

# **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

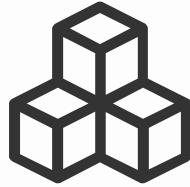
**What makes  
something  
something ?**







# **What is Object Oriented Programming?**



**Object**



**Attributes**

Object's data

**Methods**

Object's actions

**Has → Is**

# Functional Identity



## Attributes

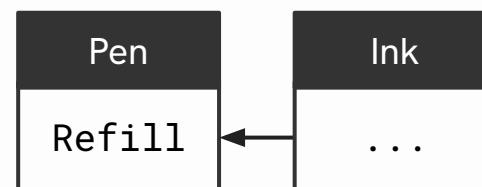
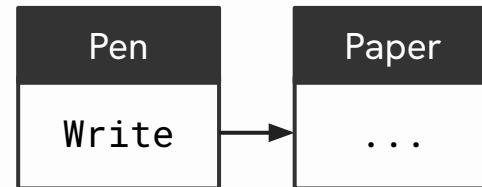
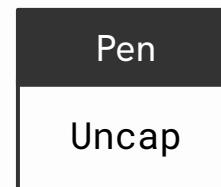
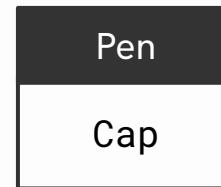
- Attributes are unique to one object

Pen	
brand	Pilot
color	Black
capped	False



## Methods

- Methods can change itself or others



# Object Similarities

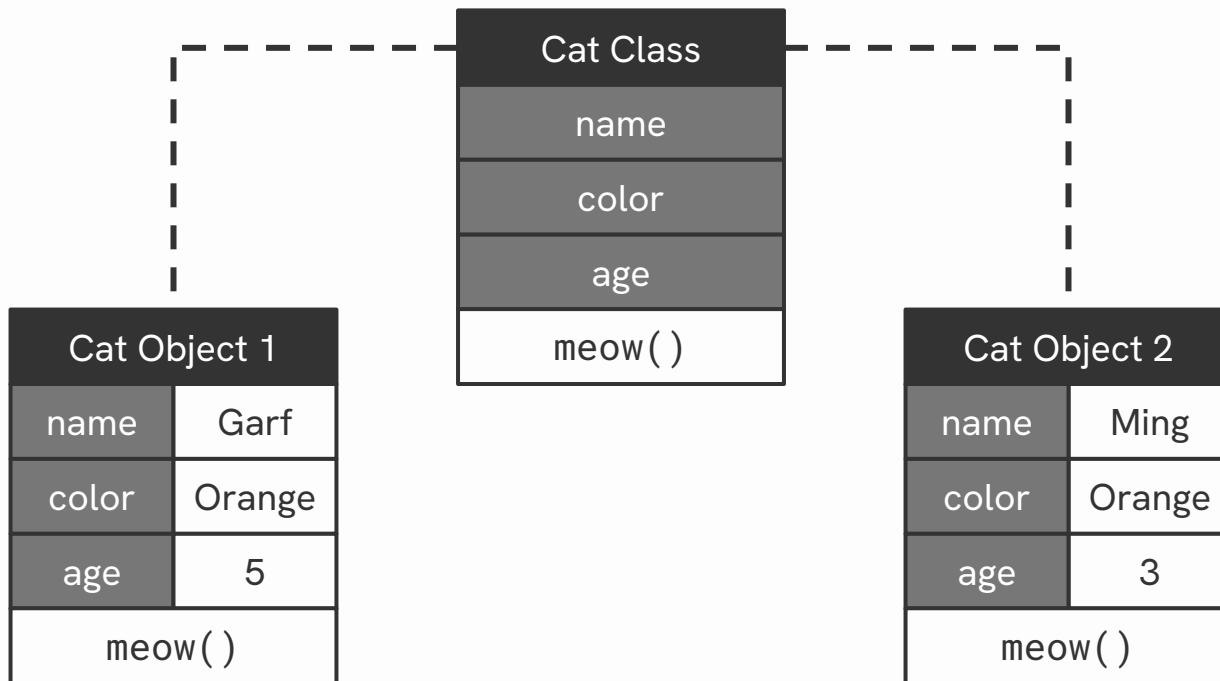
cat1	
name	Garf
color	Orange
age	5
meow()	

cat2	
name	Ming
color	Orange
age	3
meow()	

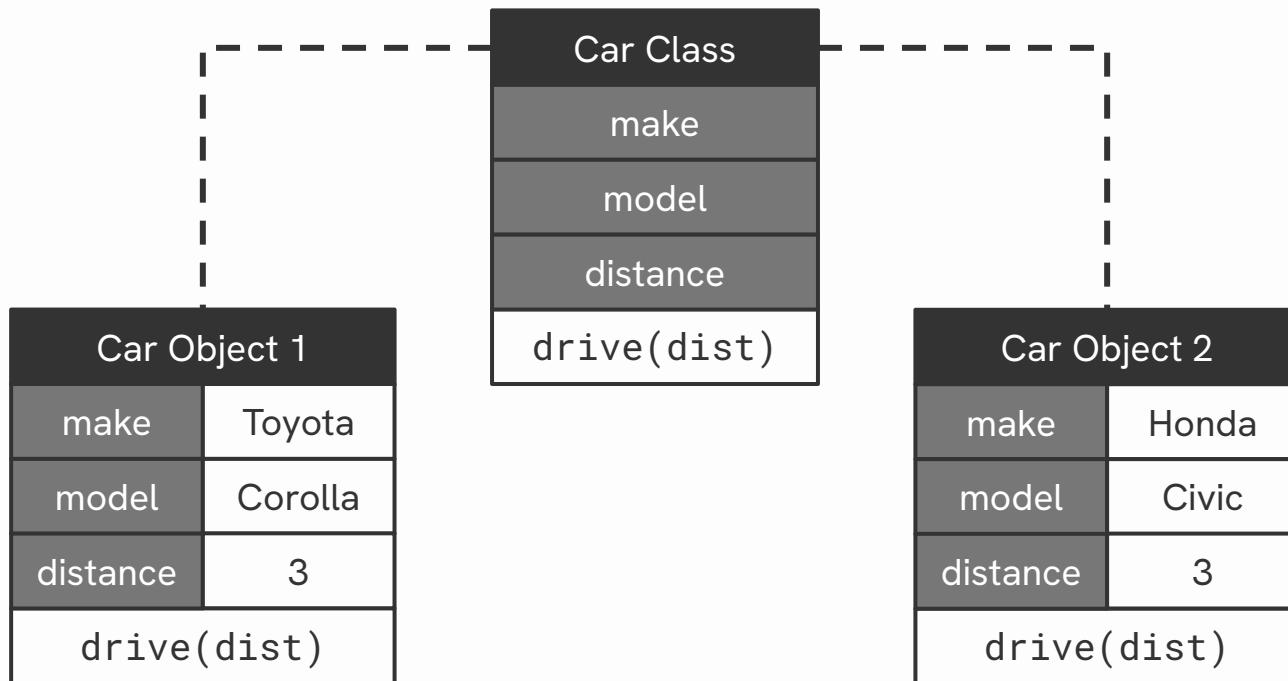
cat3	
name	Mona
color	Black
age	2
meow()	

