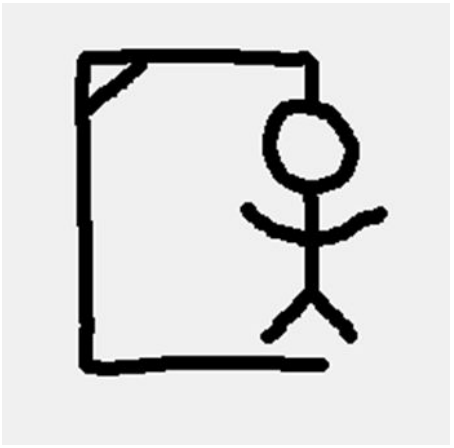
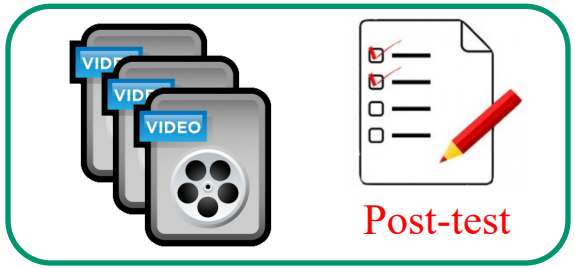
CMPUT 175
Introduction to Foundations of Computing



Simple enciphering and Hangman Word Game Design and Implementation with Python



You should view the vignettes:

Transposition Cipher

Need for Data Structures

Post-test will close on January 21st

Objectives

- Revision of Programming concepts learned in CMPUT 174
- Review of Basic Python from CMPUT 174
- Discussing how to solve a problem to be implemented in a programming language
- Implementing simple enciphering and Hangman word game in Python
- Maybe learn something new
- Have fun!

Simple Enciphering (Deciphering)

rrd ehtHa#tcao sd ea#oe crihewg#Mhe mf ahsm#
yeinpobyaoe# rveuri.sms#b ewt r e.#

Private key is “CMPT175”

Method: get order of letters in the Private key

C	M	P	T	1	7	5
3	4	5	6	0	2	1

0 1 2 3 4 5 6
➔157CMPT
CMPT175
➔3456021

My brot
her rec
eived a
new co
mputer
for his
birthd
ay. He
has awe
some ga
mes.

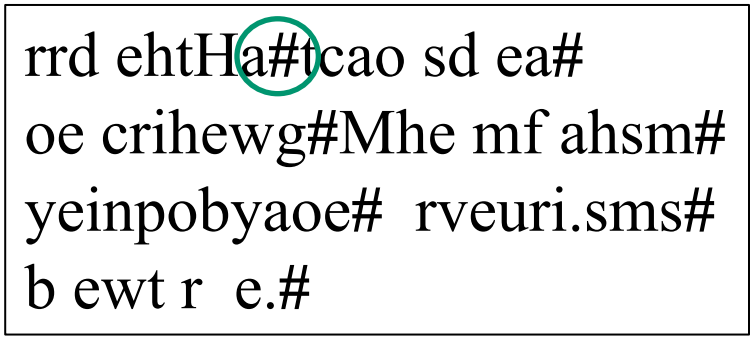
My brother received a new computer for his birthday. He has awesome games.

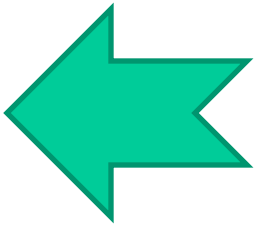
Simple Enciphering (Enciphering)

My brother received a new computer for his birthday. He has awesome games.

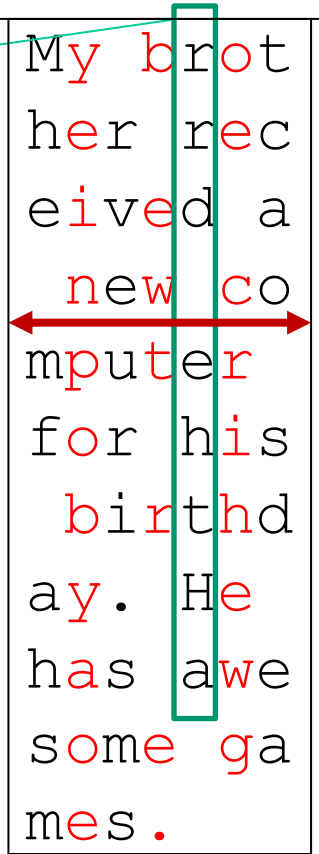
Private key is “CMPT175”

