PRACTICAL COURSE ON

Python with Statistics

MORNING SESSION HANDBOOK

Foundation Tools for Data Analysis

Follow-Along Workbook
Complete with Examples, Exercises & Solutions

Morning Session Overview

Welcome! This handbook guides you step-by-step through the morning session. Each section includes:

- 📝 Concepts explained in plain language
- 💻 Copy-paste ready code examples
- ▶️ Expected outputs so you know it's working
- ✓ Checkpoint exercises to test your understanding
- 💡 Tips to avoid common mistakes

Session Schedule

Time	Topic
09:00-09:30	Setup & Introduction
09:30-11:00	Module 1: Python Fundamentals
11:00-11:15	☕ Morning Break
11:15-12:45	Module 2: Data Wrangling with Pandas
12:45-13:30	🍽️ Lunch Break

Part 1: Setup & Introduction

⏰ Time: 09:00 - 09:30 (30 minutes)

Why Python for Data Analysis?

Before we dive into code, let's understand why Python has become the preferred tool for data analysis and statistics:

- **Automation:** Write code once, run it on any size dataset. No manual copying.
- **Reproducibility:** Your analysis is a script. Anyone can run it and get the same results.
- **Integration:** Combine data cleaning, statistics, visualization, and reporting in one place.
- **Free & Open Source:** No expensive licenses. Works on any computer.
- **Huge Community:** Millions of users. Any problem you face, someone has solved it.

Setting Up Google Colab

Google Colab is a free, cloud-based Python environment. No installation needed—just open your browser and start coding.

Step-by-Step Setup

1. Open your web browser (Chrome, Firefox, Safari, or Edge)

2. Go to: https://colab.research.google.com

3. Sign in with your Google account

4. Click the orange 'New Notebook' button (top left)

5. You'll see a blank notebook with a code cell

> 💡 **TIP:** Bookmark https://colab.research.google.com for easy access. You'll use it daily!

Understanding the Colab Interface

A Colab notebook contains cells. Each cell is a container for code or text.

- **Code cells:** Where you write Python commands (gray background)
- **Text cells:** Where you write notes or explanations (white background)
- **Play button (▶️):** Click to run the code in a cell
- **+ Code button:** Add a new code cell
- **+ Text button:** Add a new text cell

How to Run Code

Two ways to run code in a cell:

- Press Shift + Enter → Runs code and moves to next cell

- Press Ctrl + Enter (Windows) or Cmd + Enter (Mac) → Runs code, stays in same cell

Your First Python Program

Every programmer starts with 'Hello World.' This simple program confirms everything is working correctly.

📋 **Type this code:**

```
print('Hello World')
```

Output:

```
Hello World
```

If you see 'Hello World' appear below your code, congratulations! You just ran your first Python program.

💡 **TIP:** The print() function displays output. You'll use it constantly to check your work.

Testing Different Outputs

Try these variations to get comfortable:

Example 1: Print your name

```
print('My name is [Your Name]')
```

Example 2: Print a calculation

```
print(2 + 2)
```

Output:

```
4
```

Example 3: Print multiple lines

```
print('Python is fun!')
print('Statistics is powerful!')
```

Output:

```
Python is fun!
Statistics is powerful!
```

✓ CHECKPOINT EXERCISE 1

Print your name and your favorite number on separate lines.

💡 *Hint:* Use two print() statements, one for each line.

Common Setup Problems & Solutions

Problem	Solution
Problem: Code doesn't run when I click Play	Solution: Wait 10-15 seconds. Colab needs to connect to a server (you'll see 'Connecting...' at top-right).
Problem: I see 'SyntaxError'	Solution: Check your quotes and parentheses. They must match exactly: print('text')

Problem: Nothing appears after running code	Solution: Did you use print()? Without it, Python calculates but doesn't display.
Problem: Notebook freezes	Solution: Click 'Runtime' → 'Restart runtime' from the top menu.

Part 2: Module 1 - Python Fundamentals

⏰ Time: 09:30 - 11:00 (90 minutes)

1.1 Variables — Naming Your Data

What Is a Variable?

A variable is a labeled container that stores a value. Think of it like a box with a label on it:

- You put a value inside the box (like the number 250)
- You write a label on the box (like 'num_students')
- Whenever you say 'num_students', Python knows you mean 250

Creating Your First Variable

📋 **Example:**

```
num_students = 250
print(num_students)
```

Output:

```
250
```

Breakdown:

- `num_students` — The variable name (the label on your box)
- `=` — Assignment operator (puts the value in the box)
- `250` — The value being stored
- `print()` — Shows us what's inside the variable

Why Variables Matter for Statistics

Compare these two approaches:

❌ **Without variables (unclear):**

```
print(0.72 * 100)
```

Output:

```
72.0
```

What does 0.72 mean? What are we calculating? Unclear.