**What makes them different/same?**

# Classes to Objects



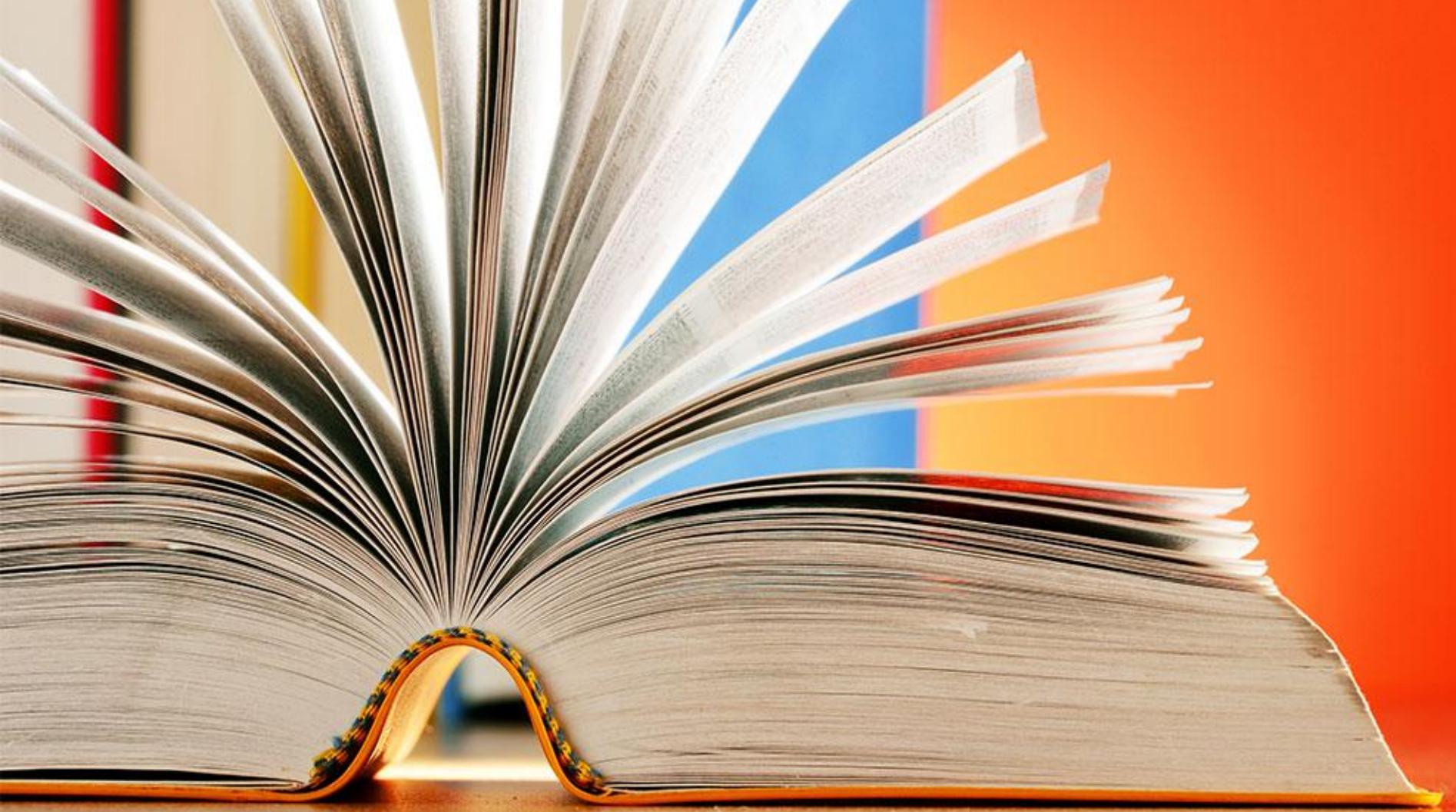
# Classes to Objects





## Modelling Exercise





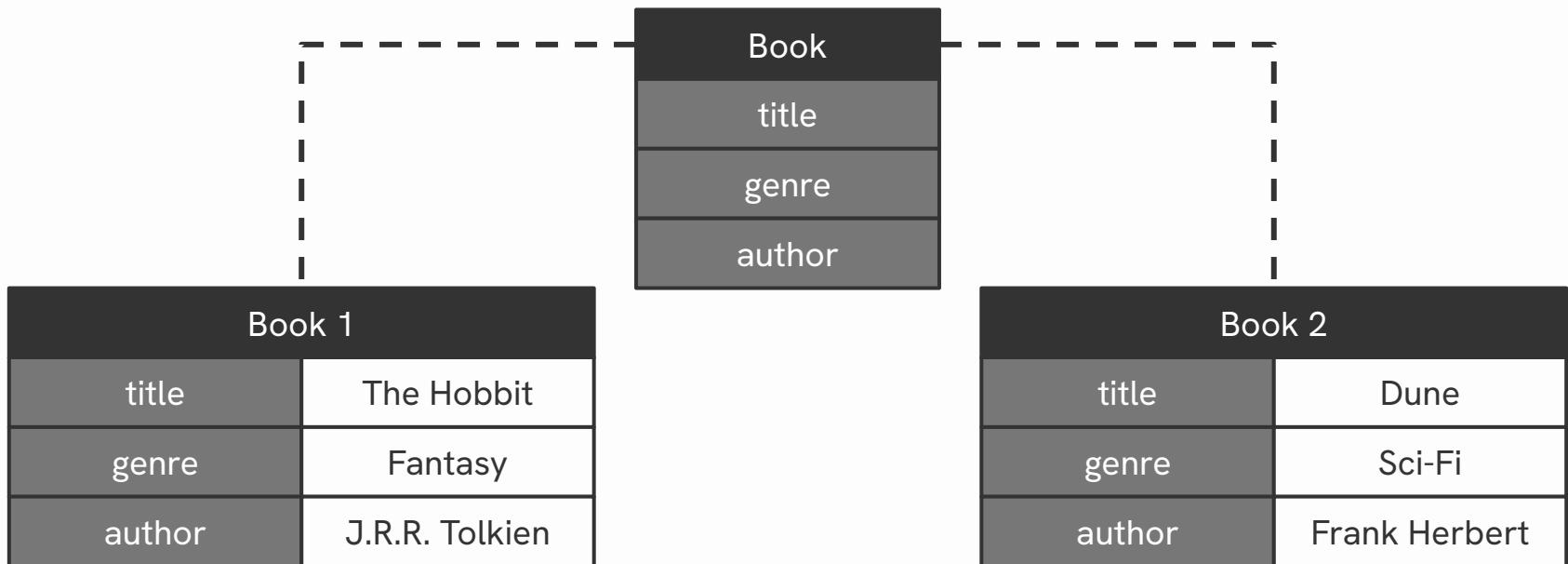




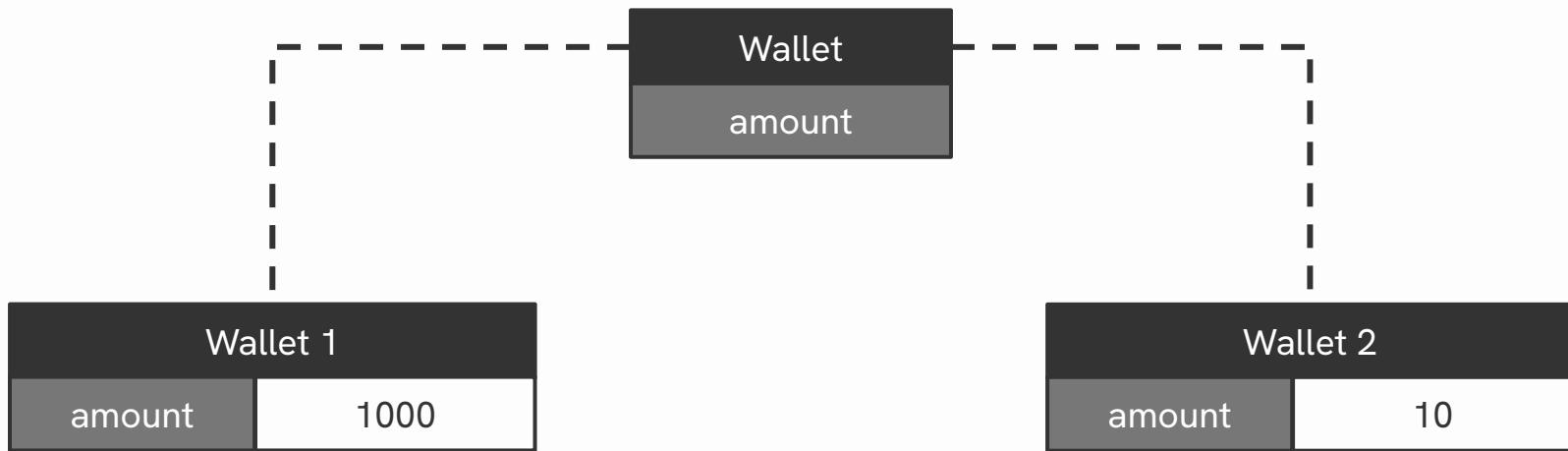
**BPI**



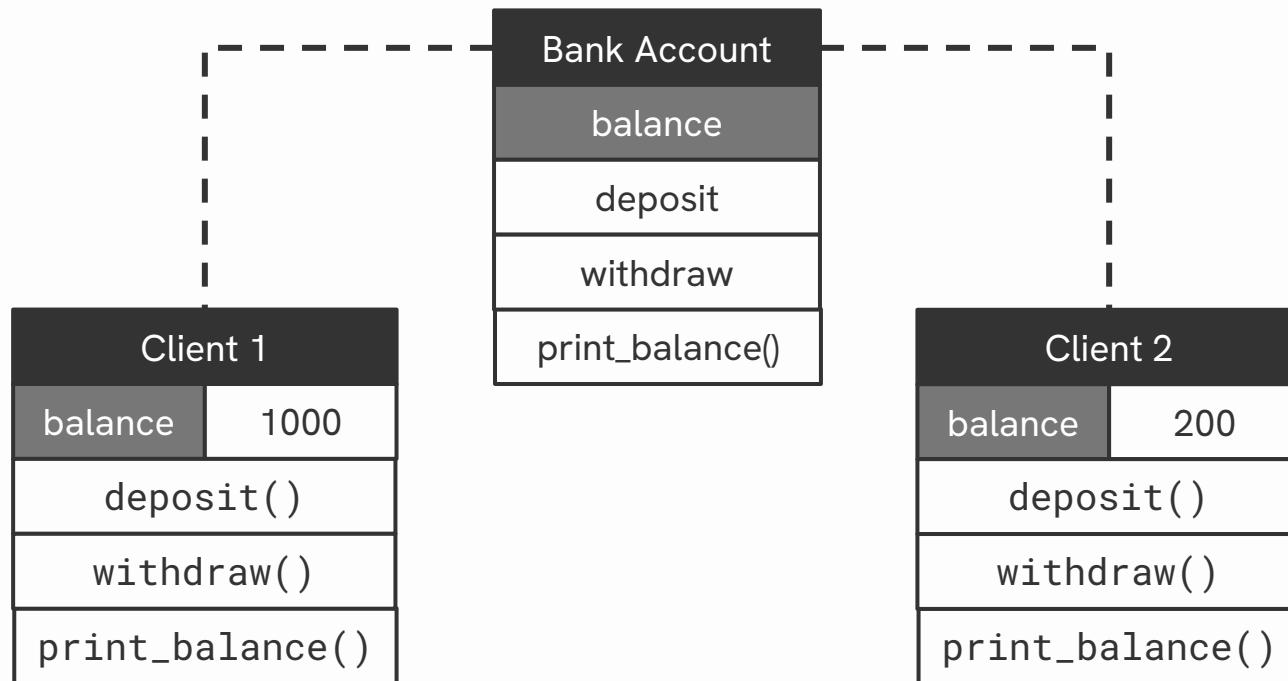
# Book



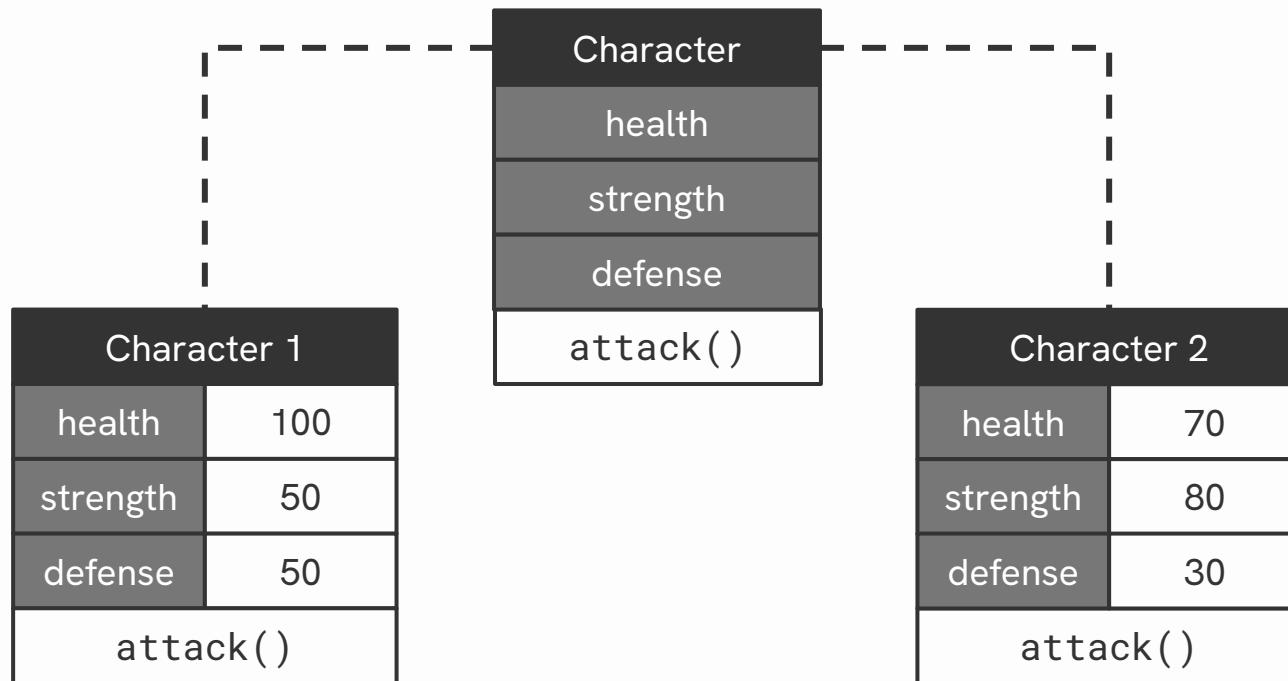
# Wallet

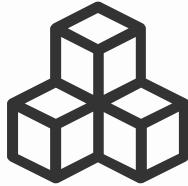


# Bank Account

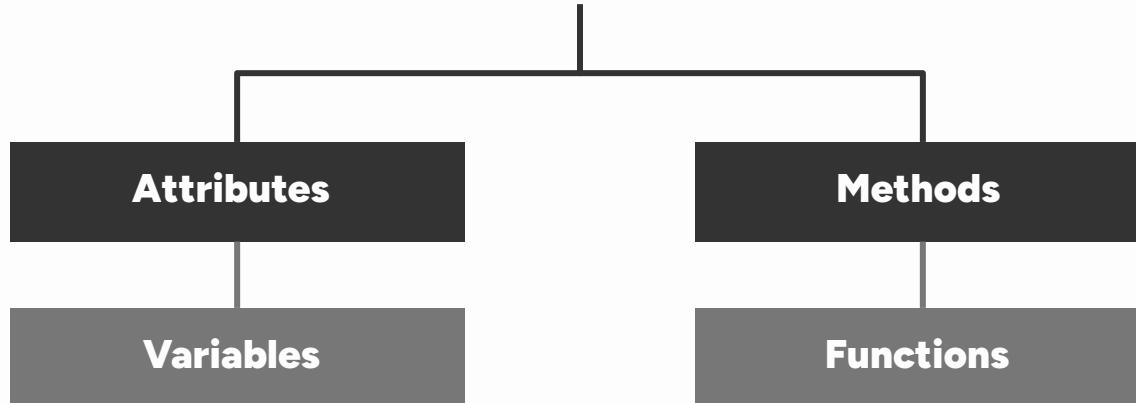


# Game Character



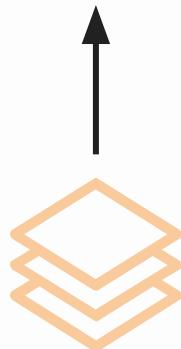


## Object

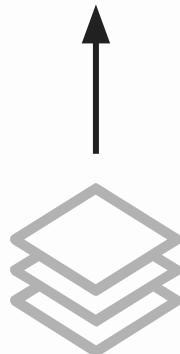


# Functional Approach

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



paper\_color



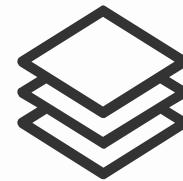
paper\_color



painter\_color

# Object Oriented Approach

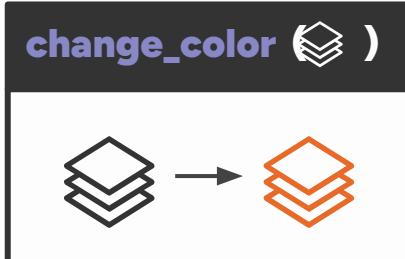
```
marker.change_color(paper)
```

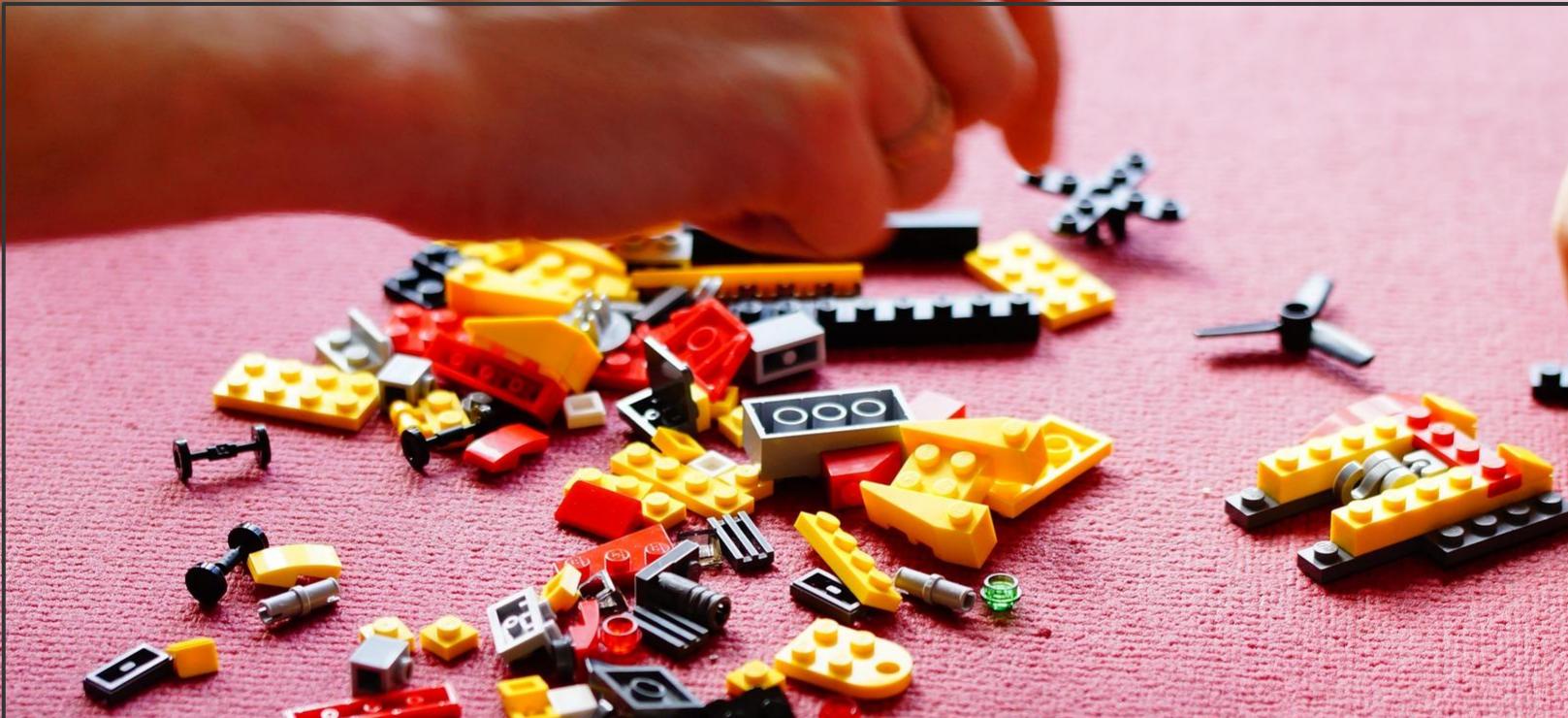


Color: Orange



Color: Orange





## Building Exercise

# Example Class

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15
```

# Object Creation

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3  
4     employee1 = Employee()  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15
```

# Multiple Object Creation

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3  
4     employee1 = Employee()  
5     employee2 = Employee()  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15
```

# Class Constructor

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3     def __init__(self):  
4         print("Employee created")  
5  
6     employee1 = Employee()  
7     employee2 = Employee()  
8  
9  
10  
11  
12  
13  
14  
15
```

# Class Constructor

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3     def __init__(self, name):  
4         print(f"Employee {name} created")  
5  
6     employee1 = Employee("Richard")  
7     employee2 = Employee("Jelly")  
8  
9  
10  
11  
12  
13  
14  
15
```

# Class Constructor

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3     def __init__(self, name, id):  
4         print(f"Employee {name} created with ID {id}")  
5  
6     employee1 = Employee("Richard", "1234")  
7     employee2 = Employee("Jelly", "9876")  
8  
9  
10  
11  
12  
13  
14  
15
```

# Object Attributes

employee.py

```
1 class Employee:  
2     """Class representation for employee data"""  
3     def __init__(self, name, id):  
4         self.name = name  
5         self.id = id  
6         print(f"Employee {self.name} created with ID {self.id}")  
7  
8     employee1 = Employee("Richard", "1234")  
9     employee2 = Employee("Jelly", "9876")  
10    print("Employee 1 Name:", employee1.name)  
11    print("Employee 2 Name:", employee2.name)  
12  
13  
14  
15
```

# Object Attributes

**self .name**

**employee1 .name**

# Object Methods

employee.py

```
1 class Employee:
2     """Class representation for employee data"""
3     def __init__(self, name, id):
4         self.name = name
5         self.id = id
6         self.tasks = []
7         print(f"Employee {self.name} created with ID {self.id}")
8
9     def add_work(self, task):
10        print(f"Added work {task} to {self.name}")
11        return self.tasks.append(task)
12
13
14 employee1 = Employee("Richard", "1234")
15 employee2 = Employee("Jelly", "9876")
16 employee1.add_work("Create Slides")
17 employee1.add_work("Present report")
```

# Object Methods

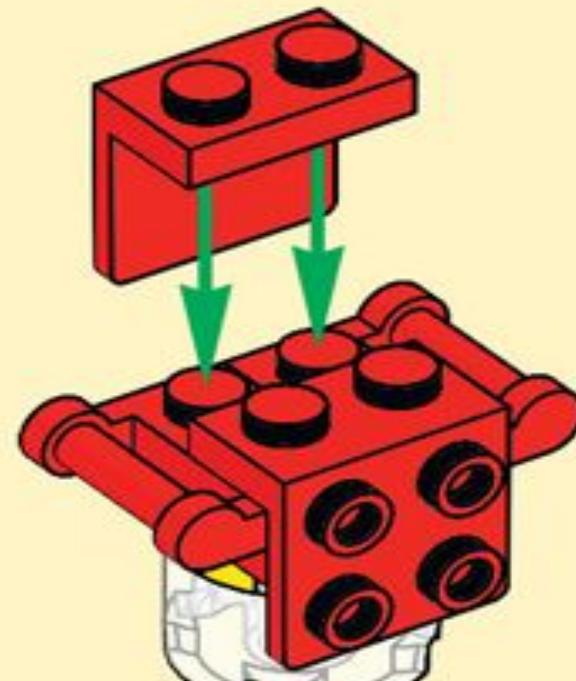
**employee .add\_work (task )**  
**add\_work (employee, task )**

