Advanced Topics in Comp Sci Activity—The `Stack` Class

ASSIGNMENT OVERVIEW

In this assignment you'll create a Python implementation of the abstract data type "stack."

This assignment is worth 20 points and is due on the *crashwhite.polytechnic.org* server at 23:59:59 on the date given in class.

BACKGROUND

A "stack" is an ordered collection of items. Items can be "pushed" onto the end of the stack, and "popped" from that same end of the stack. The item at the end of the stack can be "peeked" at (identified without removing it from the stack), and the "size" of the stack can be identified at any time. Also, the stack can be identified as being "empty" (a size of 0).

A common analogy is that of placing books in a single pile on a desk. Placing a book on the desk is a "push" operation. Placing another book on top of that is a "push" operation. Placing one more is another "push", and now the size of the stack is 3. The statement "is_empty" is false at this point. You can "peek" at the stack to see the book on top, but none of the books underneath are visible. A single "pop" operation removes the top book from the stack, and two additional pop operations will return the stack to its empty state, where the "size" is 0 and "is_empty" is true.

Because of the way items are *pushed* and *popped* on a stack, it is sometimes called a *Last In-First Out* (LIFO) structure.

PROGRAM SPECIFICATION

Write a Python class `Stack` that implements this *abstract data type*. The `Stack` class will include:

a. a constructor which creates an empty stack
b. the `.push(item)` method to push an item onto the stack
c. the `.pop()` method to remove an item from the stack, and return that removed item as a result
d. the `.peek()` method which returns the value of the top item in the stack
e. the `.size()` method which returns the number of items in the stack
f. the `.is_empty()` method which returns `True` or `False`, depending on the state of the stack

DELIVERABLES

atds.py

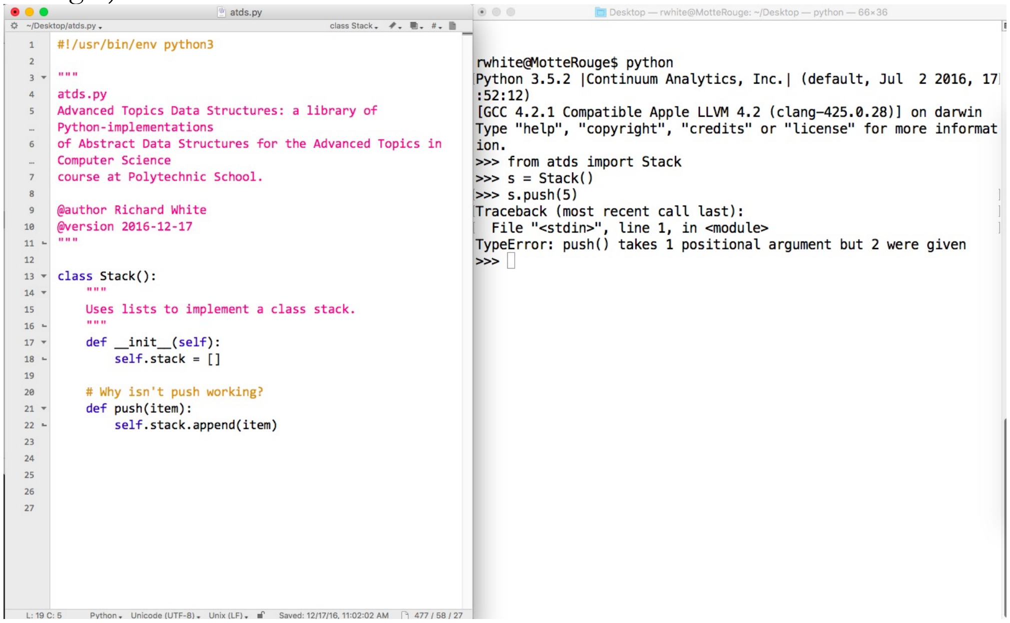
This single file ("atds" = "Advanced Topics Data Structures") will be used to collect a number of different implementations of abstract data types as we proceed through the course.

To submit your assignment for grading, copy your file to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline.

ASSIGNMENT NOTES

- Use Python's `list` data structure as a foundation for implementing the `Stack` class. You'll find that some of Python's `list` methods are very similar to the methods we're implementing for the `Stack` class.
- You have two ways of testing your development of the `Stack` class.

