CSC110 Lecture 27: Queues and Priority Queues

David Liu and Tom Fairgrieve, Department of Computer Science

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

Announcements and Today’s Plan

Announcements

- 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)

Today you'll learn to...

1. Define a custom exception type and use it as part of a method's public interface.
2. Define and implement two new abstract data types, the Queue and Priority Queue.
3. Compare implementations of these ADTs by analysing their running times.

Exceptions as part of the public interface

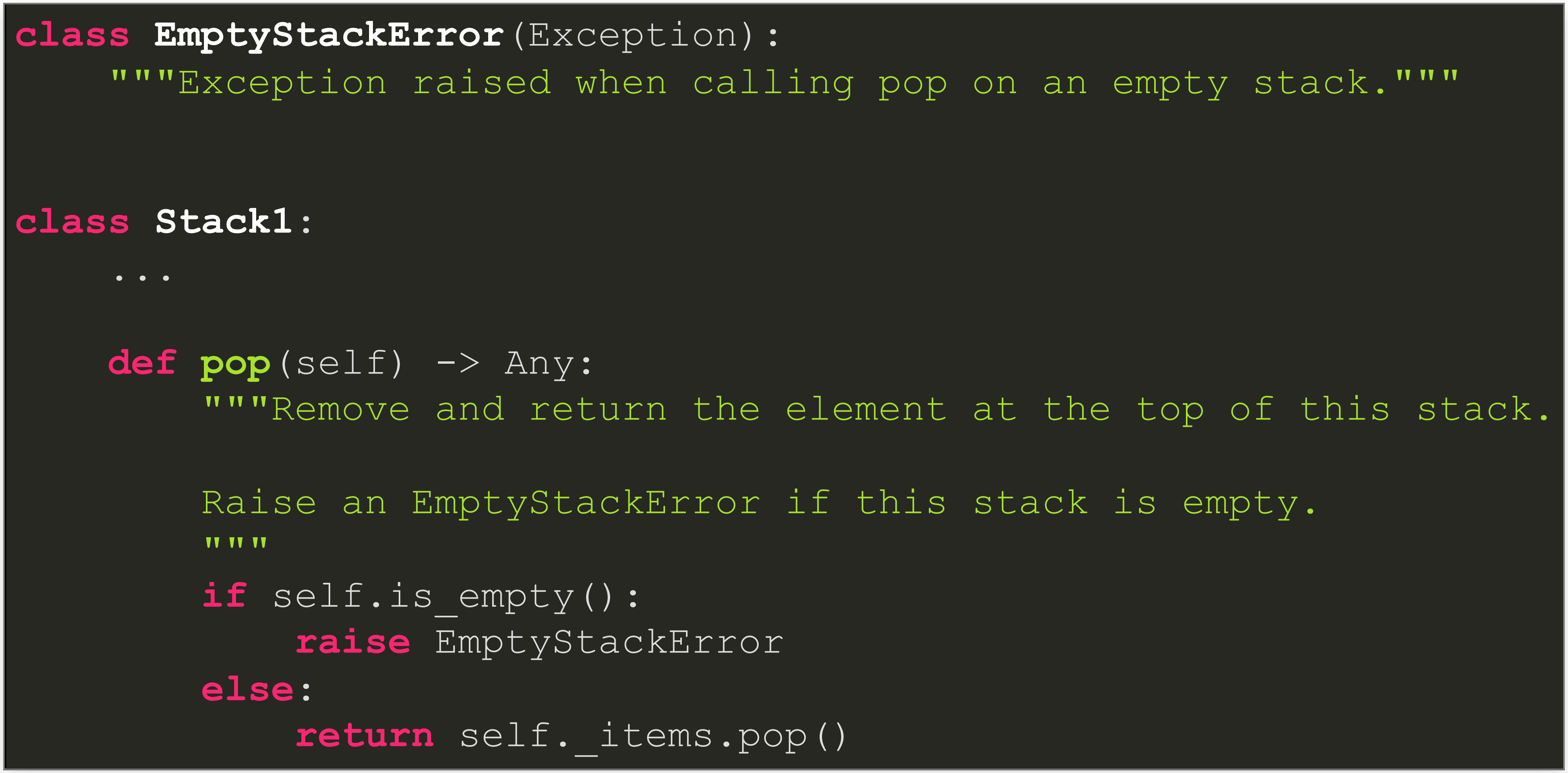
```python
class Stack
    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        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.

