TD - Piles et files

I - Création d'une pile

```
class Pile:
    _pile = []

    def __init__(self):
        self._pile = []

    def stack(self, element):
        """Add element at the end of the pile"""
        self._pile.append(element)

    def top(self):
        """Return the last element of the pile without modifying it"""
        return self._pile[-1:]

    def unstack(self):
        """Return the last element of the pile, and remove it from the pile."""
        return self._pile.pop()

    def lenght(self):
        """Return lenght of the pile."""
        return len(self._pile)

    def isEmpty(self):
        """Return true if pile is empty."""
        return len(self._pile) == 0

    def toString(self):
        """Return pile in a str type"""
        return str(self._pile)
```

II - Vérification du parenthésage d'une expression

```
from exercice_1 import Pile


def parenthesisTest(chaine):
    pile = Pile()
    for char in chaine:
        if char == '(':
            pile.stack(1)
        elif char == ')':
            if not pile.isEmpty():
                pile.unstack()
            else:
```

```
                return False
    if pile.isEmpty():v
        return True
    else:
        return False
```

III - Décimal / Binaire

Question III.1

```
def div2(x):
    return (x//2, x % 2)
```

Question III.2

```
def stackBinary(decimalNumber):
    pile = Pile()
    while decimalNumber >= 1:
        a = div2(decimalNumber)
        decimalNumber = a[0]
        pile.stack(a[1])
    return pile
```

Question III.3

```
print(stackBinary(256).toString())
```

Question III.4

```
def displayIt(pile):
    a, i = '', 0
    iMax = pile.lenght() % 4
    while not pile.isEmpty():
        if (i == iMax):
            a += ' '
            iMax, i = 4, 0
        a += str(pile.unstack())
        i += 1
    return a

if __name__ == '__main__':
    print(displayIt(stackBinary(525)))
```

Résultats:

```
10 0000 1101
```

IV - Notation polonaise inverse

Question IV.1

```
def isOperator(char):
    op = [
        '+',
        '-'
        '/',
        '*'
    ]
    return char in op
```

Question IV.2

```
def calculOp(a, b, op):
    if op == '+':
        return a + b
    elif op == '-':
        return a - b
    elif op == '/':
        return a / b
    elif op == '*':
        return a * b
    else:
        raise '{} ne fait pas partie de la liste des opérateurs'.format(op)
```

Question IV.3

```
def listWords(chaine):
    return [a for a in chaine.replace(' ', '')]
```

Question IV.4

```
from exercice_1 import Pile

def polonaise(chaine):
    pile = Pile()
```

```
    for char in listWords(chaine):
        if isOperator(char):
            if pile.lenght() >= 2:
                a = pile.unstack()
                b = pile.unstack()
                pile.stack(calculOp(a, b, char))
            else:
                raise 'Error ! (oups)'
        elif char.isnumeric():
            pile.stack(float(char))
        else:
            raise 'Error, a non numeric character has been found ({})'.format(
                char)
    if pile.lenght() == 1:
        return pile.unstack()
    else:
        raise 'Error (oups) !'


class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'


def test(chaine, exceptedResult):
    try:
        a = polonaise(chaine)
        if a == exceptedResult:
            color = bcolors.OKGREEN
        else:
            color = bcolors.FAIL
        print(color + str(int(a)) + bcolors.ENDC)
    except:
        print(bcolors.FAIL + 'Error' + bcolors.ENDC)


if __name__ == '__main__':
    test('34+', 7)
    test('723+*', 35)
    test('23+7*', 35)
```

V - Création d'une file

```
class File:
    _file = []
```

```
    def __init__(self):
        self._file = []

    def push(self, element):
        """Add element at the end of the file"""
        self._file.append(element)

    def top(self):
        """Return the first element of the file without modifying it"""
        return self._file[:1]

    def pop(self):
        """Return the first element of the file, and remove it from the file."""
        a = self._file[:1]
        self._file = self._file[1:]
        return a

    def lenght(self):
        """Return lenght of the file."""
        return len(self._file)

    def isEmpty(self):
        """Return true if file is empty."""
        return len(self._file) == 0

    def toString(self):
        """Return file in a str type"""
        return str(self._file)
```