Python: Day 03

Object-Oriented Programming

Previous Agenda

01

Lists & Tuple

Ordered Group

02

Dictionary & Set

Unordered Group

03

String

Handling Text

04

File Handling

Data outside code

05

Comprehension

Iteration Shortcut

06

Lab Session

Culminating Exercise

Agenda

01

Definition

Data-Centric Approach

02

Hierarchy

Organizing Data

03

Polymorphism

Handling data types

04

Encapsulation

Data Hiding

05

GUI

Introduction to Tkinter

06

Lab Session

Culminating Exercise

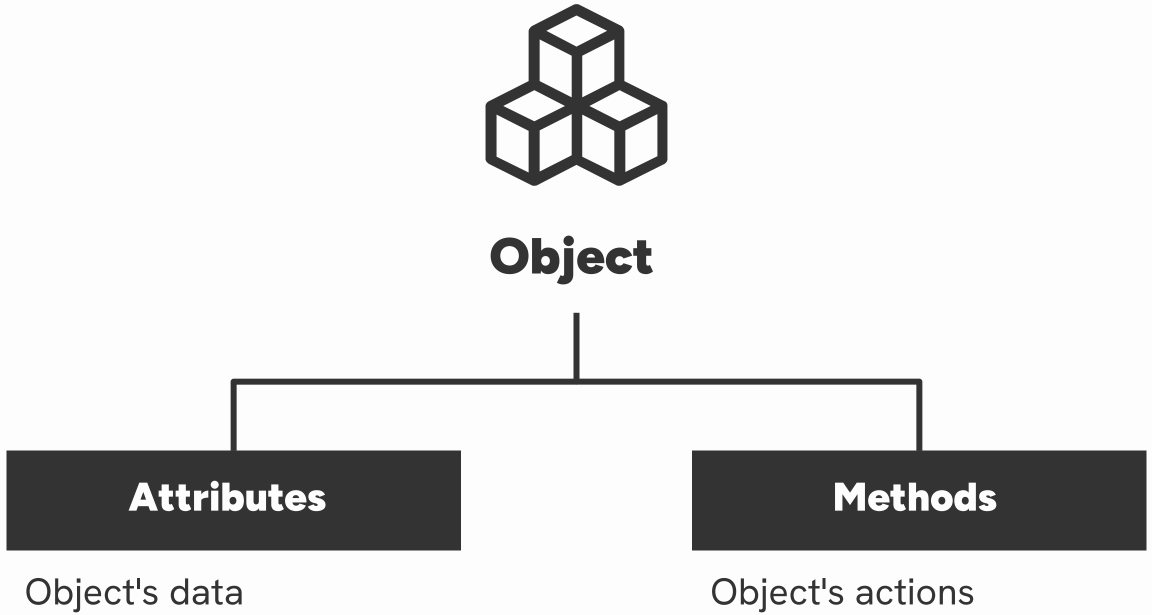
01

Definition

Programming with a focus on concepts

What makes something **something** ?

What is Object Oriented Programming?



Has → Is

Functional Identity

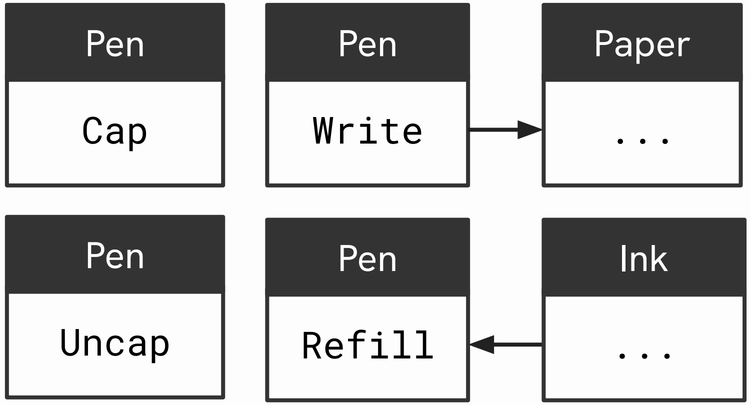
Attributes

- Attributes are unique to one object

Pen	
brand	Pilot
color	Black
capped	False

Methods

- Methods can change itself or others



Object Similarities

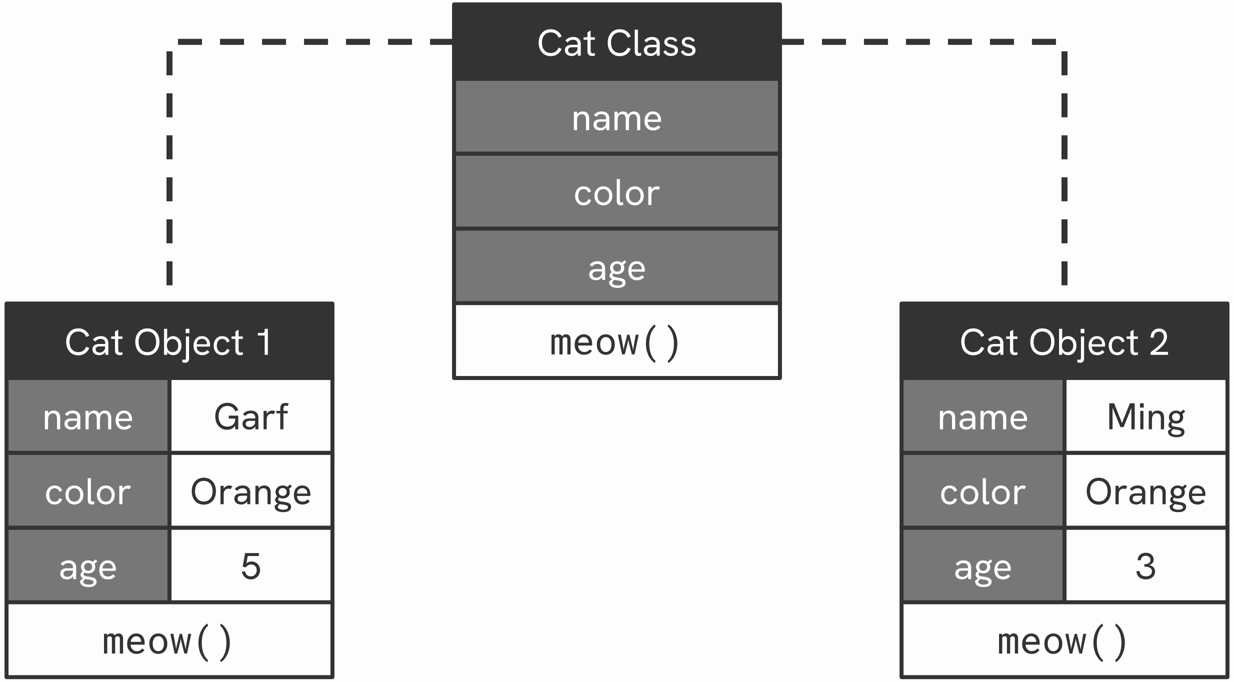
cat1	
name	Garf
color	Orange
age	5
`meow()`	

cat2	
name	Ming
color	Orange
age	3
`meow()`	

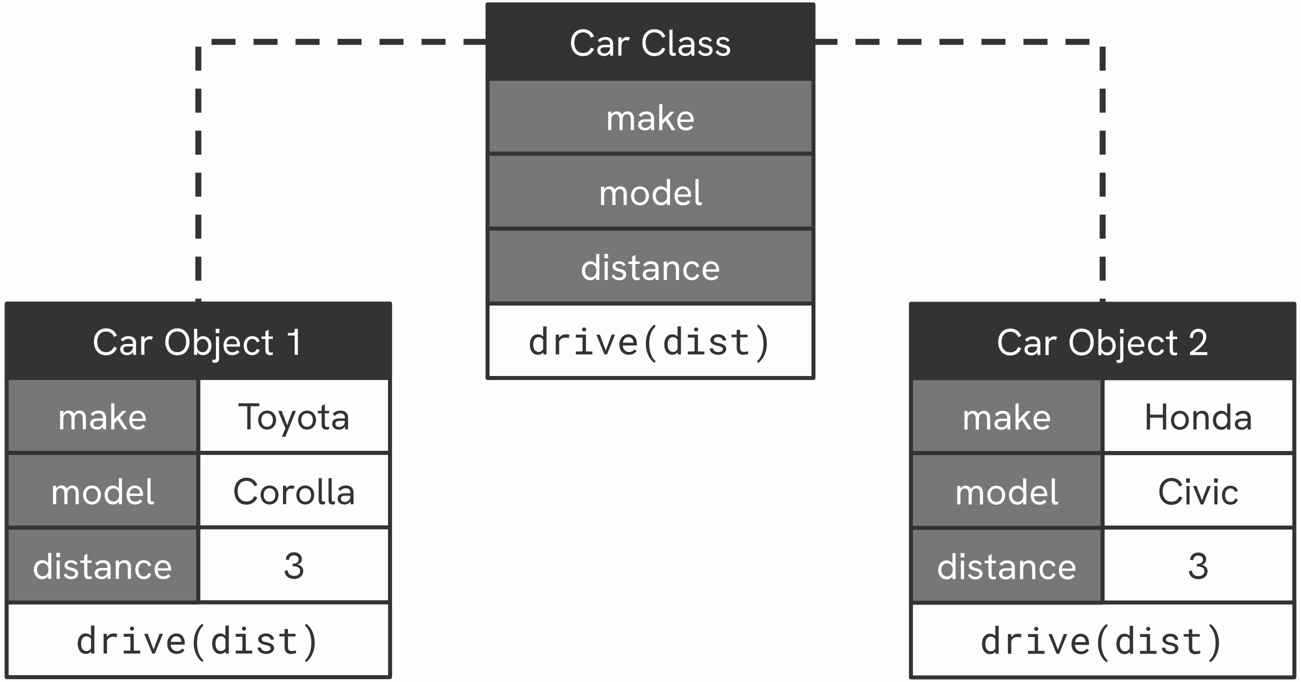
cat3	
name	Mona
color	Black
age	2
`meow()`	

What makes them different/same?

Classes to Objects



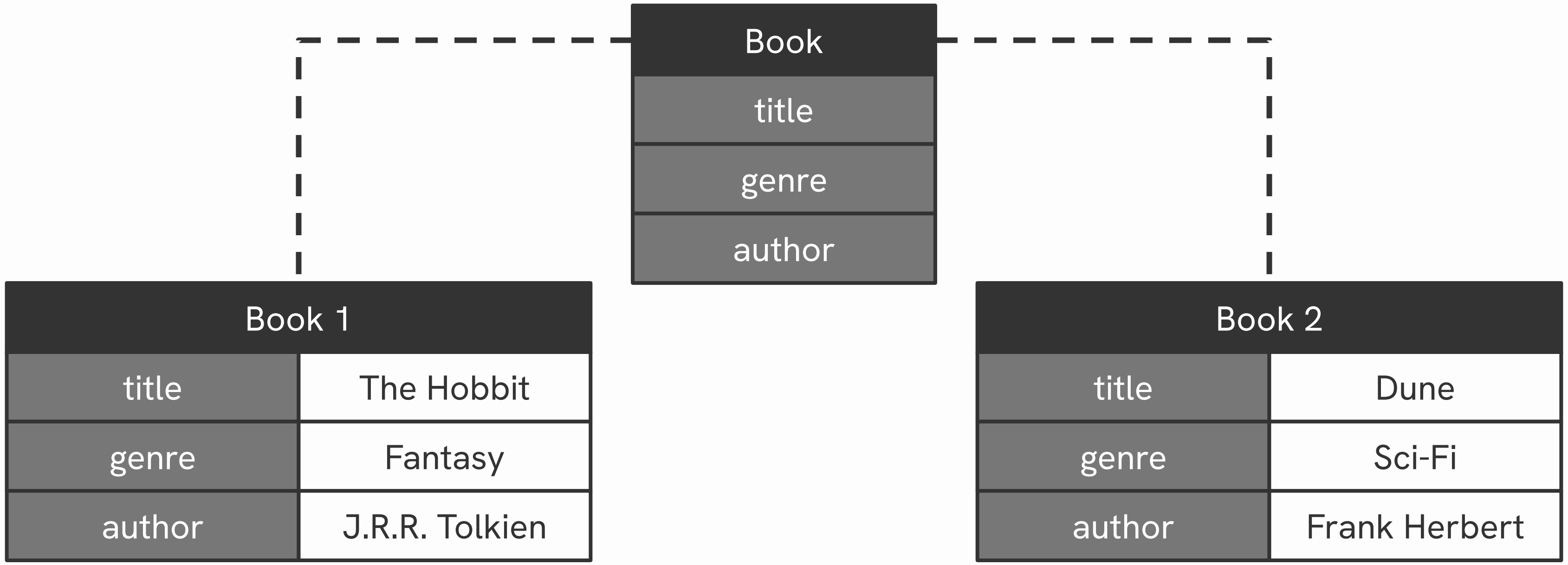
Classes to Objects



Modelling Exercise

BPI
BPI
Makati Main
OPEN
WELCOME

Book

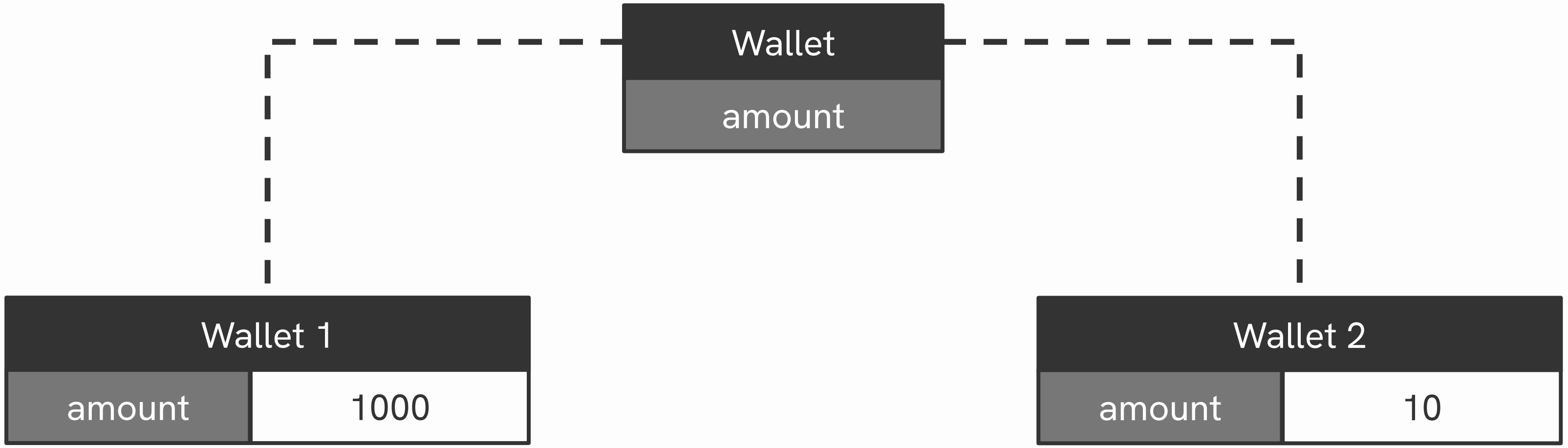


Book
title
genre
author

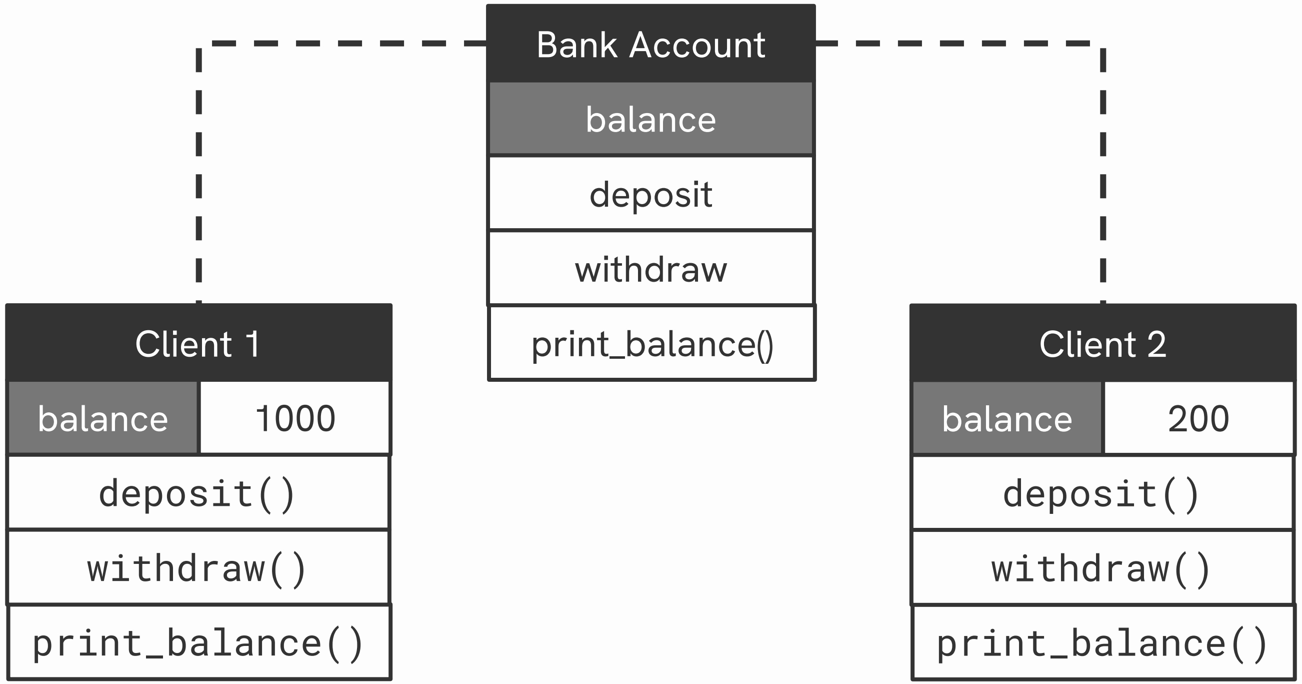
Book 1	
title	The Hobbit
genre	Fantasy
author	J.R.R. Tolkien

