📑 Python Data Types / Object Types – Full Guide with 🧠 Tips & 🧪 Examples

---

📌 1. Numbers 🔢

Types: `int`, `float`, `complex`, `Decimal`, `Fraction`, `binary`

```
a = 1234        # int
b = 3.1415      # float
c = 3 + 4j      # complex
d = 0b111       # binary => 7
```

Advanced:

```
from decimal import Decimal
from fractions import Fraction

Decimal('0.1') + Decimal('0.2')  # More precise than float
Fraction(1, 3) + Fraction(1, 6)  # Exact fractions
```

✅ Use when:

- Accuracy matters (Decimal)
- Need fractional math (Fraction)

📌 2. Strings 🧵

Types: `str`, `bytes`, `unicode`

```
username = "chaiaurcode"
username[0]         # 'c'
username[-1]        # 'e'
username[1:4]       # 'hai'

# Immutable!
username[0] = 'C'   # ❌ TypeError!
```

Other forms:

```
b = b'a\x01c'       # bytes
u = u'sp\xc4m'      # Unicode
```

🧠 Use `dir()` to explore string methods:

```
>>> dir(username)
['capitalize', 'center', 'count', ..., 'upper']
```

📌 3. Lists 📚

Mutable, ordered collection.

```
my_list = [1, [2, 'three'], 4.5]
my_list.append('new')
my_list[1][1] = "3️⃣"  # Nested update
```

Generate list:

```
list(range(10))  # [0, 1, ..., 9]
```

⛏️ Common methods:

```
my_list.pop()
my_list.sort()
my_list.reverse()
```

📌 4. Tuples 📦

Immutable ordered collection

```
t = (1, 'spam', 4, 'U')
tuple('spam')     # ('s', 'p', 'a', 'm')
```

Use when:

- Values should not change
- Used as `dict` keys or `set` elements

🧠 Named Tuple:

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
p = Point(1, 2)
p.x  # 1
```

📌 5. Dictionaries 🧾

Key-value mapping. Unordered (until Python 3.6+) and mutable.

```
profile = {'name': 'Darshan', 'language': 'Python'}
profile['age'] = 22
```

Constructor syntax:

```
dict(hours=10, topic='DSA')
```

🧠 Common methods:

```
profile.keys()
profile.values()
profile.items()
```

📌 6. Sets 🔣

Unordered, **no duplicates**, mutable.

```
set1 = {'a', 'b', 'c'}
set2 = set('abc')  # same

set1.add('d')
set1.remove('b')
```

Use when:

- Need unique items
- Fast lookup & membership test

📌 7. Files 📂

Python uses **file objects** to read/write files.

```
f = open('eggs.txt', 'r')
binary_file = open(r'C:\ham.bin', 'wb')
```

🧠 Always close or use `with`:

```
with open('eggs.txt') as f:
    content = f.read()
```

📌 8. Booleans ✅ ❌

```
is_on = True
is_off = False
```

Used in conditions:

```
if is_on:
    print("Power is ON 🔋")
```

📌 9. None 🤷

Represents the **absence of value**.

```
x = None

if x is None:
    print("Nothing here!")
```

Used for:

- Optional args
- Default values
- Placeholder

📌 10. Functions, Modules, Classes 🧠 📦 🏛

🔸 Function:

```python
def greet(name):
    return f"Hi {name}"
```

🔹 Module:

```python
import math
math.sqrt(16)  # 4.0
```

🔹 Class:

```python
class Person:
    def __init__(self, name):
        self.name = name
```

All are **objects** and can be passed around!

📌 11. Advanced Types 🔮

🔮 Type	📝 Example / Use Case
🎁 Decorator	`@login_required` – modify function behavior
🔁 Generator	`yield` based memory-efficient iterator
⏩ Iterator	Custom `__iter__` and `__next__`
🧬 Metaclass	Control class creation dynamically

🔬 Introspecting Types in Shell

✅ Check Type:

```python
type(username)      # <class 'str'>
type(profile)       # <class 'dict'>
```

✅ Check Attributes & Methods:

