Python: Day 03

Object-Oriented Programming

Previous Agenda

01

Lists & Tuple

Ordered Group

02

Dictionary & Set

Unordered Group

03

String

Handling Text

04

Comprehension

Iteration Shortcut

05

File Handling

Data outside code

06

Lab Session

Culminating Exercise

Agenda

01

Definition

Data-Centric Approach

02

Relationship

Code Reuse

03

Structure

Code Architecture

04

GUI

Introduction to Tkinter

05

Lab Session

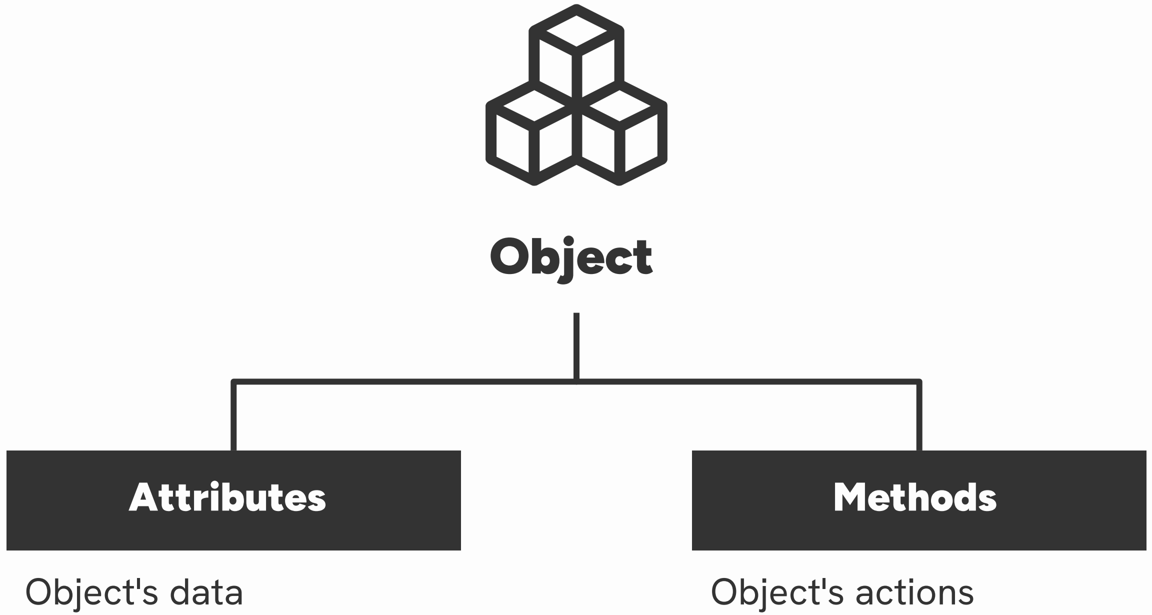
Culminating Exercise

01

Definition

Programming with a focus on concepts

What makes something something ?



Has+Can → Is

Functional Identity

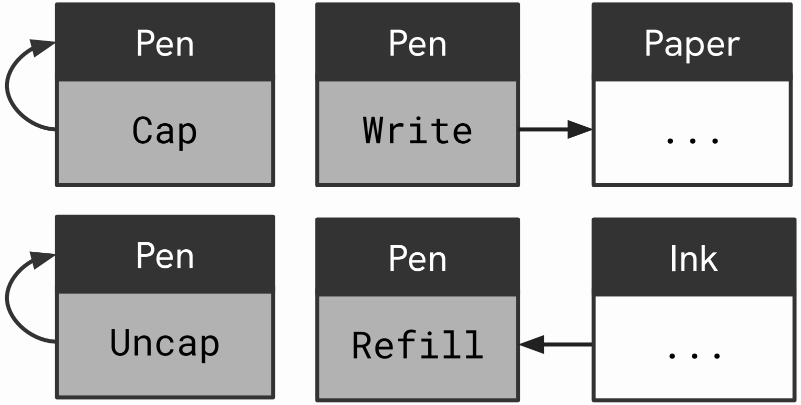
Attributes

- Attributes are unique to one object

Pen	
brand	Pilot
color	Black
capped	False

Methods

- Methods can change itself or others



Object Similarities

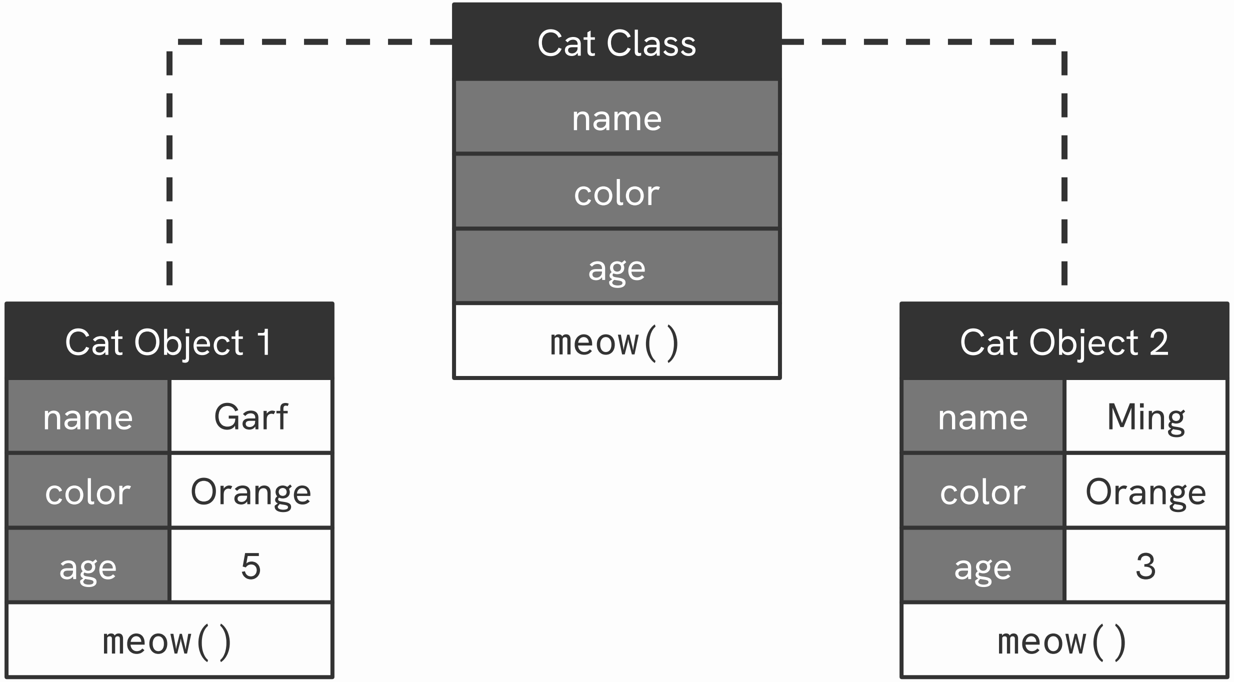
cat1	
name	Garf
color	Orange
age	5
meow	

cat2	
name	Ming
color	Orange
age	3
meow	

cat3	
name	Mona
color	Black
age	2
meow	

What makes them different/same?

Classes to Objects



Classes as Requirements

	Cat Class
☐	name
☐	color
☐	age
☐	`meow()`

	cat	
🟩	name	Garf
🟩	color	Orange
🟩	age	5
🟩	`meow()`	

	dog	
🟩	name	Cliff
🟩	color	Red
🟩	age	2
🟥	`bark()`	

	tom	
🟩	name	Tom
🟩	color	gray
🟩	age	6
	shows	22
🟩	`meow()`	
	`hammer()`	

Classes as Templates

class Package	
ID	
Description	
Address	

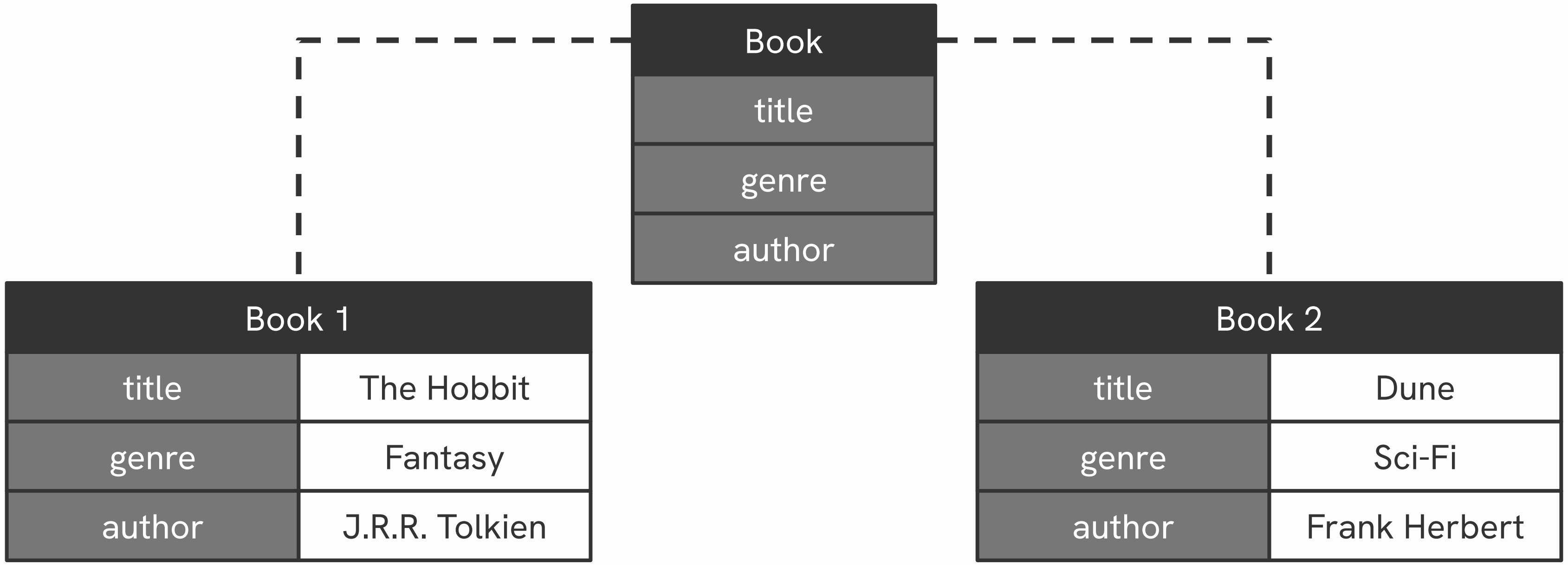
Package	
Id	1231
Description	Cup Noodles
Address	Tokyo, Japan

Package	
Id	11211
Description	Candy
Address	Manila, Philippines

Modelling Exercise

BPI
OPEN
WELCOME
BPI
Makati Main

Book

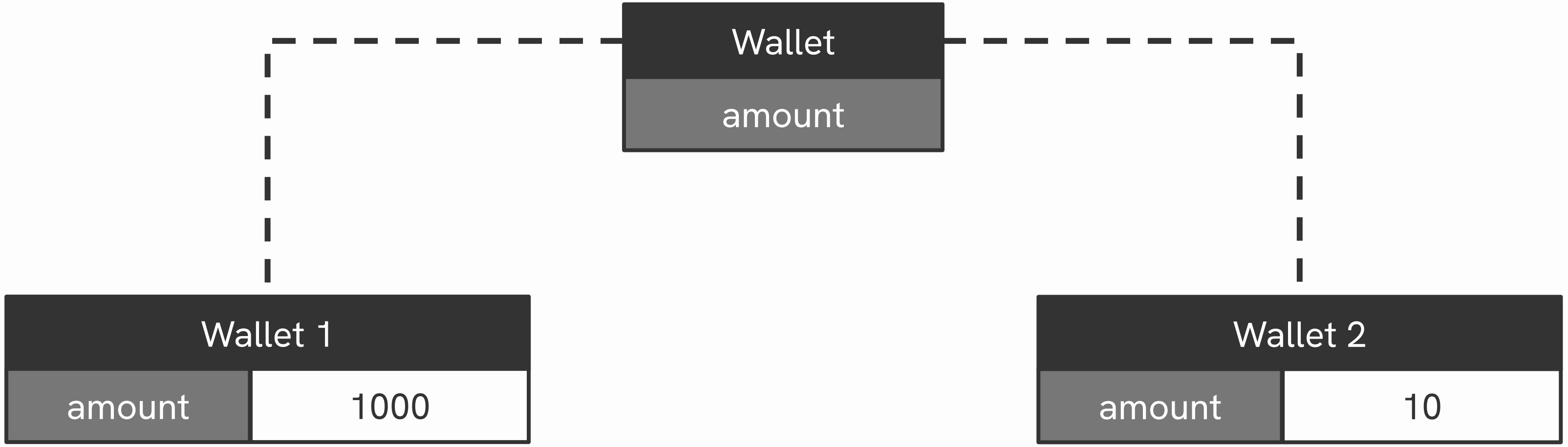


Book
title
genre
author

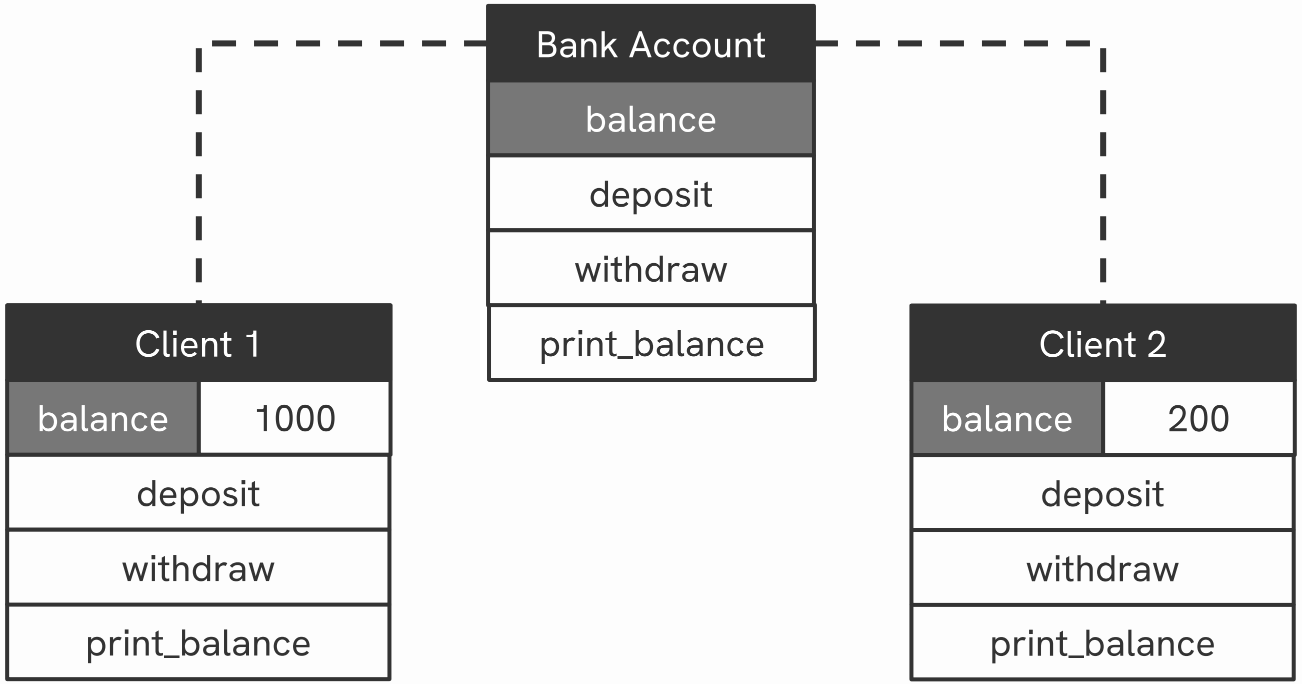
Book 1	
title	The Hobbit
genre	Fantasy
author	J.R.R. Tolkien

