# List-based Stacks and Queues

## CMPT 145

# Learning Objectives

After studying this chapter, a student should be able to:

- Employ data abstraction to implement a queue.
- Employ data abstraction to implement a stack.
- Explain the concept of an adapter in the context of abstract data types.
- Describe the list-based implementation of stacks and queues.
- Explain some of the reasons that wrapping an ADT around a list can improve correctness and adaptability.

# Two implementations

```
# initialize data structures
expr_list = exprn.split()
evalu8 = []

# evaluate the expression
for c in expr_list:

  if c == '*':
    v1 = evalu8.pop()
    v2 = evalu8.pop()
    evalu8.append(v1*v2)
  ...
```

```
import Queue as Q
# initialize data structures
expr_list = exprn.split()
evalu8 = S.Stack()
expr = Q.Queue()

# put items into Queue
for c in expr_list:
  expr.enqueue(c)

# evaluate the expression
while not expr.is_empty():
  c = expr.dequeue()

  if c == '*':
    v1 = evalu8.pop()
    v2 = evalu8.pop()
    evalu8.push(v1*v2)
  ...
```

# Using ADTs instead of lists

- Clarifies the algorithm
- Prevents errors, e.g., `pop()` vs. `pop(0)`
- Enhances adaptability (in general)
- Makes the script longer

# Adapting Python Lists to Implement Queues

- An adaptor is a program that makes ADT $A$ look like another ADT $B$
- If we have ADT $A$, we can write operations that make $A$ look like it's a $B$.
- We'll adapt Python lists to implement a Queue
- Very common programming task!
- Usually not this trivial!

Code walk-through

# Adapting Python Lists to Implement Stacks

- We'll adapt Python lists to implement a Stack

# Code walk-through

# Example 1

1. Change the Queue code so that the front and back are reversed.
2. Change the Stack code so that the top is on the opposite end of the list.

In both cases, change the ADT implementation only. Code that uses Queue or Stack will not change!

# Discussion 2

For the following scenarios:

- Is it better to use Stack and Queue ADTs;
- Or is it okay to use Python lists without any abstraction?

1. CMPT 145, for educational purposes!
2. Large projects, important projects, paid projects.
3. Your code is not part of a long term solution.
4. Your code is a very short-lived prototype
5. Your code will not run more than a few times
6. Software engineering course assignments and projects