```python
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()
```

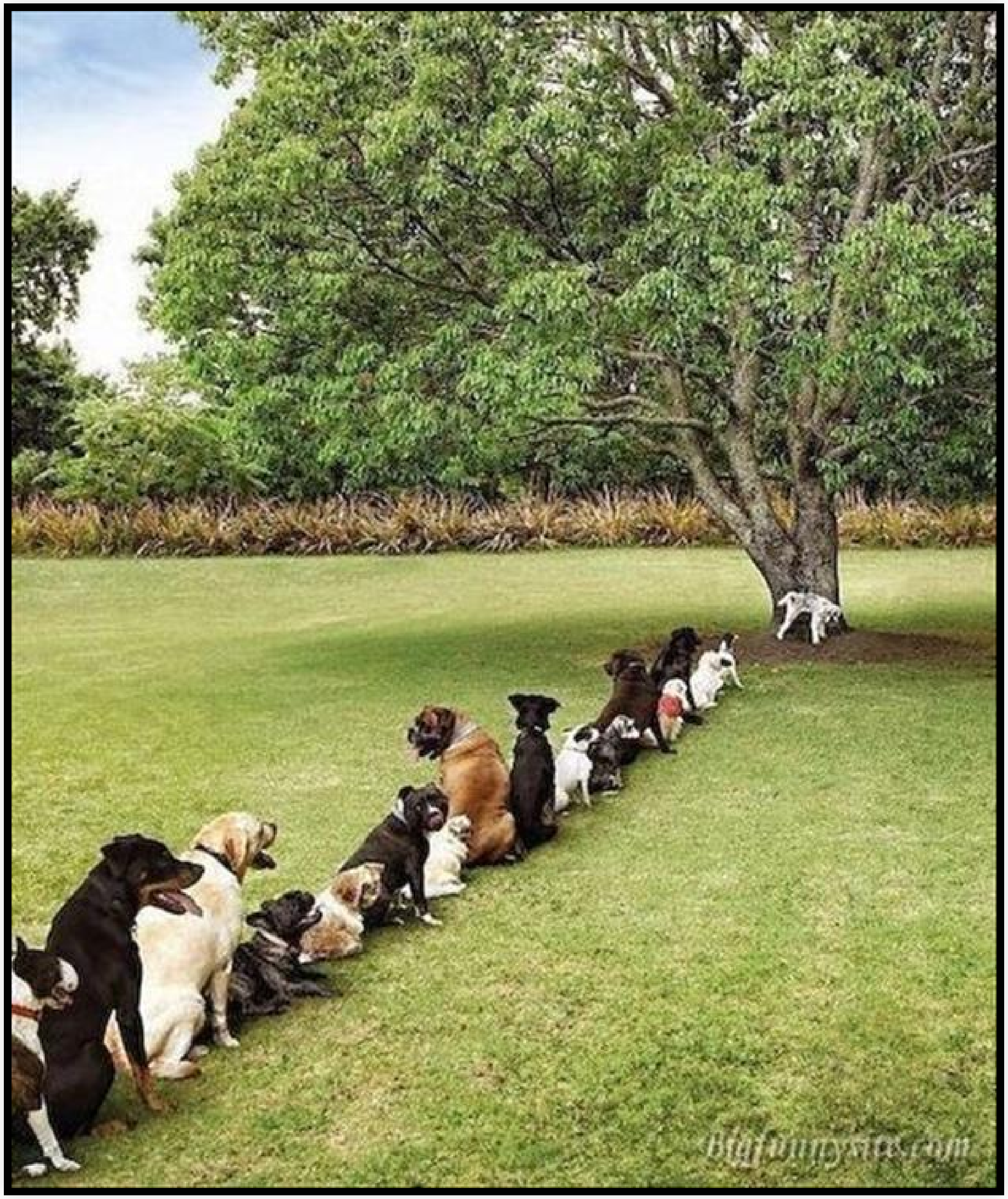
Now, `EmptyStackError` is part of the public interface of the `Stack1` class.

Implementors can customize the error message that a user sees.

Users can handle this exception when calling `pop`.

(See Course Notes for details.)

