Python Modules & File Operations

Building with Libraries and Managing Data Files

CS 101 - Fall 2025

Welcome to the World of Python Libraries!

> Why Use Modules?
>
> **Modules are like LEGO blocks for programming!** Instead of building everything from scratch, we can use pre-built pieces that experts have already created.
> **Real-World Analogy**: * Like using a **calculator app** instead of building your own calculator * Like using **GPS navigation** instead of memorizing every road * Like using a **recipe book** instead of inventing cooking from scratch!

What You Get

- **Pre-written functions** - tested and reliable
- **Specialized tools** - for math, dates, files, web
- **Time savings** - focus on your logic, not reinventing wheels
- **Professional quality** - used by millions of developers

Today's Journey

1. **Part 1**: Using built-in modules (`calendar`, `math`, `random`)
2. **File Basics**: Reading and writing `.txt` and `.csv` files

3. **Part 2**: Creating your own custom modules

Magic Question: *"What amazing things can I build when I have the right tools?"*

Part 1: Using Python's Built-in Modules

The `import` Statement - Your Magic Key!

> **How to Access Python's Treasure Chest**
>
> The `import` statement is like having a **key to a massive toolshed** filled with specialized tools!

Basic Import Syntax

```
# Method 1: Import entire module
import math
result = math.sqrt(16)  # Use math.function_name

# Method 2: Import specific functions
from math import sqrt, pi
result = sqrt(16)       # Use function directly
area = pi * radius**2

# Method 3: Import with nickname
import calendar as cal
print(cal.month_name[3])  # March
```

```python
# Method 4: Import everything (be careful!)
from math import *
result = sqrt(16)       # All functions available
```

Best Practices

```python
#  GOOD: Clear what you're using
import math
import calendar as cal
from random import randint

#  GOOD: Import specific items
from math import sqrt, pi, sin, cos

#  BE CAREFUL: Importing everything
from math import *  # Can cause name conflicts

#  TIP: Check what's available
import math
print(dir(math))  # Shows all available functions
```

Pro Tip: Import statements go at the top of your file!

Exploring the `calendar` Module

> Time Travel with Python!
>
> The `calendar` module lets you work with dates, find specific days, and even create calendars for any year!

```python
import calendar as cal

# Display a full month calendar
print(cal.month(2025, 11))  # November 2025

# Get information about dates
print(f"Today is: {cal.day_name[cal.weekday(2025, 11, 4)]}")  # Monday

# Find which day of week a date falls on (0=Monday, 6=Sunday)
day_number = cal.weekday(2025, 12, 25)  # Christmas 2025
```

```
print(f"Christmas 2025 falls on: {cal.day_name[day_number]}")

# Get month as nested lists (perfect for programming!)
month_calendar = cal.monthcalendar(2025, 11)
print("November 2025 as lists:")
for week in month_calendar:
    print(week)  # Each week is a list [Mon, Tue, Wed, Thu, Fri, Sat, Sun]
```

> **!** Important
>
> **Interactive Demo**: Let's find your birthday! What day of the week were you born on?

Calendar Magic: Finding Special Days!

Finding Thanksgiving

```
import calendar as cal

def find_thanksgiving(year):
    """Find Thanksgiving (4th Thursday of November)"""
    # Get November calendar as nested lists
    november = cal.monthcalendar(year, 11)

    # Find 4th Thursday
    # Check if first week has Thursday
    if november[0][cal.THURSDAY] != 0:
        # First week has Thursday, so 4th is week 3 (0-indexed)
        thanksgiving = november[3][cal.THURSDAY]
    else:
        # First week has no Thursday, so 4th is week 4
        thanksgiving = november[4][cal.THURSDAY]

    return thanksgiving

# Test it out!
for year in [2024, 2025, 2026]:
    day = find_thanksgiving(year)
    print(f"Thanksgiving {year}: November {day}")
```

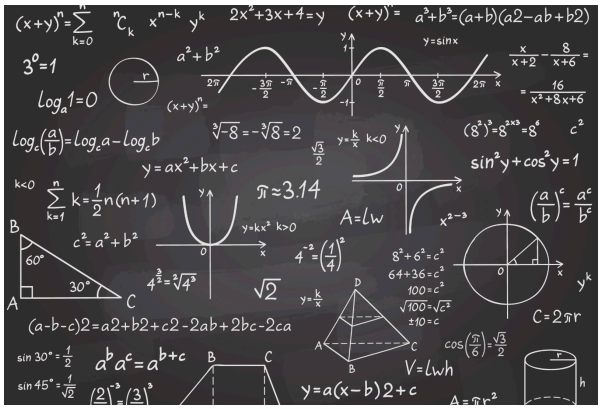
Weekend Counter

```python
def count_weekends_in_month(year, month):
    """Count how many Saturdays and Sundays in a month"""
    month_cal = cal.monthcalendar(year, month)
    saturdays = sundays = 0

    for week in month_cal:
        if week[cal.SATURDAY] != 0:  # 0 means no day
            saturdays += 1
        if week[cal.SUNDAY] != 0:
            sundays += 1

    return saturdays, sundays

# How many weekend days in December 2025?
sat, sun = count_weekends_in_month(2025, 12)
print(f"December 2025: {sat} Saturdays, {sun} Sundays")
print(f"Total weekend days: {sat + sun}")
```

Your Turn: Modify this to count school days during the month (Mon-Fri, no weekends days)!

