ISU | ISTINYE UNIVERSITY
ISTANBUL

COE203: Advanced Programming with Python

Fall 25-26

WEEK 3: Functions, Scripting Fundamentals, Advanced File Ops

Presented by **Asst. Prof. Yiğit Bekir Kaya**

Week 3: Functions & Smart Data Storage

Building Better Code

📚 LAST WEEK

- Lists → Store collections
- Loops → Process automatically
- Files → Read/write data

🎯 THIS WEEK

Functions → Organize code

Dictionaries → Smart storage

❓ **Quick Poll: Who has copy-pasted the same code 3+ times in Project 1?**

Copy-Paste Code is Bad Code

The Repetition Problem

❌ **REPETITIVE CODE**

```
score1 = 85
if score1 >= 90: grade1 = 'A'
elif score1 >= 80: grade1 = 'B'
else: grade1 = 'C'

score2 = 92
if score2 >= 90: grade2 = 'A'
elif score2 >= 80: grade2 = 'B'
else: grade2 = 'C'

score3 = 78
if score3 >= 90: grade3 = 'A'
elif score3 >= 80: grade3 = 'B'
else: grade3 = 'C'
```

😱 **PROBLEMS**

15	250	50
lines for 3 students	lines for 50 students	places to fix bugs

💭 **Think-Pair-Share: How would you fix this?**

(3 minutes)

Functions: Write Once, Use Forever

The Solution

✅ BETTER CODE

```
def calculate_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    else:
        return 'C'

# Use it!
grade1 = calculate_grade(85)  # 'B'
grade2 = calculate_grade(92)  # 'A'
grade3 = calculate_grade(78)  # 'C'
```

🔧 FUNCTION ANATOMY

```
def function_name(parameters):  # ← Definition
    # Do work here
    return result              # ← Send answer

result = function_name(arguments)  # ← Call it
```

Write once | Easy bugs fix | Easy testing | Readable

Three Function Patterns

Input → Output

TYPE 1 No Input, No Output (Actions)

```
def print_welcome():
    print("Welcome!")
    print("=" * 30)

print_welcome()  # Just prints
```

TYPE 2 Input, Output (Calculators)

```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

temp = celsius_to_fahrenheit(25)  # 77.0
```

TYPE 3 Input, No Output (Actions + Data)

```
def greet_user(name):
    print(f"Hello, {name}!")

greet_user("Alice")  # Prints only
```

⚡ Rule: Need data back? Use return!

When to Use Functions

Real-World Patterns

💰 USE CASE 1: Repeated Calculations

```
def calculate_total(price, tax_rate=0.08):
    return price * (1 + tax_rate)

total1 = calculate_total(50.00)
total2 = calculate_total(120.00)
```

✉️ USE CASE 2: Complex Logic

```
def is_valid_email(email):
    return '@' in email and '.' in email

if is_valid_email(user_input):
    print("Valid!")
```

📁 USE CASE 3: Breaking Down Big Problems

```
def organize_files():
    create_folders()      # Step 1
    scan_files()          # Step 2
    move_files()          # Step 3
    generate_report()     # Step 4
```

✨ Code appears twice? Make it a function!

Variable Scope: Where Can You Use Variables?

Quick Rule

❌ **LOCAL (inside function)**

```
def calculate():
    result = 10 * 5  # Only exists inside
    return result

print(result)  # ERROR! Doesn't exist here
```

✅ **GLOBAL (outside functions)**

```
TAX_RATE = 0.08  # Available everywhere

def calculate_total(price):
    return price * (1 + TAX_RATE)  # Can use
```

🗒️ **SIMPLE RULES**

🔒 Variables in functions = **local** (vanish after)

🌍 Variables outside = **global** (exist everywhere)

Dictionaries: Smart Data Storage

Looking Up Information

❌ PROBLEM WITH LISTS

```
student = ["Alice", 20, "CS", 3.8]
name = student[0]  # Which is which? 🤔
```

