MODULE 1: CORE PYTHON & DATA

WEEK: 1 LECTURE: 2

DATE: 18/08/2025

INSTRUCTOR: ORANGZAIB RAJPOOT

PYTHON'S FOUNDATION: SYNTAX, VARIABLES & DATA TYPES

Today's Agenda

- The Rules of the Road - Python Syntax & Variables
  - Python's Guiding Philosophy
  - Syntax: Indentation and Comments
  - Variables: Your Program's Memory
  - Variable Naming Rules
  - Interactive Exercises throughout
- The Building Blocks - Core Data Types
  - Introduction to Data Types with type()
  - Numeric Types: Integers (int) and Floats (float)
  - Text Type: Strings (str) and String Formatting
  - Boolean Type: True and False (bool)
  - Interactive Exercises for each data type
- Putting It All Together - Type Casting & Hands-On Lab
  - Type Casting: How to Change a Variable's Type
  - The input() function and its challenges
  - Hands-On Lab: The Variable Playground
  - Q&A and Wrap-up

PYTHON'S PHILOSOPHY, SYNTAX & DYNAMIC TYPING THE ZEN OF PYTHON: IMPORT THIS

Key Difference #1: Indentation is Syntax

This is the most critical concept to grasp. Unlike languages that use curly braces {} to denote code blocks, Python uses whitespace.

- **What it means:** The indentation of your code directly affects its execution. It's not just for style.
- **The Rule:** The standard is **four spaces** per indentation level.
- **Why?** It forces clean, readable code for everyone. You can't write messy, inconsistently formatted code blocks in Python.

```
# In a language like Java or C++:
# if (x > y) {
#   System.out.println("x is greater");
#   // more code...
# }

# The Python equivalent:
x = 10
y = 5
if x > y:
    print("x is greater")  # This block is defined by its indentation
    print("This is also inside the if statement")

print("This is outside the if statement") # This line is not indented
```

IN CLASS EXERCISE: SPOT THE ERROR

```
# --- BROKEN CODE ---
name = "Alice"
if name == "Alice":
print("Hello, Alice!")
else:
    print("You are not Alice.")
```

COMMENTS: NOTES TO YOURSELF

- Comments are lines in your code that Python ignores. They are for humans to read. Use them to explain what your code does.

```
# This is a single-line comment. It explains the next line of code.
# Calculate the area of a circle with a radius of 5
pi = 3.14159
radius = 5
area = pi * (radius ** 2) # The ** operator means 'to the power of'
```

VARIABLES & NAMING CONVENTIONS

- Think of a variable as a **labeled box** where you can store a piece of information. You give the box a name, and you can put data inside it. You can also change what's inside the box later.
- **snake_case:** In Python, the standard for variable and function names is to use all lowercase letters, with words separated by underscores.
- **Contrast:** This is different from the **camelCase** (firstName) often used in Java or JavaScript. While camelCase will work, snake_case is the accepted community standard.

Variable Naming Conventions (The Rules for Labels)

- **Must start with a letter or an underscore (_).** (name is good, _name is good, 1name is bad).
- **Cannot contain spaces.** Use underscores instead. This is called **snake_case.** (first_name is good, firstname is good, first name is bad).
- **Can only contain letters, numbers, and underscores.** (user_age_1 is good, user-age is bad).
- **Names are case-sensitive.** (age, Age, and AGE are three different variables).
- **Use meaningful names!** (user_email is much better than ue).

```
# Pythonic way (snake_case)
first_name = "Guido"
user_email_address = "guido@python.org"

# Non-Pythonic (camelCase)
# firstName = "Guido"
# userEmailAddress = "guido@python.org"
```

DYNAMIC TYPING

Python is a **dynamically typed** language.

- **What it means:** You do not need to declare the type of a variable. The type is inferred at runtime. A variable can even hold different types of data throughout the program's execution.
- **Contrast:** In **statically typed** languages (like C++ or Java), you must declare the type, and it cannot be changed. int age = 30;

```
# This is perfectly valid in Python
my_variable = 101          # my_variable is an integer
print(type(my_variable))

my_variable = "Now I'm a string" # The same variable now holds a string
print(type(my_variable))
```

THE BUILDING BLOCKS - CORE DATA TYPES

- Every piece of data in Python has a "type." This tells Python what kind of data it is and what you can do with it. You can always check a variable's type using the built-in type() function.

```
my_variable = "Hello"
print(type(my_variable)) # Output: <class 'str'>
```

A DEEPER DIVE INTO CORE DATA TYPES NUMBERS: INT AND FLOAT

- **Integers (int):** Whole numbers, both positive and negative.
  - user_count = 25
  - temperature = -10
- **Floats (float):** Numbers with a decimal point.
  - price = 19.99
  - pi_value = 3.14159
- **Division:** This is a common trip-up for programmers from other languages.
  - / (True Division): Always returns a float.
  - // (Floor Division): Discards the fractional part and returns an integer (or float if one of the numbers was a float).

