CSC110 Lecture 26: Abstract Data Types and Stacks

David Liu, Department of Computer Science

*Navigation tip for web slides: press **?** to see keyboard navigation controls.*

Announcements and Today's Plan

- Assignment 4 has been posted, **due Wednesday!**
  - Check out the A4 FAQ (+ corrections)
  - Additional TA office hours
  - Review advice on academic integrity
- Term Test 3 info has been posted
  - And the Reference Sheets
- **No tutorial this Friday** (to give you more time to prepare for the term test)

Story so far

Data: data types, literals, operators, comprehensions, data classes

Functions: using built-in functions, methods; defining our own functions

Control flow: if statements, for loops, while loops

Function correctness: formal specification, preconditions, representation invariants; unit tests, property-based tests

Proof and algorithms: number theory, cryptography

Running time: Big-O/Omega/Theta; running-time analysis and worst-case running-time analysis

Over the next two weeks, we'll take a deep dive into **defining our own data types** in Python.

Today you'll learn to...

1. Define the terms abstract data type and concrete data type.
2. Define the Stack abstract data type and implement it as a Python class.
3. Define the term private instance attribute and explain its purpose in class design.
4. Compare two different implementations of the Stack ADT by analysing their running times.

Data types, abstract and concrete

We've seen built-in Python data types (e.g., `int`, `list`, `set`).

We've defined our own Python data types using data classes.

These are called **concrete data types** since they have a concrete implementation in Python code.

Also called **classes** in Python.

Note: Every data class is a class, but not every class is a data class. In this week's prep, you learned how to define a Python class without using `@dataclass`!

An **abstract data type (ADT)** is an abstract (no code) definition of a data type:

- what data is stored
- what operations can be performed on the data

Abstract data types are independent of programming language.

They form a common vocabulary that computer scientists and software engineers can use to communicate across programming languages/libraries/etc.

Set ADT

- Data: a collection of unique elements
- Operations: get size, insert a value, remove a specified value, check membership in the set.

List ADT

- Data: an ordered sequence of elements (possibly with duplicates)
- Operations: get size, access element by index, insert a value at a given index, remove a value at a given index

Mapping ADT

- Data: a collection of key-value pairs, where each key is unique and associated with a single value
- Operations: get size, lookup a value for a given key, insert a new key-value pair, remove a key-value pair, update the value associated with a given key

Warning!

There is **not** a one-to-one correspondence between abstract and concrete data types (in Python or any other programming language).

E.g., a Python `list` can be used to implement the Set ADT, and a Python `dict` can be used to implement the List ADT.

New abstract data types

Over the next two lectures, we'll study three new abstract data types: **Stack**, **Queue**, and **Priority Queue**.

For each one, we'll learn about:

- its abstract definition (independent of programming language)
- how to implement it as a Python class
- how to compare different implementations by analysing their running times

The Stack Abstract Data Type

Stack ADT

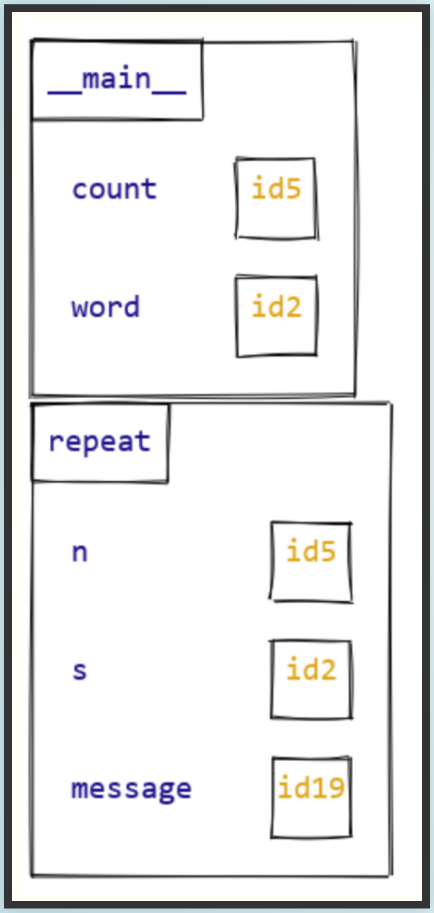
- Data: A collection of items
- Operations:
  - determine whether the stack is empty
  - add an item (`push`)
  - remove the most recently-added item (`pop`)