Method: get order of letters in Private key





Slice the message by column in order



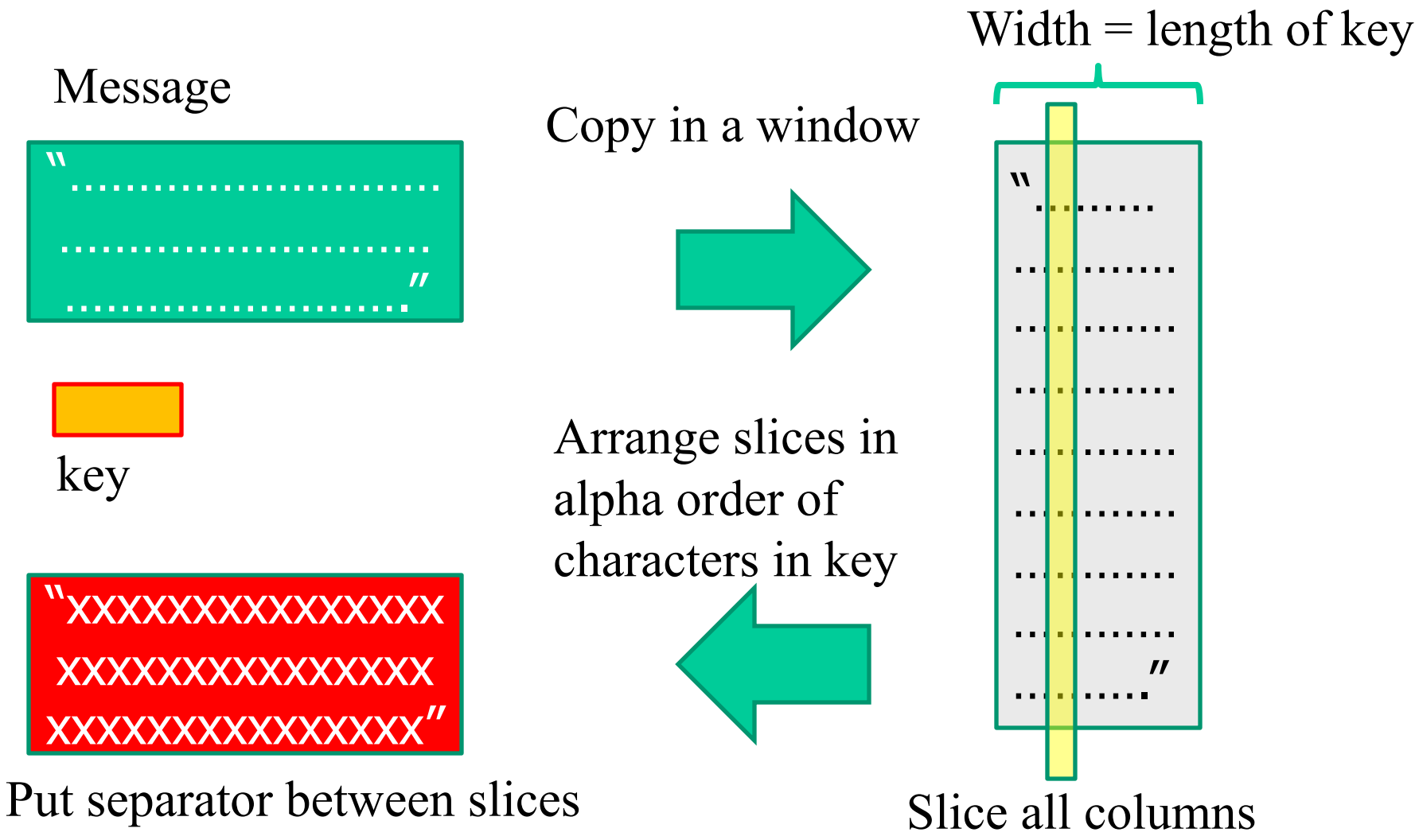
Width = length of key

Simple Enciphering

- Encryption
  - Input — Message + Private Key
  - Data processing — Do the trick to manipulate the string
  - Output — Encrypted Message

- Decryption
  - Input — Encrypted Message + Private Key
  - Data processing — Do the reverse trick to manipulate the string
  - Output — Clear Message

Data Processing



Simple Enciphering

- We need 2 strings one for the clear messages and one for the enciphered message

myMessage = “My brother received a new computer for his birthday. He has awesome games.”