- x = 10
- y = 3

- print(x + y) # Addition: 13
- print(x - y) # Subtraction: 7
- print(x * y) # Multiplication: 30
- print(x / y) # True Division: 3.333...
- print(x // y) # Floor Division (discards remainder): 3
- print(x % y) # Modulus (returns the remainder): 1
- print(x ** y) # Exponent (10 to the power of 3): 1000

IN-CLASS EXERCISE: SIMPLE MATH

- Create two variables, item_price and quantity. Assign them numeric values. Calculate the total_cost and print it.
- You have 25 slices of pizza and 7 people. Calculate and print:
  - How many slices each person gets (//).
  - How many slices will be left over (% modulus operator).

TEXT TYPE: STRINGS (STR)

- Strings are used to represent text. You can create them with single (') or double (") quotes.

```
single_quote_string = 'This is a string.'

double_quote_string = "This is also a string."
```

- **String Concatenation:** You can "add" strings together to join them.

```
first_name = "Grace"
last_name = "Hopper"
full_name = first_name + " " + last_name # The " " adds a space
print(full_name) # Output: Grace Hopper
```

F-STRINGS (FORMATTED STRING LITERALS)

- Introduced in Python 3.6, this is the modern, preferred way to format strings. It's faster and more readable than older methods.

```
language = "Python"

version = 3.9

# The f-string allows you to embed expressions directly inside {}.

message = f"I am programming in {language} version {version}."

print(message)
```

Create variables for first_name and favorite_language. Use an f-string to print a sentence like: "My name is Ada and my favorite programming language is Python."

- **Useful String Methods:** Strings are objects with built-in functions (methods).

```
raw_data = "   UserID:12345   "
clean_data = raw_data.strip()      # Removes leading/trailing whitespace -> "UserID:12345"
user_id = clean_data.replace("UserID:", "") # Replaces a substring -> "12345"
print(user_id)
```

- Given the variable dirty_string = "---HELLO, WORLD!---". Write a single line of code that chains string methods to produce the output: "hello, world!". (Hint: you might need .strip() and .lower()).

BOOLEAN TYPE: BOOL

- Booleans represent one of two values: True or False. They are the foundation of decision-making in programming. Note the capital T and F.
- Booleans are often the result of comparisons:

```
x = 10
y = 5

is_greater = x > y      # is_greater is now True
is_equal = (x == y)     # is_equal is now False
is_not_equal = (x != y) # is_not_equal is now True

print(f"Is x greater than y? {is_greater}")
```

In a boolean context (like an if statement), many things besides True are considered "truthy." However, some specific values are considered **"Falsy"**:

- The number 0
- An empty string ""
- An empty list [] (we'll learn about these later)
- The special value None

IN CLASS EXERCISE: SIMPLE COMPARISON

- Create a variable my_age and another variable voting_age = 18. Write a line of code that prints True or False depending on whether my_age is old enough to vote (>=).

```
user_name = input("Enter your name (or leave blank): ")

# This works because an empty string is "Falsy"
if user_name:
    print(f"Welcome, {user_name}!")
else:
    print("Welcome, guest!")
```

- This allows for more concise and readable code than if user_name != "":.

TYPE CASTING

- Because Python is dynamically typed, you sometimes need to explicitly tell it to convert data from one type to another. The most common scenario is handling user input.
- **The input() function always returns a string.**

```
# --- BROKEN CODE ---

current_year = 2025

birth_year_str = input("What year were you born? ") # User enters "1995"


# age = current_year - birth_year_str

# print(age)

# This will cause a TypeError: can only concatenate str (not "int") to str


# --- CORRECTED CODE ---

birth_year_int = int(birth_year_str) # Explicitly cast the string to an integer

age = current_year - birth_year_int

print(f"You are approximately {age} years old.")
```

IN CLASS EXERCISE: FIX THE BUG (BY TYPE CASTING)

```
item_price_str = "19.99"

tax_rate = 0.07

# Add your fix here!

total_price = item_price_str * (1 + tax_rate)

print(f"Total price: {total_price}")
```

EXERCISE 1: PERSONAL BIO CREATOR

- Create variables for your name (string), age (integer), city (string), and is_learning_python (boolean).
- Use a single multi-line f-string to print a formatted biography.

EXERCISE 2: INTERACTIVE TIP CALCULATOR

- Ask the user for the bill_total using input().
- Ask the user for the tip_percentage they want to leave (e.g., 15, 18, 20).
- **Calculations:**
  - Cast the inputs to the correct numeric types (float for the bill, int for the tip percentage).
  - Convert the tip percentage to a decimal (e.g., 20 becomes 0.20).
  - Calculate the tip_amount and the grand_total.
- Print a formatted summary: "For a bill of $XX.XX, a XX% tip is $Y.YY, for a grand total of $Z.ZZ."

EXERCISE 3: STRING MANIPULATION & PARSING

- You are given a string: log_entry = "INFO:2025-08-18:User 'admin' logged in successfully."
- Your task is to extract the date, the username, and the message.
- Use string methods like .split() and .replace() to parse the string.
- Print the extracted information cleanly:

Date: 2025-08-18

Username: admin

Message: User logged in successfully.

EXERCISE 4 (CHALLENGE): SIMPLE VENDING MACHINE LOGIC

- Create a variable balance = 2.00.
- Create a variable item_price = 1.50.
- Ask the user if they are a student ("yes" or "no").
- A 10% discount is applied if the user is a student.
- Use an if statement to check their answer and adjust the item_price if necessary.
- Determine if their balance is sufficient to buy the item (at the potentially discounted price).
- Print a final boolean True or False to the variable can_purchase.