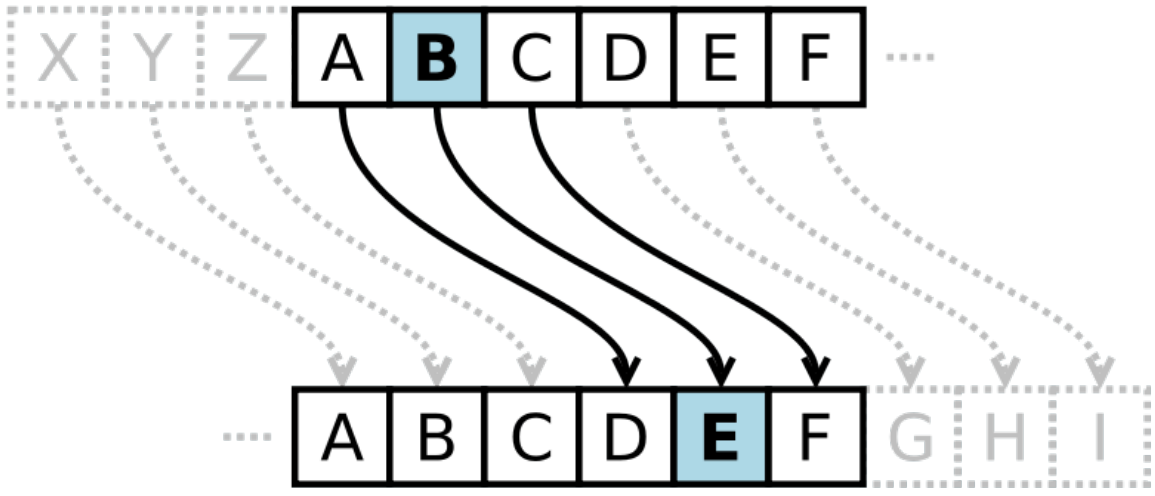
COMP105 Lecture 10

The Caesar Cipher

The Caesar Cipher



The **Caesar cipher**:

- Shift every letter in the string forward by 3
- Wrap around when you get to 'z'

The Caesar Cipher

Examples:

- `"abcde"` $\rightarrow$ `"defgh"`
- `"vwxyz"` $\rightarrow$ `"yzabc"`

There is no particular reason to shift by 3

- You can shift by any number between 0 and 25

For example, shifting by 13 (ROT13):

- `"abcde"` $\rightarrow$ `"nopqr"`
- `"vwxyz"` $\rightarrow$ `"ijklm"`

The tasks

We will build **three** functions

1. A function to encode strings: `caesar_enc string offset`

2. A function to decode strings: `caesar_dec string offset`

3. A function to crack strings: `caesar_crack string`

All code from today's lecture is available on the website

Working with characters

The `Data.Char` package has some useful functions for working with characters

The **ord** function takes a character and turns it into an integer

```
ghci> ord 'c'
99
```

The **chr** function does the opposite conversion

```
ghci> chr 98
'b'
```

Working with characters

These functions convert lower case characters to numbers between 0 and 25

```
import Data.Char

char2int c = ord c - ord 'a'

int2char i = chr (i + ord 'a')
```

```
ghci> char2int 'c'
2

ghci> int2char 25
'z'
```

Doing a Caesar shift

```
shift c offset =
    let
        converted = char2int c
    in
        int2char ((converted + offset) `mod` 26)
```

```
ghci> shift 'a' 3
'd'
```

The `mod` implements the wrap-around

Doing a Caesar shift

But what if the string contains non-lower case letters?

- Let's leave them unchanged

```
shift c offset =
    let
        converted = char2int c
        is_lower  = converted >= 0 && converted < 26
    in
        if is_lower
        then int2char ((converted + offset) `mod` 26)
        else c
```

```
ghci> shift ' ' 3
' '
```

Encoding a string

We want to encode every character in the input string

- Use recursion

```
caesar_enc [] _ = []
caesar_enc (x:xs) offset = shift x offset
                         : caesar_enc xs offset
```

```
ghci> caesar_enc "hello there" 3
"khoor wkhuh"
```

Decoding a string

To decode, we just make the offset negative, so the string is unshifted

- We can use the same recursive algorithm

```
caesar_dec (x:xs) offset = shift x (-offset)
                         : caesar_dec xs offset
```

```
ghci> caesar_dec "khoor wkhuh" 3
"hello there"
```