Python Functions, Modules, Files & Exceptions

Day 4: Building Reusable, Real-World Python Applications

DAY 4 OVERVIEW

What We'll Master Today

01

Functions & Reusability

Write code once, use it everywhere

02

Parameters & Returns

Build flexible, powerful functions

03

Modules & Packages

Leverage Python's ecosystem

04

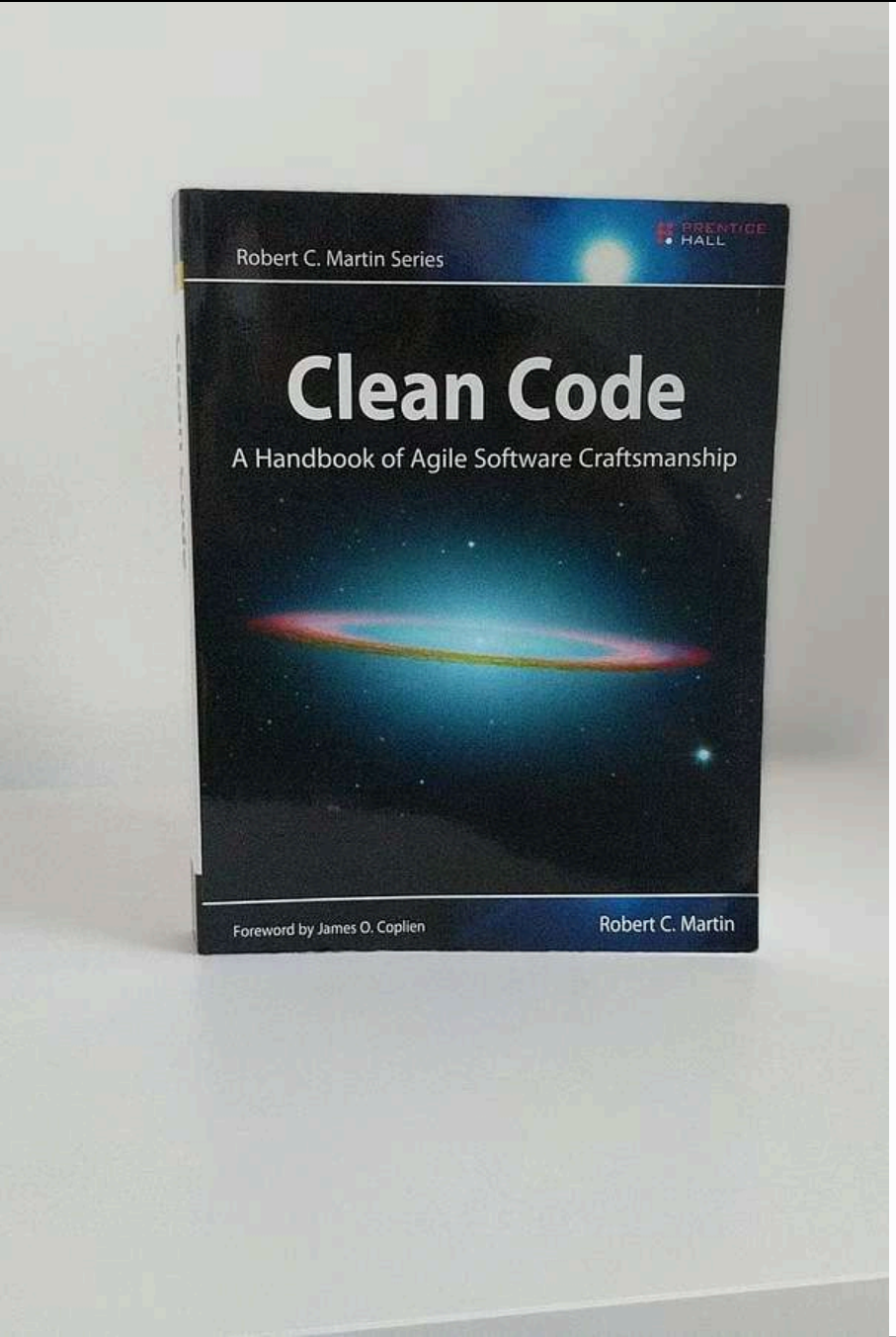
File Operations

Read, write, and process data

05

Exception Handling

Build robust, error-proof code



Why Functions Matter

The Problem

Without functions, you're constantly rewriting the same code. It's time-consuming, error-prone, and makes your programs harder to maintain and debug.

