

CSC148-Section:L0301/L0401

Week#3-Wednesday

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

Outline

- Generalize Stack/Sack/Queue into Container
- Exceptions
 - custom exceptions
 - try: except
- Testing your code

Generalize Stack, Sack, Queue as Container

- The methods (add/remove/is_empty) in Container super class

```
raise NotImplementedError("Override this!")
```

- To give developers the **freedom** to have **different implementations**: using **lists**, **dictionaries** in implementing **subclasses** (Stack/Sack/Queue)
- Container insures** that all **subclasses** will have a **common API between them**, so we can write **client code** that **works with any** stack, sack, or other... Containers

```
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!")
```

```
# suppose L is list[Container]
```

```
for c in L:
    for i in range(1000):
        c.add(i)
    while not c.is_empty():
        print(c.remove())
```

... so we'll make Stack, Sack subclasses of Container!



Exceptions-custom exceptions

- They are raised by **ERRORS** detected during execution
- If exceptions are not handled by your program, you may end up getting something like this

```
>>> 1/0
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ZeroDivisionError: division by zero
>>> L=[]
>>> L.pop()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
IndexError: pop from empty list
```

- In our implementation of Stack and Sack we want to raise **our own exception** if user try to remove from an empty Container

“**EmptyContainerException**”

What happens if we did not raise our own exception?

```
>>> from stack import *
>>> s = Stack()
>>> s.remove()
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    File "D:\csc148\lectures\week3\stack.py", li
        return self._contents.pop()
IndexError: pop from empty list
```

If we did not implement our own exception
we get IndexError exception

Creating our own exception class

1- Create a class and name it as you like

```
class EmptyContainerException(Exception):
```

```
    """
```

```
    Exceptions called when empty Container used  
    inappropriately
```

```
    """
```

```
pass
```

2- subclass the Python Exception class

3- we will ignore the implantation for now
as we just want to raise the exception and
leave handling the error to the Python
Exception class

Creating our own exception class

```
""" implement stack ADT
"""

from container import Container, EmptyContainerException
# from typing import Any

class Stack(Container):
    """ Last-in, first-out (LIFO) stack.
    """

    def __init__(self) -> None: ...

    def add(self, obj: object) -> None: ...

    def remove(self) -> object:
        """Remove and return top element of Stack self...."""
        if self.is_empty():
            raise EmptyContainerException
        else:
            return self._storage.pop()
```

4- import the **EmptyContainerException** class
In our implementation we placed this class in the same file of the Container class

5- raise the exception
In this case we want to raise it if a user tries to remove from empty Stack

Creating our own exception class

```
+  
>>> from stack import *  
>>> s = Stack()  
>>> s.remove()  
Traceback (most recent call last):  
  File "<input>", line 1, in <module>  
    File "D:\csc148\lectures\week3\stack.py", line 47  
        raise EmptyContainerException  
container.EmptyContainerException: Uh-oh, nothing t
```

this is name of the our exception we raised

Exceptions-try: except

- Try....except are used to **handle exceptions** and **prevent** your code from getting **terminated** by Python.
- The syntax is as follows:

try:

code that may cause error

except Exception **as** e:

print("error message to inform the user")

print(e) # show the error

Exceptions- try: Except--example: 1

```
def make_errors() -> None:
    try:
        1/0
        #print(int("hello"))
    except ZeroDivisionError as zd:
        print("Do not divide by Zero Please")
        print(zd)
    except ValueError as ve:
        print("Do not covert a string to int")
        print(ve)
    except Exception as e:
        print("General Exception")
        print(e)

    print("more code in this method")

make_errors()
print("my code continue running normally")
print(42)
```

output

```
C:\Users\AAA\AppData\Local\Programs\Python\Python38-32\python.exe
D:/csc148/lectures/week3/w3/try_exc.py
Do not divide by Zero Please
division by zero
more code in this method
my code continue running normally
42
```

Exceptions- try: Except--example: 2

```
# Experiment with exceptions changing what is commented out in the try block
class SpecialException(Exception):
    """class docstring here --- child of Exception"""
    pass
class ExtremeException(SpecialException):
    """ grandchild of Exception"""
    pass
```

Exceptions- try: Except--example: 2

```
if __name__ == '__main__':
    num = 1, denum = 0
    try:
        if (denum == 0):
            # raise SpecialException('I am a SpecialException')
            # raise Exception('I am an Exception')
            raise ExtremeException('I am an ExtremeException')
        else:
            print(num/denum)
    # block to run if SpecialException was raised
    # use the name se if one is detected
    except SpecialException as se:
        print(se)
        print('caught as SpecialException')
    except ExtremeException as ee:
        print(ee)
        print('caught as ExtremeException')
    except Exception as e:
        print(e)
        print('caught as Exception')
    print('I am outside try')
    print('my code did not stop due to exception')
```

Output

```
I am an ExtremeException
caught as SpecialException
I am outside try
my code did not stop due to exception
```

Order exceptions by more specific first
- ExtremeException should go above SpecialException

Testing your code

- You have been using docstring for testing as you develop
- Today we will use: Python unittest to **make sure** that a particular **implementation remains consistent** with your **ADT's** interface
- What is Python unittest?
 - A framework provided by Python that supports test automation
- How to use unittest?
 - See the slides next