Book 2	
title	Dune
genre	Sci-Fi
author	Frank Herbert

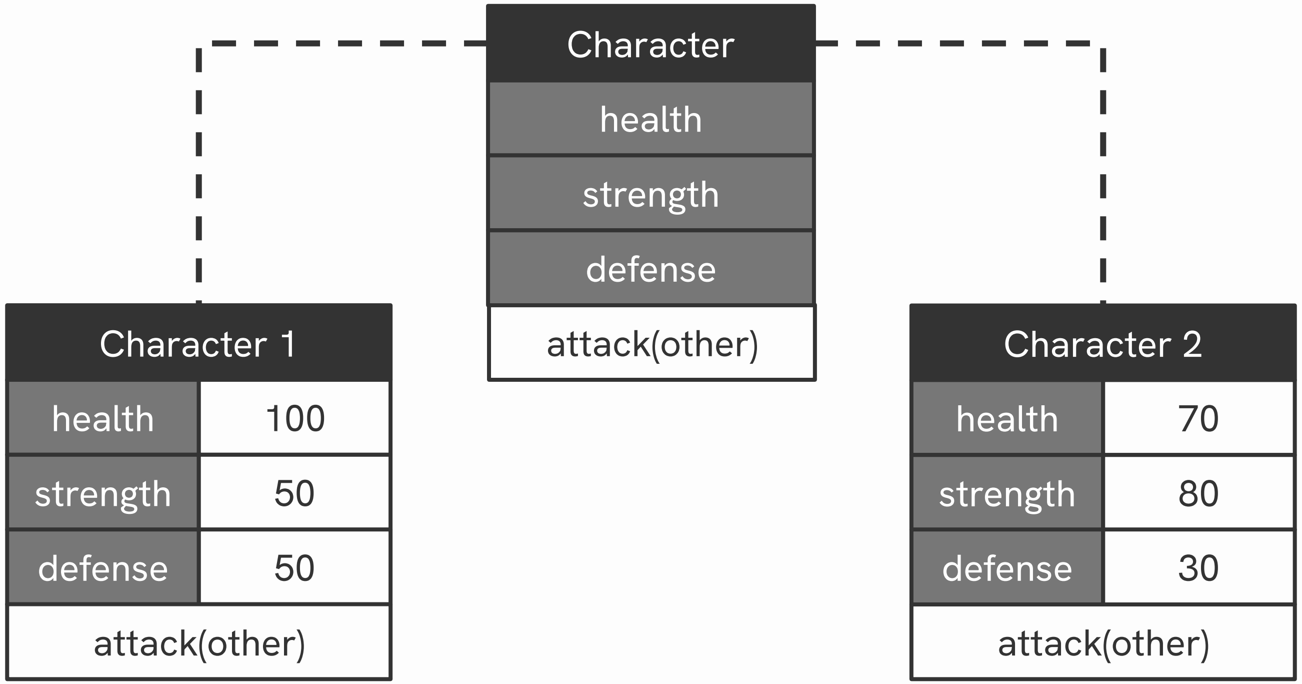
Wallet

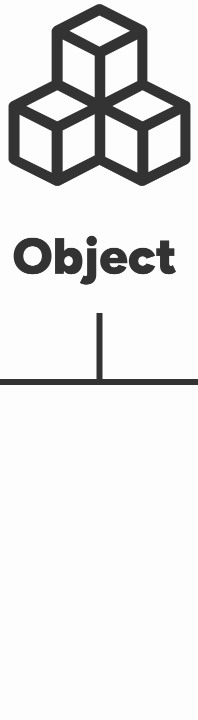


Bank Account



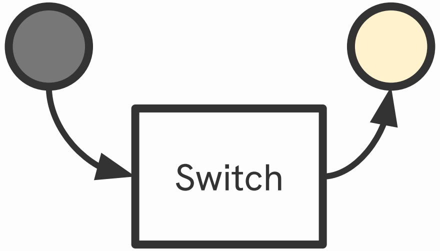
Game Character




Object
Attributes
Methods
Variables
Functions

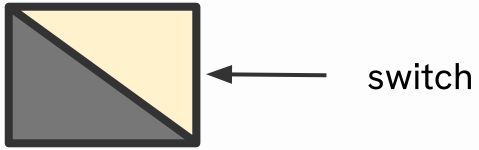
Functional Approach

```
light_status = switch(light_status)
print(light_status)
```



OOP Approach

```
light.switch()
print(light.status)
```



Building Exercise

Example Class

employee.py

```python
class Employee:
    pass
```

Object Creation

employee.py

```python
class Employee:
    pass

employee1 = Employee()
```

Object Attribute Write

employee.py

```python
class Employee:
    pass

employee1 = Employee()
employee1.name = "Richard"
```

Object Attribute Read

employee.py

```
class Employee:
    pass

employee1 = Employee()
employee1.name = "Richard"
print(employee1.name)
```

Object Attributes

employee.py

```
class Employee:
    pass

employee1 = Employee()
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)
```

Multiple Objects

employee.py

```
class Employee:
    pass

employee1 = Employee()
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)

employee2 = Employee()
employee2.name = "Jelly"
employee2.id = "9876"
print(employee2.name, employee2.id)
```

Class Constructor

employee.py

```python
class Employee:
    def __init__(self):
        print("Employee created")

employee1 = Employee()
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)

employee2 = Employee()
employee2.name = "Jelly"
employee2.id = "9876"
print(employee2.name, employee2.id)
```

Constructor Parameter

employee.py

```python
class Employee:
    def __init__(self, name):
        print(f"Employee {name} created")

employee1 = Employee("Richard")
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)

employee2 = Employee("Jelly")
employee2.name = "Jelly"
employee2.id = "9876"
print(employee2.name, employee2.id)
```

Constructor Parameters

employee.py

```python
class Employee:
    def __init__(self, name, id):
        print(f"Employee {name} created with ID {id}")

employee1 = Employee("Richard", "1234")
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)

employee2 = Employee("Jelly", "9876")
employee2.name = "Jelly"
employee2.id = "9876"
print(employee2.name, employee2.id)


```

Object Attributes

employee.py

```python
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id
        print(f"Employee {name} created with ID {id}")

employee1 = Employee("Richard", "1234")
print(employee1.name, employee1.id)

employee2 = Employee("Jelly", "9876")
print(employee2.name, employee2.id)
```

Constructor

```python
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id

employee1 = Employee("Richard", "1234")
print(employee1.name, employee1.id)
```

```python
class Employee:
    pass

employee1 = Employee()
employee1.name = "Richard"
employee1.id = "1234"
print(employee1.name, employee1.id)
```

Object Attributes

self .name

employee1 .name

Methods

employee.py

```python
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id
        print(f"Employee {name} created with ID {id}")

    def work(self):
        print(f"Working...")

employee1 = Employee("Richard", "1234")
employee2 = Employee("Jelly", "9876")

employee1.work()
```

Method Parameter

employee.py

```python
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id
        print(f"Employee {name} created with ID {id}")

    def work(self, task):
        print(f"Working {task}...")

employee1 = Employee("Richard", "1234")
employee2 = Employee("Jelly", "9876")

employee1.work("Create Slides")
```

Object Methods

employee.py

```python
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id
        self.tasks = []
        print(f"Employee {self.name} created with ID {self.id}")

    def work(self, task):
        print(f"Working {task}...")
        self.tasks.append(task)

employee1 = Employee("Richard", "1234")
employee2 = Employee("Jelly", "9876")

employee1.work("Create Slides")
employee2.work("Present Slides")
```

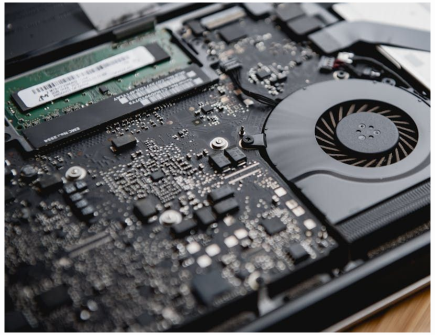
Object Oriented Programming

Tedious Setup

```
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id
        self.tasks = []

    def work(self, task):
        self.tasks.append(task)
```

Easy Usage

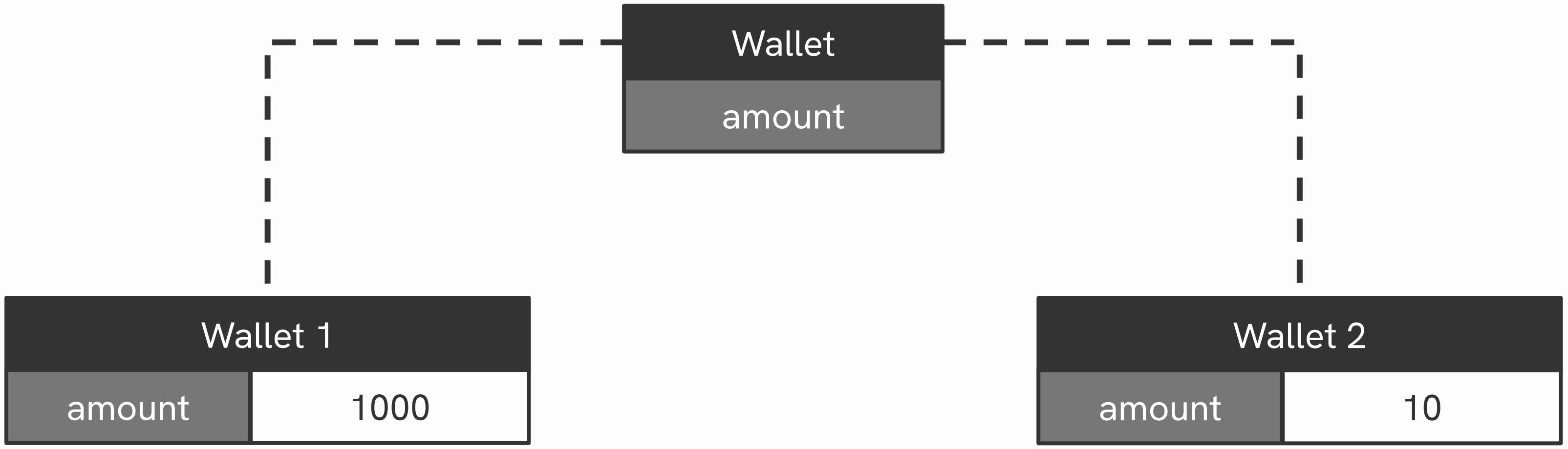
```
employee = Employee("Richard", "1234")
employee.work("Analyze report")
```



H1

Hands-On Building

Wallet



Implement: Wallet

wallet.py

```
class Wallet:
    def __init__(self, initial_amount=0):
        self.amount = initial_amount
```

Implement: Wallet

wallet.py

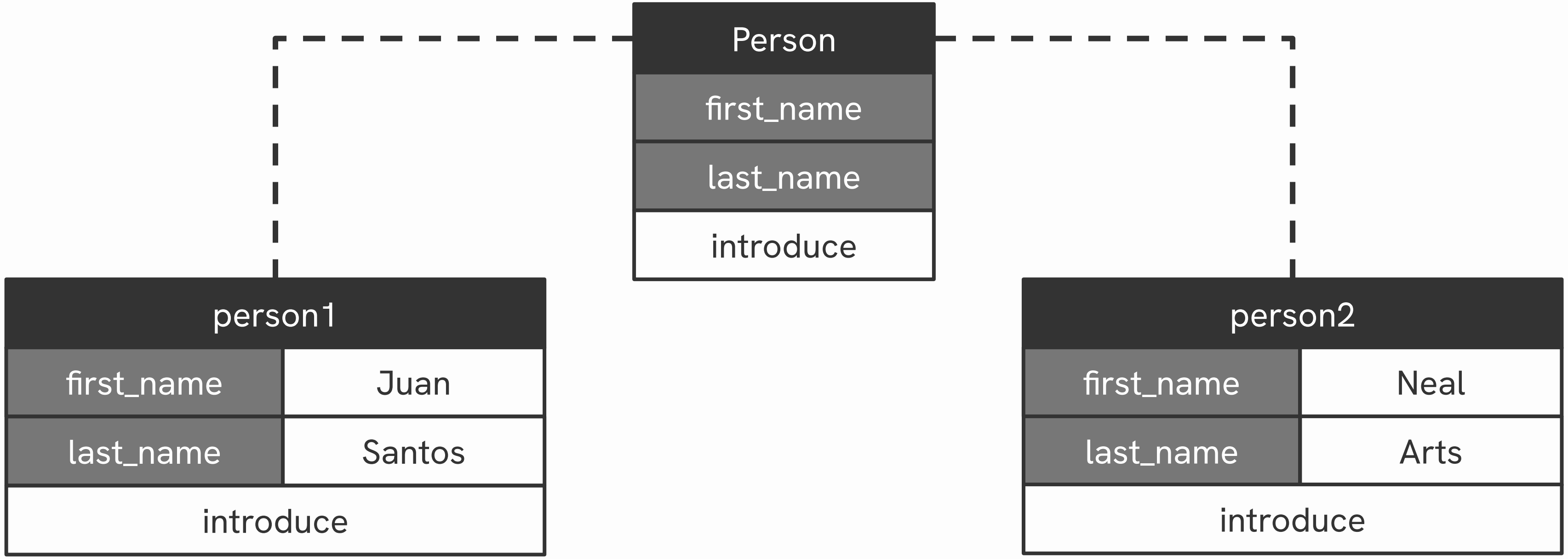
```python
class Wallet:
    def __init__(self, initial_amount=0):
        self.amount = initial_amount

transport_wallet = Wallet(500)
print("Transport Budget:", transport_wallet.amount)
```

Implement: Wallet

wallet.py

```python
class Wallet:
    def __init__(self, initial_amount=0):
        self.amount = initial_amount

transport_wallet = Wallet(500)
print("Transport Budget:", transport_wallet.amount)

food_wallet = Wallet()
food_wallet.amount += 300
print("Food Budget:", food_wallet.amount)
```

Person


Person
first_name
last_name
introduce
person1
first_name
Juan
last_name
Santos
introduce
person2
first_name
Neal
last_name
Arts
introduce

Implement: Person

person.py

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"
```

Implement: Person

person.py

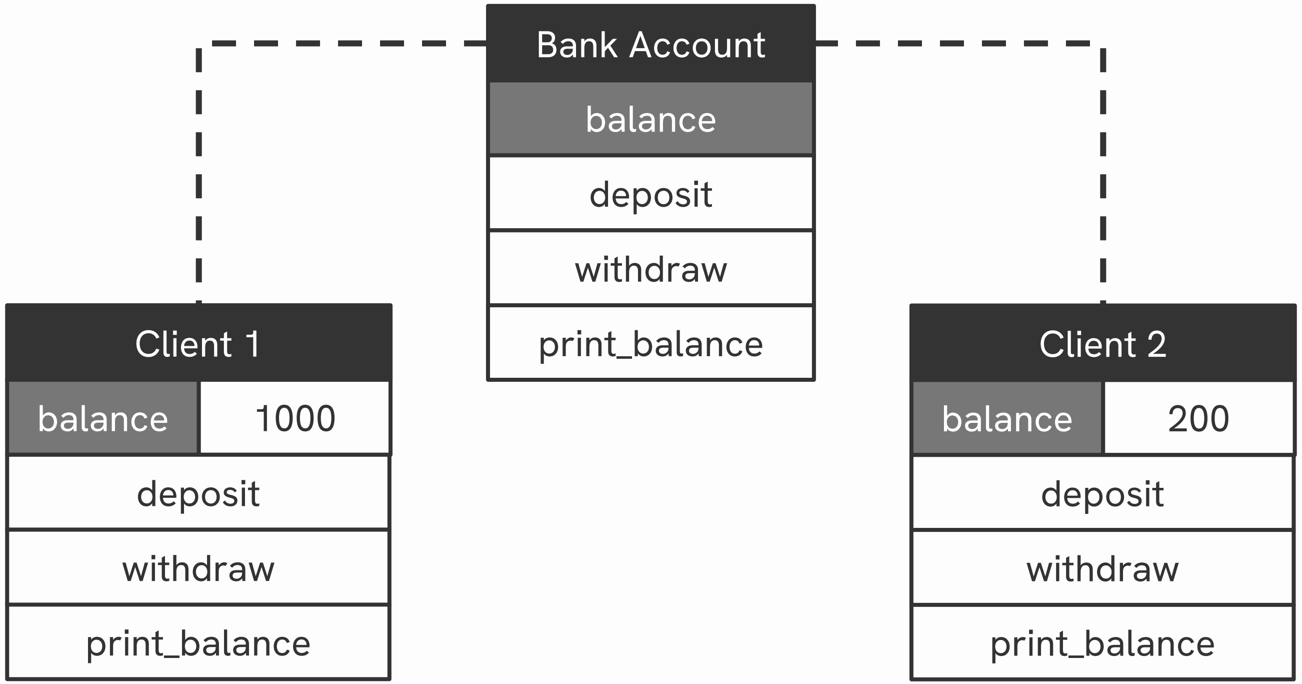
```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

person = Person("Juan", "Miguel")
person.introduce()
```

Implement: Person

person.py

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

person = Person("Juan", "Miguel")
print(person.introduce())
```

