Chapter 2 : Variables and Basic Data Types

In this chapter, we will explore one of the fundamental concepts in Python: variables and basic data types. Variables are used to store data that your program can manipulate, while data types define the kind of values a variable can hold, such as numbers, text, or logical values.

We will cover:

- How to create and name variables properly.
- Python's basic data types, including integers, floats, strings, and booleans.
- Type conversion and type checking.
- Understanding mutable and immutable types.

By the end of this chapter, you will have a solid understanding of how Python stores and manages data, which is essential for writing programs that are both correct and maintainable.

> Note: Understanding variables and data types is a foundation for all programming. Almost every program you write will involve storing, modifying, and using data, so mastering these basics early will make learning more advanced concepts much easier.

As with all chapters in this course, we will use Test-Driven Development (TDD) with pytest to practice writing tests while learning new concepts.

Python Variable

In Python, a variable is a name that refers to a value stored in memory. Variables are created automatically when a value is assigned to them, and their value can be updated at any time. Python also supports assigning multiple variables at once and is dynamically typed, meaning the type of a variable is determined by the value it holds.

```
x = 10          # Integer
name = "Alice"  # String
a, b = 1, 2     # Multiple assignment
x = 20          # Updating value
```

Key Points:

- Variables store data that your program can manipulate.
- Python automatically infers the variable type based on the assigned value.
- Variables can be reassigned to values of a different type.

Python and Dynamic Typing

Python is a dynamically typed language, which means you don't need to declare the type of a variable explicitly. However, dynamic typing does not mean that variables have no type — every value still has a type that Python tracks at runtime.

To improve code quality and reduce errors, Python supports type hints. Using type hints helps:

- Detect errors earlier, both at coding time and during runtime.
- Make your code more readable and maintainable.
- Enable better IDE support, including autocomplete, type checking, and static analysis.

Example with type hints:

```
def greet(name: str) -> str:
    return f"Hello, {name}"
```

Here, name is expected to be a string, and the function returns a string. Type hints don't enforce types at runtime but are very useful for catching mistakes early and improving development experience.

Basic Data Types

Python provides several built-in data types that help you store and work with different kinds of data. Here's a quick overview:

Category	Type	Description
Boolean	bool	Represents True or False values, often used in conditions and logical operations.
Numerics	int, float, complex	Numbers in Python: int for integers, float for decimal numbers, and complex for complex numbers with real and imaginary parts.
Text	str	A sequence of characters, used to represent text. Strings are immutable in Python.
Sequence	list, tuple, range	Ordered collections of items. list is mutable, tuple is immutable, and range generates a sequence of numbers.
Sets	set	An unordered collection of unique values. Useful for eliminating duplicates and performing set operations.
Mapped	dict	A collection of key-value pairs, where each key maps to a value. Keys must be unique.

Notes:

- Python is dynamically typed, so the type of a variable is determined automatically when a value is assigned.
- Choosing the right data type helps make your code clear, efficient, and error-free.
- Most types support useful built-in methods for easy data manipulation.

Boolean Type

The Boolean data type in Python represents truth values: True or False. Booleans are essential for logical operations, comparisons, and decision-making in programs.

Key Points

- Booleans represent either True or False.
- Often used in conditions for if, while, and other control flow statements.
- Results from comparisons produce Boolean values.

Examples

Comparison Result

```
print(5 > 3)   # True
print(2 == 4)  # False
```

Using Booleans in Conditions

```
is_raining = True

if is_raining:
    print("Take an umbrella.")
else:
    print("No umbrella needed.")
```

Logical Operations

```
a = True
b = False

print(a and b)  # False
print(a or b)   # True
print(not a)    # False
```

Why Booleans are Important

Booleans form the basis for decision-making in Python programs. They allow your code to respond differently depending on conditions, loops, or logical checks.

Numeric Data Types

Python provides three main numeric types:

1. int – Integer numbers (whole numbers)
2. float – Decimal numbers (floating-point numbers)
3. complex – Complex numbers with real and imaginary parts

Numeric types are used for calculations, comparisons, and data processing.

Examples

Integer(int)

```
x = 10
y = -5
z = 0

print(x + y)   # 5
print(x * 2)   # 20
```

Floating-Point Numbers (float)

```
pi = 3.14159
radius = 5.0

area = pi * radius ** 2
print(area)    # 78.53975
```

Complex Numbers (complex)

```
a = 2 + 3j
b = 1 - 1j

print(a + b)   # (3+2j)
print(a * b)   # (5+1j)
```

Why Numbers Are Important

Numbers are fundamental in programming. They allow us to:

- Represent real-world values (money, age, temperature).
- Perform calculations and algorithms.
- Control logic and decisions (scores, limits).
- Drive loops and iteration.
- Serve as the basis for data like text, images, and audio.

👉 Without numbers, most computations and real-world applications would not be possible.

Text Data Type (str)

The string (str) data type in Python represents a sequence of characters. Strings are commonly used to store textual information such as names, messages, or any sequence of characters.

Key Points

- Strings are immutable: once created, their content cannot be changed directly.
- Strings can be enclosed in single quotes '...', double quotes "...", or triple quotes '''...''' / """..""" for multi-line strings.
- Strings support concatenation, repetition, and indexing/slicing.

