CSC148-Section:L0301
Week#3-Monday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Stack applications
- Sack (bag)
- Generalize Stack/Sack/Queue into Container

Stack applications

- What is the output when calling m3()

- How does python execute them?
  - Using a stack see the following slides

```python
def m1():
    print("inside m1")

def m2():
    m1()
    print("inside m2")

def m3():
    m2()
    print("inside m3")

m3()
```

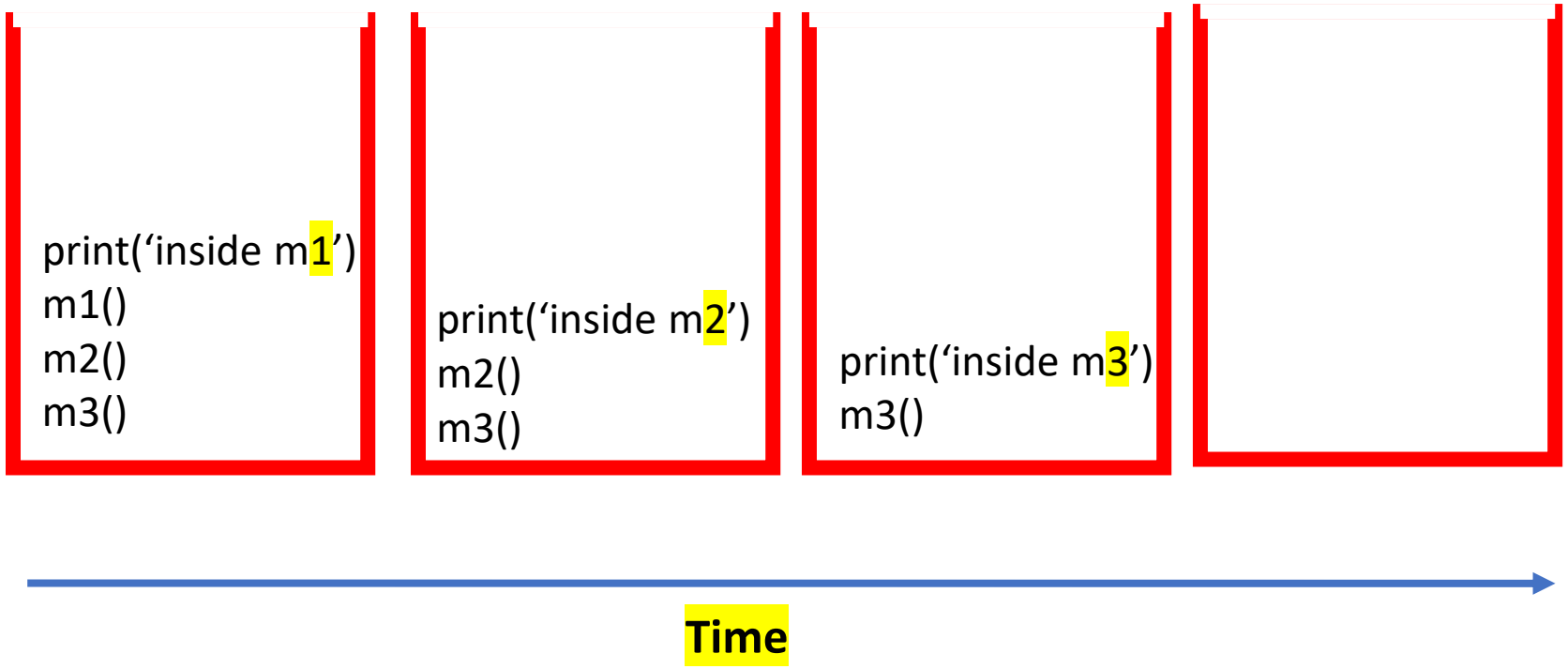
```
C:\Users\AAA\AppData\Local\Programs\Python\
inside m1
inside m2
inside m3
```

Stack applications

- Executing methods (simplified)

Time

add(m3)
add(m2)
add(m1)
add(print('inside m1'))
Nothing to add so start removing
pop(print('inside m1'))
pop(m1()) # done with m1
add(print('inside m2'))
Nothing to add so start removing
pop(print('inside m2'))
pop(m2())# done with m2
add(print('inside m3'))
Nothing to add so start removing
pop(print('inside m3'))
pop(m3())# done with m3



Stack applications

- **Matching opening and closing parentheses, brackets, braces**

e.g.: which one is correct?