The Solution

Functions let you write code once and reuse it anywhere. They make your programs modular, testable, and professional-grade.

Real-World Function Benefits

Reusability

Write once, use multiple times across your entire program

Maintainability

Fix bugs in one place instead of hunting through thousands of lines

Abstraction

Hide complex operations behind simple function calls

Testability

Test individual pieces of code independently for reliability

Your First Function

Functions in Python use the `def` keyword followed by the function name and parentheses. The code block inside defines what the function does.

```
def greet_user():
    print("Welcome to Python!")
    print("Let's learn functions together")

# Call the function
greet_user()
```

Function names should be lowercase with underscores between words (snake_case). Choose descriptive names that explain what the function does.

Parameters: Making Functions Flexible

Parameters allow functions to accept input, making them flexible and reusable in different contexts. Think of them as variables that receive values when the function is called.

```
def greet_user(name, time_of_day):
    print(f"Good {time_of_day}, {name}!")
    print("Ready to code?")

greet_user("Sarah", "morning")
greet_user("Marcus", "evening")
```

Key Concepts

- **Parameters** are defined in the function
- **Arguments** are values passed when calling
- You can have multiple parameters
- Order matters unless you use keyword arguments

Return Values: Getting Results Back

The `return` statement sends data back from a function to where it was called. This lets you use function results in calculations, assignments, or other operations.

```
def calculate_tax(amount, rate=0.08):
 tax = amount * rate
 total = amount + tax
 return total

# Use the returned value
final_price = calculate_tax(100)
print(f"Total: ${final_price}")
```

Multiple Return Values

Python functions can return multiple values as a tuple. This is incredibly useful when you need to send back several related pieces of data.

```
def analyze_data(numbers):
    total = sum(numbers)
    average = total / len(numbers)
    maximum = max(numbers)
    return total, average, maximum

sum_val, avg_val, max_val = analyze_data([10, 20, 30, 40])
```

Unpacking Returns

When a function returns multiple values, you can capture them in separate variables. This is called **tuple unpacking**.

You can also capture everything in one variable, which becomes a tuple.

CRITICAL CONCEPT

Variable Scope: Where Variables Live

Scope determines where in your code a variable can be accessed. Understanding scope prevents bugs and helps you write cleaner code.

Local Scope

Variables created inside a function exist only within that function. They're destroyed when the function ends.

Global Scope

Variables created outside functions exist throughout your entire program. They're accessible everywhere.

Best Practice

Minimize global variables. Pass data through parameters instead. This makes code more predictable and testable.

Scope in Action

```
price = 100  # Global variable

def apply_discount(discount_rate):
    # Local variables
    discount = price * discount_rate
    final_price = price - discount
    return final_price

result = apply_discount(0.2)
print(result)  # Works fine
print(discount)  # Error! discount doesn't exist here
```

Local variables are created when a function starts and deleted when it ends. This keeps your program's memory efficient and prevents naming conflicts.

The Global Keyword

While not recommended for beginners, you can modify global variables inside functions using the `global` keyword.

```
counter = 0

def increment():
    global counter
    counter += 1

increment()
increment()
print(counter)  # Prints 2
```

Use Sparingly

Modifying global variables makes code harder to debug and test. Instead, use parameters and return values to pass data in and out of functions.

MODULES & PACKAGES

Extending Python's Power

Python's strength lies in its vast ecosystem. Modules and packages let you leverage thousands of pre-built solutions instead of reinventing the wheel.

What Are Modules?

Module = Python File

A module is simply a .py file containing functions, classes, and variables you can reuse