1

H1



4

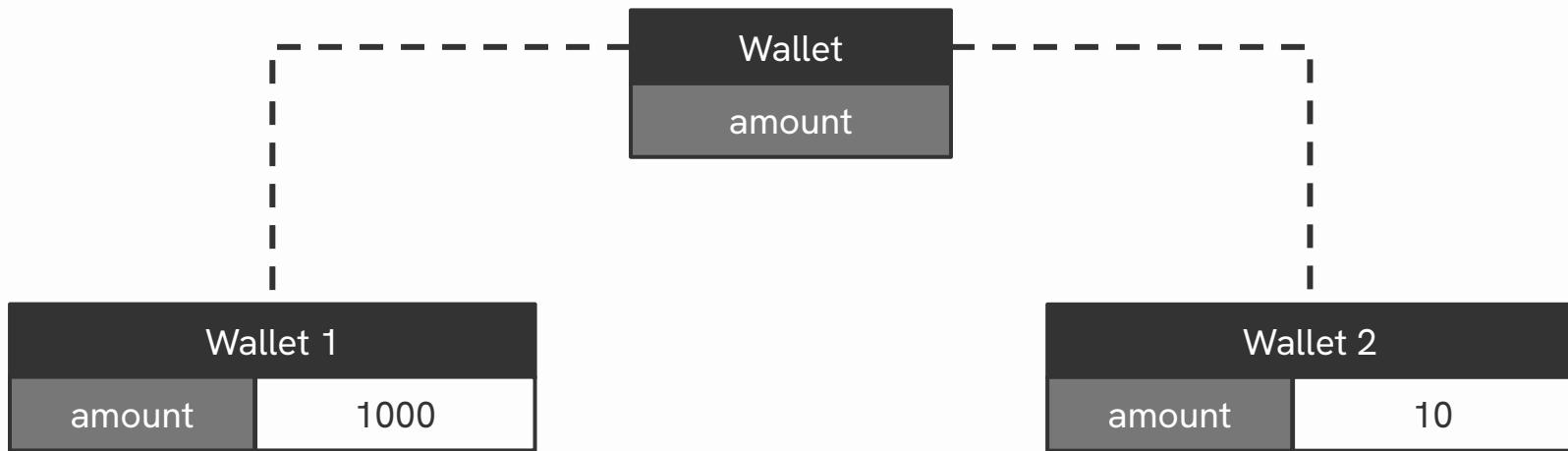


2

## Hands-On Building



# Wallet

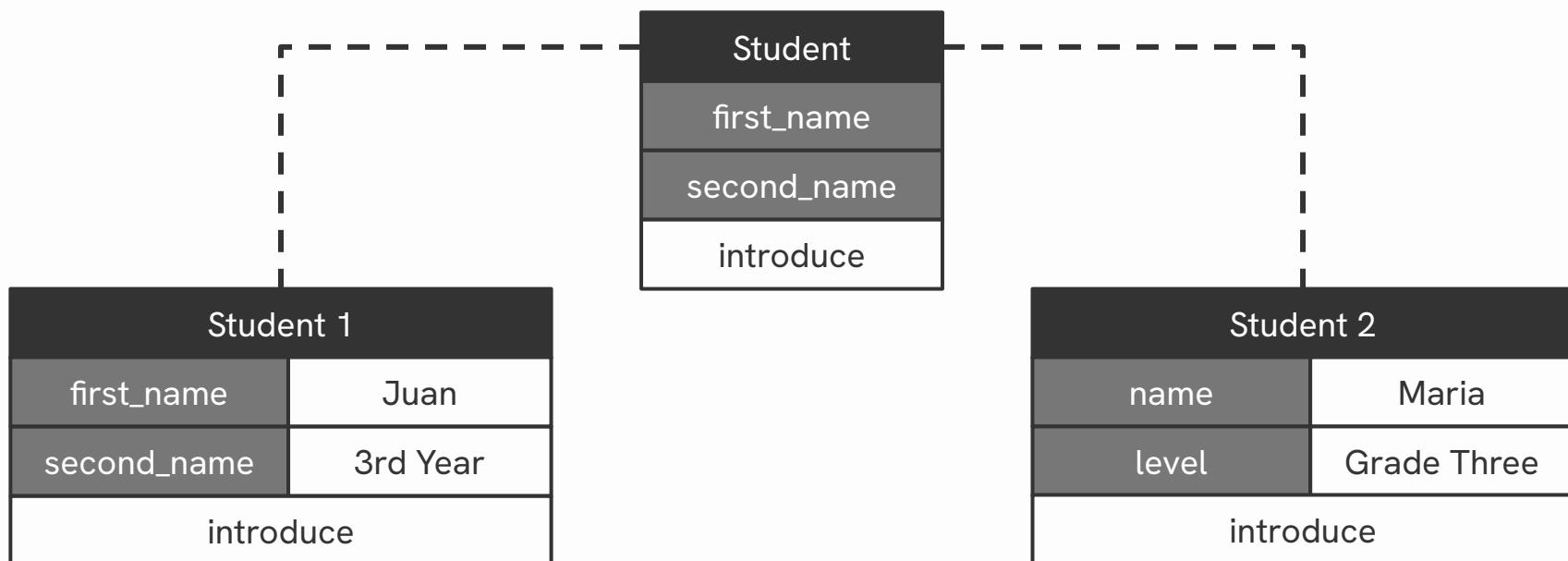


# Implement : Wallet

wallet.py

```
1 class Wallet:  
2     def __init__(self, initial_amount=0):  
3         self.amount = initial_amount  
4  
5 food_wallet = Wallet(250)  
6 food_wallet.amount += 1_000  
7  
8 print("Food Budget:", food_wallet.amount)
```

# Person

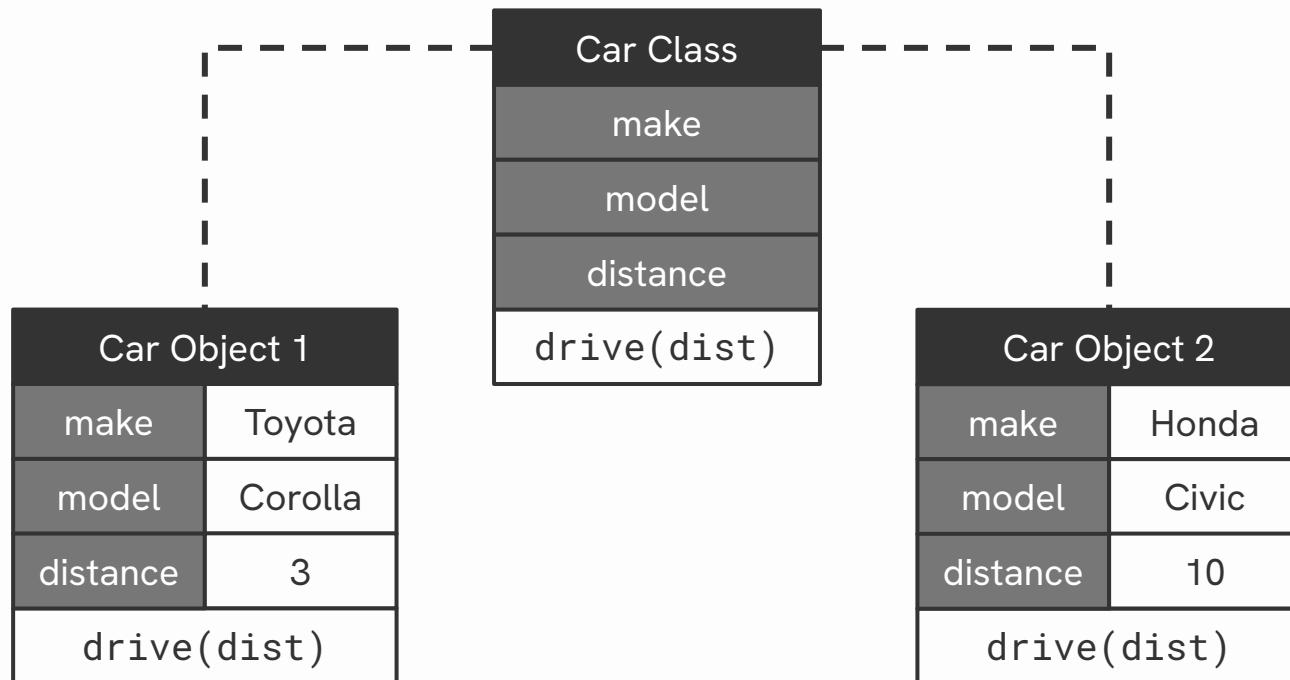


# Implement : Student

person.py

```
1 class Person:  
2     def __init__(self, first_name, last_name):  
3         self.first_name = first_name  
4         self.last_name = last_name  
5  
6     def sleep(self):  
7         print("I will sleep for eight hours")  
8  
9     def introduce(self):  
10        print(f"I'm {self.first_name} {self.last_name}!")
```

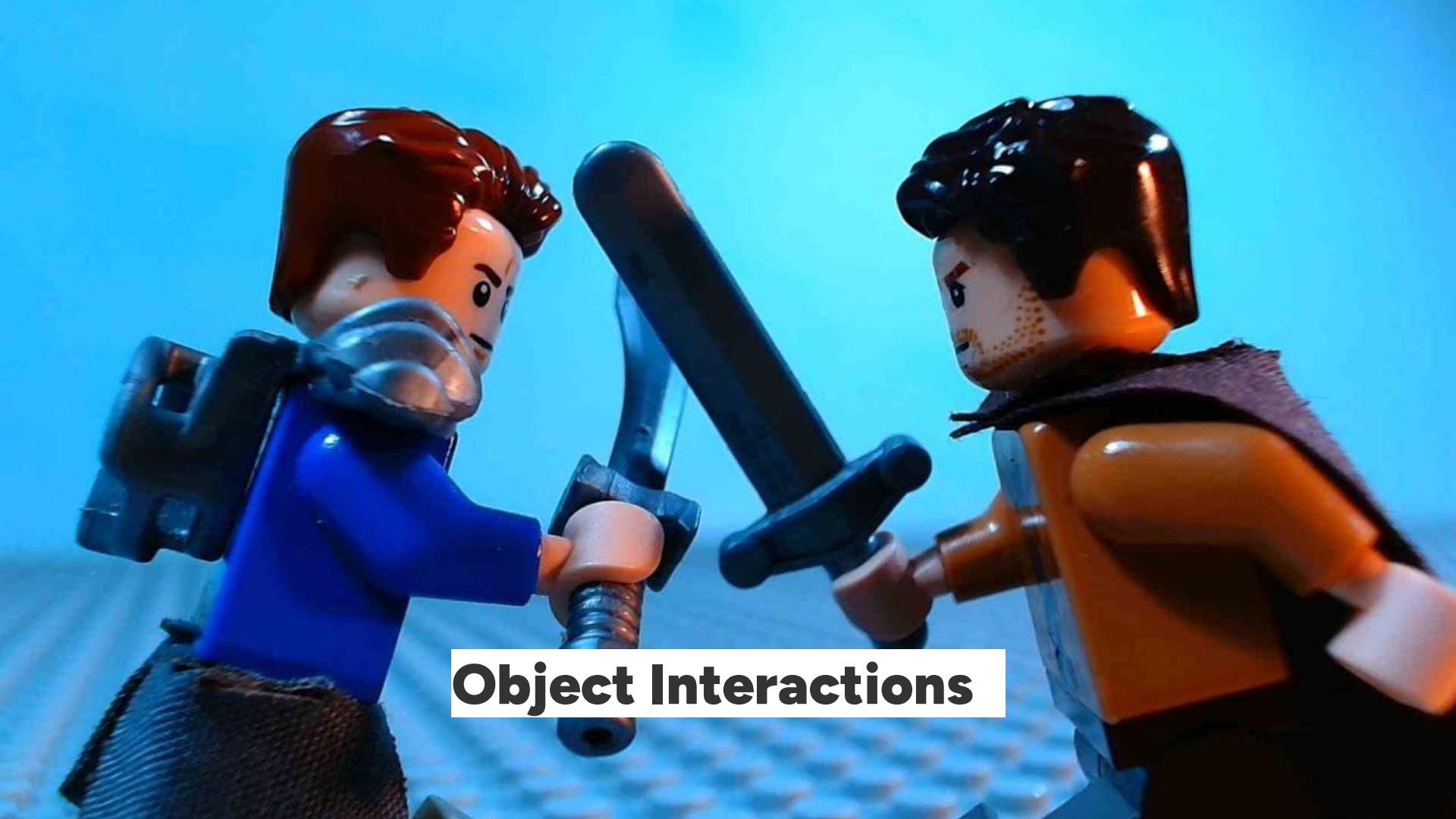
# Car



# Implement : Car

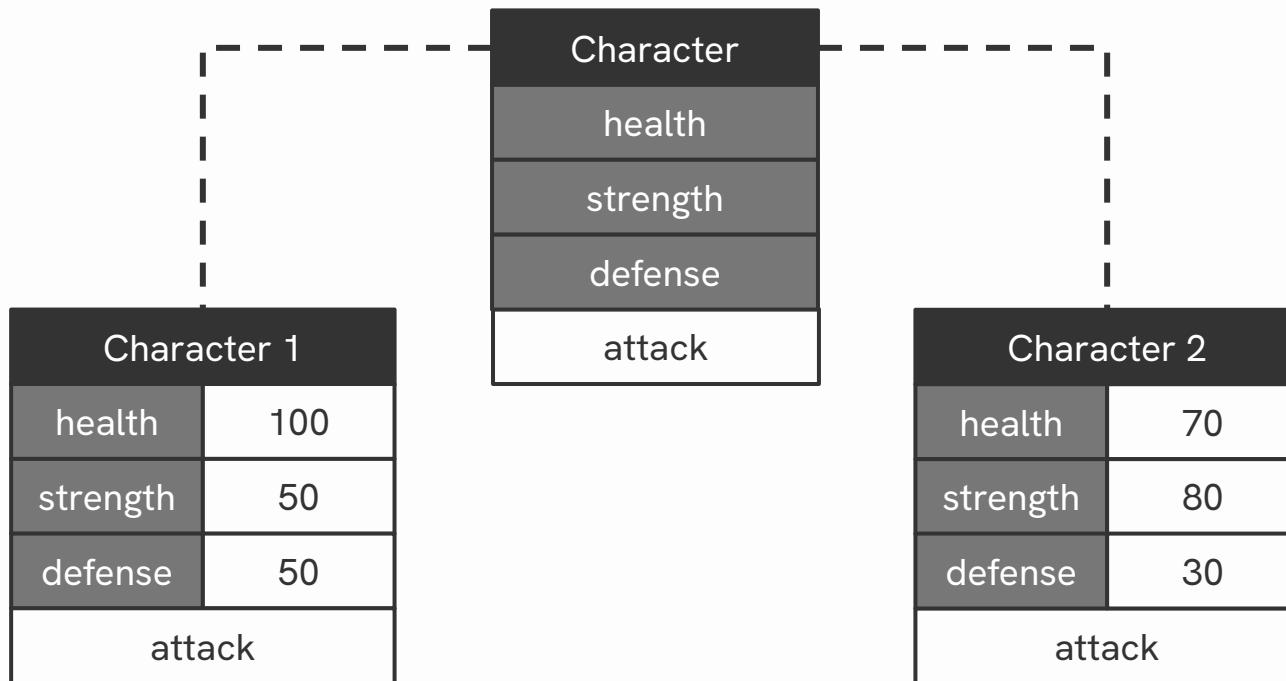
car.py

```
1 class Car:  
2     distance = 0  
3     def __init__(self):  
4         self.make = make  
5         self.model = model  
6         self.distance = 0  
7     def drive(self, dist):  
8         self.distance += dist  
9         Car.distance += dist  
10  
11 car_1 = Car("Toyota", "Corolla")  
12 car_1.drive(3)  
13  
14 car_2 = Car("Honda", "Civic")  
15 car_2.drive(10)
```

A photograph of two LEGO minifigures engaged in a sword fight. The figure on the left has brown hair and wears a blue tunic over a white shirt, holding a silver sword. The figure on the right has dark hair and wears an orange tunic over a white shirt, also holding a silver sword. They are positioned against a solid blue background.