(1 + [7 - { 8 / 3] })

(1 + [7 - { 8 / 3 }])

Stack applications

(1 + [7 - { 8 / 3 }])

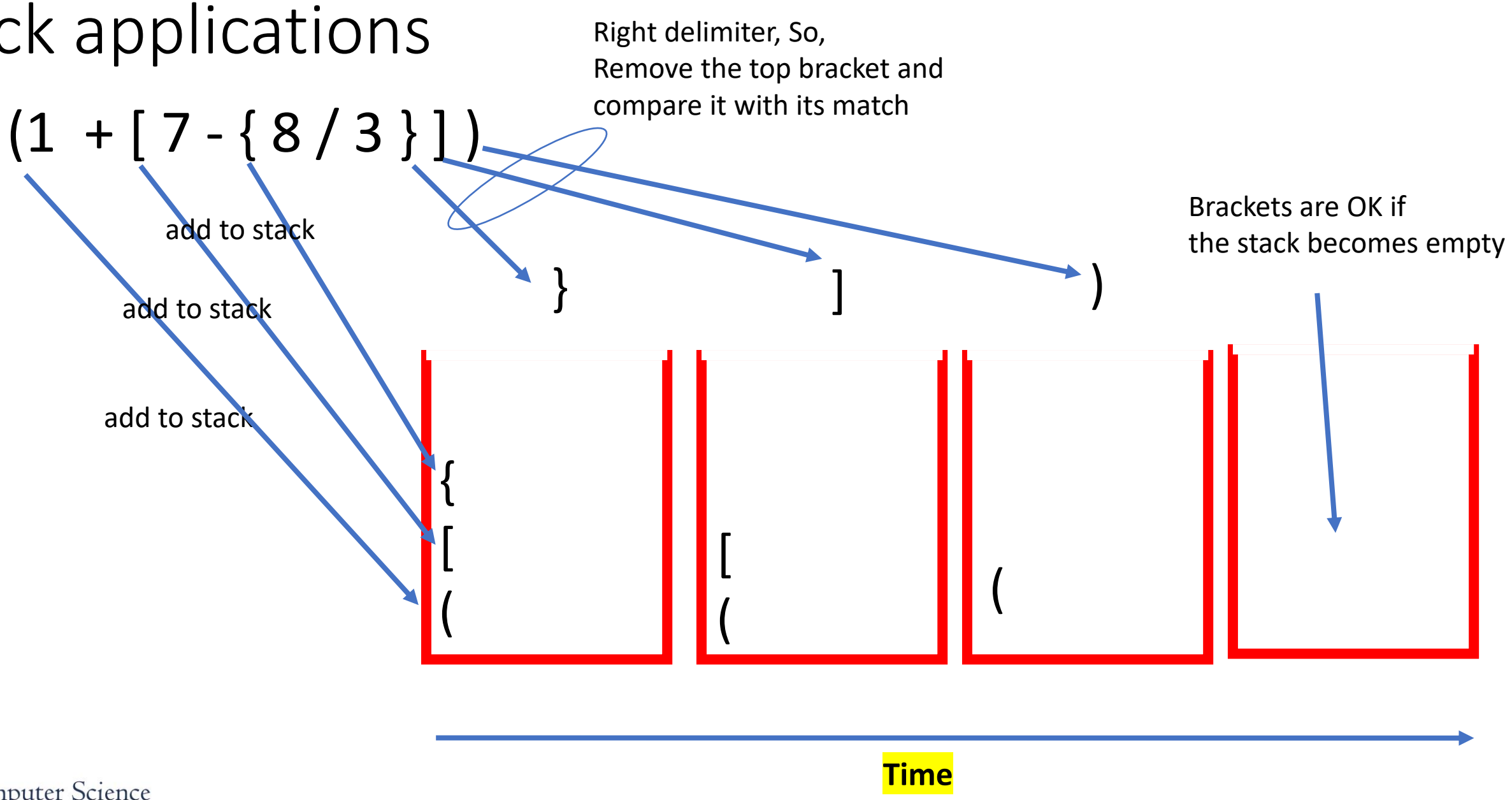
<u>Idea:</u>

Create an empty stack

Go through string from left to right:

1. Add left brackets to stack
2. Ignore none brackets
3. Found right bracket remove and compare
   - If matching continue
   - Else return false

Stack applications



```
    """
    st = stack.Stack()
    left_delim = {")": "(", "]": "[", "}": "{"}
    for c in s:
        if c not in "()[]{}":  # ignore none delimiters
            pass
        elif c in "([{":  # left add to the stack
            st.add(c)
        elif not st.is_empty():  # stack has some left brackets
            if left_delim[c] != st.remove():  # remove the delimieter at top
                                              # compare it with its match
                return False  # right does not match left
        else:
            return False
    return st.is_empty()  # to make sure that no left brackets left
```

code

- You can find the full code for **Matching opening and closing parentheses, brackets, braces** in the course website under section **MWF2 (L0301)**
- with the following file name:
  - stackt_apps_brackets.py
- Download it Try different things with them and practice
  - Do not be afraid of doing mistakes

Sack (bag) class design

Here's a description of a sack, which has similar features to a stack:

> *A sack contains items of various sorts. New items are added on to a random place in the sack, so the order items are removed from the sack is completely unpredictable. It's a mistake to try to remove an item from an empty sack, so we need to know if it is empty. We can tell how big a sack is.*

Take a few minutes to identify the main noun, verb, and attributes of the main noun, to guide our class design.