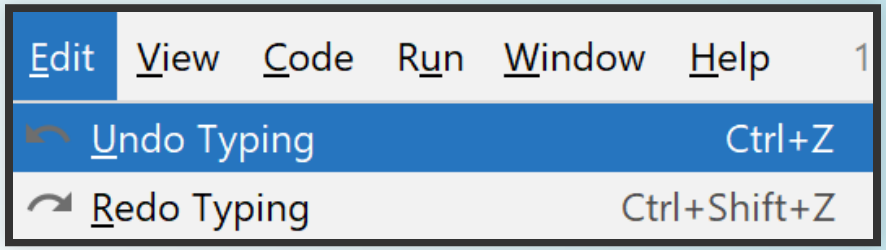
In a stack, items are removed in reverse order from how they are added. Also known as last in, first out (LIFO) order.

```
>>> s = Stack()
>>> s.is_empty()
True
>>> s.push(1)
>>> s.push(4)
>>> s.push(10)
```

```
>>> s.pop()
10
>>> s.pop()
4
>>> s.pop()
1
```

Applications of stacks



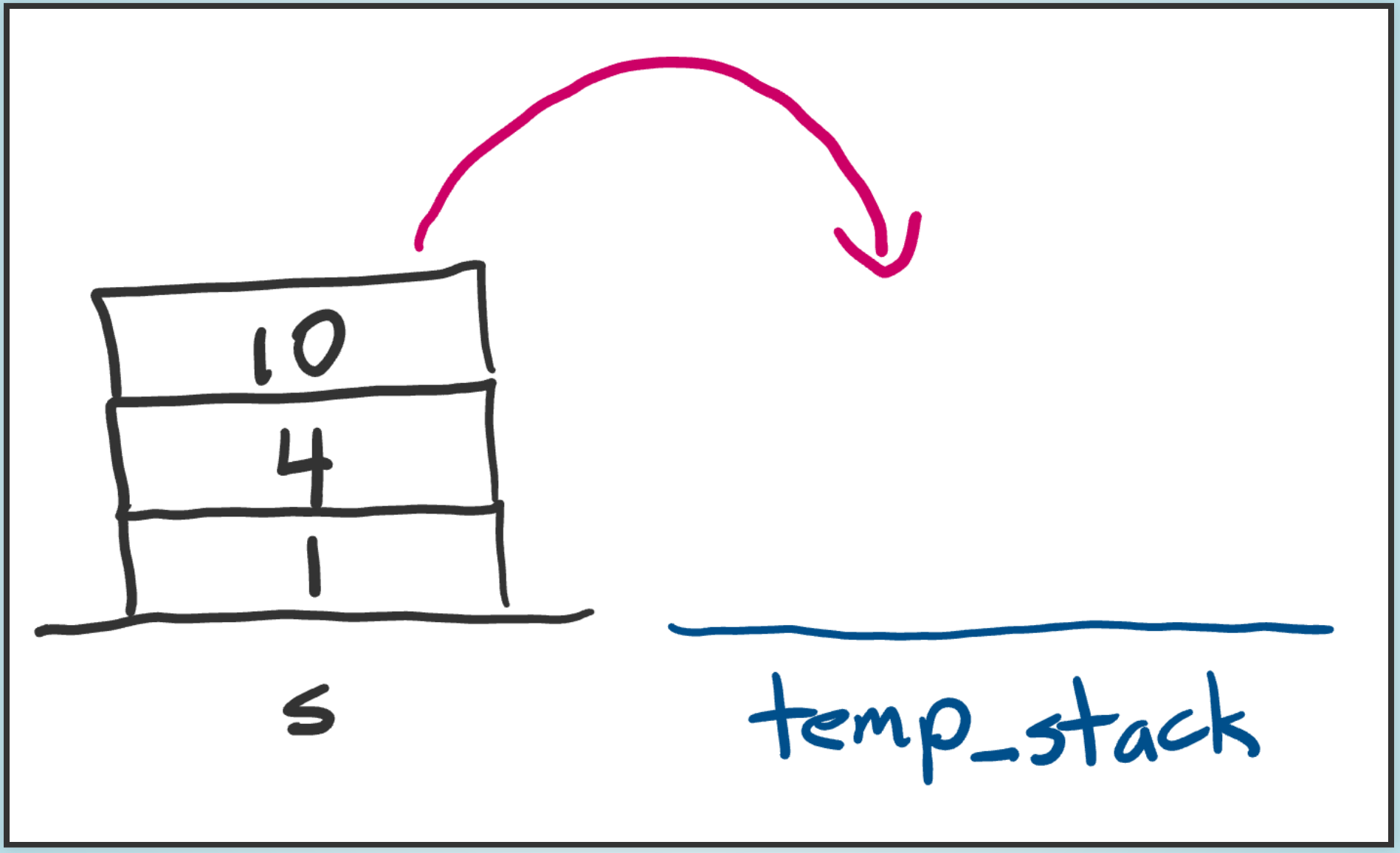


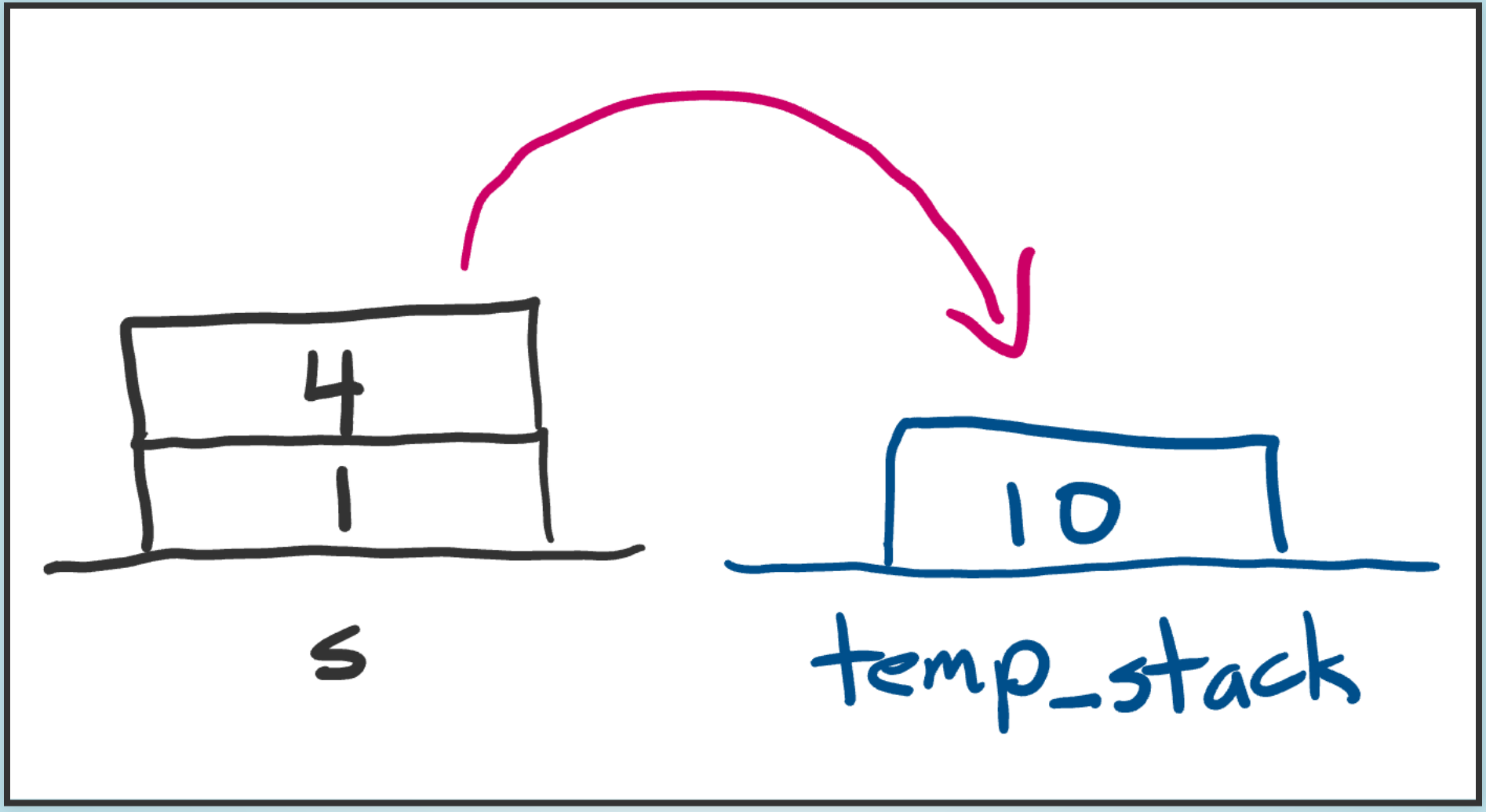
```
from typing import Any


class Stack:

    def __init__(self) -> None:
        """Initialize a new empty stack."""

    def is_empty(self) -> bool:
        """Return whether this stack contains no items."""

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack."""

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack

        Precondition: not self.is_empty()
        """
```