Bank Account

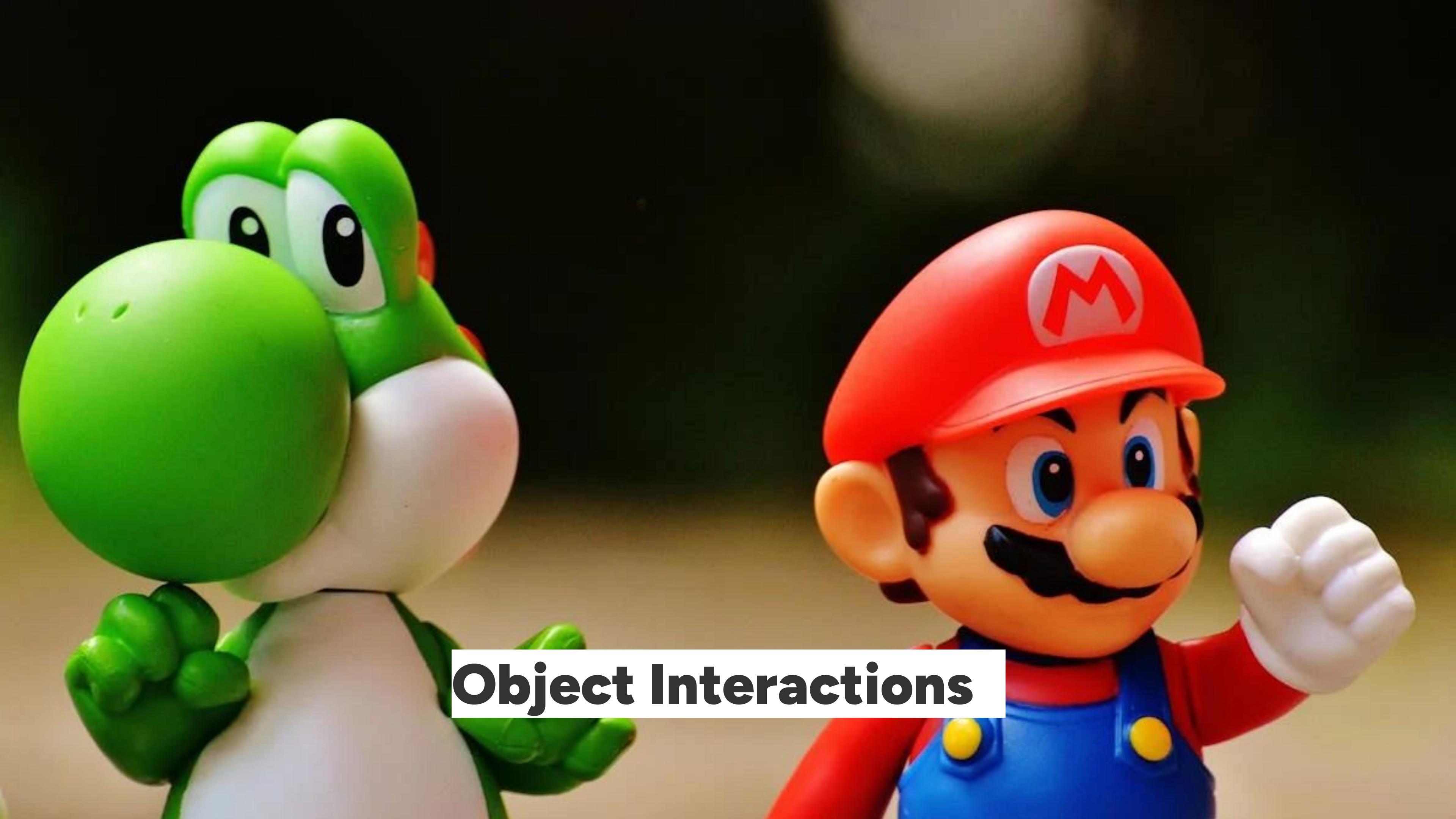


Implement: Bank Account

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def print_balance(self):
        print(self.balance)
```

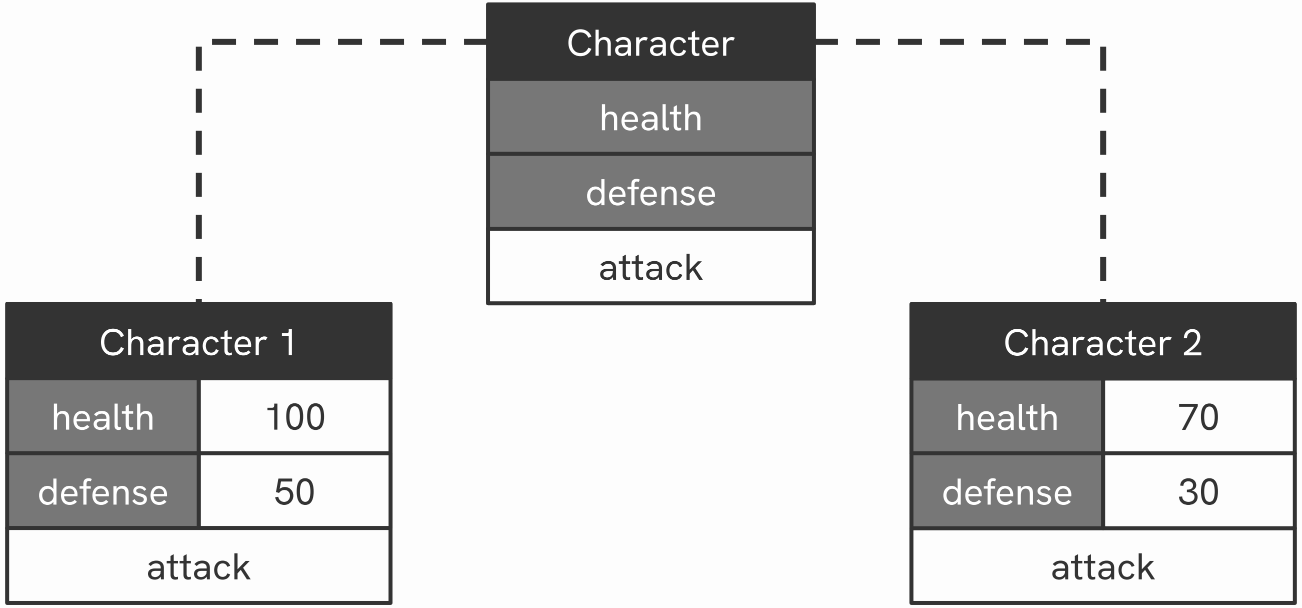
Implement: Bank Account

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def print_balance(self):
        print(self.balance)

account = BankAccount()
account.deposit(1_000)
account.print_balance()
```



Object Interactions

Game Character (Simplified)



Implement: Character

character.py

```python
class Character:
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense

    def attack(self, other):
        damage = 20 - other.defense
        other.health -= damage
```

Implement: Character

character.py

```python
class Character:
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense

    def attack(self, other):
        damage = 20 - other.defense
        other.health -= damage

player = Character()
enemy = Character()

player.attack(enemy)
print(enemy.health)
```

player
attack
enemy

Magic Methods

Magic/Dunder Methods

Dunder methods are special, built-in methods that start and end with dunders (double underscores). Using these methods change or add custom behaviors to classes.

Method Name	Input(s)	Output(s)	Note
__init__	*	None	Sets behavior when creating objects
__repr__	None	String	Used in **`str()`** and **`print()`**
__eq__	Any	Boolean	Sets behavior for **==** operations
__add__	Any	Any	Sets behavior for **+** operations

Implement: Book

```python
class Book:
    def __init__(self, title=None, genre=None, author=None):
        self.title = title
        self.genre = genre
        self.author = author

book = Book("The Hobbit", "Fantasy", "Tolkien")
print(book)
```

```
<__main__.Book object at 0x0000019FE4F27BC0>
```

Implement represent method

```python
class Book:
    def __init__(self, title=None, genre=None, author=None):
        self.title = title
        self.genre = genre
        self.author = author

    def __repr__(self):
        return f"{self.title} - {self.genre} - {self.author}"

book = Book("The Hobbit", "Fantasy", "Tolkien")
print(book)
```

```
The Hobbit - Fantasy - Tolkien
```

Implement: Score

```python
class Score:
    def __init__(self, initial_value=0):
        self.value = initial_value
    def __repr__(self):
        return f"Score: {self.value}"

score1 = Score(20)
score2 = Score(10)
print("Scores:", score1, score2)
```

Implement add method

```python
class Score:
    def __init__(self, initial_value=0):
        self.value = initial_value
    def __repr__(self):
        return f"Score: {self.value}"
    def __add__(self, other):
        return Score(self.value + other.value)

score1 = Score(20)
score2 = Score(10)
print("Scores:", score1, score2)
print("Total:", score1 + score2)
```

Implement greater than method

```python
class Score:
    def __init__(self, initial_value=0):
        self.value = initial_value
    def __repr__(self):
        return f"Score: {self.value}"
    def __add__(self, other):
        return Score(self.value + other.value)
    def __gt__(self, other):
        return self.value > other.value

score1 = Score(20)
score2 = Score(10)
print("Scores:", score1, score2)
print("Total:", score1 + score2)
print("Max Score:", max(score1, score2))
```

Implement: Candy

```python
class Candy:
    def __init__(self, flavor):
        self.flavor = flavor

choco1 = Candy("chocolate")
choco2 = Candy("chocolate")
milk = Candy("milk")

print(choco1 == milk)
print(choco1 == choco2)
```

Implement equality method

```python
class Candy:
    def __init__(self, flavor):
        self.flavor = flavor

    def __eq__(self, other):
        return self.flavor == other.flavor

choco1 = Candy("chocolate")
choco2 = Candy("chocolate")
milk = Candy("milk")

print(choco1 == milk)
print(choco1 == choco2)
```

H2

Hands-Off Building

```python
class CostTracker:
    def __init__(self):
        self items = []
    def spend(self):
        pass
    def spend(self):
        pass
    def spend(self):
        pass
    def spend(self):
        pass
    def mainloop(self):
        pass

cost_tracker = CostTracker()
cost_tracker.mainloop()
```

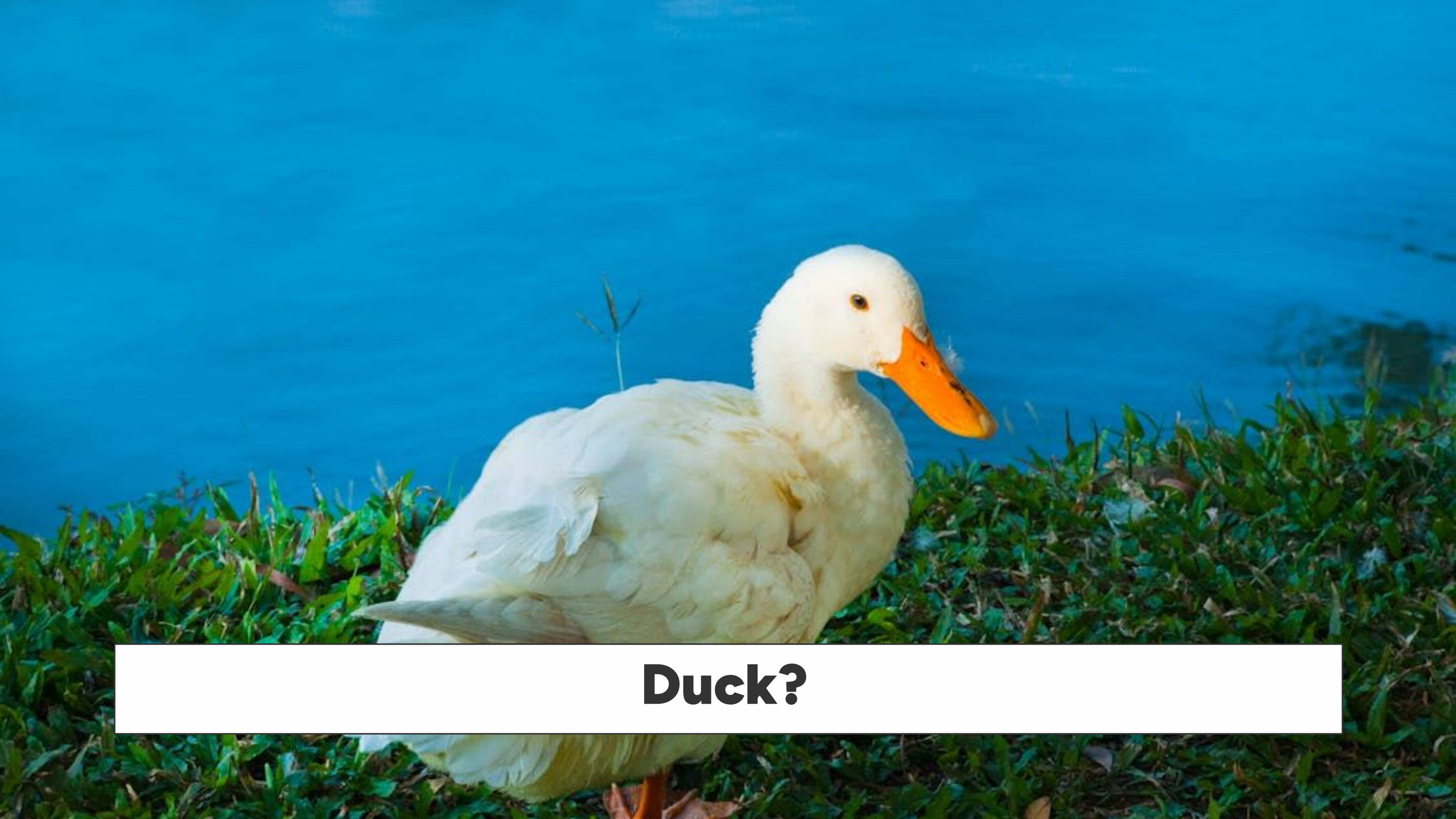
02

Relationship

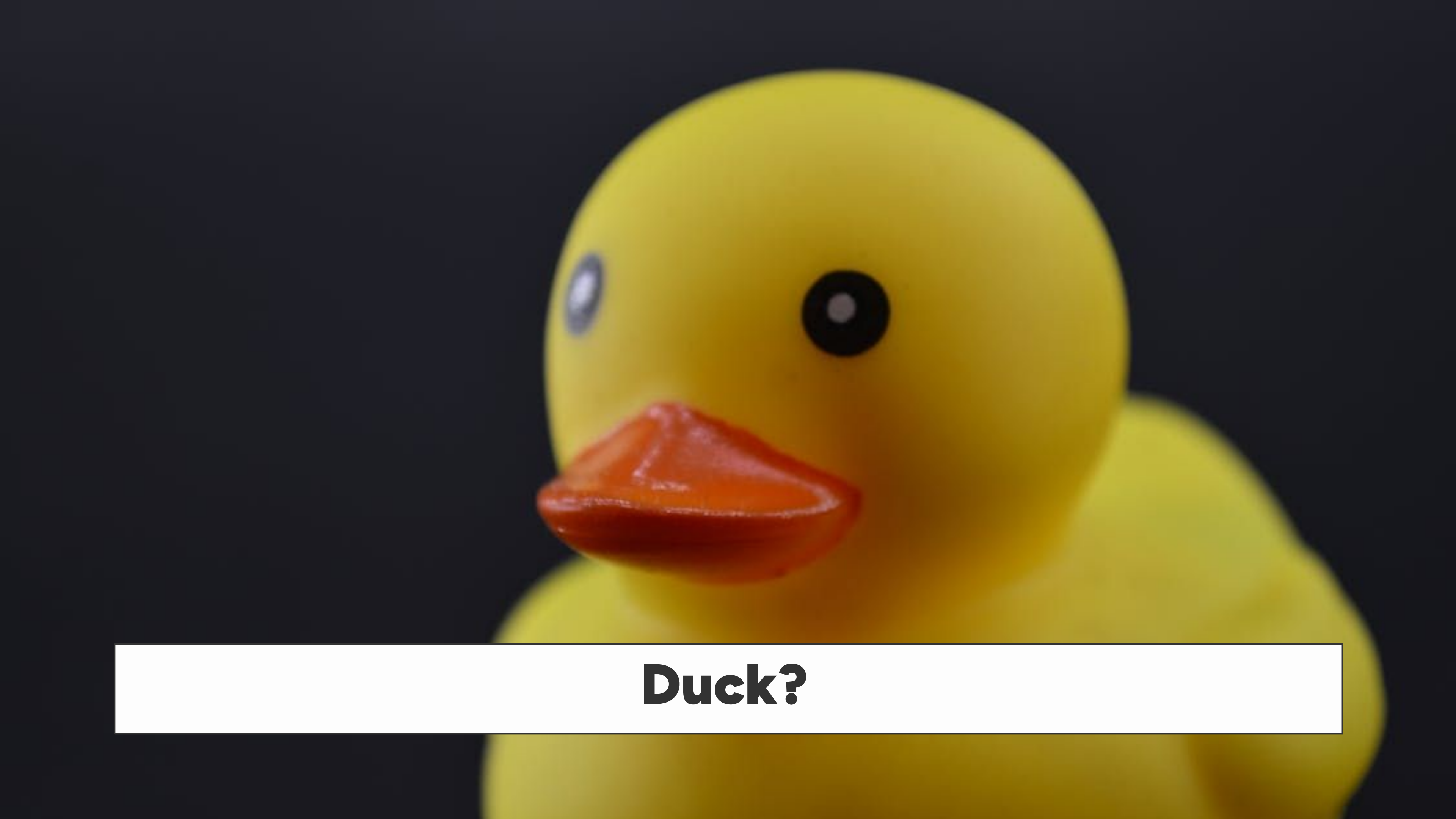
Reducing repetitive data and behavior across classes

Duck Typing

Informal Polymorphism

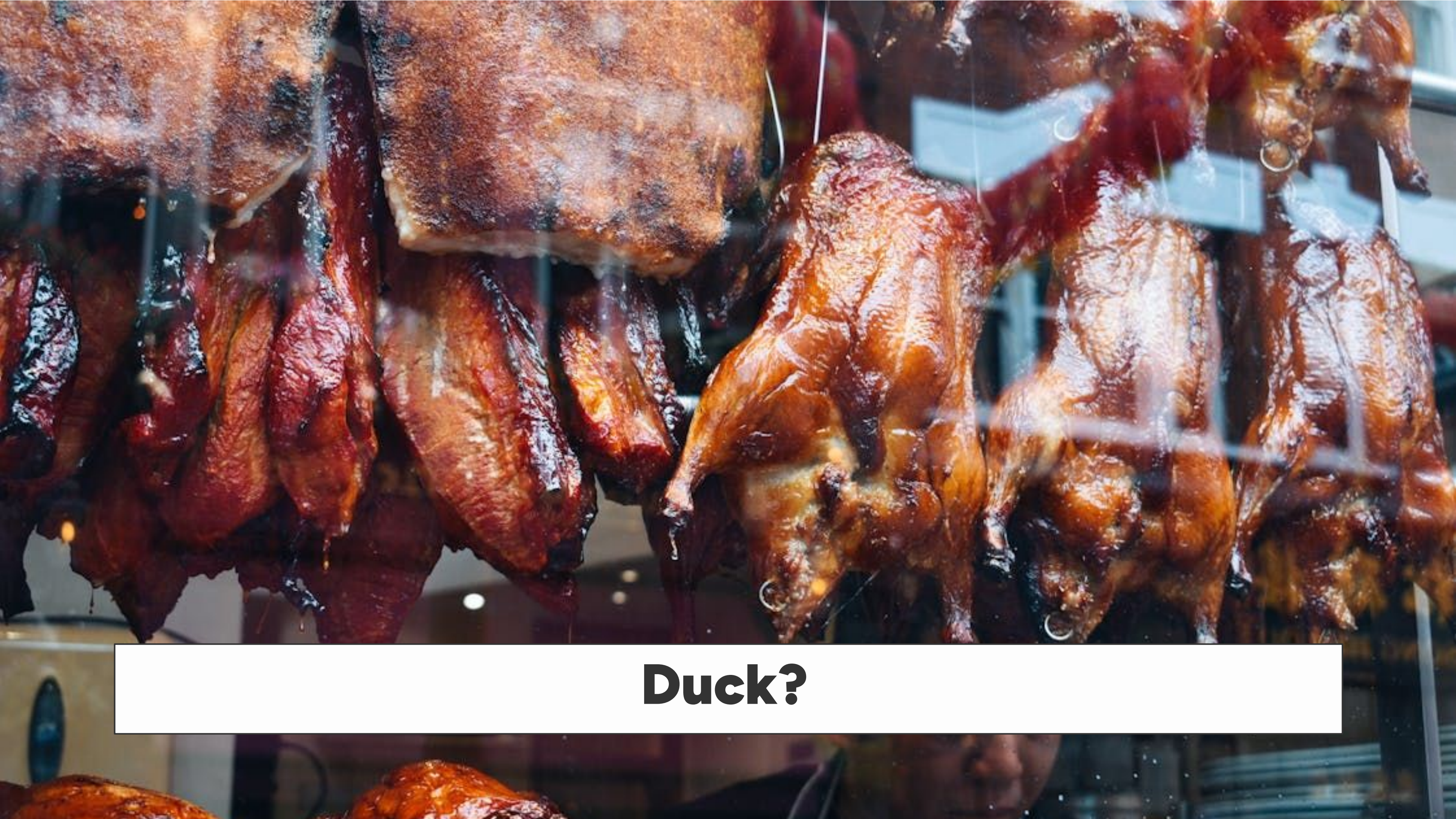


Duck?