Book 2	
title	Dune
genre	Sci-Fi
author	Frank Herbert

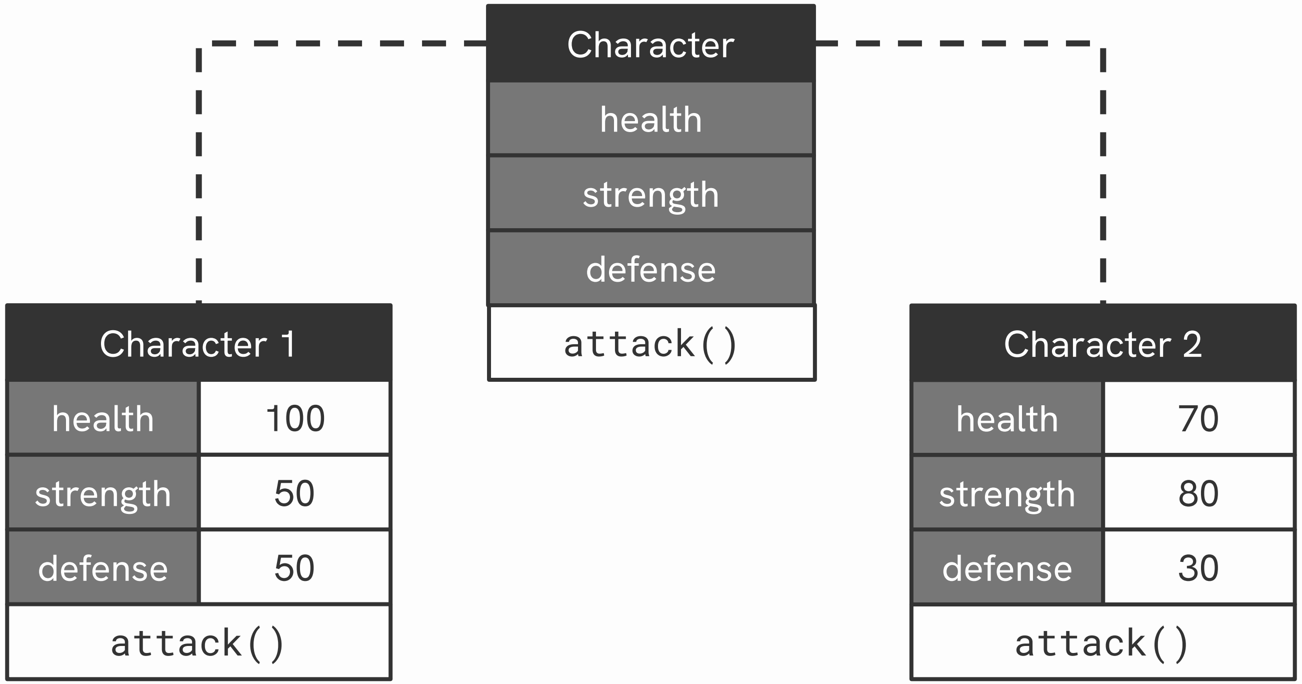
Wallet

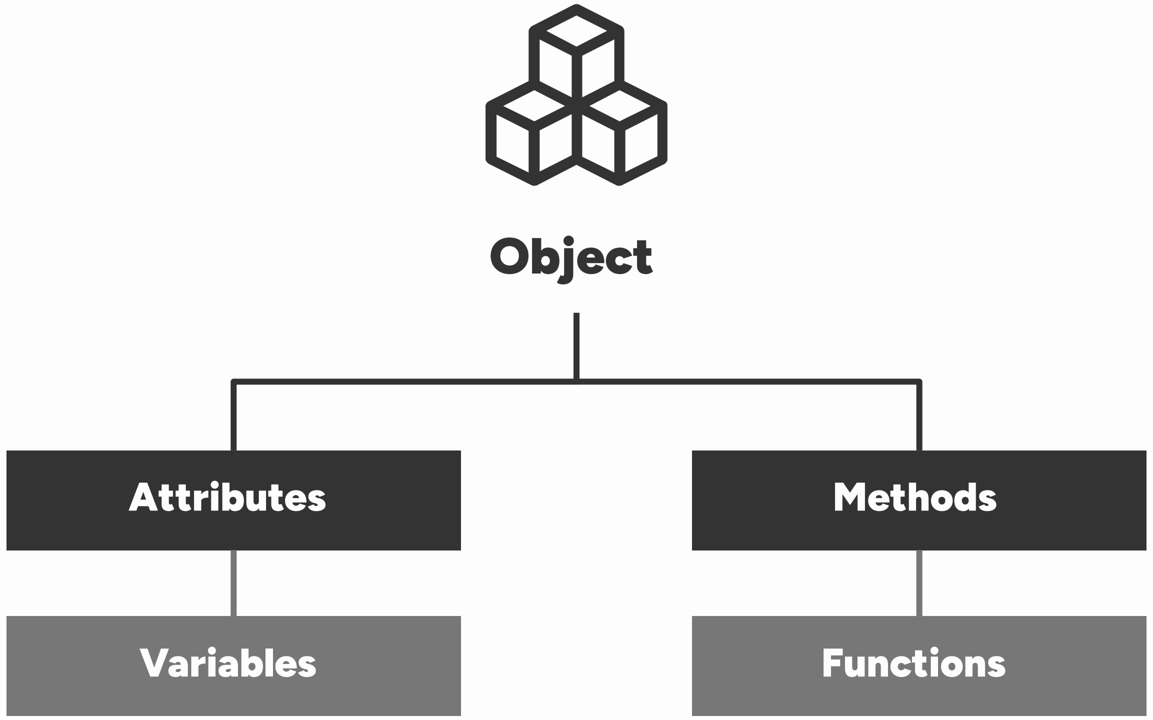


Bank Account



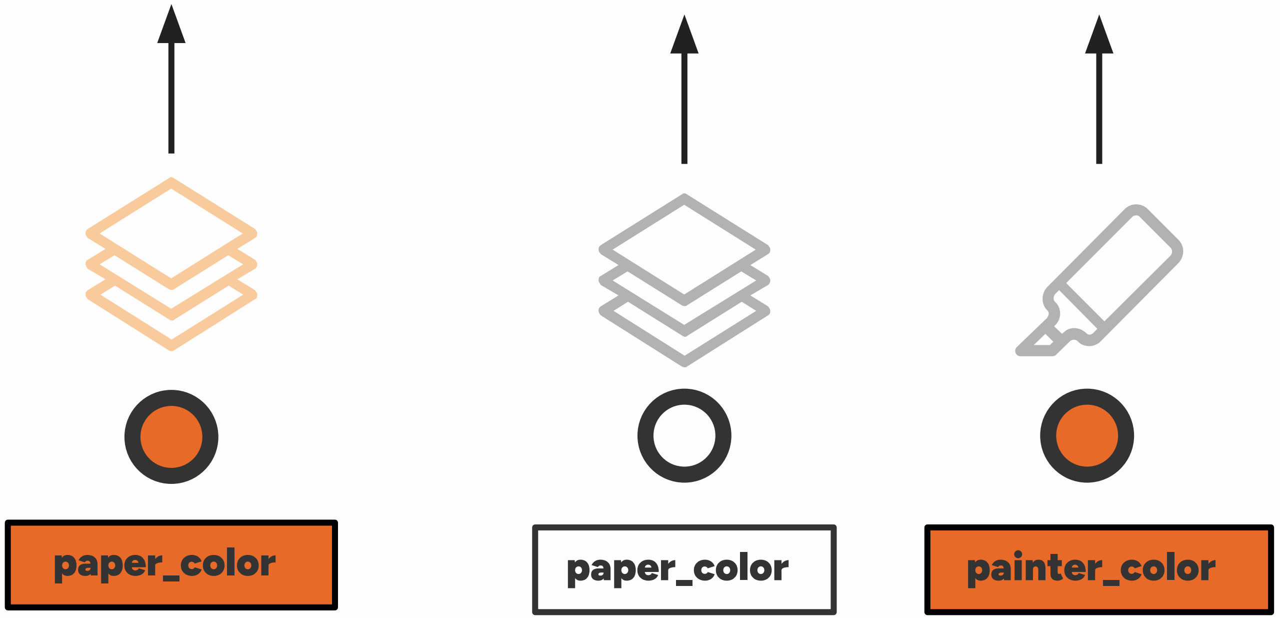
Game Character




Object
Attributes
Methods
Variables
Functions

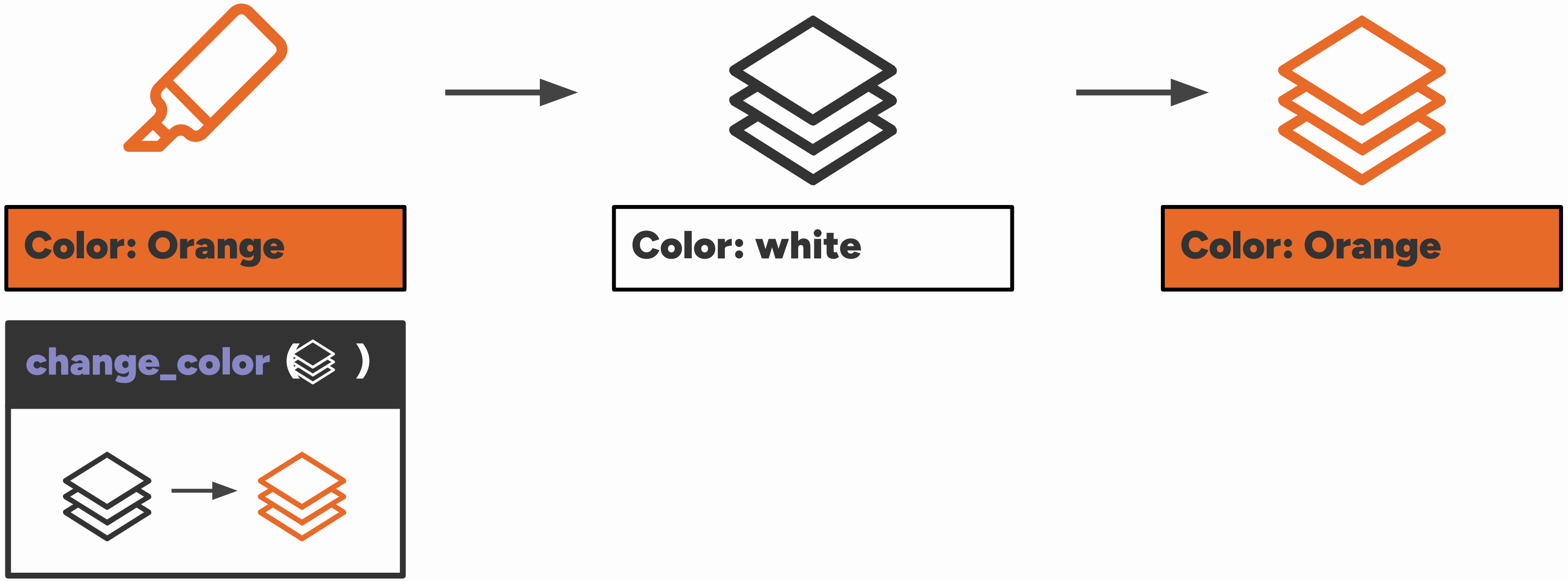
Functional Approach

```
paper_color = change_color(paper_color, marker_color)
```



Object Oriented Approach

```
marker.change_color(paper)
```



Building Exercise

Example Class

employee.py

```
class Employee:
    """Class representation for employee data"""
```

Object Creation

employee.py

```python
class Employee:
    """Class representation for employee data"""

employee1 = Employee()
```

Multiple Object Creation

employee.py

```python
class Employee:
    """Class representation for employee data"""

employee1 = Employee()
employee2 = Employee()
```

Class Constructor

employee.py

```python
class Employee:
    """Class representation for employee data"""
    def __init__(self):
        print("Employee created")

employee1 = Employee()
employee2 = Employee()
```

Class Constructor

employee.py

```python
class Employee:
    """Class representation for employee data"""
    def __init__(self, name):
        print(f"Employee {name} created")

employee1 = Employee("Richard")
employee2 = Employee("Jelly")
```

Class Constructor

employee.py

```
class Employee:
    """Class representation for employee data"""
    def __init__(self, name, id):
        print(f"Employee {name} created with ID {id}")

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

Object Attributes

employee.py

```
class Employee:
    """Class representation for employee data"""
    def __init__(self, name, id):
        self.name = name
        self.id = id
        print(f"Employee {self.name} created with ID {self.id}")

employee1 = Employee("Richard", "1234")
employee2 = Employee("Jelly", "9876")
print("Employee 1 Name:", employee1.name)
print("Employee 2 Name:", employee2.name)
```

Object Attributes

self .name

employee1 .name

Object Methods

employee.py

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

    def add_work(self, task):
        print(f"Added work {task} to {self.name}")
        return self.tasks.append(task)

employee1 = Employee("Richard", "1234")
employee2 = Employee("Jelly", "9876")
employee1.add_work("Create Slides")
employee1.add_work("Present report")
```

