CSC 148

Week 2

Sadia ‘Rain’ Sharmin

Classes begin 10 minutes after the hour

Week 2

Object-Oriented Programming

SUBTOPICS:

- Review Classes
- Composition
- Representation Invariants

Terminology

Key terms

class

instance (of a class)

instance attribute

method

initializer

dot notation

id, type, value

parameter

local variable

alias

Review Classes

Programs

The fundamental transition

The study of computer science transforms us from being users of technology to being creators of technology

Designing

As a computer scientist, you will have to design programs according to your client's specifications

Skills to learn: Design, implementation, writing (good!) documentation, (thorough!) testing

Classes

Classes and objects

What's a class?

Abstract data structure that models a real-world concept

Describes the attributes and "abilities" (methods) of that concept (called object)

Example: int, str, list, etc., or user-defined: Tweet, User, Cat, Desk, FileReader, ColourPrinter, etc.

What's an object?

Instance of a class

Everything in Python is an object!

Quizizz time!

Python Classes

https://quizizz.com/admin/quiz/61e5c7049bec76001efae954/python-classes

WORKSHEET

Object-Oriented Programming #1

Composition

Classes

Composition

Key concept: how do we design code in which different classes interact with each other?

Idea: Use existing types inside new user-defined types

Design roadmap

Classes

First step is to analyze specification:

The Twitter application allows users to broadcast short messages called tweets. A tweet includes the message content (of up to 280 characters), the user who wrote the tweet, when the tweet was created, and how many “likes” the tweet has. Once a tweet is created, it may be liked by other users. Furthermore, the tweet may be edited by its owner.

Design roadmap

Classes

Analyze specification:

The Twitter application allows users to broadcast short messages called tweets. A tweet includes the message content (of up to 280 characters), the user who wrote the tweet, when the tweet was created, and how many "likes" the tweet has. Once a tweet is created, it may be liked by other users. Furthermore, the tweet may be edited by its owner.

Design roadmap

Classes

Analyze specification:

The Twitter application allows users to broadcast short messages called tweets. A tweet includes the message content (of up to 280 characters), the user who wrote the tweet, when the tweet was created, and how many "likes" the tweet has. Once a tweet is created, it may be liked by other users. Furthermore, the tweet may be edited by its owner.

Design roadmap

Classes

Analyze specification:

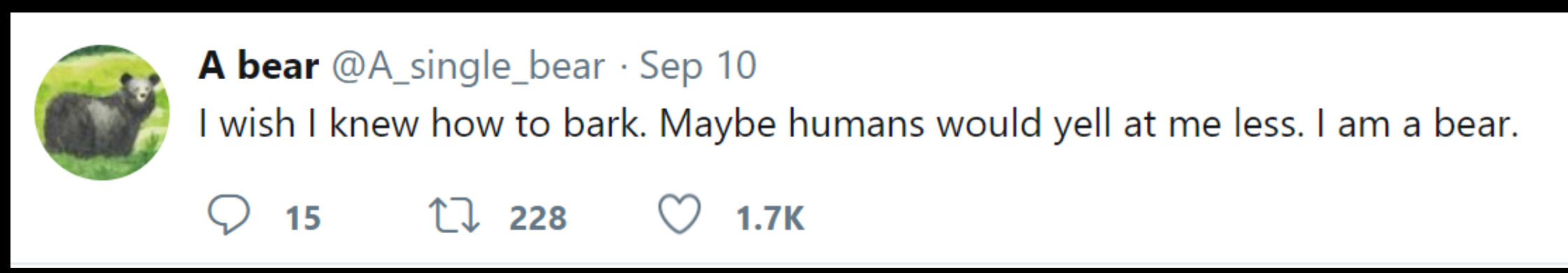
The Twitter application allows users to broadcast short messages called tweets. A tweet includes the message content (of up to 280 characters), the user who wrote the tweet, when the tweet was created, and how many "likes" the tweet has. Once a tweet is created, it may be liked by other users. Furthermore, the tweet may be edited by its owner.

Composition

A Tweet object

We made a Tweet object to represent a Tweet.

Each tweet has some content, user id (the user who created the tweet), the date it was created, and the number of likes it has.