Cipher = “rrd ehtHa#tcao sd ea#oe crihewg#Mhe mf ahsm#yeinpobyaoe# rveuri.sms#b ewt r e.#”

- We need a string for the private key

myPrivateKey = "CMPT175"

- We need a list for the ordering of the letters in the key

myOrder = **[3,4,5,6,0,2,1]**

- We need a list of strings for the snips

```
["My brot",
"her rec",
"eived a",
" new co",
"mputer ",
"for his",
" birthd",
"ay. He ",
"has awe",
"some ga",
"mes."]
```

- And some other containers for temporary variables for processing

Simple Enciphering Flow Chart

1-Read Message and PrvKey

↓

2-Get order of letters in PrvKey

↓

3-Cut the Message in snips the size of the PrvKey and align them

↓

4-Get the columns of the aligned snips in the sorted order of the letters of the PrvKey and concatenate them

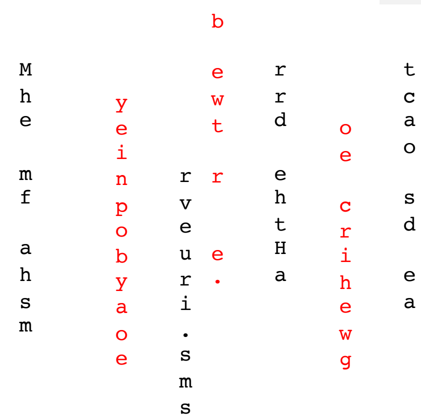
↓

5-Output new Message

myMessage = “My brother received a new computer for his birthday. He has awesome games.”
myPrivateKey = “CMPT175”

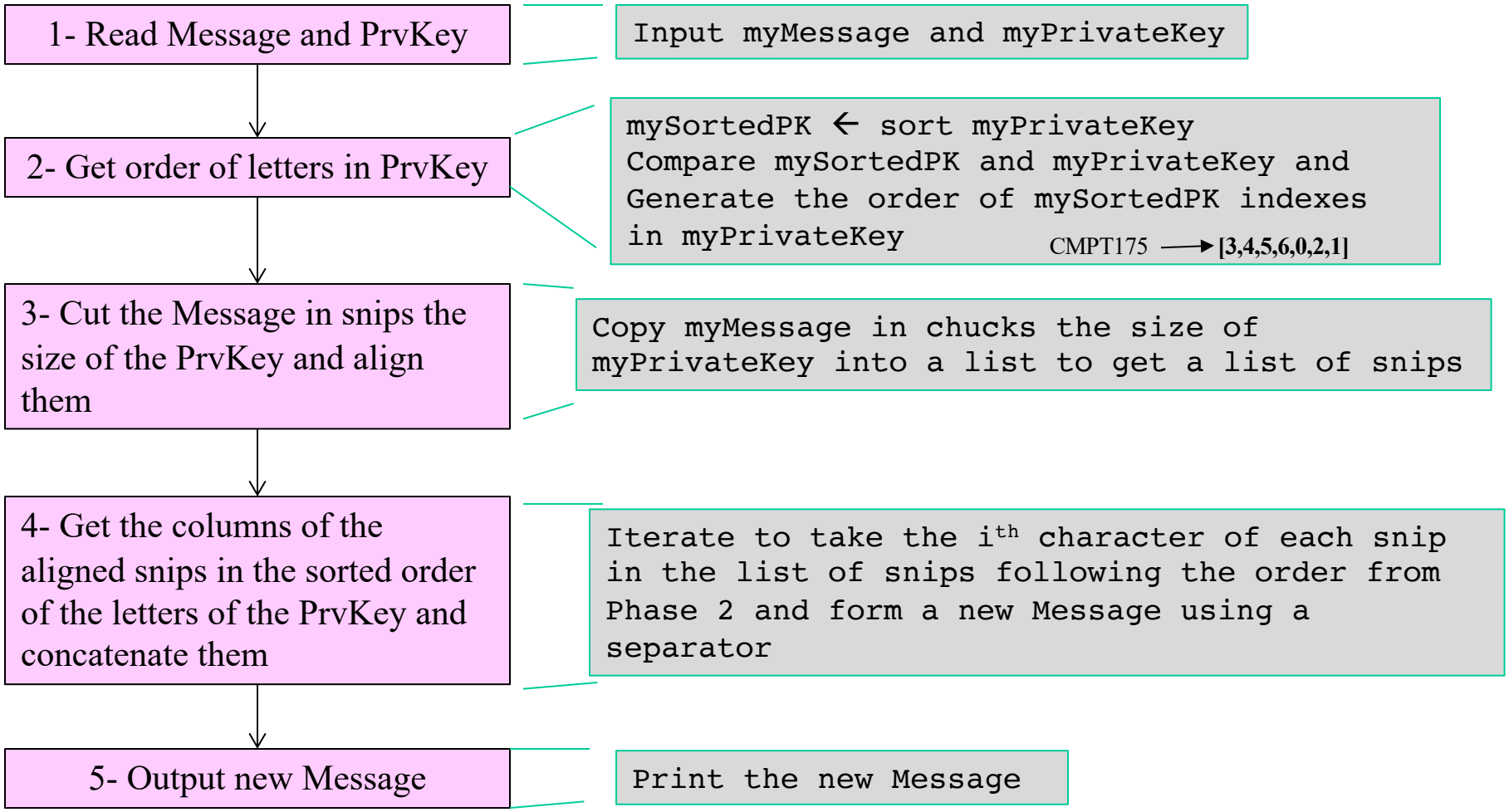
myOrder = **[3,4,5,6,0,2,1]**

```
["My brot",
"her rec",
"eived a",
" new co",
"mputer ",
"for his",
" birthd",
"ay. He ",
"has awe",
"some ga",
"mes."]
```



