LECTURE 17

STACKS AND QUEUES

MCS 260 Fall 2021

David Dumas

REMINDERS

- Homework 6 posted, due Tuesday at 10am
- Project 2 autograder opens Monday
- Project 2 due 6pm central on Fri Oct 8

JOINING PATH COMPONENTS

```
import os

fn = os.path.join("data","pride.txt")
```

Now `fn` is `"data\\pride.txt"` if running on Windows, or `"data/pride.txt"` on MacOS or Linux.

OTHER OS MODULE GOODIES

`os.path.exists(fn)` returns a boolean to indicate whether a file with the given name exists already.

More on this module later!

TWO DATA STRUCTURES

- **Stack** - LIFO (last in, first out) storage of items. Like a physical stack, where you can only access the item most recently added.
- **Queue** - FIFO (first in, first out) storage of items. Like a line or waiting list. Add to one side, remove from the other.

STACK

Adding an item is called **push**, removing an item is called **pop**.

Often used for:

- **Undo** a sequence of actions.
- **Syntax highlighting**: Which "(" matches this ")" ?

You can make a stack using a Python list:

- push becomes `list.append`
- pop becomes `list.pop`

QUEUE

Adding an item is called **enqueue**. Removing an item is called **dequeue**.

Common applications:

- **Work** to be done / data to be processed.
- **Temporary storage**, e.g. for communication.

Maybe do this with a list?

- enqueue becomes `list.append(item)`
- dequeue becomes `list.pop(0)`

WARNING

Using a list as a queue is NOT efficient.

Removing an item from the beginning of a list takes time proportional to list size.

More efficient: deque from the collections module

```
import collections
Q = collections.deque() # pronounced "deck"
Q.append("first in")    # enqueue
Q.append(260)
Q.append("last in")
while len(Q)>0:
    x = Q.popleft()     # dequeue
    print(x)
```

Output:

```
first in
260
last in
```

Notice deque implements queue operations:

- enqueue becomes `deque.append(item)`
- dequeue becomes `deque.popleft()`

Efficiency means time to add or remove an item is independent of how many items are present (like stacks).

STACK APPLICATION

Checking parenthesis matching (example of parsing)

This expression is ok:

- `((2+3) - (4*5))`

These are not:

- `((5*7))) - ((2)`
- `((2+3)-5))`

Goal: Decide if ok, give useful error if not.

parens.py

```python
"""Check arithmetic expression for balanced parentheses"""

print("Enter an arithmetic expression in parentheses:")
s = input().strip()

paren_stack = []
for i,c in enumerate(s):
    if c == "(":
        paren_stack.append(i)
    elif c == ")":
        if len(paren_stack) == 0:
            # Too many right parentheses
            print("ERROR: Extra right parenthesis")
            print(s)
            print(i*" " + "^")
            break
        paren_stack.pop()

if len(paren_stack) > 0:
    # Unclosed left parenthesis
    i = paren_stack.pop()  # Where was the left paren that's open?
    print("ERROR: Unclosed parenthesis")
```

REFERENCES

- Optional text Brookshear & Brylow discuss stacks and queues in Section 8.1
- Downey does not discuss stacks and queues in general
- *Data Structures and Algorithms in Python* by Goodrich, Tamassia, and Goldwasser discusses stacks and queues in Chapter 6.

REVISION HISTORY

- 2021-10-01 Initial publication