✅ SOLUTION - DICTIONARIES

```
student = {
    "name": "Alice",
    "age": 20,
    "major": "CS",
    "gpa": 3.8
}

print(student["name"])  # Alice — clear!
```

CREATING

```
empty = {}
grades = {
    "Alice": 85,
    "Bob": 92
}
```

ACCESSING

```
score = grades["Alice"]
score = grades.get("Bob")
score = grades.get("?", 0)
```

Working with Dictionaries

Common Operations

➕ ADDING/UPDATING

```
grades = {"Alice": 85}
grades["Bob"] = 92     # Add
grades["Alice"] = 90   # Update
```

🔍 CHECKING

```
if "Alice" in grades:
    print(grades["Alice"])
```

🔄 LOOPING

```
# Loop keys
for name in grades:
    print(name)
```

```
# Loop key-value pairs
for name, score in grades.items():
    print(f"{name}: {score}")
```

📁 REAL PROJECT 1 USE

```
categories = {
    "jpg": "Images",
    "pdf": "Documents",
    "mp3": "Music"
}
folder = categories.get("jpg", "Other")  # "Images"
```

Functions: Returning Multiple Values

Two Methods

📦 METHOD 1: Return Dictionary

```
def analyze_text(text):
    return {
        "length": len(text),
        "words": len(text.split()),
        "uppercase": text.isupper()
    }

stats = analyze_text("Hello World")
print(stats["words"])  # 2
```

🎯 METHOD 2: Return Tuple

```
def get_min_max(numbers):
    return min(numbers), max(numbers)  # Tuple

minimum, maximum = get_min_max([5, 2, 9, 1])
print(f"Min: {minimum}, Max: {maximum}")
```

🔒 TUPLES vs LISTS

```
point = (10, 20)    # Tuple - immutable ❌
items = [10, 20]    # List - mutable ✅

items[0] = 99       # Works
point[0] = 99       # ERROR!
```

💡 Use tuples for data that shouldn't change!

Quick Tour: Sets & Immutability

Other Data Types

🎲 SETS - Unique Items Only

```
extensions = {"jpg", "png", "jpg", "gif"}
print(extensions)  # {"jpg", "png", "gif"}

if "jpg" in extensions:  # Very fast!
    print("Found")
```

🔐 MUTABILITY SUMMARY

✅ **MUTABLE**

```
list[0] = 99  ✓
dict["key"] = 1  ✓
```

❌ **IMMUTABLE**

```
tuple[0] = 99  ✗
string[0] = "H"  ✗
```

🎓 Block 1 Complete!

Functions	Dictionaries	Tuples/Sets
Organize code	Smart storage	More tools

BREAK – 20m

From Fragments to Full Scripts

Block 2: Professional Script Structure

📝 WHAT WE'VE WRITTEN SO FAR

```
# Fragments in notebooks
print("Hello")
x = 5
# ... random code ...
```

⚡ WHAT PROFESSIONALS WRITE

```
$ python analyze.py --file data.csv --output report.txt
Processing data.csv...
Found 1000 rows
Analysis complete!
Results saved to report.txt
```

🎯 TODAY YOU'LL LEARN:

- Taking user input (input, argparse)
- Beautiful output (f-strings)
- String operations
- Proper script structure

Interactive Programs

Getting User Input with input()

BASIC INPUT

```
name = input("What's your name? ")
print(f"Hello, {name}!")
```

INPUT WITH VALIDATION

```
age = input("Enter age: ")
age = int(age) # Convert to number
if age >= 18:
  print("You can vote!")
```

REAL EXAMPLE - LOOP UNTIL VALID

```
def get_valid_choice():
  while True:
    choice = input("Process files? (yes/no): ").lower()
    if choice in ["yes", "y"]:
      return True
    elif choice in ["no", "n"]:
      return False
    else:
      print("Please enter yes or no")
```

Professional Input Handling

Command-Line Arguments with argparse

⚡ WHY COMMAND-LINE ARGUMENTS?