Cipher = “rrd ehtHa#tcao sd ea#oe crihewg#Mhe mf ahsm#yeinpobyaoe# rveuri.sms#b ewt r e.#”

Simple Enciphering Algorithm



Simple Enciphering in Python

1.Input message & private_key

2.sorted_pk ← sort private_key
Compare sorted_pk and private_key and determine the position of each character of private_key in sorted_pk

3.Make the length of the message a multiple of the length of the order

4.Copy message in chunks the size of order into a list to get a list of snips

5.Iterate to take the i^{th} character of each snip in the list of snips following the order from Phase 2 and form a new Message using a separator

6.Print the new Message

```
message=input("Enter your Message:")
private_key=input("Enter your Private Key:")
```

```
sorted_pk ← sort private_key
For each character in private_key
   check at what position it is in sorted_pk
and add that position to the order list
```

CMPT175 ⟶ **[3,4,5,6,0, 2,1]**

```
sorted_pk = sorted(private_key) # returns a list
order = []
for letter in private_key:
    position = sorted_pk.index(letter)
    sorted_pk[position] = ''
    order.append(position)
```

Why adding a space here?

Duplicate characters in key

Simple Enciphering in Python

private_key = “CMPUT175”
sorted_pk = ['1', '5', '7', 'C', 'M', 'P', 'T']

```
sorted_pk = sorted(private_key)
order = []
for letter in private_key:
    position = sorted_pk.index(letter)
    sorted_pk[position] = ''
    order.append(position)
```

When *letter is* ‘**C**’ position is **3 in sorted_pk**
When *letter is* ‘**M**’ position is **4 in sorted_pk**
When *letter is* ‘**P**’ position is **5 in sorted_pk**
And so on.At the end we get
order is [3,4,5,6,0,2,1]

If we have private_key = 'ABBA’
sorted_pk = ['A', 'A', 'B', 'B']

What will be the order?

[0,2,3,1]

Simple Enciphering in Python

1.Input message & private_key

2.sorted_pk ← sort private_key
Compare sorted_pk and private_key and determine the position of each character of private_key in sorted_pk

3.Make the length of the message a multiple of the length of the order

4.Copy message in chunks the size of order into a list to get a list of snips

5.Iterate to take the i^{th} character of each snip in the list of snips following the order from Phase 2 and form a new Message using a separator

6.Print the new Message

```
while (len(message) % len(order) !=0):
        message = message + ' '
```

message = 'My brother received a new computer for his birthday. He has awesome games.'

len(message)
74

74%7 – Add a space to the message
4
75%7 – Add a space to the message
5
76%7 – Add a space to the message
6
77%7 - Stop
0