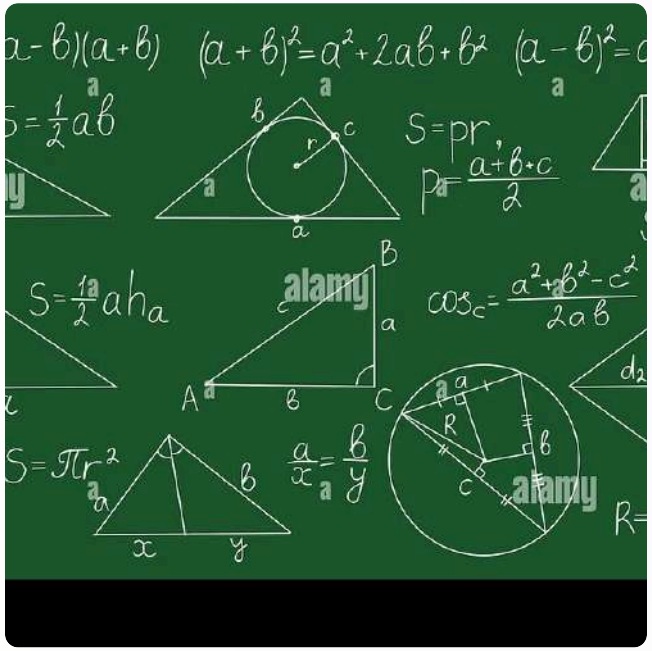
Import to Use

Use the `import` statement to bring module functionality into your program

Instant Power

Access sophisticated functionality without writing it yourself

Built-in Modules You'll Use Daily



math

Mathematical operations: square roots, trigonometry, logarithms, and constants like pi

random

Generate random numbers, shuffle lists, make random choices for games and simulations

datetime

Work with dates and times: formatting, calculations, timestamps, and timezone handling

Importing Modules

Python offers several ways to import modules. Choose the method that makes your code clearest and most efficient.

```
# Import entire module
import math
print(math.sqrt(16)) # Use math.function_name

# Import specific functions
from math import sqrt, pi
print(sqrt(16)) # Use function directly

# Import with alias
import datetime as dt
today = dt.date.today()
```

Creating Your Own Modules

calculator.py

```
def add(a, b):
    return a + b

def multiply(a, b):
    return a * b

PI = 3.14159
```

main.py

```
import calculator

result = calculator.add(5, 3)
print(result)

area = calculator.PI * (10 ** 2)
print(f"Area: {area}")
```

Any Python file can be a module. Just save your functions in a .py file and import it into other programs.

PIP & PACKAGES

The Python Package Index

500K+

Available Packages

Over half a million packages on PyPI ready to use

1B+

Monthly Downloads

Billions of package downloads by developers worldwide

100%

Free & Open

All packages are free and open-source

Installing Packages with pip

pip is Python's package installer. It downloads and installs packages from the Python Package Index (PyPI) with a single command.

```
# Install a package
pip install requests

# Install specific version
pip install pandas==1.5.0

# Upgrade a package
pip install --upgrade numpy

# List installed packages
pip list
```

Run pip commands in your terminal or command prompt, not in Python itself. Windows users might need to use `python -m pip install` instead.

Popular Packages for Beginners

requests

Make HTTP requests to APIs and websites. Perfect for web scraping and API integration.

pandas

Analyze and manipulate data in tables. Essential for data science and analysis work.

matplotlib

Create visualizations and charts. Turn your data into beautiful, informative graphics.

pillow

Process and manipulate images. Resize, crop, filter, and transform photos programmatically.

FILE HANDLING

Working with Files

Real-world programs need to save and load data. File handling lets you read input files, process data, generate reports, and persist information between program runs.

Opening and Reading Files

Python's `open()` function creates a file object you can read from or write to. Always close files when done, or use a context manager to handle it automatically.

```
# Manual approach
file = open('data.txt', 'r')
content = file.read()
file.close()

# Better: context manager (recommended)
with open('data.txt', 'r') as file:
 content = file.read()
 print(content)
# File closes automatically
```

File Opening Modes

'r' - Read

Opens for reading (default). File must exist or you'll get an error.

'w' - Write

Opens for writing. Creates new file or **overwrites** existing content.

'a' - Append

Opens for writing. Adds content to the end without erasing existing data.

'r+' - Read/Write

Opens for both reading and writing. File must exist.

