Lists and Functions in Python

Table of Contents

1. List Fundamentals
2. List Methods and Operations
3. List Comprehensions
4. Function Basics
5. Function Parameters and Arguments
6. Return Values and Scope
7. Practical Examples
8. Best Practices

---

1. List Fundamentals

Creating Lists

```
# Empty list
empty_list = []
also_empty = list()

# List with initial values
fruits = ["apple", "banana", "orange"]
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True, [1, 2, 3]]

# List from range
numbers = list(range(1, 6))        # [1, 2, 3, 4, 5]
even_numbers = list(range(0, 11, 2))  # [0, 2, 4, 6, 8, 10]
```

Accessing List Elements

```
fruits = ["apple", "banana", "orange", "grape", "kiwi"]

# Positive indexing (starts at 0)
first_fruit = fruits[0]      # "apple"
second_fruit = fruits[1]     # "banana"

# Negative indexing (starts from end)
last_fruit = fruits[-1]      # "kiwi"
second_last = fruits[-2]     # "grape"

# Check if element exists
has_apple = "apple" in fruits    # True
has_mango = "mango" in fruits    # False
```

List Slicing

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Basic slicing [start:end] (end is exclusive)
first_three = numbers[0:3]       # [0, 1, 2]
middle = numbers[3:7]            # [3, 4, 5, 6]
last_three = numbers[7:10]       # [7, 8, 9]

# Shortcuts
first_five = numbers[:5]         # [0, 1, 2, 3, 4]
from_five = numbers[5:]          # [5, 6, 7, 8, 9]
all_numbers = numbers[:]         # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Step slicing [start:end:step]
every_second = numbers[::2]      # [0, 2, 4, 6, 8]
reverse_list = numbers[::-1]     # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Modifying Lists

```
fruits = ["apple", "banana", "orange"]

# Change an element
fruits[1] = "blueberry"          # ["apple", "blueberry", "orange"]

# Change multiple elements
fruits[0:2] = ["mango", "grape"] # ["mango", "grape", "orange"]
```

---

2. List Methods and Operations

Adding Elements

```
fruits = ["apple", "banana"]

# Add single element to end
fruits.append("orange")          # ["apple", "banana", "orange"]

# Add multiple elements to end
fruits.extend(["grape", "kiwi"]) # ["apple", "banana", "orange", "grape",
"kiwi"]

# Insert at specific position
fruits.insert(1, "mango")        # ["apple", "mango", "banana", "orange",
"grape", "kiwi"]

# Concatenate lists
more_fruits = ["pear", "peach"]
all_fruits = fruits + more_fruits
```

Removing Elements

```
fruits = ["apple", "banana", "orange", "banana", "grape"]

# Remove by value (first occurrence)
fruits.remove("banana")          # ["apple", "orange", "banana", "grape"]

# Remove by index
removed = fruits.pop(2)          # Returns "banana", list becomes ["apple",
"orange", "grape"]
last_item = fruits.pop()         # Returns "grape", list becomes ["apple",
"orange"]

# Remove all elements
fruits.clear()                   # []

# Delete using del
numbers = [1, 2, 3, 4, 5]
del numbers[2]                   # [1, 2, 4, 5]
del numbers[1:3]                 # [1, 5]
```

Finding and Counting

```
numbers = [1, 3, 5, 3, 7, 3, 9]

# Find index of element
index = numbers.index(5)         # 2
index = numbers.index(3)         # 1 (first occurrence)

# Count occurrences
count = numbers.count(3)         # 3

# Check if element exists
exists = 5 in numbers            # True
not_exists = 10 not in numbers   # True
```

Sorting and Reversing

```
numbers = [3, 1, 4, 1, 5, 9, 2]
words = ["banana", "apple", "cherry"]

# Sort in place (modifies original)
numbers.sort()                   # [1, 1, 2, 3, 4, 5, 9]
words.sort()                     # ["apple", "banana", "cherry"]

# Sort in reverse order
numbers.sort(reverse=True)       # [9, 5, 4, 3, 2, 1, 1]

# Create sorted copy (original unchanged)
original = [3, 1, 4, 1, 5]
sorted_copy = sorted(original)   # [1, 1, 3, 4, 5]

# Reverse in place
numbers.reverse()                # Reverses the current order

# Create reversed copy
reversed_copy = original[::-1]   # [5, 1, 4, 1, 3]
```

List Information

```
numbers = [1, 2, 3, 4, 5]

# Length
length = len(numbers)            # 5

# Min and max
minimum = min(numbers)           # 1
maximum = max(numbers)           # 5

# Sum (for numeric lists)
total = sum(numbers)             # 15
```

---

3. List Comprehensions

Basic List Comprehensions

```
# Traditional way
squares = []
for x in range(1, 6):
    squares.append(x ** 2)       # [1, 4, 9, 16, 25]

# List comprehension way
squares = [x ** 2 for x in range(1, 6)]  # [1, 4, 9, 16, 25]

# More examples
even_numbers = [x for x in range(1, 11) if x % 2 == 0]  # [2, 4, 6, 8, 10]
words = ["hello", "world", "python"]
lengths = [len(word) for word in words]  # [5, 5, 6]
uppercase = [word.upper() for word in words]  # ["HELLO", "WORLD", "PYTHON"]
```

Conditional List Comprehensions

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filter with condition
evens = [x for x in numbers if x % 2 == 0]  # [2, 4, 6, 8, 10]

# Transform with condition
processed = [x * 2 if x % 2 == 0 else x for x in numbers]  # [1, 4, 3, 8, 5,
12, 7, 16, 9, 20]
```

---

4. Function Basics

Function Definition and Calling

```
# Simple function
def greet():
    print("Hello, World!")