Simple Enciphering in Python

message = “My brother received a new computer for his birthday. He has awesome games. ”

```
. . .
4.Copy message in chunks
the size of order into a
list to get a list of
snips

. . .
```

```
i=0
snips=[]
while i<len(message):
    snip=message[i:i+len(order)]
    snips.append(snip)
    i=i+len(order)
```

snips

```
["My brot",
"her rec",
"eived a",
" new co",
"mputer ",
"for his",
" birthd",
"ay. He ",
"has awe",
"some ga",
"mes.   "]
```

snip

```
i=0   My brot
i=7   her rec
i=14  eived a
i=21   new co
i=28  mputer
i=35  for his
i=42   birthd
i=49  ay. He
...
i=77  mes.
```

cipher = “rrd ehtHa #tcao sd ea #oe crihewg #Mhe mf ahsm#yeinpobyaoe# rveuri.sms#b ewt r e.#”

Simple Enciphering in Python

```
snips=["My brot","her rec","eived a"," new co","mputer ","for his","
birthd","ay. He ","has awe","some ga","mes.    "]
```

order= **[3,4,5,6,0, 2,1]**

```
i=0; position=4
snip ="My brot"; cipher='r'
```

```
i=0 position = 4 snip="her rec"; cipher='rr'
```

```
i=0 position = 4 snip="eived a"; cipher='rrd'
```

```
i=0 position = 4 snip=" new co"; cipher='rrd '
```

```
. . .
5.Iterate to take the
character at position of each
snip in the list of snips
following the order from Step
2 and form a new Message
using a separator

6.Print the new Message
```

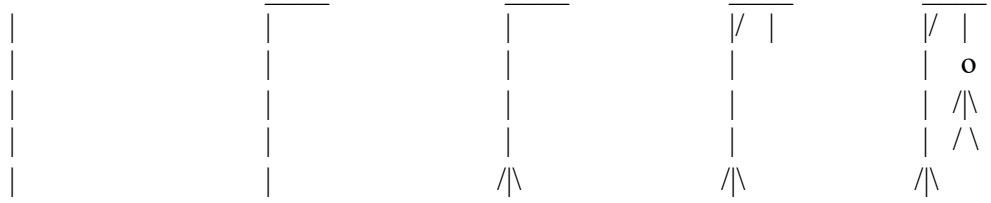
```
cipher = ''
for i in range(len(order)):
    position = order.index(i)
    for snip in snips:
        cipher = cipher+snip[position]
    cipher = cipher +'#'
print(cipher)
```

Other details in the program

- What if the original message contains "#"?
- Should we only use one separator #?
- Could we alternate between separators?
- What about a long message in a file?
- Deciphering is the exact reverse procedure
- More details in the complete python code on the course e-class page

Hangman Word Game

- Hangman is a word guessing game. One player thinks of a word, and the other tries to guess it by suggesting letters.
- The word to guess is represented by a row of dashes, giving the number of letters: _ _ _ _
- When a letter is guessed correctly it is written in its position; when a letter is incorrectly suggested a section of the gallows is drawn.
- The game ends when the player guesses and completes the word or the hang man is drawn on the gallows. The hangman and gallows are drawn in 5 steps (i.e. 5 wrong guesses)



