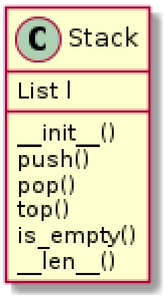
Lab 9.1 (Deadline Monday 21 March 23:59)

- Upload your code to Einstein to have it verified.

Stack

- In `stack_091.py` define a `Stack` class to model the stack abstract data type as follows:



- Each box consists of three compartments: class name, data attributes, methods.
- Read the notes on stack methods to determine their required behaviour.
- When your class is correctly implemented, running the following program should produce the given output.

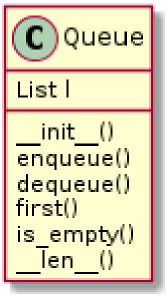
```
from stack_091 import Stack

def main():

    s = Stack()

    print(len(s))
    s.push(1)
    print(s.top())
    print(s.is_empty())
    print(s.pop())
    print(s.is_empty())
    try:
        print(s.pop())
    except IndexError:
        print('Error')
    try:
        print(s.top())
    except IndexError:
        print('Error')
    s.push(1)
    s.push(2)
    s.push(3)
    print(len(s))
    print(s.pop())
    print(s.pop())
    print(s.pop())
    print(s.is_empty())

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

```
0
1
```

```
False
1
True
Error
Error
3
3
2
1
True
```

Queue

- In *queue_091.py* define a `Queue` class to model the queue abstract data type as follows:



- Each box consists of three compartments: class name, data attributes, methods.
- Read the notes on queue methods to determine their required behaviour.
- When your class is correctly implemented, running the following program should produce the given output.

```python
from queue_091 import Queue

def main():

    q = Queue()

    print(len(q))
    q.enqueue(1)
    print(q.first())
    print(q.is_empty())
    print(q.dequeue())
    print(q.is_empty())
    try:
        print(q.dequeue())
    except IndexError:
        print('Error')
    try:
        print(q.first())
    except IndexError:
        print('Error')
    q.enqueue('cat')
    q.enqueue('dog')
    q.enqueue('fish')
    print(len(q))
    print(q.dequeue())
    print(q.dequeue())
    print(q.dequeue())
    print(q.is_empty())
```

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

```
0
1
False
1
True
Error
Error
3
cat
dog
fish
True
```

Brackets

- In `brackets_091.py` define a function called `matcher()` that takes a single string parameter.
- The `matcher()` function checks that all left and right brackets in the supplied string match.
- `matcher()` should return `True` if brackets match and `False` otherwise.
- Brackets that need to be matched are `(){}[]`.
- For example:

```
from brackets_091 import matcher
import sys

tests = ['()',
'(())',
'(({}))',
'(())(){}{(([]))}',
'(()',
'(()){()]',
')(()){([()])}']

def main():

    for test in tests:
        print(matcher(test.strip()))

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

```
True
True
True
True
False
False
False
```

- Hints:

  1. Make good use of a stack in solving this problem and include a copy of your stack class definition from `stack_091.py` in `brackets_091.py`.

2. If lefties are `({[` then righties are `)}]`.
3. Push lefties and pop on meeting a righty.

RPN calculator

- Reverse Polish Notation (RPN) is a mathematical notation in which every operator follows all of its operands. Examples:
    - `2 + 3` is expressed as `2 3 +`
    - `2 + sqrt(3)` is expressed as `2 3 r +`
    - `1 + 2 * 3` is expressed as `1 2 3 * +`
    - `5 * -2` is expressed as `5 2 n *`
    - `sqrt(-(2*(1-(2+3))))` is expressed as `2 1 2 3 + - * n r`
- In *`rpn_091.py`* define a function called `calculator()` that takes a single parameter `line` (an RPN expression read from `stdin`). The `calculator()` function computes the value of the RPN expression. Should the RPN expression be invalid then the `calculator()` function raises an `IndexError` exception. For example:

```
from rpn_091 import calculator
import sys

tests = ['5',
'8.5 2 /',
'2 3 +',
'2 3 r +',
'1 2 3 * +',
'5 2 n *',
'1 2 3 + -',
'2 1 2 3 + - *',
'2 1 2 3 + - * n',
'2 1 2 3 + - * n r',
'6 +',
'9 r']

def main():

    for test in tests:
        try:
            a = calculator(test.strip())
            print('{:.2f}'.format(a))
        except IndexError:
            print('Invalid RPN expression')

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

```
5.00
4.25
5.00
3.73
7.00
-10.00
-4.00
-8.00
8.00
2.83
Invalid RPN expression
3.00
```

- Hints:

  1. Convert all user-supplied numbers to floats.
  2. In solving this problem, again, it might help to make use of a stack.
  3. You might find two dictionaries useful: `binops` maps from each of `+-*/` to a corresponding function while `uniops` maps from each of `nr` to a corresponding function.
  4. When you encounter a number in an RPN expression push it onto the stack. If you encounter an operator pop its arguments from the stack (one or two), apply the operator to the popped argument(s) and push the result onto the stack.
  5. If after processing an RPN expression you are left with a single number on the stack it is the answer (congratulations!) and your `calculator()` function should return it. Otherwise `calculator()` should raise an `IndexError` exception.
- Here is some code to inspire you:

```
from math import sqrt

binops = {'+': float.__add__,
          '-': float.__sub__,
          '*': float.__mul__,
          '/': float.__truediv__}

uniops = {'n': float.__neg__,
          'r': sqrt}

# To add 3 and 5 we can do something like this...
print(binops['+'](3.0, 5.0))

# To calcualte the square root of 9 we can do something like this...
print(uniops['r'](9.0))
```

```
8.0
3.0
```