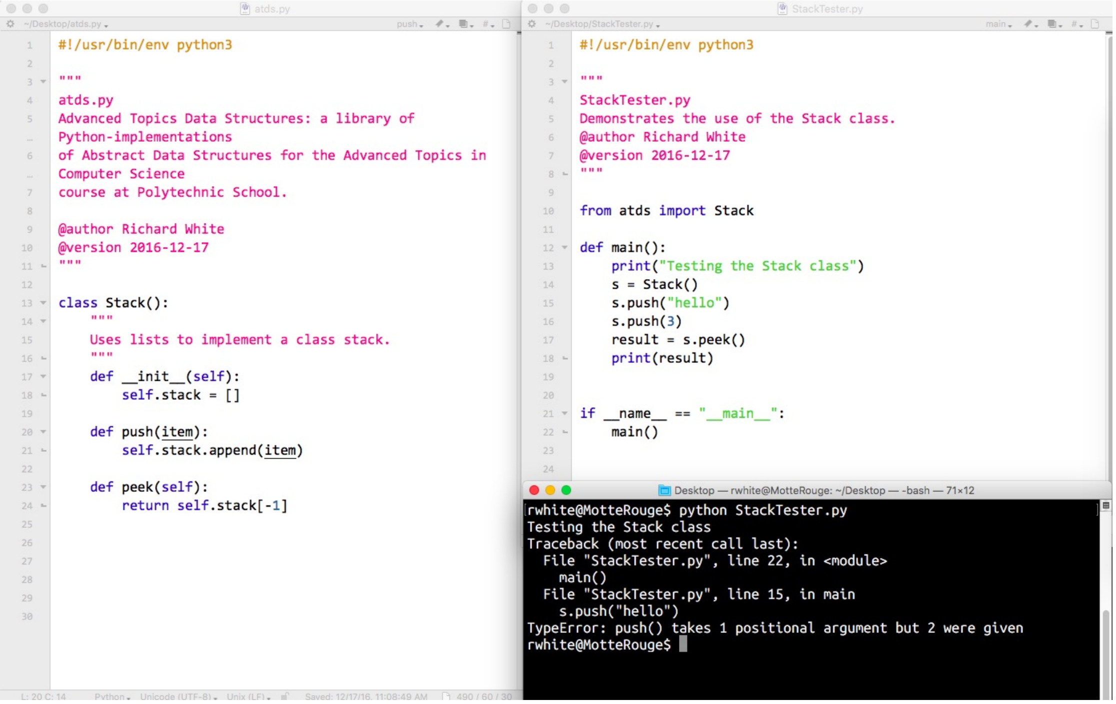
- Begin writing the class, and after adding each new feature, start up Python in interactive mode, and issue commands one at a time to interact with your `Stack` class. Here's a screenshot of me testing my own `Stack` class (being written on the left) in the Python interpreter (running on the right):



```python
#!/usr/bin/env python3

"""
atds.py
Advanced Topics Data Structures: a library of
Python-implementations
of Abstract Data Structures for the Advanced Topics in
Computer Science
course at Polytechnic School.

@author Richard White
@version 2016-12-17
"""

class Stack():
    """
    Uses lists to implement a class stack.
    """
    def __init__(self):
        self.stack = []

    # Why isn't push working?
    def push(item):
        self.stack.append(item)
```

```
rwhite@MotteRouge$ python
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17
:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more informat
ion.
>>> from atds import Stack
>>> s = Stack()
>>> s.push(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: push() takes 1 positional argument but 2 were given
>>> 
```