✅ **With variables (clear):**

```
pass_rate = 0.72
percentage = pass_rate * 100
print(percentage)
```

Output:

```
72.0
```

Now it's obvious: we're converting a pass rate to a percentage. Variables make your code self-documenting.

Python Data Types

Python recognizes different types of data. The three most important:

1. Numbers

Integers (int) — Whole numbers with no decimal point

📋 **Examples:**

```
sample_size = 100
num_students = 250
age = 25
```

Floats (float) — Decimal numbers

📋 **Examples:**

```
pass_rate = 0.72
mean_score = 78.5
p_value = 0.03
```

💡 **TIP:** Python automatically knows if a number is int or float based on the decimal point. 100 is int, 100.0 is float.

2. Text (Strings)

Strings (str) — Text data enclosed in quotes

📋 **Examples:**

```
country = 'Malaysia'
course_name = 'Applied Statistics'
group = "Control"  # double quotes work too
```

⚠️ **COMMON MISTAKE:** Forgetting quotes around text. Writing country = Malaysia (without quotes) causes an error. Python thinks Malaysia is a variable name, not text.

3. True/False (Booleans)

Booleans (bool) — True or False values (note: capitalized!)

📋 **Examples:**

```
is_significant = True
survey_done = False
passed_test = True
```

💡 **TIP:** True and False must be capitalized. true or false (lowercase) will cause errors.

Multiple Variables at Once

📋 **Complete example:**

```
# Creating several variables
num_students = 250
pass_rate = 0.72
course_name = 'Applied Statistics'
is_significant = True

# Printing them
print(num_students)
print(pass_rate)
print(course_name)
print(is_significant)
```

Output:

```
250
0.72
Applied Statistics
True
```

Variable Naming Rules

Follow these rules to avoid errors:

- ✓ Must start with a letter or underscore: age, _temp, student_name
- ✗ Cannot start with a number: 1st_name ❌
- ✓ Can contain letters, numbers, and underscores: score_2023, final_grade
- ✗ Cannot contain spaces: student name ❌ → use student_name ✓
- ✓ Case-sensitive: age and Age are DIFFERENT variables
- ✗ Cannot use Python keywords: print, if, for, class ❌

Good vs Bad Variable Names

❌ Bad	✅ Good	Why?
x	num_students	Descriptive names explain what the variable represents
var1	pass_rate	Meaningful names prevent confusion
data	survey_responses	Be specific
tmp	temp_celsius	Even temporary variables should be clear

Calculations with Variables

📋 **Example: Calculate percentage**

```
total_students = 120
passed = 87

pass_percentage = (passed / total_students) * 100
print(pass_percentage)
```

Output:

```
72.5
```

📋 Example: Calculate average

```
score1 = 85
score2 = 92
score3 = 78

average = (score1 + score2 + score3) / 3
print(average)
```

Output:

```
85.0
```

Updating Variables

📋 Variables can change:

```
score = 75
print(score)

score = 82  # Changed the value
print(score)
```

Output:

```
75
82
```

✓ CHECKPOINT EXERCISE 2

Create variables for:
• Sample size (n = 120)
• Mean score (mean = 78.3)
• Standard deviation (std = 12.5)

Then print all three values.
💡 *Hint:* Create each variable with = and print each one.

✓ CHECKPOINT EXERCISE 3

Calculate the percentage of students who scored above 80, given:
• Total students = 150
• Students above 80 = 45

Store the result in a variable and print it.
💡 *Hint:* percentage = (students_above_80 / total_students) * 100

1.2 Lists — Storing Multiple Values

What Is a List?

A list is a container that holds multiple values in order. Instead of creating score1, score2, score3, you create one list called scores that holds all of them.

📋 **Creating a list:**

```
scores = [78, 55, 82, 90, 46, 73, 88, 61]
```

Breakdown:

- `scores` — Variable name
- `[ ]` — Square brackets indicate a list
- 78, 55, 82, ... — Values separated by commas

Accessing List Items

Python numbers list positions starting from 0 (not 1!):

📋 **Example:**

```
scores = [78, 55, 82, 90, 46, 73, 88, 61]

print(scores[0])    # First item
print(scores[3])    # Fourth item
print(scores[-1])   # Last item
```

Output:

```
78
90
61
```

Index	Meaning
0	First item
-1	Last item
3	Fourth item

⚠ **COMMON MISTAKE:** Trying to access scores[1] to get the first item. Python starts counting from 0, so scores[1] gives you the SECOND item!