Duck?

Duck?



Duck?

""If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.""

—Duck Typing

Has+Can → Is

Implement: Ducks

ducks.py

```
class Duck:
    def __init__(self, beak):
        self.beak = beak
    def swim(self):
        print("Swimming")
    def quack(self):
        print("Quack")
```

```
class RubberDuck:
    def __init__(self, beak):
        self.beak = beak
    def swim(self):
        print("Splish Splosh")
    def quack(self):
        print("Squeak Quack")
```

```
class DuckPerson:
    def __init__(self, beak):
        self.beak = beak
    def swim(self):
        print("Swim hehe!")
    def quack(self):
        print("Quack hehe")
```

```
class RoastedDuck:
    def __init__(self, serving):
        self.serving = serving
```

Informal Polymorphism

Objects demonstrate Informal Polymorphism when they have similar function signatures that can react appropriate for their own type

```
ducks = [
    Duck(beak="Real"),
    RubberDuck(beak="Rubber"),
    DuckPerson(beak="Costume"),
]

for duck in ducks:
    duck.quack()
```

Implement: Knight

character.py

```
class Character:
    ...

class Knight:
    def __init__(self, health=10, defense=10):
        self.health = health
        self.defense = defense
    def attack(self, other):
        damage = self.defense - other.defense
        other.health -= damage

player = Knight(defense=30)
enemy = Character()
player.attack(enemy)
print(enemy.health)
```

Implement: Savers

```
import json

class JSONSaver:
    def save(self, data):
        with open("output.json", "w") as file:
            json.dump(data, file, indent=4)

class TextSaver:
    def save(self, data):
        with open("output.txt", "w") as file:
            for key, value in data.items():
                file.write(f"{key}: {value}\n")

event = {"type": "Error", "message": "server crashed"}
for saver in [JSONSaver(), TextSaver()]:
    saver.save(event)
```

H2

Payment

validation.py

```
class CashPayment():
    def __init__(self, amount):
        self.amount = amount

    def total(self):
        return self.amount

payments = [
    CashPayment(1_000)
]

for payment in payments:
    print(payment.total())
```

```python
class CreditPayment():
    def __init__(self, amount, limit):
        """Set attributes here"""
    def total(self):
        """Raise error if amount is beyond limit"""
```

```python
class OnlinePayment():
    def __init__(self, amount, fee):
        """Set attributes here"""
    def total(self):
        """Return amount + fee"""
```

```python
class DiscountedPayment():
    def __init__(self, amount, discount):
        """Set attributes here"""
    def total(self):
        """Return amount - discount"""
```

Inheritance

Explicit class structure

Code Redundancy

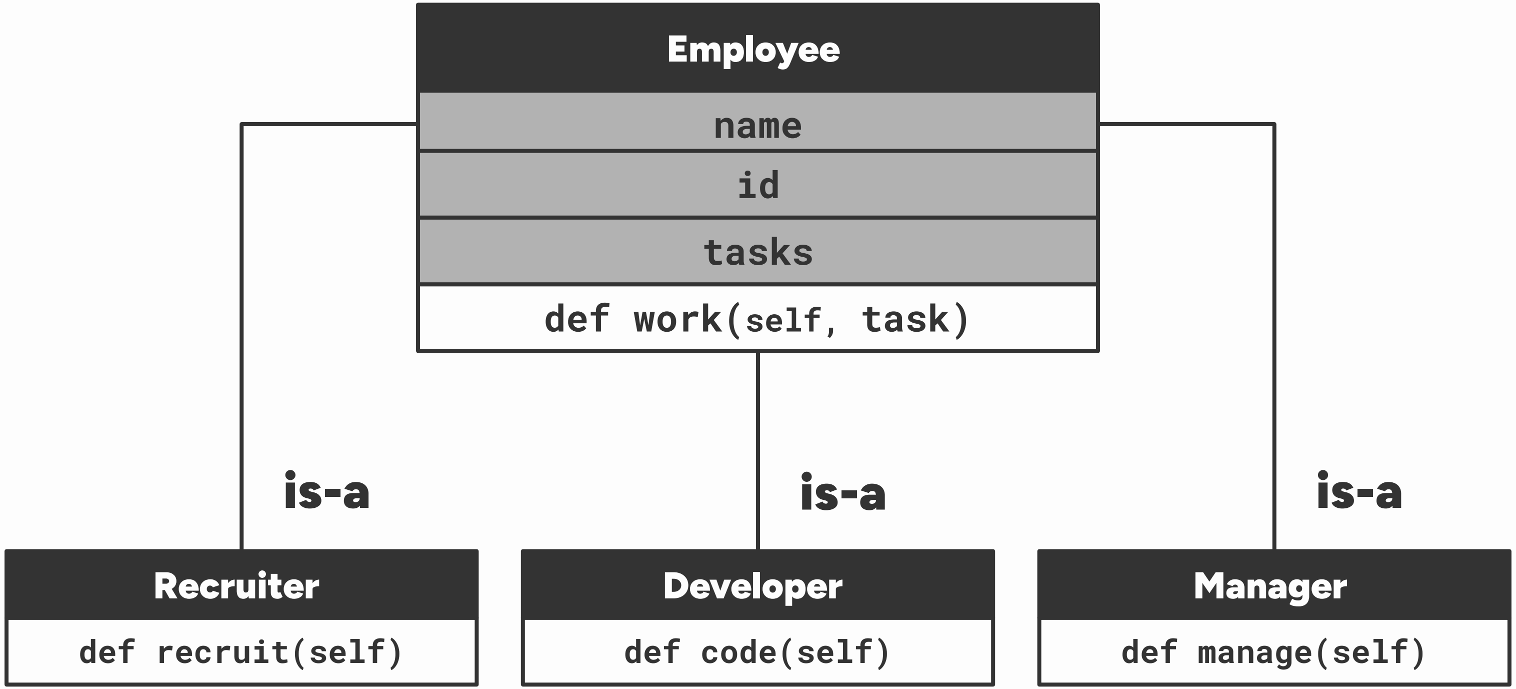
```
class Recruiter:
    def __init__(self, name, id)
    def add_work(self)
    def recruit(self)
```

```
class Manager:
    def __init__(self, name, id)
    def add_work(self)
    def manage(self)
```

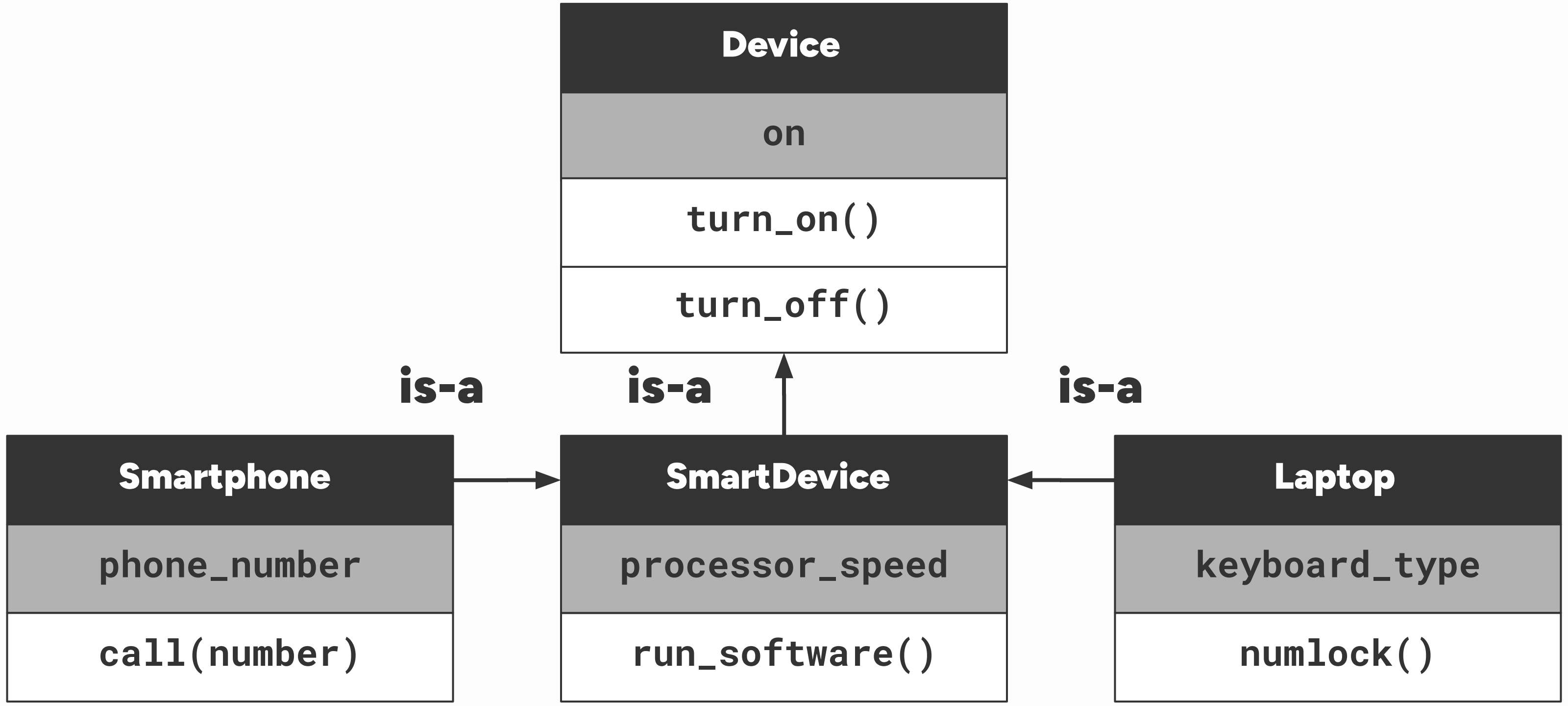
```
class Developer:
    def __init__(self, name, id)
    def add_work(self)
    def code(self)
```

```
class Designer:
    def __init__(self, name, id)
    def add_work(self)
    def design(self)
```

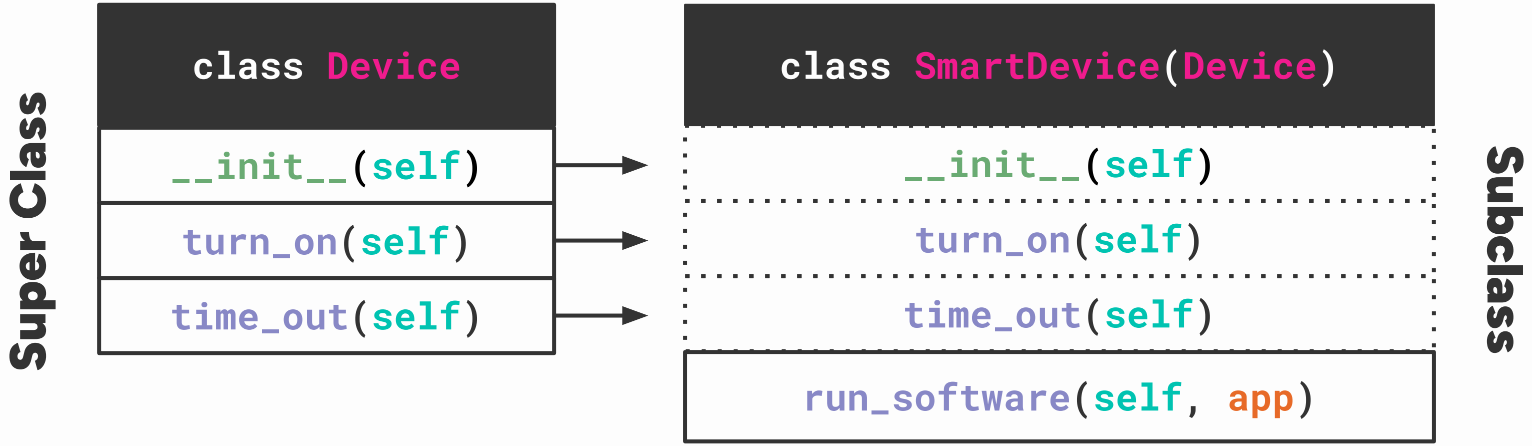
Hierarchy Example



Hierarchy Example 2



Class Inheritance



Student Class

student.py

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    pass
```

Override Methods

student.py

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    def introduce(self):
        return "I'm a student."
```

Override Methods

student.py

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    def introduce(self):
        return super().introduce() + ". " + "I'm a student."
```

Student Class

student.py

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    def __init__(self, level):
        self.level = level

    def introduce(self):
        return super().introduce() + ". " + "I'm a student."
```

Student Class

student.py

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    def __init__(self, first_name, last_name, level):
        self.first_name = first_name
        self.last_name = last_name
        self.level = level

    def introduce(self):
        return super().introduce() + ". " + "I'm a student."
```

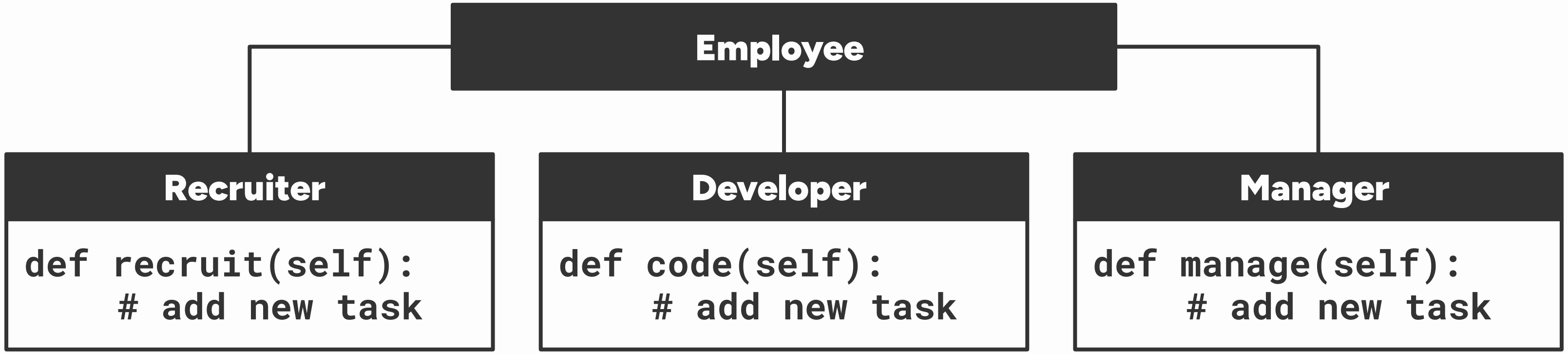
Student Class

student.py

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def introduce(self):
        return f"I'm {self.first_name} {self.last_name}!"

class Student(Person):
    def __init__(self, first_name, last_name, level):
        super().__init__(first_name, last_name)
        self.level = level

    def introduce(self):
        return super().introduce() + ". " + "I'm a student."
```

H3

Employee Chart



```
def recruit(self):
    # add new task
```

```
def code(self):
    # add new task
```

```
def manage(self):
    # add new task
```

03

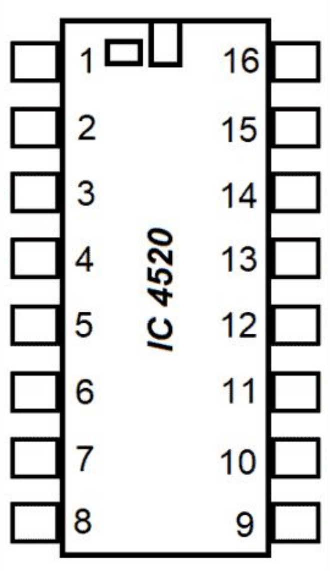
Structure

Designing classes for long term collaboration

Encapsulation

Manage which parts are accessible to the public

Simplification



Security

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def print_balance(self):
        print(self.balance)

bank_account = BankAccount()
bank_account.balance += 999_999_999
```

Public Example

```python
class Counter:
    def __init__(self):
        self.value = 0

counter = Counter()
print("Counter:", counter.value)

counter.value += 1
print("Counter:", counter.value)
```

Protected Example

protected_wallet.py

```python
class ProtectedWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount

budget = Protected Wallet()

budget._amount += 1000
print("Budget:", budget._amount)
```

Protected Example

protected_wallet.py

```python
class ProtectedWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount

    def amount(self):
        return self._amount

budget = ProtectedWallet()

budget._amount += 1000
print("Budget:", budget.amount())
```

Protected Example

protected_wallet.py

```python
class ProtectedWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount

    @property
    def amount(self):
        return self._amount

budget = ProtectedWallet()

budget._amount += 1000
print("Budget:", budget.amount)
```

Protected Example

protected_wallet.py

```python
class ProtectedWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount

