Minor in AI

Revising the Basics

1 Introduction: The Retail Shop Case Study

Imagine you own a clothing store and want to understand your customers better. Some visit frequently and give high ratings, while others rarely visit. How would you categorize them? This is called **customer segmentation** - grouping customers based on their behavior.

In our session, we used this real-world scenario to learn Python programming. We'll build a program that:

1. Asks customers about their shopping experience
2. Collects their visit frequency
3. Categorizes them as *loyal*, *regular*, or *occasional*
4. Provides personalized feedback

Through this case study, we'll learn fundamental programming concepts that form the foundation of data science and AI.

2 Core Programming Concepts

2.1 Variables: Your Program's Memory

Variables are like labeled boxes where we store information. In our retail example:

- `user_name` stores the customer's name
- `score` stores satisfaction rating (1-10)
- `visits` stores number of shop visits

Why meaningful names matter

Good: `customer_rating`
Bad: `cr` or `x`
Clear names make code self-documenting and easier to understand.

2.2 Input/Output: Talking to Your Program

We use `input()` to get data from users and `print()` to show results:

```
# Get customer information
user_name = input("Enter your name: ")
rating = input("Rate your experience (1-10): ")
visits = input("How many visits this month? ")
```

Explanation:

- The `input()` function displays a message (prompt) and waits for keyboard input
- When user presses Enter, the text is stored as a **string** (text data)
- The equals sign (=) is the *assignment operator* that stores values in variables
- All three inputs will be stored as text, even if numbers are entered

2.3 Type Conversion: Making Data Useful

Since inputs are strings, we need to convert them to numbers for calculations:

```
# Convert to appropriate data types
score = float(rating)    # Convert to decimal number
visits = int(visits)     # Convert to whole number
```

Explanation:

- `float()` converts text to decimal numbers (e.g., "7.5" → 7.5)
- `int()` converts text to whole numbers (e.g., "3" → 3)
- Without conversion, "5" + "2" would become "52" (text concatenation) instead of 7
- Conversion fails if text contains non-numeric characters

Data Types Cheat Sheet

Type	Use
`str`	Text data (e.g., names)
`int`	Whole numbers (e.g., visit counts)
`float`	Decimal numbers (e.g., ratings)
`bool`	True/False values

2.4 Conditional Logic: Making Decisions

This is where we implement our customer categorization using `if-elif-else` statements:

```
if score >= 7 and visits >= 3:
    category = "loyal"
elif score >= 5 and visits >= 2:
    category = "regular"
else:
    category = "occasional"
```

Explanation:

- `if` checks the first condition: high rating AND frequent visits
- `elif` (else-if) checks an alternative condition if first fails
- `else` catches all other cases that don't meet previous conditions
- `and` requires both conditions to be true simultaneously
- Indentation (spaces before lines) defines code blocks - crucial in Python!

Conditional Logic Explained

Real-world analogy: Just like a store manager deciding:
"IF customer visits ¿ 3 times AND rating ¿ 7, THEN give loyal discount"
The program follows the same decision-making process.

2.5 Putting It All Together

Here's the complete customer segmentation program:

```
# Collect customer information
name = input("Enter your name: ")
rating = input("Rate your experience (1-10): ")
visits = input("How many visits this month? ")

# Convert to numbers
score = float(rating)
visit_count = int(visits)

# Categorize customer
if score >= 7 and visit_count >= 3:
    category = "loyal"
elif score >= 5 and visit_count >= 2:
    category = "regular"
else:
    category = "occasional"

# Show result
print(f"\nHello {name}!")
print(f"You are our {category} customer!")
```

Explanation:

- **Stage 1:** Collect raw inputs as text
- **Stage 2:** Convert to numerical values for processing
- **Stage 3:** Apply business logic using conditional statements
- **Stage 4:** Generate personalized output using f-strings
- The `\n` creates a newline for better formatting
- f-strings (f"...") allow embedding variables directly in text

2.6 Debugging: Finding and Fixing Errors

Debugging is like detective work. Let's examine a gradient descent implementation with intentional errors:

```
# Buggy gradient descent code (excerpt)
area_input = input("Enter area in sq ft:"  # Missing closing parenthesis
price_input = input("Enter actual price in    :")

area = float(area_input   # Missing closing parenthesis
actual_price = float(price_input)

for step in range(steps)  # Missing colon
    if true_labels[i] == predicted_labels[i]  # Missing colon
        correct_predictions = correct_predictions + 1
```

Explanation:

- Parentheses must always come in pairs - both opening and closing

- Colons are required at the end of conditionals and loop declarations
- Python uses indentation instead of braces to define code blocks
- Error messages often point to the line AFTER the actual mistake
- Debugging tip: Read code aloud to catch syntax errors

Common Python Errors

Error	Solution
Missing : after conditions	Add colon
Missing parentheses	Check opening/closing pairs
Type mismatches	Use proper conversions
Incorrect indentation	Use consistent 4-space indents

2.7 Homework: Linear Regression Model

Complete this rent prediction program:

```
# Linear Regression Model: Rent = 15 * Area + 3000
user_name = input("Enter tenant name: ")
area_input = input("Enter area in sq ft: ")

# Type conversion
area = float(area_input)

# Rent prediction
rent = 15 * area + 3000

# Budget check
budget = float(input("Enter your budget: "))
if rent <= budget:
    status = "affordable"
else:
    status = "beyond budget"

# Output
print(f"Hello {user_name}")
print(f"Predicted rent:    {rent:.2f}")
print(f"This property is {status} for you")
```

Explanation:

- Linear regression predicts values using a mathematical formula
- Formula components: 15 (cost per sq ft), 3000 (base rent)
- Budget comparison uses conditional logic (if-else)
- `f"rent:.2f"` formats rent as currency with 2 decimal places
- Real-world connection: Similar models power real estate apps

3 Conclusion: Key Takeaways

Today we explored how programming bridges real-world problems and computational solutions. Through our retail case study, we learned:

- **Variables** store information like containers
- **Input/Output** enables user interaction
- **Type conversion** transforms data for processing
- **Conditionals** implement decision-making logic
- **Debugging** is essential for fixing code errors

These fundamentals form the foundation for advanced AI concepts. Remember:

Programming Mantra

"Good programmers don't just write code - they solve real problems through logical thinking and clear implementation."

In our next session, we'll extend these concepts to machine learning algorithms and data analysis techniques. Practice these fundamentals through the homework exercises!