Input of the problem

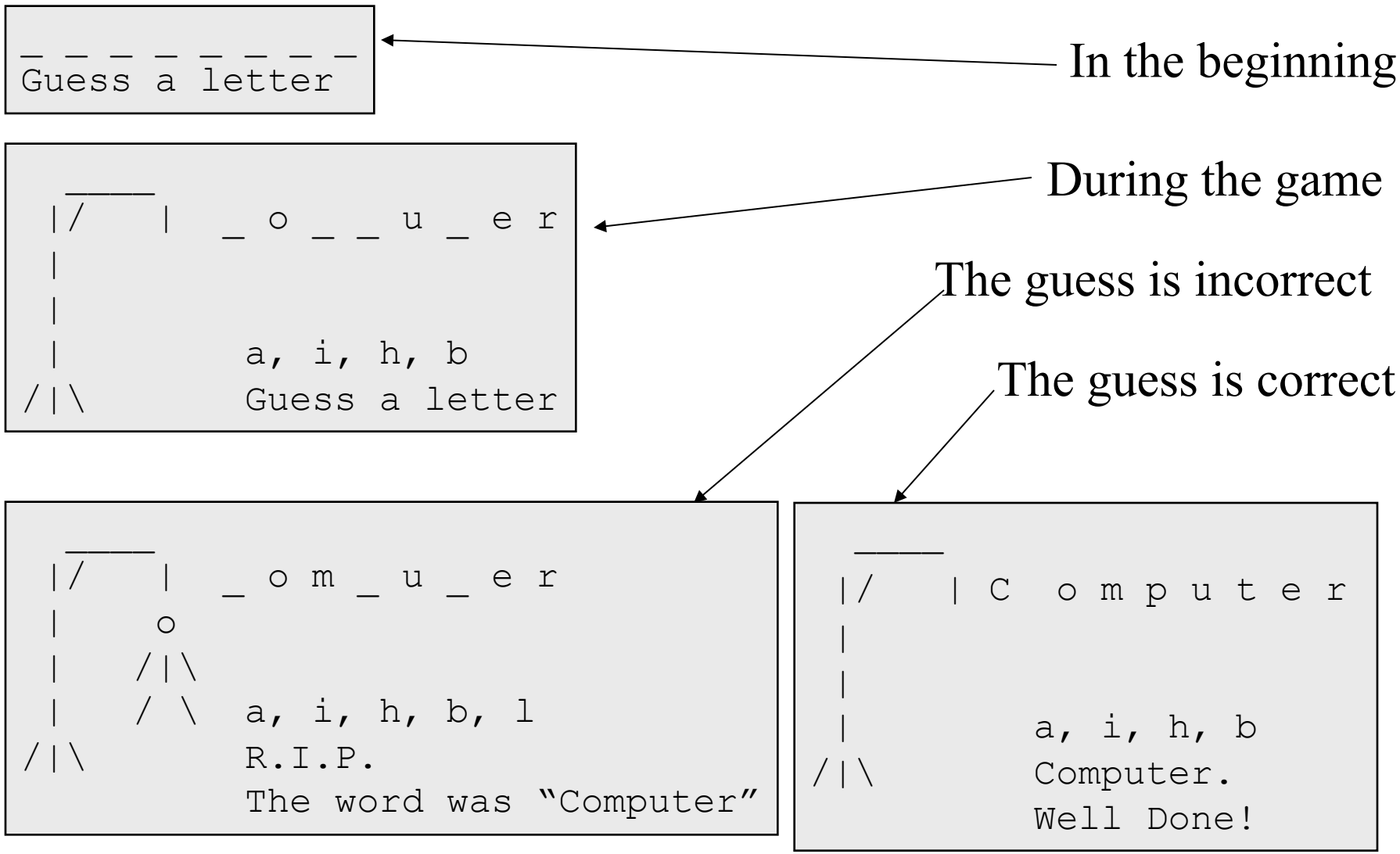
- There is a text file of words
- Each line of the file contains a word
- E.g. Cat
- We will traverse the file sequentially

- Letter guesses are entered on the keyboard

Expected Output

- Initially
  - Dashes indicating the number of letters in the word eg: _ _ _ (assume *cat* is the word)
- During the guesses (assuming *a* was suggested)
  - Dashes where there still missing letters eg:_a_
  - Suggested letters in their correct position eg:_a_
  - List of wrong suggestions in the order they were suggested. eg: h, e, s, o
  - If the suggested letter is already in the correct position or the the suggested letter is in the list of wrong suggestions then it should be ignored.
  - The drawing of the hangman at the step equal to the number of wrong suggestions.
- When the game is over
  - Display the word and either R.I.P. if hangman is finished or "Well done" if the word is correctly guessed.

Examples of Output



Process

Initialization

- Read the file to get a word
- Display dashes

Game

- Repeat until game over
  - Get letter suggestion
  - If letter in word, display letter in position
  - If letter not in word, draw gallows

Conclusion

- If word guessed, display "well done"
- If word not guessed, display "R.I.P."

Visualizing the problem

Read word W

Display *Template*

repeat

When to stop?

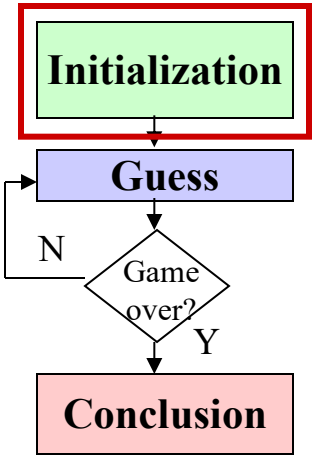
1- Read character C
2- Correct → update Template
3- NOT Correct →
  3.1- Update Gallows
  3.2- Append C in List of wrong guesses
4- Display *Template*, *Gallows*, and List

Display RIP ← ? → Display Well Done

[example]