    @property
    def amount(self):
        return self._amount
    @amount.setter
    def amount(self, new_amount):
        self._amount = new_amount

budget = ProtectedWallet()

budget.amount += 1000
print("Budget:", budget.amount)
```

protected_wallet.py

```python
class ProtectedWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount

    @property
    def amount(self):
        return self._amount
    @amount.setter
    def amount(self, new_amount):
        if new_amount > 10_000:
            raise ValueError("Amount Too Large")

        self._amount = new_amount

budget = ProtectedWallet()

budget.amount += 1000
print("Budget:", budget.amount)
```

private_wallet.py

```python
class PrivateWallet:
    def __init__(self, initial_amount=0):
        self.__amount = initial_amount

    @property
    def amount(self):
        return self.__amount
    @amount.setter
    def amount(self, new_amount):
        if new_amount > 10_000:
            raise ValueError("Amount Too Large")

        self.__amount = new_amount

budget = PrivateWallet()

budget.amount += 1000
print("Budget:", budget.amount)
```

H4

Safe Banking

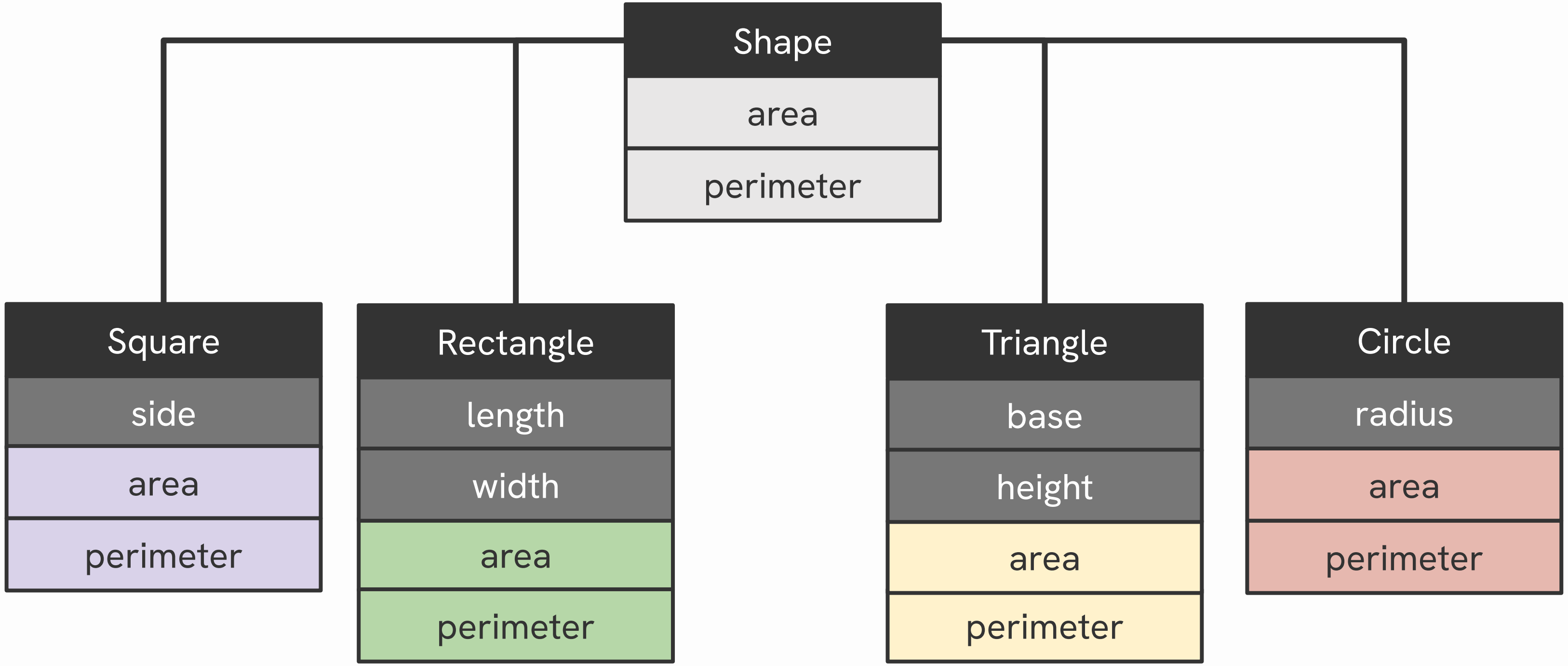
Secure: Bank Account

```python
class BankAccount:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        self.balance -= amount

    def print_balance(self):
        print(self.balance)
```

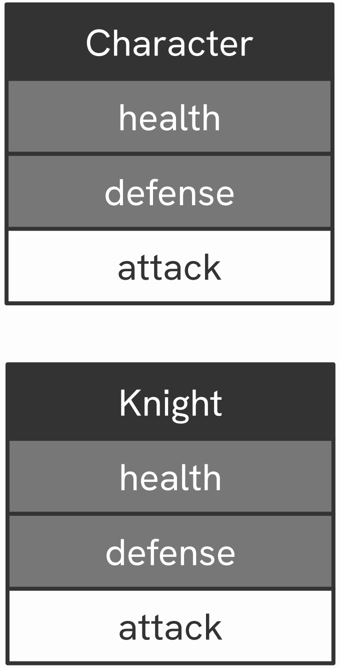
Abstraction

Contractual Implementation

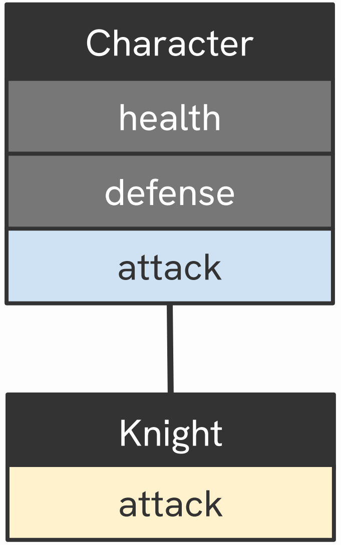
Shapes


Shape
area
perimeter
Square
side
area
perimeter
Rectangle
length
width
area
perimeter
Triangle
base
height
area
perimeter
Circle
radius
area
perimeter

Recall: Game Character



Character Scheme



rpg.py

```python
class Character:
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    def attack(self, other):
        damage = 20 - self.defense
        other.health -= damage

class Knight(Character):
    pass
```

rpg.py

```python
class Character:
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    def attack(self, other):
        damage = 20 - self.defense
        other.health -= damage

class Knight(Character):
    def attack(self, other):
        damage = self.defense - other.defense
        other.health -= damage
```

rpg.py

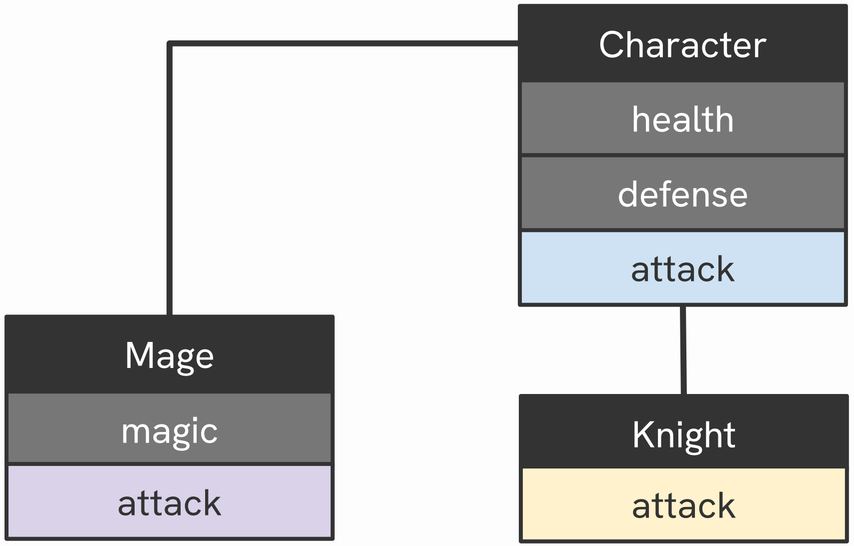
```python
class Character:
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    def attack(self, other):
        raise NotImplementedError()

class Knight(Character):
    def attack(self, other):
        damage = self.defense - other.defense
        other.health -= damage

enemy = Character()
knight = Knight()
knight.attack(enemy)
```

Formal Polymorphism

rpg.py

```
from abc import ABC, abstractmethod

class Character(ABC):
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    @abstractmethod
    def attack(self, other):
        raise NotImplementedError()

class Knight(Character):
    def attack(self, other):
        damage = self.defense - other.defense
        other.health -= damage
```

Character Scheme



rpg.py

```python
from abc import ABC, abstractmethod

class Character(ABC):
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    @abstractmethod
    def attack(self, other):
        raise NotImplementedError()

class Mage(Character):
    def __init__(self, health=100, defense=10, magic=10):
        self.health = health
        self.defense = defense
        self.magic = magic
```

rpg.py

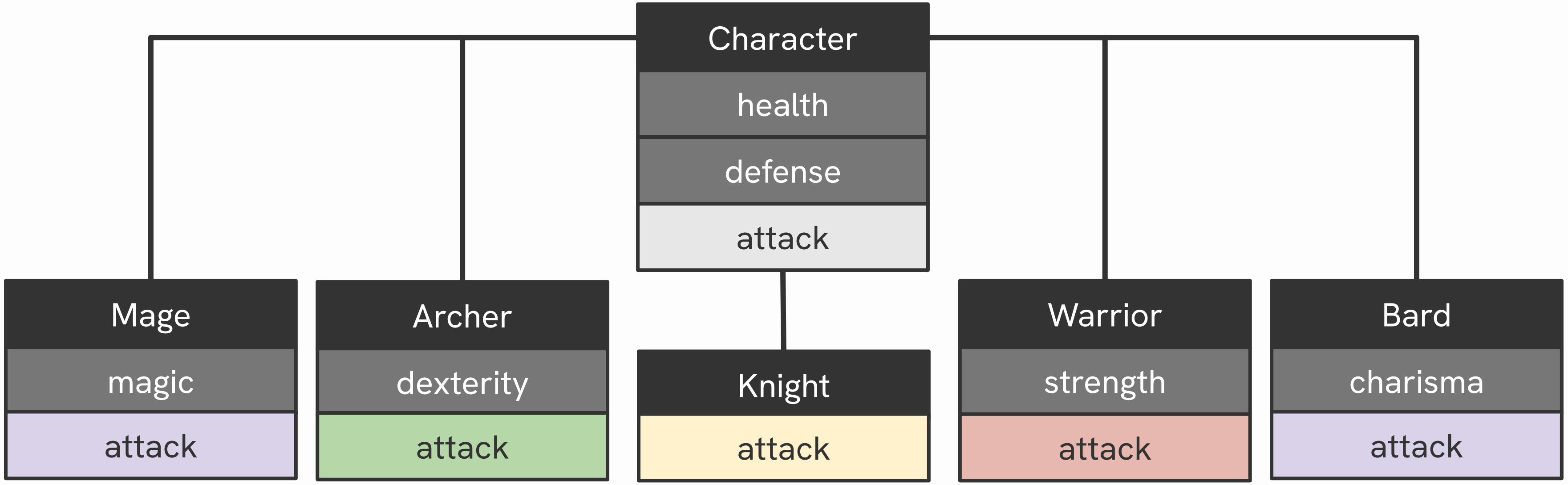
```python
from abc import ABC, abstractmethod

class Character(ABC):
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    @abstractmethod
    def attack(self, other):
        raise NotImplementedError()

class Mage(Character):
    def __init__(self, health=100, defense=10, magic=10):
        self.health = health
        self.defense = defense
        self.magic = magic
    def attack(self, other):
        damage = self.magic - other.defense
        other.health -= damage
```

rpg.py

```python
from abc import ABC, abstractmethod

class Character(ABC):
    def __init__(self, health=100, defense=10):
        self.health = health
        self.defense = defense
    @abstractmethod
    def attack(self, other):
        raise NotImplementedError()

class Mage(Character):
    def __init__(self, health=100, defense=10, magic=10):
        super().__init__(health, defense)
        self.magic = magic
    def attack(self, other):
        damage = self.magic - other.defense
        other.health -= damage
```

H5

Job Tree



`rpg.py`

SOLID Principle

Conceptual Discussion on Design Principles

Single Responsibility Rule

A class should have only one reason to change. It should only have one job or responsibility.

```
class User:
    def __init__(self, name):
        self.name = name

    def save(self):
        print(f"Saving {self.name} to database")

    def send_email(self):
        print(f"Sending email to {self.name}")
```

Single Responsibility Rule

A class should have only one reason to change. It should only have one job or responsibility.

```
class User:
    def __init__(self, name):
        self.name = name

class UserRepository:
    def save(self, user):
        print(f"Saving {user.name} to database")

class EmailService:
    def send_email(self, user):
        print(f"Sending email to {user.name}")
```

Open/Closed Principle

Classes (even functions and modules) should be open for extension but closed for modification

```
class AreaCalculator:
    def calculate_area(self, shape):
        if isinstance(shape, Rectangle):
            return shape.width * shape.height
        elif isinstance(shape, Circle):
            return 3.14 * shape.radius ** 2
```

Open/Closed Principle

Classes (even functions and modules) should be open for extension but closed for modification

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2

class AreaCalculator:
    def calculate_area(self, shape):
        return shape.area()
```

```
class Shape:
    def area(self):
        pass
```

Liskov Substitution Principle

Subclasses must be able to substitute their superclass without issues

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def set_width(self, width):
        self.width = width

    def set_height(self, height):
        self.height = height

    def get_area(self):
        return self.width * self.height
```

```
class Square(Rectangle):
    def __init__(self, side):
        super().__init__(side, side)

    def set_width(self, width):
        self.width = width
        self.height = width

    def set_height(self, height):
        self.height = height
        self.width = height
```

Liskov Substitution Principle

Subclasses must be able to substitute their superclass without issues

```
class Shape:
    def get_area(self):
        pass
```

```
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def get_area(self):
        return self.width * self.height
```

```
class Square(Shape):
    def __init__(self, side):
        self.side = side

    def get_area(self):
        return self.side * self.side
```

Interface Segregation Principle

Subclasses should not be forced to implement methods it doesn’t need

```
class CoffeeMachine:
    def make_espresso(self): pass
    def make_latte(self): pass
    def make_hot_chocolate(self): pass

class EspressoMachine(CoffeeMachine):
    def make_espresso(self):
        print("Espresso ready!")
    def make_latte(self):
        raise Exception("This machine can't make latte")
    def make_hot_chocolate(self):
        raise Exception("This machine can't make hot chocolate")
```

Interface Segregation Principle

Subclasses should not be forced to implement methods it doesn’t need

```
class FancyMachine(
    EspressoMaker,
    LatteMaker,
    HotChocoMaker
):
    def make_espresso(self):
        print("Espresso ready!")
    def make_latte(self):
        print("Latte ready!")
    def make_hot_chocolate(self):
        print("Hot choco ready!")
```

```
class EspressoMaker:
    def make_espresso(self):
        Pass

class LatteMaker:
    def make_latte(self):
        pass

class TeaMaker:
    def make_tea(self):
        pass
```

Dependency Inversion Principle

High-level modules should not depend on low-level modules. Rely on abstractions

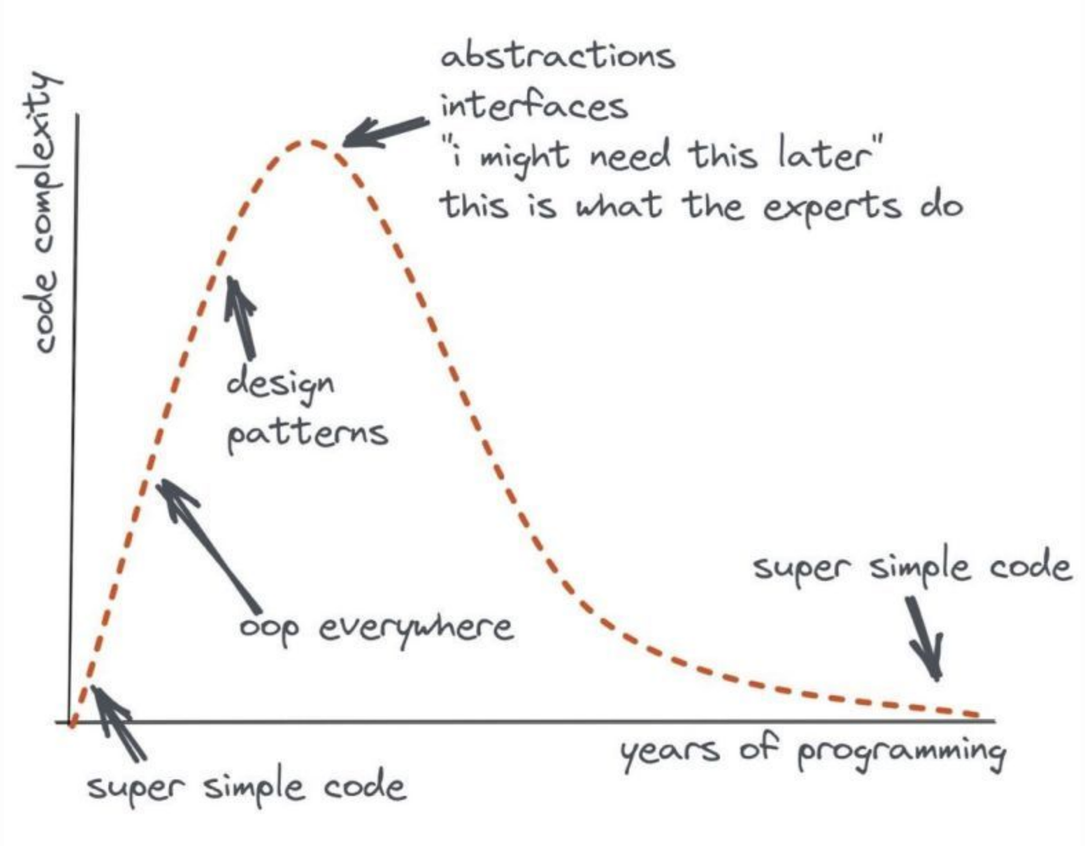
```
class LightBulb:
    def turn_on(self):
        print("Light on")