The Math Import

> **!** Important
>
> **Q**: How do you charge-up python for extra functionality when doing math and other things?
> **A**: You would use `import math` in your code to add number crunching power.



The `math` Module - Your Mathematical Superpower!

Essential Math Functions

```
import math

# Basic operations
print(f"Square root of 64: {math.sqrt(64)}")
print(f"2 to the power of 8: {math.pow(2, 8)}")
print(f"Absolute value of -42: {math.fabs(-42)}")

# Rounding functions
x = 3.7
print(f"Floor of {x}: {math.floor(x)}")    # 3
print(f"Ceiling of {x}: {math.ceil(x)}")   # 4

# Advanced functions
print(f"Factorial of 5: {math.factorial(5)}")  # 120
print(f"Log base 10 of 100: {math.log10(100)}")  # 2.0

# Constants
print(f"Pi: {math.pi}")
print(f"e (Euler's number): {math.e}")
```

Trigonometry & Geometry

```
import math

# Trigonometric functions (angles in radians)
```

```python
angle_degrees = 45
angle_radians = math.radians(angle_degrees)

print(f"sin(45°): {math.sin(angle_radians):.3f}")
print(f"cos(45°): {math.cos(angle_radians):.3f}")
print(f"tan(45°): {math.tan(angle_radians):.3f}")

# Convert back to degrees
result = math.degrees(angle_radians)
print(f"Back to degrees: {result}")

# Distance between points (Pythagorean theorem)
def distance(x1, y1, x2, y2):
    return math.sqrt((x2-x1)**2 + (y2-y1)**2)

# Distance from (0,0) to (3,4)
print(f"Distance: {distance(0, 0, 3, 4)}")  # 5.0
```

! Important

Rather than having to write code to approximate things, have the `math` library do it for you!
Q: Why use a library in Python, anyway?
A: You would use a library to make your code more readable (and *trustworthy*) to a maintainer. Having a published library fit to handle your heavy lifting help to ensure smooth operation which will facilitate maintenance.

Interactive Math Demo: Circle Calculator!

i Build a Circle Calculator Together!

Let's use the `math` module to create useful tools!

```python
import math

def circle_calculator(radius):
    """Calculate everything about a circle given its radius"""
    # Area =  × r²
    area = math.pi * radius**2
```

```
    # Circumference = 2 ×   × r
    circumference = 2 * math.pi * radius

    # Diameter = 2 × r
    diameter = 2 * radius

    return {
        'radius': radius,
        'diameter': diameter,
        'area': round(area, 2),
        'circumference': round(circumference, 2)
    }

# Test with different radii
radii = [1, 5, 10, 25]
print("  Circle Calculator Results:")
print("-" * 50)
print(f"{'Radius':<8} {'Diameter':<10} {'Area':<12} {'Circumference'}")
print("-" * 50)

for r in radii:
    info = circle_calculator(r)
    print(f"{info['radius']:<8} {info['diameter']:<10} {info['area']:<12} {info['circumferenc
```

Your Turn: Can you add a function to calculate the area of a sphere? (Hint: 4 r^2)!?

The `random` Module - Adding Unpredictability!

Random Numbers & Choices

```
import random

# Random integers
dice_roll = random.randint(1, 6)
print(f"Dice roll: {dice_roll}")

# Random floating point numbers
random_percent = random.random()  # 0.0 to 1.0
print(f"Random percentage: {random_percent:.2%}")

# Random from a specific range
```

```
temperature = random.uniform(20.0, 35.0)  # 20°C to 35°C
print(f"Random temperature: {temperature:.1f}°C")

# Random choices from a list
colors = ['red', 'blue', 'green', 'yellow', 'purple']
chosen_color = random.choice(colors)
print(f"Random color: {chosen_color}")

# Multiple random choices (with replacement)
team = random.choices(colors, k=3)
print(f"Random team colors: {team}")
```

Shuffling & Sampling

```
import random

# Shuffle a list (modifies original)
deck = ['A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K']
print(f"Original deck: {deck[:5]}...")  # Show first 5
random.shuffle(deck)
print(f"Shuffled deck: {deck[:5]}...")

# Random sample (without replacement)
students = ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve', 'Frank']
volunteers = random.sample(students, 3)  # Pick 3 students
print(f"Today's volunteers: {volunteers}")

# Set seed for reproducible results
random.seed(42)  # Always gives same "random" sequence
print(f"Seeded random: {random.randint(1, 100)}")
print(f"Next seeded random: {random.randint(1, 100)}")
```

! Important

Interestingly, there is nothing really random about this generator – all numbers come from a look-up system. If you want truly random numbers, then you will have to use another system.
Q: Would this random number generator library serve in security applications?
A: The Random library is used for low security applications.

Let's Build: A Password Generator!

