Python Data Structures & Object-Oriented Programming (OOP) – Cheat Sheet (Theory Only)

---

1. Sequences in Python

A sequence is an ordered collection of elements that supports indexing and slicing.

Types of sequences:

- **Lists** → Mutable ([1, 2, 3])
- **Tuples** → Immutable ((1, 2, 3))
- **Strings** → Immutable ("hello")
- **Ranges** → Immutable (range(5) → [0,1,2,3,4])

Example:

```
my_list = [10, 20, 30]
print(my_list[1])  # Output: 20

my_tuple = (5, 10, 15)
print(my_tuple[-1])  # Output: 15
```

Slicing a Sequence:

```
s = "Python"
print(s[1:4])  # Output: yth
```

---

2. Mapping and Sets

Mappings: Dictionaries (dict)

Dictionaries store key-value pairs and are **mutable**.

Keys are unique and must be **immutable** (e.g., int, str, tuple).

Example:

```
student = {"name": "Alice", "age": 21, "course": "CS"}
print(student["name"])  # Output: Alice

# Adding a new key-value pair
student["grade"] = "A"
```

Common Dictionary Methods:

```
print(student.keys())   # Output: dict_keys(['name', 'age', 'course',
'grade'])
print(student.values()) # Output: dict_values(['Alice', 21, 'CS', 'A'])
print(student.items())  # Output: dict_items([('name', 'Alice'), ('age', 21),
('course', 'CS'), ('grade', 'A')])
```

Sets (set)

Unordered collection of unique elements (no duplicates).

Example:

```
my_set = {1, 2, 3, 2, 3}
print(my_set)  # Output: {1, 2, 3} (duplicates removed)
```

Set Operations:

```
A = {1, 2, 3}
B = {3, 4, 5}

print(A | B)  # Union → {1, 2, 3, 4, 5}
print(A & B)  # Intersection → {3}
```

```
print(A - B)  # Difference → {1, 2}
print(A ^ B)  # Symmetric Difference → {1, 2, 4, 5}
```

---

3. Object-Oriented Programming (OOP) in Python

Classes and Instances

A **class** is a blueprint for creating objects.
An **instance** is an actual object created from a class.
Example:

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def details(self):
        return f"{self.brand} {self.model}"

# Creating instances
car1 = Car("Toyota", "Camry")
print(car1.details())  # Output: Toyota Camry
```

Inheritance in Python

Inheritance allows a **child class to inherit attributes and methods** from a **parent class**.
Example:

```
class Animal:
    def speak(self):
        return "Animal Sound"

class Dog(Animal):  # Dog class inherits from Animal
    def speak(self):
        return "Bark"

dog = Dog()
print(dog.speak())  # Output: Bark
```

Types of Inheritance:

- **Single Inheritance** → One parent, one child.
- **Multiple Inheritance** → A child inherits from multiple parents.
- **Multilevel Inheritance** → Inheritance across multiple generations.

---

4. Exception Handling in Python

Exceptions are runtime errors that can be handled using try-except blocks.
Prevents program crashes due to errors like ZeroDivisionError, ValueError, etc.
Example:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")  # Output: Cannot divide by zero
```

Using finally for Cleanup:

```
try:
    file = open("data.txt", "r")
    content = file.read()
```

```
except FileNotFoundError:
    print("File not found")
finally:
    file.close()
```

---

5. Regular Expressions (re module)

Regular expressions (regex) help in pattern matching in strings.
The re module provides functions for regex operations.
Common Regex Functions:

Function	Description
re.match()	Checks if the pattern matches the start of a string
re.search()	Searches for the first occurrence of a pattern
re.findall()	Finds all occurrences of a pattern
re.sub()	Replaces a pattern with another string

Example: Searching for a Word

```
import re

text = "Python is fun"
match = re.search(r"Python", text)
if match:
    print("Match found!")  # Output: Match found!
```

Example: Extracting Digits from a String

```
import re

text = "My number is 12345"
digits = re.findall(r"\d+", text)  # Finds all numbers
print(digits)  # Output: ['12345']
```

Example: Validating an Email Address

```
import re

email = "test@example.com"
pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"

if re.match(pattern, email):
    print("Valid Email")  # Output: Valid Email
else:
    print("Invalid Email")
```

Key Takeaways

Sequences: Ordered data structures (Lists, Tuples, Strings).
Mappings & Sets:

- **Dictionaries store key-value pairs**.
- **Sets store unique elements** and support mathematical operations.

Classes & Instances: Python supports **Object-Oriented Programming (OOP)**.
Inheritance: Allows **code reusability** by extending parent classes.
Exception Handling: try-except blocks prevent **runtime crashes**.
Regular Expressions (re Module): Used for **pattern matching and text manipulation**.

This **Python Programming Cheat Sheet** covers **sequences, mappings, sets, dictionaries, classes, inheritance, exception handling, and regular expressions**. Let me know if you need further explanations!