❌ Manual every time:

```
source = input("Source directory: ")
```

✅ Automatic:

```
$ python script.py --source Downloads/
```

BASIC ARGPARSE

```
import argparse
parser = argparse.ArgumentParser(description='Process files')
parser.add_argument('--source', required=True, help='Source dir')
parser.add_argument('--dest', required=True, help='Destination')
parser.add_argument('--type', default='all', help='File type')
args = parser.parse_args()
print(f"Source: {args.source}")
```

RUNNING IT

```
$ python script.py --help # Auto-generated help!
$ python script.py --source Downloads/ --dest Organized/
```

argparse Power Features

Flags, Types, and Choices

BOOLEAN FLAGS

```
parser.add_argument('--verbose', action='store_true')
if args.verbose:
  print("Detailed output...")
```

TYPED ARGUMENTS

```
parser.add_argument('--count', type=int, default=10)
for i in range(args.count):
  process_item(i)
```

LIMITED CHOICES

```
parser.add_argument('--type',
  choices=['all', 'images', 'docs'])
# Only allows these values!
```

💡 **argparse validates inputs automatically!**

String Mastery

Essential String Operations for Real Programs

CASE HANDLING

```
ext = "JPG".lower() # "jpg"
name = "alice".upper() # "ALICE"
title = "hello".capitalize() # "Hello"
```

CHECKING CONTENT

```
filename = "photo.jpg"
if filename.endswith(".jpg"):
  print("Image file")
if filename.startswith("backup_"):
  print("Backup file")
```

CLEANING INPUT

```
user_input = " jpg "
clean = user_input.strip() # "jpg"
```

SPLITTING & JOINING

```
path = "user/docs/file.txt"
parts = path.split('/') # ["user", "docs", "file.txt"]
words = ["Hello", "World"]
sentence = " ".join(words) # "Hello World"
```

Real Project 1 Example

Normalizing File Extensions

🎯 THE PROBLEM

Users save files with inconsistent extensions:

- Photo.JPG
- image.jpeg
- pic.jpg

All should go to the SAME folder!

THE SOLUTION

```
def normalize_extension(filename):
  ext = filename.split('.')[-1] # Get last part
  ext = ext.lower() # Lowercase
  ext = ext.strip() # Remove spaces
  # Handle variations
  if ext in ["jpeg", "jpg"]:
    return "jpg"
  return ext
ext = normalize_extension("Photo.JPG") # "jpg"
```

✅ String operations make your code robust!

F-Strings: Beautiful Formatted Output

Professional Output Formatting

OLD WAY (UGLY) ❌

```
print("Found " + str(count) + " files in " + folder)
```

NEW WAY (F-STRINGS) ✅

```
print(f"Found {count} files in {folder}")
```

NUMBER FORMATTING

```
price = 19.99999
print(f"Price: ${price:.2f}") # Price: $20.00
percent = 0.8547
print(f"Done: {percent:.1%}") # Done: 85.5%
```

EXPRESSIONS INSIDE

```
files = ["a.jpg", "b.pdf"]
print(f"Found {len(files)} files")
name = "alice"
print(f"Hello, {name.upper()}!") # Hello, ALICE!
```

Multi-line F-Strings

Creating Professional Reports

MULTI-LINE REPORTS

```
report = f"""
Files Organized: {count}
Source: {source}
Destination: {dest}
Time: {time:.2f} seconds
"""
print(report)
```

OUTPUT EXAMPLE

```
Files Organized: 42
Source: Downloads/
Destination: Organized/
Time: 1.23 seconds
```

💡 **F-strings make your programs look professional!**

Proper Script Structure

The if __name__ == '__main__': Pattern

❌ THE PROBLEM

```
# script.py
def process_data():
  print("Processing...")
process_data() # Runs immediately when imported!
```