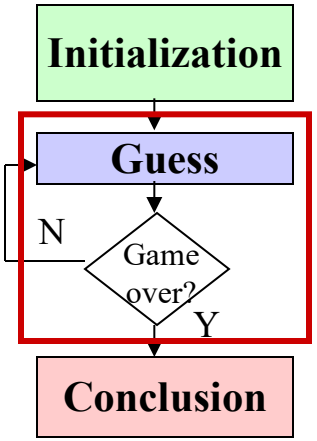
Template is _a_
List of wrong guesses is h, e, s, o
Gallows is

Pseudo-code



- Initialization
  - Open file;
  - Read one word W from file;
  - Close file;
  - Check length of W
  - Initialize template ← ['_'] * len(word)
  - Initialize hangman.step ← 0
  - Initialize game_over ← false
  - Initialize wrong_guesses ← []

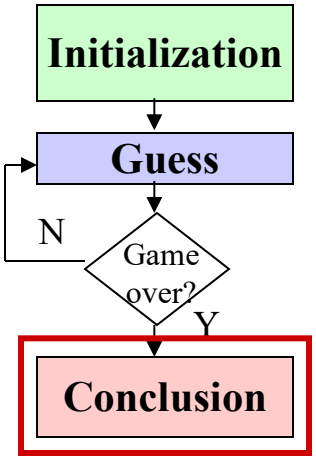
Pseudo-code



- Repeated Guess
  - Display template;
  - Read Character C from keyboard
  - If Exists(C, W)
    - Update(template,C)
  - Else
    - hangman.step ← hangman.step + 1
    - Display hangman
    - Append (C, wrong_guesses)
    - Display wrong_guesses
  - game_over ← (hangman.step=5) or (template.full)

Pseudo-code

- Conclusion
  - If template.full
    - Display “Well Done”
  - else
    - Display W
    - Display “R.I.P.”