> Practical Application: Security Tool
>
> Combine `random` and `string` modules to create a secure password generator!

```
import random
import string

def generate_password(length=12, include_symbols=True):
    """Generate a secure random password"""

    # Build character set
    characters = string.ascii_letters + string.digits  # a-z, A-Z, 0-9

    if include_symbols:
        # Add safe symbols (avoid confusing ones)
        safe_symbols = "!@#$%^&*+-="
        characters += safe_symbols

    # Generate password
    password = ''.join(random.choices(characters, k=length))

    return password

def password_strength_checker(password):
    """Analyze password strength"""
    score = 0
    feedback = []

    if len(password) >= 8:
        score += 1
    else:
        feedback.append("Use at least 8 characters")

    if any(c.isupper() for c in password):
        score += 1
    else:
        feedback.append("Add uppercase letters")

    if any(c.islower() for c in password):
        score += 1
```

```
    else:
        feedback.append("Add lowercase letters")

    if any(c.isdigit() for c in password):
        score += 1
    else:
        feedback.append("Add numbers")

    if any(c in "!@#$%^&*+-=" for c in password):
        score += 1
    else:
        feedback.append("Add special characters")

    strength_levels = ["Very Weak", "Weak", "Fair", "Good", "Strong"]
    strength = strength_levels[min(score, 4)]

    return strength, feedback

# Generate and test passwords
print("  Password Generator Demo")
print("-" * 40)

for length in [8, 12, 16]:
    password = generate_password(length)
    strength, tips = password_strength_checker(password)
    print(f"Length {length}: {password}")
    print(f"Strength: {strength}")
    if tips:
        print(f"Tips: {', '.join(tips)}")
    print()
```

> **!** Important
>
> **Q**: Why is is important to have such a random looking password?
> **A**: Finding a password is hard word! We remember that this operation is a `O(n)` operation where each of `n` possible passwords must be checked individually.

Your Turn: Module Exploration!

> **!** Hands-On Activity: Discover More Modules!
>
> Try these mini-challenges to explore Python's built-in modules:

Challenge 1: `datetime` Module

```
import datetime

# Find your age in days
birth_date = datetime.date(2008, 5, 15)  # Your birthday
today = datetime.date.today()
age_days = (today - birth_date).days

print(f"You are {age_days} days old!")

# What day will it be in 100 days?
future = today + datetime.timedelta(days=100)
print(f"In 100 days it will be: {future}")

# Your challenge:
# Find what day of the week you were born!
```

Challenge 2: `statistics` Module

```
import statistics

# Analyze test scores
scores = [88, 92, 76, 95, 82, 90, 87, 93, 79, 91]

print(f"Mean: {statistics.mean(scores)}")
print(f"Median: {statistics.median(scores)}")
print(f"Mode: {statistics.mode(scores)}")  # Most common
print(f"Std Dev: {statistics.stdev(scores):.2f}")

# Your challenge:
# Add functions to find the grade distribution!
# (How many A's, B's, C's, etc.)
```

File Operations: Working with Data Files

Understanding File Types & When to Use Them

> **File Formats for Data Storage**
>
> Before we dive into code, let's understand **what** files we'll work with and **why**!

Text Files (`.txt`)

- **Best for**: Simple text, notes, logs, configuration
- **Human-readable**: Yes, open in any text editor
- **Structure**: Free-form text, paragraphs
- **Examples**:
  - Student essays
  - Game logs
  - Simple data lists
  - Configuration files

```
Hello World!
This is a simple text file.
Each line can contain any text.
Numbers: 123, 456, 789
```

CSV Files (`.csv`)

- **Best for**: Structured data, spreadsheet-like information
- **Human-readable**: Yes, opens in Excel/Google Sheets
- **Structure**: Rows and columns, separated by commas
- **Examples**:
  - Student grades
  - Inventory lists
  - Survey responses
  - Scientific data

```
Name,Age,Grade,Subject
Alice,16,95,Math
Bob,17,87,Science
Charlie,16,92,Math
```

Reading Text Files - Your First File Adventure!

> **Step-by-Step File Reading**
>
> Reading files is like opening a book - you need to open it, read the contents, then close it when done!

```
# Method 1: Basic file reading (manual close)
file = open('student_essay.txt', 'r')  # 'r' = read mode
content = file.read()  # Read entire file
print(content)
file.close()  # Always close the file!

# Method 2: Read line by line
file = open('student_essay.txt', 'r')
for line_number, line in enumerate(file, 1):
    print(f"Line {line_number}: {line.strip()}")  # strip() removes \n
file.close()

# Method 3: Best practice - using 'with' (auto-closes!)
with open('student_essay.txt', 'r') as file:
    content = file.read()
    print(content)
# File automatically closes when indented block ends!

# Method 4: Read all lines into a list
with open('student_essay.txt', 'r') as file:
    lines = file.readlines()  # Returns list of lines
    print(f"File has {len(lines)} lines")
    for i, line in enumerate(lines):
        print(f"{i+1}: {line.strip()}")
```

Writing Text Files - Saving Your Data!

Creating New Files

```
# Write a new file
with open('my_story.txt', 'w') as file:  # 'w' = write mode
    file.write("Once upon a time...\n")
    file.write("There was a student learning Python.\n")
    file.write("They discovered the magic of files!")
```

```
# Writing multiple lines at once
lines = [
    "Chapter 1: The Beginning\n",
    "It was a dark and stormy night.\n",
    "Perfect weather for coding!\n"
]

with open('novel.txt', 'w') as file:
    file.writelines(lines)  # Write list of lines

print("Files created successfully!")
```

Appending to Files

```
# Add to existing file (don't overwrite!)
with open('my_story.txt', 'a') as file:  # 'a' = append mode
    file.write("\n\nChapter 2: The Adventure Continues\n")
    file.write("Our hero learned about file modes:\n")
    file.write("- 'r' for reading\n")
    file.write("- 'w' for writing (overwrites!)\n")
    file.write("- 'a' for appending (adds to end)\n")

# Safe writing with error handling
filename = 'log.txt'
try:
    with open(filename, 'a') as file:
        file.write(f"Entry at {datetime.datetime.now()}\n")
        file.write("Everything working great!\n")
    print("Log entry added!")
except FileNotFoundError:
    print(f"Could not find {filename}")
```

Working with CSV Files - Structured Data!

