraries like FEniCS.

Part III

Going forward

Chapter 5

Review

We have used several books by Ahrens (2022), Russell et al. (2021), Bolton (2021), Polya & Conway (2014), Bird & Ross (2020) and Bird (2021). These sources have helped you understand complex concepts.

Chapter 2:

- Purpose of SI Units: Provide a consistent framework for scientific and technical measurements.
- Advantages of SI Units: Facilitate clear communication and data comparison across various fields and countries.
- Fundamental Units of SI: Meter, kilogram, second, ampere, kelvin, mole, and candela.
- Method Name: Unity fraction method.
- Purpose: Converting one unit of measurement into another.
- Methodology: Multiplying by fractions equal to one, where the numerator and denominator represent the same quantity in different units.

References

Ahrens, S. (2022). *How to take smart notes: One simple technique to boost writing, learning and thinking* (2nd ed. edition). Sönke Ahrens.

Bird, J. O. (2021). *Bird's engineering mathematics* (Ninth edition). Routledge.

Bird, J. O., & Ross, C. T. F. (2020). *Mechanical engineering principles* (Fourth edition). Routledge.

Bolton, W. (2021). *Engineering science* (Seventh edition). Routledge.

Polya, G., & Conway, J. H. (2014). *How to solve it: A new aspect of mathematical method* (With a Foreword by John H. Con ed. edition). Princeton University Press.

Russell, P. A., Jackson, L., & Embleton, W. (2021). *Applied mechanics for marine engineers* (7th edition). Reeds.

Index