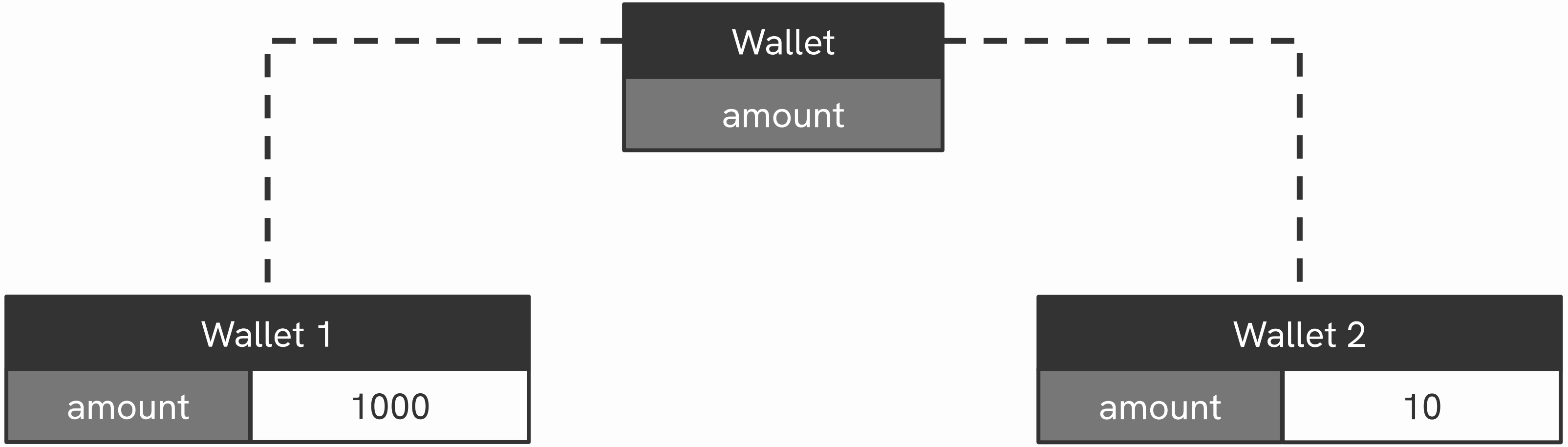
Object Methods

```
employee .add_work (task )
add_work (employee , task )
```

H1

Hands-On Building

Wallet

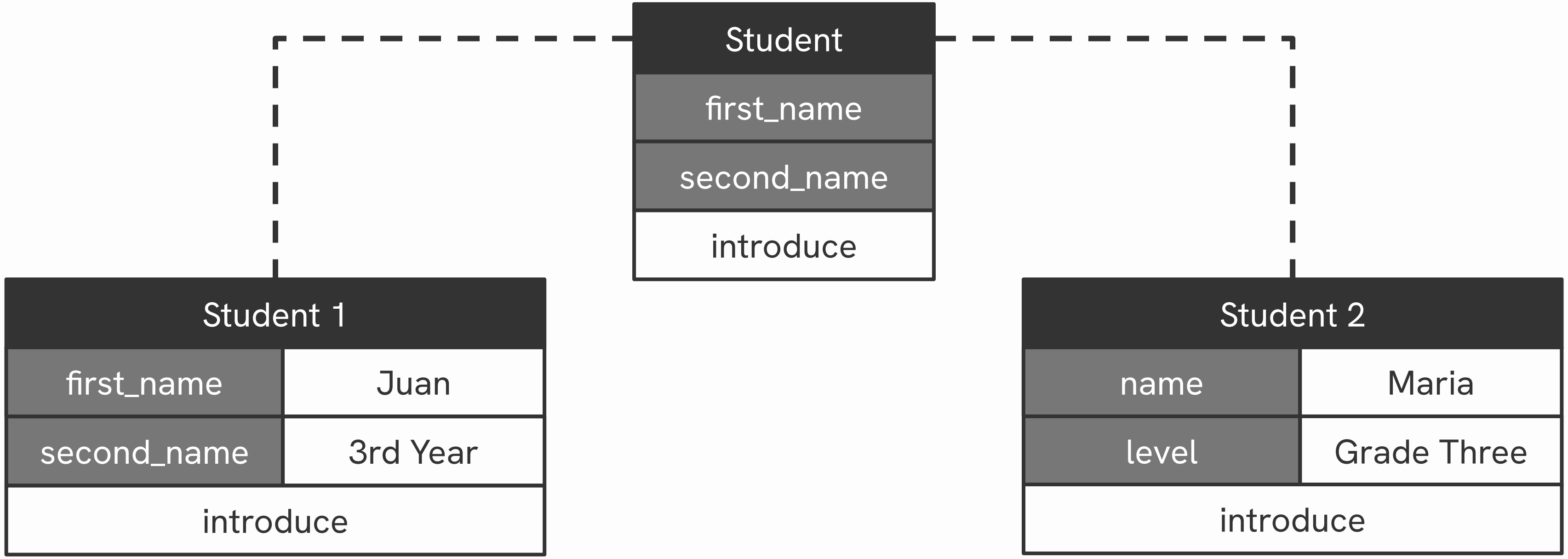


Implement: Wallet

wallet.py

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

food_wallet = Wallet(250)
food_wallet.amount += 1_000

print("Food Budget:", food_wallet.amount)
```

Person

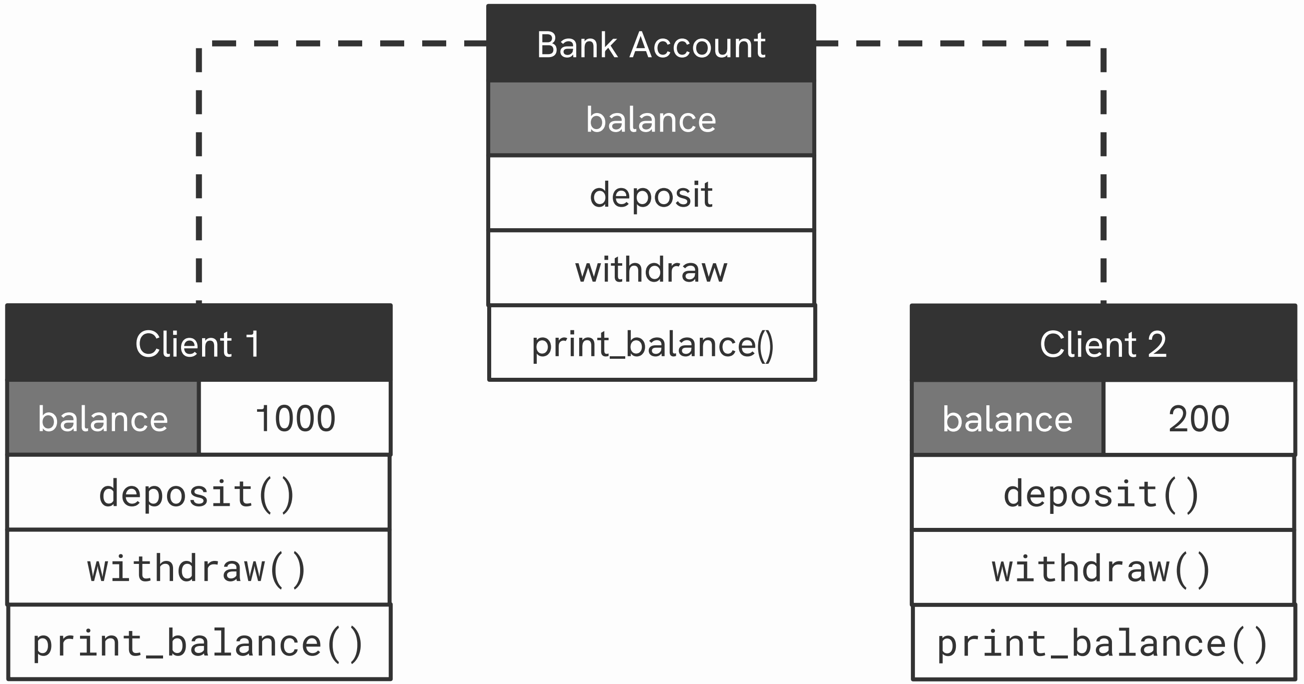

Student
first_name
second_name
introduce
Student 1
first_name
Juan
second_name
3rd Year
introduce
Student 2
name
Maria
level
Grade Three
introduce

Implement: Student

person.py

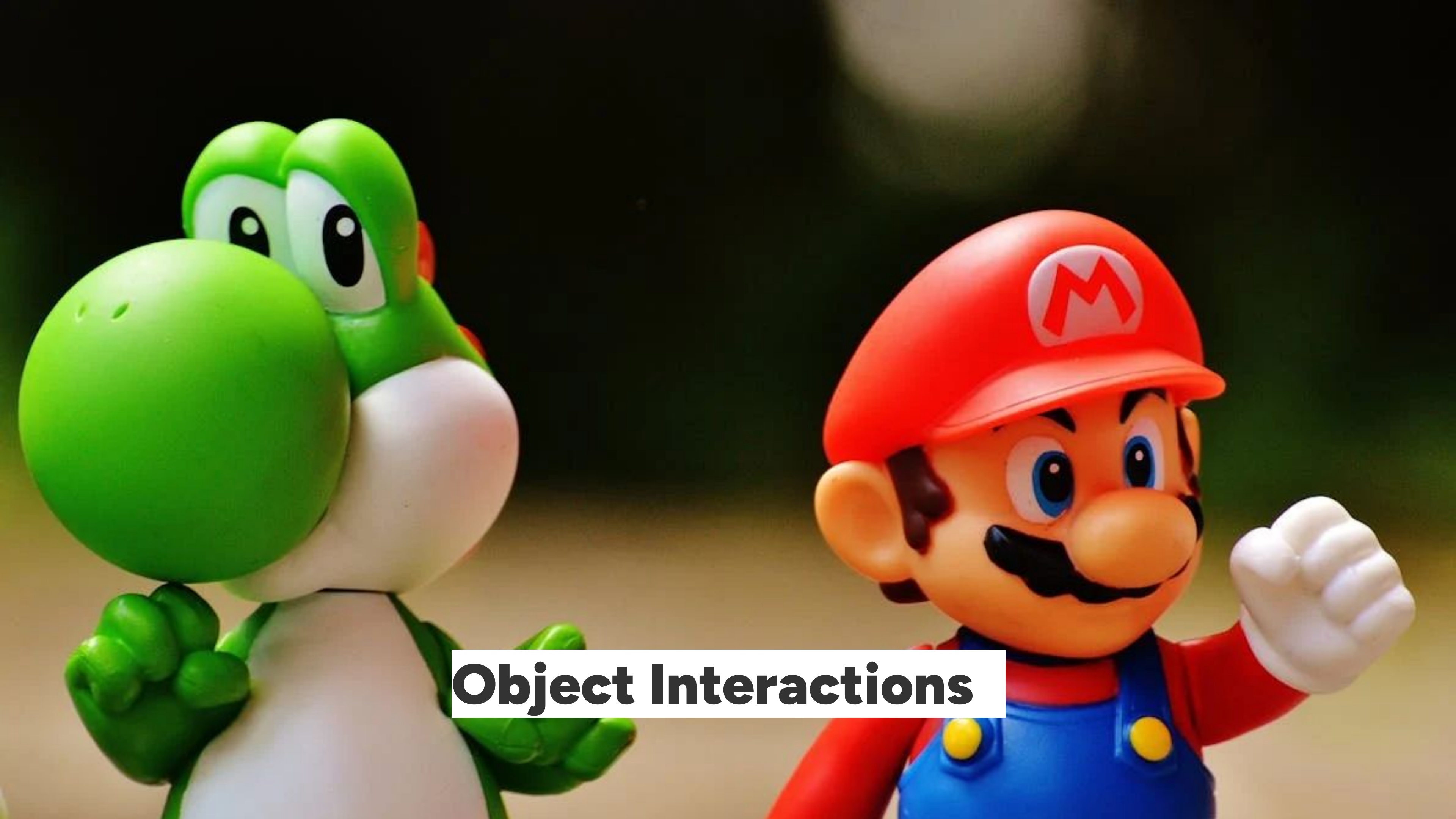
```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
        print("Created person")

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

Bank Account



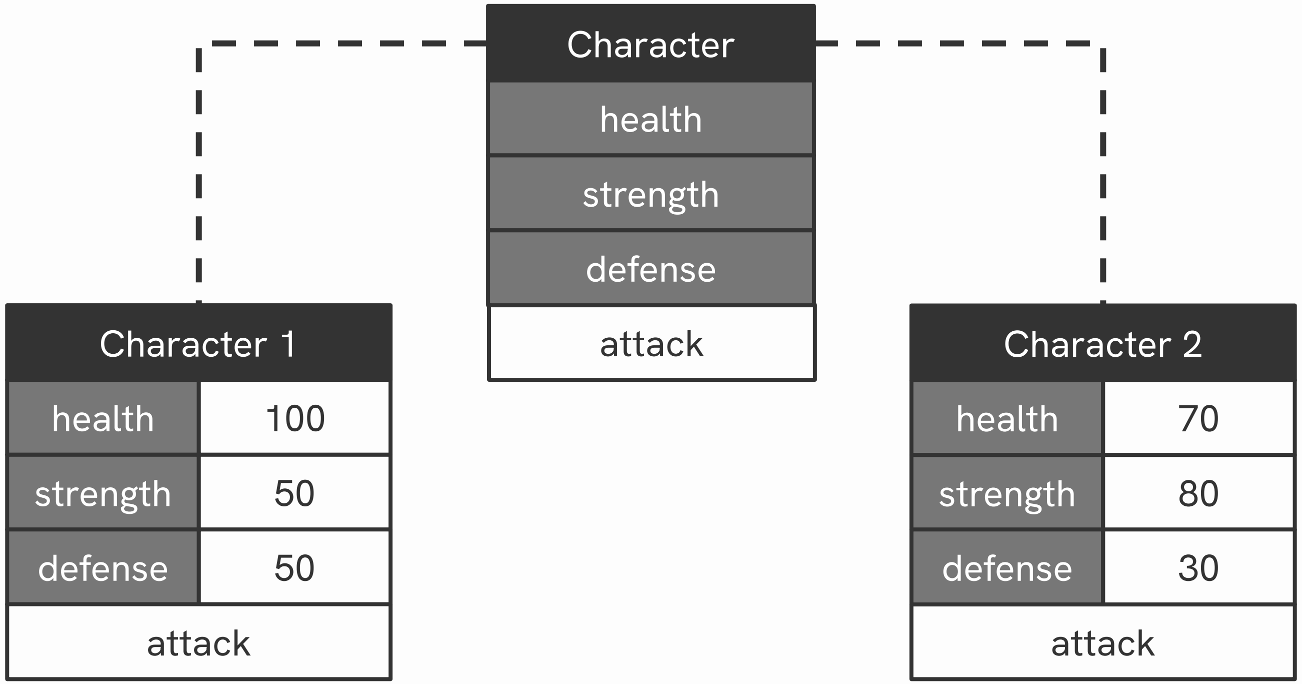
Implement: Bank Account

```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)
```



Object Interactions

Game Character



Implement: Character

character.py

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

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

player = Character(strength=100)
enemy = Character()

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

H2

Hands-Off Building

Implement: Cost Tracker

CostTracker
items
def spend(self)
def refund(self)
def show(self)
def mainloop(self)

Challenge: Cost Tracker

CostTracker
items
`def spend(self)`
`def refund(self)`
`def show(self)`
`def mainloop(self)`
`def save(self)`
`def load(self)`

