Session 1.1: Welcome to Programming!

Your Journey from Zero to ML-Ready Starts Here

Lance Muwayi

Session 1.1: Welcome to Programming!

Your Journey from Zero to ML-Ready Starts Here

Lance Muwayi

Today's Journey

- Part 1: What is Programming? Your First Program
- Part 2: Workshop - Create Your First Programs
- Break - Celebrate Your Success!
- Part 3: Saving Your Work with Git
- Part 4: Workshop - Practice Git
- Part 5: Quiz & Next Steps

Part 1: Welcome to Programming

You're about to become a programmer!

Your Journey Starts Here 🌟

- You're making a career change - that takes courage!
- No programming experience? Perfect! We start from zero
- By Week 8, you'll build real data analysis programs
- Next: AWS Machine Learning University (MLU) Program (you'll be ready!)
- Success story: Many career changers become top ML engineers
- Today: Write your first program in 15 minutes
- YOU CAN DO THIS! 💪

What is Programming? (Simple Explanation)

- Programming = Giving instructions to computers
- Like writing a recipe, but for computers
- Computers are very literal (do exactly what you say)
- Real examples you use daily:
  - Excel formulas = programming!
  - Email filters = programming!
  - GPS navigation = programming!
- You already think like a programmer - now learn the language!

Why Python? (Perfect for Beginners)

- Easiest programming language to learn
- Reads almost like English
- Most popular for data science and ML
- Huge job market (Python developers in high demand)
- Immediate results - see your code work right away!
- Companies using Python:
    - Google, Netflix, NASA, Instagram, Spotify, etc

Your 8-Week Roadmap

- Week 1-2: Learn the basics (variables, text, numbers)
  - Goal: Get comfortable, remove fear
- Week 3-4: Make decisions & repeat tasks (core skills)
  - Goal: Write useful programs
- Week 5-6: Organize code & work with files
  - Goal: Process real data
- Week 7-8: ML preparation & final project
  - Goal: Ready for ML program! 🎓

Installing Python - Step by Step

1. Go to python.org/downloads
2. Download Python 3.10 or newer
3. Windows: CHECK 'Add Python to PATH' ✓ (IMPORTANT!)
    - Mac: Python may already be installed
4. Install with default settings
5. Test: Open Terminal/Command Prompt
    - Type: python –version
    - See: Python 3.x.x = SUCCESS! ✓
    - (We'll help you during workshop if any issues)

Your First Program - Hello World!

- Every programmer starts here
- It's a tradition (seriously!)
- This proves everything works
- Ready? Here's your first program:
- print("Hello, World!")
- That's it! You're officially a programmer! 🎉

Let's Break Down Your First Program

```
print("Hello, World!")

# What each part means:
# print() = built-in function (pre-made tool)
# 'Hello, World!' = text (called a 'string')
# Quotes tell Python: this is text

# Try changing it:
print("My name is Sarah")
print("I am learning Python")
print("This is exciting!")

# Run each one - see the output!
```

What Are Variables? (Box Analogy)

Variable = labeled box that holds information

Real life:

- Box labeled 'age' contains: 32
- Box labeled 'name' contains: Sarah
- Box labeled 'job' contains: Marketing Manager

In Python:

- age = 32
- name = 'Sarah'
- job = 'Marketing Manager'

Computer remembers these for you!

Creating Your First Variables

```
# About you (change these to your info!)
name = 'Sarah Johnson'
age = 32
previous_career = 'Marketing Manager'
wants_ml = True

# Display them
print(name)
print(age)
print(previous_career)

# Python remembers all of these!
# You can use them anytime in your program
```

Variable Names - The Rules

- ✓ CAN use:
  - Letters: name, age, user
  - Numbers (not first): age2, user1
  - Underscores: first_name, user_age
- ✗ CANNOT use:
  - Spaces: first name (use first_name)
  - Starting with number: 2nd_name
  - Special characters: name!, user@email
- Best practice: Use descriptive names
  - ✓ student_age (clear!)
  - ✗ x (confusing)

Data Types - What Can Variables Hold?

- Python has 4 basic types (memorize these!):

1. Text (strings):
   - 'Hello' or "Hello" - use quotes
2. Whole numbers (integers):
   - 42, 100, -5 - no quotes
3. Decimals (floats):
   - 3.14, 98.6, 0.001 - has decimal point
4. True/False (booleans):
   - True or False - no quotes, capital T/F

Data Types in Action

```
# String (text) - use quotes
name = 'Alice'
city = "Boston"

# Integer (whole number) - no quotes
age = 28
year = 2024

# Float (decimal) - has decimal point
temperature = 98.6
price = 19.99

# Boolean (True/False) - no quotes
is_student = True
has_experience = False
# Python figures out the type automatically!
```

Working with Numbers (You Already Know This!)

