🔒 Tuples Quick Reference Guide

Unit 4 - Lesson 4

What is a Tuple?

A **tuple** is an **ordered, immutable** collection. Once created, it cannot be changed.

```
# List (mutable - CAN change)         Tuple (immutable - CAN'T change)
my_list = [1, 2, 3]                    my_tuple = (1, 2, 3)
my_list.append(4)   # ✅ Works          my_tuple.append(4)  # ❌ Error!
```

Creating Tuples

Method	Example	Result
Parentheses	`point = (10, 20)`	`(10, 20)`
No parentheses	`point = 10, 20`	`(10, 20)`
From list	`tuple([1, 2, 3])`	`(1, 2, 3)`
Empty tuple	`empty = ()`	`()`
Single item	`single = (42,)`	`(42,)`

⚠️ Single-Item Tuple Gotcha!

```
not_tuple = (42)    # ❌ This is just the number 42
is_tuple = (42,)    # ✅ This is a tuple (note the comma!)
also_tuple = 42,    # ✅ This is also a tuple
```

What You CAN Do (Read Operations)

```python
song = ("Blinding Lights", "The Weeknd", 2020, 200)

# Indexing
song[0]          # "Blinding Lights"
song[-1]         # 200

# Slicing
song[1:3]        # ("The Weeknd", 2020)

# Length
len(song)        # 4

# Membership
"The Weeknd" in song    # True

# Looping
for item in song:
    print(item)

# Count occurrences
nums = (1, 2, 2, 3)
nums.count(2)    # 2

# Find index
song.index("The Weeknd")  # 1
```

What You CAN'T Do (Write Operations)

```python
point = (100, 200)

# ❌ Cannot modify items
point[0] = 999
# TypeError: 'tuple' object does not support item assignment

# ❌ Cannot add items
point.append(300)
# AttributeError: 'tuple' object has no attribute 'append'

# ❌ Cannot remove items
point.remove(100)
# AttributeError: 'tuple' object has no attribute 'remove'

# ❌ No: append, insert, extend, remove, pop, clear, sort, reverse
```

Tuple Unpacking

Basic Unpacking

```python
# Pack values into a tuple
song_data = ("Bad Guy", "Billie Eilish", 2019)

# Unpack into variables
title, artist, year = song_data

print(title)   # "Bad Guy"
print(artist)  # "Billie Eilish"
print(year)    # 2019
```

Unpacking Must Match!

```
point = (10, 20, 30)

x, y = point       # ❌ ValueError: too many values
x, y, z, w = point # ❌ ValueError: not enough values
x, y, z = point    # ✅ Works!
```

Safe Unpacking Pattern

```
def safe_unpack(data, expected_count):
    """Safely unpack data with validation."""
    try:
        if expected_count == 2:
            x, y = data
            return x, y
        elif expected_count == 3:
            x, y, z = data
            return x, y, z
    except ValueError:
        return None  # Invalid data
```

Multiple Return Values

Returning Multiple Values

```
def get_min_max(numbers):
    """Return both min and max."""
    return min(numbers), max(numbers)  # Returns a tuple!

# Unpack the result
minimum, maximum = get_min_max([5, 2, 8, 1, 9])
print(f"Min: {minimum}, Max: {maximum}")  # Min: 1, Max: 9
```

Validation Pattern

```python
def validate_input(value):
    """Return (is_valid, error_message)."""
    if not value:
        return False, "Value cannot be empty"
    if len(value) < 3:
        return False, "Value must be at least 3 characters"
    return True, "Valid"

# Usage
is_valid, message = validate_input("ab")
if not is_valid:
    print(f"Error: {message}")
```

Stats Pattern

```python
def calculate_stats(numbers):
    """Return (total, average, count)."""
    if not numbers:
        return 0, 0, 0

    total = sum(numbers)
    count = len(numbers)
    average = total / count

    return total, average, count

# Usage
total, avg, count = calculate_stats([10, 20, 30])
```