Examples

Creating Strings

```
name = "Alice"
greeting = 'Hello'
multiline = """This is
a multi-line
string."""
```

Concatenation and Repetition

```
full_greeting = greeting + " " + name
print(full_greeting)  # Hello Alice

echo = "Hi! " * 3
print(echo)           # Hi! Hi! Hi!
```

Indexing and Slicing

```
word = "Python"
print(word[0])     # 'P' (first character)
print(word[-1])    # 'n' (last character)
print(word[0:4])   # 'Pyth' (slice)
```

Why Strings Are Important

- Strings are essential for user input/output, messages, and text processing.
- Combining strings with functions and tests ensures your text-handling code is correct and reliable.

Sequence Data Types

In Python, sequences are ordered collections of items. They allow us to store multiple values in a single variable and access them using an index.

Python provides several built-in sequence types:

Types	Descriptions
list	Mutable, ordered collection of items

Types	Descriptions
tuple	Immutable, ordered collection of items
range	Represents a sequence of numbers, commonly used in loops

1. List (list)

- Mutable (can change after creation).
- Items can be of different data types.
- Supports operations like append, remove, slicing, etc.

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])        # apple

fruits.append("mango")
print(fruits)           # ['apple', 'banana', 'cherry', 'mango']
```

2. Tuple (tuple)

- Immutable (cannot be changed once created).
- Useful for fixed collections of values.

```
coordinates = (10, 20)
print(coordinates[0])   # 10

# coordinates[0] = 15  # ❌ Error: Tuples are immutable
```

3. Range (range)

- Generates a sequence of numbers.
- Often used with loops.

```
nums = range(5)         # 0,1,2,3,4
print(list(nums))       # [0, 1, 2, 3, 4]

nums2 = range(2, 10, 2) # start=2, stop=10, step=2
print(list(nums2))      # [2, 4, 6, 8]
```

Why Sequences Are Important

- Lists are the most common collection type in Python.
- Tuples are safe for fixed data.
- Ranges make loops efficient.

They form the foundation for data manipulation, iteration, and algorithm design in Python.

Set (set)

In Python, a set is an unordered collection of unique elements, inspired by set theory in mathematics.

Key Characteristics

- No duplicate values.
- Elements must be immutable (e.g., numbers, strings, tuples).
- Unordered → no indexing.

Creating Set

```
fruits = {"apple", "banana", "cherry"}
empty_set = set()   # not {}
```

Basic Set Operations (from Set Theory)

- Union (A ∪ B) → combine elements from both sets.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A | B)   # {1, 2, 3, 4, 5}
```

- Intersection (A ∩ B) → common elements.

```
print(A & B)   # {3}
```

- Difference (A – B) → elements in A but not in B.

```
print(A - B)   # {1, 2}
```

- Symmetric Difference (A Δ B) → elements in A or B but not both.

```
print(A ^ B)   # {1, 2, 4, 5}
```

Use Cases

- Removing duplicates from a list.
- Membership tests (x in set).
- Mathematical set operations for data analysis.

Mapped Data Type (dict)

In Python, a dictionary (dict) is a collection of key-value pairs, similar to a real dictionary where words (keys) map to their meanings (values).

Key Characteristics

- Keys must be unique and immutable (e.g., strings, numbers, tuples).
- Values can be of any type (mutable or immutable).
- Order is preserved (since Python 3.7+).

Creating Dictionaries

```
# Using curly braces
student = {"name": "Alice", "age": 20, "grade": "A"}

# Using dict() function
user = dict(id=1, username="john_doe")
```

Accessing Values

```
print(student["name"])     # Alice
print(student.get("age"))  # 20
```

Adding / Updating Values

```
student["age"] = 21         # update
student["major"] = "Math"   # add new key-value
```

Removing Elements

```
student.pop("grade")      # remove by key
del student["major"]      # delete key
student.clear()           # empty dict
```

Iterating over Dictionaries

```
for key, value in student.items():
    print(key, ":", value)
```

Use Cases

- Representing structured data (e.g., JSON-like objects).
- Fast lookup by keys (like IDs, names).
- Configuration storage, mappings, caching, etc.

Recap — Variables and Basic Data Types

In this chapter, we learned the building blocks of Python programming:

🔷 Variables

- A variable is a name that refers to a value.
- Created when you assign a value (x = 10).
- Values can be updated with new assignments.
- Python is dynamically typed → type depends on the assigned value.
- With type hints, we can improve code readability, error detection, and IDE support.

🔷 Basic Data Types

Python provides several fundamental data types:

1. Boolean (bool) → Represents True or False, used in logical operations and conditions.
2. Numbers (int, float, complex) → For whole numbers, decimals, and complex calculations.
3. Text (str) → A sequence of characters for storing and processing text.
4. Sequences (list, tuple, range) → Ordered collections of data (mutable or immutable).
5. Sets (set) → Unordered collection of unique values, supports set theory operations.
6. Mapped (dict) → Key-value pairs, useful for structured data and fast lookups.

🔷 Why These Matter

- Booleans help with decision-making (if, while).
- Numbers allow calculations, measurements, and algorithms.
- Strings handle text-based information.
- Sequences organize data in ordered collections.
- Sets manage uniqueness and perform mathematical set operations.
- Dictionaries represent structured mappings, similar to real-world data records.

✅ Summary: Variables give names to values, and Python's basic data types define what kind of values we can store and manipulate. Together, they form the foundation of every Python program.

After learning about this chapter, you have to try assignment.
Got to Assignment