    def turn_off(self):
        print("Light off")

class LightSwitch:
    def __init__(self, bulb):
        self.bulb = bulb

    def operate(self):
        self.bulb.turn_on()
```

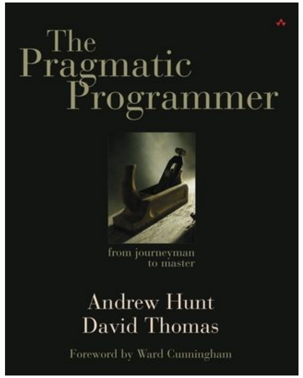
Dependency Inversion Principle

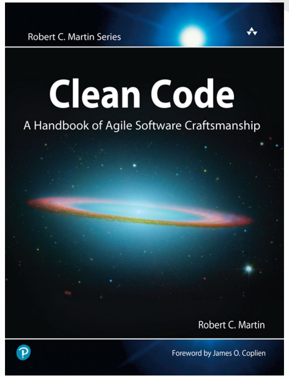
High-level modules should not depend on low-level modules. Rely on abstractions

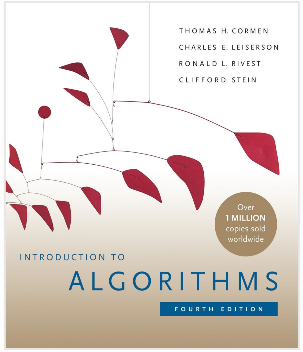
```
class LightSwitch:
    def __init__(self, device):
        self.device = device

    def operate(self):
        self.device.turn_on()
```

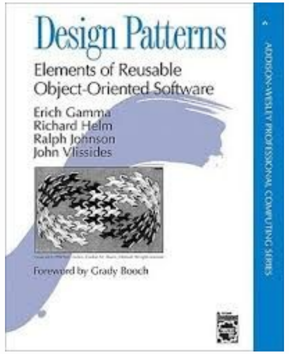
```
class Switchable:
    def turn_on(self):
        pass

    def turn_off(self):
        pass

class LightBulb(Switchable):
    def turn_on(self):
        print("Light on")

    def turn_off(self):
        print("Light off")
```


code complexity
abstractions
interfaces
"i might need this later"
this is what the experts do
design patterns
oop everywhere
super simple code
super simple code
years of programming


The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas
Foreword by Ward Cunningham


Robert C. Martin Series
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
Foreword by James O. Coplien


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
Over 1 MILLION copies sold worldwide
INTRODUCTION TO ALGORITHMS
FOURTH EDITION


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Custom Exception

Create your own errors

Custom Error

custom_error.py

```python
class CustomError(Exception):
    pass

raise CustomError("yikes")
```

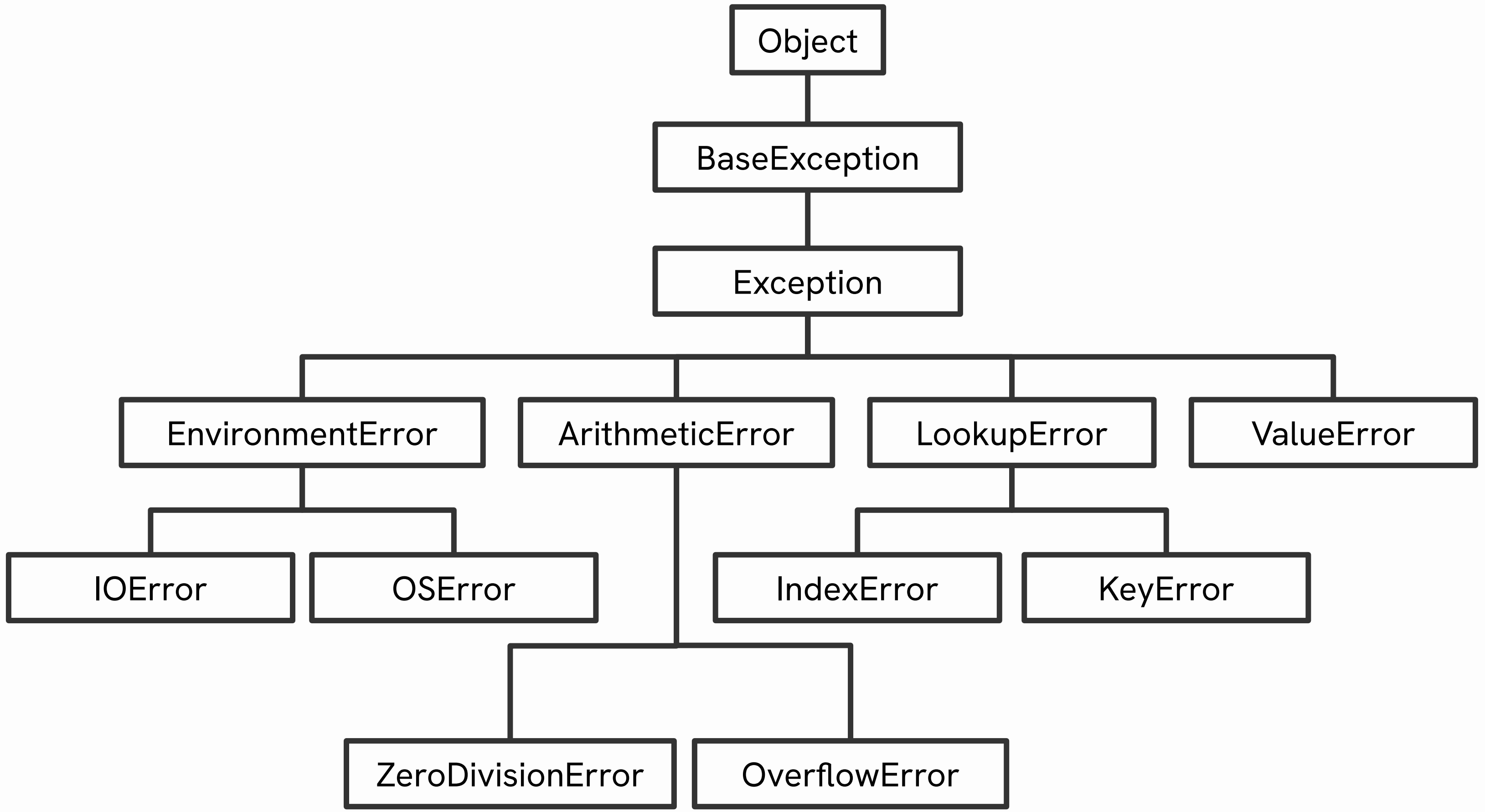
custom_error.py

```python
class CustomError(Exception):
    def __init__(self, message):
        super().__init__(message)

raise CustomError("yikes")
```

Custom Error (Specific)

It is best practice to inherit from the closest existing error class

```python
class InvalidChoiceError(ValueError):
    pass

options = ("rock","paper", "scissors")
user_choice = input("Pick move (rock/paper/scissors): ")

if user_choice not in options:
    raise InvalidChoiceError()
```


Object
BaseException
Exception
EnvironmentError
ArithmeticError
LookupError
ValueError
IOError
OSError
IndexError
KeyError
ZeroDivisionError
OverflowError

Quick Exercise: Number Error

number_error.py

```python
number = input("Enter positive number [1,100]: ")

# If input not a number, raise a custom error
# If input is not positive, raise a custom error
# If input is not between 1 and 100, raise a custom error
```

05

GUI

Graphical User Interface

Python GUI Libraries

Tkinter

Standard GUI toolkit available in (almost) all Python distributions immediately. Easy to understand and great for building simple applications quickly.

PyQt

Python bindings or implementations for the Qt application framework. It has a lot of flexible components and great for building complex applications.

Kivy

Library built specifically for multi-touch platforms (mobile) but can be used in Desktops as well. Good for complex, cross-platform applications.

Window

hello_world.py

```
import tkinter

root = tkinter.Tk()

root.mainloop()
```

Window (with Title)

hello_world.py

```python
import tkinter

root = tkinter.Tk()
root.title("Sample GUI Application")

root.mainloop()
```

Window (with Size)

hello_world.py

```python
import tkinter

root = tkinter.Tk()
root.title("Sample GUI Application")
root.geometry("1200x400")

root.mainloop()
```

Label

Adding text to the window

Label

hello_world.py

```python
import tkinter

root = tkinter.Tk()
root.title("Sample GUI Application")
root.geometry("1200x400")

label = tkinter.Label(root, text="Hello")
label.pack()

root.mainloop()
```

Multiple Labels

hello_world.py

```python
import tkinter

root = tkinter.Tk()
root.title("Sample GUI Application")
root.geometry("1200x400")

label = tkinter.Label(root, text="Hello")
label.pack()

next_label = tkinter.Label(root, text="World")
next_label.pack()

root.mainloop()
```

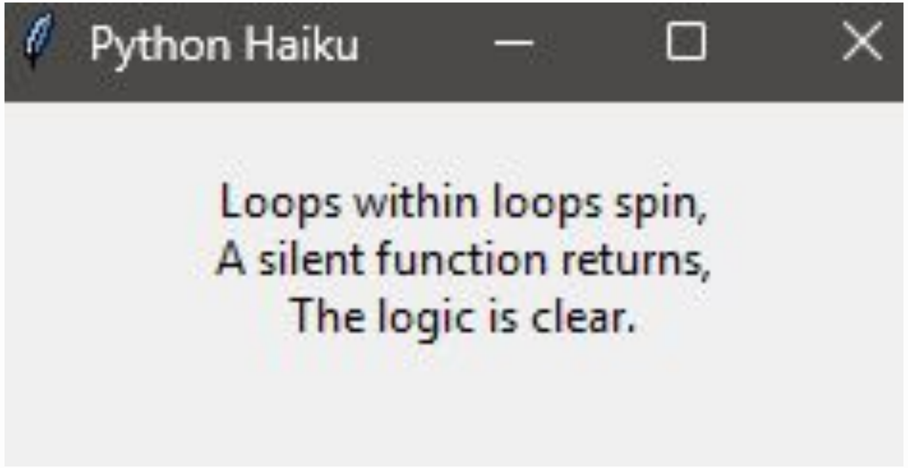
Multiline Label

multiline.py

```python
import tkinter

root = tkinter.Tk()

message = """
Hello
World
"""

label = tkinter.Label(root, text=message)
label.pack()

root.mainloop()
```

Quick Exercise: Haiku

Recreate the following window using label(s)



Loops within loops spin,
A silent function returns,
The logic is clear.

haiku.py

Properties

Adding styling and layout to components

Component Font Style

props.py

```python
import tkinter

root = tkinter.Tk()
label = tkinter.Label(
    root,
    text="Hello",
    font=("Arial", 14, "bold italic")
)
label.pack()
root.mainloop()
```

Find Other Fonts Available

font_families.py

```python
import tkinter
from tkinter import font

root = tkinter.Tk()

all_fonts = font.families()
print(all_fonts)
```

Component Color

props.py

```python
import tkinter

root = tkinter.Tk()
label = tkinter.Label(
    root,
    text="Hello",
    font=("Arial", 14, "bold italic")
    fg="red",
    bg="yellow",
)
label.pack()
root.mainloop()
```

Component Size

props.py

```python
import tkinter

root = tkinter.Tk()
label = tkinter.Label(
    root,
    text="Hello",
    font=("Arial", 14, "bold italic")
    fg="red",
    bg="yellow",
    width=100,
    height=20,
)
label.pack()
root.mainloop()
```

Component Pad

props.py

```python
import tkinter

root = tkinter.Tk()
label = tkinter.Label(
    root,
    text="Hello",
    font=("Arial", 14, "bold italic"),
    fg="red",
    bg="yellow",
    width=100,
    height=20,
    padx=10,
    pady=200,
)
label.pack()
root.mainloop()
```

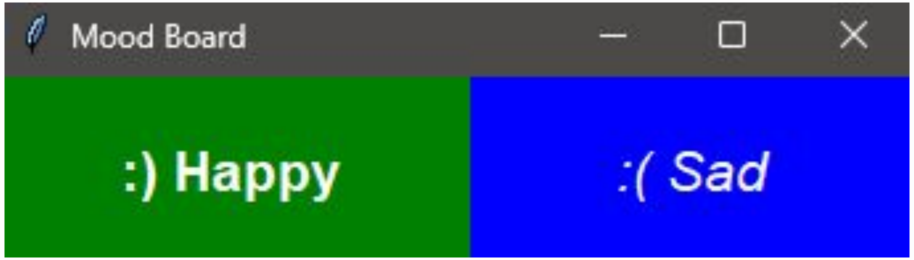
Component Pack Side

sides.py

```python
import tkinter

root = tkinter.Tk()

label1 = tkinter.Label(root, text="Left")
label1.pack(side="left")

label2 = tkinter.Label(root, text="Right")
label2.pack(side="right")

root.mainloop()
```

Quick Exercise: Mood Board

Recreate the following window using properties and label(s)



mood_board.py

Entry

Asking the user for text input

Blank Entry

entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

root.mainloop()
```

Entry Bind

entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

def show_input(event):
    print("Enter pressed")

root.bind("<Return>", show_input)
root.mainloop()
```

Component Methods

```
value = component.get()
component.set(value)
```

Entry Echo

entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

def show_input(event):
    given_text = entry.get()
    print(given_text)

root.bind("<Return>", show_input)
root.mainloop()
```

Entry Echo

entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

def show_input(event):
    given_text = entry.get()
    print(given_text)

root.bind("<Return>", show_input)
root.bind("<space>", show_input)
root.mainloop()
```

Available Bindings

Type of Key	**Behavior**
Numbers	`<0>, <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>`
Lowercase Letters	`<a>, <b>, <c>, ...`
Uppercase Letters	`<A>, <B>, <C>, ...`
Space	`<space>`
Special Keys	`<Return>, <Tab>, <Shift>, <Alt_L>, <Escape>, ...`
Function Keys	`<F1>, <F2>, <F3>, ...`
Navigation Keys	`<Left>, <Right>, <Up>, <Down>`
Multiple Keys	`<Control-Shift-s>`

Entry Marker

entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

def show_input(event):
    given_text = entry.get()
    label = tkinter.Label(root, text=given_text)
    label.pack()

root.bind("<Return>", show_input)
root.bind("<space>", show_input)
root.mainloop()
```

String Variable

Dynamic text for components

String Variable

string_var.py

```python
import tkinter

root = tkinter.Tk()

text = tkinter.StringVar(root, value="Hello")
label = tkinter.Label(root, textvariable=text)
label.pack()

root.mainloop()
```

Dynamic Label

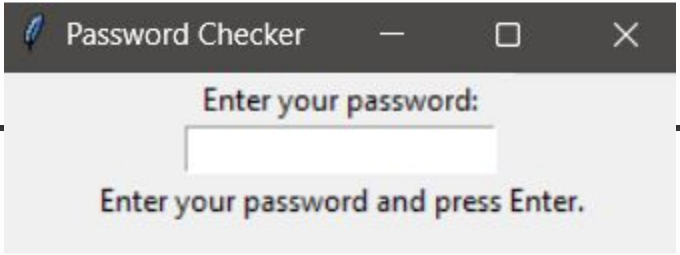
entry_bind.py

```python
import tkinter

root = tkinter.Tk()

entry = tkinter.Entry(root)
entry.pack()

user_input = tkinter.StringVar(root, value="Enter any text")
label = tkinter.Label(root, textvariable=user_input)
label.pack()

def show_input(event):
    given_text = entry.get()
    user_input.set(given_text)
...
```

Component Pattern

```
var = tkinter.Var()
comp = Comp(...)
comp.pack()
```

Quick Exercise: Password Checker

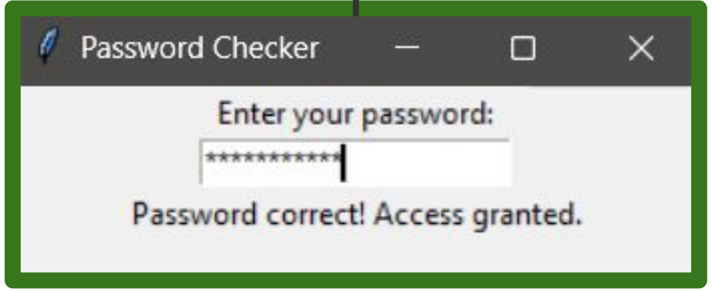


Password Checker

Enter your password:

Enter your password and press Enter.

password_checker.py

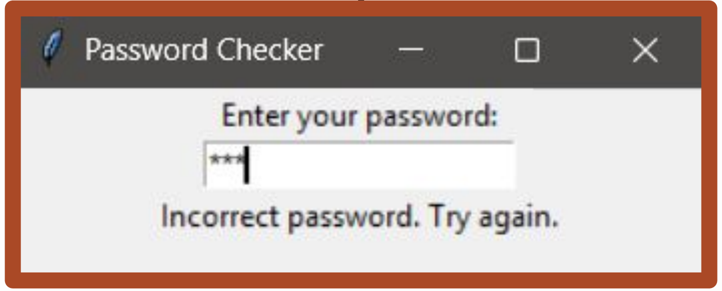


Password Checker

Enter your password:

****

Password correct! Access granted.



Password Checker

Enter your password:

***

Incorrect password. Try again.

