Python Object-Oriented Programming Cheat Sheet

1. Classes and Objects

Class Definition

python

```python
class Customer:
    def __init__(self, name, email):
        self.__name = name
        self.__email = email

    def get_customer_info(self):
        return f"Name: {self.__name}, Email: {self.__email}"
```

Creating Objects

python

```python
# Create an instance of Customer class
customer1 = Customer("John Doe", "john@example.com")

# Access methods
info = customer1.get_customer_info()
print(info)  # Output: Name: John Doe, Email: john@example.com
```

Class vs Instance Attributes

python

```python
class Counter:
    count1 = 0  # Class attribute (shared by all instances)

    def __init__(self):
        self.count2 = 0  # Instance attribute (unique to each instance)

    def increase_count1(self):
        self.__class__.count1 += 1  # Accessing class attribute

    def increase_count2(self):
        self.count2 += 1  # Accessing instance attribute
```

2. Encapsulation

Private Attributes

python

```python
class Person:
    def __init__(self, name, age):
        self.__name = name  # Private attribute (name mangling)
        self.__age = age    # Private attribute
```

Accessor (Getter) and Mutator (Setter) Methods

python

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

    # Getter methods
    def get_name(self):
        return self.__name

    def get_age(self):
        return self.__age

    # Setter methods
    def set_name(self, name):
        self.__name = name

    def set_age(self, age):
        if age > 0:  # Validation
            self.__age = age
        else:
            print("Age must be positive")
```

3. Abstraction

Hiding Implementation Details

python

```python
class DatabaseConnection:
    def __init__(self, host, username, password):
        self.__host = host
        self.__username = username
        self.__password = password
        self.__connection = None

    def connect(self):
        # Abstract away complex connection logic
        print(f"Connecting to {self.__host}...")
        self.__connection = "Connected"

    def execute_query(self, query):
        # User doesn't need to know how queries are executed
        print(f"Executing: {query}")
        return "Results"
```

4. Inheritance

Basic Inheritance

python

```python
# Base/Parent class
class Vehicle:
    def __init__(self, make, model, mileage, price):
        self.__make = make
        self.__model = model
        self.__mileage = mileage
        self.__price = price

    def get_make(self):
        return self.__make

    def get_price(self):
        return self.__price

# Derived/Child class
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, doors):
        # Call the parent class initializer
        Vehicle.__init__(self, make, model, mileage, price)
        # Initialize child-specific attribute
        self.__doors = doors

    def get_doors(self):
        return self.__doors
```

Using super()

python

```python
class Car(Vehicle):
    def __init__(self, make, model, mileage, price, doors):
        # Call the parent class initializer using super()
        super().__init__(make, model, mileage, price)
        self.__doors = doors
```

Method Overriding

python

```python
class Animal:
    def __init__(self, species):
        self.__species = species

    def show_species(self):
        print(f"I am a {self.__species}")

    def make_sound(self):
        print("Grrrr")

class Cat(Animal):
    def __init__(self):
        super().__init__("Cat")

    # Override the parent method
    def make_sound(self):
        print("Meow!")
```

5. Polymorphism

Same Interface, Different Implementation

python

```python
class Animal:
    def make_sound(self):
        pass

class Dog(Animal):
    def make_sound(self):
        return "Woof!"

class Cat(Animal):
    def make_sound(self):
        return "Meow!"

class Cow(Animal):
    def make_sound(self):
        return "Moo!"

# Polymorphic function
def animal_sound(animal):
    print(animal.make_sound())

# Creating objects
dog = Dog()
cat = Cat()
cow = Cow()

# Same function call, different behavior
animal_sound(dog)  # Output: Woof!
animal_sound(cat)  # Output: Meow!
animal_sound(cow)  # Output: Moo!
```

Type Checking

python

```python
def show_animal_info(creature):
    if isinstance(creature, Animal):
        creature.show_species()
        creature.make_sound()
    else:
        print("This is not an animal!")
```

6. Special Methods

str Method (String Representation)

python

```python
class Customer:
    def __init__(self, name, email):
        self.__name = name
        self.__email = email

    def __str__(self):
        return f"Name: {self.__name}, Email: {self.__email}"

# Now print(customer) will use the __str__ method
customer = Customer("John", "john@example.com")
print(customer)  # Output: Name: John, Email: john@example.com
```

7. Modules

Storing Classes in Modules

python

```python
# Save in file Customer.py
class Customer:
    def __init__(self, name, email):
        self.__name = name
        self.__email = email

    def get_customer_info(self):
        return f"Name: {self.__name}, Email: {self.__email}"
```

Importing Modules

python

```python
# Option 1: Import the module
import Customer
customer = Customer.Customer("John", "john@example.com")