# Object Interactions

# Game Character



# Implement: Character

character.py

```
1 class Character:
2     def __init__(self, health=10, strength=10, defense=10):
3         self.health = health
4         self.strength = strength
5         self.defense = defense
6
7     def attack(self, other):
8         damage = self.strength - other.defense
9         other.health -= damage
10
11 player = Character(strength=100)
12 enemy = Character()
13
14 player.attack(enemy)
15 print(enemy.health)
```



2x

62

**H2**

2x

6019155



4x

4226876



8x

6195183



2x

6273152



6x

6172366



1x

6092565



3x

302221



2x

4540040



2x

6096682



1x

6258135



4x

6117074



1x

6105976



3x

403224



4x

## Hands-Off Building



5x



3x



1x



1x



3x

# Implement: Bank Account

```
Bank Account
balance
def deposit(self, amount)
def withdraw(self, amount)
def print_balance(self)
```

bank\_account.py

02

# Hierarchy

Reducing redundancy in classes

# Inheritance

Explicit class structure

# Code Redundancy

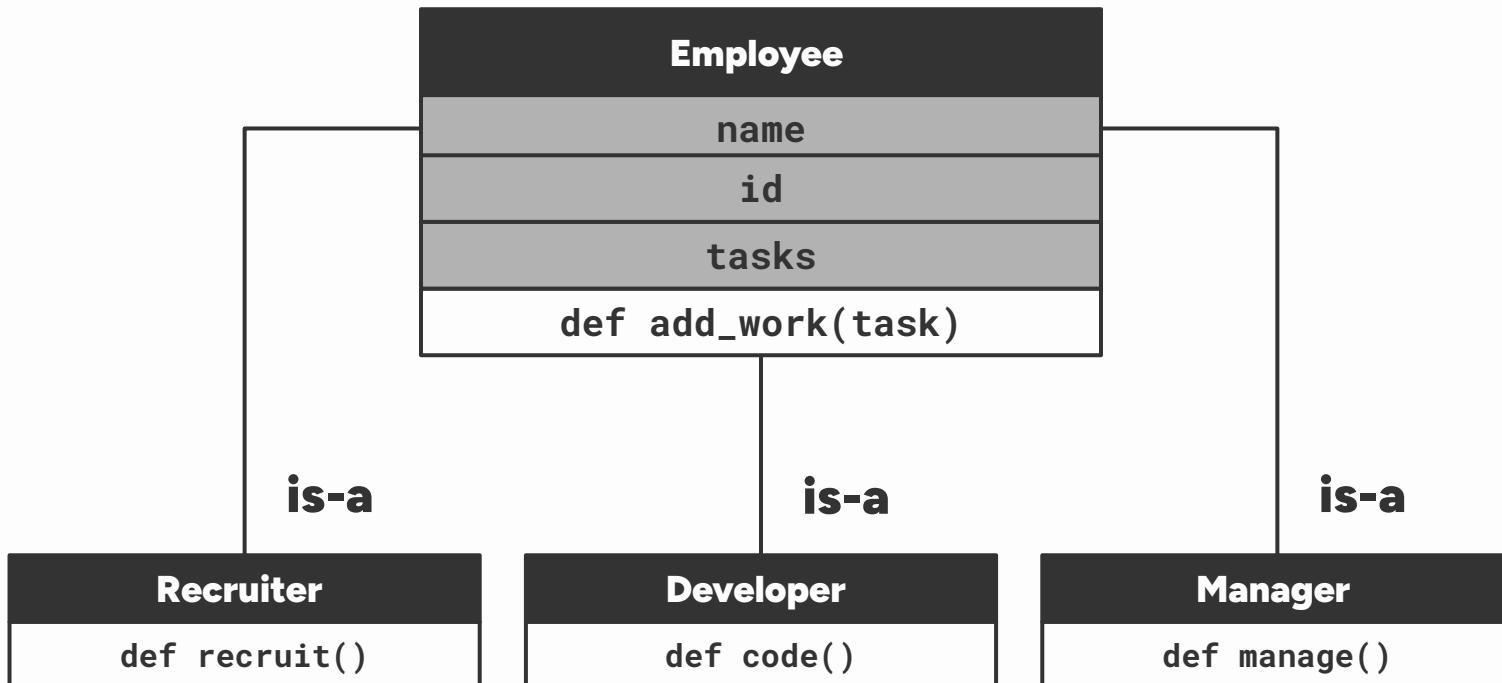
```
class Recruiter:  
    def __init__(self, name, id)  
    def add_work(self)  
    def recruit(self)
```

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

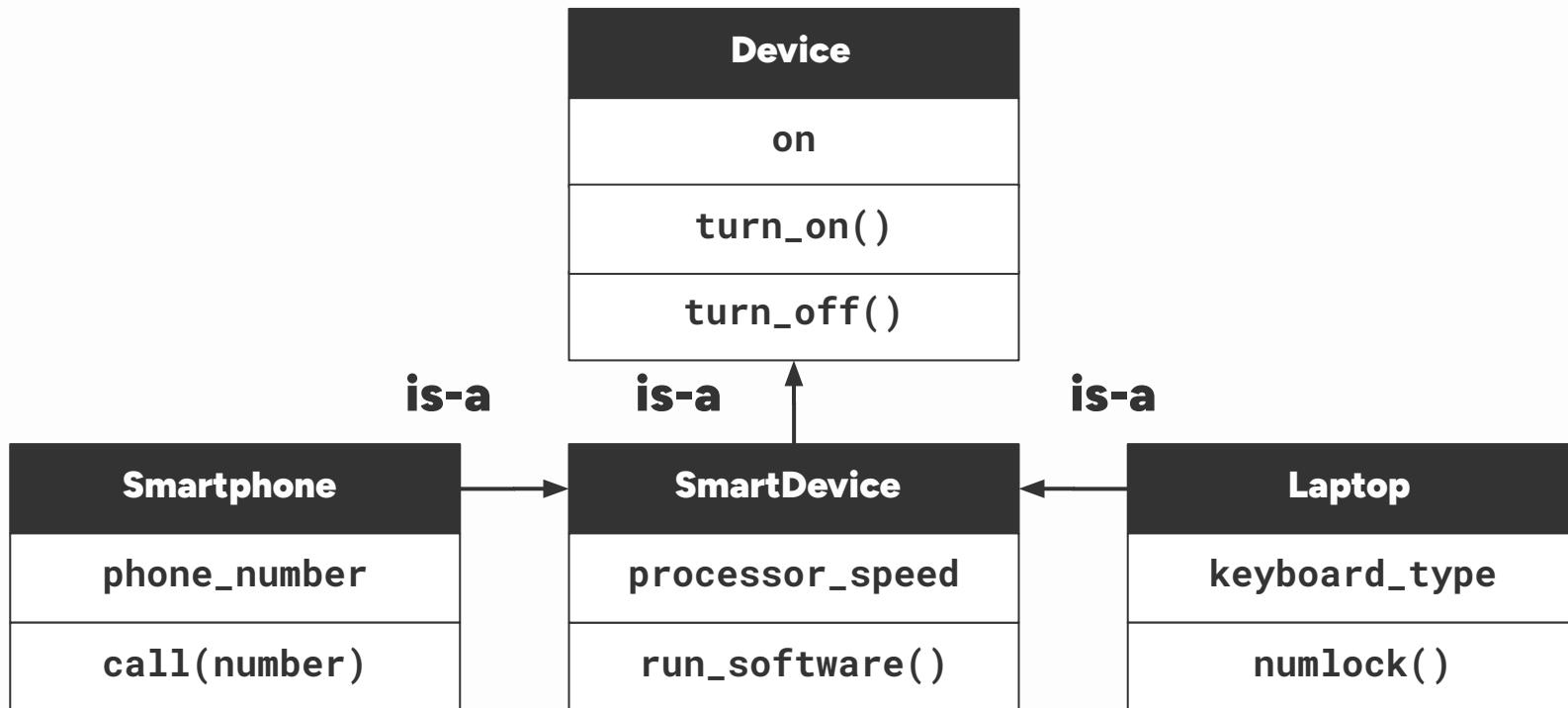
```
class Developer:  
    def __init__(self, name, id)  
    def add_work(self)  
    def code(self)
```

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

# Hierarchy Example



# Hierarchy Example 2

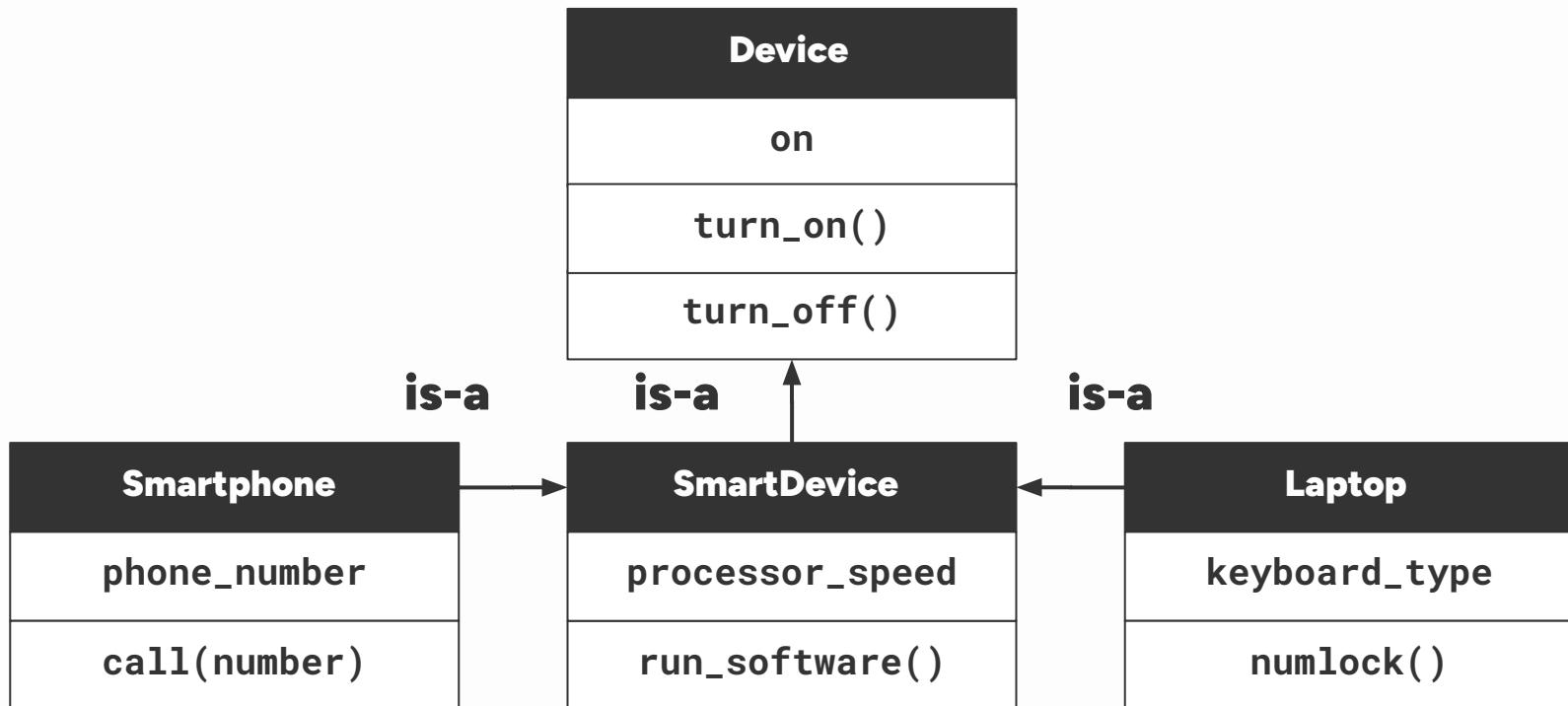




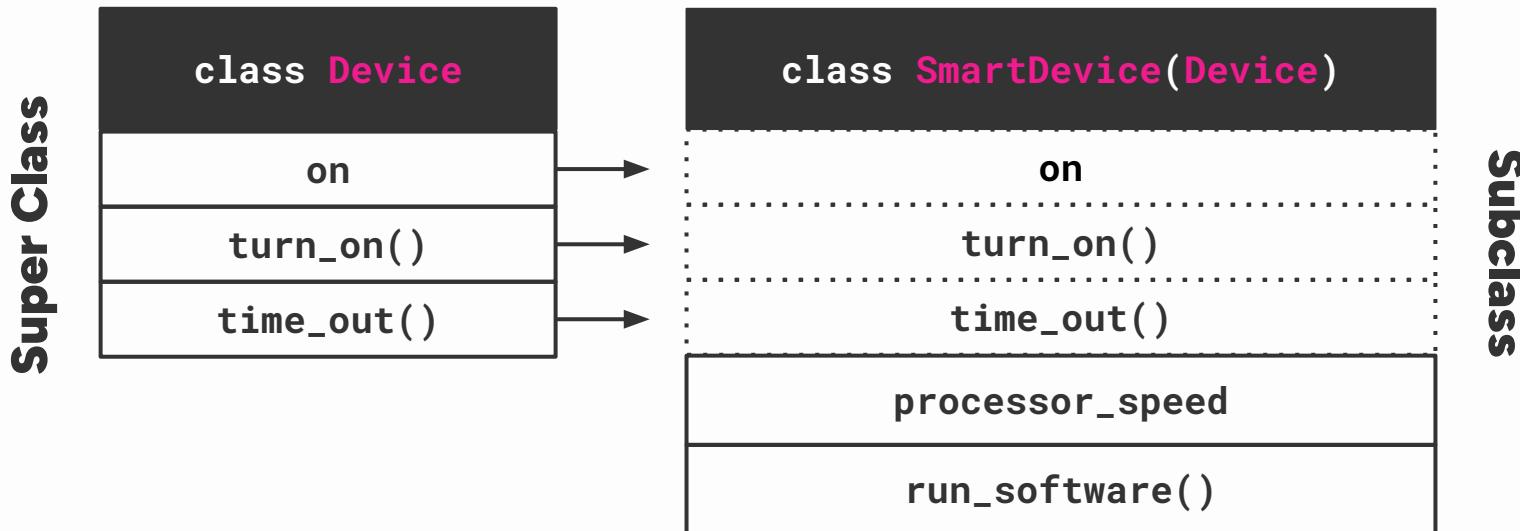




# Hierarchy



# Class Inheritance



# Implement: Student Class

student.py

```
class Person:
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    def sleep(self):
        print("I will sleep for eight hours")
    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() + f" I'm a {self.level} student."
```

# Student Class

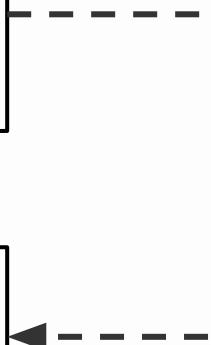
student.py

```
class Person:  
    def __init__(self, first_name, last_name):  
        self.first_name = first_name  
        self.last_name = last_name  
    def sleep(self):  
        print("I will sleep for eight hours")  
    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() + f" I'm a {self.level} student."
```

# Silent Inheritance

```
class Person:  
    def sleep(self):  
        print("I will sleep for eight hours")
```

```
class Student(Parent):  
    def sleep(self):  
        print("I will sleep for eight hours")
```



# Overriding

```
class Person:  
    def sleep(self):  
        print("I will sleep for eight hours")
```

```
class Student(Parent):  
    def sleep(self):  
        print("I will sleep for six hours")
```