List Length

📋 **How many items in the list?**

```
scores = [78, 55, 82, 90, 46, 73, 88, 61]
print(len(scores))
```

Output:

```
8
```

Basic List Calculations

📋 **Sum and average:**

```
scores = [78, 55, 82, 90, 46, 73, 88, 61]

total = sum(scores)
count = len(scores)
average = total / count

print('Total:', total)
print('Count:', count)
print('Average:', average)
```

Output:

```
Total: 573
Count: 8
Average: 71.625
```

Adding Items to a List

Adding with .append():

```
scores = [78, 55, 82]
print(scores)

scores.append(95)  # Add 95 to the end
print(scores)
```

Output:

```
[78, 55, 82]
[78, 55, 82, 95]
```

Removing Items from a List

Removing with .remove():

```
scores = [78, 55, 82, 90, 46]
print(scores)

scores.remove(46)  # Remove the value 46
print(scores)
```

Output:

```
[78, 55, 82, 90, 46]
[78, 55, 82, 90]
```

List Slicing

Getting a portion:

```
scores = [78, 55, 82, 90, 46, 73, 88, 61]

first_three = scores[0:3]  # Items 0, 1, 2
last_two = scores[-2:]     # Last 2 items

print('First three:', first_three)
print('Last two:', last_two)
```

Output:

```
First three: [78, 55, 82]
```

```
Last two: [88, 61]
```

💡 **TIP:** Slicing uses [start:end] where end is NOT included. scores[0:3] gives items 0, 1, 2 (not 3).

Lists with Different Data Types

📋 **Not just numbers:**

```
# List of strings
students = ['Ali', 'Siti', 'Raj', 'Lin']
print(students)

# List of mixed types
student_info = ['Ali', 25, True, 85.5]
print(student_info)
```

Output:

```
['Ali', 'Siti', 'Raj', 'Lin']
['Ali', 25, True, 85.5]
```

✓ CHECKPOINT EXERCISE 4

Create a list of 5 test scores: [65, 78, 82, 90, 55]
Then:
1. Print the first score
2. Print the last score
3. Calculate and print the average

💡 *Hint:* Use scores[0] for first, scores[-1] for last, sum(scores)/len(scores) for average

✓ CHECKPOINT EXERCISE 5

Create a list with 3 student names: ['Ahmad', 'Siti', 'Raj']
Add a fourth name 'Lin' to the list
Print the updated list

💡 *Hint:* Use .append() to add the name

1.3 Loops — Automating Repetition

Why Loops Matter

Imagine you have 100 student scores and need to print each one. Without loops, you'd write 100 print() statements. With loops, you write 2 lines of code.

The for Loop

📋 **Basic for loop:**

```
scores = [78, 55, 82, 90, 46]

for score in scores:
    print(score)
```

Output:

```
78
55
82
90
46
```

How it works:

1. Python takes the first item from the list (78) and stores it in the variable 'score'
2. It runs the indented code (print(score))
3. It repeats for each item in the list

⚠️ **COMMON MISTAKE:** Forgetting the colon (:) at the end of the for line. Python requires it!

⚠️ **COMMON MISTAKE:** Not indenting the code inside the loop. Python uses indentation (4 spaces or Tab) to know what's inside the loop.

Adding Custom Messages

📋 **Loop with text:**

```
scores = [78, 55, 82, 90, 46]

for score in scores:
    print('Student scored:', score)
```

Output:

```
Student scored: 78
Student scored: 55
Student scored: 82
Student scored: 90
Student scored: 46
```

Calculations Inside Loops

📋 **Modify each item:**

```
scores = [78, 55, 82, 90, 46]
```

```
for score in scores:
    grade = score + 5  # Add 5 bonus points
    print('Original:', score, '→ With bonus:', grade)
```

Output:

```
Original: 78 → With bonus: 83
Original: 55 → With bonus: 60
Original: 82 → With bonus: 87
Original: 90 → With bonus: 95
Original: 46 → With bonus: 51
```

Conditional Logic in Loops

📋 **Pass/Fail classification:**

```
scores = [78, 55, 82, 90, 46, 73]

for score in scores:
    if score >= 60:
        print(score, '→ Pass')
    else:
        print(score, '→ Fail')
```

Output:

```
78 → Pass
55 → Fail
82 → Pass
90 → Pass
46 → Fail
73 → Pass
```

Breakdown of if/else:

- `if score >= 60:` — Check if score is 60 or higher
- `    print(...)` — If True, run this (notice the indent)
- `else:` — Otherwise...
- `    print(...)` — Run this instead

List Comprehension (Advanced but Useful)

List comprehension is a compact way to transform lists. It's like a loop and a list creator combined into one line.

📋 **Transform all items:**

```
scores = [78, 55, 82, 90, 46, 73]

# Add 5 bonus points to every score
adjusted = [s + 5 for s in scores]
print(adjusted)
```

Output:

```
[83, 60, 87, 95, 51, 78]
```

📋 **Filter items:**

```
scores = [78, 55, 82, 90, 46, 73]

# Keep only passing scores (>= 60)
passed = [s for s in scores if s >= 60]
print(passed)
```

Output:

```
[78, 82, 90, 73]
```

💡 **TIP:** List comprehension is optional but powerful. If it feels confusing now, use regular for loops. You can learn comprehensions later.