How to use unittest? E.g. StackEmptyTestCase

```
"""unit tests for Stack
"""

import unittest
from stack import Stack

class StackEmptyTestCase(unittest.TestCase):
    """Test behaviour of an empty Stack."""
    def setUp(self):
        """Set up an empty stack."""
        self.stack = Stack()
    def tearDown(self):
        """Clean up."""
        self.stack = None
    def testIsEmpty(self):
        """Test is_empty() on empty Stack."""
        self.assertTrue(self.stack.is_empty(),
                        "is_empty returned False on empty")
    def testadd(self):
        """Test add to empty Stack."""

        self.stack.add("foo")
        self.assertEqual(self.stack.remove(), "foo")
```

1- import module **unittest** and the class you want test (in this example **Stack**)

2- subclass **unittest.TestCase**

3- override special methods
setUp()
tearDown()

4- begin each method that carries out a test with the string **test** then the method name.
e.g **testIsEmpty()** test the method **is_empty()** in the stack

5- use assert to check expected outcome
assertEqual/assertTrue/assertFalse

Another test case for stack

Tests a range of different values

```
class StackAllTestCase(unittest.TestCase):
    """Comprehensive tests of (non-empty) Stack."""

    def setUp(self):
        """Set up an empty stack."""
        self.stack = Stack()

    def tearDown(self):
        """Clean up."""
        self.stack = None

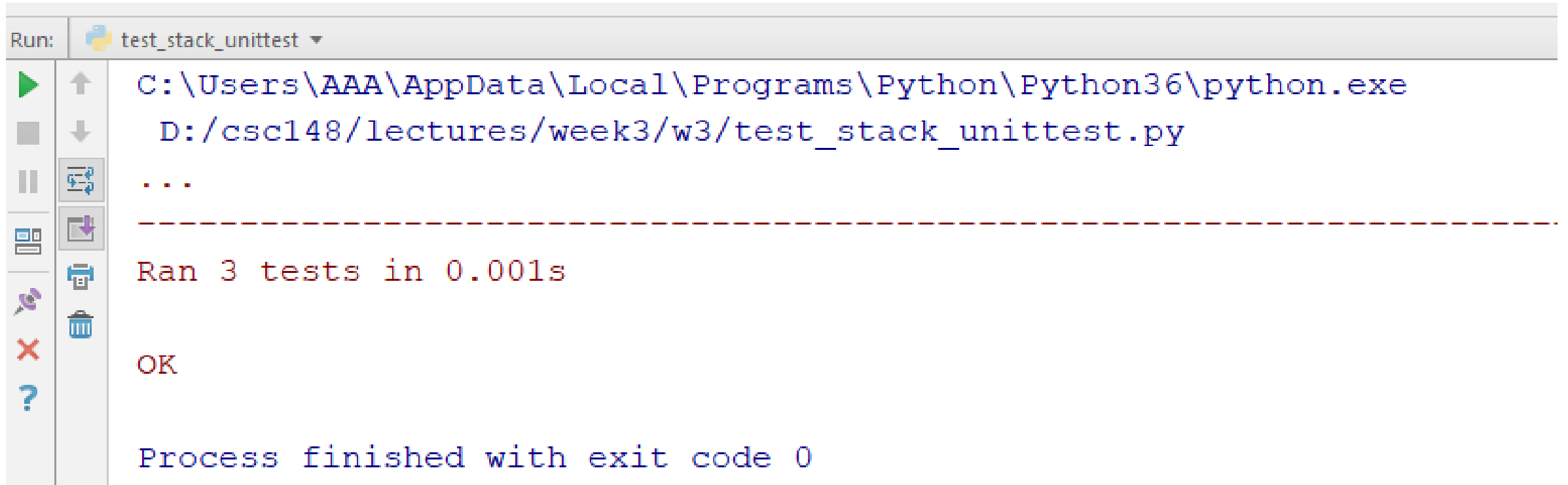
    def testAll(self):
        """Test adding and removing multiple elements."""

        for item in range(20):
            self.stack.add(item)
            self.assertTrue(not self.stack.is_empty(),
                            "is_empty() returned True on non-empty Stack!")

        expect = 19
        while not self.stack.is_empty():
            assert self.stack.remove() == expect, \
                ('Something wrong on top of the Stack! Expected ' +
                 str(expect) + '.')
            expect -= 1
```



Output of running unittests



The screenshot shows a Python IDE's Run console. The top bar indicates the command being executed is `test_stack_unittest`. The console output shows the execution of `python.exe` on the file `D:/csc148/lectures/week3/w3/test_stack_unittest.py`. The output consists of three dots, a dashed red line, the text `Ran 3 tests in 0.001s`, the text `OK`, and the message `Process finished with exit code 0`. On the left side of the console, there is a vertical toolbar with icons for running, stepping through, and other debugging actions.

```
Run: test_stack_unittest ▼  
C:\Users\AAA\AppData\Local\Programs\Python\Python36\python.exe  
D:/csc148/lectures/week3/w3/test_stack_unittest.py  
...  
-----  
Ran 3 tests in 0.001s  
OK  
Process finished with exit code 0
```


General remarks when using unittest?

- compose tests before and during implementation
- choosing test cases
 - since you can't test every input, try to think of representative cases:
 - smallest argument(s): 0, empty list or string, ...
 - boundary case: moving from 0 to 1, empty to non-empty, ...
 - “typical” case
- isolate units
 - test classes separately
 - test (related) methods separately

Where Can I find the code presented in class

- You can find the full code in the course website under section **MWF2 (L0301) and MWF3 (L0401)**
- with the following file names:
 - try_except_example.py
 - exceptions.py
 - test_stack_unittest.py
 - test_sack_unittest.py
- Download them Try different things with them and practice
 - Do not be afraid of doing mistakes

Announcements

- A1 due in less than 1 week
- Lab1 marks are posted but the paper_based is not ready yet
- Demo 1: example from last year will be posted soon.