Composition

Our Tweet class

```
class Tweet:
    """A tweet, like in Twitter.

    === Attributes ===
    content: the contents of the tweet.
    userid: the id of the user who wrote the tweet.
    created_at: the date the tweet was written.
    likes: the number of likes this tweet has received.
    """
    content: str
    userid: str
    created_at: date
    likes: int
```

Composition

A Twitter User object

The user who made the Tweet can be represented as an object too.

Say we want to implement class User, based on this spec:

A Twitter user has a unique identity, a short bio, and a list of tweets that they created. A user may create a new tweet or follow another user.

A bear
@A_single_bear

Hello, I am a bear.

A bear @A_single_bear · Sep 10
I wish I knew how to bark. Maybe humans would yell at me less. I am a bear.

15 228 1.7K

Composition

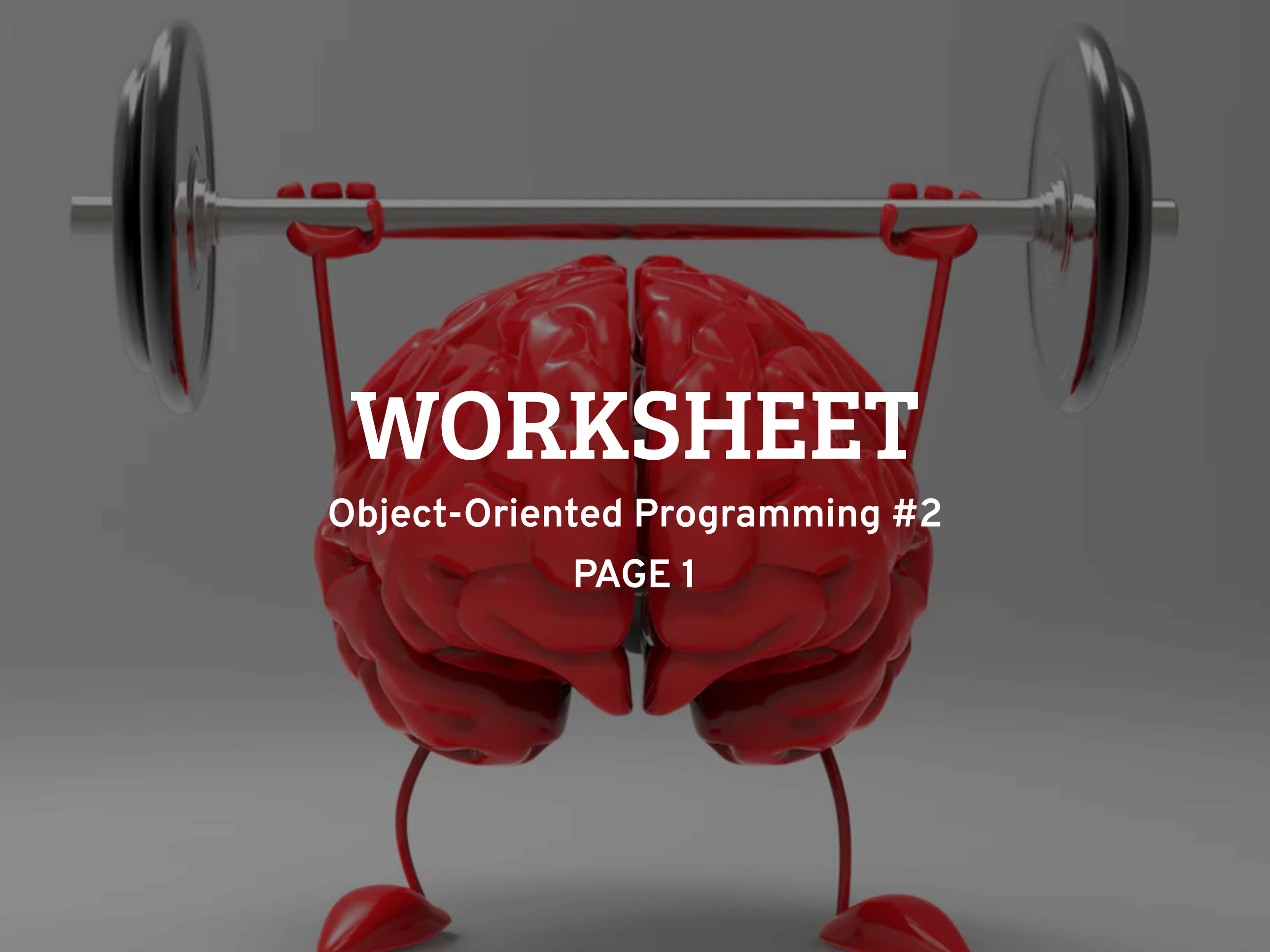
The start of a User class

```
class User:
    """A Twitter user.

    === Attributes ===
    userid: the userid of this Twitter user.
    bio: the bio of this Twitter user.
    tweets: a list of the tweets that this user has made.
    """
    userid: str
    bio: str
    tweets: [**FILL THIS IN**]
```

**What data type should tweets be?

Think on it for a bit, don't answer yet!



