Welcome to CS 1

• Lecture will start at 10:40 (let's wait for everyone).
• If you have any question, please ask in the chat.
• Zoom link will be the same for all CS lectures in Q3.
• Please note that lecture will be recorded.
5: Cryptography and Functions
Today

• Homework #2
• Introduction to Cryptography
• Functions in Python
• Homework #3
• “Modern” cryptographic techniques
Homework #2
# I changed the line below to introduce another way
# The result is equivalent to what was given before
# This method is called list comprehension

numbers = [ int(x) for x in input("Write some numbers: ").split() ]

max_value = -100000  # to represent minus infinity
max_index = -1

for k in range(len(numbers)):  # You can change this line
    if numbers[k] > max_value:
        max_value = numbers[k]
        max_index = k

print("max_value =", max_value)
print("max_index =", max_index)
s = input("Input a string:")
a = list(s.encode("ascii"))    # You may remove this line
for i in range(len(a)):
    # You can change this line
    if 97 <= a[i] and a[i] <= 122:
        print( chr( a[i] ) ) # or print(s[i])

# In Python, you can compare directly string
# Comparison is based on the lexicographic (i.e. alphabetic) order
s = input("Input a string:")
for c in s:
    if 'a' <= c <= 'z':
        print(c)
Cryptography
Encode the message such that its contents will be hidden for the outside observer.

Can be used for stored data as well, rather than for transmitted data.

Model of encrypted communication
Caesar Cipher

• Invented and used by Gaius Julius Caesar (100BC-44BC)
• Algorithm (k-letter shift in the alphabet)
  • Each letter is replaced by the k-th letter of the alphabet, which follows it.

Ex: alright!

\[
\begin{array}{cccccccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} & \text{h} & \ldots & \text{w} & \text{x} & \text{y} & \text{z} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
\text{d} & \text{e} & \text{f} & \text{g} & \text{h} & \text{i} & \text{j} & \text{k} & \ldots & \text{z} & \text{a} & \text{b} & \text{c}
\end{array}
\]

douljkw!
Cryptography

Encryption algorithm (cipher)
An algorithm for confidential communication.

Encryption
Creates a ciphertext from the plaintext (message) using the key.

Decryption
Restores the original plaintext from the ciphertext using the key.

Encryption
Plaintext
alright!

Decryption
Ciphertext
douljkw!
Functions
Cryptographic Functions

• **Encryption function**
  \[
  \text{enc}_{\text{caesar}}(\text{key } k, \text{plaintext } m)
  \]
  \[\rightarrow \text{the ciphertext } c \text{ obtained by the } k\text{-letter shift}\]

• **Decryption function**
  \[
  \text{dec}_{\text{caesar}}(\text{key } k, \text{ciphertext } c)
  \]
  \[\rightarrow \text{the plaintext } m \text{ obtained by the inverse } k\text{-letter shift}\]

• **Ex:**  \[\text{enc}(3, "Good") = "Grrg" \quad \text{dec}(3, "Grrg") = "Good"\]
Implement multiplication with ±1 and loop only

```
mult.py
x = int(input("x = "))
y = int(input("y = "))
product = 0
while y > 0:
    product = product + x
    y -= 1
print("x*y -> ", product)
```

```
add.py
x = int(input("x = "))
y = int(input("y = "))
res = x
while y>0:
    res += 1
    y -= 1
print("x+y=", res)
```

- Cheating using + ?
- Ok since we implemented add using only basic elements of comp.
- New element of computation: +
Function

Implement multiplication with ±1 and loop only

```
mult.py
x = int(input("x = "))
y = int(input("y = "))
product = 0
while y > 0:
    product = add(product, x)
y -= 1
print("x*y -> ", product)
```

```
add.py

def add(x, y):
    res = x
    while y>0:
        res += 1
        y -= 1
    return res
```
def add(x, y):
    res = x
    while y > 0:
        res += 1
        y -= 1
    return res

def mult(x, y):
    product = 0
    while y > 0:
        product = add(product, x)
        y -= 1
    return product

x = int(input("x = "))
y = int(input("y = "))
res = mult(x, y)
print("x*y=", res)
**Functions**

```python
def add(a, b):
    res = a
    while b > 0:
        res += 1
        b -= 1
    return res

def mult(x, y):
    product = 0
    while y > 0:
        product = add(product, x)
        y -= 1
    return product
```

**Definition of addition function**

**IN:** values $a, b$

**OUT:** $a + b$

**Arguments:** $a, b$

They will take the values of variables used when the function is called.

**Local variables:** res, $a, b$

They exist only during the computation of the function.

Computation is completed with the `return` statement.

**Returned value:** res

**Call/evaluation** of add(., .) function

**Result:** product + $x$
Homework #3

Deadline: Nov 16, 10:30
(only one week)
def enc(k, m):
    """Encode the message m with Caesar cipher and shift key k. Change only lowercase characters, keep other characters""
    
a = list(m.encode("ascii"))
    # TO WRITE HERE#
    c = bytes(a).decode("ascii")
    return c

# Main program
k = 3
plaintext = input("Plaintext: ")
ciphertext = enc(k, plaintext)
print(ciphertext)
code_a = ord('a')                       # Compute the char code of 'a' (=97)
nb_letters = 26

msg = input("Enter a string: ")         # User can choose a string
cc = list(msg.encode("ascii"))          # cc means Character Codes here

for i in range(len(msg)):               # Iterate for the length of the msg
    char = msg[i]                       # Get the i-th character from msg
    code = cc[i]                        # Get the i-th char code
    offset = code - code_a
    if 0 <= offset < nb_letters:       # Check if the char is a lowercase letter
        print(f"{char} : {code} , {offset}"")
    else:
        print(f"{char} : {code}"
ALPHABET = range(ord('a'), ord('z')+1)  # Create a "list" containing all char codes
# corresponding to lowercase letters

msg = input("Enter a string: ")  # User can choose a string

for char in msg:
    code = ord(char)
    offset = code - ALPHABET[0]
    if code in ALPHABET:  # Check if the char is a lowercase letter
        print(char, ":", code, ",", offset)
    else:
        print(char, ":", code)

Often many options to solve the same problem!
Modern Cryptography and Secure Communication
Modern Cryptography

Evolution of encryption schemes:

Symmetric ciphers

Caesar cipher: Used in the Ancient Rome.
Enigma: Used by Germans during the World War II.
DES: In 1977, adopted as a standard in the US. Some modifications are still in use.
AES: In 2001, adopted as a standard in the US. Currently used worldwide.

1980-ies

Public-key encryption

Public key – used for encryption, available to everyone (public).
Private key – used for decryption (private).
### Key distribution

#### Symmetric encryption

$k = \text{symmetric key (private)}$

Let us use this key

#### Public-key encryption

$pk = \text{public key (available to everyone)}$

$sk = \text{secret key (private)}$

Use this for encryption
Attack from Eavesdropper

Public-key encryption

Use this for encryption

Alice’s

Bob

Charlie

Alice’s

Alice

Bob

Charlie

Alice

This problem takes 1 billion years to solve!!

Encryption using

Plaintext candidates

\[ m_1, m_2, \ldots, m_j \]

One of them will match

Alice’s

pk

Ciphertext

C

Encryption using

Alice’s

pk
Summary

- Functions and subroutines
  \[ \text{add} : (a, b) \rightarrow a + b \]

- Cryptography
  - Caesar cipher
  - Symmetric vs. public-key encryption

```python
def add(a, b):
    return a + b
```