Reading File Content

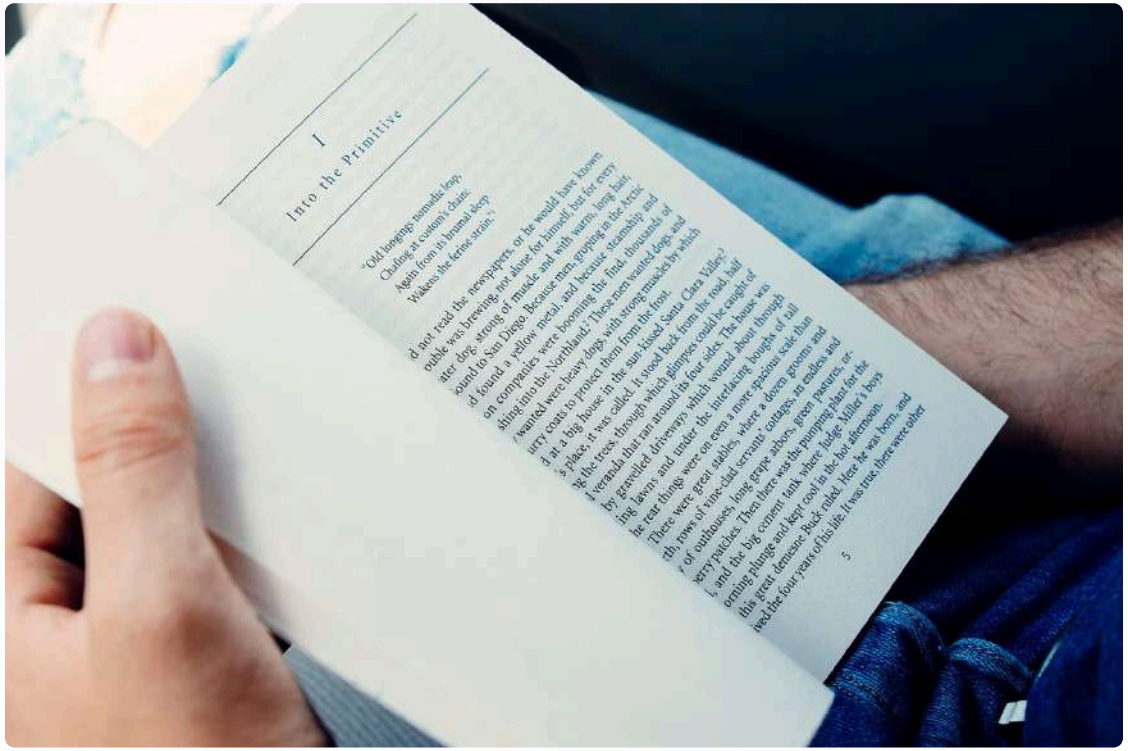
```
# Read entire file
with open('story.txt') as f:
    content = f.read()

# Read line by line
with open('data.txt') as f:
    for line in f:
        print(line.strip())

# Read all lines into list
with open('items.txt') as f:
    lines = f.readlines()
```

Best Practices

- Use `read()` for small files
- Loop line-by-line for large files
- Use `strip()` to remove newlines
- Always use `with` statements



Writing to Files

Writing files lets you save program output, generate reports, log events, and create data files for other programs to use.

```
# Write new content (overwrites file)
with open('output.txt', 'w') as file:
    file.write("Hello, World!\n")
    file.write("Python is awesome!\n")

# Append to existing file
with open('log.txt', 'a') as file:
    file.write("New log entry\n")
```