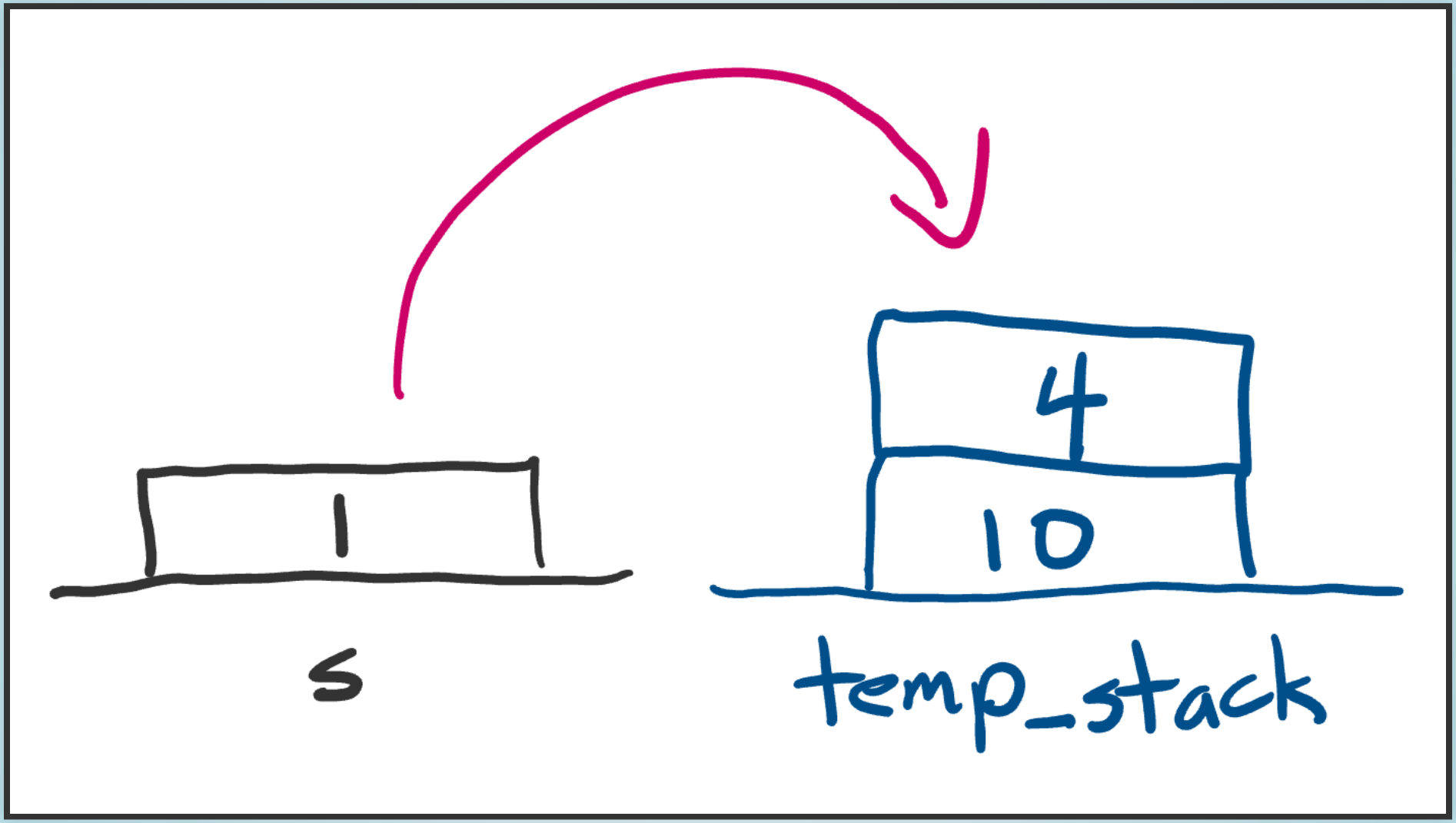
Demo

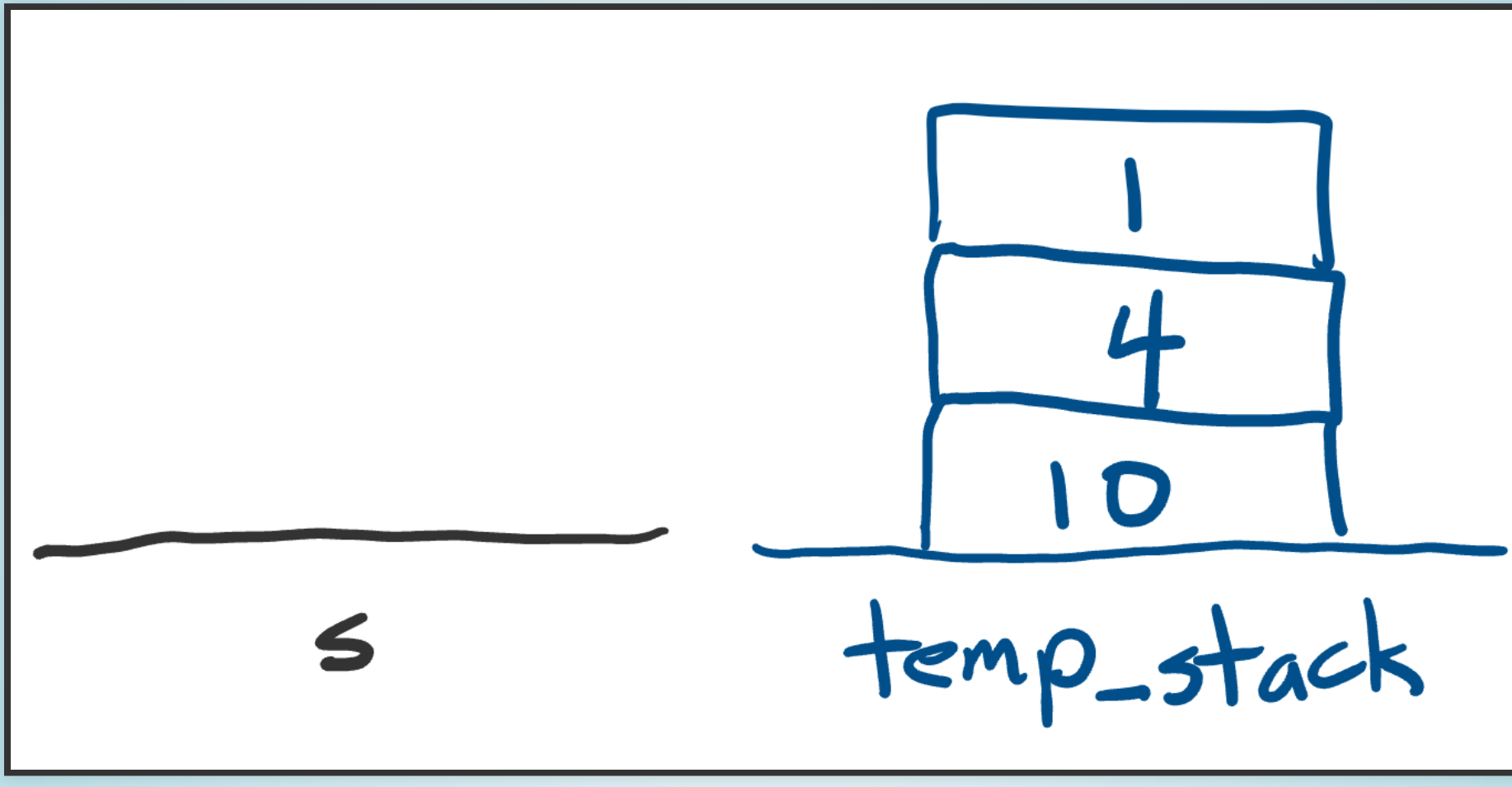
Exercise 1: Using Stacks

(Comment: This is really good practice debugging!)