> **!** CSV = Comma Separated Values
>
> CSV files are perfect for data that fits in rows and columns, like spreadsheets!

```
import csv

# Reading CSV files - Method 1: Basic reading
with open('students.csv', 'r') as file:
    csv_reader = csv.reader(file)
    headers = next(csv_reader)  # Get the first row (headers)
    print(f"Headers: {headers}")

    for row_number, row in enumerate(csv_reader, 1):
        print(f"Student {row_number}: {row}")

# Reading CSV files - Method 2: Dictionary reader (easier!)
with open('students.csv', 'r') as file:
    csv_reader = csv.DictReader(file)  # Treats first row as headers

    for student in csv_reader:
        name = student['Name']
        grade = student['Grade']
        subject = student['Subject']
        print(f"{name} got {grade}% in {subject}")

# Writing CSV files
student_data = [
    ['Name', 'Age', 'Grade', 'Subject'],
    ['Alice', 16, 95, 'Math'],
    ['Bob', 17, 87, 'Science'],
    ['Charlie', 16, 92, 'Math'],
    ['Diana', 16, 89, 'English']
]

with open('new_students.csv', 'w', newline='') as file:
    csv_writer = csv.writer(file)
    csv_writer.writerows(student_data)  # Write all rows at once

print("CSV file created!")
```

Interactive File Demo: Grade Book Manager!

> **Let's Build Something Useful!**
>
> Create a grade book that can save and load student data!

```
import csv
from datetime import datetime

class GradeBook:
    def __init__(self, filename='gradebook.csv'):
        self.filename = filename
        self.students = []
        self.load_grades()

    def load_grades(self):
        """Load existing grades from CSV file"""
        try:
            with open(self.filename, 'r') as file:
                csv_reader = csv.DictReader(file)
                self.students = list(csv_reader)
                print(f"Loaded {len(self.students)} student records")
        except FileNotFoundError:
            print(f"No existing gradebook found. Starting fresh!")
            self.students = []

    def add_student(self, name, subject, grade):
        """Add a new student grade"""
        student = {
            'Name': name,
            'Subject': subject,
            'Grade': str(grade),
            'Date': datetime.now().strftime('%Y-%m-%d')
        }
        self.students.append(student)
        print(f"Added: {name} - {subject}: {grade}%")

    def save_grades(self):
        """Save all grades to CSV file"""
        if not self.students:
            print("No grades to save!")
            return
```

```
        with open(self.filename, 'w', newline='') as file:
            fieldnames = ['Name', 'Subject', 'Grade', 'Date']
            csv_writer = csv.DictWriter(file, fieldnames=fieldnames)
            csv_writer.writeheader()
            csv_writer.writerows(self.students)
        print(f"Saved {len(self.students)} grades to {self.filename}")

    def get_student_average(self, name):
        """Calculate average grade for a student"""
        student_grades = [int(s['Grade']) for s in self.students if s['Name'] == name]
        if student_grades:
            return sum(student_grades) / len(student_grades)
        return None

    def display_gradebook(self):
        """Display all grades in a nice format"""
        if not self.students:
            print("Gradebook is empty!")
            return

        print("\n GRADEBOOK REPORT")
        print("-" * 50)
        print(f"{'Name':<12} {'Subject':<10} {'Grade':<6} {'Date'}")
        print("-" * 50)

        for student in self.students:
            print(f"{student['Name']:<12} {student['Subject']:<10} {student['Grade']+'%':<6}

# Demo the gradebook
gb = GradeBook()

# Add some sample grades
gb.add_student("Alice", "Math", 95)
gb.add_student("Bob", "Science", 87)
gb.add_student("Alice", "Science", 92)
gb.add_student("Charlie", "Math", 88)

# Display and save
gb.display_gradebook()
gb.save_grades()

# Show average for Alice
```

```python
alice_avg = gb.get_student_average("Alice")
if alice_avg:
    print(f"\nAlice's average: {alice_avg:.1f}%")
```

Part 2: Creating Your Own Modules

What Makes a Module? Building Your First Module!

> **i** Modules Are Just Python Files!
>
> Any `.py` file can become a module that other programs can import and use!

Create `math_helpers.py`

```python
"""
Math Helper Functions
A collection of useful mathematical utilities
"""

import math

def area_circle(radius):
    """Calculate the area of a circle"""
    return math.pi * radius ** 2

def area_rectangle(length, width):
    """Calculate the area of a rectangle"""
    return length * width

def area_triangle(base, height):
    """Calculate the area of a triangle"""
    return 0.5 * base * height

def distance_2d(x1, y1, x2, y2):
    """Calculate distance between two points"""
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

def is_perfect_square(n):
```

```
    """Check if a number is a perfect square"""
    if n < 0:
        return False
    root = int(math.sqrt(n))
    return root * root == n

# Module-level variables (constants)
GOLDEN_RATIO = (1 + math.sqrt(5)) / 2
PI_ROUNDED = round(math.pi, 4)

# Demo function that runs when module is imported
def demo():
    """Demonstrate all functions in this module"""
    print("  Math Helpers Demo")
    print("-" * 30)
    print(f"Circle area (r=5): {area_circle(5):.2f}")
    print(f"Rectangle area (4x6): {area_rectangle(4, 6)}")
    print(f"Triangle area (b=8, h=5): {area_triangle(8, 5)}")
    print(f"Distance (0,0) to (3,4): {distance_2d(0, 0, 3, 4)}")
    print(f"Is 16 perfect square? {is_perfect_square(16)}")
    print(f"Golden ratio: {GOLDEN_RATIO:.6f}")
```