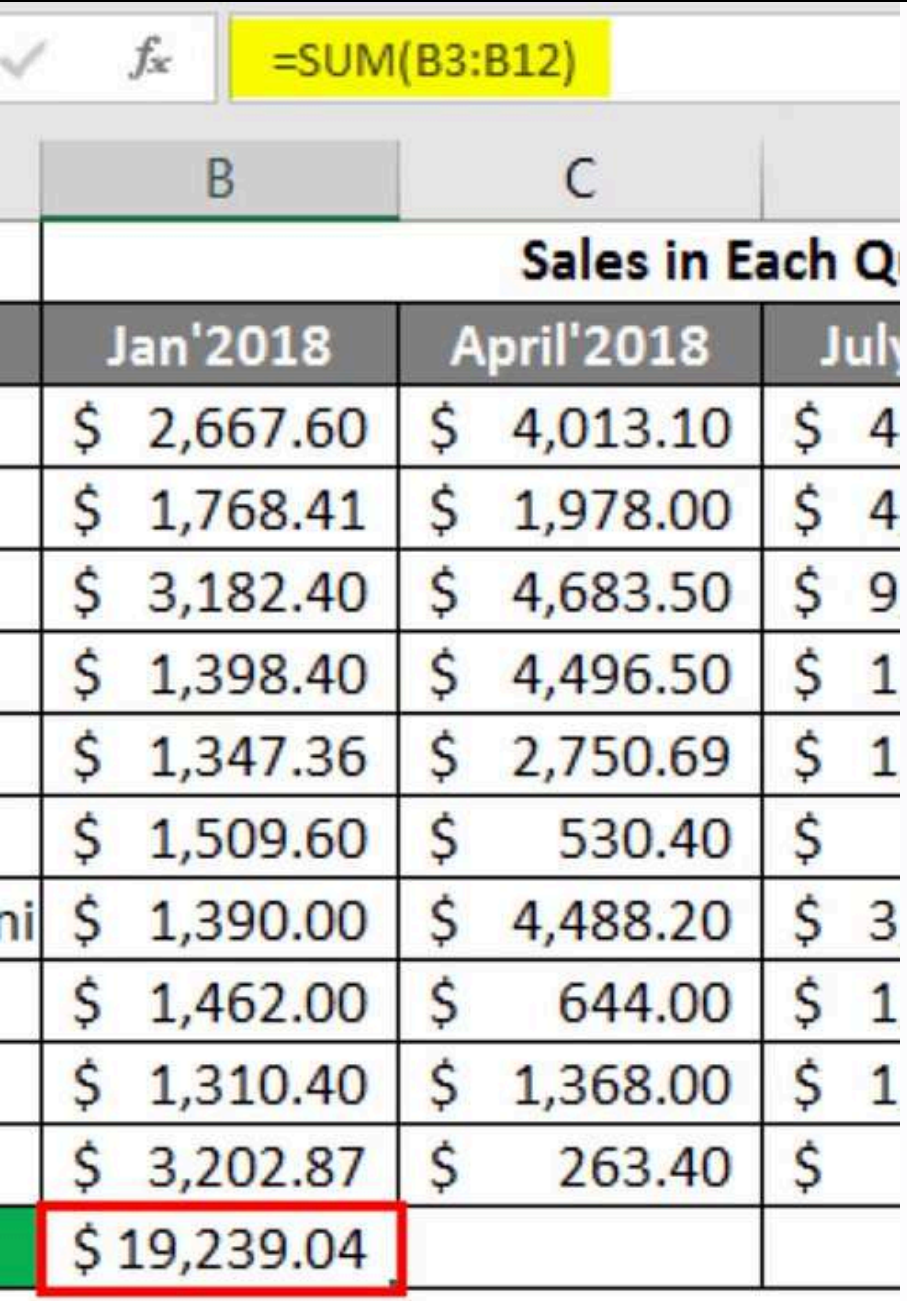
The 'w' mode will delete all existing content. Use 'a' mode to add to a file without erasing it. Don't forget the \n character for new lines!

Real-World Example: Todo List

```
def add_task(task):
 with open('tasks.txt', 'a') as file:
 file.write(f"{task}\n")

def show_tasks():
 try:
 with open('tasks.txt', 'r') as file:
 tasks = file.readlines()
 for i, task in enumerate(tasks, 1):
 print(f"{i}. {task.strip()}")
 except FileNotFoundError:
 print("No tasks yet!")

add_task("Learn Python functions")
add_task("Practice file handling")
show_tasks()
```



CSV FILES

Working with CSV Data

CSV (Comma-Separated Values) files are the most common format for structured data. They're used everywhere: databases, Excel exports, data analysis, and web applications.

The CSV Module

Python's built-in `csv` module handles all the complexity of parsing CSV files, including quoted fields, different delimiters, and edge cases.

```
import csv

# Reading CSV
with open('customers.csv', 'r') as file:
 reader = csv.reader(file)
 for row in reader:
 print(row) # Each row is a list

# Reading with headers
with open('customers.csv', 'r') as file:
 reader = csv.DictReader(file)
 for row in reader:
 print(row['name'], row['email'])
```

Writing CSV Files

Writing Lists

```
import csv

data = [
    ['Name', 'Age', 'City'],
    ['Alice', 30, 'NYC'],
    ['Bob', 25, 'LA']
]

with open('people.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(data)
```

Writing Dictionaries

```
import csv

data = [
 {'name': 'Alice', 'age': 30},
 {'name': 'Bob', 'age': 25}
]

with open('people.csv', 'w', newline='') as f:
 writer = csv.DictWriter(f, fieldnames=['name', 'age'])
 writer.writeheader()
 writer.writerows(data)
```

CSV Processing Example

```
import csv

def calculate_average_price(filename):
    total = 0
    count = 0

    with open(filename, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            total += float(row['price'])
            count += 1

    return total / count if count > 0 else 0

avg = calculate_average_price('products.csv')
print(f"Average price: ${avg:.2f}")
```

This pattern—reading CSV data, processing it, and calculating results—is incredibly common in data analysis and business applications.