- Programming math = regular math
- Addition: +
- Subtraction: -
- Multiplication: * (not ×)
- Division: / (not ÷)
- Examples:
    - 10 + 5 = 15
    - 20 - 8 = 12
    - 3 * 4 = 12
    - 10 / 2 = 5

Calculator in Python

```
# Basic math
total = 10 + 5
print(total)  # Shows: 15

# Real-world example: monthly expenses
rent = 1500
food = 400
transport = 200
total_expenses = rent + food + transport
print('Monthly expenses:', total_expenses)

# Shows: Monthly expenses: 2100
# You just processed data! This is what ML does
# (but with millions of numbers)
```

Comments - Notes to Yourself/Others

- Comments = notes that Python ignores
- Use # to make a comment
- Why comments matter for beginners:
  - Explain what code does (for future you!)
  - Leave notes to yourself
  - Disable code temporarily
- Good comments explain WHY, not WHAT:
  - ✗ age = 25 # Set age to 25
  - ✓ age = 25 # Minimum age for program eligibility

Using Comments Effectively

```python
# This is a comment - Python skips it
# Calculate monthly budget
income = 5000
expenses = 3200
savings = income - expenses  # What's left over
print('Monthly savings:', savings)

# Comments help you remember:
# - What you were thinking
# - Why you made choices
# - How to use the code later

# Tip: Write comments like you're teaching someone!
```

Why This Matters for Your ML Journey

- Everything you learned today is used in ML:
- Variables → Store data, model parameters
- Numbers → ML is all math on numbers
- Text → Process labels, categories
- Math operations → Training algorithms
- Example ML workflow:
   1. Load data (variables)
   2. Process numbers (math)
   3. Train model (lots of calculations)
   4. Make predictions (more math)

You're learning the foundation! 🏗️

Part 2: Workshop

Write Your First Programs!

💻 Exercise 1: All About You

Create a file called: about_me.py

Your task:

1. Create variables for:
    - Your name (string)
    - Your age (integer)
    - Your previous career (string)
    - Are you excited to learn ML? (True/False)
2. Print each one with a label

Example output:

- Name: Sarah Johnson
- Age: 32
- Previous career: Marketing

💻 Exercise 2: Budget Calculator

Create a file called: budget.py

Your task:

1. Create variables for monthly expenses:
    - rent = your rent
    - food = your food budget
    - transport = your transport costs
    - entertainment = fun money
2. Calculate total expenses
3. If you have income, calculate savings
4. Print results nicely

This is data processing - what you'll do in ML!

💻 Exercise 3: ML Program Countdown

Create a file called: countdown.py

Your task:

1. Calculate days until ML program starts:
    - weeks_left = 8
    - days_per_week = 7

2. Calculate total days

3. Calculate hours (days × 24)

4. Print motivational message with calculations
    - Example: 'Only 56 days until ML program!'

Make it encouraging! 💪

☕ Celebrate!

You wrote real programs! Take a break

Part 3: Git - Saving Your Work

Professional skill for your resume

What is Git? (Non-Technical Explanation)

- Git = 'Save Game' button for your code
- Like Microsoft Word 'Track Changes' but better:
    - Save versions as you work
    - Go back to any previous version
    - See exactly what you changed
    - Work with others without conflicts
- Every professional programmer uses Git
- It's expected in job interviews
- You'll put 'Git' on your resume! 📄

Why You NEED Version Control

Real scenarios (will happen to you!):

- ❌ Without Git:
  - Code worked yesterday, broken today - can't undo
  - Accidentally deleted important code
  - Want to try new approach but might break everything
  - Lost track of what changed

- ✅ With Git:
  - 'Rewind' to any working version
  - Try risky changes safely
  - Always know what changed
  - Professional workflow

Installing Git (One-Time Setup)

Step 1: Download

- Go to: git-scm.com/downloads
- Download for your OS (Windows/Mac/Linux)

Step 2: Install

- Windows: Use all default settings
- Mac: May already be installed

Step 3: Verify

- Open Terminal/Command Prompt
- Type: git --version
- See version number = Success! ✓

(We'll help during workshop if needed)

Configure Git (Tell It Who You Are)

```
# Do this ONCE (after installing)
git config --global user.name "Your Full Name"
git config --global user.email "your.email@example.com"

# Example:
git config --global user.name "Sarah Johnson"
git config --global user.email "sarah.j@email.com"

# Why?
# - Git tracks WHO made changes
# - Important for team projects
# - Shows up in your commit history
```

Understanding Git Workflow (3 Simple Steps)

Think of it like this:

1. Working Directory = Your desk
   - Where you write code
2. Staging Area = Your bag
   - Choose what to save
3. Repository = Your file cabinet
   - Permanent saved versions

Process: Work on desk → Put in bag → File away

In Git: Edit code → git add → git commit

Your First Git Repository

```
# Step 1: Create project folder
mkdir python_practice # Make directory. Press Enter.

cd python_practice # 'cd' means 'change directory' - it moves you into the folder
you just created. Press Enter.

# Quick check: Where am I now?
pwd # → Should show something like: /c/Users/YourName/python_practice (forward
slashes on Git Bash)

# Step 2: Initialize Git
git init
# You should see a message like: 'Initialized empty Git repository in
/path/to/python_practice/.git/'

# Output: 'Initialized empty Git repository'
# That's it! This folder is now tracked by Git

# What happened?
# - Git created hidden .git folder
# - Stores all your save history
# - DON'T delete .git folder!
```