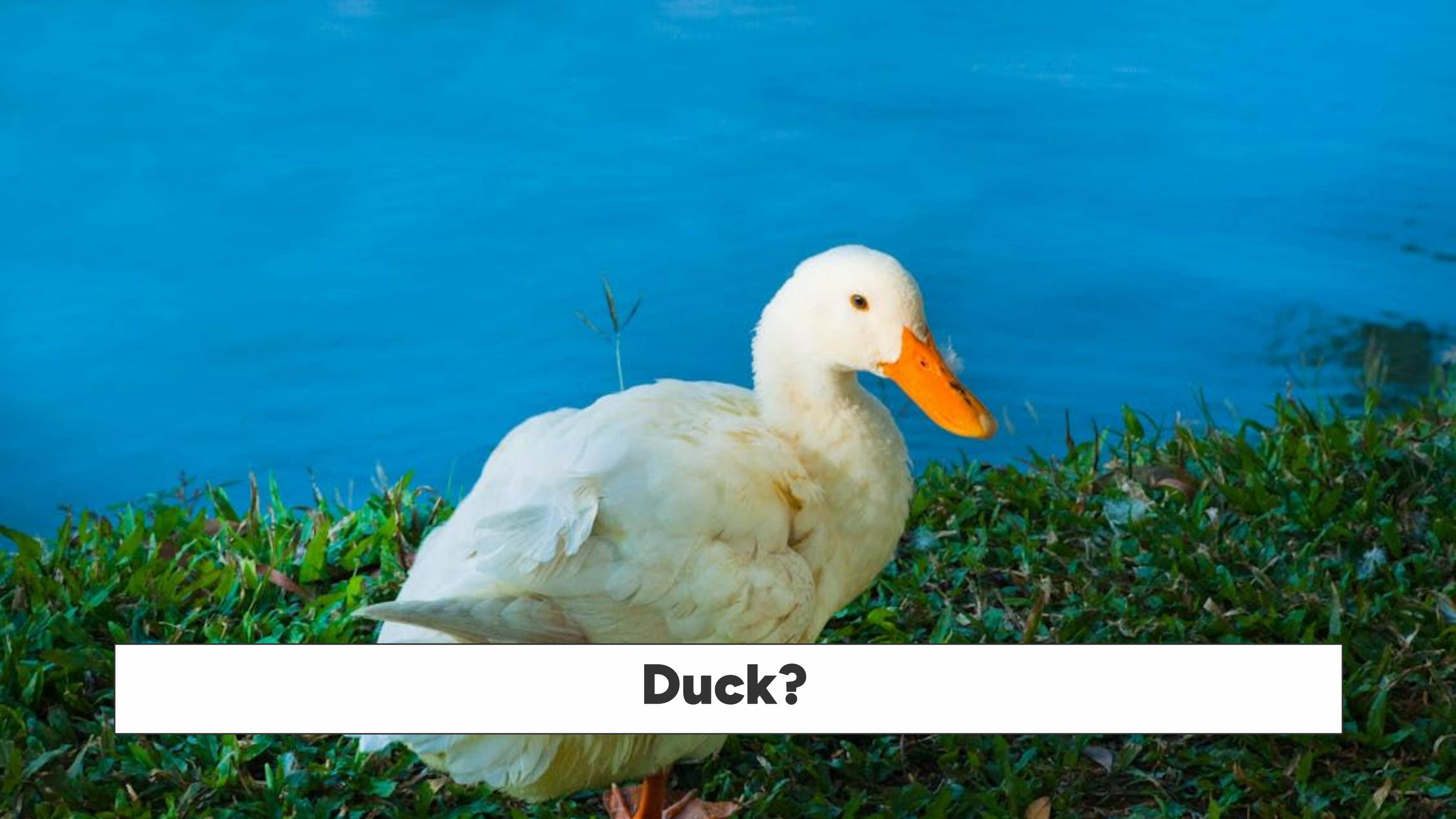
02

Hierarchy

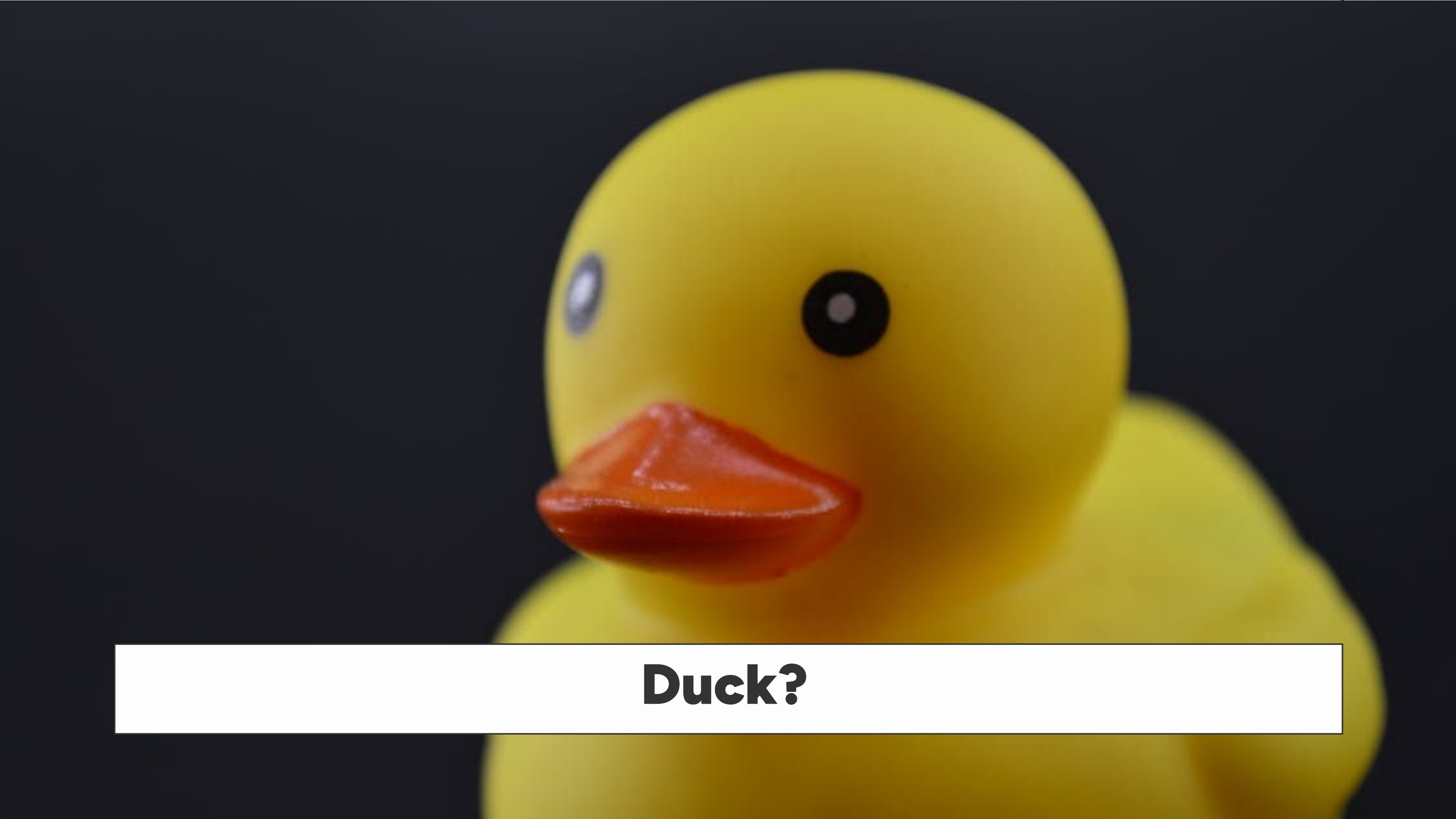
Reducing redundancy in classes

Duck Typing

Informal Polymorphism

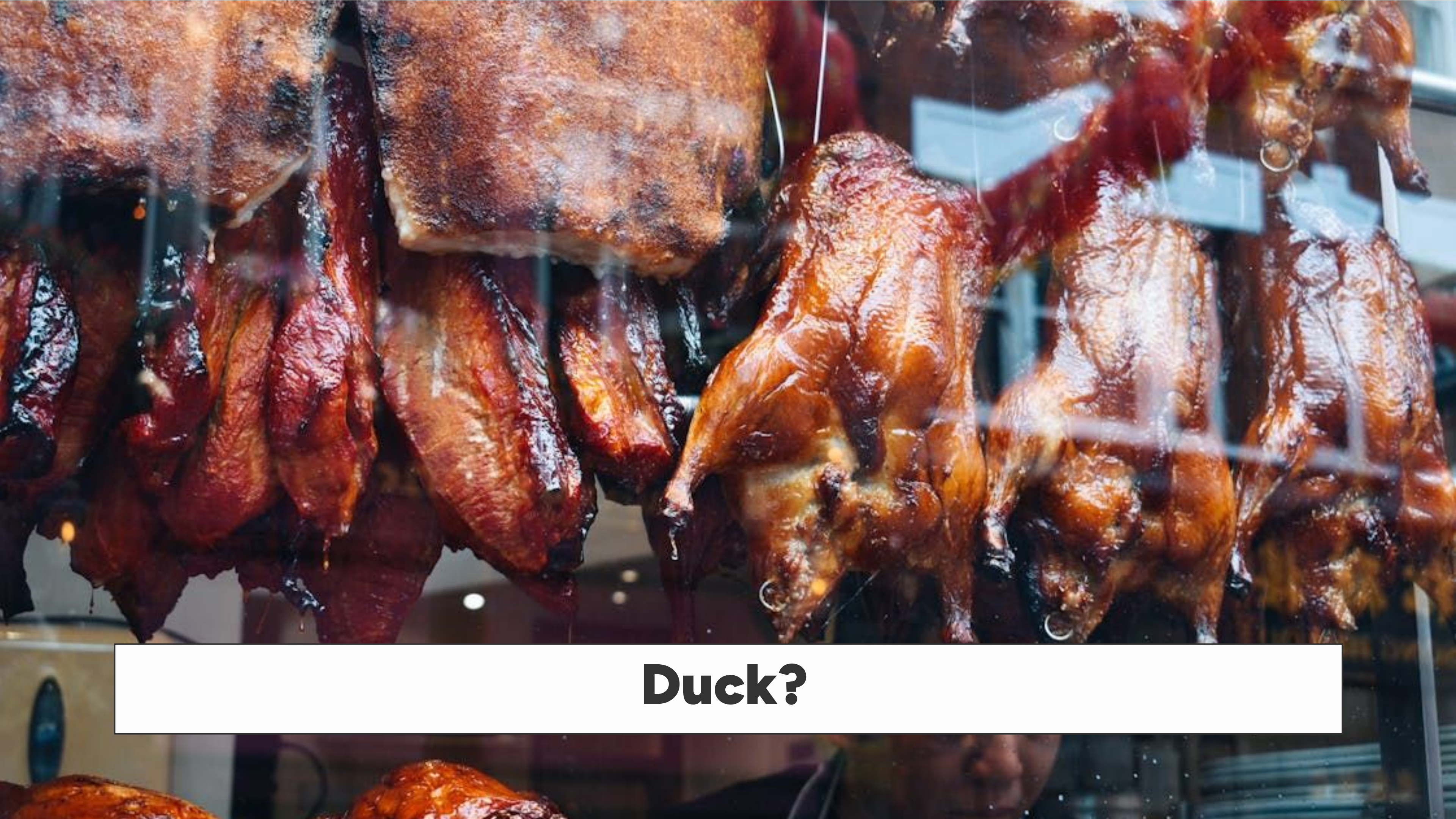


Duck?



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 → 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

```python
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)
```

H2

Validation

`validation.py`

ImageFileValidator

```
def __init__(self, path)
def valid(self) -> bool
JPG or PNG or JPEG
```

DocumentFileValidator

```
def __init__(self, path, pages)
def valid(self) -> bool
PDF and pages > 0
```

AudioFileValidator

```
def __init__(self, path, length)
def valid(self) -> bool
MP3 or WAV and length > 0
```

VideoFileValidator

```
def __init__(self, path, length, res)
def valid(self) -> bool
Is MP4 and res is 720/1080 and length > 0
```

Inheritance

Explicit class structure

Code Redundancy

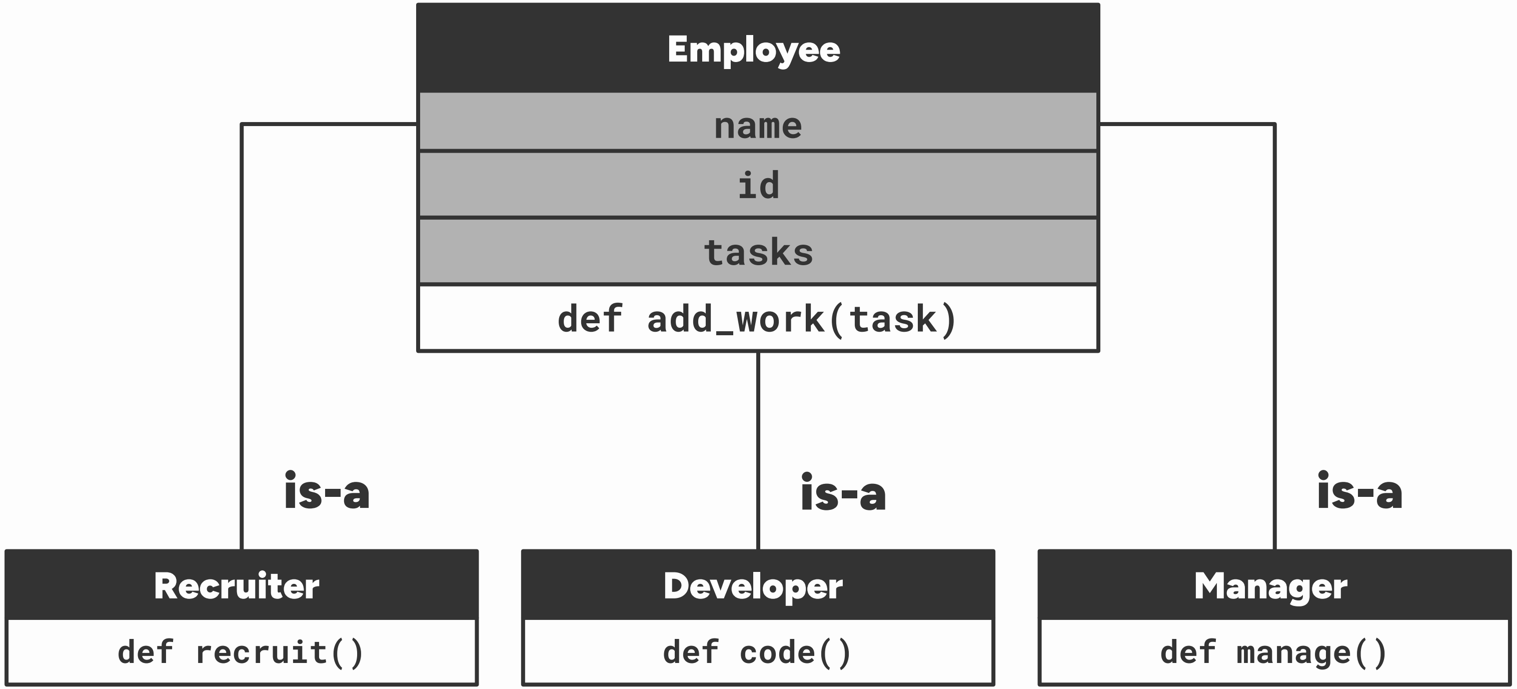
```
class Recruiter:
    def __init__(self, name, id)
    def add_work(self)
    def recruit(self)
```

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

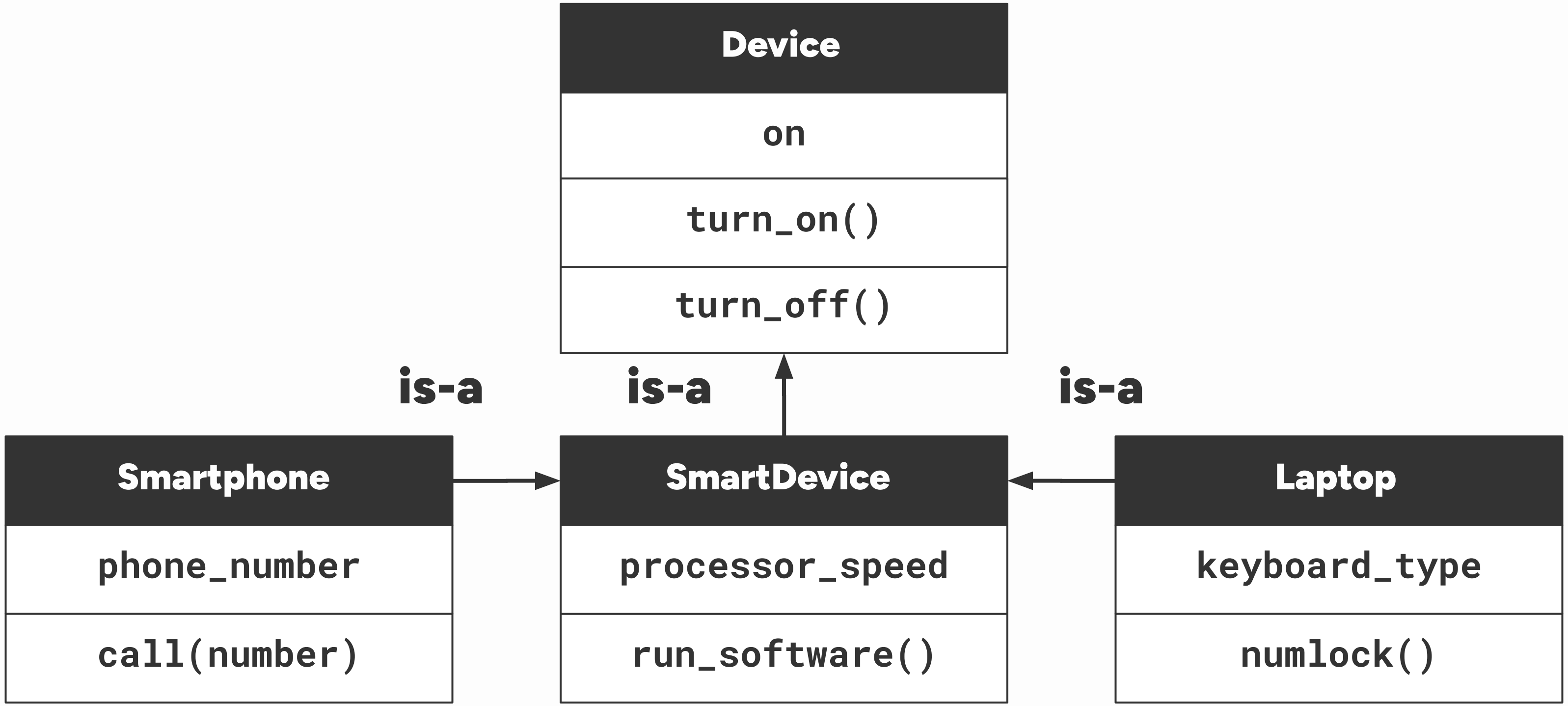
```
class Developer:
    def __init__(self, name, id)
    def add_work(self)
    def code(self)
```