Using Your Module

```
# Import your custom module
import math_helpers

# Use the functions
radius = 10
area = math_helpers.area_circle(radius)
print(f"Circle with radius {radius} has area: {area:.2f}")

# Import specific functions
from math_helpers import distance_2d, is_perfect_square

# Use imported functions directly
dist = distance_2d(1, 1, 4, 5)
print(f"Distance: {dist:.2f}")

# Check perfect squares
numbers = [16, 17, 25, 30, 36]
for num in numbers:
```

```
    result = is_perfect_square(num)
    print(f"{num} is perfect square: {result}")

# Access module constants
from math_helpers import GOLDEN_RATIO, PI_ROUNDED
print(f"Golden ratio: {GOLDEN_RATIO}")
print(f"Pi (rounded): {PI_ROUNDED}")

# Run the demo
math_helpers.demo()
```

Advanced Module Features: Making It Professional!

> **Professional Module Structure**
>
> Let's add advanced features that make modules robust and user-friendly!

```
"""
student_toolkit.py
A comprehensive toolkit for student-related operations
"""

import csv
import json
from datetime import datetime
import statistics

class StudentManager:
    """Class to manage student data and operations"""

    def __init__(self):
        self.students = []

    def add_student(self, name, age, grade_level):
        """Add a new student"""
        student = {
            'id': len(self.students) + 1,
            'name': name,
            'age': age,
            'grade_level': grade_level,
```

```
            'grades': [],
            'created': datetime.now().isoformat()
        }
        self.students.append(student)
        return student['id']

    def add_grade(self, student_id, subject, grade, date=None):
        """Add a grade for a student"""
        if date is None:
            date = datetime.now().strftime('%Y-%m-%d')

        for student in self.students:
            if student['id'] == student_id:
                grade_entry = {
                    'subject': subject,
                    'grade': grade,
                    'date': date
                }
                student['grades'].append(grade_entry)
                return True
        return False

    def get_student_average(self, student_id):
        """Calculate student's average grade"""
        for student in self.students:
            if student['id'] == student_id:
                grades = [g['grade'] for g in student['grades']]
                return statistics.mean(grades) if grades else 0
        return None

# Utility functions
def calculate_gpa(grades):
    """Convert grades to GPA (4.0 scale)"""
    if not grades:
        return 0.0

    def grade_to_points(grade):
        if grade >= 97: return 4.0
        elif grade >= 93: return 3.7
        elif grade >= 90: return 3.3
        elif grade >= 87: return 3.0
        elif grade >= 83: return 2.7
```

```
        elif grade >= 80: return 2.3
        elif grade >= 77: return 2.0
        elif grade >= 73: return 1.7
        elif grade >= 70: return 1.3
        elif grade >= 67: return 1.0
        else: return 0.0

    points = [grade_to_points(grade) for grade in grades]
    return sum(points) / len(points)

def save_to_csv(students, filename):
    """Save student data to CSV file"""
    with open(filename, 'w', newline='') as file:
        writer = csv.writer(file)
        writer.writerow(['ID', 'Name', 'Age', 'Grade Level', 'Average'])

        for student in students:
            avg_grade = statistics.mean([g['grade'] for g in student['grades']]) if student[
            writer.writerow([
                student['id'],
                student['name'],
                student['age'],
                student['grade_level'],
                round(avg_grade, 2)
            ])

def load_from_json(filename):
    """Load student data from JSON file"""
    try:
        with open(filename, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return []

def save_to_json(students, filename):
    """Save student data to JSON file"""
    with open(filename, 'w') as file:
        json.dump(students, file, indent=2)

# Module constants
GRADE_SCALE = {
    'A+': (97, 100),
```

```
    'A': (93, 96),
    'A-': (90, 92),
    'B+': (87, 89),
    'B': (83, 86),
    'B-': (80, 82),
    'C+': (77, 79),
    'C': (73, 76),
    'C-': (70, 72),
    'D': (60, 69),
    'F': (0, 59)
}

def get_letter_grade(numeric_grade):
    """Convert numeric grade to letter grade"""
    for letter, (min_grade, max_grade) in GRADE_SCALE.items():
        if min_grade <= numeric_grade <= max_grade:
            return letter
    return 'F'

# Special behavior when module is run directly
if __name__ == "__main__":
    print("  Student Toolkit Demo")
    print("-" * 30)

    # Create a student manager
    sm = StudentManager()

    # Add some students
    alice_id = sm.add_student("Alice Johnson", 16, 11)
    bob_id = sm.add_student("Bob Smith", 17, 12)

    # Add grades
    sm.add_grade(alice_id, "Math", 95)
    sm.add_grade(alice_id, "Science", 92)
    sm.add_grade(alice_id, "English", 89)

    sm.add_grade(bob_id, "Math", 87)
    sm.add_grade(bob_id, "Science", 91)

    # Show results
    for student in sm.students:
        avg = sm.get_student_average(student['id'])
```

```
        letter = get_letter_grade(avg)
        gpa = calculate_gpa([g['grade'] for g in student['grades']])

        print(f"\n{student['name']} (ID: {student['id']})")
        print(f"  Average: {avg:.1f}% ({letter})")
        print(f"  GPA: {gpa:.2f}")
        print(f"  Grades: {[g['grade'] for g in student['grades']]}")

    # Save data
    save_to_csv(sm.students, 'student_report.csv')
    save_to_json(sm.students, 'student_data.json')
    print("\n Data saved to CSV and JSON files!")
```

Using Your Custom Module: Example 1 of Real-World Applications!

