Intelligence Academy

Center for Research and Development

Chapter 1: Python Foundations – Your First Step into Code: Part 03

Mejbah Ahammad
Intelligence Academy

Chapter 1: Python Foundations – Your First Step into Code

Chapter 1: Python Foundations – Your First Step into Code

1. **1.1 Introduction to Python** Learn what Python is, its history, and why it’s widely used today.
2. **1.2 Installing Python and Setting Up the Environment** Step-by-step guide on installing Python, setting up VS Code or Jupyter Notebook, and verifying your setup.
3. **1.3 Writing Your First Python Program** A simple ‘Hello, World¡ program and executing scripts from terminal or IDE.
4. **1.4 Understanding Python Syntax and Structure** Learn about indentation, code blocks, statements, and line continuation.
5. **1.5 Variables and Data Types** Explore Python’s core data types: integers, floats, strings, booleans, and dynamic typing.

Chapter 1: Python Foundations – Your First Step into Code

6. **1.6 Basic Input and Output** Using ‘input()‘ and ‘print()‘ for interactive programs.
7. **1.7 Comments and Code Readability** How to write single-line and multi-line comments, and best practices for clean code.
8. **1.8 Exercises and Practice Problems** Practice questions covering all topics in Chapter 1.
9. **1.9 Summary and What’s Next** Recap of key points and a teaser of the next chapter (e.g., control flow or functions).

1.4 Understanding Python Syntax and Structure

Python’s Design Philosophy: Python emphasizes readability and simplicity. Unlike many other languages that use braces `{}` or keywords to define code blocks, Python uses **indentation**.

1. Indentation (Block Structure):

- Indentation is used to define blocks of code such as loops, functions, conditionals.
- Standard convention: use 4 spaces per indentation level (avoid using tabs).
- Example:

```python
if 5 > 3:
    print("Five is greater than three.")
```

1.4 Understanding Python Syntax and Structure

- Incorrect indentation results in a `IndentationError`.

2. Statements and Code Lines:

- Each line in Python is typically one statement.
- You can write multiple statements on one line using a semicolon `;` (not recommended).
- Example:

```python
x = 5; y = 10; print(x + y)
```

3. Line Continuation:

- Python allows implicit and explicit line continuation.

1.4 Understanding Python Syntax and Structure

- Implicit: inside parentheses, brackets, or braces.
- Explicit: using a backslash \ at the end of the line.
- Examples:

```python
# Implicit
total = (1 + 2 + 3 +
         4 + 5)

# Explicit
x = 10 + \
    20 + \
    30
```

4. Comments:

1.4 Understanding Python Syntax and Structure

- Single-line comments start with #
- Multi-line comments can be written using triple quotes (”’ or """)
- Example:

```python
# This is a single-line comment

"""
This is a multi-line comment
or docstring in Python
"""
```

5. Code Readability Tips:

- Stick to consistent indentation (PEP8 recommends 4 spaces).

1.4 Understanding Python Syntax and Structure

- Leave blank lines between functions and logical sections.
- Use meaningful variable and function names.

Summary: Python’s elegant syntax makes it intuitive and beginner-friendly, but it demands attention to whitespace and structure.

1.5 Variables and Data Types

What is a Variable?

A variable is a name that refers to a value stored in memory. In Python, variables are created when you assign a value using the equals sign =.

```python
x = 5               # Integer
name = "Mejbah" # String
price = 19.99   # Float
is_valid = True # Boolean
```

Dynamic Typing in Python:

- Python is dynamically typed — you don't need to declare variable types.
- Variable types are inferred from the assigned value.
- You can reassign a variable to a different type:

1.5 Variables and Data Types

```python
x = 10         # Initially an integer
x = "ten"      # Now a string
```

Core Data Types:

- **Integers (int):** Whole numbers (positive or negative)

```python
age = 25
```

- **Floating-point Numbers (float):** Numbers with decimals

```python
pi = 3.14159
```

- **Strings (str):** Sequence of characters, enclosed in quotes

```python
language = "Python"
```

1.5 Variables and Data Types

- **Booleans (bool):** Represent truth values: `True` or `False`

```python
is_active = False
```

Type Checking and Conversion:

- Use `type()` to check the data type:

```python
print(type(language))  # Output: <class 'str'>
```

- Use type conversion functions:

```python
int("10")      # Converts string to integer
float("3.14") # Converts string to float
str(100)       # Converts integer to string
bool(1)        # Converts to True
```

1.5 Variables and Data Types

Naming Rules for Variables:

- Can include letters, numbers, and underscores.
- Must begin with a letter or underscore.
- Case-sensitive: `name` and `Name` are different.
- Avoid using Python keywords (e.g., `if`, `class`, `return`).

Summary: Variables in Python are simple to use due to dynamic typing. Understanding data types is essential for controlling program logic, calculations, and memory.

