# A Comprehensive Guide to Python Data Types

In Python, every value has a data type. The data type determines what kind of data a variable can hold and what operations can be performed on it. Choosing the right data type is crucial for writing code that is readable, efficient, and correct.

This guide covers Python's core data types, organized into logical categories.

## Table of Contents

---

## 1. Numeric Types

These types are used to represent numbers.

### Integer ( `int` )

- **What is it?** Whole numbers, positive or negative, without decimals. Python integers have unlimited precision, meaning they can be as large as your computer's memory allows.
- **Key Characteristics:**
  - **Immutable:** You cannot change an integer object. Operations like `x = x + 1` create a *new* integer object.
- **Why use it?** For counting and representing discrete, whole quantities.
- **When to use it (Use Cases):**
  - Counting items (e.g., number of users, loop counters).
  - Unique identifiers (e.g., `user_id = 1294` ).
  - Indexing into sequences like lists and strings.
- **Example:**

```
user_count = 1_000_000 # Underscores are for readability
temperature = -10
```

### Floating-Point Number ( `float` )

- **What is it?** Numbers with a decimal point or in exponential form. They represent real numbers.
- **Key Characteristics:**
  - **Immutable.**
  - **Precision Issues:** Floats are based on the IEEE 754 standard and can have small precision errors. For example, `0.1 + 0.2` is not exactly `0.3` .
- **Why use it?** For representing continuous quantities or measurements where absolute precision is not the top priority.
- **When to use it (Use Cases):**

    - Scientific calculations and measurements (e.g., distance, weight, sensor readings).
    - Financial calculations, but with caution. For high-precision financial math, use the `Decimal` module.
    - Percentages and fractions.
- **Example:**

```
pi_approx = 3.14159
price = 19.99
scientific_notation = 6.022e23 # 6.022 x 10^23
```

### Complex Number ( `complex` )

- **What is it?** Numbers with a real and an imaginary part, denoted by `a + bj`.
- **Key Characteristics:**
    - **Immutable.**
- **Why use it?** It's a specialized type for specific mathematical domains.
- **When to use it (Use Cases):**
    - Electrical engineering (e.g., signal processing).
    - Advanced physics and mathematical modeling.
- **Example:**

```
c = 2 + 3j
print(c.real) # Output: 2.0
print(c.imag) # Output: 3.0
```

---

## 2. Text Sequence Type

### String ( `str` )

- **What is it?** An ordered sequence of Unicode characters used to represent text.
- **Key Characteristics:**
    - **Immutable:** You cannot change a character within a string. Any modification (like `.upper()` or `.replace()`) creates a *new* string.
    - **Ordered:** The characters have a defined sequence. You can access them by index (e.g., `my_string[0]`).
- **Why use it?** It is the universal type for handling all textual data. Its immutability makes it predictable and safe to use as dictionary keys.
- **When to use it (Use Cases):**
    - Storing names, messages, and sentences.
    - File paths and URLs.
    - Representing structured text data like JSON or XML.
- **Example:**

```
name = "Guido van Rossum"
message = 'Python is "fun"!'
multiline_doc = """
This is a document
spanning multiple lines.
"""
```

```
first_char = name[0] # 'G'
# name[0] = 'g' # This would raise a TypeError
```

---

## 3. Sequence Types

These represent ordered collections of items.

### List ( `list` )

- **What is it?** An ordered, general-purpose collection of items. It is the most versatile sequence type.
- **Key Characteristics:**
  - **Mutable:** You can add, remove, and change items after the list is created.
  - **Ordered:** Items maintain their position.
  - **Allows Duplicates:** A list can contain the same item multiple times.
  - **Heterogeneous:** Can contain items of different data types (e.g., `[1, "hello", True]` ).
- **Why use it?** It's the default choice for a collection of items, especially when you expect the collection to change in size or content during the program's execution.
- **When to use it (Use Cases):**
  - A list of items in a shopping cart.
  - A collection of user-submitted data.
  - A sequence of steps to be executed.
- **Example:**

```
shopping_list = ["apples", "bananas", "milk"]
shopping_list.append("bread") # Add an item
shopping_list[0] = "green apples" # Change an item
print(shopping_list) # ['green apples', 'bananas', 'milk', 'bread']
```

### Tuple ( `tuple` )

- **What is it?** An ordered, immutable collection of items. Think of it as a "read-only" list.
- **Key Characteristics:**
  - **Immutable:** Once a tuple is created, you cannot change its contents.
  - **Ordered:** Items maintain their position.
  - **Allows Duplicates.**
  - **Heterogeneous.**
- **Why use it?** For data integrity. Its immutability guarantees that the collection of items will not be accidentally modified. This also makes it "hashable," so it can be used as a key in a dictionary.
- **When to use it (Use Cases):**
  - Returning multiple values from a function: `return x, y` .
  - Storing fixed data that should not change, like RGB color values: `(255, 0, 0)` .

    - As keys in a dictionary where a composite key is needed: `locations[(lat, lon)] = "City"`.
    - Representing a fixed record of data: `person = ("John Doe", 30, "Engineer")`.

- **Example:**

```python
point_2d = (10, 20)
# point_2d[0] = 15 # This would raise a TypeError

# Unpacking a tuple
x, y = point_2d
print(f"X: {x}, Y: {y}") # X: 10, Y: 20
```

---

## 4. Mapping Type

### Dictionary ( `dict` )

- **What is it?** An unordered (before Python 3.7) or insertion-ordered (Python 3.7+) collection of key-value pairs.