Import and Use

```
# Using your student toolkit
from student_toolkit import (
    StudentManager,
    calculate_gpa,
    get_letter_grade,
    GRADE_SCALE
)

# Create a classroom management system
classroom = StudentManager()

# Add students from your class
students_data = [
    ("Emily Chen", 16, 11),
    ("Marcus Rodriguez", 17, 11),
    ("Sarah Kim", 16, 11),
    ("Jordan Williams", 17, 11)
]

student_ids = []
for name, age, grade_level in students_data:
    student_id = classroom.add_student(name, age, grade_level)
    student_ids.append(student_id)
    print(f"Added {name} with ID: {student_id}")
```

```
# Add grades for each student
subjects = ["Math", "Science", "English", "History"]
import random

print("\n Adding random grades...")
for student_id in student_ids:
    for subject in subjects:
        # Generate realistic grades (70-100)
        grade = random.randint(70, 100)
        classroom.add_grade(student_id, subject, grade)

# Generate class report
print("\n CLASS REPORT")
print("=" * 50)
for student in classroom.students:
    avg = classroom.get_student_average(student['id'])
    letter = get_letter_grade(avg)
    grades_list = [g['grade'] for g in student['grades']]
    gpa = calculate_gpa(grades_list)

    print(f"{student['name']:<15} | Avg: {avg:5.1f}% | Grade: {letter} | GPA: {gpa:.2f}")
```

> **!** Important
>
> There are many different ways to write *Real-World* applications, but this is a good way to begin a project.

Using Your Custom Module: Example 2 of Real-World Applications!

Module Organization

```
# Create a package structure
"""
my_school_package/
   __init__.py          # Makes it a package
   students.py          # Student management
   grades.py           # Grade calculations
   reports.py          # Report generation
   utils.py            # Utility functions
"""
```

```python
# __init__.py file content:
"""
School Management Package
A complete toolkit for managing students and grades
"""

from .students import StudentManager
from .grades import calculate_gpa, get_letter_grade
from .reports import generate_report
from .utils import save_data, load_data

__version__ = "1.0.0"
__author__ = "Your Name"

# Package-level constants
DEFAULT_SUBJECTS = ["Math", "Science", "English", "History"]
PASSING_GRADE = 70

def quick_demo():
    """Quick demonstration of package features"""
    print(f"School Package v{__version__}")
    print("Available modules:")
    print("- StudentManager: Manage student data")
    print("- Grade calculators: GPA and letter grades")
    print("- Report generators: Create summaries")
    print("- Data utilities: Save/load functionality")
```

Module Best Practices & Documentation!