```python
dir(username)
help(str)
```

🧾 Summary Table

🔢 **Type**	🔄 **Mutable?**	✅ **Use When...**
`int`, `float`	❌ No	Numeric calculation
`str`	❌ No	Text values, messages
`list`	✅ Yes	Ordered, changeable data
`tuple`	❌ No	Ordered but fixed data
`dict`	✅ Yes	Key-value pairs (JSON-like)
`set`	✅ Yes	Unique unordered items
`file`	✅ Yes	I/O operations
`bool`	❌ No	Decision making
`NoneType`	❌ No	Represent missing value
`function`	✅ Yes	Behavior abstraction
`class`	✅ Yes	Create objects with attributes/methods
`generator`	✅ Yes	Lazy evaluation of sequences

🎯 `username[-1]` in Python 🐍

🧠 What it does:

`username[-1]` is **indexing** a string using a **negative index**.

✅ Syntax:

```
variable_name[index]
```

- **Positive index**: starts from the beginning (left to right)
- **Negative index**: starts from the end (right to left)

📌 Example:

```
username = "chaiaurcode"

print(username[-1])
```

📤 Output:

```
'e'
```

🔢 How indexing works in Python:

Index	0	1	2	3	4	5	6	7	8	9	10
Character	c	h	a	i	a	u	r	c	o	d	e
Neg Index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

So, 👉 `username[0]` → `'c'` 👉 `username[-1]` → `'e'` 👉 `username[-2]` → `'d'` 👉 `username[-3]` → `'o'`

⚠️ If the index is out of range:

```
username = "code"

print(username[-10])
```

❌ Output:

```
IndexError: string index out of range
```

🧪 Real Use Cases:

```
# Get the last character
last = username[-1]

# Get the last 3 characters
last3 = username[-3:]

# Check if ends with 'e'
if username[-1] == 'e':
    print("Ends with 'e'")
```

📊 Python: `List` vs `Tuple` – Complete Comparison 💡

🔢 Sr	Feature	`list` 📋	`tuple` 📦
1️⃣	**Mutability**	✅ Mutable (can be changed)	❌ Immutable (cannot be changed)

🔢 Sr	Feature	`list` 📋	`tuple` 📦
2️⃣	**Syntax**	`[1, 2, 3]`	`(1, 2, 3)`
3️⃣	**Performance**	Slower (due to flexibility)	Faster (fixed size)
4️⃣	**Use Case**	Dynamic, changeable data	Fixed data, read-only scenarios
5️⃣	**Methods**	`.append()`, `.pop()`, `.sort()`	Limited: `.count()`, `.index()`
6️⃣	**Memory**	Consumes more memory	Memory efficient
7️⃣	**Hashable**	❌ No (cannot be used as `dict` key)	✅ Yes (if all elements are immutable)
8️⃣	**Iteration**	Slightly slower	Slightly faster
9️⃣	**Safety**	Risk of accidental change	Safer (protected from change)
🔟	**Conversion**	`list(my_tuple)`	`tuple(my_list)`

🧪 Example

🔹 List

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # [1, 2, 3, 4]
```

🔹 Tuple

```
my_tuple = (1, 2, 3)
my_tuple[0] = 10  # ❌ Error: TypeError
```

🧠 Use When?

✅ Use `list` when:

- Data changes frequently (e.g., user input, database rows)
- Need to add/remove/update items

✅ Use `tuple` when:

- Data is constant (e.g., coordinates, dates)
- Used as keys in `dict`
- For better performance in large-scale apps

🧠 Bonus Tip: Tuple Unpacking

```
x, y = (10, 20)
print(x)  # 10
print(y)  # 20
```

Works with lists too!

🔐 Interview Insight

- `tuple` is **hashable** if it contains only immutable types → usable in `set`, `dict` keys
- `list` is **not hashable** → mutable → unsafe as dict key

```
my_dict = {(1, 2): "hello"}  ✅
my_dict = {[1, 2]: "hello"}  ❌ TypeError
```

🧾 Summary Table

Feature	`List` 📋	`Tuple` 📦
Mutable	✅ Yes	❌ No
Ordered	✅ Yes	✅ Yes
Duplicates	✅ Allowed	✅ Allowed
Performance	🐢 Slower	⚡ Faster
Methods	Many (CRUD)	Few (readonly)
Hashable	❌ No	✅ Yes (if elements are)
Use for	Changing data	Constant or fixed data