# Function with parameters
def greet_person(name):
    print(f"Hello, {name}!")

# Function with return value
def add_numbers(a, b):
    result = a + b
    return result

# Function with multiple parameters and return
def calculate_area(length, width):
    area = length * width
    return area

# Calling functions
greet()                          # "Hello, World!"
greet_person("Alice")            # "Hello, Alice!"
sum_result = add_numbers(5, 3)   # 8
area = calculate_area(10, 5)     # 50
```

Function Parameters

```
# Required parameters
def divide(a, b):
    return a / b

# Default parameters
def greet(name, greeting="Hello"):
    return f"{greeting}, {name}!"

# Variable number of arguments (*args)
def sum_all(*numbers):
    return sum(numbers)

# Keyword arguments (**kwargs)
def create_profile(**info):
    for key, value in info.items():
        print(f"{key}: {value}")

# Examples
print(greet("Alice"))                    # "Hello, Alice!"
print(greet("Bob", "Hi"))                # "Hi, Bob!"
print(sum_all(1, 2, 3, 4, 5))          # 15
create_profile(name="Alice", age=25, city="Boston")
```

---

5. Function Parameters and Arguments

Parameter Types

```
# Positional parameters
def describe_pet(name, animal_type):
    print(f"I have a {animal_type} named {name}")

describe_pet("Buddy", "dog")    # Positional arguments
describe_pet(animal_type="cat", name="Whiskers")  # Keyword arguments

# Default values
def make_pizza(size, *toppings, crust="thin"):
    print(f"Making a {size}-inch pizza with {crust} crust")
    for topping in toppings:
        print(f"  - {topping}")

make_pizza(12, "pepperoni", "mushrooms")
make_pizza(16, "cheese", "olives", "peppers", crust="thick")
```

Argument Unpacking

```
def calculate_stats(numbers):
    return {
        'sum': sum(numbers),
        'avg': sum(numbers) / len(numbers),
        'min': min(numbers),
        'max': max(numbers)
    }

# List unpacking
def add_three(a, b, c):
    return a + b + c

numbers = [1, 2, 3]
result = add_three(*numbers)    # Same as add_three(1, 2, 3)

# Dictionary unpacking
def create_user(name, age, email):
    return f"User: {name}, Age: {age}, Email: {email}"

user_data = {"name": "Alice", "age": 25, "email": "alice@email.com"}
user = create_user(**user_data)
```

---

6. Return Values and Scope

Return Statements

```
# Single return value
def square(x):
    return x * x

# Multiple return values
def get_name_parts(full_name):
    parts = full_name.split()
    first_name = parts[0]
    last_name = parts[-1]
    return first_name, last_name  # Returns a tuple
```

```
# Using multiple return values
first, last = get_name_parts("Alice Johnson")

# Early return
def check_age(age):
    if age < 0:
        return "Invalid age"
    if age < 18:
        return "Minor"
    if age < 65:
        return "Adult"
    return "Senior"

# No explicit return (returns None)
def print_message(msg):
    print(msg)
    # Implicitly returns None
```

Variable Scope

```
# Global scope
global_var = "I'm global"

def scope_example():
    # Local scope
    local_var = "I'm local"
    print(global_var)    # Can access global
    print(local_var)     # Can access local

def modify_global():
    global global_var
    global_var = "Modified global"

# Function parameters are local
def process_data(data):
    data = data.upper()  # This doesn't modify the original
    return data

original = "hello"
processed = process_data(original)
print(original)      # Still "hello"
print(processed)     # "HELLO"
```

7. Practical Examples

Example 1: List Processing Functions

```
def analyze_numbers(numbers):
    """Analyze a list of numbers and return statistics."""
    if not numbers:
        return {"error": "Empty list provided"}

    return {
```

```python
        "count": len(numbers),
        "sum": sum(numbers),
        "average": sum(numbers) / len(numbers),
        "minimum": min(numbers),
        "maximum": max(numbers),
        "range": max(numbers) - min(numbers)
    }

def filter_even_numbers(numbers):
    """Return a new list containing only even numbers."""
    return [num for num in numbers if num % 2 == 0]

def remove_duplicates(items):
    """Return a new list with duplicates removed, preserving order."""
    seen = set()
    result = []
    for item in items:
        if item not in seen:
            seen.add(item)
            result.append(item)
    return result