WORKSHEET

Object-Oriented Programming #2

PAGE 1

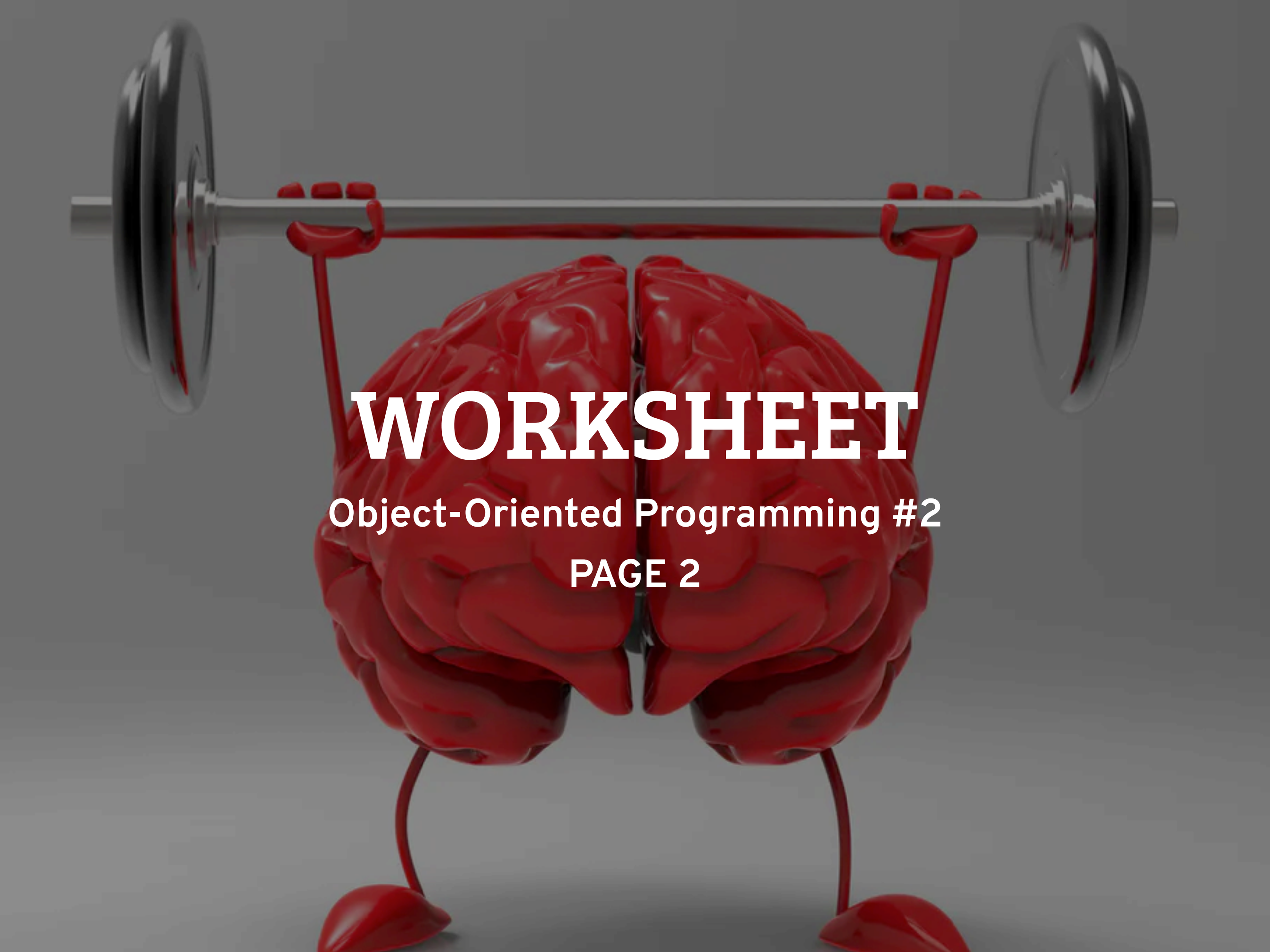
Composition

User and Tweet

Composition is a relationship between two classes where instances of one class contain references to instances of the other

Class Tweet and class User have a **composition** relationship. A user is "composed" of Tweet(s), among other things.