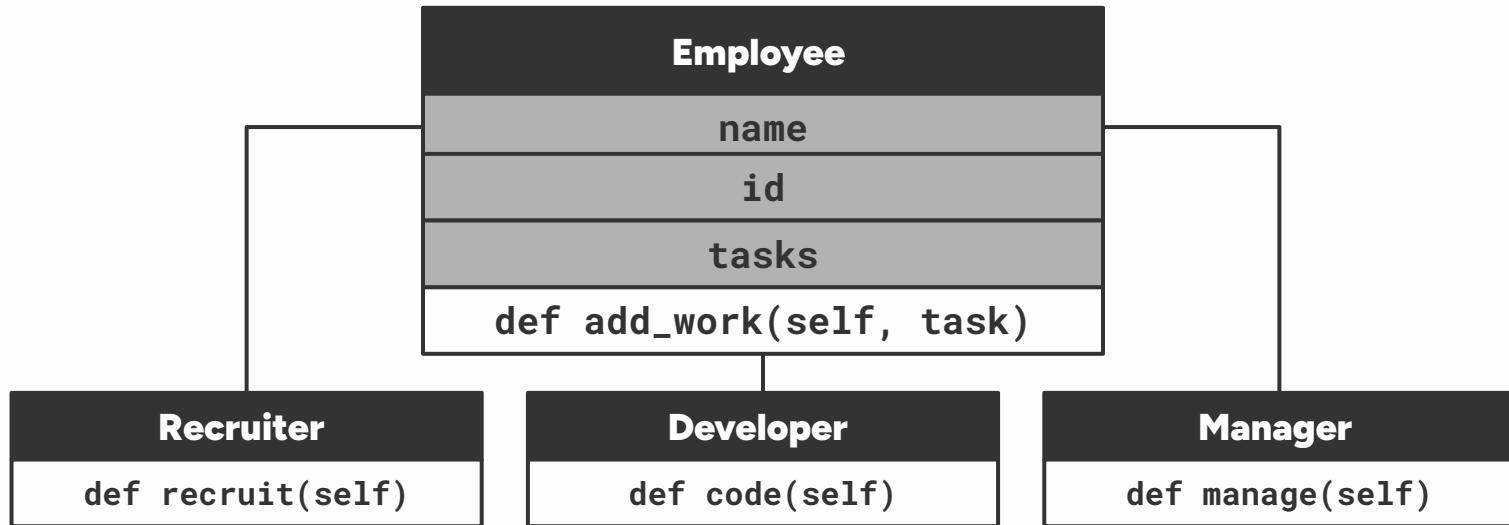


# Example: Writer

user.py

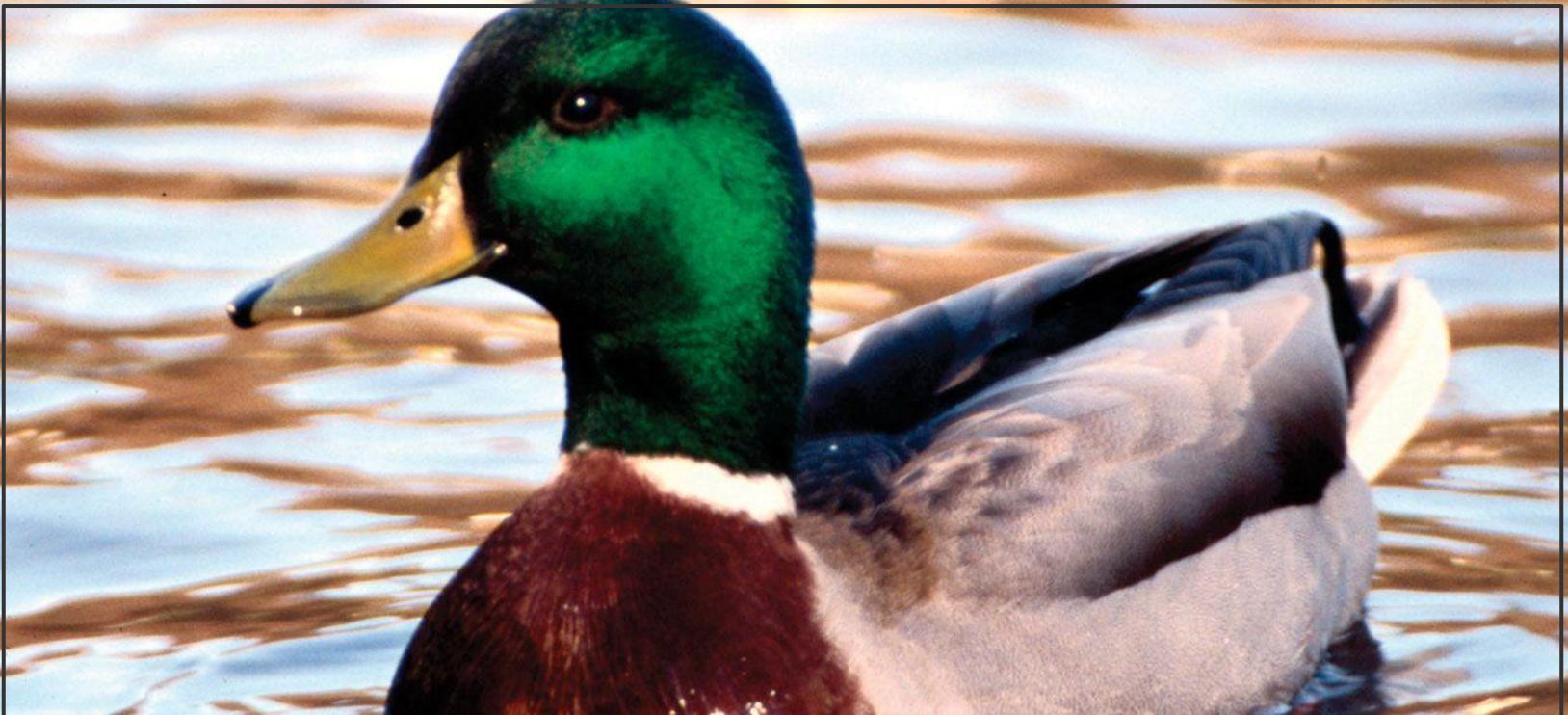
```
class User:  
    def __init__(self, username, email):  
        self.username = username  
        self.email = email  
  
    def display_info(self):  
        return f"User: {self.username} Email: {self.email}"  
  
class Writer(User):  
    def __init__(self, username, email, articles):  
        super().__init__(username, email)  
        self.articles = articles  
  
    def write_article(self, title):  
        print(f"{self.username} is writing '{title}'...")  
        self.articles += 1
```

# Implement: Employees



# 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

```
1  class Character:  
...  
17  
18 class Knight:  
19     def __init__(self, health=10, defense=10):  
20         self.health = health  
21         self.defense = defense  
22     def attack(self, other):  
23         damage = self.defense - other.defense  
24         other.health -= damage  
25  
26 player = Knight(defense=30)  
27 enemy = Character()  
28 player.attack(enemy)  
29 print(enemy.health)
```

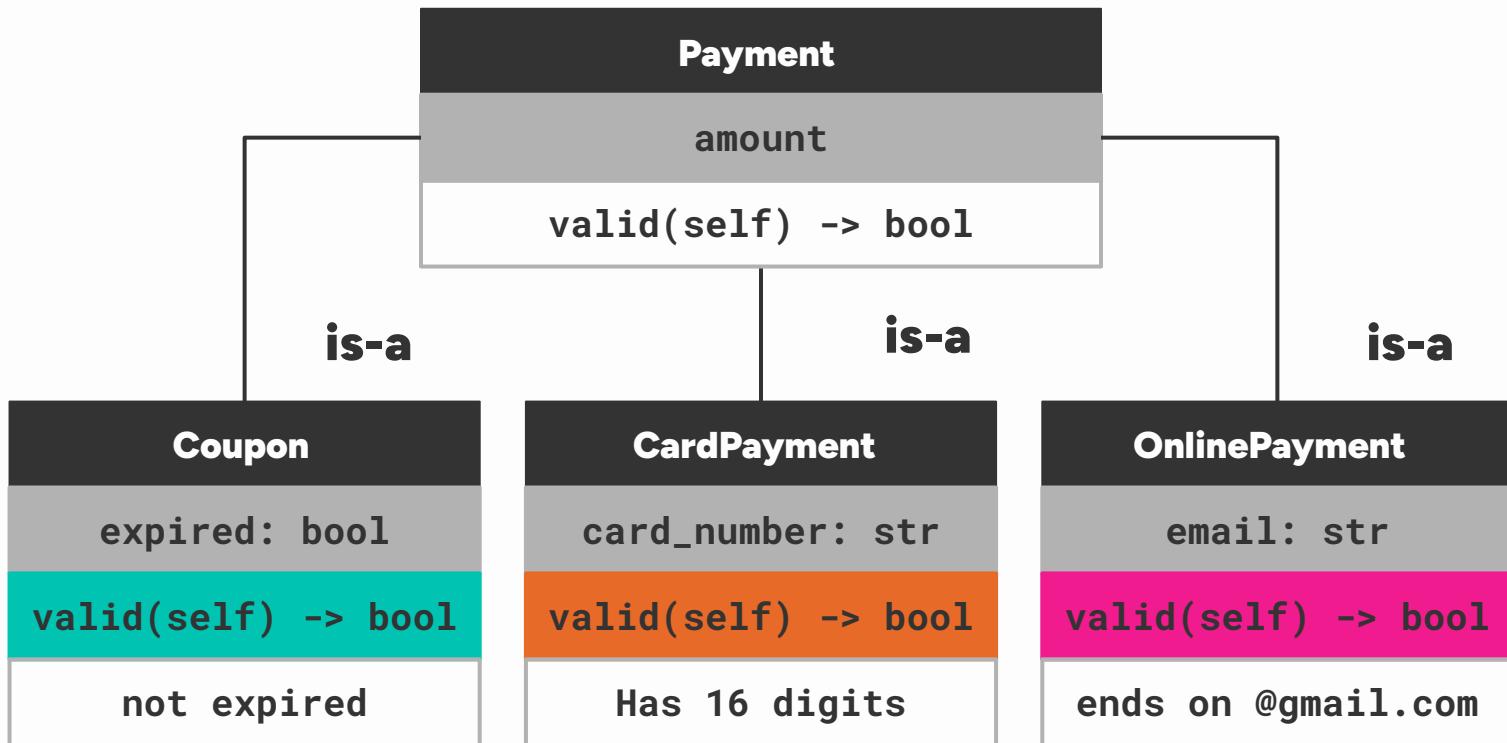
H3

# Payment Time

Yes, it's time to pay



## payment.py



03

# Structure

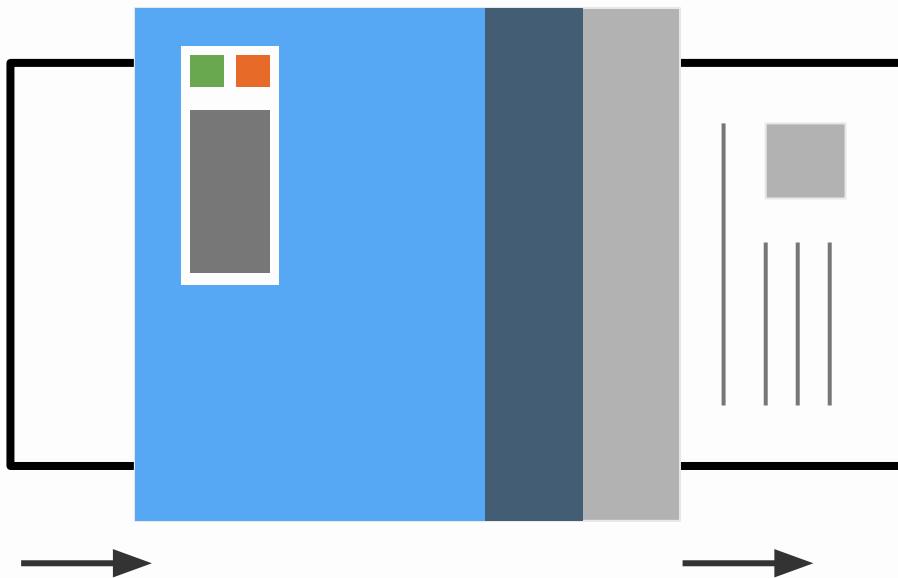
Appropriate Data Representation

# **Encapsulation**

Manage which parts are accessible to the public

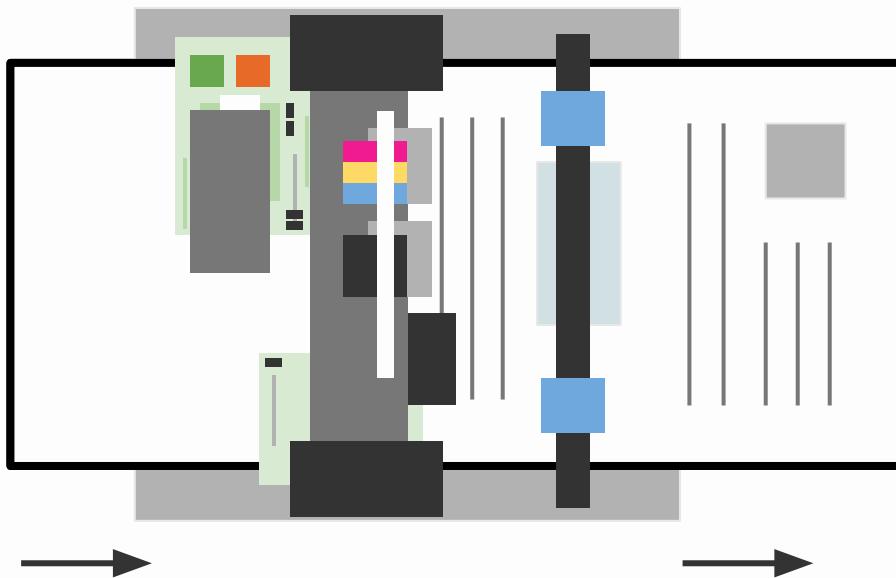
# Strategic Data Hiding

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

```
1 class Counter:  
2     def __init__(self):  
3         self.value = 0
```

Example Class

`self.value`

Other Class



# Protected Attribute

```
1 class Wallet:  
2     def __init__(self, initial_amount=0):  
3         self._amount = initial_amount
```

Wallet

self.\_amount

OnlineWallet

self.\_amount

# Wallet

```
1 class Wallet:  
2     def __init__(self, initial_amount=0):  
3         self._amount = initial_amount  
4  
5 food_wallet = Wallet(250)  
6 food_wallet.amount += 1_000  
7  
8 print("Food Budget:", food_wallet.amount)
```

# Update: Wallet

wallet.py

```
1 class Wallet:
2     def __init__(self, initial_amount=0):
3         self._amount = initial_amount
4
5     def get_amount(self):
6         print(f"Showing amount: {self._amount}")
7         return self._amount
8
9     def set_amount(self, amount):
10        print(f"Setting amount to {amount}")
11        self._amount += amount
12
13 food_wallet = Wallet(250)
14 food_wallet.set_amount(food_wallet.get_amount() + 1_000)
15
16 print("Food Budget:", food_wallet.get_amount())
```

# Update: Wallet

wallet.py

```
1 class Wallet:
2     def __init__(self, initial_amount=0):
3         self._amount = initial_amount
4
5     @property
6     def amount(self):
7         print(f"Showing amount: {self._amount}")
8         return self._amount
9
10    @amount.setter
11    def amount(self, amount):
12        print(f"Setting amount to {amount}")
13        self._amount += amount
14
15 food_wallet = Wallet(250)
16 food_wallet.amount += 1_000
17
18 print("Food Budget:", food_wallet.amount)
```

# Private Attribute

```
1 class Account:  
2     def __init__(self, initial_balance=0):  
3         self.__balance = initial_balance
```

