

# Lecture 12

**Topics:** *Chapter 5. Computing with strings*

5.4 String representation and message encoding

5.5 String methods

5.6 Lists have methods too

5.7 From encoding to encryption

## 5.4 String representation and message encoding

**How the strings are stored in a computer?**

## 5.4 String representation and message encoding

### How the strings are stored in a computer?

each character is translated into a number, and the entire string is stored as a sequence of (binary) numbers in computer memory

## 5.4 String representation and message encoding

Computer systems today use industry standard encodings of characters that is understandable by all kinds of computers.

## 5.4 String representation and message encoding

Computer systems today use industry standard encodings of characters that is understandable by all kinds of computers.

One of these standards is **ASCII** (**American Standard Code for Information Interchange**).

**ASCII** uses numbers through 0 to 127 to represent characters typically found on an (American) computer keyboard, as well as certain special values known as *control codes*.

## 5.4 String representation and message encoding

Computer systems today use industry standard encodings of characters that is understandable by all kinds of computers.

One of these standards is [ASCII](#) ([American Standard Code for Information Interchange](#)).

[ASCII](#) uses numbers through 0 to 127 to represent characters typically found on an (American) computer keyboard, as well as certain special values known as *control codes*.

For example,

[A-Z](#) are represented by values 65-90

[a-z](#) are represented by values 97-122

[0-9](#) are represented by values 48-57

## 5.4 String representation and message encoding

Computer systems today use industry standard encodings of characters that is understandable by all kinds of computers.

One of these standards is [ASCII](#) ([American Standard Code for Information Interchange](#)).

In Python:

To get an ASCII code of a character use command  
`ord(<character>)`

'ord' stands for 'ordinal'

Command `chr` goes the other direction.

## 5.4 String representation and message encoding

### ASCII

American Standard Code for Information Interchange

In Python interpreter's interactive window:

```
>>> ord('f')  
102
```

```
>>> ord('5')  
53
```

```
>>> chr(80)  
'P'
```

```
>>> chr(90)  
'Z'
```



## 5.4 String representation and message encoding

### ASCII

American Standard Code for Information Interchange

**Example:** Let's write a program that will encode our message by the ASCII code.

Here is the *design/algorithm*:

take the message to encode from the user

for each character in the message

    print the ASCII code of the character (same line, separated by space)

## 5.4 String representation and message encoding

ASCII

American Standard Code for Information Interchange

Should we write a decoding program?

## 5.4 String representation and message encoding

ASCII

American Standard Code for Information Interchange

Should we write a decoding program?

Yes, but let's take a look at functions that work on lists..... first

## 5.5 String methods

On page 148 **Table 5.2** we are given a list of string methods:

Function	Meaning
<code>s.capitalize()</code>	Copy of <code>s</code> with only the first character capitalized.
<code>s.center(width)</code>	Copy of <code>s</code> centered in a field of given width.
<code>s.count(sub)</code>	Count the number of occurrences of <code>sub</code> in <code>s</code> .
<code>s.find(sub)</code>	Find the first position where <code>sub</code> occurs in <code>s</code> .
<code>s.join(list)</code>	Concatenate <code>list</code> into a string, using <code>s</code> as separator.
<code>s.ljust(width)</code>	Like <code>center</code> , but <code>s</code> is left-justified.
<code>s.lower()</code>	Copy of <code>s</code> in all lowercase characters.
<code>s.lstrip()</code>	Copy of <code>s</code> with leading white space removed.
<code>s.replace(oldsub, newsub)</code>	Replace all occurrences of <code>oldsub</code> in <code>s</code> with <code>newsub</code> .
<code>s.rfind(sub)</code>	Like <code>find</code> , but returns the rightmost position.
<code>s.rjust(width)</code>	Like <code>center</code> , but <code>s</code> is right-justified.
<code>s.rstrip()</code>	Copy of <code>s</code> with trailing white space removed.
<code>s.split()</code>	Split <code>s</code> into a list of substrings (see text).
<code>s.title()</code>	Copy of <code>s</code> with first character of each word capitalized.
<code>s.upper()</code>	Copy of <code>s</code> with all characters converted to uppercase.

