Python Programming Guide

From Basics to Advanced Concepts

Table of Contents

1. Fundamentals

```
## Variables and Types
name: str = "Alice"  ## Type hinting
age: int = 30        ## Integer
PI: float = 3.14159  ## Float
is_valid: bool = True

## Formatted strings
print(f"{name} is {age} years old")


## Control Flow
numbers = [1, 2, 3, 4, 5]

## List comprehension with condition
squares = [x**2 for x in numbers if x % 2 == 0]
print(squares)  ## [4, 16]
```

2. Data Structures

```
## Advanced Dictionary Usage
from collections import defaultdict

word_counts = defaultdict(int)
for word in ["apple", "banana", "apple"]:
    word_counts[word] += 1
print(word_counts)  ## defaultdict(<class 'int'>, {'apple': 2, 'banana': 1})
```

```
## Named Tuples
from typing import NamedTuple

class Point(NamedTuple):
    x: float
    y: float

p = Point(1.5, 2.5)
print(p.x, p.y)  ## 1.5 2.5
```

3. Functions & Modules

```
## Type Hints and Decorators
from typing import Callable

def log_execution(func: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        print(f"Executing {func.__name__}")
        return func(*args, **kwargs)
    return wrapper

@log_execution
def add_numbers(a: int, b: int) -> int:
    return a + b

print(add_numbers(2, 3))
```

4. OOP Concepts

```
## Class Inheritance and Mixins
class Loggable:
    def log(self, message: str):
        print(f"[{self.__class__.__name__}] {message}")

class Shape(Loggable):
    def area(self) -> float:
        raise NotImplementedError

class Circle(Shape):
    def __init__(self, radius: float):
        self.radius = radius
        self.log("Circle created")

    def area(self) -> float:
        return 3.14 * self.radius ** 2
```

5. Advanced Features

```
## Context Managers
from contextlib import contextmanager

@contextmanager
def timed_operation(name: str):
    import time
    start = time.time()
    try:
        yield
    finally:
        duration = time.time() - start
        print(f"{name} took {duration:.2f} seconds")

with timed_operation("Data Processing"):
    ## Complex operation here
    time.sleep(0.5)


## Generator Expressions
def fibonacci(n: int):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b

print(list(fibonacci(10)))  ## [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

6. Modern Python

```
## Async/Await
import asyncio

async def fetch_data(url: str):
    print(f"Fetching {url}")
    await asyncio.sleep(1)
    return f"Data from {url}"

async def main():
    results = await asyncio.gather(
        fetch_data("https://api.com/1"),
        fetch_data("https://api.com/2")
    )
    print(results)

asyncio.run(main())


## Data Classes
from dataclasses import dataclass

@dataclass
class User:
    name: str
    age: int
    email: str = ""

user = User("Alice", 30)
print(user)  ## User(name='Alice', age=30, email='')
```