Buttons

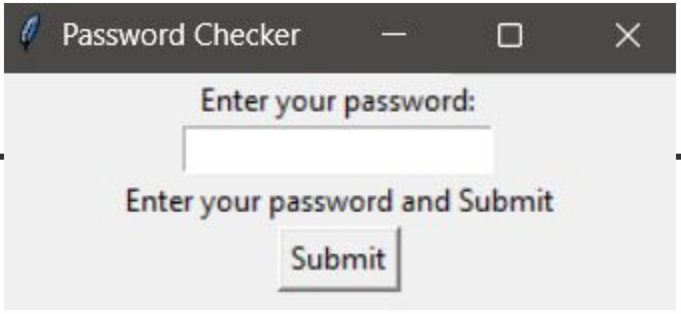
Trigger functions on command

Dynamic Label (Submit)

entry_button.py

```
...

def show_input():
    given_text = entry.get()
    user_input.set(given_text)

button = tkinter.Button(root, text="Submit", command=show_input)
button.pack()
root.mainloop()
```

Quick Exercise: Password Checker

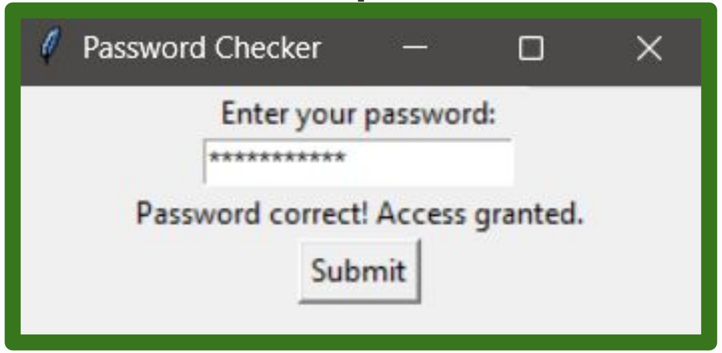

Password Checker

Enter your password:

Enter your password and Submit

Submit

password_checker.py

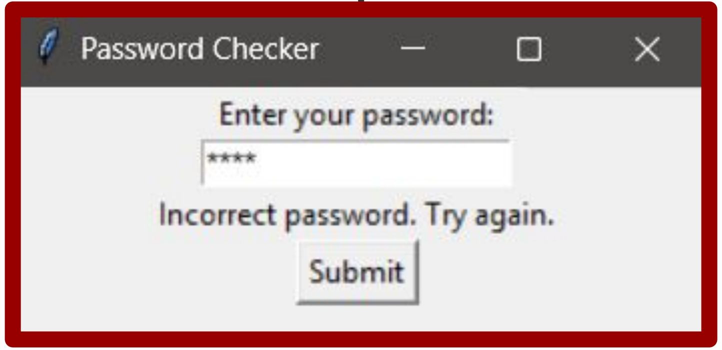

Password Checker

Enter your password:

```
***********
```

Password correct! Access granted.

Submit


Password Checker

Enter your password:

```
****
```

Incorrect password. Try again.

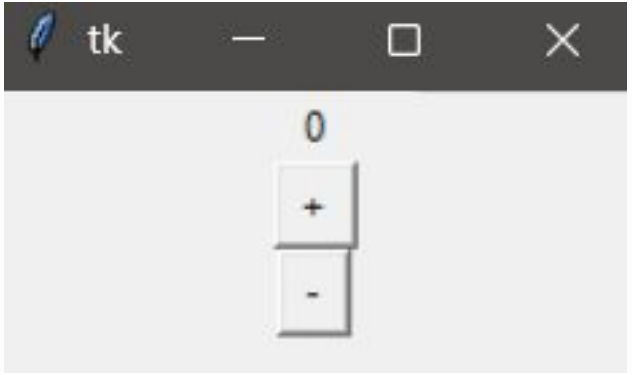
Submit

Counter

counter.py

```python
import tkinter

root = tkinter.Tk()
count = tkinter.IntVar(root, value=0)
label = tkinter.Label(root, textvariable=count)
label.pack()

def increment():
    new_value = count.get() + 1
    count.set(new_value)

button = tkinter.Button(root, text=" + ", command=increment)
button.pack()

root.mainloop()
```

Quick Exercise: Full Counter



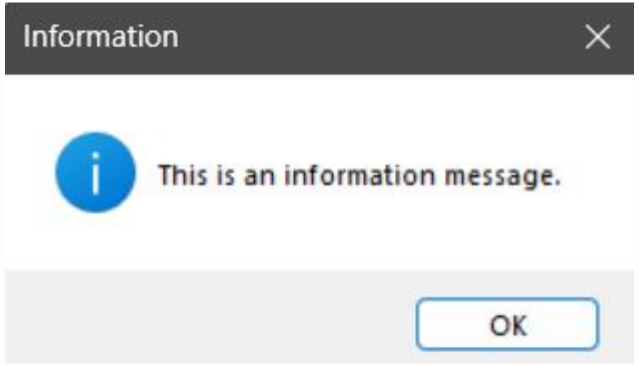
full_counter.py

Message Boxes

Sudden message displays for the user

Information Box

```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

messagebox.showinfo(
    "Information",
    "This is an information message."
)

root.mainloop()
```

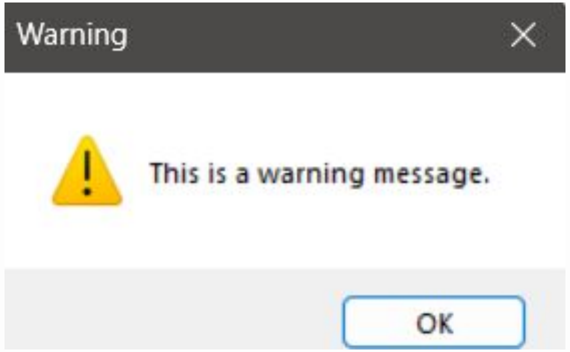

Information

This is an information message.

information_box.py

Warning Box

```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

messagebox.showwarning(
    "Warning",
    "This is a warning message."
)

root.mainloop()
```

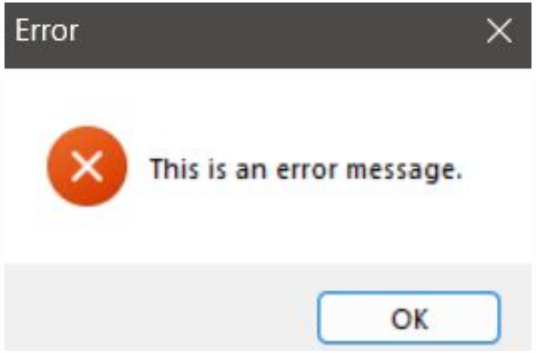

Warning

This is a warning message.

warning_box.py

Error Message Box

```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

messagebox.showerror(
    "Error",
    "This is an error message."
)

root.mainloop()
```

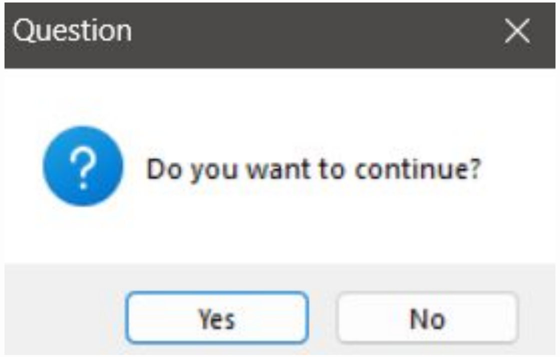

Error

This is an error message.

error_box.py

Question Message Box

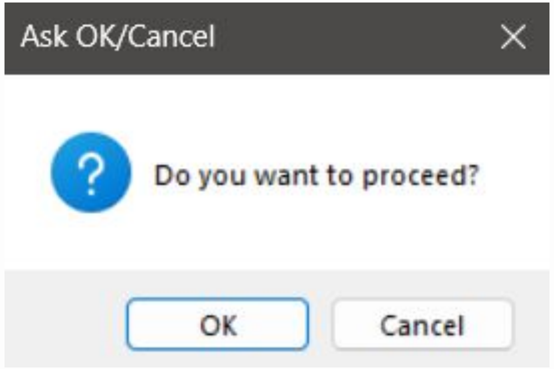
```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

# yes or no
response = messagebox.askquestion(
    "Question",
    "Do you want to continue?"
)

root.mainloop()
```



question_box.py

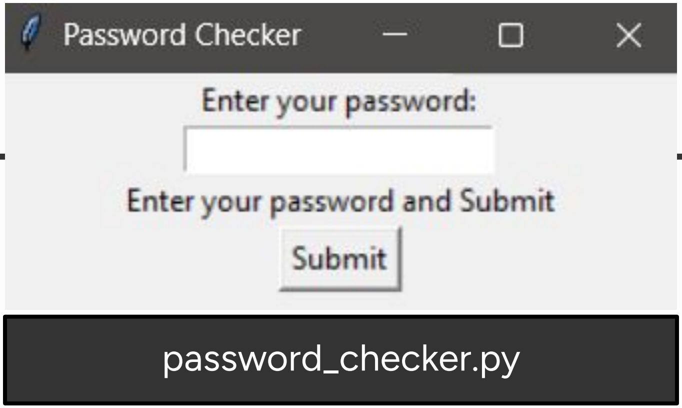
Ask OK Message Box

```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

# true or false
response = messagebox.askokcancel(
    "Ask OK/Cancel",
    "Do you want to proceed?"
)

root.mainloop()
```



ok_box.py

Quick Exercise: Password Checker

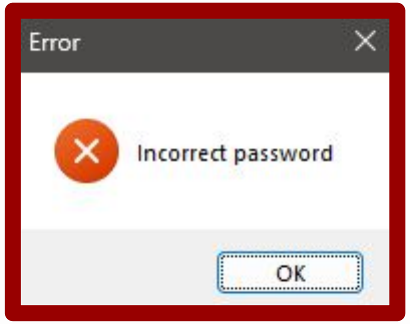

Password Checker

Enter your password:

Enter your password and Submit

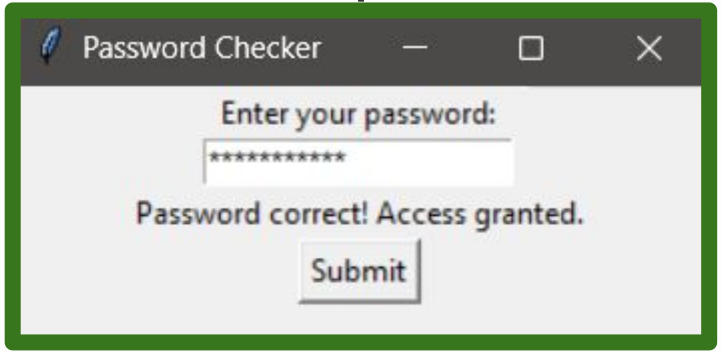
Submit

password_checker.py


Error

Incorrect password

OK

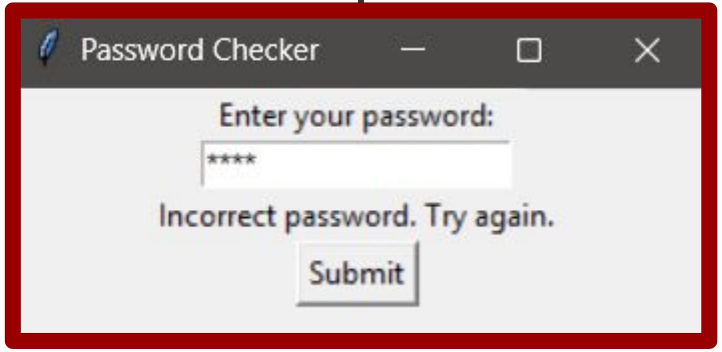

Password Checker

Enter your password:

```
***********
```

Password correct! Access granted.

Submit


Password Checker

Enter your password:

```
****
```

Incorrect password. Try again.

Submit

Input Components

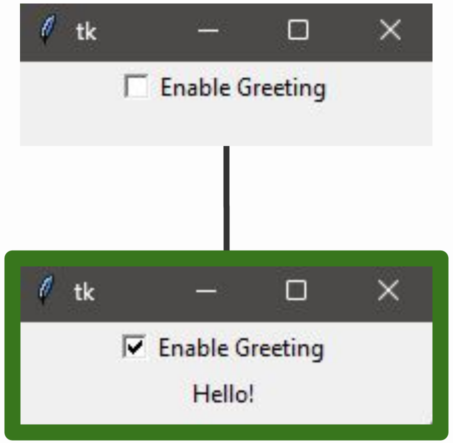
Other basic components for getting user data

Checkbox

checkbox.py

```python
import tkinter

root = tkinter.Tk()

check_value = tkinter.BooleanVar()
checkbox = tkinter.Checkbutton(
    root,
    text="Enable",
    variable=check_value
)
checkbox.pack()

root.mainloop()
```

Quick Exercise: First Greeting



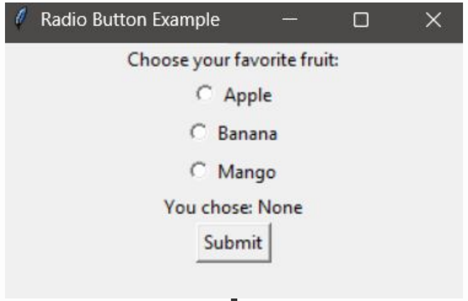
first_greeting.py

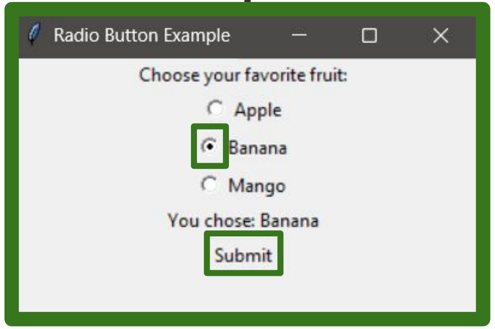
Radio Buttons

radio.py

```python
import tkinter

root = tkinter.Tk()

radio_var = tkinter.StringVar(value="Option A")
radio1 = tkinter.Radiobutton(
    root, text="Option A", variable=radio_var, value="Option A")
radio1.pack()

radio2 = tkinter.Radiobutton(
    root, text="Option B", variable=radio_var, value="Option B")
radio2.pack()

root.mainloop()
```

Quick Exercise: Store Select





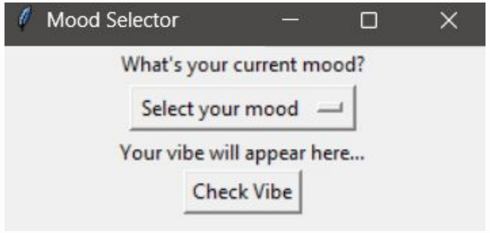
store select_.py

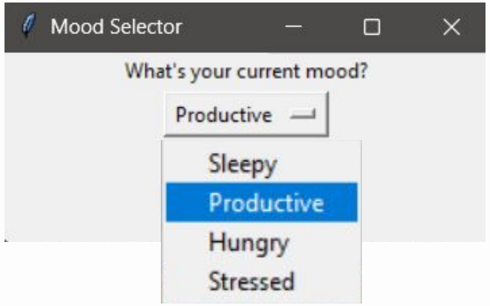
Dropdown

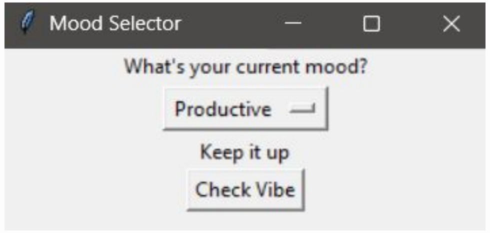
dropdown.py

```python
import tkinter

root = tkinter.Tk()

dropdown_var = tkinter.StringVar(value="Choice 1")
dropdown_menu = tkinter.OptionMenu(
    root, dropdown_var,
    "Choice 1",
    "Choice 2",
    "Choice 3"
)
dropdown_menu.pack()

root.mainloop()
```

Quick Exercise: Check Vibe







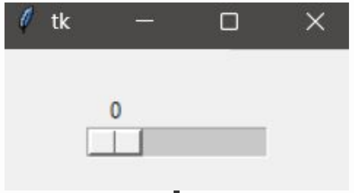
check_vibe.py

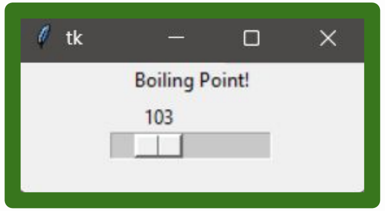
Slider

slider.py

```python
import tkinter

root = tkinter.Tk()

slider_value = tkinter.IntVar(value=0)
slider = tkinter.Scale(
    root,
    from_=0,
    to=100,
    orient="horizontal",
    variable=slider_value
)
slider.pack()

root.mainloop()
```

Quick Exercise: Thermostat





Boiling Point!

thermostat.py

Simple Dialog

simple_dialog.py

```python
import tkinter
from tkinter import simpledialog

root = tkinter.Tk()

def ask_all():
    name = simpledialog.askstring("String", "Your name?")
    age = simpledialog.askinteger("Integer", "Your age?")
    score = simpledialog.askfloat("Float", "Your score?")
    if name and age and score:
        message = f"{name} | {age} | {score}"
        tkinter.Label(root, text=message).pack()

tkinter.Button(root, text="Start", command=ask_all).pack()
root.mainloop()
```

ListBox

listbox.py

```python
import tkinter

root = tkinter.Tk()
items = tkinter.StringVar(value=["Item 1", "Item 2", "Item 3"])
listbox = tkinter.Listbox(
    root,
    listvariable=items,
    selectmode=tkinter.MULTIPLE,
)
listbox.pack()

def show_selection():
    selection = [listbox.get(index) for index in listbox.curselection()]
    print("Selected:", selection)

button = tkinter.Button(root, text="Show Selection", command=show_selection)
button.pack()
root.mainloop()
```