# Option 2: Import specific class
from Customer import Customer
customer = Customer("John", "john@example.com")

# Option 3: Import with alias
import Customer as cust
customer = cust.Customer("John", "john@example.com")
```

8. Exception Handling

Basic Exception Handling

python

```python
def divide_numbers():
    try:
        num1 = int(input("Enter a number: "))
        num2 = int(input("Enter another number: "))
        result = num1 / num2
        print(f"{num1} divided by {num2} is {result}")
    except ValueError:
        print("Please enter valid numbers")
    except ZeroDivisionError:
        print("Cannot divide by zero")
```

Try-Except-Else-Finally

python

```python
def read_file(filename):
    try:
        file = open(filename, 'r')
        content = file.read()
    except IOError:
        print(f"Could not read file {filename}")
        return None
    else:
        print("File read successfully")
        return content
    finally:
        try:
            file.close()
            print("File closed")
        except:
            pass  # If file wasn't opened successfully
```

Catching All Exceptions

python

```python
try:
    # Code that might raise an exception
    pass
except IOError:
    # Handle IO errors specifically
    pass
except:
    # Handle all other exceptions
    print("An unknown error occurred")
```

9. File I/O

Writing to a File

python

```python
def write_data(filename, data):
    try:
        file = open(filename, 'w')
        file.write(data)
        file.close()
        print(f"Data written to {filename}")
    except IOError:
        print(f"Error writing to {filename}")
```

Reading from a File

python

```python
def read_data(filename):
    try:
        file = open(filename, 'r')
        data = file.read()
        file.close()
        return data
    except IOError:
        print(f"Error reading from {filename}")
        return None
```

Reading Line by Line

python

```python
def read_lines(filename):
    lines = []
    try:
        file = open(filename, 'r')
        line = file.readline()

        while line != '':
            lines.append(line.strip())  # strip() removes newline characters
            line = file.readline()

        file.close()
        return lines
    except IOError:
        print(f"Error reading from {filename}")
        return None
```

Using for Loop to Read Lines

python

```python
def read_lines_with_for(filename):
    lines = []
    try:
        file = open(filename, 'r')
        for line in file:
            lines.append(line.strip())
        file.close()
        return lines
    except IOError:
        print(f"Error reading from {filename}")
        return None
```

10. Persistence with Shelve

Saving Objects to Shelve

python

```python
import shelve

class Student:
    def __init__(self, admin_no, gpa):
        self.__admin_no = admin_no
        self.__gpa = gpa

    def get_admin_no(self):
        return self.__admin_no

    def get_gpa(self):
        return self.__gpa

    def set_gpa(self, gpa):
        self.__gpa = gpa

# Create some students
student1 = Student("S1001", 3.5)
student2 = Student("S1002", 4.0)

# Open or create a shelf
students_dict = {}
db = shelve.open('students.db', 'c')

try:
    # Check if we already have data
    if 'students' in db:
        students_dict = db['students']

    # Add students to dictionary
    students_dict[student1.get_admin_no()] = student1
    students_dict[student2.get_admin_no()] = student2

    # Save back to shelve
    db['students'] = students_dict
except:
    print("Error working with shelf file")
finally:
    db.close()  # Make sure to close the shelf
```

Retrieving Objects from Shelve

python

```python
import shelve

db = shelve.open('students.db', 'r')
try:
    if 'students' in db:
        students_dict = db['students']

        # Access a specific student
        if 'S1001' in students_dict:
            student = students_dict['S1001']
            print(f"Student ID: {student.get_admin_no()}, GPA: {student.get_gpa()}")

        # Loop through all students
        for admin_no, student in students_dict.items():
            print(f"Student ID: {admin_no}, GPA: {student.get_gpa()}")
except:
    print("Error reading from shelf file")
finally:
    db.close()  # Make sure to close the shelf
```

11. Design Patterns: Is-A vs Has-A Relationships

Is-A Relationship (Inheritance)

python

```python
# A Car IS-A Vehicle
class Vehicle:
    def __init__(self, make):
        self.__make = make

class Car(Vehicle):
    def __init__(self, make, model):
        super().__init__(make)
        self.__model = model
```

Has-A Relationship (Composition)

python

```python
# A Car HAS-A Engine
class Engine:
    def __init__(self, horsepower):
        self.__horsepower = horsepower

    def start(self):
        print("Engine started")

class Car:
    def __init__(self, make, model, engine_hp):
        self.__make = make
        self.__model = model
        self.__engine = Engine(engine_hp)  # Composition

    def start_engine(self):
        print(f"Starting {self.__make} {self.__model}")
        self.__engine.start()
```