Table 5.2: Some string methods

## 5.5 String methods

On page 148 **Table 5.2** we are given a list of string methods:

Function	Meaning
<code>s.capitalize()</code>	Copy of <code>s</code> with only the first character capitalized.
<code>s.center(width)</code>	Copy of <code>s</code> centered in a field of given width.
<code>s.count(sub)</code>	Count the number of occurrences of <code>sub</code> in <code>s</code> .
<code>s.find(sub)</code>	Find the first position where <code>sub</code> occurs in <code>s</code> .
<code>s.join(list)</code>	Concatenate <code>list</code> into a string, using <code>s</code> as separator.
<code>s.ljust(width)</code>	Like <code>center</code> , but <code>s</code> is left-justified.
<code>s.lower()</code>	Copy of <code>s</code> in all lowercase characters.
<code>s.lstrip()</code>	Copy of <code>s</code> with leading white space removed.
<code>s.replace(oldsub, newsub)</code>	Replace all occurrences of <code>oldsub</code> in <code>s</code> with <code>newsub</code> .
<code>s.rfind(sub)</code>	Like <code>find</code> , but returns the rightmost position.
<code>s.rjust(width)</code>	Like <code>center</code> , but <code>s</code> is right-justified.
<code>s.rstrip()</code>	Copy of <code>s</code> with trailing white space removed.
<code>s.split()</code>	Split <code>s</code> into a list of substrings (see text).
<code>s.title()</code>	Copy of <code>s</code> with first character of each word capitalized.
<code>s.upper()</code>	Copy of <code>s</code> with all characters converted to uppercase.

Table 5.2: Some string methods

## 5.5 String methods

**Example:** a decoder from ASCII

*Design / Algorithm:*

Input: the sequence of numbers separated by space, as string

```
message = ""
```

```
for each number in the sequence:
```

```
    convert it to the corresponding Unicode character
```

```
    add the character to the end of the message
```

Output: decoded string

## 5.6 Lists have methods, too

Here are few more methods for Lists:

<code>r.append(x)</code>	appends the element to the end of the list
<code>r.count(x)</code>	returns the number of <code>x</code> 's occurrences
<code>r.index(x)</code>	returns smallest index <code>k</code> such that <code>r[k] == x</code>
<code>r.insert(i, x)</code>	inserts element <code>x</code> into <code>i</code> <sup>th</sup> place
<code>r.pop([i])</code>	returns the <code>i</code> <sup>th</sup> element of the list and removes it from the list
<code>r.remove(x)</code>	removes the first occurrence of element <code>x</code> in the list
<code>r.reverse()</code>	reverses the elements/items of <code>r</code> in place
<code>r.sort()</code>	sorts the elements/items of <code>r</code> in place (numerical, or alphabetical order, or ... )

*Note that majority of these methods are modifying the original list*

## 5.6 Lists have methods, too

Here are few more methods for Lists:

<code>r.append(x)</code>	appends the element to the end of the list
<code>r.count(x)</code>	returns the number of <code>x</code> 's occurrences
<code>r.index(x)</code>	returns smallest index <code>k</code> such that <code>r[k] == x</code>
<code>r.insert(i, x)</code>	inserts element <code>x</code> into <code>i</code> <sup>th</sup> place
<code>r.pop([i])</code>	returns the <code>i</code> <sup>th</sup> element of the list and removes it from the list
<code>r.remove(x)</code>	removes the first occurrence of element <code>x</code> in the list
<code>r.reverse()</code>	reverses the elements/items of <code>r</code> in place
<code>r.sort()</code>	sorts the elements/items of <code>r</code> in place (numerical, or alphabetical order, or ... )

*Note that majority of these methods are modifying the original list*



## 5.6 Lists have methods, too

**Example:** Let's write a program that given the date in the format `mm/dd/yyyy` will be displaying it in the form `month's name, year`

Example of input:           01/20/1979

The output produced:    January 20, 1979

*Design / algorithm:*

get the date (format `mm/dd/yyyy`, as string)

split the date into three strings (*month*, *day*, *year*)

find the month in the list of months' names (by index)

output the date in new format

## 5.6 Lists have methods, too

**Advanced Example:** given a phrase (no punctuation symbols, only letters and spaces) sort the letters alphabetically, keeping the spaces in place

Example of input: Mary likes ice cream

The output: Maac ceeei ikl mrrsy

*Design / algorithm:*

get the input (string *inp*)

split *inp* using space as separator (list *inpL*)

split *inp* into list of letters, ignoring spaces, then sort

create new list and fill it correspondingly with *inpL* (*result*)

join the list using " " as separator (*resultingString*)

## 5.7 From encoding to encryption

*Encoding* using industry-standard mappings of characters into numbers is mainly used for storage.