Sack (bag) class design

- Name: Sack
- Public Attributes: None
- Methods: add, remove, is_empty

- remove should be unpredictable.

implementation possibilities

- The same as Stack except:
  - remove()
  - Slightly different

```
def remove(self) -> object:
    """ Remove and return some random element of Sack self.

    Assume Sack self is not empty.
    >>> s = Sack()
    >>> s.add(7)
    >>> s.remove()
    7
    """
    if self.is_empty():
        raise EmptyContainerException
    else:
        i = random.randint(0, len(self._storage)-1)
        return self._storage.pop(i)
```

ADTs

- Stack
- Sack
- Queue

ADTs

- Stack - LIFO
- Sack – Random out
- Queue – FIFO

- All have add/remove/is_empty
  - Generalize them into Container

Why Container? Example #1

container_simple_client.py

- We want all subclasses to pass any client code created for Container.
- The parameter c takes different types of subclasses (polymorphism)

```
from container import Container
from stack import Stack
from sack import Sack
def fill(c: Container) -> None:
    c.add(3)
    c.add(4)
    c.add(5)
    c.add(6)

def dele(c: Container) -> None:
    c.remove()
    c.remove()

s = Stack()
b = Sack()
fill(s)
print('s:'+str(s._storage))
fill(b)
print('b:'+str(b._storage))

dele(s)
print('s:'+str(s._storage))
dele(b)
print('b:'+str(b._storage))
```

Different result every time we run the code because **remove()** in sack (bag)

```
D:/csc148/lectures/
s:[3, 4, 5, 6]
b:[3, 4, 5, 6]
s:[3, 4]
b:[3, 5]
```

```
D:/csc148/lectures/weel
s:[3, 4, 5, 6]
b:[3, 4, 5, 6]
s:[3, 4]
b:[3, 6]
```

```
D:/csc148/lectures/we
s:[3, 4, 5, 6]
b:[3, 4, 5, 6]
s:[3, 4]
b:[3, 4]
```

Why Container?
Example #2

container_client.py

- We want to all subclasses to pass any client code created for Container.

```
def container_cycle(c: Container, i: int) -> None:
    """ Cycle i items through Container c.
    """
    for n in range(i):
        c.add(n)

    while not c.is_empty():
        print(c.remove())
```

container_cycle will work with Sack/Stack/Queue or any subclass that implements Container

Why Container? Example #2

container_client.py

- We want to all subclasses to pass any client code created for Container.

```
L = [Stack(), Sack(), Queue()]
for s in L:
    print("\nCycling through {}".format(s))
    container_cycle(s, 10)
```

container_cycle will work with Sack/Stack/Queue or any subclass that implements Container

How to generalize

1. Create super class Container
2. Include all methods signatures
3. rais NotImplementedError
4. Let subclasses inherit the super class
   - class Stack(Container)
   - class Sack(Container)

```
class Container:
subclasses are to be instantiated.
    """
    def __init__(self) -> None:
        raise NotImplementedError("Override this!")

    def add(self, obj: object) -> None:
        raise NotImplementedError("Override this!")

    def remove(self) -> object:
        raise NotImplementedError("Override this!")

    def is_empty(self) -> bool:
        raise NotImplementedError("Override this!")
```

How to generalize

If a subclass of a Container does not implement a method PyCharm will indicate that

```
3  from container import Container, EmptyContainerException
4  import random
5
6
7  class Sack(Container):
```

no particular order.

Class Sack must implement all abstract methods less... (Ctrl+F1)
This inspection detects when not all abstract properties/methods are defined in a subclass

Container

```
""" implement Container
"""
class EmptyContainerException(Exception):
    """
    Exceptions called when empty Container used inappropriately
    """
    pass

class Container:
    """ Container with add, remove, and is_empty methods.
    This is an abstract class that is not meant to be instantiated itself,
    but rather subclasses are to be instantiated.
    """
    def __init__(self) -> None:
        """
        Create a new Container self.
        """
        self._contents = None
        raise NotImplementedError("Override this!")
```

Container

```
def add(self, obj: object) -> None:
    """
    Add obj to Container self.
    """
    raise NotImplementedError("Override this!")

def remove(self) -> object:
    """
    Remove and return an object from Container self.
    Assume that Container self is empty.
    """
    raise NotImplementedError("Override this!")
```

Container

```
def is_empty(self) -> bool:
    """
    Return whether Container self is empty.
    """
    raise NotImplementedError("Override this!")
```

Where Can I find the code presented in class

- You can find the full code for Stack/Sack as list and Container in the course website under section **MWF2 (L0301)**
- with the following file names:
  - stackt.py
  - sack.py
  - container.py
  - container_simple_client.py
  - container_client.py

- Download them Try different things with them and practice
  - Do not be afraid of doing mistakes

Announcements

- Lab3 is posted
- A1 is due in 8 days
- Make use of office hours and pizza