- **Key Characteristics:**
    - **Mutable:** You can add, remove, and change key-value pairs.
    - **Ordered (since Python 3.7):** Keys are kept in the order they were inserted.
    - **Keys are Unique:** Each key in a dictionary must be unique.
    - **Keys must be Immutable:** Keys must be of a hashable type (e.g., `str`, `int`, `tuple`). Values can be anything.

- **Why use it?** For extremely fast lookups. When you need to associate data (a value) with a unique identifier (a key), a dictionary is the perfect tool.

- **When to use it (Use Cases):**
    - Representing a JSON object.
    - Storing user profile information: `{'username': 'alice', 'email': 'a@b.com'}`.
    - Passing keyword arguments to functions.
    - As a fast lookup table or cache.

- **Example:**

```python
user_profile = {
    "username": "jane_doe",
    "email": "jane@example.com",
    "is_active": True,
    "login_attempts": 3
}

# Accessing a value by key
print(user_profile["username"]) # "jane_doe"
```

```python
# Adding a new key-value pair
user_profile["last_login"] = "2023-10-27"
```

---

## 5. Set Types

These are collections of unique items.

### Set ( `set` )

- **What is it?** An unordered collection of unique, immutable items.
- **Key Characteristics:**
  - **Mutable:** You can add or remove items from a set.
  - **Unordered:** Items have no defined position; you cannot index them.
  - **No Duplicates:** Sets automatically discard duplicate items.
  - **Heterogeneous** (but all items must be hashable/immutable).
- **Why use it?** For its two superpowers: (1) ensuring uniqueness and (2) performing high-speed membership testing ( `item in my_set` ). It is also ideal for mathematical set operations (union, intersection, difference).
- **When to use it (Use Cases):**
  - Removing duplicates from a list: `unique_items = set(my_list)` .
  - Checking for the presence of an item in a large collection (much faster than a list).
  - Finding common items between two collections ( `set1.intersection(set2)` ).
- **Example:**

  ```python
  tags = {"python", "data", "web", "python"}
  print(tags) # {'web', 'python', 'data'} - duplicates removed, order not guaranteed

  tags.add("devops") # Add an item

  # Membership testing is very fast
  if "python" in tags:
      print("This is a Python post.")
  ```

### Frozen Set ( `frozenset` )

- **What is it?** An immutable version of a set.
- **Key Characteristics:**
  - **Immutable:** You cannot add or remove items after creation.
  - **Unordered.**
  - **No Duplicates.**
- **Why use it?** Because it is immutable, it is also "hashable." This means you can use a `frozenset` as a key in a dictionary or as an item in another set, which you cannot do with a regular `set` .

- **When to use it (Use Cases):**
  - As a key in a dictionary where the key represents a unique combination of items.
- **Example:**

```python
allowed_permissions = frozenset(["read", "write"])

# A dictionary where keys are sets of permissions
permission_groups = {
    allowed_permissions: "Standard User",
    frozenset(["read", "write", "execute"]): "Admin"
}
```

---

## 6. Boolean Type

### Boolean ( `bool` )

- **What is it?** Represents one of two values: `True` or `False`. It is a subclass of `int`, where `True` is `1` and `False` is `0`.
- **Key Characteristics:**
  - **Immutable.**
- **Why use it?** For representing truth values and controlling the flow of a program.
- **When to use it (Use Cases):**
  - In `if`, `elif`, and `while` statements to make decisions.
  - To store a state (e.g., `is_logged_in = True`).
  - Return values for functions that check a condition (e.g., `is_valid(data)`).
- **Example:**

```python
is_authenticated = True
has_errors = False

if is_authenticated and not has_errors:
    print("Access granted.")
```

---

## 7. None Type

### None ( `None` )

- **What is it?** A special data type that has only one possible value: `None`.
- **Key Characteristics:**
  - **Immutable.**

- **Why use it?** To represent the absence of a value or a null state. It is distinct from `False`, `0`, or an empty string `""`.
- **When to use it (Use Cases):**
    - To initialize a variable before it is assigned a meaningful value.
    - As a default value for optional function arguments.
    - As a return value for a function that found nothing or did not produce a result.
- **Example:**

```
winner = None # The game hasn't finished yet

def find_user(user_id):
    # ... search logic ...
    if user_found:
        return user_object
    else:
        return None # Indicate that no user was found
```

---

## 8. Summary Table: Quick Reference

| Data Type | Mutable? | Ordered? | Allows Duplicates? | Key Use Case |
|---|---|---|---|---|
| **int** | Immutable | N/A | N/A | Counting, IDs, whole numbers. |
| **float** | Immutable | N/A | N/A | Measurements, scientific data. |
| **str** | Immutable | Yes | Yes | All textual information. |
| **list** | **Mutable** | **Yes** | **Yes** | **General-purpose, modifiable collection of items.** |
| **tuple** | **Immutable** | **Yes** | **Yes** | **Fixed/read-only collection, data integrity, dict keys.** |
| **dict** | **Mutable** | Yes (3.7+) | No (keys) | **Fast key-value lookups, storing structured data.** |
| **set** | **Mutable** | No | **No** | **Uniqueness, fast membership testing, math operations.** |
| **frozenset** | **Immutable** | No | **No** | **As a key in a dict or item in another set.** |
| **bool** | Immutable | N/A | N/A | Control flow, representing state. |
| **None** | Immutable | N/A | N/A | Representing the absence of a value. |