*Encryption* is used for keeping information secret. It uses the encoding which does not follow simple industry-standard mappings of characters into numbers.

## 5.7 From encoding to encryption

*Encoding* using industry-standard mappings of characters into numbers is mainly used for storage.

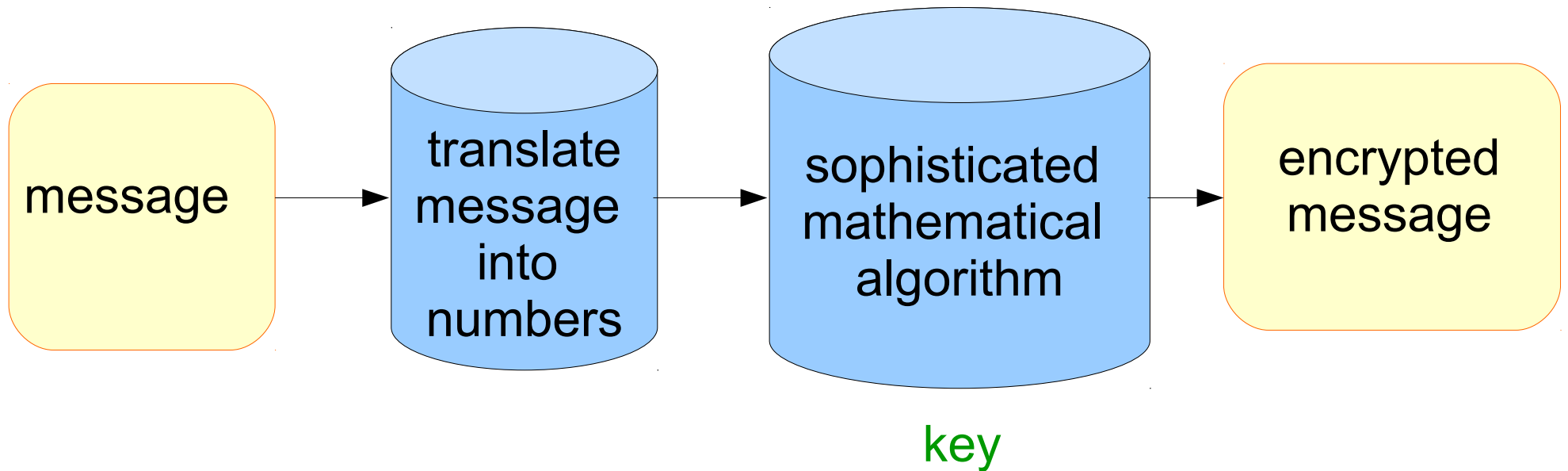
*Encryption* is used for keeping information secret. It uses the encoding which does not follow simple industry-standard mappings of characters into numbers.

The study of encryption methods is a sub-field of mathematics and computer science, called *cryptology*.

The original message is called *plaintext*, the resulting encrypted message is called *ciphertext*.

## 5.7 From encoding to encryption

Modern approaches to encryption:



## 5.7 From encoding to encryption

### Caesar cipher

See programming exercise 7 on page 172

The cipher is based on the idea of shifting each letter in the *plaintext message* a fixed number (called the *key*) of positions in the alphabet.

**Example:** assume key = 2  
Then word **Sourpuss** → **Uqwtrwuu**  
to decode: shift 2 letters left

## 5.7 From encoding to encryption

### Caesar cipher

See programming exercise 7 on page 172

The cipher is based on the idea of shifting each letter in the *plaintext message* a fixed number (called the *key*) of positions in the alphabet.

**Example:** assume key = 2  
Then word **Sourpluss** → **Uqwtrwuu**  
to decode: shift 2 letters left

Use ASCII codes      `chr(ord(ch) + key)`

## 5.7 From encoding to encryption

### Caesar cipher

*Design / algorithm:*

get the *key* value from the user

get the *message* to translate from the user

for char in message:

    newChar = chr(ord(char)+key)

    append the code to the resulting encrypted message

display the encrypted message



## 5.7 From encoding to encryption

### Caesar cipher

We designed and implemented the encryption algorithm.

If you want, design and implement the decryption algorithm!