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

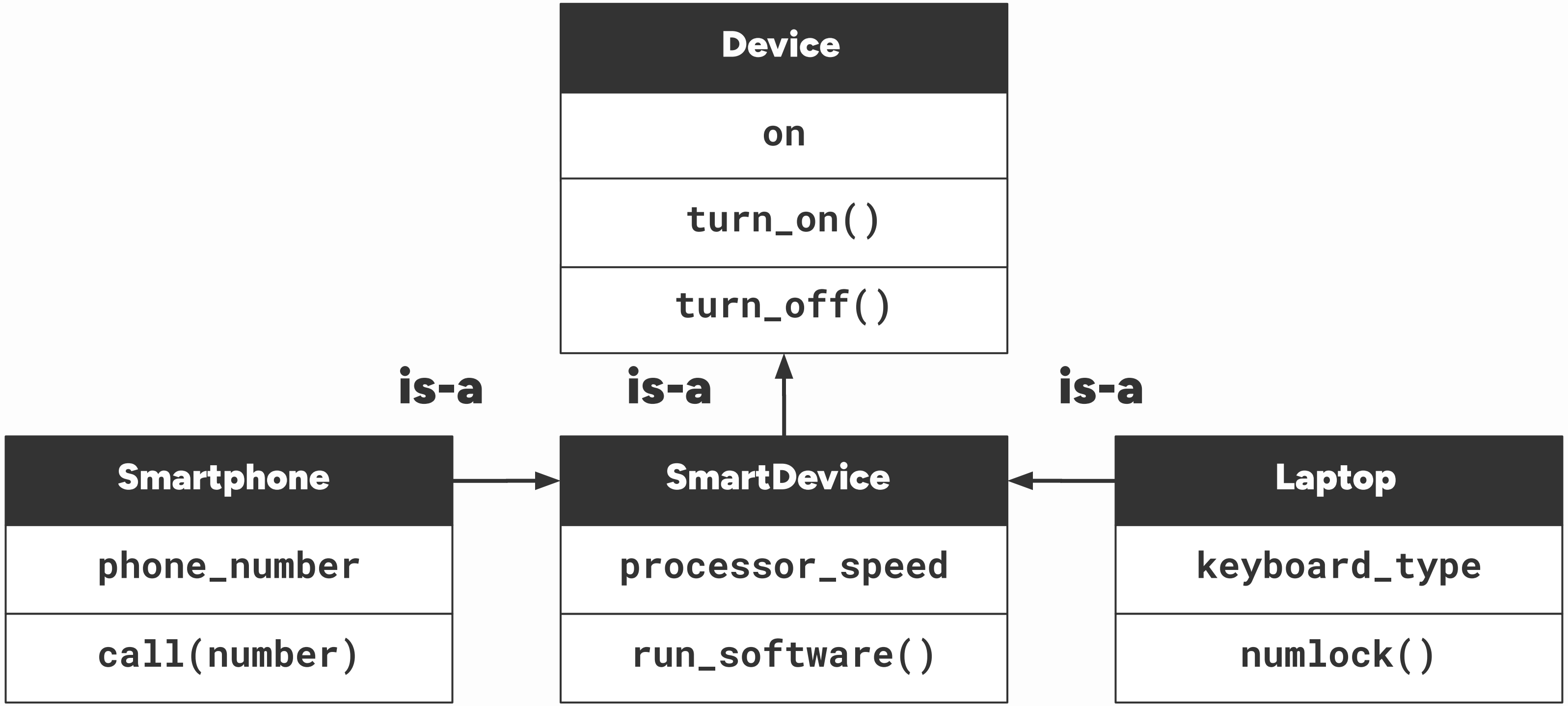
Hierarchy Example



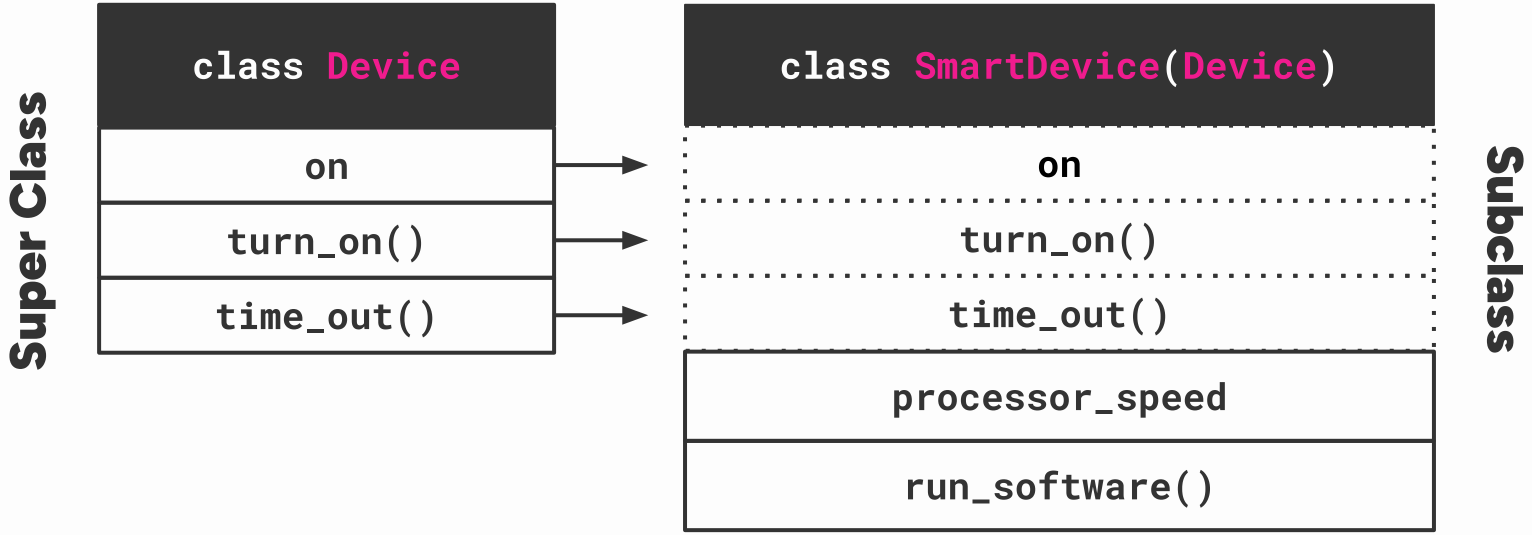
Hierarchy Example 2



Hierarchy


Device
on
turn_on()
turn_off()
is-a
is-a
is-a
Smartphone
phone_number
call(number)
SmartDevice
processor_speed
run_software()
Laptop
keyboard_type
numlock()

Class Inheritance



Student Class

student.py

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
        print("Created person")

    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
        print("Created person")

    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

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
        print("Created person")

    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
        print("Created person")

    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
        print("Created person")

    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

```python
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
        print("Created person")

    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."
```

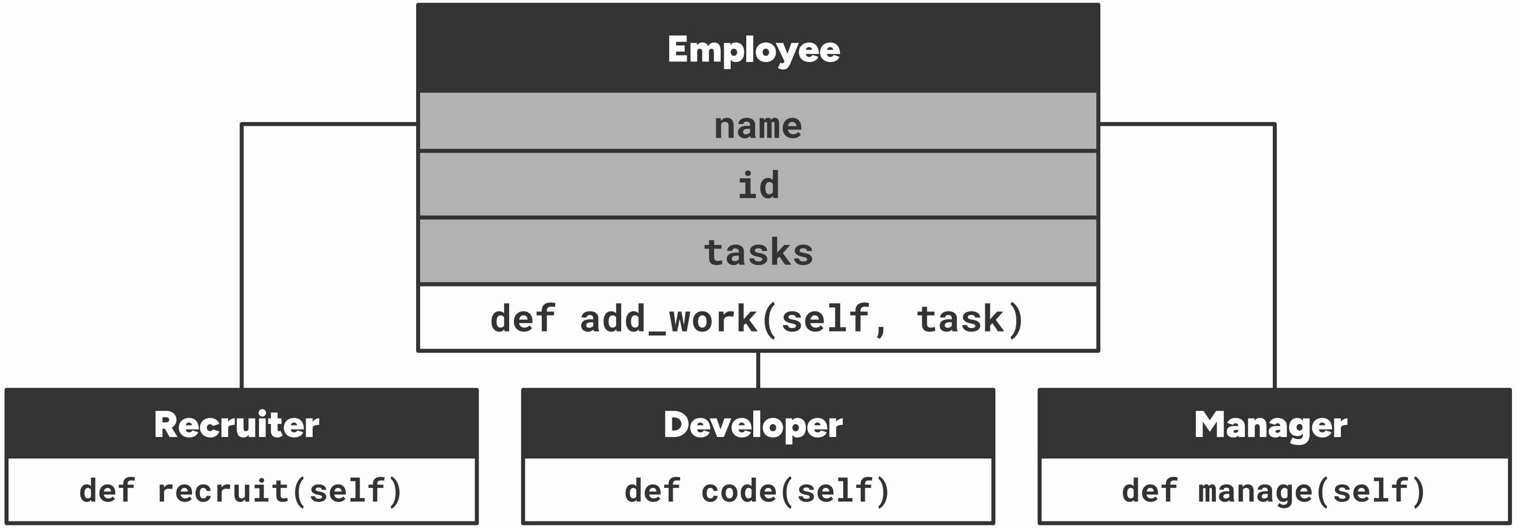
Example: Writer

writer.py

```python
class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email

    def display_info(self):
        return f"User: {self.username} Email: {self.email}"

class Writer(User):
    def __init__(self, username, email, articles):
        super().__init__(username, email)
        self.articles = articles

    def write_article(self, title):
        print(f"{self.username} is writing '{title}'...")
        self.articles += 1
```

H3

Employee Chart



03

Structure

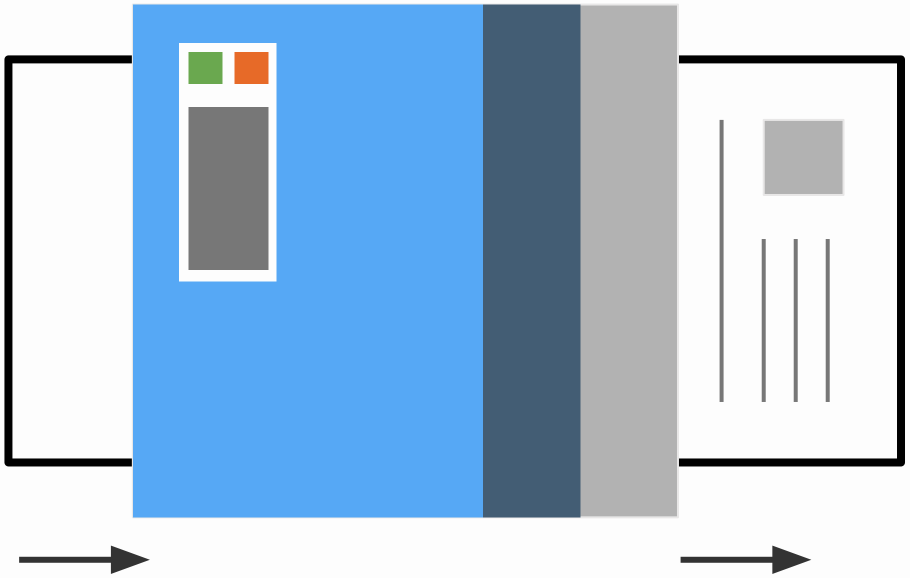
Appropriate Data Representation

Encapsulation

Manage which parts are accessible to the public

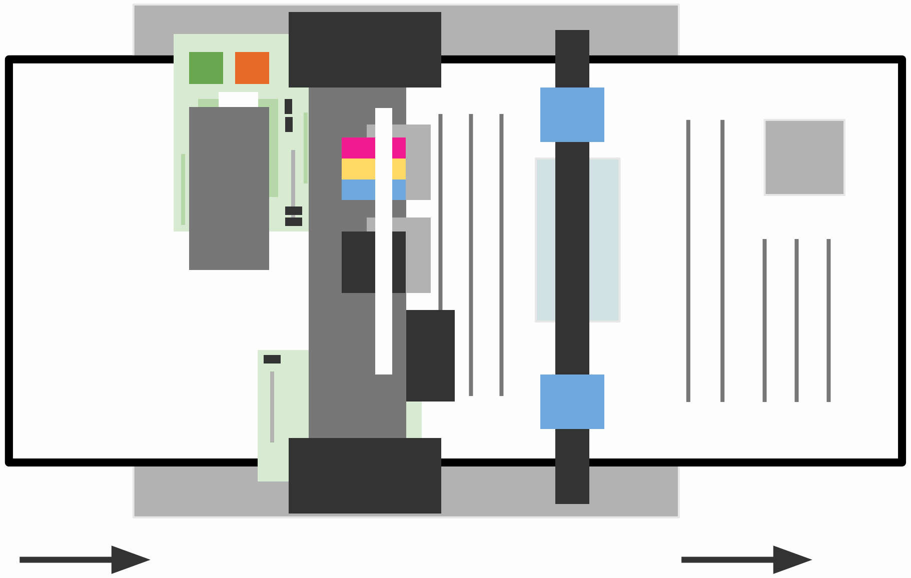
Strategic Data Hiding

Manage which variables are accessible to the public



Strategic Data Hiding

Manage which variables are accessible to the public



Why not show the parts of a printer?