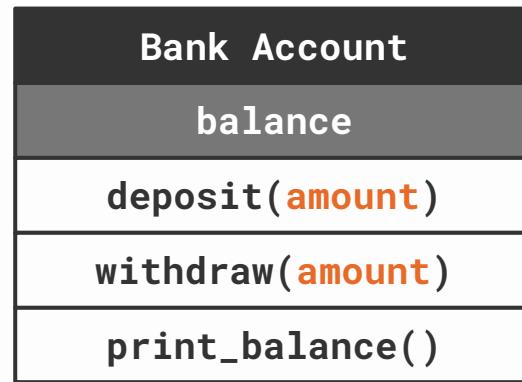
Account

self.\_\_balance

PremiumAccount



# Refactor: Prevent unnecessary changes



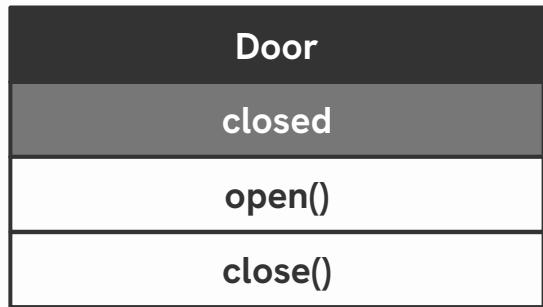
bank\_account.py

H4



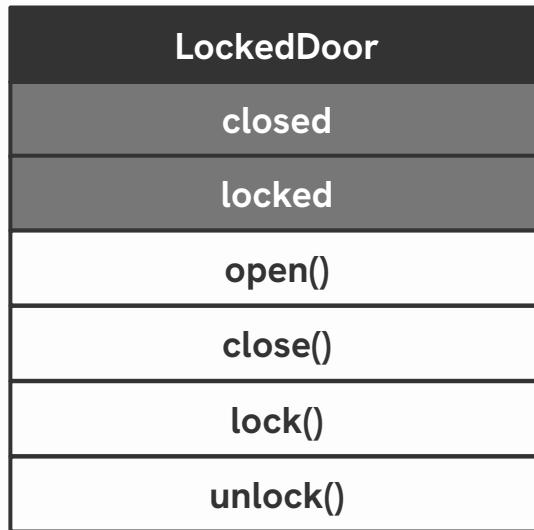
**Close the Door**

# Implement: Door



door.py

# Challenge: Locked Door



`locked_door.py`

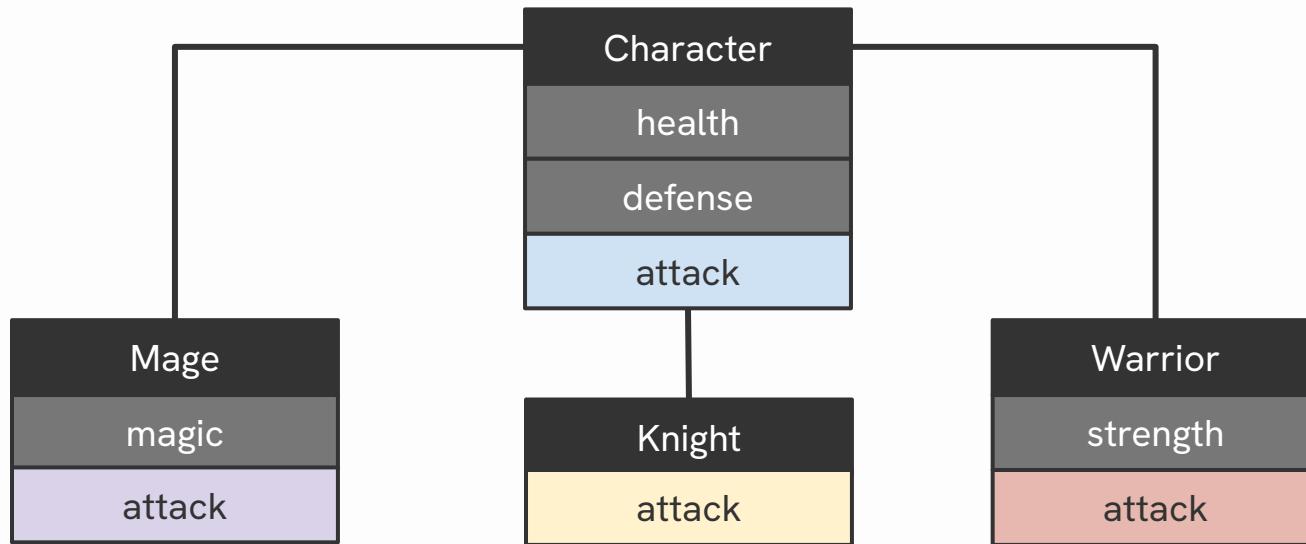
# **Abstraction**

Contractual Implementation

# Recall: Game Character



# Character Scheme



# Initial Implementation

rpg.py

```
1 class Character:
2     def __init__(self, health=10, defense=10):
3         self._health = health
4         self._defense = defense
5     def attack(self, other):
6         raise NotImplementedError()
7
8 class Knight(Character):
9     pass
10
11 enemy = Character()
12 knight = Knight()
13 knight.attack(enemy)
```

# Formal Polymorphism

rpg.py

```
1 from abc import ABC, abstractmethod
2
3 class Character(ABC):
4     def __init__(self, health=10, defense=10):
5         self._health = health
6         self._defense = defense
7     @abstractmethod
8     def attack(self, other):
9         raise NotImplementedError()
10
11 class Knight(Character):
12     def attack(self, other):
13         damage = self._defense - other._defense
14         other._health -= damage
```

Warrior

H6



Thief



Monk



Red Mage



White Mage

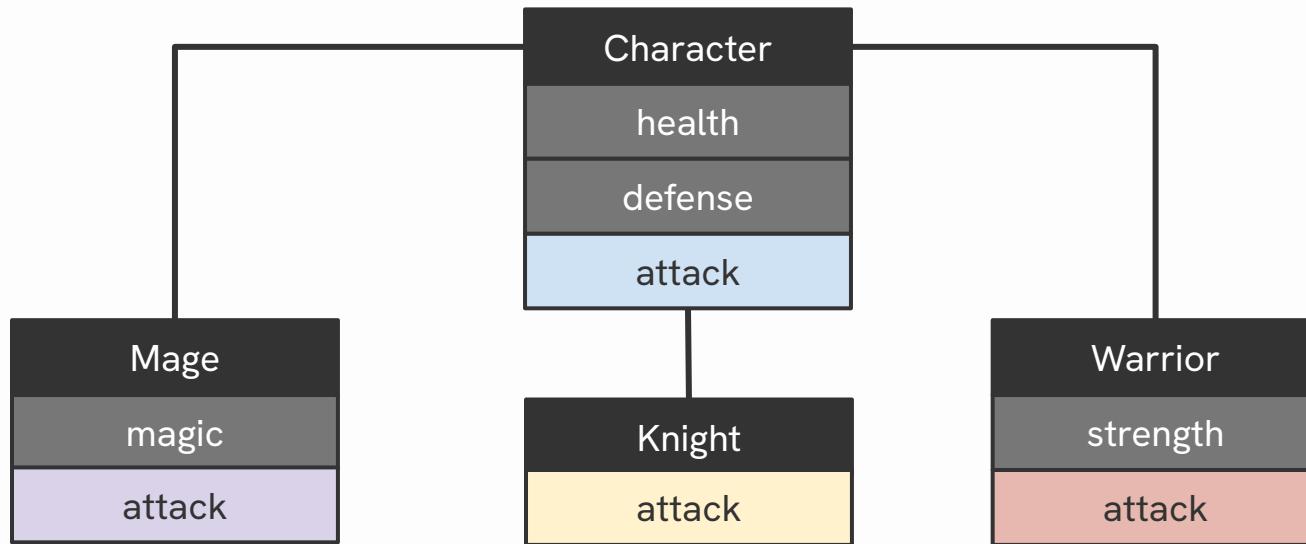


Black Mage



Class Tree

# Character Scheme



rpg.py

# **Custom Exception**

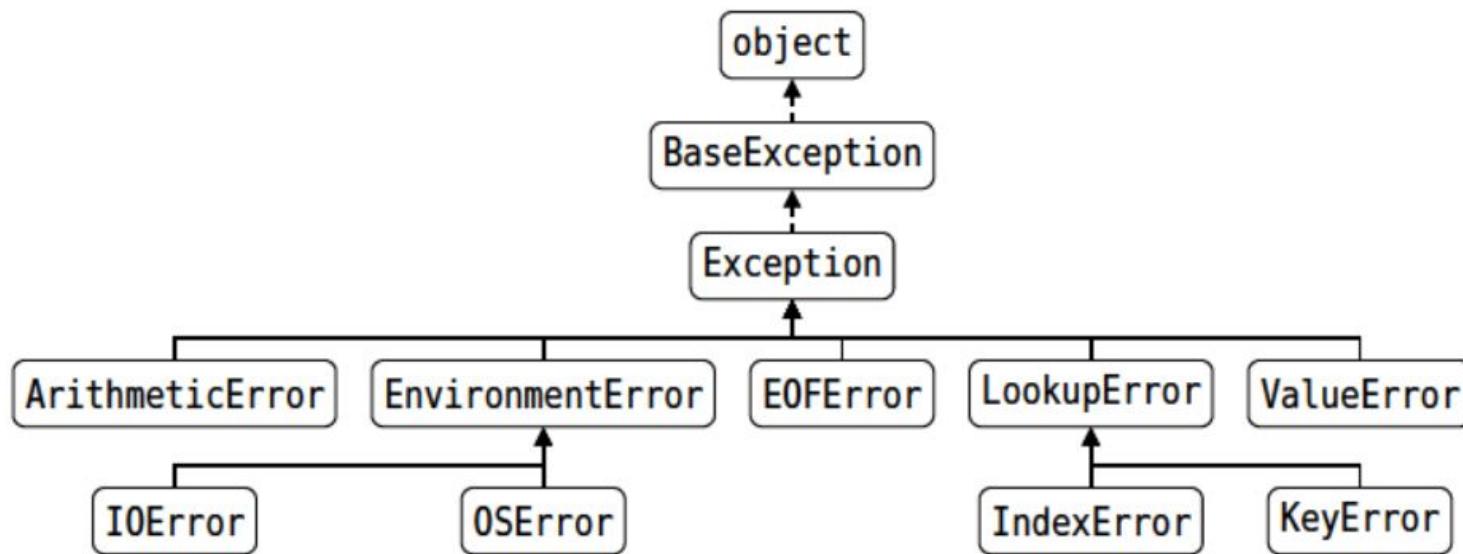
Create your own errors

# Custom Error

custom\_error.py

```
1 class CustomError(Exception):
2     pass
3
4 raise CustomError("yikes")
```

# Exception Hierarchy Excerpt



# Custom Error (Specific)

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

```
1 class InvalidChoiceError(ValueError):
2     pass
3
4 options = ("rock", "paper", "scissors")
5 user_choice = input("Pick move (rock/paper/scissors): ")
6
7 if user_choice not in options:
8     raise InvalidChoiceError()
```

# Roughly Equivalent Error

custom\_error.py

```
1 class CustomError(Exception):
2     def __init__(self, message):
3         super().__init__(message)
4
5 raise CustomError("yikes")
```

custom\_error\_complex.py

```
1 class CustomError(Exception):
2     def __init__(self, message, id):
3         message = f"Custom Error [{id}]: {message}"
4         super().__init__(message)
5
6 raise CustomError("yikes", 'a')
```

# Quick Exercise: Number Error

number\_error.py

```
1 number = input("Enter positive number [1,100]: ")
2
3 # If input not a number, raise a custom error
4 # If input is not positive, raise a custom error
5 # If input is not between 1 and 100, raise a custom error
```

A close-up photograph of a woman with blonde hair and a tattooed arm, wearing a green shirt, holding a large LEGO set. The set features Mickey Mouse dressed as a sorcerer with a blue pointed hat and a red robe, standing on a circular base. The base is filled with various LEGO minifigures, including a wizard, a woman in a red dress, a small yellow lion cub, and a person in a green outfit. There are also small buildings, trees, and a broomstick. The background is blurred, showing shelves with more LEGO sets.

# 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
<code>__init__</code>	*	None	Sets behavior when creating objects
<code>__str__</code>	None	String	Used in <code>str()</code> and <code>print()</code>
<code>__eq__</code>	Any	Boolean	Sets behavior for <code>==</code> operations
<code>__add__</code>	Any	Any	Sets behavior for <code>+</code> operations
<code>__len__</code>	None	Integer	Sets behavior when used in <code>len()</code>

# Implement: Book

```
1 class Book:  
2     def __init__(self, title=None, genre=None, author=None):  
3         self.title = title  
4         self.genre = genre  
5         self.author = author  
6  
7 book1 = Book("The Hobbit", "Fantasy", "Tolkien")  
8 book2 = Book("Dune", "Sci-Fi", "Herbert")  
9 print(book1)
```

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

# Magic Method `__repr__`

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

```
1 class Book:  
2     def __init__(self, title=None, genre=None, author=None):  
3         self.title = title  
4         self.genre = genre  
5         self.author = author  
6  
7     def __repr__(self):  
8         return f"{self.title} [{self.genre}] - {self.author}"  
9  
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

```
1 class Wallet:  
2     def __init__(self, initial_amount=0):  
3         self.amount = initial_amount  
4  
5     def __add__(self, other):  
6         new_amount = self.amount + other.amount  
7         return Wallet(new_amount)  
8  
9 food_wallet = Wallet(250)  
10 transport_wallet = Wallet(1000)  
11 total_wallet = food_wallet + transport_wallet  
12  
13 print("Food Budget: ", food_wallet.amount)  
14 print("Transport Budget: ", transport_wallet.amount)  
15 print("Total Budget: ", total_wallet.amount)
```

# Object Identity

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

```
1 class Candy:  
2     def __init__(self, flavor):  
3         self.flavor = flavor  
4  
5     choco1 = Candy("chocolate")  
6     choco2 = Candy("chocolate")  
7     milk = Candy("milk")  
8  
9     print(choco1 == milk)  
10    print(choco1 == choco2)
```

# Magic Method `__eq__`

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

```
1 class Candy:  
2     def __init__(self, flavor):  
3         self.flavor = flavor  
4  
5     def __eq__(self, other):  
6         return self.flavor == other.flavor  
7  
8 choco1 = Candy("chocolate")  
9 choco2 = Candy("chocolate")  
10 milk = Candy("milk")  
11  
12 print(choco1 == milk)  
13 print(choco1 == choco2)
```

05

# GUI

Graphical User Interface

# Benefits of Graphical User Interfaces



## User Experience

Easier to understand and provides appeal and interactivity



## Separation

Clearly Separate Frontend (Design, UI/UX) and Backend (Logic)



## Limitations

Limit the possible edge cases and directly get needed data type

# 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

01\_hello\_world.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 root.mainloop()
```

# Window (with Title)

01\_hello\_world.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 root.title("Sample GUI Application")  
5  
6 root.mainloop()
```

# Window (with Size)

01\_hello\_world.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 root.title("Sample GUI Application")  
5 root.geometry("1200x400")  
6  
7 root.mainloop()
```

# **Label**

Adding text to the window

# Label

01\_hello\_world.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 root.title("Sample GUI Application")  
5 root.geometry("1200x400")  
6  
7 label = tkinter.Label(root, text="Hello")  
8 label.pack()  
9  
10 root.mainloop()
```

# Multiple Labels

01\_hello\_world.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 root.title("Sample GUI Application")  
5 root.geometry("1200x400")  
6  
7 label = tkinter.Label(root, text="Hello")  
8 label.pack()  
9  
10 next_label = tkinter.Label(root, text="World")  
11 next_label.pack()  
12  
13 root.mainloop()
```

# Multiline Label

02\_multiline.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 message = """  
6 Hello  
7 World  
8 """  
9  
10 label = tkinter.Label(root, text=message)  
11 label.pack()  
12  
13 root.mainloop()
```

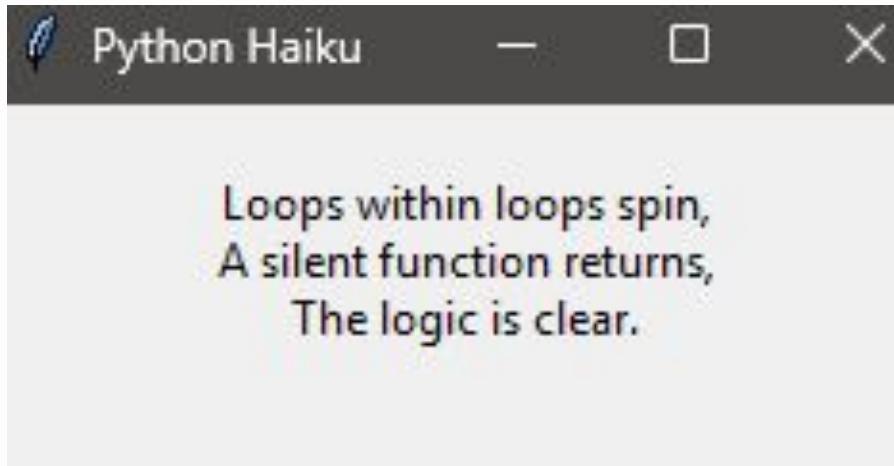
# Message

03\_message.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 text = "Sentence 1 Sentence 2 Sentence 3"  
6 message = tkinter.Message(root, text=text)  
7 message.pack()  
8  
9 root.mainloop()
```

# Quick Exercise: Haiku

Recreate the following window using label(s)



04\_haiku.py

# Properties

Adding styling and layout to components

# Component Font Style

05\_font\_style.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 label = tkinter.Label(  
6     root,  
7     text="Hello",  
8     font=("Arial", 14, "bold italic"))  
9  
10 label.pack()  
11  
12 root.mainloop()
```

# Find Other Fonts Available

06\_font\_families.py

```
1 import tkinter  
2 from tkinter import font  
3  
4 root = tkinter.Tk()  
5  
6 all_fonts = font.families()  
7 print(all_fonts)
```

# Component Color

07\_colors.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 label = tkinter.Label(  
6     root,  
7     text="Hello",  
8     fg="red",  
9     bg="yellow"  
10 )  
11 label.pack()  
12  
13 root.mainloop()
```