- Write a full tester program of your own—a separate Python main program—that imports your `Stack` class, tries to construct a `Stack` object, and tries to interact with it. In this case, you typically have three windows open: one for editing the class, one for editing the tester, and a Terminal for running the tester. Here's a screenshot of me doing some debugging while working with the `atds.py` program and my own `Stack` tester:



```python
#!/usr/bin/env python3

"""
atds.py
Advanced Topics Data Structures: a library of
Python-implementations
of Abstract Data Structures for the Advanced Topics in
Computer Science
course at Polytechnic School.

@author Richard White
@version 2016-12-17
"""

class Stack():
    """
    Uses lists to implement a class stack.
    """
    def __init__(self):
        self.stack = []

    def push(item):
        self.stack.append(item)

    def peek(self):
        return self.stack[-1]
```

```python
#!/usr/bin/env python3

"""
StackTester.py
Demonstrates the use of the Stack class.
@author Richard White
@version 2016-12-17
"""

from atds import Stack

def main():
    print("Testing the Stack class")
    s = Stack()
    s.push("hello")
    s.push(3)
    result = s.peek()
    print(result)


if __name__ == "__main__":
    main()
```

```
rwhite@MotteRouge$ python StackTester.py
Testing the Stack class
Traceback (most recent call last):
  File "StackTester.py", line 22, in <module>
    main()
  File "StackTester.py", line 15, in main
    s.push("hello")
TypeError: push() takes 1 positional argument but 2 were given
rwhite@MotteRouge$ 
```

- Ultimately, once you have upload your completed `atds.py` file, it will be analyzed with a tester similar to the one shown at the end of this document.

GETTING STARTED

1. With paper and pencil, and perhaps in collaboration with a partner, identify the methods that you'll be writing as part of this assignment.

2. Create an `atds.py` file that will include your `Stack` class

3. Use Python in interactive mode to import the `atds` package, and then try to construct a `Stack` object. As you add methods to the `Stack` class, test them in interactive mode.

4. As the development of the Stack class proceeds, consider creating a full `stack_tester.py`, a Python main program that will run your `Stack` objects through a series of tests. It's more efficient than having to interactively create a Stack object and manipulate it every time you add a new feature.

5. When your program is completed (but before the deadline), copy `atds.py` to the server as indicated above.

EXTENSIONS

1. Parentheses can be used to organize and prioritize operations in complex mathematical expressions, and it's important for those parentheses to be "balanced"—for every opening parenthesis "(" there needs to be a closing parenthesis ")". Write a program `paren_checker` with a boolean function `is_valid(expr)` that uses a stack to check parentheses in an expression. The function should return `True` if the parentheses are balanced and `False` if they are not.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. There were two testing methods described in this assignment: interactive testing using the Python interpreter, and writing a formal tester to run on your code as it's being developed. What are the advantages and disadvantages of each system?

2. In the `stack_tester.py` code listed below, the `try-except` feature in Python is used. What does this feature do?

3. What are the advantage of using `try-except` in a program? What are the disadvantages?

REFERENCES

```
#!/usr/bin/env python3

"""
stack_tester.py
Demonstrates the use of the Stack class.
@author Richard White
@version 2016-12-17
"""

from atds import Stack

def main():
    print("Testing the Stack class")
    testsPassed = 0
    try:
        s = Stack()
        testsPassed += 1
        print("Test passed: stack created")
    except:
        print("Test failed: couldn't initialize stack")

    try:
        s.push("hello")
        s.push(3)
        testsPassed += 1
        print("Test passed: items pushed")
    except:
        print("Test failed: couldn't push onto stack")

    try:
        result = s.peek()
        if (result == 3):
            testsPassed += 1
            print("Test passed: peeked at item")
        else:
            print("Test failed: incorrect peek value")
    except:
        print("Test failed: couldn't peek at stack")

    try:
        result = s.pop()
        if (result == 3):
            testsPassed += 1
            print("Test passed: item popped")
        else:
            print("Test failed: incorrect pop result")
    except:
        print("Test failed: couldn't pop")

    try:
        result = s.is_empty()
        if (not result):
            testsPassed += 1
            print("Test passed: is_empty returned correct result")
        else:
            print("Test failed: stack has items, but indicated empty")
    except:
        print("Test failed: is_empty() method unavailable")

    try:
        result = s.size()
        if (result == 1):
            testsPassed += 1
            print("Test passed: correct size returned")
        else:
            print("Test failed: incorrect size returned")
    except:
        print("Test failed: size() method unavailable")

    try:
        s.pop()
    except:
        pass

    try:
        result = s.is_empty()
        if (result):
            testsPassed += 1
            print("Test passed: .is_empty() correctly indicating empty status")
        else:
            print("Test failed: stack failed to indicate empty status")
    except:
        print("Test failed: is_empty() unavailable")

    print(str(testsPassed) + "/7 tests passed")

if __name__ == "__main__":
    main()
```