✅ THE SOLUTION

```
# script.py
def process_data():
  print("Processing...")
if __name__ == '__main__':
  process_data() # Only runs when executed directly
```

WHAT THIS MEANS

Running directly:

```
$ python script.py
```

→ __name__ is '__main__', so code runs

Importing in another file:

```
from script import process_data
```

→ __name__ is 'script', so code doesn't run

Complete Professional Structure

Template for All Your Scripts

```
#!/usr/bin/env python3
"""Grade Calculator - Calculate letter grades"""
import argparse
def calculate_grade(score):
  """Convert score to letter grade."""
  if score >= 90:
    return 'A'
  elif score >= 80:
    return 'B'
  else:
    return 'F'
def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--scores', nargs='+',
          type=int, required=True)
  args = parser.parse_args()
  for score in args.scores:
    grade = calculate_grade(score)
    print(f"Score {score}: {grade}")
```

🎯 **KEY COMPONENTS:**

✓ Docstring at top
✓ Imports
✓ Helper functions
✓ main() function with argparse
✓ if __name__ == '__main__':

Running Your Professional Script

Command-Line Usage Examples

BASIC USAGE

```
$ python grades.py --scores 85 92 78 88
Grade Report
====================================
Students: 4
Average: 85.8
Range: 78 - 92
```

WITH VERBOSE FLAG

```
$ python grades.py --scores 85 92 78 88 --verbose
Grade Report
====================================
Students: 4
Average: 85.8
Range: 78 - 92
Student 1: B
Student 2: A
Student 3: C
Student 4: B
```

⚡ Your scripts can now be run like professional tools!

Production-Ready Script Checklist

What Makes Code Professional?

✅ STRUCTURE

- ✅ Docstring at top
- ✅ Imports at top
- ✅ Functions for reusable logic
- ✅ main() function with argparse
- ✅ if __name__ == '__main__':

✅ QUALITY

- ✅ Descriptive function/variable names
- ✅ Comments for complex logic
- ✅ F-strings for output
- ✅ Input validation
- ✅ Helpful error messages

✅ FUNCTIONALITY

- ✅ Works from command line
- ✅ Accepts arguments
- ✅ Provides feedback
- ✅ Handles basic errors

Project 1: This Week's Goals

Final Week Before Presentations!

✅ MUST-HAVES

- Organizes files by type
- Uses functions
- Uses dictionaries for categories
- Clear variable names
- README with usage instructions

⭐ NICE-TO-HAVES (Pick 1-2)

- Command-line arguments
- Dry-run mode (preview only)
- Statistics report
- Custom categories from config file
- Interactive mode

📅 NEXT WEEK:

- 2-minute live presentations
- Code submission to GitHub
- Show it working!

Block 2 Recap

What You Learned

🎯 KEY SKILLS MASTERED:

1. User Input

- input() for interactive programs
- Input validation loops

2. argparse

- Command-line arguments
- Flags, types, choices

3. String Operations

- .lower(), .strip(), .split()
- .endswith(), .startswith()

4. F-Strings

- Beautiful formatted output
- Number formatting

🏗️ SCRIPT STRUCTURE:

```
1. Docstring
2. Imports
3. Helper functions
4. main() function
5. if __name__ == '__main__':
```

🚀 READY FOR BLOCK 3!

Advanced File Operations & Integration

BREAK – 20m

File Operations Review + New Tools

Block 3: Working with File Systems

📁 What You Already Know:

```
with open('file.txt', 'r') as f:
    content = f.read()
```

🚀 Today: Working with File SYSTEMS

```
import os
import shutil

files = os.listdir('Downloads/')  # List all files
os.makedirs('Images/', exist_ok=True)  # Create folder
shutil.move('a.jpg', 'Images/a.jpg')  # Move file
```

💡 Not just reading files - controlling entire directories!