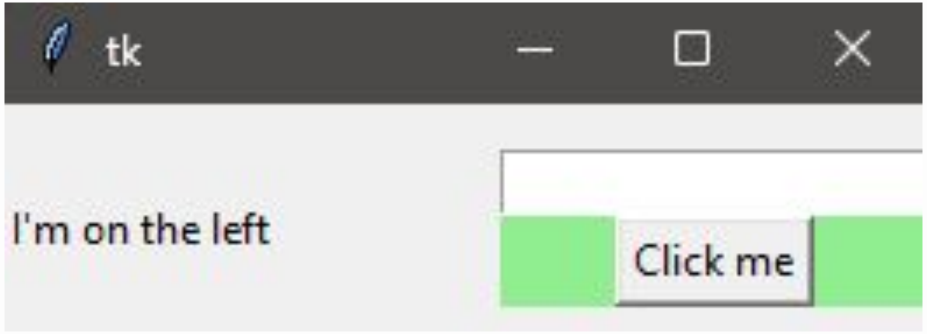
Layout

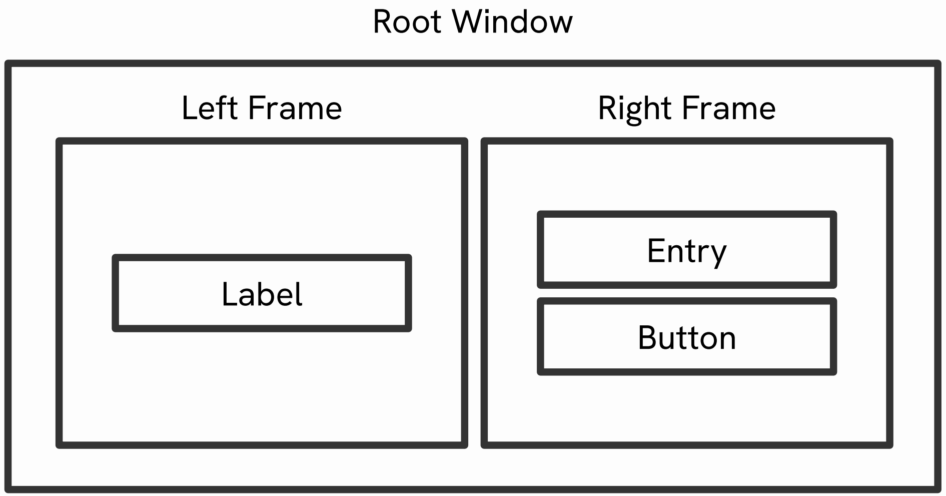
Setup the layouting for all of the components by group

Frames

frames.py

```python
import tkinter
root = tkinter.Tk()

left_frame = tkinter.Frame(root, bg="lightblue")
left_frame.pack(side="left")

left_label = tkinter.Label(left_frame, text="I'm on the left")
left_label.pack()

right_frame = tkinter.Frame(root, bg="lightgreen")
right_frame.pack(side="right")

right_entry = tkinter.Entry(right_frame)
right_entry.pack()

right_button = tkinter.Button(right_frame, text="Click me")
right_button.pack()

root.mainloop()
```


tk
I'm on the left
Click me

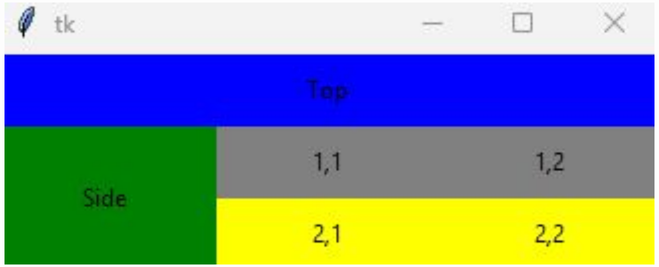

Root Window
Left Frame
Right Frame
Label
Entry
Button

Grids

grids.py

```python
import tkinter
root = tkinter.Tk()

top = tkinter.Label(root, text="Top", bg="blue", width=40, height=2)
top.grid(row=0, column=0, columnspan=3, sticky="nsew")

side = tkinter.Label(root, text="Side", bg="green", width=15, height=4)
side.grid(row=1, column=0, rowspan=2, sticky="nsew")
cell_1_1 = tkinter.Label(root, text="1,1", bg="gray", width=15, height=2)
cell_1_1.grid(row=1, column=1)
cell_1_2 = tkinter.Label(root, text="1,2", bg="gray", width=15, height=2)
cell_1_2.grid(row=1, column=2)
cell_2_1 = tkinter.Label(root, text="2,1", bg="yellow", width=15, height=2)
cell_2_1.grid(row=2, column=1)
cell_2_2 = tkinter.Label(root, text="2,2", bg="yellow", width=15, height=2)
cell_2_2.grid(row=2, column=2)

root.mainloop()
```

```
top = tkinter.Label(root, text="Top")
top.grid(row=0, column=0, columnspan=3)

side = tkinter.Label(root, text="Side")
side.grid(row=1, column=0, rowspan=2)

cell_1_1 = tkinter.Label(root, text="1,1")
cell_1_1.grid(row=1, column=1)

cell_1_2 = tkinter.Label(root, text="1,2")
cell_1_2.grid(row=1, column=2)

cell_2_1 = tkinter.Label(root, text="2,1")
cell_2_1.grid(row=2, column=1)

cell_2_2 = tkinter.Label(root, text="2,2")
cell_2_2.grid(row=2, column=2)
```



Frame and Grids

frames_grid.py

```python
import tkinter

root = tkinter.Tk()
root.title("Login Form")

form_frame = tkinter.Frame(root, padx=20, pady=20)
form_frame.pack()

tkinter.Label(form_frame, text="Username:").grid(row=0, column=0)
username_entry = tkinter.Entry(form_frame)
username_entry.grid(row=0, column=1)

tkinter.Label(form_frame, text="Password:").grid(row=1, column=0)
password_entry = tkinter.Entry(form_frame, show="*")
password_entry.grid(row=1, column=1)

login_button = tkinter.Button(form_frame, text="Login")
login_button.grid(row=2, column=0, columnspan=2, pady=10)
root.mainloop()
```

Class Organization

tkinter_class.py

```python
import tkinter

class Application(tkinter.Tk):
     def __init__(self):
          super().__init__()
          self.title("Tkinter Class Structure")
          self.geometry("300x200")
          self.create_widgets()

     def create_widgets(self):
          label = tkinter.Button(self, text="Hello", command=self.hello)
          label.pack()

     def hello(self):
          print("Hello")

app = Application()
app.mainloop()
```

06

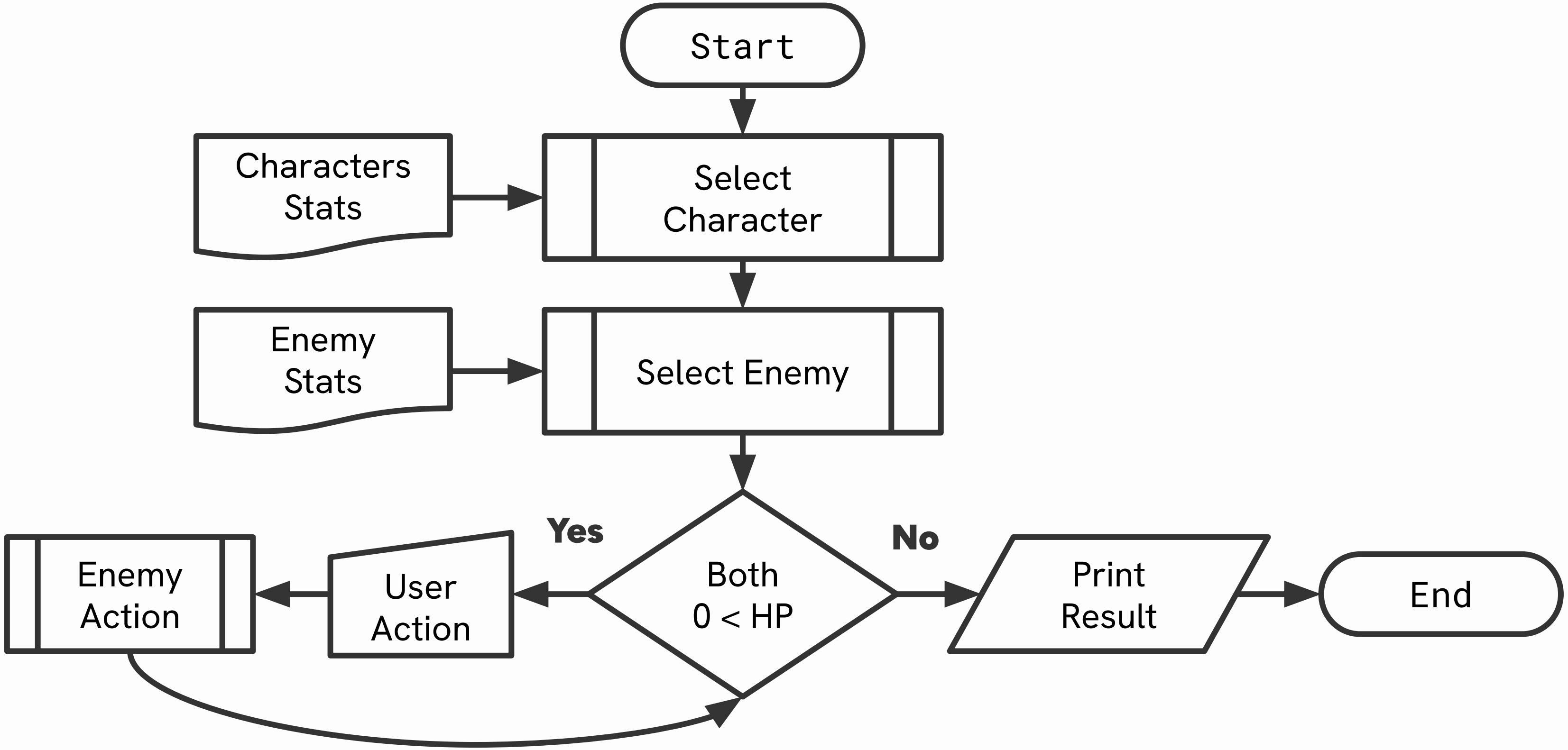
Lab Session

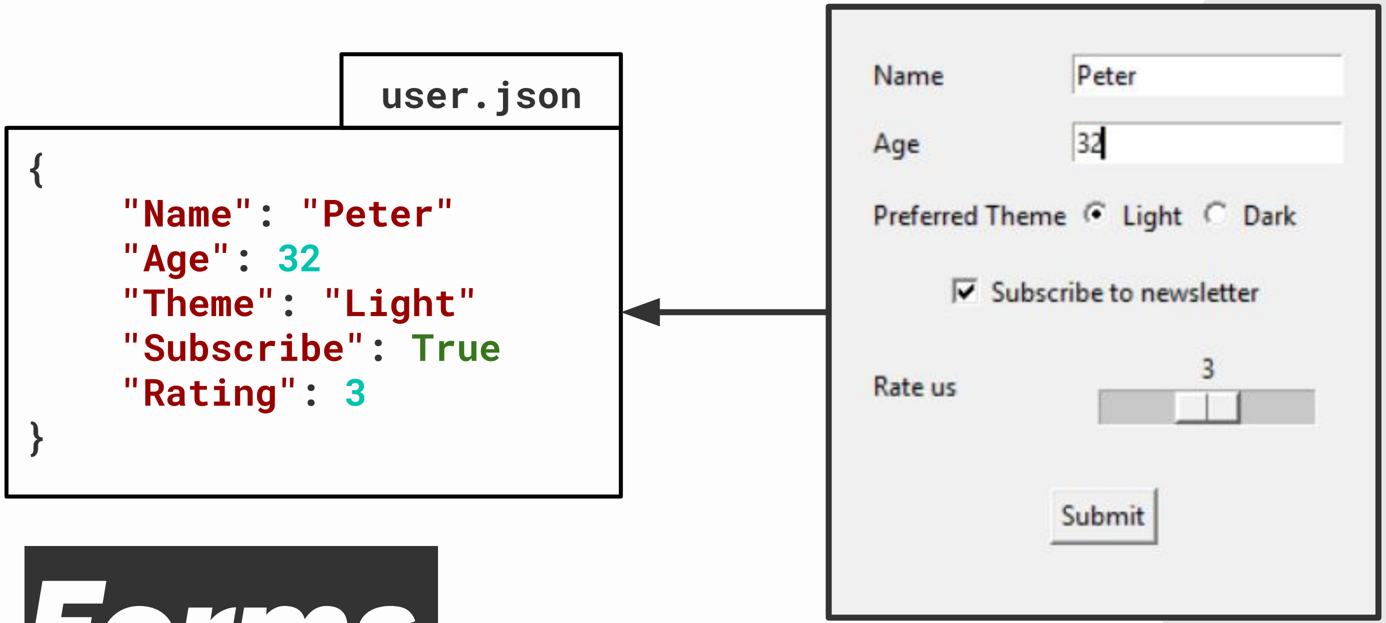
All the Major Features Covered

УЛИЦА 1905 ГОДА
АРБАТСКАЯ
БОРОВИЦКАЯ
КРОПОТКИНСКАЯ
КИЕВСКАЯ
ПАРК КУЛЬТУРЫ
КИЕВСКАЯ
ОКТЯБРЬ

Battle!

Battle! Game Flow



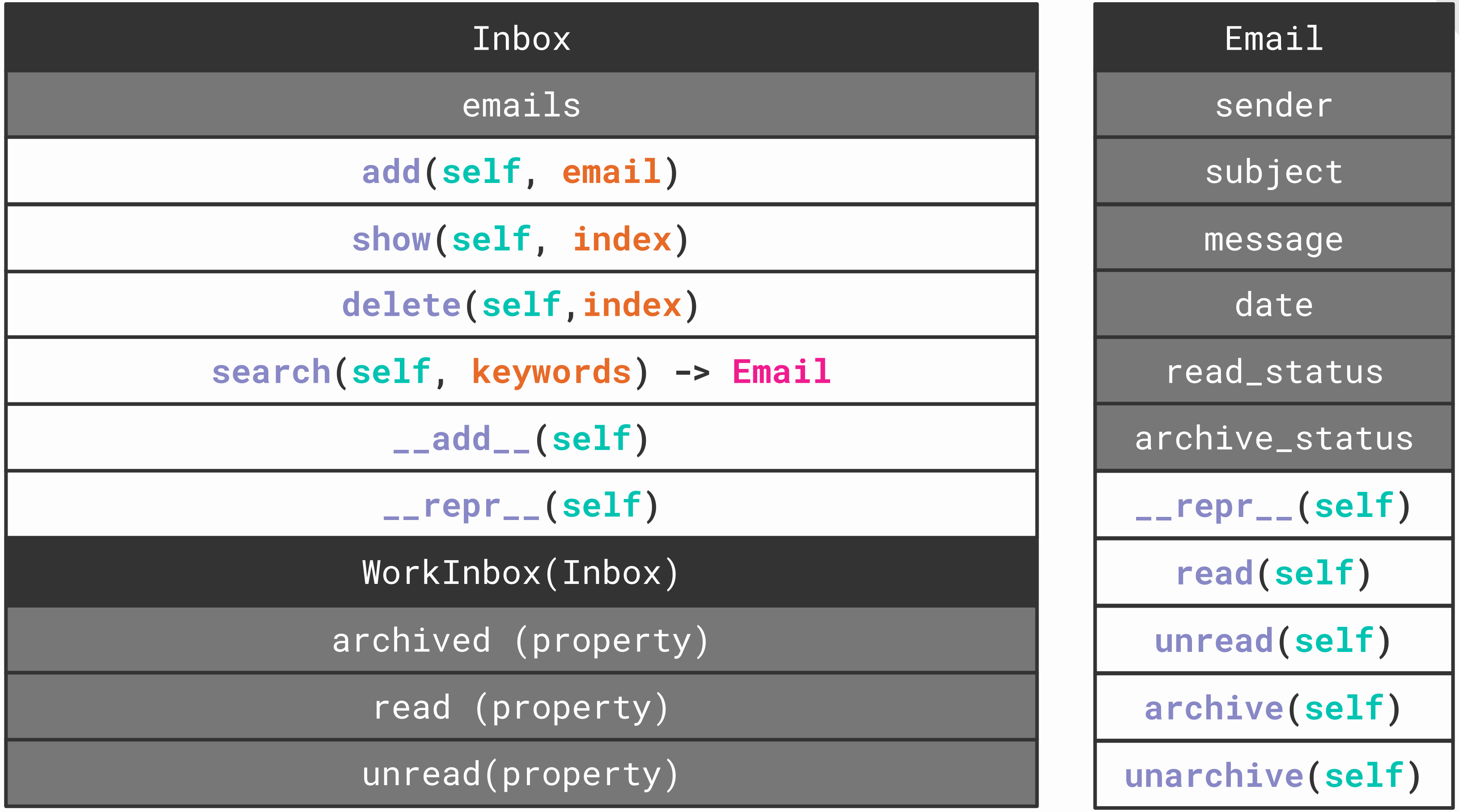

user.json

```
{
    "Name": "Peter"
    "Age": 32
    "Theme": "Light"
    "Subscribe": True
    "Rating": 3
}
```

Name	Peter
Age	32
Preferred Theme	[✓ Light] Dark
✓ Subscribe to newsletter	
Rate us	3

Forms

Inbox



Inbox
emails
add(self, email)
show(self, index)
delete(self,index)
search(self, keywords) -> Email
__add__(self)
__repr__(self)
WorkInbox(Inbox)
archived (property)
read (property)
unread(property)

Email
sender
subject
message
date
read_status
archive_status
__repr__(self)
read(self)
unread(self)
archive(self)
unarchive(self)

Sneak Peak

01

Packaging

Internal and external files

02

Multiple Tasks

Handling bottlenecks

03

Best Practices

Professional Development

04

Web Dev

Introduction to Flask

05

Lab Session

Culminating Exercise

Python: Day 03

Object-Oriented Programming