> **!** Writing Professional Modules
>
> Examples of a good way to write code. Follow these practices to create modules that others (and future you!) can easily use!

```python
"""
geometry_tools.py
Advanced Geometric Calculations and Shape Analysis

This module provides classes and functions for working with 2D and 3D geometric shapes.
Perfect for math classes, game development, and engineering applications.
```

```
Author: "Your Name"
Date: November 2025
Version: 1.0

Example usage:
    from geometry_tools import Circle, Rectangle, calculate_polygon_area

    circle = Circle(5)
    print(f"Circle area: {circle.area()}")

    rect = Rectangle(4, 6)
    print(f"Rectangle perimeter: {rect.perimeter()}")
"""

import math
from typing import List, Tuple

class Shape:
    """Base class for all geometric shapes"""

    def __init__(self, name: str):
        self.name = name

    def area(self) -> float:
        """Calculate the area of the shape"""
        raise NotImplementedError("Subclasses must implement area()")

    def perimeter(self) -> float:
        """Calculate the perimeter of the shape"""
        raise NotImplementedError("Subclasses must implement perimeter()")

    def __str__(self) -> str:
        """String representation of the shape"""
        return f"{self.name} - Area: {self.area():.2f}, Perimeter: {self.perimeter():.2f}"

class Circle(Shape):
    """A circle defined by its radius"""

    def __init__(self, radius: float):
        super().__init__("Circle")
        if radius <= 0:
            raise ValueError("Radius must be positive")
```

```
        self.radius = radius

    def area(self) -> float:
        """Calculate circle area using   × r²"""
        return math.pi * self.radius ** 2

    def perimeter(self) -> float:
        """Calculate circle circumference using 2 ×   × r"""
        return 2 * math.pi * self.radius

    def diameter(self) -> float:
        """Get the circle's diameter"""
        return 2 * self.radius

class Rectangle(Shape):
    """A rectangle defined by length and width"""

    def __init__(self, length: float, width: float):
        super().__init__("Rectangle")
        if length <= 0 or width <= 0:
            raise ValueError("Length and width must be positive")
        self.length = length
        self.width = width

    def area(self) -> float:
        """Calculate rectangle area using length × width"""
        return self.length * self.width

    def perimeter(self) -> float:
        """Calculate rectangle perimeter using 2 × (length + width)"""
        return 2 * (self.length + self.width)

    def is_square(self) -> bool:
        """Check if the rectangle is actually a square"""
        return abs(self.length - self.width) < 1e-10  # Account for floating point

def calculate_polygon_area(vertices: List[Tuple[float, float]]) -> float:
    """
    Calculate area of a polygon using the shoelace formula

    Args:
        vertices: List of (x, y) coordinate tuples defining the polygon
```

```
    Returns:
        The area of the polygon

    Raises:
        ValueError: If fewer than 3 vertices provided

    Example:
        triangle = [(0, 0), (4, 0), (2, 3)]
        area = calculate_polygon_area(triangle)  # Returns 6.0
    """
    if len(vertices) < 3:
        raise ValueError("A polygon must have at least 3 vertices")

    n = len(vertices)
    area = 0.0

    for i in range(n):
        j = (i + 1) % n
        area += vertices[i][0] * vertices[j][1]
        area -= vertices[j][0] * vertices[i][1]

    return abs(area) / 2.0

# Module constants with clear documentation
PI = math.pi  # More readable alias for
DEGREES_TO_RADIANS = math.pi / 180  # Conversion factor
RADIANS_TO_DEGREES = 180 / math.pi  # Conversion factor

# Utility functions
def convert_to_radians(degrees: float) -> float:
    """Convert degrees to radians"""
    return degrees * DEGREES_TO_RADIANS

def convert_to_degrees(radians: float) -> float:
    """Convert radians to degrees"""
    return radians * RADIANS_TO_DEGREES

# Demo and testing code
if __name__ == "__main__":
    print("  Geometry Tools Demo")
    print("-" * 40)
```

```
    # Test Circle
    circle = Circle(5)
    print(circle)
    print(f"Diameter: {circle.diameter():.2f}")

    # Test Rectangle
    rect = Rectangle(4, 6)
    print(rect)
    print(f"Is square? {rect.is_square()}")

    # Test Square
    square = Rectangle(5, 5)
    print(square)
    print(f"Is square? {square.is_square()}")

    # Test Polygon
    triangle = [(0, 0), (4, 0), (2, 3)]
    triangle_area = calculate_polygon_area(triangle)
    print(f"Triangle area: {triangle_area:.2f}")

    # Test conversions
    angle_deg = 90
    angle_rad = convert_to_radians(angle_deg)
    print(f"{angle_deg}° = {angle_rad:.4f} radians")
```

Hands-On Challenge: Build Your Dream Module!

> **Final Project: Create Your Own Utility Module!**
>
> Choose one of these ideas or create your own unique module:

Option 1: Game Utilities

```
"""
game_utils.py - Gaming Helper Functions
"""

import random

def roll_dice(sides=6, count=1):
    """Roll dice and return results"""
```

```
    return [random.randint(1, sides) for _ in range(count)]

def calculate_damage(base, weapon_bonus, critical=False):
    """Calculate game damage with bonuses"""
    damage = base + weapon_bonus
    if critical:
        damage *= 2
    return damage

class Player:
    def __init__(self, name, health=100):
        self.name = name
        self.health = health
        self.max_health = health
        self.inventory = []

    def take_damage(self, damage):
        self.health = max(0, self.health - damage)
        return self.health <= 0  # Returns True if player died

# Your additions:
# - Magic spells system
# - Inventory management
# - Experience points calculation
```

Option 2: Text Analysis

```
"""
text_analyzer.py - Analyze written text
"""

import string

def word_count(text):
    """Count words in text"""
    return len(text.split())

def character_frequency(text):
    """Count frequency of each character"""
    frequency = {}
    for char in text.lower():
        if char.isalnum():
```

```
            frequency[char] = frequency.get(char, 0) + 1
    return frequency

def reading_level(text):
    """Calculate approximate reading level"""
    words = len(text.split())
    sentences = text.count('.') + text.count('!') + text.count('?')
    if sentences == 0:
        return 0
    avg_words_per_sentence = words / sentences
    # Simple formula for reading level
    return min(12, max(1, int(avg_words_per_sentence / 2)))

# Your additions:
# - Sentiment analysis (positive/negative)
# - Common word finder
# - Text summarizer
```