Tuples as Dictionary Keys

```python
# ❌ Lists cannot be dict keys (they're mutable)
# locations = {[0, 0]: "origin"}  # TypeError!

# ✅ Tuples CAN be dict keys (they're immutable)
game_map = {
    (0, 0): "spawn",
    (10, 5): "treasure",
    (25, 30): "boss"
}

# Access by coordinate
pos = (10, 5)
print(game_map[pos])        # "treasure"
print(game_map.get((99, 99), "empty"))  # "empty"
```

Tuples Inside Other Structures

List of Tuples

```python
# High scores: (name, score)
high_scores = [
    ("Player1", 9500),
    ("Player2", 8700),
    ("Player3", 8200)
]

# Sort by score (second item)
high_scores.sort(key=lambda x: x[1], reverse=True)

# Unpack while looping
for name, score in high_scores:
    print(f"{name}: {score}")
```

Dictionary with Tuple Values

```python
# Color palette with RGB tuples
colors = {
    "discord_blurple": (88, 101, 242),
    "success_green": (67, 181, 129),
    "danger_red": (237, 66, 69)
}

# Unpack when accessing
r, g, b = colors["discord_blurple"]
hex_color = f"#{r:02x}{g:02x}{b:02x}"
```

Tuples as Metadata in Dicts

```python
song = {
    "title": "Levitating",
    "artist": "Dua Lipa",
    "metadata": (2020, 203, 103)  # (year, duration, bpm)
}

# Unpack metadata
year, duration, bpm = song["metadata"]
```

When to Use Tuples vs Lists

Use **TUPLES** for...	Use **LISTS** for...
Coordinates `(x, y)`	Playlists that change
RGB colors `(r, g, b)`	Items you add/remove
Multiple return values	Data you sort/filter
Dictionary keys	User collections
Fixed configuration	Growing datasets
Data that shouldn't change	Mutable sequences

Quick Decision Guide

- **Need to modify it later?** → List
- **Fixed/constant data?** → Tuple
- **Return multiple values?** → Tuple
- **Use as dict key?** → Tuple
- **Not sure?** → Start with list

Common Patterns

Pattern 1: Coordinate Systems

```python
def move_player(position, direction):
    """Move player and return new position tuple."""
    x, y = position
    dx, dy = direction
    return (x + dx, y + dy)

pos = (100, 200)
pos = move_player(pos, (10, -5))  # (110, 195)
```

Pattern 2: Named Returns

```python
def fetch_user_data(user_id):
    """Return (user_dict, error_message) tuple."""
    if user_id <= 0:
        return None, "Invalid user ID"

    user = {"id": user_id, "name": "Player"}
    return user, None

# Usage
user, error = fetch_user_data(123)
if error:
    print(f"Error: {error}")
else:
    print(f"Welcome, {user['name']}!")
```

Pattern 3: Config Constants

```python
# Define as tuples so they can't be accidentally modified
WINDOW_SIZE = (1920, 1080)
DEFAULT_COLOR = (255, 255, 255)
SPAWN_POINT = (0, 0)

# Use in code
width, height = WINDOW_SIZE
```

Quick Syntax Reference

```python
# Creating
t = (1, 2, 3)          # With parentheses
t = 1, 2, 3            # Without parentheses
t = tuple([1, 2, 3])   # From list
t = (42,)              # Single item (need comma!)

# Accessing
t[0]                   # First item
t[-1]                  # Last item
t[1:3]                 # Slice

# Unpacking
x, y, z = t            # Into variables
a, b, c = my_func()    # From function return

# Checking
len(t)                 # Length
item in t              # Membership
t.count(item)          # Count occurrences
t.index(item)          # Find index

# As dict key
d = {(0, 0): "origin"}
d[(0, 0)]              # Access value
```

Common Errors

Error	Cause	Fix
`TypeError: 'tuple' object does not support item assignment`	Trying to modify tuple	Use list if you need to modify
`ValueError: too many values to unpack`	More items than variables	Match variable count to tuple length
`ValueError: not enough values to unpack`	Fewer items than variables	Match variable count to tuple length
`TypeError: unhashable type: 'list'`	Using list as dict key	Use tuple instead

Remember: Tuples are your tool for **protecting data** and **returning multiple values**. When in doubt, ask: "Should this data ever change?" If no → tuple! 🔒