10
4
1
s
temp_stack

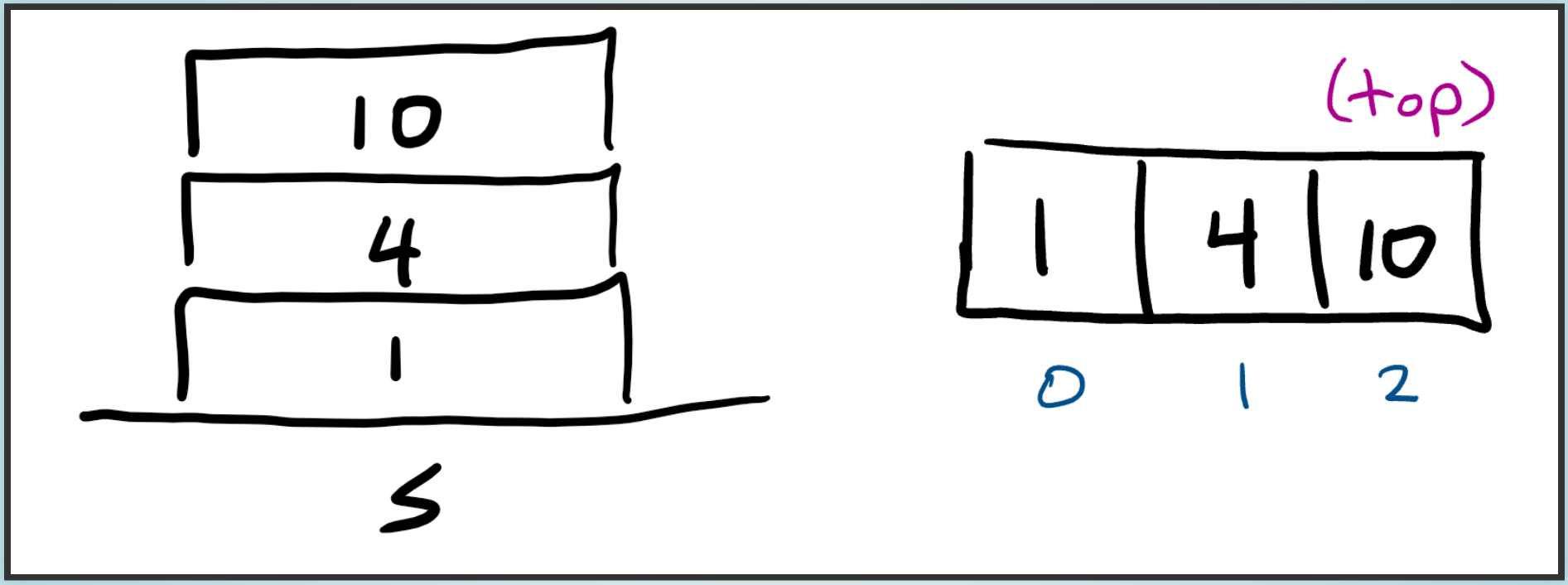

4
1
10
s
temp_stack


1
4
10
s
temp_stack


1
4
10
s
temp_stack

Implementing a Stack in Python

Idea: store the items in the stack as a list, using the end of the list to represent the top of the stack.



To Pycharm!

The `items` attribute

For all classes we've seen so far, every instance attribute is part of their public interface.

All users of the class can access this attribute...

...which may or may not be a good thing!

A **private instance attribute** is an instance attribute that is not meant to be accessed by users of the class, only the class' methods.

Opposite: **public instance attribute**.

In Python, private instance attributes are named with a leading underscore _.

Demo!

Warning!

Unlike some other programming languages, in Python private instance attributes can still be accessed by external code!

So what's the point of marking an attribute as private?