The Queue ADT

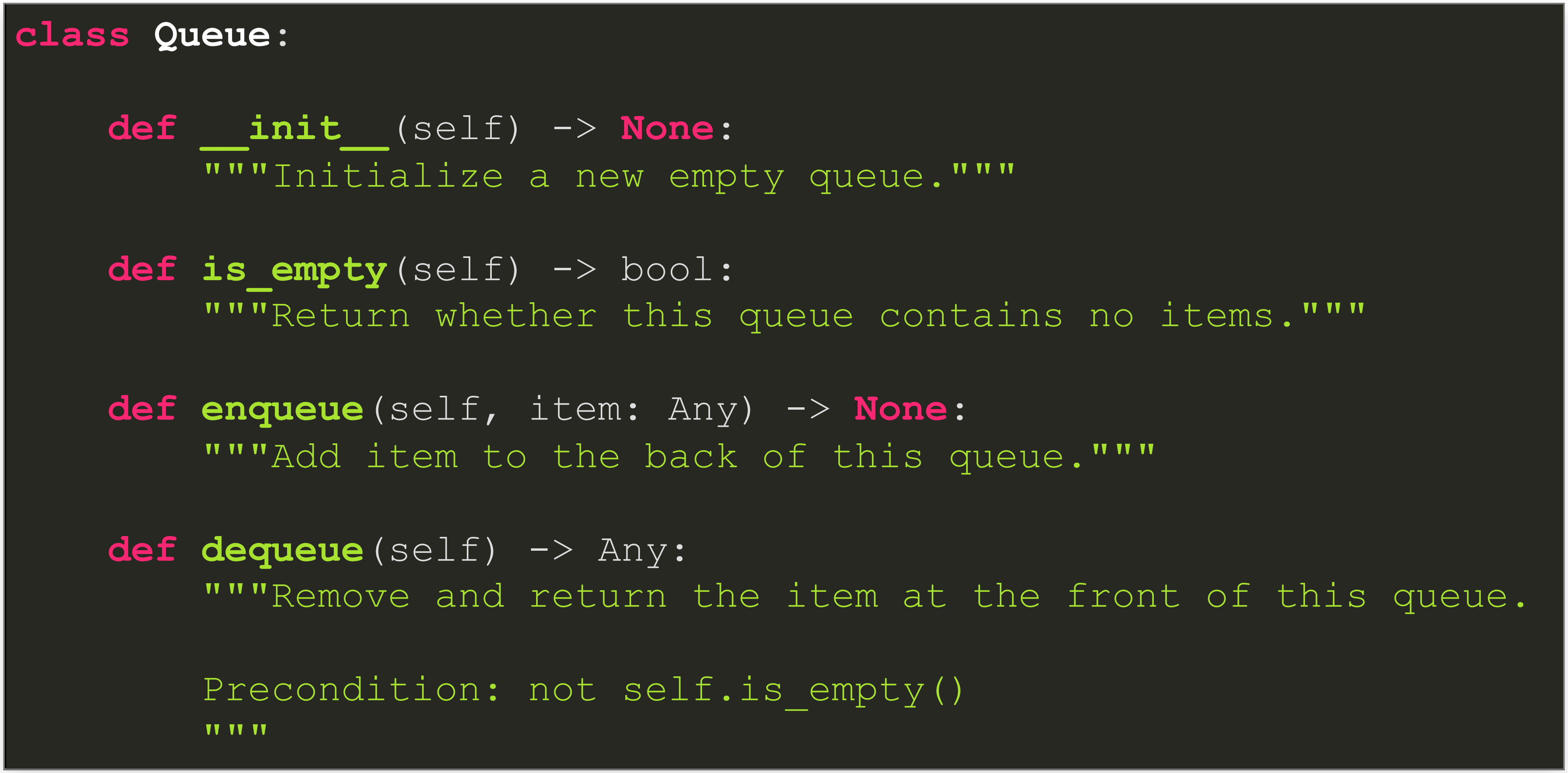


Queue

- Data: A collection of items
- Operations:
  - determine whether the queue is empty
  - add an item (`enqueue`)
  - remove the least recently-added item (`dequeue`)

Items are removed from a queue in the same order as how they are added.

Also known as first in, first out (FIFO) order.



