Get to know each other!

What’s one hobby you’ve never tried, but always wanted to?
Announcements

Your first full prep!

If you just joined the course and missed Lab 1 and/or Prep 2, email csc148-2018-09@cs.toronto.edu.

Please double-check lab room assignments on Wednesday.
Object-oriented programming
The fundamental transition

The study of computer science transforms us from being users of technology to being creators of technology.
Key terms and phrases

class
instance (of a class)
instance attribute
method
initializer
dot notation

id, type, value
parameter
local variable
alias
Rebinding self

def mutate(self, x):
    self = NewObject(x)  

Rebinding self doesn’t mutate anything!
Demo: Code analysis in action
Learning Tips: What to do after lecture

**Review:** summarize, question, re-explain

**Share:** meet with a friend or study group

**Get help:** come to office hours!
Composition of classes

HOW DO WE DESIGN CODE IN WHICH DIFFERENT CLASSES INTERACT WITH EACH OTHER?
Users are objects too

A Twitter user has an id, a biography, and tweets (among other things).

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.
Learning Tips: What to do after lecture

**Review**: summarize, question, re-explain

**Share**: meet with a friend or study group

**Get help**: come to office hours!
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: List[Tweet]
User and Tweet

Composition

A relationship between two classes where instances of one class contain references to instances of the other.

“has” relationship, e.g. “user has tweets”
Representation invariants

HOW DO WE DOCUMENT PROPERTIES THAT MUST BE TRUE FOR EVERY INSTANCE OF A GIVEN CLASS?
Tweets can have at most 280 characters

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; what do we call these restrictions, and how do we communicate them?
Representation invariant

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`
Let’s get started!

A representation invariant is a property of the instance attributes (including type annotations) that every instance of a class must satisfy.

Warm-up: complete the first page of today’s worksheet.
Today: two questions about RIs

1. Why should we care about representation invariants?

2. How do we enforce representation invariants?
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.
Representation invariants 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 of self for every Tweet method.
“Explicit is better than implicit.”
Enforcing representation invariants

Every method must ensure that self satisfies all representation invariants are satisfied after the method ends.

The representation invariants of a class are postconditions of self for every Tweet method.
Strategy 1: Preconditions

Require client code to call methods with “good” inputs, so that the methods won’t violate the representation invariants.
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.
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.
Discuss the pros and cons of each

**Strategy 1:** use preconditions

**Strategy 2:** ignore bad inputs

**Strategy 3:** fix bad inputs
Direct attribute access

Even if our methods are perfect, client code can access and mutate instance attributes directly.

*Documenting representation invariants is essential.*
Privacy

Instance attributes and methods can be marked as private by spelling their name with a leading underscore, e.g. `_content`.

Marking an attribute/method as private signals that client code should not access it.
Privacy is about communication

A private attribute/method could be...
- very complicated
- subject to several representation invariants
- seemingly unrelated to the actual purpose of the class
- changed at any time
Interface vs. Implementation