⚠ EXCEPTION HANDLING

What Are Exceptions?

1

Code Runs Normally

Your program executes line by line as expected

2

Error Occurs

Something unexpected happens: missing file, invalid input, network timeout

3

Exception Raised

Python creates an exception object describing the error

4

Program Crashes (Without Handling)

If unhandled, the exception stops your program with an error message

5

Graceful Recovery (With Handling)

With proper handling, your program recovers and continues running

Why Exception Handling Matters

User Experience

Show helpful error messages instead of cryptic crashes. Guide users toward correct input.

Robustness

Handle edge cases gracefully. Your program continues working even when things go wrong.

Debugging

Log errors for later analysis. Understand what went wrong and when it happened.

Try-Except Basics

The `try-except` block lets you attempt risky code and handle errors if they occur. It's like saying "try this, but if it fails, do this instead."

```
try:
 number = int(input("Enter a number: "))
 result = 100 / number
 print(f"Result: {result}")
except ValueError:
 print("That's not a valid number!")
except ZeroDivisionError:
 print("Cannot divide by zero!")
except Exception as e:
 print(f"Something went wrong: {e}")
```

The Complete Try Block

try

Code that might raise an exception

except

Handles specific errors when they occur

else

Runs only if no exceptions occurred

finally

Always runs, whether exception occurred or not. Perfect for cleanup.

Try-Except-Else-Finally

```
try:
 file = open('data.txt', 'r')
 data = file.read()
 number = int(data)
except FileNotFoundError:
 print("File not found!")
except ValueError:
 print("File doesn't contain a valid number")
else:
 # Only runs if no exceptions
 print(f"Successfully read: {number}")
finally:
 # Always runs
 try:
 file.close()
 print("File closed")
 except:
 pass
```

Common Python Exceptions

FileNotFoundError

Trying to open a file that doesn't exist

ValueError

Converting invalid data types, like int("hello")

ZeroDivisionError

Dividing any number by zero

KeyError

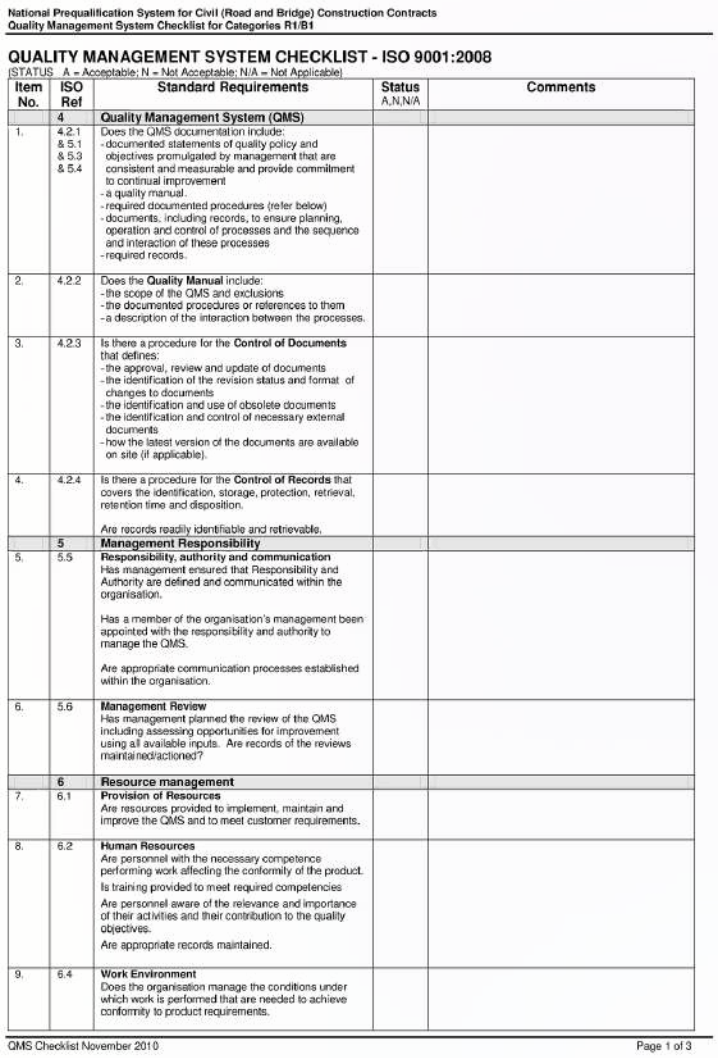
Accessing a dictionary key that doesn't exist