1.6 Basic Input and Output

1. Displaying Output with `print()`

- The `print()` function is used to display text or variable values to the screen.
- You can print strings, numbers, or variables.
- Example:

```python
print("Welcome to Python!")
name = "Mejbah"
print("Hello,", name)
```

- `print()` can also format output using f-strings:

```python
age = 25
print(f"{name} is {age} years old.")
```

1.6 Basic Input and Output

2. Taking Input with `input()`

- `input()` pauses the program and waits for the user to type something.
- It always returns a **string**, even if the user types a number.
- Example:

```python
username = input("Enter your name: ")
print("Welcome,", username)
```

3. Type Conversion with Input

- Since `input()` returns a string, you may need to convert it to a number:

```python
num1 = input("Enter a number: ")
num1 = int(num1)  # Convert string to integer
```

1.6 Basic Input and Output

```python

# Or directly in one line
age = int(input("Enter your age: "))
```

- Use `float()` for decimal input:

```python
height = float(input("Enter your height in meters: "))
```

4. Multi-Line Output

```python
print("Hello!\nWelcome to Python Programming.\nLet's begin.")
```

5. Custom Separators and Endings

- `print()` accepts optional parameters like sep and end:

1.6 Basic Input and Output

```python
print("Python", "is", "fun", sep="-")   # Output: \
    Python-is-fun
print("Loading", end="...")            # Keeps cursor on \
    the same line
```

Summary: Input and output functions make your Python programs interactive. Always remember to convert input values to the appropriate type before performing calculations.

1.7 Comments and Code Readability

What are Comments?

Comments are notes in your code that are ignored by the Python interpreter. They are used to explain the logic, describe steps, or leave reminders for yourself or other developers.

1. Single-Line Comments

- Start with the hash symbol #.
- Anything after # on the same line is ignored by Python.
- Example:

```python
# This is a single-line comment
print("Hello, World!")  # This prints a message
```

1.7 Comments and Code Readability

2. Multi-Line Comments (Docstrings)

- Use triple quotes: ”’ ... ”’ or """ ... """
- Typically used to describe functions, classes, or modules.
- Example:

```python
"""
This is a multi-line comment.
Useful for documentation and descriptions.
"""
print("Running the program...")
```

3. Docstrings for Functions

```python
def greet(name):
```

1.7 Comments and Code Readability

```python
    """This function greets the person passed in as a \
        parameter."""
    print(f"Hello, {name}!")
```

`greet.__doc__` will return the docstring.

4. Best Practices for Readable Code

- Use comments to explain why, not what (the code already shows what it does).
- Keep comments concise and relevant.
- Follow PEP 8: the official Python style guide.
- Use meaningful variable and function names:

1.7 Comments and Code Readability

```python
# Bad:
x = 10

# Good:
student_age = 10
```

- Break long code into logical sections with line breaks and section comments.
- Avoid redundant comments:

```python
# Increment x by 1
x = x + 1  # OK

x = x + 1  # BAD: No need to repeat obvious logic
```

1.7 Comments and Code Readability

Summary: Well-commented and cleanly written code is easier to read, understand, and maintain. Always write code like someone else (or future you) will read it.

1.8 Exercises and Practice Problems

Objective: Reinforce your understanding of Python basics through practical coding exercises.

Exercise Set A – Basic Syntax & Variables

1. Write a Python program that prints your name and age.
2. Create variables to store your country, university, and favorite subject. Then print them.
3. Assign an integer to a variable, then reassign it to a float and print both types using `type()`.
4. Try printing multiple values in one line using both `,` and `f-strings`.

1.8 Exercises and Practice Problems

Exercise Set B – Input & Output

1. Ask the user for their name and greet them with “Hello, [name]!”
2. Create a simple calculator that:
    - Asks the user to enter two numbers
    - Adds them and prints the result
3. Take a number input from the user and print its square.
4. Write a program that asks for a favorite food, then prints:

```
I love [food] too!
```

Exercise Set C – Data Types & Conversion

1.8 Exercises and Practice Problems

1. Convert a string input to integer and float. Print both.
2. Check and print the type of variables: `"True"`, `5.0`, `3`, `False`
3. Create and print the result of a Boolean expression, e.g., `5 > 2`

Exercise Set D – Readability & Comments

1. Rewrite a cluttered program with:
    - Proper indentation
    - Descriptive variable names
    - At least 2 useful comments
2. Add a multi-line comment to describe a program’s purpose.

1.8 Exercises and Practice Problems

Bonus Challenges (Optional)

1. Write a program that:
   - Takes user's name and age
   - Calculates and prints the year they will turn 100
2. Create a simple interactive menu using print + input, like:

```
Choose an option:
1. Say Hello
2. Say Goodbye
```

3. Use line continuation (\) to break a long arithmetic expression across lines.

1.8 Exercises and Practice Problems

Tip: Run your code, make mistakes, and fix them. That’s the best way to learn!

Submission: Prepare your code as a ‘.py‘ file or Jupyter notebook and upload this in your github.