Reasons to Encapsulate

Code Security

Prevent unauthorized read or write operations to sensitive data and processes within the code

Simplification

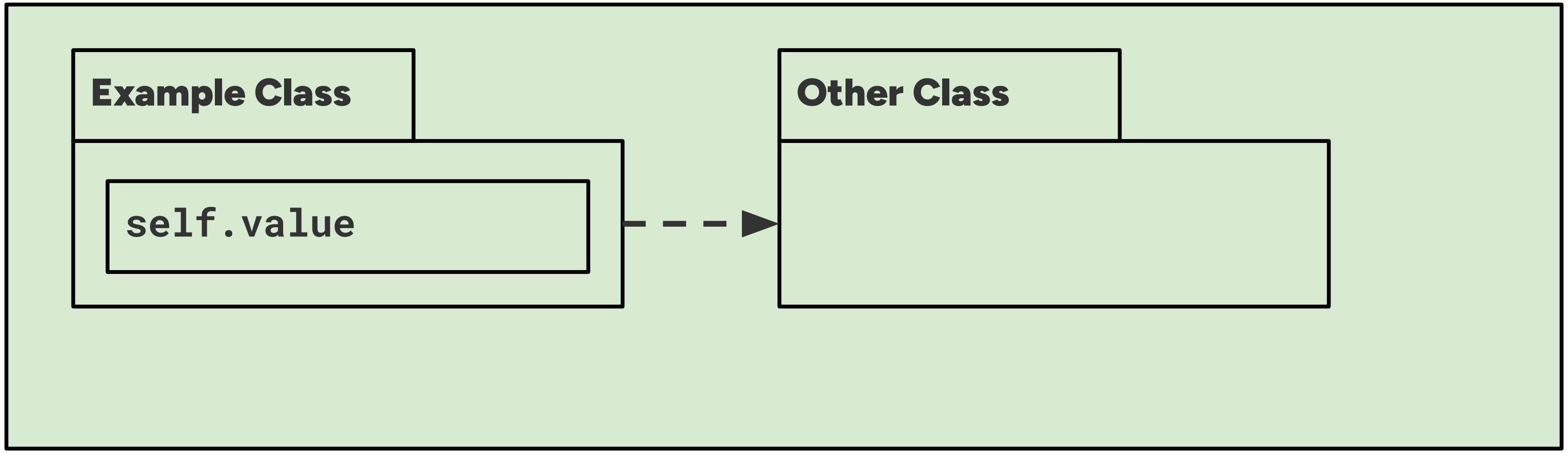
Not every detail of a process needs to be known. Classes can set up their own logic to handle changes

Maintainability

Less access to data means less suspects when debugging problems or issues when developing

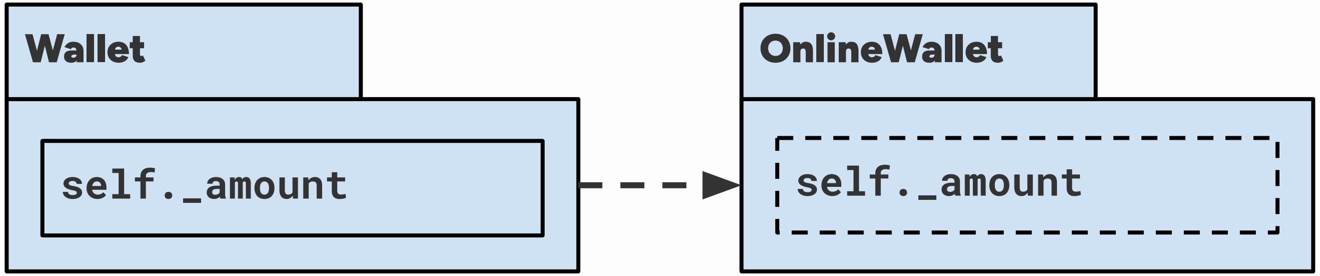
Public Attribute

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



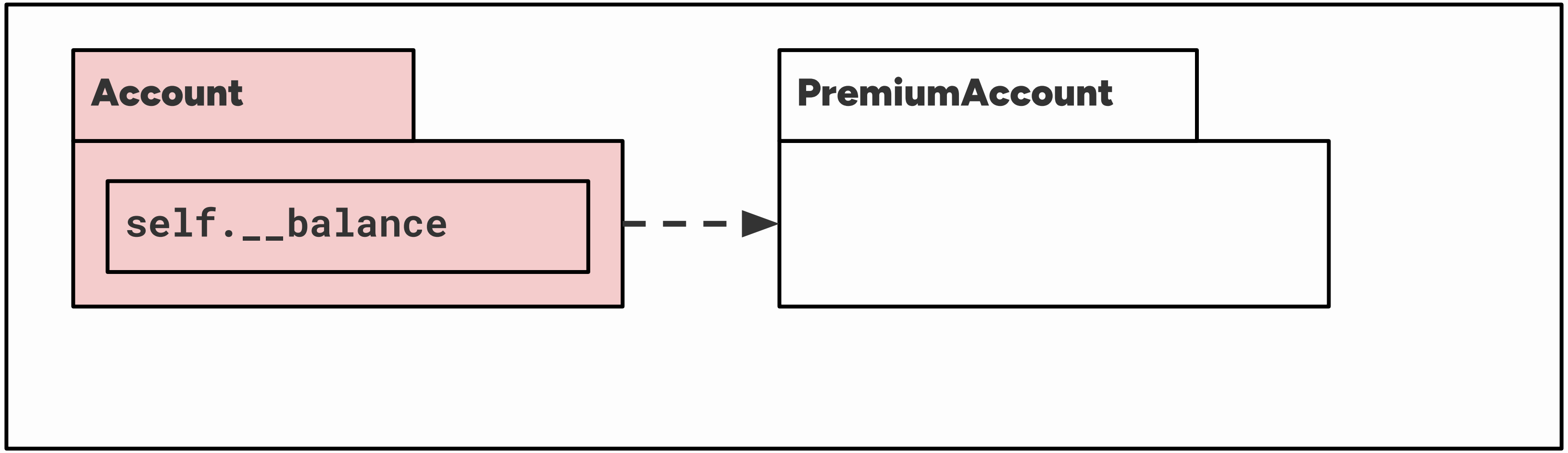
Protected Attribute

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



Private Attribute

```python
class Account:
    def __init__(self, initial_balance=0):
        self.__balance = initial_balance
```



Secured Wallet

secured_wallet.py

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

food_wallet = SecuredWallet(250)
print("Food Budget:", food_wallet.amount)
```

Secured Wallet

secured_wallet.py

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

    def get_amount(self):
        print(f"Showing amount: {self._amount}")
        return self._amount

food_wallet = SecuredWallet(250)
print("Food Budget:", food_wallet.get_amount())
```

Secured Wallet

secured_wallet.py

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

    @property
    def amount(self):
        print(f"Showing amount: {self._amount}")
        return self._amount

food_wallet = SecuredWallet(250)

print("Food Budget:", food_wallet.amount)
```

secured_wallet.py

```python
class SecuredWallet:
    def __init__(self, initial_amount=0):
        self._amount = initial_amount
    @property
    def amount(self):
        print(f"Showing amount: {self._amount}")
        return self._amount

food_wallet = SecuredWallet(250)
print("Food Budget:", food_wallet.amount)
```

wallet.py

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

food_wallet = Wallet(250)
print("Food Budget:", food_wallet.amount)
```

Secured Wallet

secured_wallet.py

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

     @property
     def amount(self):
          print(f"Showing amount: {self._amount}")
          return self._amount

     @amount.setter
     def amount(self, amount):
          print(f"Setting amount to {amount}")
          self._amount += amount

food_wallet = Wallet(250)
food_wallet.amount += 1_000

print("Food Budget:", food_wallet.amount)
```

H4

Safe Banking

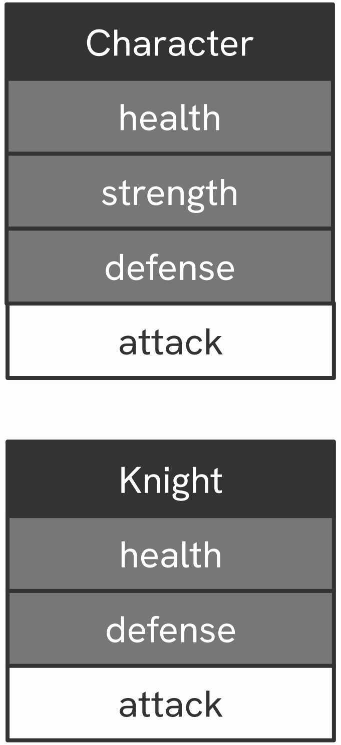
Refactor: Bank Account

```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)
```

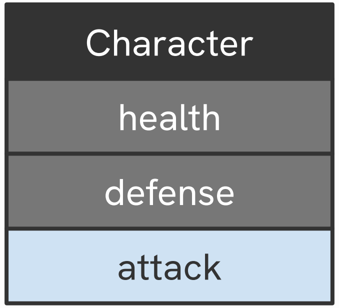
Abstraction

Contractual Implementation

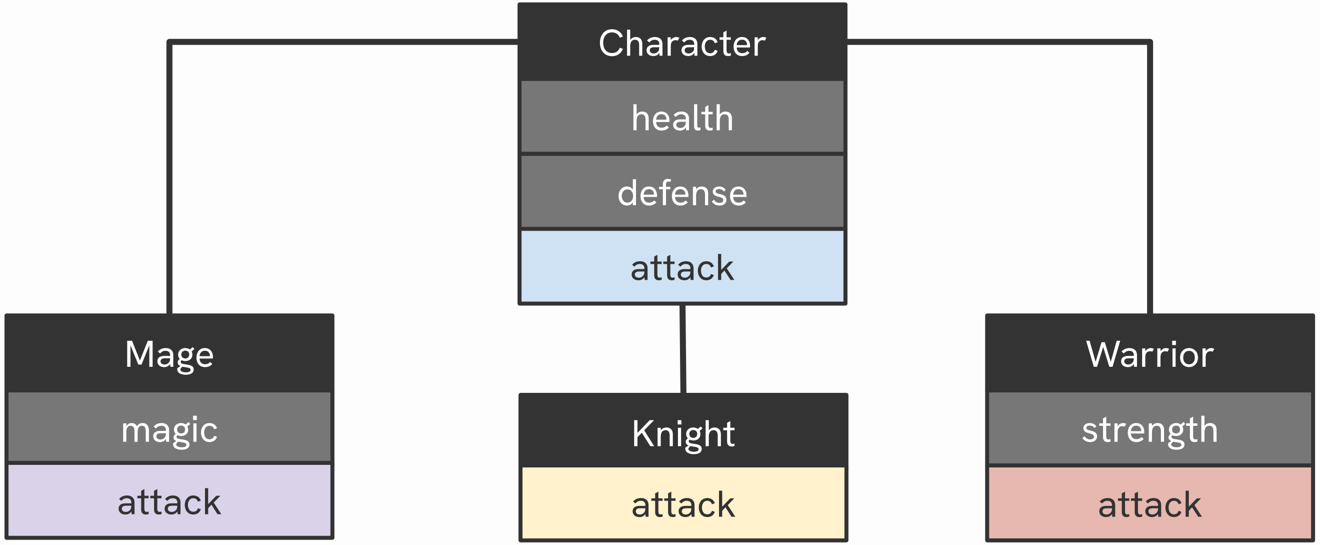
Recall: Game Character



Character Scheme



Character Scheme



Initial Implementation

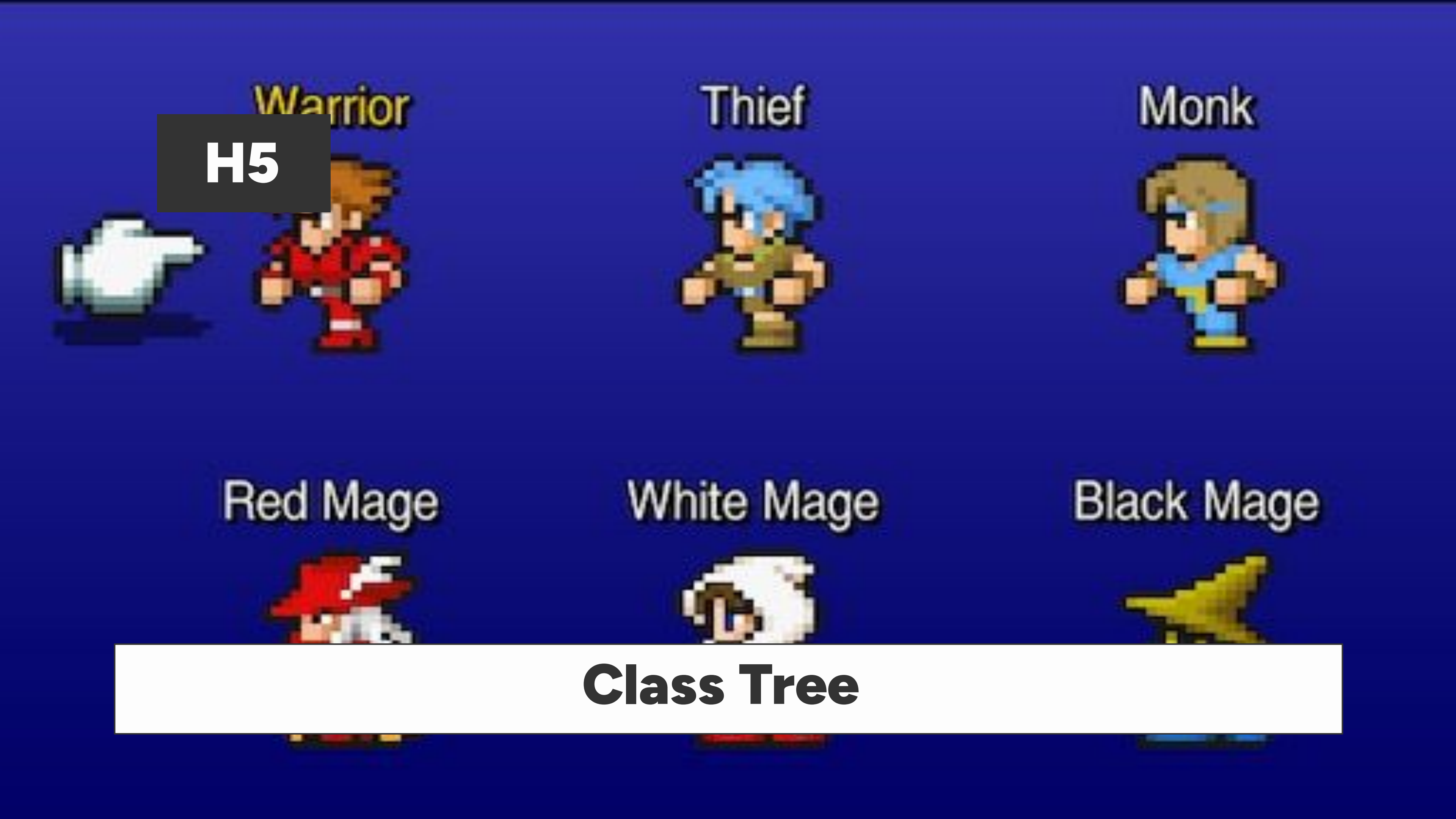
rpg.py

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

class Knight(Character):
    pass

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

Formal Polymorphism

rpg.py

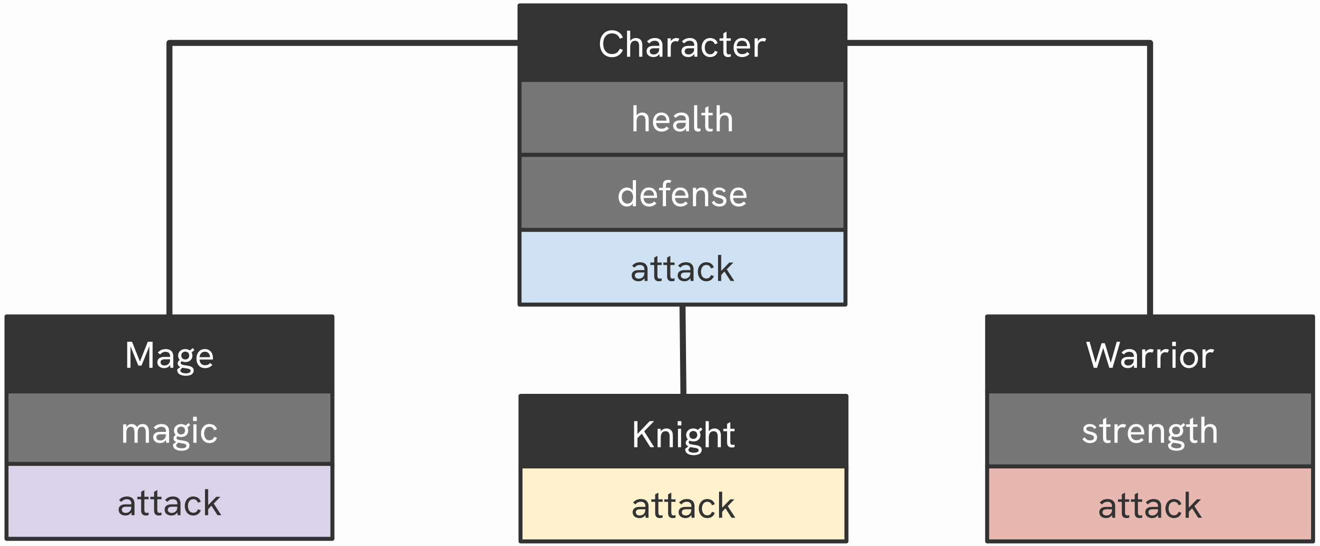
```python
from abc import ABC, abstractmethod