The os Module Essentials

Block 3: Navigating File Systems

📋 Listing and Checking:

```
files = os.listdir('.')  # Current directory
exists = os.path.exists('file.txt')
is_file = os.path.isfile('data.csv')
is_dir = os.path.isdir('folder/')
```

🛤️ Path Operations:

```
# Join paths (cross-platform!)
path = os.path.join('folder', 'subfolder', 'file.txt')

# Split paths
name, ext = os.path.splitext('photo.jpg')
# ('photo', '.jpg')

folder, file = os.path.split('/docs/file.txt')
# ('/docs', 'file.txt')
```

📁 Creating:

```
os.makedirs('a/b/c',
  exist_ok=True)
```

📊 File Info:

```
size = os.path.getsize(
  'file.txt')
modified = os.path.getmtime(
  'file.txt')
```

Moving and Copying Files

Block 3: shutil - File Operations

```
import shutil

# Move (rename or relocate)
shutil.move('old.txt', 'new.txt')
shutil.move('file.txt', 'folder/file.txt')

# Copy
shutil.copy('source.txt', 'dest.txt')
shutil.copytree('folder/', 'backup/')  # Copy entire directory

# Delete (careful!)
os.remove('file.txt')  # Delete file
shutil.rmtree('folder/')  # Delete directory
```

✅ **Safe Pattern:**

```
if os.path.exists(source):
    if not os.path.exists(dest_folder):
        os.makedirs(dest_folder)
    shutil.move(source, dest)
```

⚠ **Always check before deleting!**

Cross-Platform Paths

Block 3: Windows vs Mac/Linux

⚠️ **The Problem:**

- Windows: `C:\Users\Alice\file.txt`
- Mac/Linux: `/Users/Alice/file.txt`

❌ **Don't Do:**

```
# Breaks on Windows
path = 'folder/file.txt'

# Breaks on Mac
path = 'folder\\file.txt'
```

✅ **Always Do:**

```
path = os.path.join(
    'folder',
    'subfolder',
    'file.txt'
)
# Works everywhere!
```

💡 **os.path.join() = Write once, run anywhere!**

Building a File Analyzer

Block 3: Complete Example (Part 1)

🔧 Helper Functions:

```
def get_extension(filename):
    """Get normalized file extension."""
    return os.path.splitext(filename)[1][1:].lower()

def format_size(bytes):
    """Convert bytes to human-readable format."""
    for unit in ['B', 'KB', 'MB', 'GB']:
        if bytes < 1024:
            return f"{bytes:.1f} {unit}"
        bytes /= 1024
```

📊 Example Usage:

`get_extension("photo.JPG")`
→ "jpg"

`format_size(1536000)`
→ "1.5 MB"

File Analyzer: Main Logic

Block 3: Complete Example (Part 2)

```
def analyze_directory(path):
    """Return statistics about directory contents."""
    if not os.path.exists(path):
        return None

    stats = {
        'total_files': 0,
        'total_size': 0,
        'extensions': {}
    }

    for item in os.listdir(path):
        full_path = os.path.join(path, item)

        if os.path.isfile(full_path):
            stats['total_files'] += 1
            stats['total_size'] += os.path.getsize(full_path)

            ext = get_extension(item) or 'no extension'
            stats['extensions'][ext] = \
                stats['extensions'].get(ext, 0) + 1

    return stats
```

🎯 **Returns dictionary with complete analysis!**

File Analyzer: Beautiful Output

Block 3: Complete Example (Part 3)

```
def main():
    parser = argparse.ArgumentParser(
        description='Analyze directory'
    )
    parser.add_argument('--path', default='.')
    args = parser.parse_args()

    stats = analyze_directory(args.path)

    print(f"\nAnalyzing: {args.path}")
    print("=" * 50)
    print(f"Total files: {stats['total_files']}")
    print(f"Total size: {format_size(stats['total_size'])}")
    print(f"\nFile types:")

    for ext, count in sorted(stats['extensions'].items()):
        print(f"  {ext:15} : {count:3} files")
```