Saving Your Work (The Git Cycle)

```
# Saving Your Work - The Git Cycle (Hello World Example)
echo "print('Hello world')" > hello.py # Use double quotes around the whole thing — safest on
Windows/Git Bash

# Quick check: Does the file exist and contain the right code?
cat hello.py # Should show: print('Hello world') # (If nothing shows → the file wasn't created —
try again)

# Step 1: Always check status first (very good habit!)
git status # → You should see: hello.py (under "Untracked files")

# Step 2: Stage the file (tell Git "watch this file")
git add hello.py

# Step 3: Commit (actually save the snapshot forever!)
git commit -m "Add my first Python program"
# You should see something like:
# [main (root-commit) abc1234] Add my first Python program # 1 file changed, 1 insertion(+)
# create mode 100644 hello.py
# Bonus: See your beautiful commit history
git log --oneline
# Shows: abc1234 Add my first Python program
# Now run your program (use winpty on Windows/Git Bash!)
winpty python hello.py # → Output: Hello world # (If output doesn't show instantly, try: python -
u hello.py)
```

git status - Your Best Friend

ALWAYS use 'git status' before doing anything

It tells you:

- What files changed
- What's staged (ready to commit)
- What's not staged
- What's untracked (new files)

Think of it as:

- 'What's on my desk right now?'

Use it constantly - it guides you!

Commit Messages - Tell a Story

Each commit needs a message

Message = short description of what you did

Good messages:

- ✓ 'Add budget calculator'
- ✓ 'Fix bug in expense calculation'
- ✓ 'Complete Exercise 2 from Session 1'

Bad messages:

- ✗ 'update'
- ✗ 'stuff'
- ✗ 'changes'

Future you will thank you for good messages!

Viewing Your History

```
# See all your commits
git log

# Shows:
# - Commit ID (unique identifier)
# - Author and date
# - Your message

# Easier to read version:
git log --oneline

# Shows:
# a1b2c3d Add budget calculator
# e4f5g6h Add hello world program
# This is your progress timeline!
```

Git Best Practices for Beginners

1. Commit after each working feature
   - (not after every line - after it WORKS)
2. Use 'git status' before every command
   - (seriously, every time)
3. Write clear commit messages
   - (your future self will thank you)
4. Commit before trying risky changes
   - (creates restore point)
5. Don't be afraid to experiment
   - (Git protects you!)

Part 4: Workshop

Git Mastery Practice

💻 Exercise 4: Complete Git Workflow

Let's save all your work from today!

Step 1: Create project folder (5 min)

- Make folder: python_week1
- Navigate into it
- Run: git init

Step 2: Save your programs (10 min)

- Put about_me.py in folder
- git add about_me.py
- git commit -m 'Add about me program'
- Repeat for budget.py and countdown.py

Step 3: Make a change (5 min)

- Edit one file (add comment)
- git status (see what changed)
- git add, git commit
- git log --oneline (see your history!)

Part 5: Quiz & Wrap-up

Celebrate Your Progress!

📝 Session 1.1 Quiz

🎉 YOU DID IT! Session 1.1 Complete

- What you accomplished today:

✓ Installed Python (you're set up!)

✓ Wrote your first programs (you're a programmer!)

✓ Created and used variables (storing data!)

✓ Learned Git (professional skill!)

✓ Saved your work (protected forever!)

From zero to programmer in 3 hours!

This is just the beginning! 🚀

Homework (Light for First Session)

1. Complete the quiz (10 questions - online)
2. Practice what you learned:
   - Create 3 more programs about different topics
   - Use variables and print()
   - Save each with Git (practice the workflow)
3. Watch (if time): 'What is Programming?' video
   - Link in course materials
4. Reflect: Write 1 paragraph about:
   - 'Why I want to learn machine learning'

Remember: Practice makes progress! 30 min/day is perfect

Next Session: Working with Text

Session 1.2 (Next class):

You'll learn:

- String operations (cutting, combining text)
- Text processing (clean messy data)
- F-strings (professional formatting)
- More Git practice (build muscle memory)

Why it matters:

- Text is everywhere in data
- ML often processes text (NLP)
- Real-world data is messy - you'll clean it!

Homework prepares you - please do it! 📚

Encouragement….& Support

Remember:

- ✨ Every expert was once a beginner
- ✨ You did amazing today!
- ✨ Making mistakes = learning (it's good!)
- ✨ Ask questions (no 'dumb' questions)
- ✨ I am here to help you succeed

Resources:

- Course forum (ask anything)
- Office hours (get help)
- Study groups (learn together)

See you next session! You've got this! 💪