IndexError

Accessing a list index that's out of range

AttributeError

Calling a method or attribute that doesn't exist



National Prequalification System for Civil (Road and Bridge) Construction Contracts
Quality Management System Checklist for Categories R1/B1

QUALITY MANAGEMENT SYSTEM CHECKLIST - ISO 9001:2008

(STATUS A = Acceptable; N = Not Acceptable; N/A = Not Applicable)

Item No.	ISO Ref	Standard Requirements	Status A,N,N/A	Comments
	4	**Quality Management System (QMS)**		
1.	4.2.1 & 5.1 & 5.3 & 5.4	Does the QMS documentation include: - documented statements of quality policy and objectives promulgated by management that are consistent and measurable and provide commitment to continual improvement - a quality manual. - required documented procedures (refer below) - documents, including records, to ensure planning, operation and control of processes and the sequence and interaction of these processes - required records.		
2.	4.2.2	Does the **Quality Manual** include: - the scope of the QMS and exclusions - the documented procedures or references to them - a description of the interaction between the processes.		
3.	4.2.3	Is there a procedure for the **Control of Documents** that defines: - the approval, review and update of documents - the identification of the revision status and format of changes to documents - the identification and use of obsolete documents - the identification and control of necessary external documents - how the latest version of the documents are available on site (if applicable).		
4.	4.2.4	Is there a procedure for the **Control of Records** that covers the identification, storage, protection, retrieval, retention time and disposition. Are records readily identifiable and retrievable.		
	5	**Management Responsibility**		
5.	5.5	**Responsibility, authority and communication** Has management ensured that Responsibility and Authority are defined and communicated within the organisation. Has a member of the organisation's management been appointed with the responsibility and authority to manage the QMS. Are appropriate communication processes established within the organisation.		
6.	5.6	**Management Review** Has management planned the review of the QMS including assessing opportunities for improvement using all available inputs. Are records of the reviews maintained/actioned?		
	6	**Resource management**		
7.	6.1	**Provision of Resources** Are resources provided to implement, maintain and improve the QMS and to meet customer requirements.		
8.	6.2	**Human Resources** Are personnel with the necessary competence performing work affecting the conformity of the product. Is training provided to meet required competencies Are personnel aware of the relevance and importance of their activities and their contribution to the quality objectives. Are appropriate records maintained.		
9.	6.4	**Work Environment** Does the organisation manage the conditions under which work is performed that are needed to achieve conformity to product requirements.		

QMS Checklist November 2010 Page 1 of 3

Best Practices for Exception Handling

Do This ✓

- Catch specific exceptions
- Provide helpful error messages
- Use finally for cleanup
- Log errors for debugging
- Fail gracefully with alternatives

Avoid This ☒

- Catching all exceptions with bare except
- Hiding errors silently
- Using exceptions for control flow
- Raising generic Exception objects
- Ignoring important errors

Real-World Error Handling

```
def safe_divide(a, b):
 """Safely divide two numbers with error handling"""
 try:
 result = a / b
 except ZeroDivisionError:
 print("Error: Cannot divide by zero")
 return None
 except TypeError:
 print("Error: Both arguments must be numbers")
 return None
 else:
 return result
 finally:
 print("Division operation completed")

print(safe_divide(10, 2)) # Returns 5.0
print(safe_divide(10, 0)) # Returns None, prints error
print(safe_divide(10, "2")) # Returns None, prints error
```

PCAP EXAM TIPS

PCAP Certification Focus Areas

Functions Parameters, return values, scope, default arguments	**Modules** Importing, creating modules, namespaces
Files Open modes, reading/writing, context managers	**Exceptions** Try-except, exception hierarchy, raising exceptions

Key Takeaways

Functions enable code reuse

Write once, use everywhere. They're the foundation of professional programming.