“HAS A” relationship, e.g. “user HAS tweets”



WORKSHEET

Object-Oriented Programming #2

PAGE 2

DEMO

User and Tweet

See tweet.py

Representation

Invariants

Why do we need them?

Key point: How do we document the properties that must be true for every instance of a given class?

Rep Invariants

Why do we need them?

Every instance attribute has a *type annotation*, which restricts the kind of value this attribute can have.

But we often want to restrict attributes values even further.

Example: Tweets can have at most 280 characters

What do we call these restrictions, and how do we communicate them?

Rep Invariants

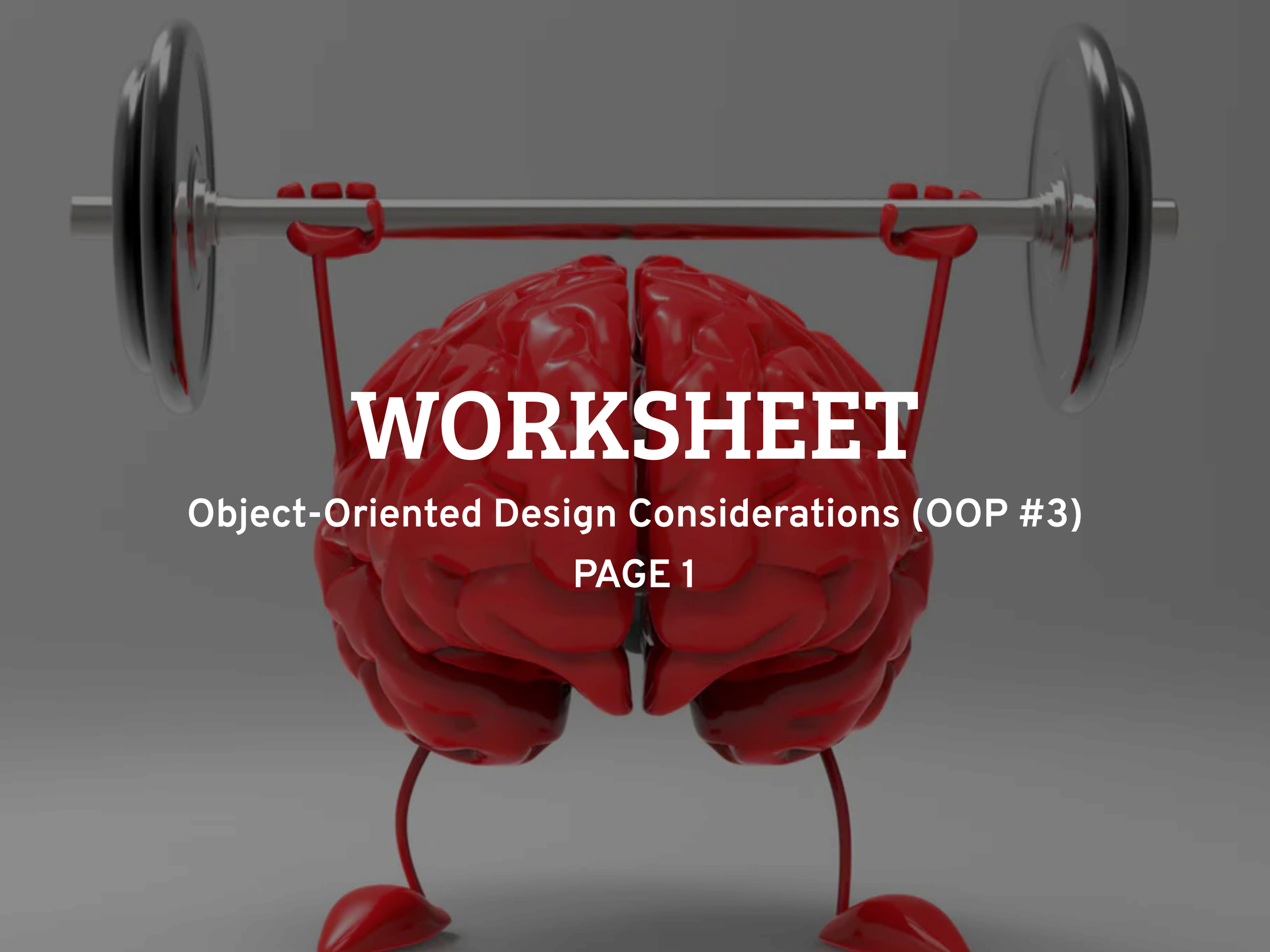
Definition

A **representation invariant** is a property of the instance attributes that every instance of a class must satisfy.

Example

(in words) This tweet's content is at most 280 characters.

(in code) `len(self.content) <= 280`



WORKSHEET

Object-Oriented Design Considerations (OOP #3)

PAGE 1

Today's takeaways

Two questions about RIs

1. Why should we care about representation invariants?

2. How do we enforce representation invariants?

Why do we care?

1. RIs as assumptions

A representation invariant is a property that every instance of a class must satisfy.

When given an instance of that class, we can *assume that every representation invariant is satisfied*.

Why do we care?

1. RIs as assumptions

```
class Tweet:
    def like(self, n: int) -> None:
        self.likes += n
```

self is an instance of Tweet, so we assume that all RIs are satisfied when this method is called.

The representation invariants of Tweet are ***preconditions for every Tweet method.***

How do we enforce?

2. RIs as responsibilities

A representation invariant is a property that every instance of a class must satisfy.

When a method returns, we must *ensure that every representation invariant is satisfied*.

How do we enforce?

2. RIs as responsibilities

```