Solution :: Week and Weekend Days

Solution: How many days are there without weekend days in a month?

```
# Finding the number of days of the week, less weekend days, each month.

import calendar as cal

# Find days in the week
def count_weekends_in_month(year, month):
    """Count how many Saturdays and Sundays in a month"""
    month_cal = cal.monthcalendar(year, month)
    saturdays = sundays = 0

    for week in month_cal:
        if week[cal.SATURDAY] != 0:  # 0 means no day
            saturdays += 1
        if week[cal.SUNDAY] != 0:
            sundays += 1

    return saturdays, sundays

def getMonthLen(year, month):
```

```
    """ Find the number of days in a month"""
    myCal = cal.monthcalendar(year, month)
    # get a count of days in month
    count = 0

    for i in range(len(myCal)):
        line =  myCal[i]
        # print(f"my line is {line}")
        for j in line:
            #print(f"j is {j}")
            if j != 0:
                count = count + 1
    #print(f"Number of days : {count}")
    return count

#Find number of week days
month = 12
year = 2025
print(f" Month = {month}")
print(f" Year = {year}")
sat, sun = count_weekends_in_month(year, month)
print(f" Number of weekend days : {sat + sun} ")
print(f" Number of days in month : {getMonthLen(year, month)}")
print(f" Number of weekdays : {getMonthLen(year, month) - (sat + sun)} ")
```

```
# Finding the number of days of the week, less weekend days, each month.

import calendar as cal

# Find days in the week
def count_weekends_in_month(year, month):
    """Count how many Saturdays and Sundays in a month"""
    month_cal = cal.monthcalendar(year, month)
    saturdays = sundays = 0

    for week in month_cal:
        if week[cal.SATURDAY] != 0:  # 0 means no day
            saturdays += 1
        if week[cal.SUNDAY] != 0:
            sundays += 1

    return saturdays, sundays
```

```
def getMonthLen(year, month):
    """ Find the number of days in a month"""
    myCal = cal.monthcalendar(year, month)
    # get a count of days in month
    count = 0

    for i in range(len(myCal)):
        line =  myCal[i]
        # print(f"my line is {line}")
        for j in line:
            #print(f"j is {j}")
            if j != 0:
                count = count + 1
    #print(f"Number of days : {count}")
    return count

#Find number of week days
month = 12
year = 2025
print(f" Month = {month}")
print(f" Year = {year}")
sat, sun = count_weekends_in_month(year, month)
print(f" Number of weekend days : {sat + sun} ")
print(f" Number of days in month : {getMonthLen(year, month)}")
print(f" Number of weekdays : {getMonthLen(year, month) - (sat + sun)} ")
```

```
 Month = 12
 Year = 2025
 Number of weekend days : 8
 Number of days in month : 31
 Number of weekdays : 23
```

Return to Challenge Slide

Solution :: Finding Areas

```
import math

def circle_calculator(radius):
```

```
    """Calculate everything about a circle given its radius"""
    # Area =   × r²
    area = math.pi * radius**2

    # Circumference = 2 ×   × r
    circumference = 2 * math.pi * radius

    # Diameter = 2 × r
    diameter = 2 * radius

    # area of sphere = 4 * math.pi * r * r
    sphereArea = 4 * math.pi * radius**2

    # return a dictionary
    return {
        'radius': radius,
        'diameter': diameter,
        'area': round(area, 2),
        'circumference': round(circumference, 2),
        'sphereArea': round(sphereArea, 2)
    }

# Test with different radii
radii = [1, 5, 10, 25]
print("  Circle Calculator Results:")
print("-" * 80)
print(f"{'Radius':<8} {'Diameter':<10} {'Area':<12} {'Circumference':<16} {'Sphere Area'}")
print("-" * 80)

for r in radii:
    info = circle_calculator(r)
    # pull values from dictionary structure
    print(f"{info['radius']:<8} {info['diameter']:<10} {info['area']:<12} {info['circumferenc
```

```
import math

def circle_calculator(radius):
    """Calculate everything about a circle given its radius"""
    # Area =   × r²
    area = math.pi * radius**2

    # Circumference = 2 ×   × r
```

```
    circumference = 2 * math.pi * radius

    # Diameter = 2 × r
    diameter = 2 * radius

    # area of sphere = 4 * math.pi * r * r
    sphereArea = 4 * math.pi * radius**2

    # return a dictionary
    return {
        'radius': radius,
        'diameter': diameter,
        'area': round(area, 2),
        'circumference': round(circumference, 2),
        'sphereArea': round(sphereArea, 2)
    }

# Test with different radii
radii = [1, 5, 10, 25]
print("  Circle Calculator Results:")
print("-" * 80)
print(f"{'Radius':<8} {'Diameter':<10} {'Area':<12} {'Circumference':<16} {'Sphere Area'}")
print("-" * 80)

for r in radii:
    info = circle_calculator(r)
    # pull values from dictionary structure
    print(f"{info['radius']:<8} {info['diameter']:<10} {info['area']:<12} {info['circumferenc
```

```
  Circle Calculator Results:
--------------------------------------------------------------------------------
Radius   Diameter   Area         Circumference    Sphere Area
--------------------------------------------------------------------------------
1        2          3.14         6.28             6.28
5        10         78.54        31.42            31.42
10       20         314.16       62.83            62.83
25       50         1963.5       157.08           157.08
```

Return to Challenge Slide

Summary & Next Steps: You're Now a Module Master!

> What You've Accomplished Today!
>
> You've mastered the essential skills of working with Python modules and files!

Skills Unlocked

- **Import mastery**: Using built-in modules (`calendar`, `math`, `random`)
- **File operations**: Reading/writing text and CSV files

- **Module creation**: Building your own reusable code libraries
- **Professional practices**: Documentation, error handling, organization

Modules You've Explored: - `calendar` - Date calculations and calendar generation - `math` - Mathematical functions and constants - `random` - Random numbers and choices - `csv` - Structured data handling - `datetime` - Working with dates and times

Real-World Applications

- **Data Analysis**: Reading CSV files, calculating statistics
- **File Management**: Organizing and processing text data
- **Code Organization**: Creating reusable function libraries
- **Problem Solving**: Using the right tool for each task

Next Adventures: - Explore more built-in modules (`json`, `requests`, `sqlite3`) - Create package structures with multiple modules - Publish your modules for others to use - Integrate modules into larger projects

Keep Exploring!

> Important
>
> **Remember**: Python has hundreds of built-in modules and thousands of third-party packages. The skills you learned today are your keys to unlock infinite possibilities!
> **Pro Tip**: Check out the Python Module Index to discover more amazing tools!