✓ CHECKPOINT EXERCISE 6

You have these temperatures in Celsius: [20, 25, 30, 15, 28]

Write a loop that prints each temperature with the message:
'Temperature: [X]°C'

Example output: Temperature: 20°C

💡 *Hint:* Use a for loop with print('Temperature:', temp, '°C')

✓ CHECKPOINT EXERCISE 7

You have these test scores: [45, 78, 82, 55, 90, 38]

Create a new list containing only scores that are 60 or above.
Print the filtered list.

💡 *Hint:* Use list comprehension: [s for s in scores if s >= 60] or a regular for loop with if

1.4 Functions — Reusable Code Blocks

What Is a Function?

A function is a named block of code that performs a specific task. Instead of writing the same code multiple times, you write it once as a function and call it whenever needed.

Think of it like a recipe: You write the recipe once, then follow it every time you want to make that dish.

Creating a Simple Function

📋 **Basic function:**

```
def greet():
    print('Hello, welcome to Python!')

# Now use (call) the function
greet()
```

Output:

```
Hello, welcome to Python!
```

Breakdown:

- `def` — Keyword that starts a function definition
- `greet` — Function name (you choose this)
- `():` — Parentheses and colon (required syntax)
- `    print(...)` — Function body (indented code)
- `greet()` — Calling (running) the function

Functions with Parameters

Parameters let you pass information into a function. Like giving ingredients to a recipe.

📋 **Function with parameter:**

```
def greet_student(name):
    print('Hello,', name, '!')

greet_student('Ali')
greet_student('Siti')
greet_student('Raj')
```

Output:

```
Hello, Ali !
Hello, Siti !
Hello, Raj !
```

Functions with Multiple Parameters

📋 **Two parameters:**

```
def calculate_percentage(passed, total):
```

```
    percentage = (passed / total) * 100
    print('Pass rate:', percentage, '%')

calculate_percentage(87, 120)
calculate_percentage(45, 50)
```

Output:

```
Pass rate: 72.5 %
Pass rate: 90.0 %
```

Functions that Return Values

Instead of printing inside the function, you can return a value to use elsewhere.

📋 **Function with return:**

```
def calculate_average(scores):
    total = sum(scores)
    count = len(scores)
    average = total / count
    return average  # Send the result back

# Use the returned value
scores1 = [78, 82, 90]
scores2 = [65, 70, 75]

avg1 = calculate_average(scores1)
avg2 = calculate_average(scores2)

print('Class A average:', avg1)
print('Class B average:', avg2)
```

Output:

```
Class A average: 83.33333333333333
Class B average: 70.0
```

💡 **TIP:** return sends a value back to whoever called the function. You can store it in a variable or use it in calculations.

Practical Example: Grade Classification

📋 **Complete example:**

```
def classify_grade(score):
    if score >= 90:
        return 'A'
    elif score >= 80:
        return 'B'
    elif score >= 70:
        return 'C'
    elif score >= 60:
        return 'D'
    else:
        return 'F'

# Use it on one score
print(classify_grade(85))
```

```
# Use it on a list of scores
scores = [78, 55, 82, 90, 46, 73, 88, 61]
for score in scores:
    grade = classify_grade(score)
    print('Score:', score, '→ Grade:', grade)
```

Output:

```
B
Score: 78 → Grade: C
Score: 55 → Grade: F
Score: 82 → Grade: B
Score: 90 → Grade: A
Score: 46 → Grade: F
Score: 73 → Grade: C
Score: 88 → Grade: B
Score: 61 → Grade: D
```

Why Functions Are Powerful

- **Reusability:** Write once, use many times
- **Organization:** Break complex problems into smaller pieces
- **Testing:** Test each function independently
- **Readability:** classify_grade(85) is clearer than nested if statements

✓ CHECKPOINT EXERCISE 8

Create a function called celsius_to_fahrenheit that:
• Takes one parameter: celsius temperature
• Converts it to Fahrenheit using: (celsius × 9/5) + 32
• Returns the Fahrenheit value

Then test it with: 0, 25, 100
💡 *Hint:* def celsius_to_fahrenheit(celsius):
return (celsius * 9/5) + 32

✓ CHECKPOINT EXERCISE 9

Create a function called count_passing that:
• Takes a list of scores as parameter
• Counts how many scores are 60 or above
• Returns the count

Test with: [45, 78, 82, 55, 90, 38, 67]
💡 *Hint:* Use a for loop with if statement to count, or list comprehension