class Tweet:
    def like(self, n: int) -> None:
        """Record the fact that this tweet
        received <n> likes."""
```

Example: The method must ensure that, at the end, `self.likes >= 0`.

The representation invariants of Tweet are ***postconditions for every Tweet method*.**

How do we enforce?

Strategy 1: Preconditions

Require client code to call methods with “good” inputs, so that the methods won’t violate the representation invariants.

Make no promises if it doesn’t. Therefore, don’t need to check that preconditions are met.

How do we enforce?

Strategy 2: Ignore "bad" inputs

Accept a wide range of inputs, and if an input would cause a representation invariant to be violated, do nothing instead.

Also known as *failing silently*.

How do we enforce?

Strategy 3: Fix "bad" inputs

Accept a wide range of inputs, and if an input would cause a representation invariant to be violated, change it to a "reasonable" or default value before continuing with the rest of the function.

How do we enforce?

Why we write down the RIs

Documenting representation invariants is essential!

How do we enforce?

Why we write down the RIs

The assumption part is handy, but is only legitimate if the responsibility part is fulfilled.

Writing down the Representation Invariants helps ensure:

- You remember these responsibilities
- Others on your team remember these responsibilities
- Now and a year from now when you/they are revising the code!

the zen of python

“Explicit is better than implicit.”

Recap: Meaning and implications of an RI

When we declare a Representation Invariant, we promise that for every instance of the class, it will be true before and after any method call.

Each method has a responsibility to ensure that every representation invariant is satisfied.

- They are a post-condition of every method.

Each method can assume it is true when the method begins.

- They are a pre-condition of every method.