# Usage
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 4, 6]
stats = analyze_numbers(data)
evens = filter_even_numbers(data)
unique = remove_duplicates(data)

print(f"Statistics: {stats}")
print(f"Even numbers: {evens}")
print(f"Unique numbers: {unique}")
```

Example 2: Text Processing with Lists

```python
def word_frequency(text):
    """Count frequency of each word in text."""
    words = text.lower().split()
    frequency = {}
    for word in words:
        # Remove punctuation
        clean_word = word.strip(".,!?;:")
        if clean_word:
            frequency[clean_word] = frequency.get(clean_word, 0) + 1
    return frequency

def find_longest_words(words, min_length=5):
    """Find words longer than min_length."""
    return [word for word in words if len(word) >= min_length]

def reverse_words(sentence):
    """Reverse each word in a sentence."""
    words = sentence.split()
    reversed_words = [word[::-1] for word in words]
    return " ".join(reversed_words)

# Usage
```

```
text = "Python programming is fun and powerful. Python is great!"
frequencies = word_frequency(text)
long_words = find_longest_words(text.split())
reversed_sentence = reverse_words("Hello world")

print(f"Word frequencies: {frequencies}")
print(f"Long words: {long_words}")
print(f"Reversed: {reversed_sentence}")
```

Example 3: Student Grade Management

```
def add_student_grade(gradebook, student_name, grade):
    """Add a grade for a student."""
    if student_name not in gradebook:
        gradebook[student_name] = []
    gradebook[student_name].append(grade)

def calculate_student_average(grades):
    """Calculate average grade for a student."""
    if not grades:
        return 0
    return sum(grades) / len(grades)

def get_class_statistics(gradebook):
    """Calculate class-wide statistics."""
    all_grades = []
    student_averages = {}

    for student, grades in gradebook.items():
        avg = calculate_student_average(grades)
        student_averages[student] = avg
        all_grades.extend(grades)

    class_average = sum(all_grades) / len(all_grades) if all_grades else 0

    return {
        "class_average": class_average,
        "student_averages": student_averages,
        "highest_grade": max(all_grades) if all_grades else 0,
        "lowest_grade": min(all_grades) if all_grades else 0
    }

# Usage
gradebook = {}
add_student_grade(gradebook, "Alice", 85)
add_student_grade(gradebook, "Alice", 92)
add_student_grade(gradebook, "Bob", 78)
add_student_grade(gradebook, "Bob", 88)

stats = get_class_statistics(gradebook)
print(f"Class statistics: {stats}")
```

8. Best Practices

Function Design Principles

```python
# Good: Single responsibility
def calculate_tax(price, tax_rate):
    return price * tax_rate

def format_currency(amount):
    return f"${amount:.2f}"

# Good: Descriptive names
def get_user_by_email(email, users):
    for user in users:
        if user['email'] == email:
            return user
    return None

# Good: Use docstrings
def fibonacci(n):
    """
    Generate the first n numbers in the Fibonacci sequence.

    Args:
        n (int): Number of Fibonacci numbers to generate

    Returns:
        list: List of Fibonacci numbers
    """
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]

    fib = [0, 1]
    for i in range(2, n):
        fib.append(fib[i-1] + fib[i-2])
    return fib
```

List Best Practices

```python
# Good: Use list comprehensions for simple transformations
squares = [x**2 for x in range(10)]

# Good: Use meaningful variable names
student_names = ["Alice", "Bob", "Charlie"]
test_scores = [85, 92, 78]

# Good: Check if list is empty before processing
def process_items(items):
    if not items:
        return "No items to process"

    processed = []
    for item in items:
        processed.append(item.upper())
    return processed

# Good: Use appropriate methods
# Use extend() instead of multiple append() calls
fruits = ["apple", "banana"]
new_fruits = ["orange", "grape"]
fruits.extend(new_fruits)  # Better than multiple append()

# Good: Don't modify list while iterating
numbers = [1, 2, 3, 4, 5]
# Bad: for num in numbers: numbers.remove(num)
# Good: numbers = [num for num in numbers if condition]
```

Common Patterns

```
# Pattern 1: Processing items with conditions
def process_grades(grades):
    passed = [grade for grade in grades if grade >= 60]
    failed = [grade for grade in grades if grade < 60]
    return passed, failed

# Pattern 2: Grouping items
def group_by_length(words):
    groups = {}
    for word in words:
        length = len(word)
        if length not in groups:
            groups[length] = []
        groups[length].append(word)
    return groups

# Pattern 3: Accumulating results
def running_totals(numbers):
    totals = []
    current_total = 0
    for num in numbers:
        current_total += num
        totals.append(current_total)
    return totals

# Usage examples
grades = [85, 45, 92, 58, 76, 89]
passed, failed = process_grades(grades)

words = ["cat", "dog", "elephant", "ant", "bear"]
grouped = group_by_length(words)

numbers = [1, 2, 3, 4, 5]
totals = running_totals(numbers)  # [1, 3, 6, 10, 15]
```