# Component Size

08\_size.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 label = tkinter.Label(  
6     root,  
7     text="Hello",  
8     width=100,  
9     height=20  
10 )  
11 label.pack()  
12  
13 root.mainloop()
```

# Component Pad

09\_pads.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 label = tkinter.Label(  
6     root,  
7     text="Hello",  
8     padx=10,  
9     pady=200  
10 )  
11 label.pack()  
12  
13 root.mainloop()
```

# Component Pack Side

10\_sides.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 label1 = tkinter.Label(root, text="Left")  
6 label1.pack(side="left")  
7  
8 label2 = tkinter.Label(root, text="Right")  
9 label2.pack(side="right")  
10  
11 root.mainloop()
```

# Quick Exercise: Mood Board

Recreate the following window using properties and label(s)



11\_mood\_board.py

# **Entry**

Asking the user for text input

# Blank Entry

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
6 entry.pack()  
7  
8 root.mainloop()
```

# Entry Bind

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
6 entry.pack()  
7  
8 def show_input(event):  
9     print("Enter pressed")  
10  
11 root.bind("<Return>", show_input)  
12 root.mainloop()
```

# Entry Echo

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
entry.pack()  
6  
7  
8 def show_input(event):  
9     given_text= entry.get()  
10    print(given_text)  
11  
12 root.bind("<Return>", show_input)  
13 root.mainloop()
```

# Entry Echo

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
entry.pack()  
6  
7  
8 def show_input(event):  
9     given_text= entry.get()  
10    print(given_text)  
11  
12 root.bind("<Return>", show_input)  
root.bind("<space>", show_input)  
13  
14 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

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
entry.pack()  
6  
7  
8 def show_input(event):  
9     given_text = entry.get()  
10    label = tkinter.Label(root, text=given_text)  
label.pack()  
11  
12  
13 root.bind("<Return>", show_input)  
root.bind("<space>", show_input)  
14  
15 root.mainloop()
```

# **String Variable**

Dynamic text for components

# String Variable

13\_string\_var.py

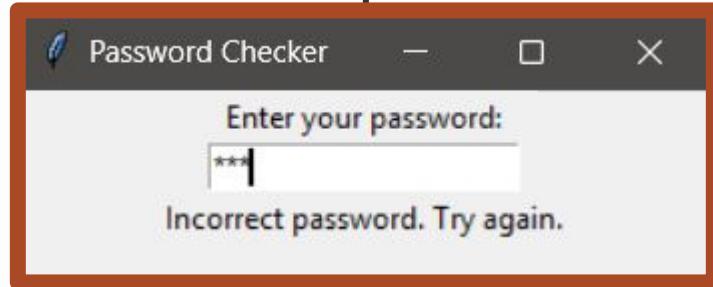
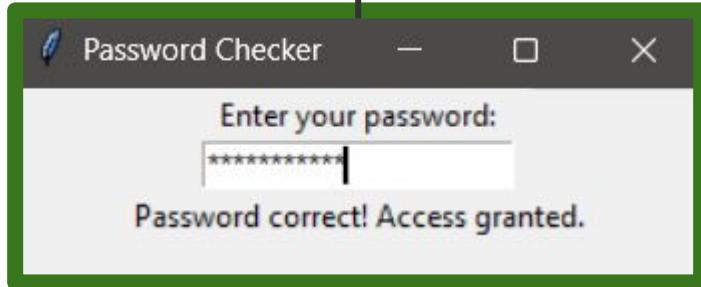
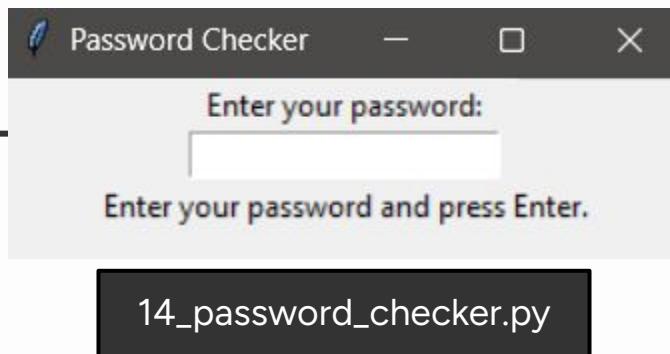
```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 text = tkinter.StringVar(root, value="Hello")  
6 label = tkinter.Label(root, textvariable=text)  
7 label.pack()  
8  
9 root.mainloop()
```

# Dynamic Label

12\_entry\_bind.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 entry = tkinter.Entry(root)  
entry.pack()  
6  
7  
8 user_input = tkinter.StringVar(root, value="Enter any text")  
9 label = tkinter.Label(root, textvariable=user_input)  
label.pack()  
10  
11  
12 def show_input(event):  
13     given_text = entry.get()  
14     user_input.set(given_text)  
15 ...
```

# Quick Exercise: Password Checker



# **Buttons**

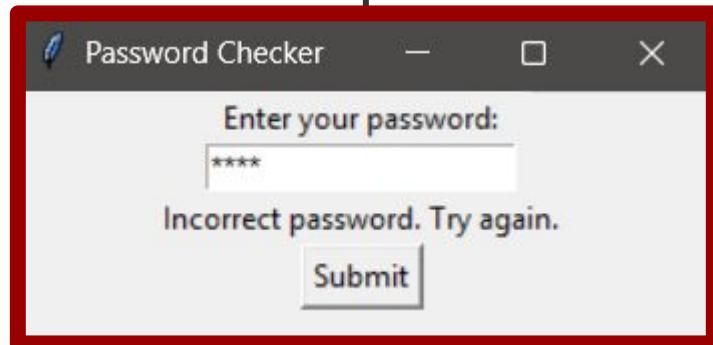
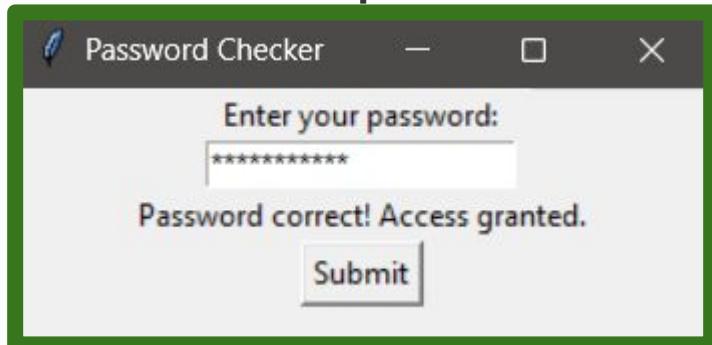
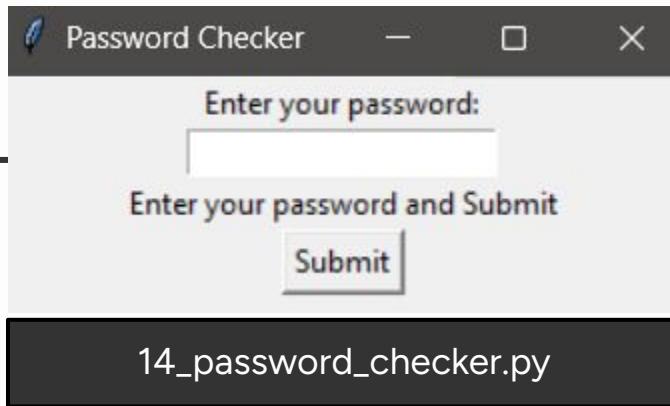
Trigger functions on command

# Dynamic Label (Submit)

15\_entry\_button.py

```
...
12 def show_input():
13     given_text = entry.get()
14     user_input.set(given_text)
15
16 button = tkinter.Button(root, text="Submit", command=show_input)
17 button.pack()
18 root.mainloop()
```

# Quick Exercise: Password Checker



# Counter

16\_counter.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 count = tkinter.IntVar(root, value=0)  
5 label = tkinter.Label(root, textvariable=count)  
6 label.pack()  
7  
8 def increment():  
9     new_value = count.get() + 1  
10    count.set(new_value)  
11  
12 button = tkinter.Button(root, text=" + ", command=increment)  
13 button.pack()  
14  
15 root.mainloop()
```

# Quick Exercise: Full Counter



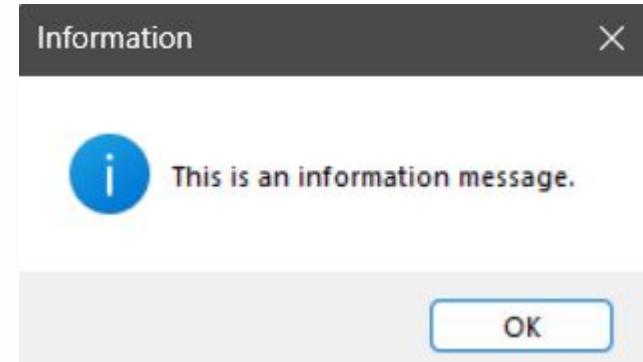
17\_full\_counter.py

# **Message Boxes**

Sudden message displays for the user

# Information Box

```
1 import tkinter  
2 from tkinter import messagebox  
3  
4 root = tkinter.Tk()  
5  
6 messagebox.showinfo(  
7     "Information",  
8     "This is an information message."  
9 )  
10  
11 root.mainloop()  
12  
13  
14  
15
```



18\_information\_box.py

# Warning Box

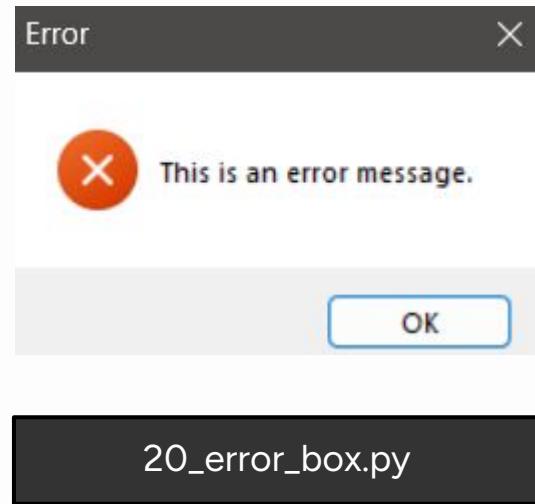
```
1 import tkinter  
2 from tkinter import messagebox  
3  
4 root = tkinter.Tk()  
5  
6 messagebox.showwarning(  
7     "Warning",  
8     "This is a warning message."  
9 )  
10  
11 root.mainloop()  
12  
13  
14  
15
```



19\_warning\_box.py

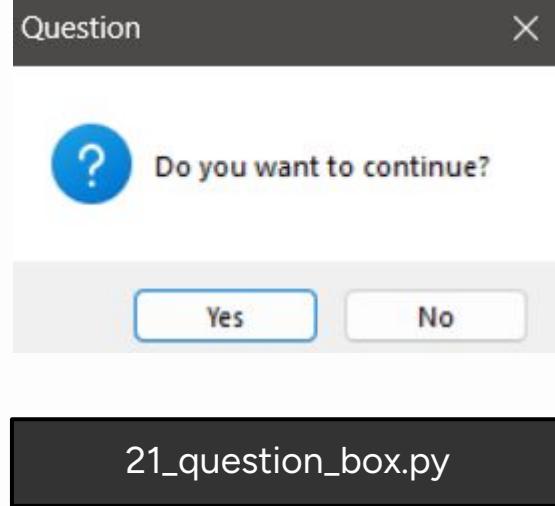
# Error Message Box

```
1 import tkinter  
2 from tkinter import messagebox  
3  
4 root = tkinter.Tk()  
5  
6 messagebox.showerror(  
7     "Error",  
8     "This is an error message."  
9 )  
10  
11 root.mainloop()  
12  
13  
14  
15
```



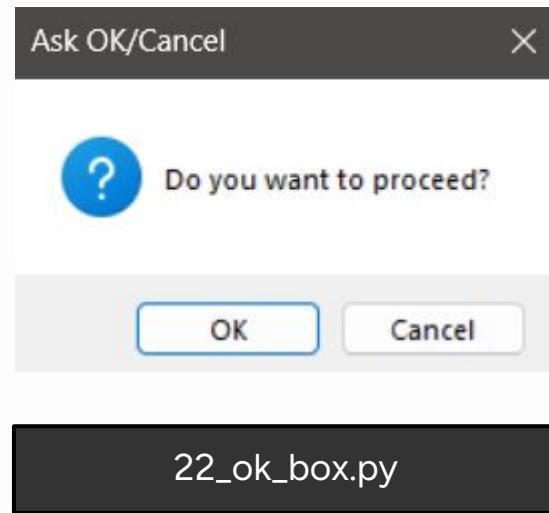
# Question Message Box

```
1 import tkinter  
2 from tkinter import messagebox  
3  
4 root = tkinter.Tk()  
5  
6 # yes or no  
7 response = messagebox.askquestion(  
8     "Question",  
9     "Do you want to continue?  
10 )  
11  
12 root.mainloop()  
13  
14  
15
```

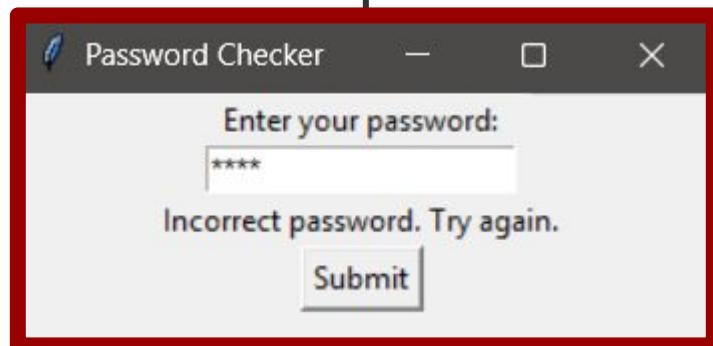
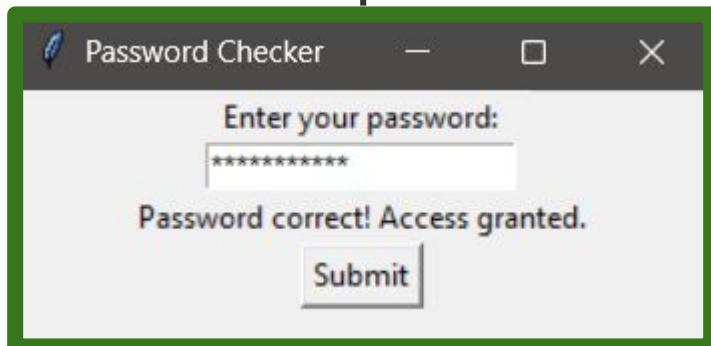


# Ask OK Message Box

```
1 import tkinter  
2 from tkinter import messagebox  
3  
4 root = tkinter.Tk()  
5  
6 # true or false  
7 response = messagebox.askokcancel(  
8     "Ask OK/Cancel",  
9     "Do you want to proceed?"  
10 )  
11  
12 root.mainloop()  
13  
14  
15
```



# Quick Exercise: Password Checker



# **Input Components**

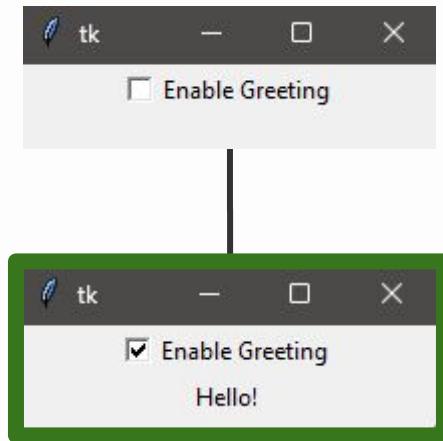
Other basic components for getting user data

# Checkbox

23\_checkbox.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 check_value = tkinter.BooleanVar()  
6 checkbox = tkinter.Checkbutton(  
7     root,  
8     text="Enable",  
9     variable=check_value  
10 )  
11 checkbox.pack()  
12  
13 root.mainloop()  
14  
15
```