Rep Invariants

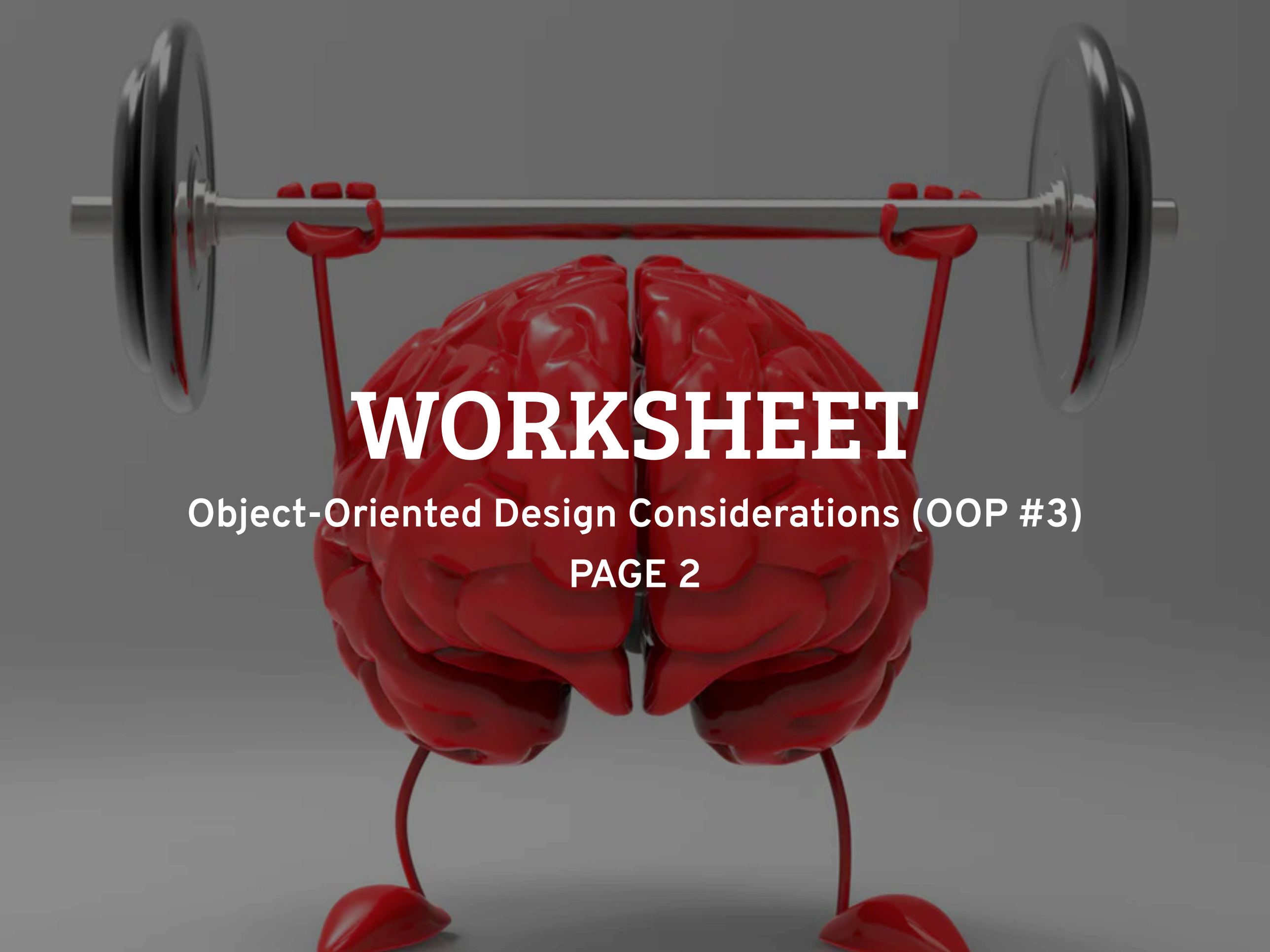
Representation Invariants matter!

They document critical properties that must be understood in order to:

✓write correct code

✓write efficient code

This matters for our own code and for client code that accesses or mutates instance attributes directly.



WORKSHEET

Object-Oriented Design Considerations (OOP #3)

PAGE 2

Design considerations

When adding some new feature in a class, consider what you already have and what you cannot implement without extra attributes and/or methods

Remember: Redundant information is bad (memory space inefficiency, prone to bugs)

Data encapsulation

Privacy

General idea: attributes or methods of a class are made **"private"** to **hide implementation details** or **protect them from unauthorized use**

Basically, only allow reading and writing to them via special methods that the class designer can control

Marking an attribute/method as private **signals that client code should not access it**

Privacy

How to mark something as private

Instance attributes and methods can be marked as **private** by spelling their name with a leading underscore,

e.g. `_content for Tweet`

This means they should not be used outside of class definition

python philosophy

“We're all adults here”

Privacy

It's about communication

A private attribute/method could be...

- very complicated
- subject to several representation invariants
- changed (in name, type, or meaning) at any time

In Python, if you want to access some part of program or data, you generally can, but if you're messing with private data, you're taking a risk because the programmer probably marked it as private for a reason.

Interface vs. Implementation

Privacy