class Character(ABC):
    def __init__(self, health=10, 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
```



H5

Class Tree

Character Scheme



`rpg.py`

Custom Exception

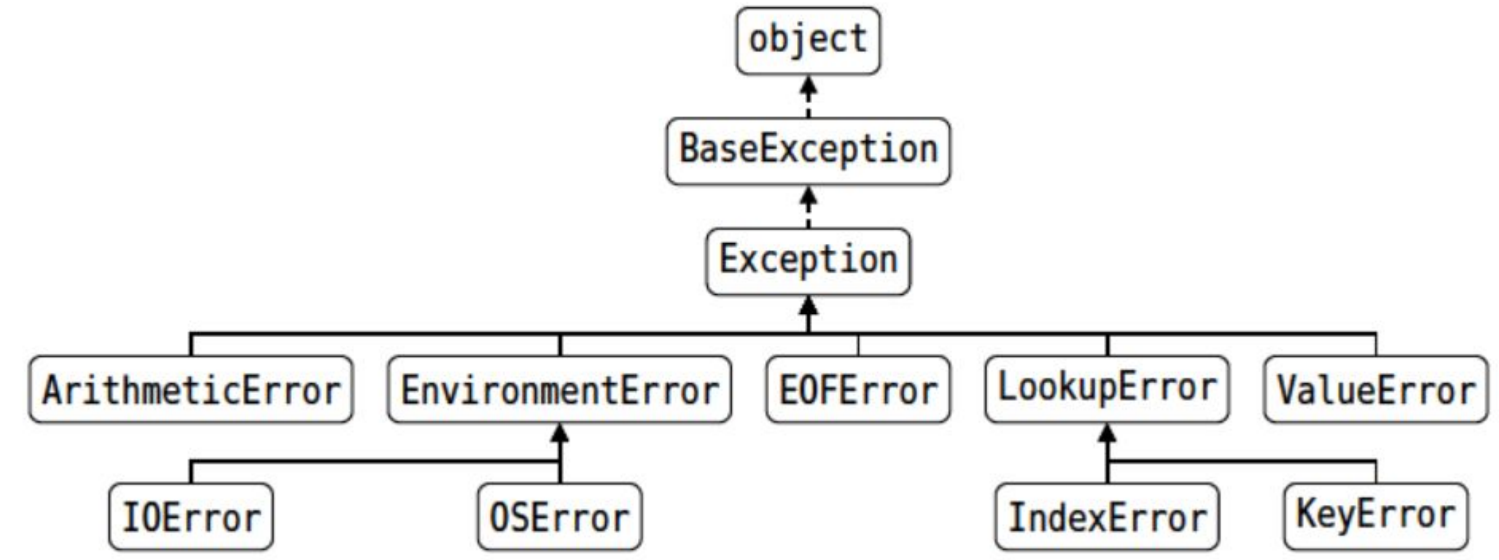
Create your own errors

Custom Error

custom_error.py

```python
class CustomError(Exception):
    pass

raise CustomError("yikes")
```

Exception Hierarchy Excerpt



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()
```

Roughly Equivalent Error

custom_error.py

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

raise CustomError("yikes")
```

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
```

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
__str__	None	String	Used in **`str()`** and **`print()`**
__eq__	Any	Boolean	Sets behavior for **==** operations
__add__	Any	Any	Sets behavior for **+** operations
__len__	None	Integer	Sets behavior when used in **`len()`**

Implement: Book

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

book1 = Book("The Hobbit", "Fantasy", "Tolkien")
book2 = Book("Dune", "Sci-Fi", "Herbert")
print(book1)
```

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

Magic Method __repr__

The `__repr__` dunder method defines what is used if the object is printed

```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}"

book1 = Book("The Hobbit", "Fantasy", "Tolkien")
book2 = Book("Dune”, "Sci-Fi", "Herbert")
print(book1)
```

```
The Hobbit [Fantasy] - Tolkien
```

Magic Method __add__

The **__add__** dunder method defines the result when an **+** operation is used with the object

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

    def __add__(self, other):
        new_amount = self.amount + other.amount
        return Wallet(new_amount )

food_wallet = Wallet(250)
transport_wallet = Wallet(1000)
total_wallet = food_wallet + transport_wallet

print("Food Budget: ", food_wallet.amount)
print("Transport Budget: ", transport_wallet.amount)
print("Total Budget: ", total_wallet.amount)
```

Object Identity

Python uses the memory location of an object to check for equality

```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)
```

Magic Method __eq__

The **__eq__** dunder method defines whether two objects are equal (or not)

```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)
```

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

```python
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()
```

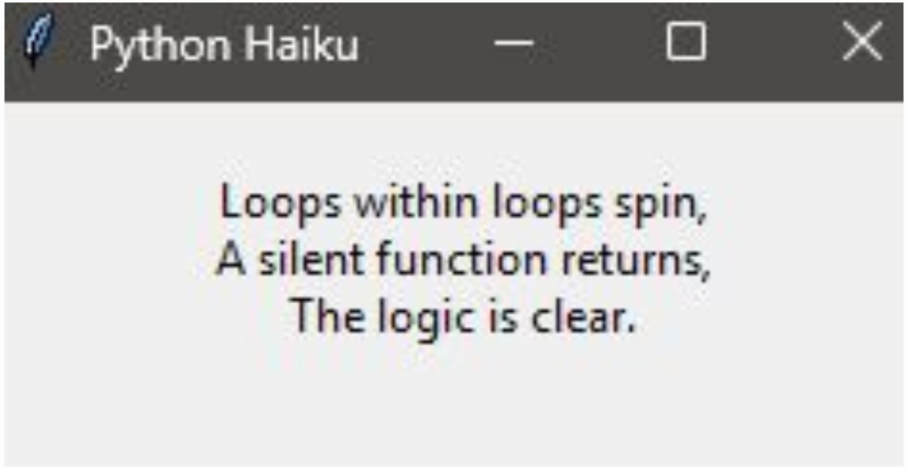
Multiline Label

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

```
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()
```

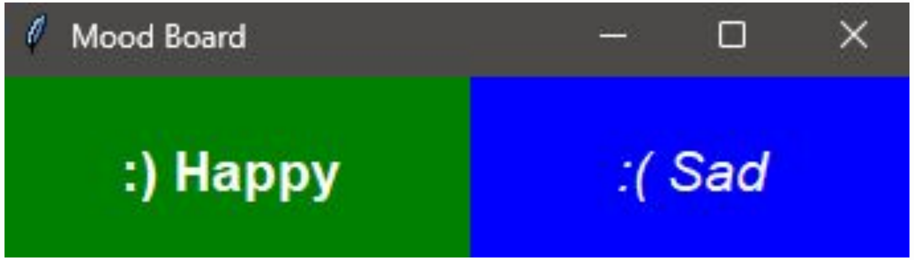
Component Pack Side

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)



:) Happy

:(Sad

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

```
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()
```

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

```
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

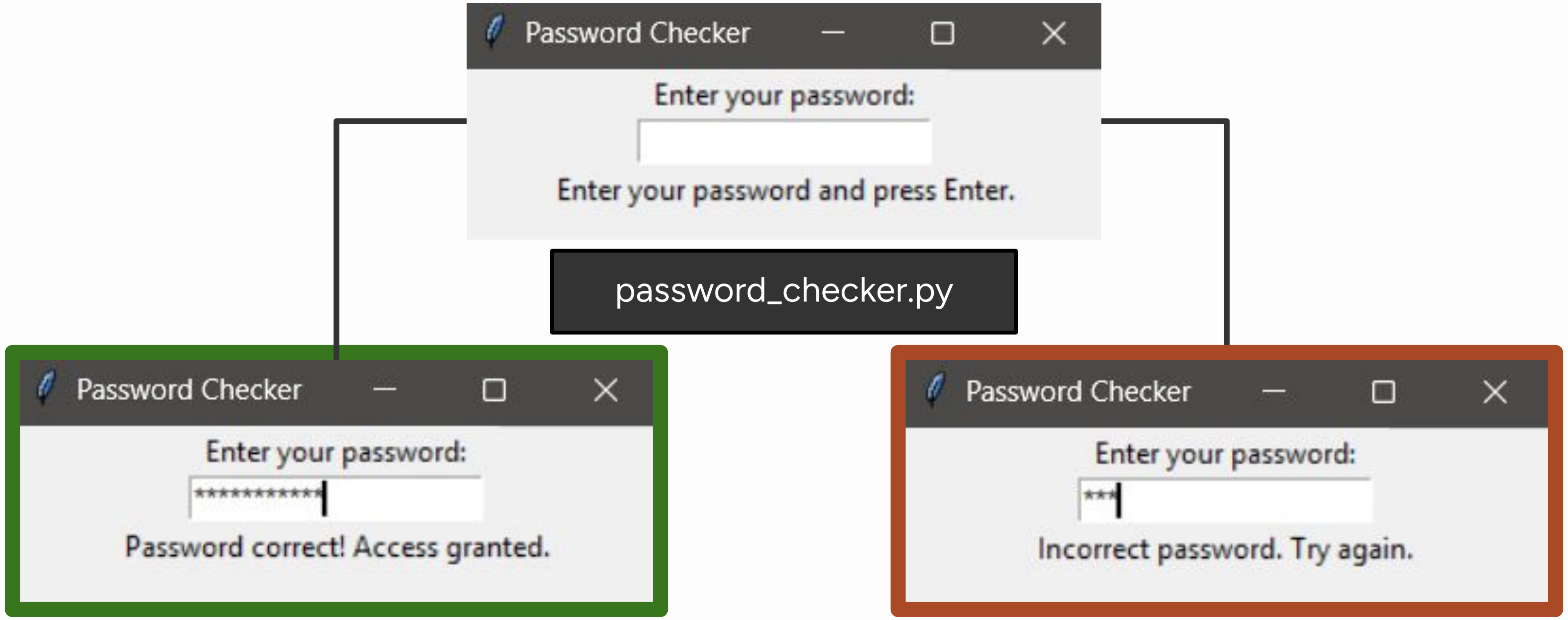
```python
import tkinter

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)
...
```

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

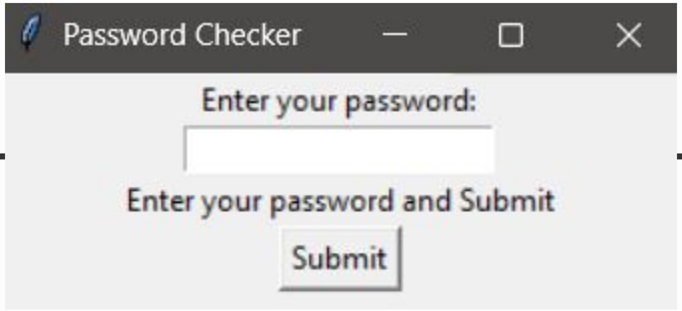
Trigger functions on command

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()
```

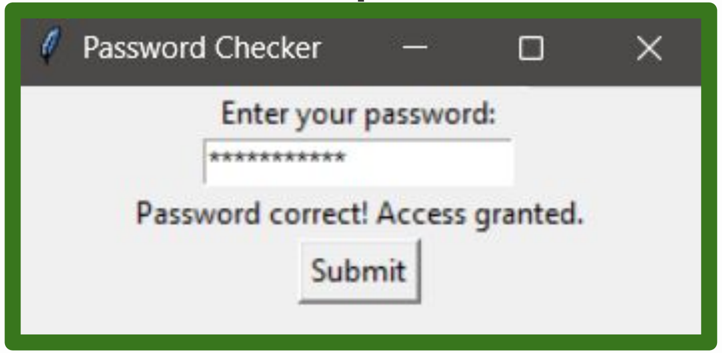
Quick Exercise: Password Checker



Enter your password:

Enter your password and Submit

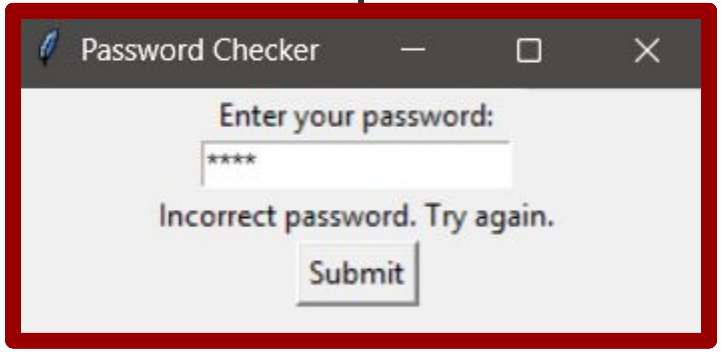
password_checker.py



Enter your password:

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

Password correct! Access granted.



Enter your password:

```
****
```

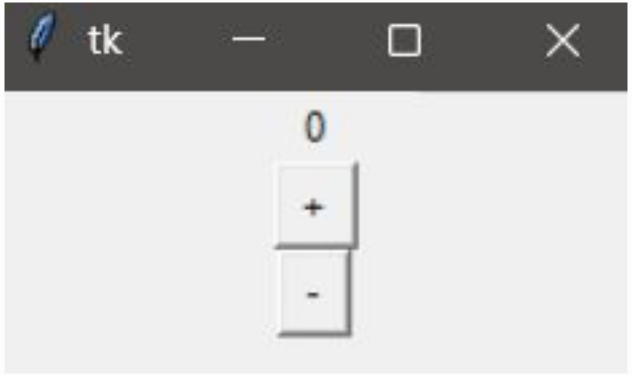
Incorrect password. Try again.