```
class Queue:

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

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

    def enqueue(self, item: Any) -> None:
        """Add item to the back of this queue."""

    def dequeue(self) -> Any:
        """Remove and return the item at the front of this queue.

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

```
>>> q = Queue()
>>> q.is_empty()
True
>>> q.enqueue('hello')
>>> q.enqueue('goodbye')
>>> q.enqueue('!')
```

```
>>> q.dequeue()
'hello'
>>> q.dequeue()
'goodbye'
>>> q.dequeue()
'!'
```

Implementing a Queue

Idea: store the items in the queue in a list, using the front of the list to represent the front of the queue.

To PyCharm!

Exercise 1: Queue implementation and running time analysis

There isn’t always a clear “best” implementation!

Queue Operation	**“Front of list” runtime**	**“Back of list” runtime**
`enqueue`	$\Theta(1)$	$\Theta(n)$
`dequeue`	$\Theta(n)$	$\Theta(1)$

The Priority Queues ADT

T3 O1 D3 O1

Priority Queue

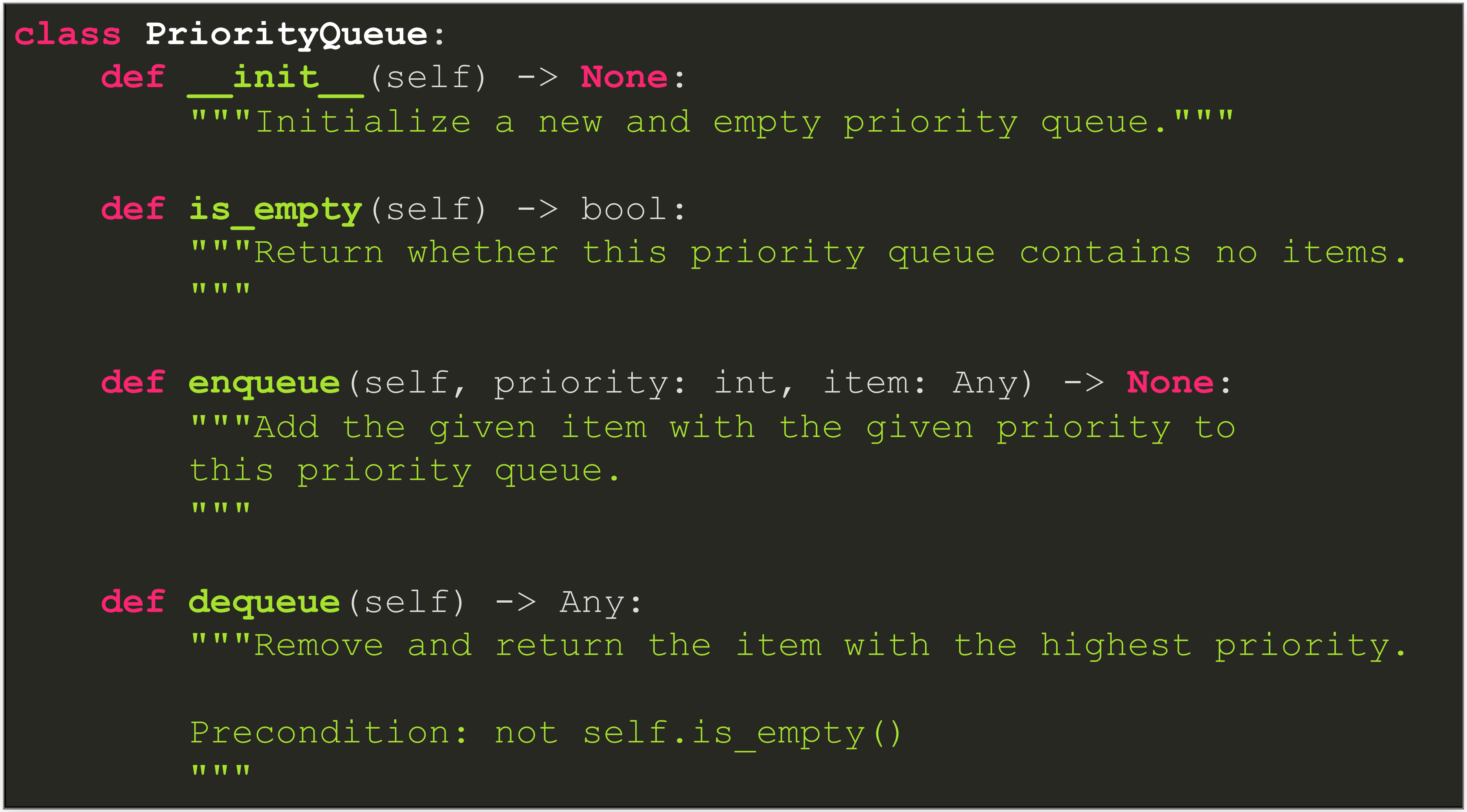
- Data: A collection of items and their priorities
- Operations:
  - determine whether the priority queue is empty
  - add an item with a given priority (`enqueue`)
  - remove the item with the highest priority (`dequeue`)

```
>>> pq = PriorityQueue()
>>> pq.is_empty()
True
>>> pq.enqueue(1, 'hello')
>>> pq.enqueue(5, 'goodbye')
>>> pq.enqueue(2, 'hi')
>>> pq.dequeue()
'goodbye'
```