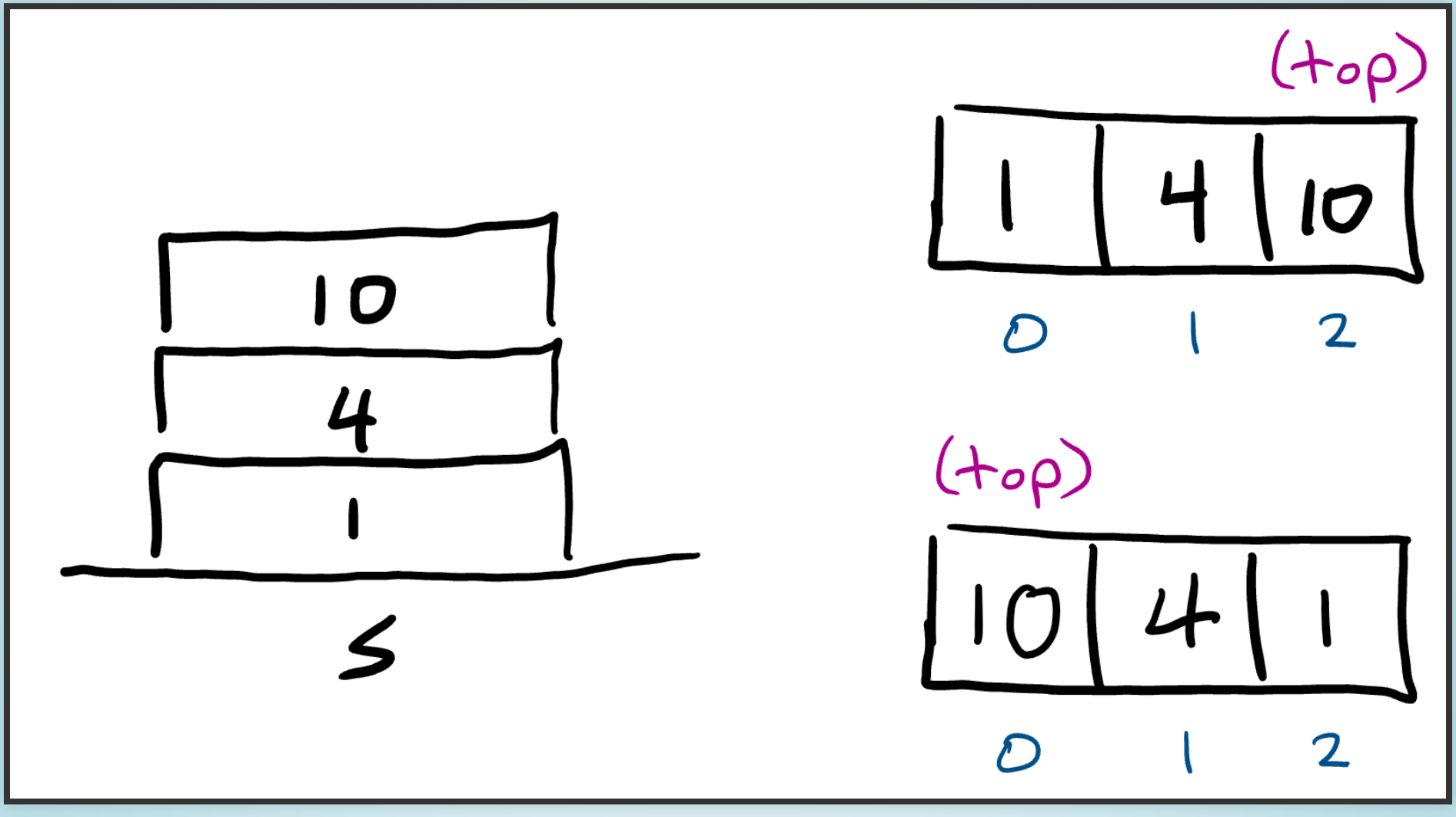
Marking attributes as private communicates intent.

- Reduces cognitive load on the user of the class ("one less thing to worry about")
- Gives freedom to implementer of the class to modify or remove this attribute at a later point.

Stack running-time analysis

Exercise 2: Stack implementation and running-time analysis

Comparing stack implementations



Comparing stack implementations

	`Stack1`	`Stack2`
Correct?	Yes	Yes
Code complexity	Simple	Simple
Efficiency (`push`/`pop`)	$\Theta(1)$	$\Theta(n)$

Exceptions as part of the public interface (if time)

```
class Stack:
    def pop(self) -> Any:
        """Remove and return the element at the top of this stack

        Preconditions:
            - not self.is_empty()
        """
```

Preconditions are a restriction on the person using the class, who must verify that the precondition is satisfied before calling the method.

```
if not my_stack.is_empty():
    top_item = my_stack.pop()
```

Letting it fail (demo)

Defining a custom exception

```
class EmptyStackError(Exception):
    """Exception raised when calling pop on an empty stack."""


class Stack1:
    ...

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack

        Raise an EmptyStackError if this stack is empty.
        """
        if self.is_empty():
            raise EmptyStackError
        else:
            return self._items.pop()
```

In this version, `EmptyStackError` is part of the public interface of the `Stack1` class.

Users can handle this exception when calling `pop` (see Course Notes for details).

Summary

Today you learned to...

1. Define the terms abstract data type and concrete data type.
2. Define the Stack abstract data type and implement it as a Python class.
3. Define the term private instance attribute and explain its purpose in class design.
4. Compare two different implementations of the Stack ADT by analysing their running times.

Homework

- Readings:
  - From prep: 10.1, 10.2, 10.3
  - Today: 10.4, 10.5, (if time) 10.6
  - For next class: 10.6, 10.7, 10.8
- Work on Assignment 4
- Study for Term Test 3

Balance Artist Rocky Byun (Instagram)

Photo from Vice