Running:

```
$ python analyzer.py --path Downloads/
```

```
Analyzing: Downloads/
==================================================
Total files: 47
Total size: 234.5 MB
```

File Organizer: Project 1 Pattern

Block 3: Bringing It All Together

📁 Category Mapping:

```
def get_category(extension):
    """Map extension to category folder."""
    categories = {
        'jpg': 'Images', 'png': 'Images',
        'pdf': 'Documents', 'txt': 'Documents',
        'mp3': 'Music'
    }
    return categories.get(extension.lower(), 'Other')
```

photo.JPG
→ Images/

report.PDF
→ Documents/

song.mp3
→ Music/

File Organizer: Main Logic

Block 3: The organize_files Function

```
def organize_files(source, dest, dry_run=False):
    """Organize files by type."""
    # Create category folders
    for category in ['Images', 'Documents', 'Music', 'Other']:
        os.makedirs(os.path.join(dest, category), exist_ok=True)

    # Process files
    count = 0
    for filename in os.listdir(source):
        filepath = os.path.join(source, filename)

        if not os.path.isfile(filepath):
            continue

        ext = os.path.splitext(filename)[1][1:]
        category = get_category(ext)
        dest_path = os.path.join(dest, category, filename)

        if dry_run:
            print(f"Would move: {filename} → {category}/")
        else:
            shutil.move(filepath, dest_path)
            print(f"✓ {filename} → {category}/")
            count += 1
```

🎯 dry_run parameter = safe testing!

Error Handling Preview

Block 3: What Could Go Wrong?

⚠️ Common Errors:

```
# FileNotFoundError
shutil.move('missing.txt', 'dest/')

# PermissionError
os.remove('/system/protected.txt')

# FileExistsError
shutil.move('file.txt', 'existing_file.txt')
```

✅ Basic Handling:

```
try:
    shutil.move(source, dest)
    print(f"✓ Moved {filename}")
except FileNotFoundError:
    print(f"✗ File not found: {filename}")
except PermissionError:
    print(f"✗ No permission: {filename}")
except Exception as e:
    print(f"✗ Error: {e}")
```

📝 **Note: Full error handling next week!**

Testing Your Scripts

Block 3: Safe Testing Strategy

❌ Don't:

- ✕ Test on real files first
- ✕ Test on important directories
- ✕ Skip dry-run testing

✅ Do:

1. Create test directory
2. Test with --dry-run
3. Check preview
4. Run on test directory
5. Verify results
6. NOW use on real files

📁 Sample Test Structure:

```
test/
  ├── photo.jpg
  ├── song.mp3
  └── report.pdf
```

```
$ python organizer.py --source test/ --dest output/ --dry-run
```

Best Practices Checklist

Block 3: Production-Ready Scripts

✅ Structure:

- ☑ Docstring at top
- ☑ Imports at top
- ☑ Functions for reusable logic
- ☑ main() function with argparse
- ☑ if __name__ == '__main__':

✅ Quality:

- ☑ Descriptive names
- ☑ Comments for complex logic
- ☑ os.path.join() for paths
- ☑ Input validation
- ☑ Helpful error messages
- ☑ F-strings for output

✅ Functionality:

- ☑ Works from command line
- ☑ Accepts arguments
- ☑ Provides feedback
- ☑ Handles basic errors

🏆 Professional code = Clean + Safe + User-friendly

Week 3 Complete! 🎉

Block 3: What You Learned Today

📚 Block 1: Functions & Data

- Functions (reusable code)
- Tuples (immutable)
- Dictionaries (key-value)
- Multiple returns

🎯 Block 2: Complete Scripts

- input() and argparse
- String operations
- F-strings formatting
- __name__ pattern

🗂️ Block 3: File Systems

- os module operations
- Cross-platform paths
- shutil move/copy
- Complete examples

🚀 Next Week:

Project 1 Presentations!