Note: many ways of representing “highest priority”.

In this lecture, we’re using integers, where the larger the integer, the higher the priority.

Next week, a different kind of priority!



```python
class PriorityQueue:
    def __init__(self) -> None:
        """Initialize a new and empty priority queue."""

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

    def enqueue(self, priority: int, item: Any) -> None:
        """Add the given item with the given priority to
        this priority queue.
        """

    def dequeue(self) -> Any:
        """Remove and return the item with the highest priority.

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

Exercise 2: Priority Queues

Alternate implementation: sorted priority queues

```python
class PriorityQueueSorted:
    # Private Instance Attributes:
    #   - _items: A list of the priorities and items in
    #             this priority queue, SORTED BY PRIORITY.
    ...

    def dequeue(self) -> Any:
        last_pair = self._items.pop()
        return last_pair[1]
```

`PriorityQueueSorted.dequeue` takes $\Theta(1)$ time!

```python
class PriorityQueueSorted:
    def enqueue(self, priority: int, item: Any) -> None:
        self._items.append((priority, item))

        # Sort the tuples by priority
        # (This version works if there are no ties in priorities.)
        self._items.sort()
```

`list.sort` has a worst-case running time of $\Theta(n \log n)$.

So the worst-case running time of `PriorityQueueSorted.enqueue` is $\Theta(n \log n)$!

Looking ahead

Operation	`PriorityQueueUnsorted` runtime	`PriorityQueueSorted` runtime
`enqueue`	$\Theta(1)$	$\Theta(n \log n)$
`dequeue`	$\Theta(n)$	$\Theta(1)$

It's possible to implement the PriorityQueue ADT using a data structure called a heap, so that both `enqueue` and `dequeue` have a worst-case running time of $\Theta(\log n)$.

Look forward to this in CSC263/265!

Summary

Today you learned to...

1. Define a custom exception type and use it as part of a method's public interface.
2. Define and implement two new abstract data types, the Queue and Priority Queue.
3. Compare implementations of these ADTs by analysing their running times.

Homework

- Readings:
  - Today: 10.6, 10.7, 10.8
  - on Thursday: 10.9, 10.10
- Work on Assignment 4
- Study for Term Test 3