Functional Languages 3rd practice

1. Redefine the logical operator "and" (`&&`) using pattern matching.

```
and' True True
not (and' True False)
```

2. Redefine the logical operator "or" (`||`) using pattern matching.

```
or' True True
or' True False
or' False True
not (or' False False)
```

3. Redefine the logical operator "xor" using pattern matching.

```
xor True False
xor False True
not (xor True True)
```

4. Define base-2 addition of two bits using pattern matching. Return the sum and the carry over.

```
add2 0 0 == (0,0)
add2 1 0 == (1,0)
add2 1 1 == (0,1)
```

5. Check whether a pair of parentheses or brackets matches using pattern matching.

```
paren '(' ')'
paren '[' ']'
paren '{' '}'
not (paren '(' ']')
not (paren '(' '(')
not (paren '[' 'a')
```

6. Define a calculator for the four basic arithmetic operations. Use pattern matching to inspect the operator.

```
calc (2, '+', 4) == 6
calc (2, '-', 4) == (-2)
calc (2, '*', 4) == 8
calc (5, '/', 2) == 2
```

7. Check that the parameter is a space character using pattern matching.

```
isSpace ' '
not (isSpace '_')
not (isSpace 'a')
```