Modules extend Python's power

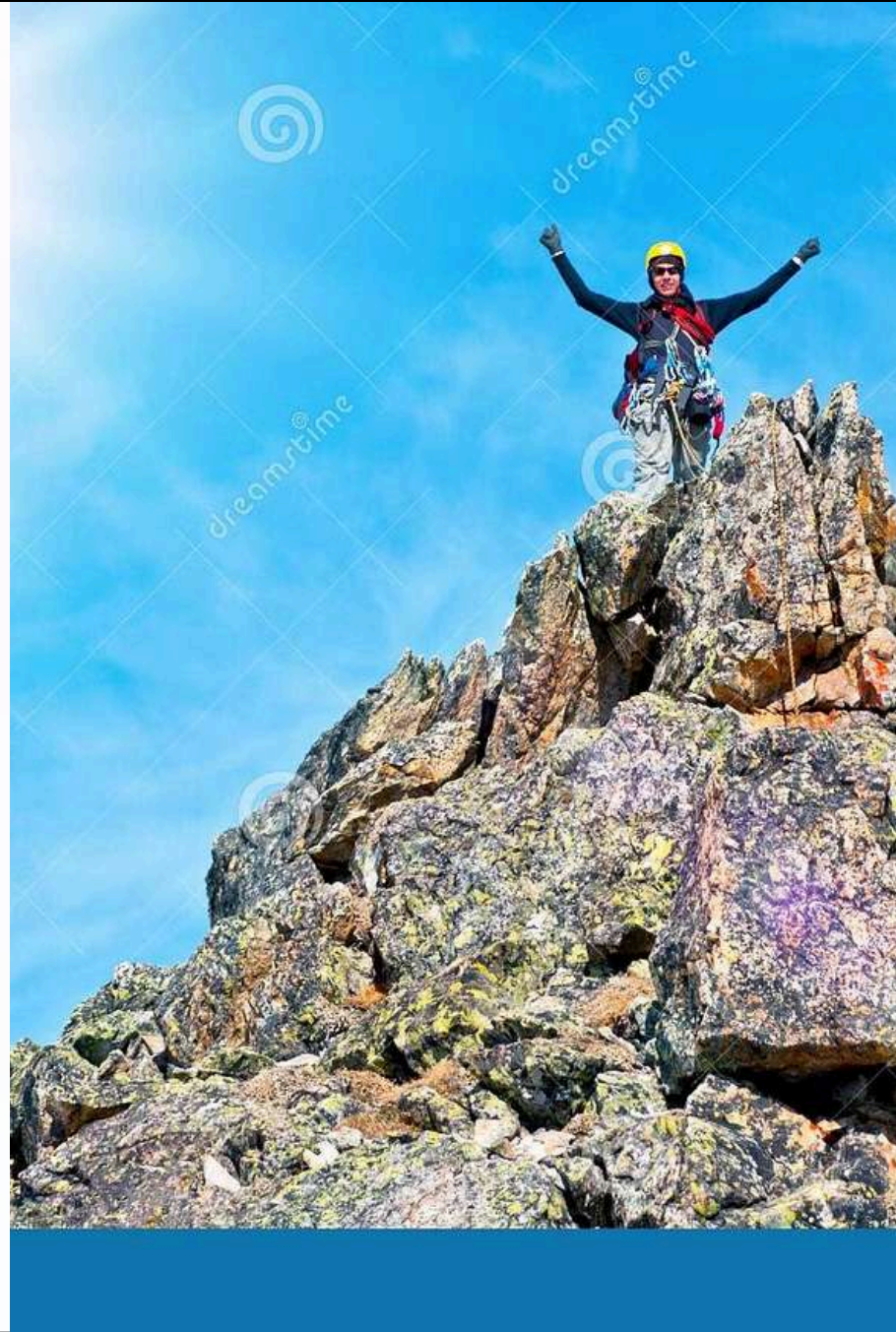
Leverage built-in modules and pip packages to solve problems faster.

File handling persists data

Read input, process information, and save results for real-world applications.

Exception handling builds robust code

Anticipate errors, handle them gracefully, and create reliable software.



Practice Makes Perfect

Your Next Steps

1. Build a file-based todo list application
2. Create a CSV data analyzer for sales reports
3. Write a module with utility functions you'll reuse
4. Practice exception handling with user input
5. Explore popular packages like requests and pandas

Tomorrow: Day 5 - Object-Oriented Programming