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



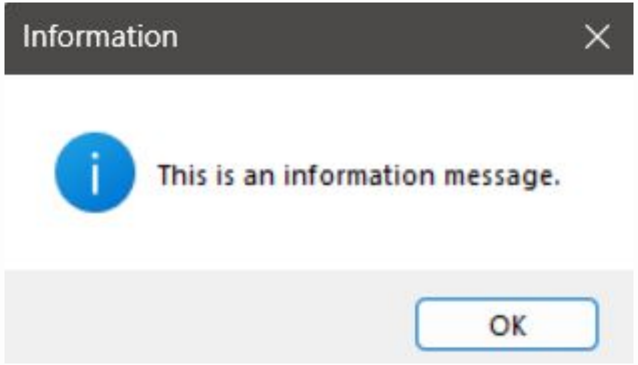
full_counter.py

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()
```

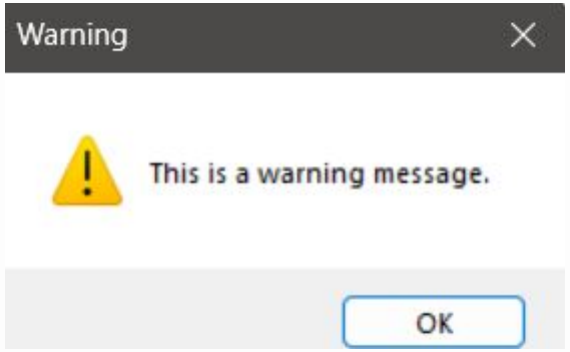

Information

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()
```

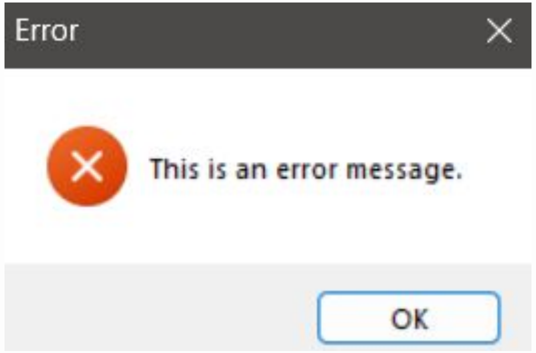

Warning

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()
```

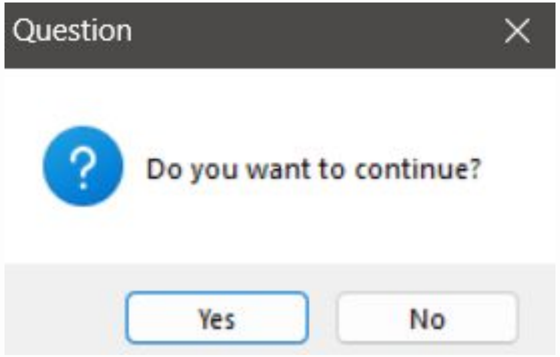

Error

This is an error message.

error_box.py

Question Message Box

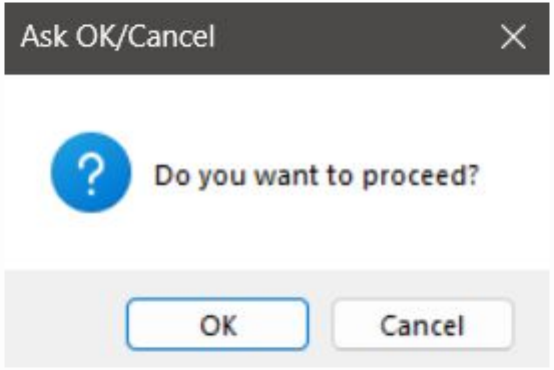
```python
import tkinter
from tkinter import messagebox

root = tkinter.Tk()

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

root.mainloop()
```



question_box.py

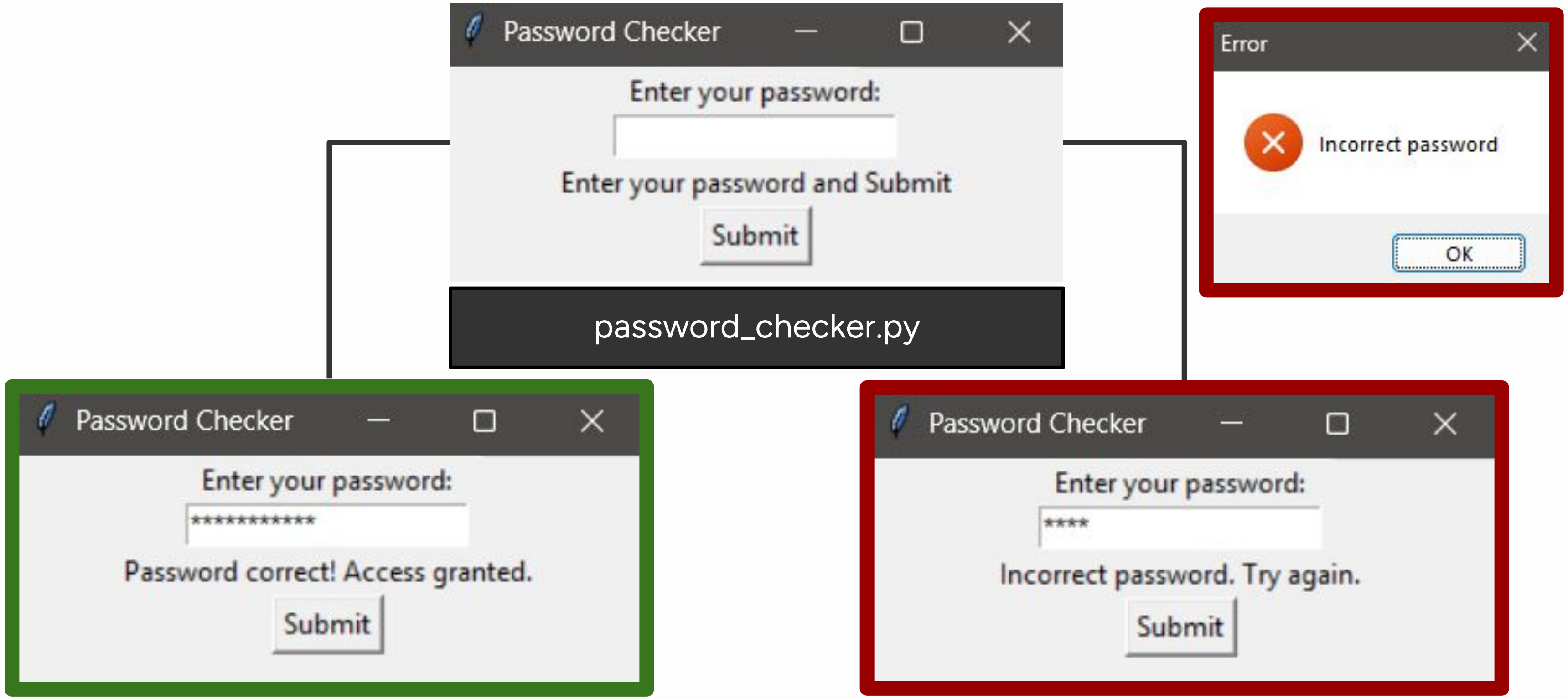
Ask OK Message Box

```
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



Input Components

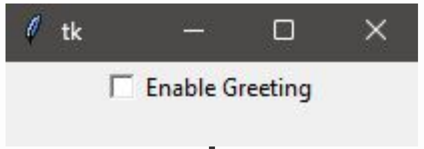
Other basic components for getting user data

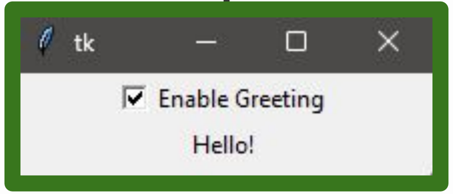
Checkbox

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





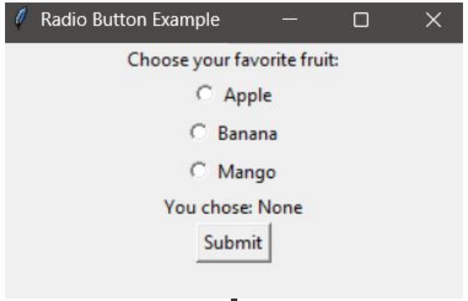
first_greeting.py

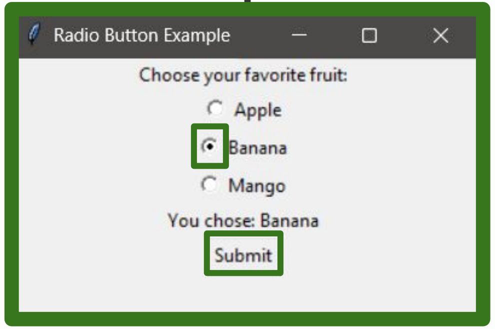
Radio Buttons

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


Radio Button Example
Choose your favorite fruit:
Apple
Banana
Mango
You chose: None
Submit


Radio Button Example
Choose your favorite fruit:
Apple
[✓ Banana]
Mango
You chose: Banana
Submit

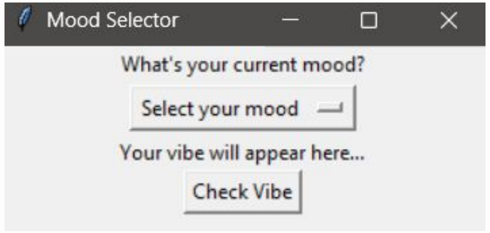
store select_.py

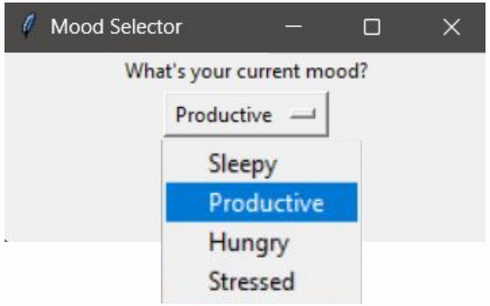
Dropdown

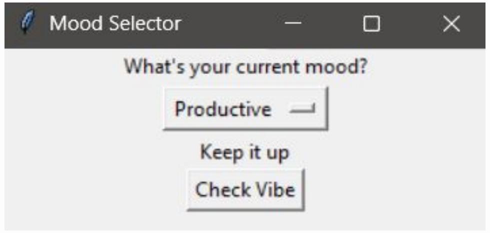
dropdown.py

```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







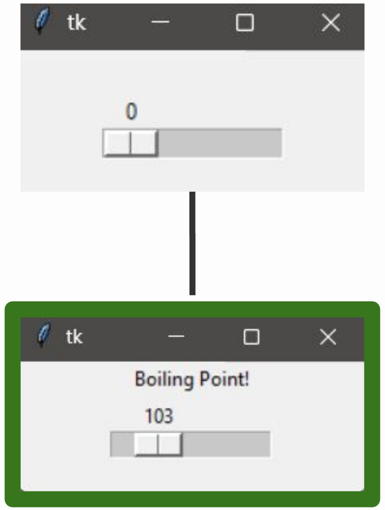
check_vibe.py

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()
```

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()
```

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

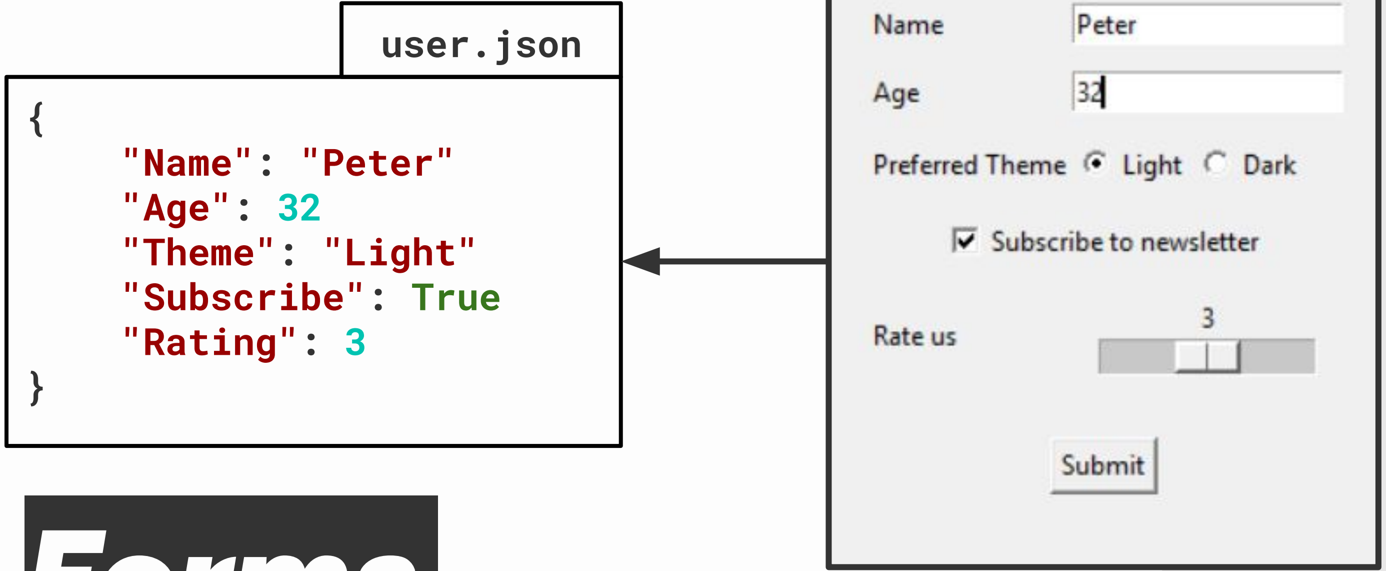
tkinter_class.py

```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



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

Name: Peter

Age: 32

Preferred Theme: [✓ Light] Dark

[✓] Subscribe to newsletter

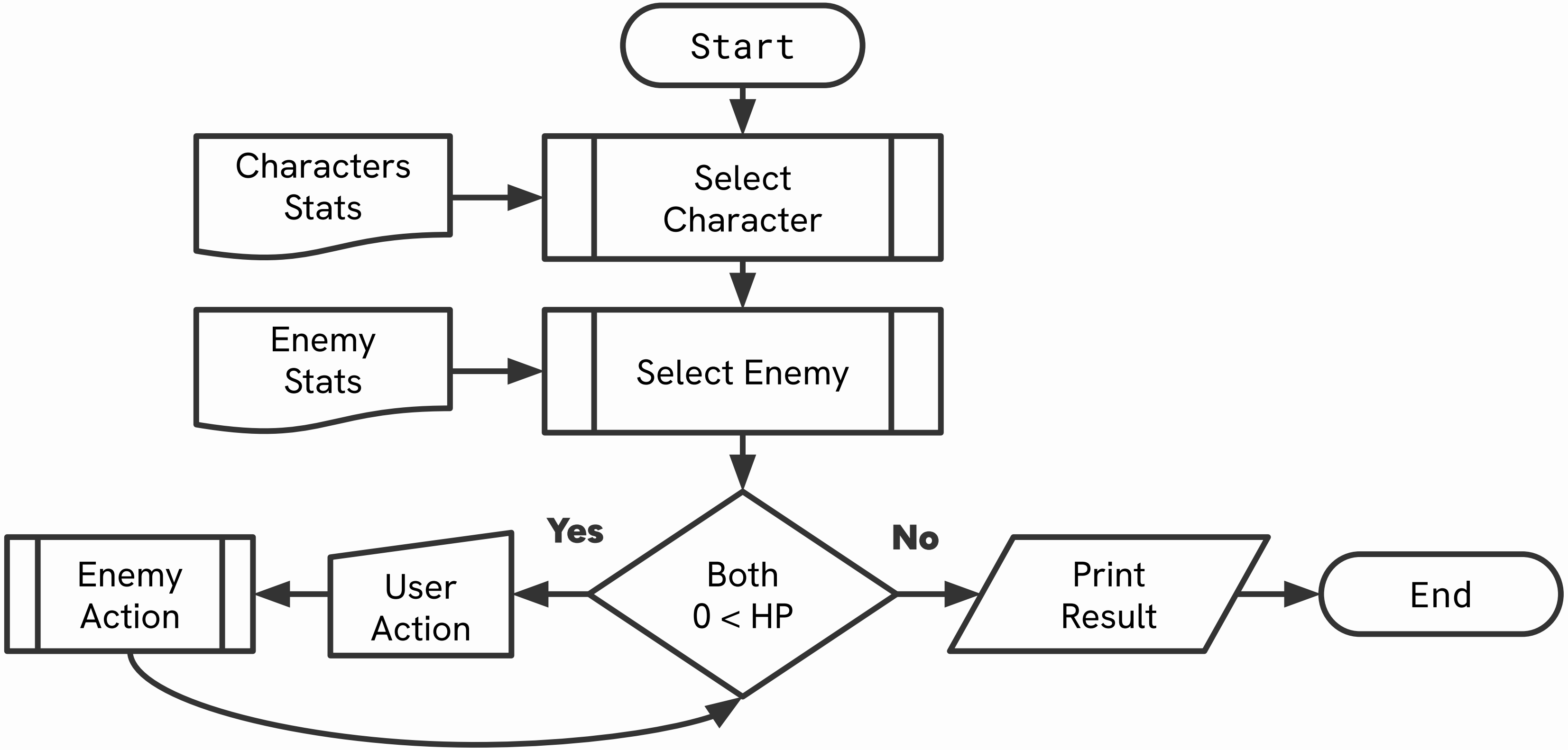
Rate us: 3

Forms

УЛИЦА
1905 ГОДА
СМОЛЕ
АРБАТСКАЯ
БОРОВИЦКАЯ
КРОПОТКИНСКАЯ
КИЕВСКАЯ
КИЕВСКАЯ
ПАРК КУЛЬТУРЫ
ОКТЯБРЬ

Battle!

Battle! Game Flow



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