# Quick Exercise: First Greeting



24\_first\_greeting.py

# Radio Buttons

25\_radio.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 radio_var = tkinter.StringVar(value="Option A")  
6 radio1 = tkinter.Radiobutton(  
7     root, text="Option A", variable=radio_var, value="Option A")  
8 radio1.pack()  
9  
10 radio2 = tkinter.Radiobutton(  
11     root, text="Option B", variable=radio_var, value="Option B")  
12 radio2.pack()  
13  
14 root.mainloop()
```

# Quick Exercise: Store Select



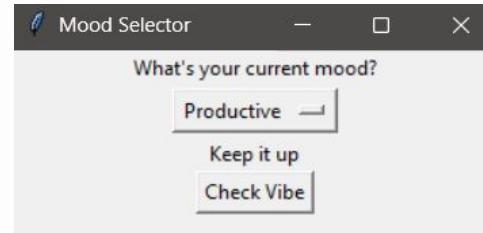
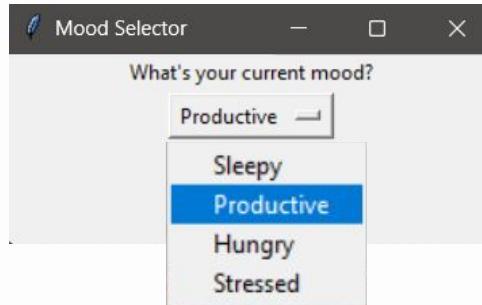
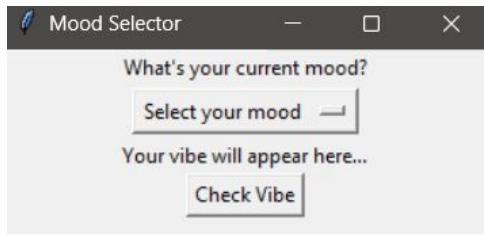
26\_store select\_.py

# Dropdown

27\_dropdown.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 dropdown_var = tkinter.StringVar(value="Choice 1")  
6 dropdown_menu = tkinter.OptionMenu(  
7     root, dropdown_var,  
8     "Choice 1",  
9     "Choice 2",  
10    "Choice 3"  
11 )  
12 dropdown_menu.pack()  
13  
14 root.mainloop()
```

# Quick Exercise: Check Vibe



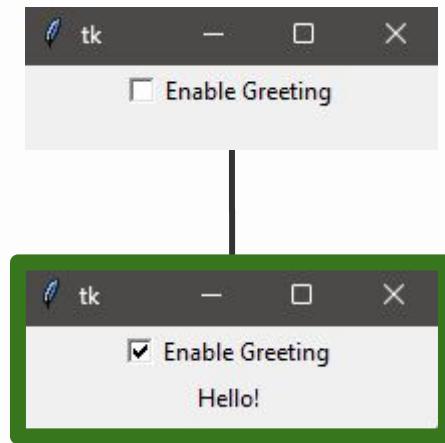
28\_check\_vibe.py

# Slider

29\_slider.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 slider_value = tkinter.IntVar(value=0)  
6 slider = tkinter.Scale(  
7     root,  
8     from_=0,  
9     to=100,  
10    orient="horizontal",  
11    variable=slider_value  
12 )  
13 slider.pack()  
14  
15 root.mainloop()
```

# Quick Exercise: Slider



30.py

# SimpleDialog

31\_simple\_dialog.py

```
1 import tkinter
2 from tkinter import simpledialog
3
4 root = tkinter.Tk()
5
6 def ask_all():
7     name = simpledialog.askstring("String", "Your name?")
8     age = simpledialog.askinteger("Integer", "Your age?")
9     score = simpledialog.askfloat("Float", "Your score?")
10    if name and age and score:
11        message = f"{name} | {age} | {score}"
12        tkinter.Label(root, text=message).pack()
13
14 tkinter.Button(root, text="Start", command=ask_all).pack()
15 root.mainloop()
```

# Layout

Setup the layouting for all of the components by group

# Frames

32\_frames.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 left_frame = tkinter.Frame(root, bg="lightblue")  
6 left_frame.pack(side="left")  
7  
8 left_label = tkinter.Label(left_frame, text="I'm on the left")  
9 left_label.pack()  
10  
11 right_frame = tkinter.Frame(root, bg="lightgreen")  
12 right_frame.pack(side="right")  
13  
14 right_entry = tkinter.Entry(right_frame)  
15 right_entry.pack()  
16 right_button = tkinter.Button(right_frame, text="Click me")  
17 right_button.pack()  
18  
19 root.mainloop()
```

# Grids

33\_grids.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4  
5 top = tkinter.Label(root, text="Top", bg="blue", width=40, height=2)  
6 top.grid(row=0, column=0, columnspan=3, sticky="nsew")  
7 side = tkinter.Label(root, text="Side", bg="green", width=15, height=4)  
8 side.grid(row=1, column=0, rowspan=2, sticky="nsew")  
9 cell_1_1 = tkinter.Label(root, text="1,1", bg="gray", width=15, height=2)  
10 cell_1_1.grid(row=1, column=1)  
11 cell_1_2 = tkinter.Label(root, text="1,2", bg="gray", width=15, height=2)  
12 cell_1_2.grid(row=1, column=2)  
13 cell_2_1 = tkinter.Label(root, text="2,1", bg="yellow", width=15, height=2)  
14 cell_2_1.grid(row=2, column=1)  
15 cell_2_2 = tkinter.Label(root, text="2,2", bg="yellow", width=15, height=2)  
16 cell_2_2.grid(row=2, column=2)  
17  
18 root.mainloop()
```

# Frame and Grids

34\_frames\_grid.py

```
1 import tkinter  
2  
3 root = tkinter.Tk()  
4 root.title("Login Form")  
5  
6 form_frame = tkinter.Frame(root, padx=20, pady=20)  
7 form_frame.pack()  
8  
9 tkinter.Label(form_frame, text="Username:").grid(row=0, column=0)  
10 username_entry = tkinter.Entry(form_frame)  
11 username_entry.grid(row=0, column=1)  
12  
13 tkinter.Label(form_frame, text="Password:").grid(row=1, column=0)  
14 password_entry = tkinter.Entry(form_frame, show="*")  
15 password_entry.grid(row=1, column=1)  
16  
17 login_button = tkinter.Button(form_frame, text="Login")  
18 login_button.grid(row=2, column=0, columnspan=2, pady=10)  
19 root.mainloop()
```

# Class Organization

35\_tkinter\_class.py

```
1 import tkinter  
2  
3 class Application(tkinter.Tk):  
4     def __init__(self):  
5         super().__init__()  
6         self.title("Tkinter Class Structure")  
7         self.geometry("300x200")  
8         self.create_widgets()  
9  
10    def create_widgets(self):  
11        label = tkinter.Button(self, text="Hello", command=self.hello)  
12        label.pack()  
13  
14    def hello(self):  
15        print("Hello")  
  
app = Application()  
app.mainloop()
```

06

# Lab Session

All the Major Features Covered

{

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

}

user.json

Name Peter

Age 32

Preferred Theme  Light  Dark

Subscribe to newsletter

Rate us

3

Submit

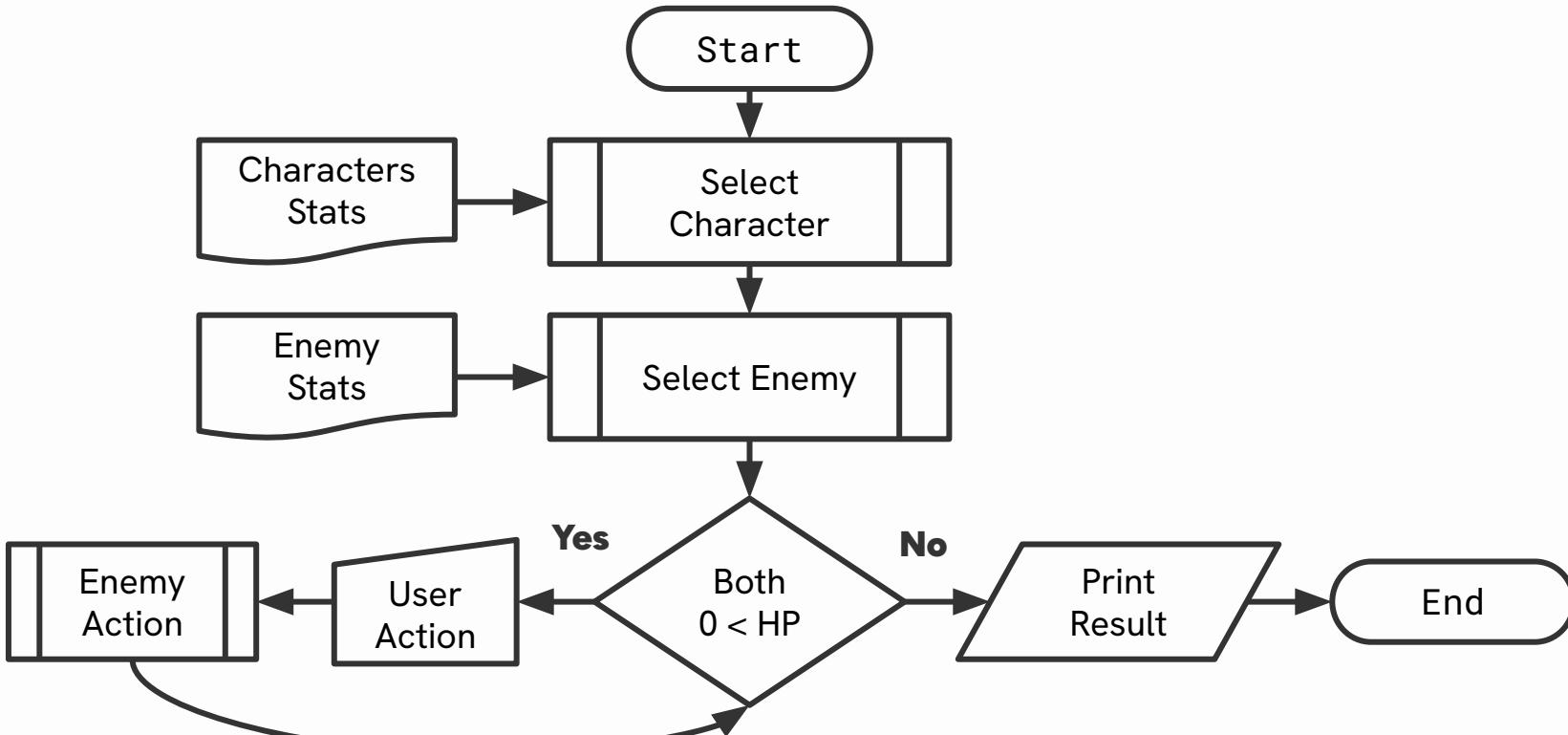
# Forms



# Battle!



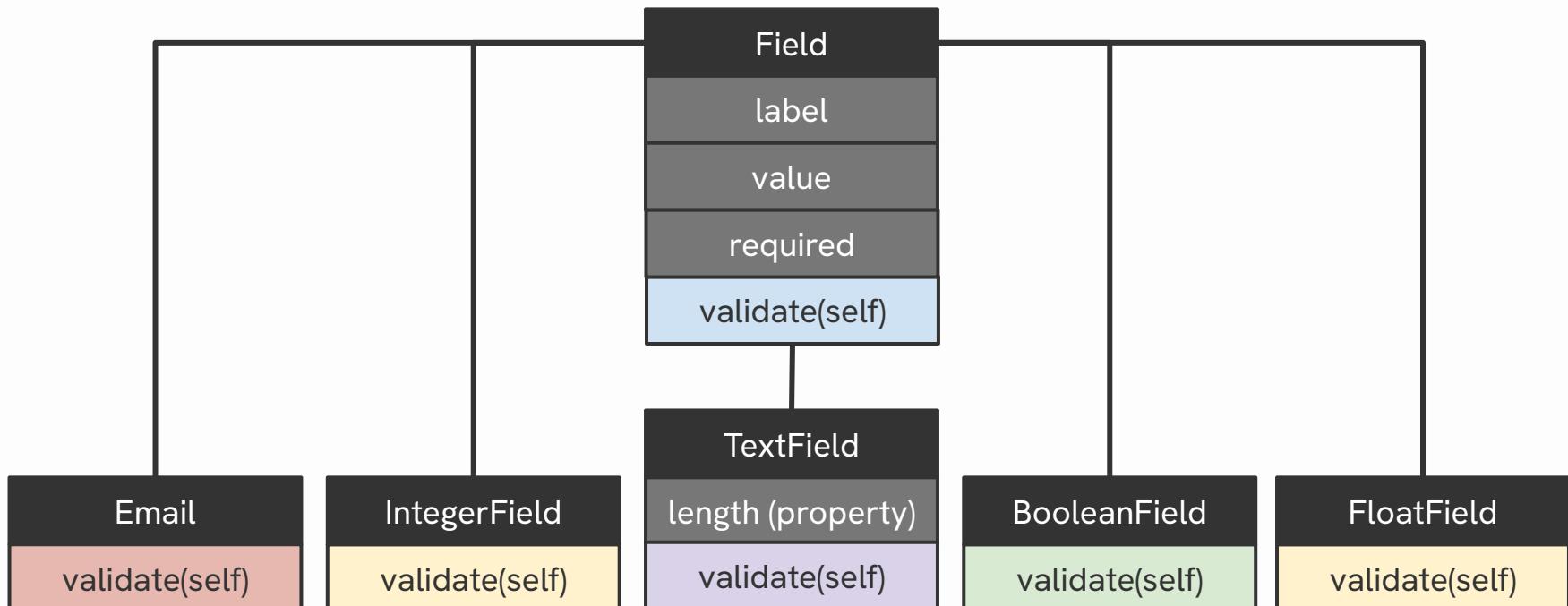
# Battle! Game Flow



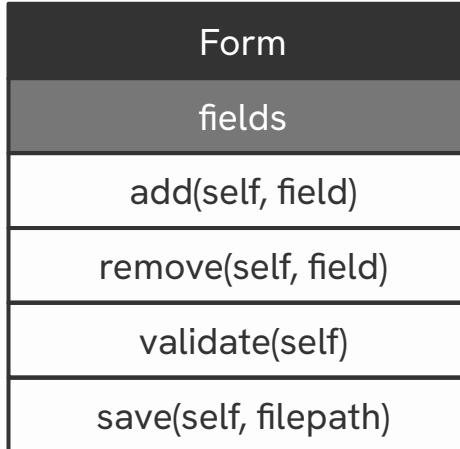
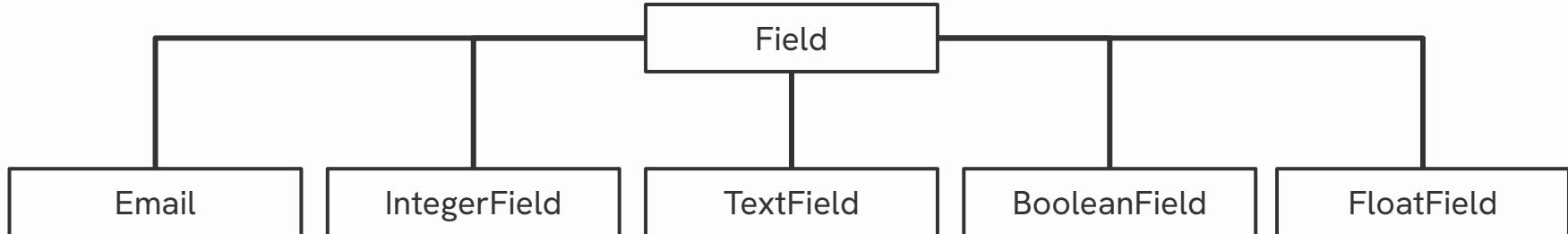
A wide-angle photograph of a vast green field, likely a grassy hillside, stretching towards a horizon under a bright blue sky dotted with wispy white clouds.

**Fields**

# Initial Fields



# Forms with Fields



# Inbox



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

Email
sender
subject
message
date
read_status
archive_status
__repr__( <b>self</b> )
read( <b>self</b> )
unread( <b>self</b> )
archive( <b>self</b> )
unarchive( <b>self</b> )

# Sneak Peak

01

## Packaging

Handling Python Files

02

## Concurrency

Multitasking techniques

03

## Best Practices

Writing Pythonic code

04

## Testing

Code correctness

05

## Web Dev

Introduction to Flask

06

## Lab Session

Culminating Exercise

# **Python: Day 03**

Object-Oriented Programming