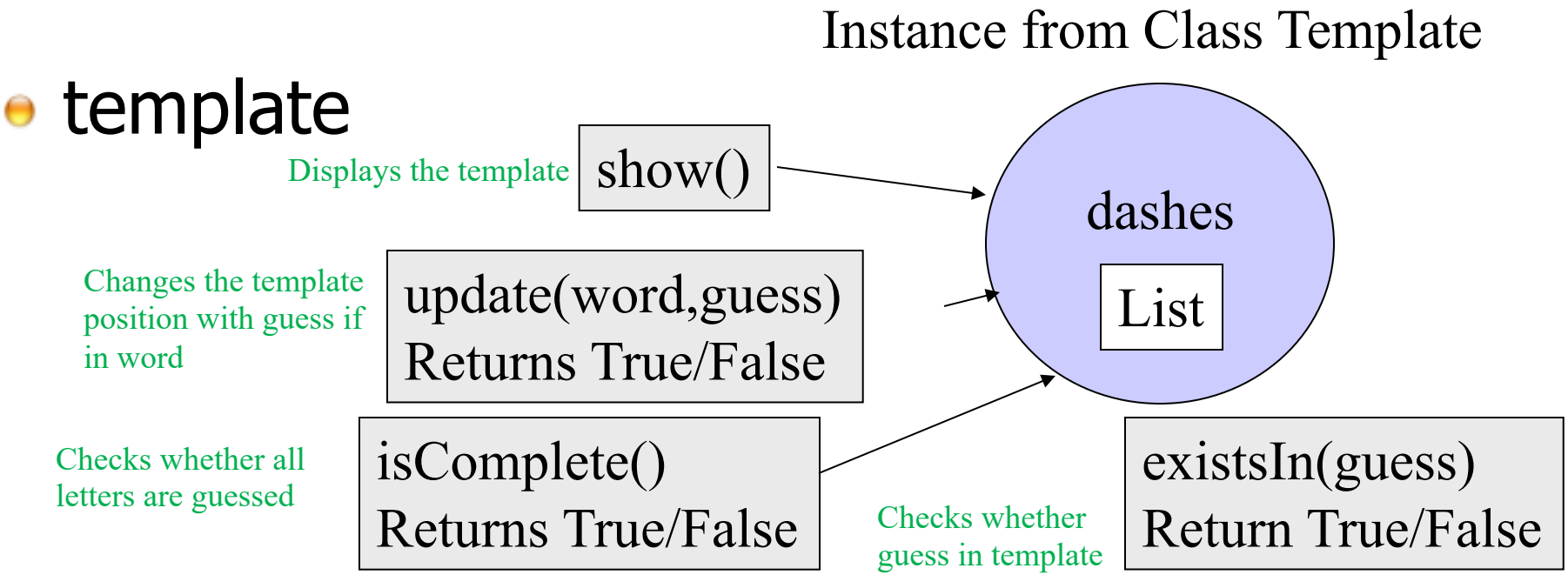


Data Structure Needed

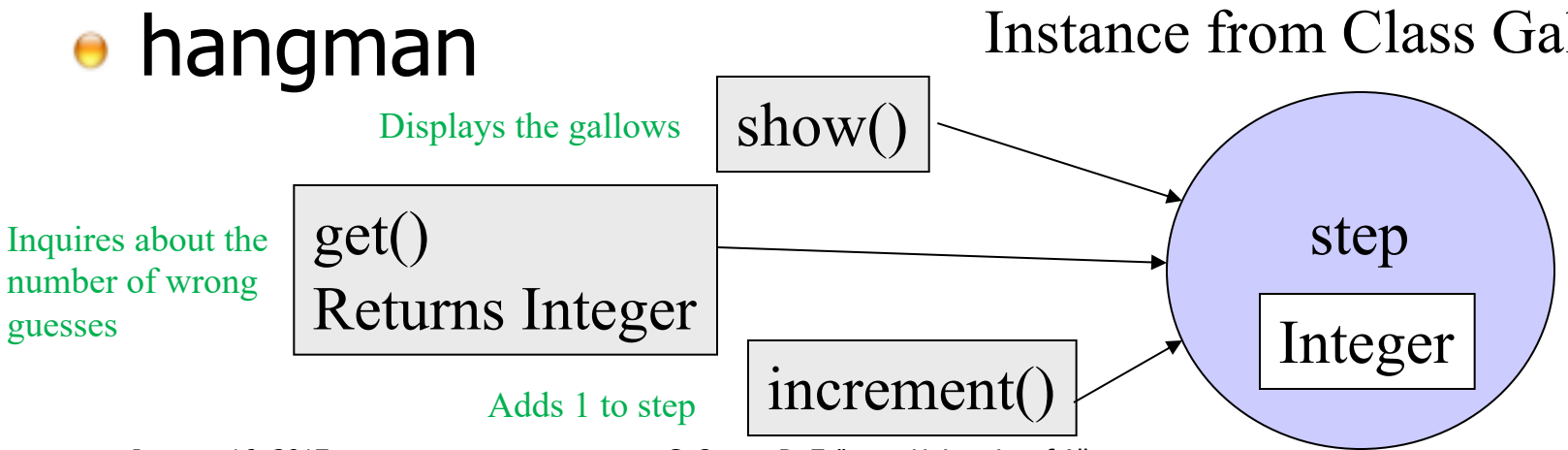
word	String
guess	String
template	Object
hangman	Object
game_over	Boolean
incorrect	List

Objects

- template

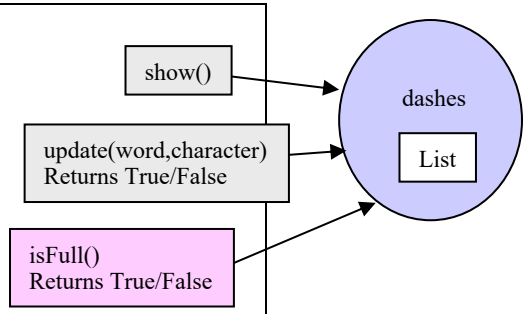


- hangman

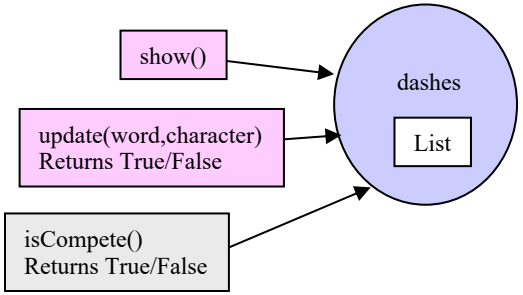


Implementation of classes with Python

```
### class Template
class Template:
    def __init__(self,size):
        self.dashes = ["_"]*size

    def isComplete(self):
        if "_" in self.dashes: return False
        else: return True
```



Implementation of classes with Python




```
def update(self, word, guess):
    found=False
    for i in range(len(word)):
        if guess == word[i]:
            self.dashes[i]=guess
            found=True
    return found
```

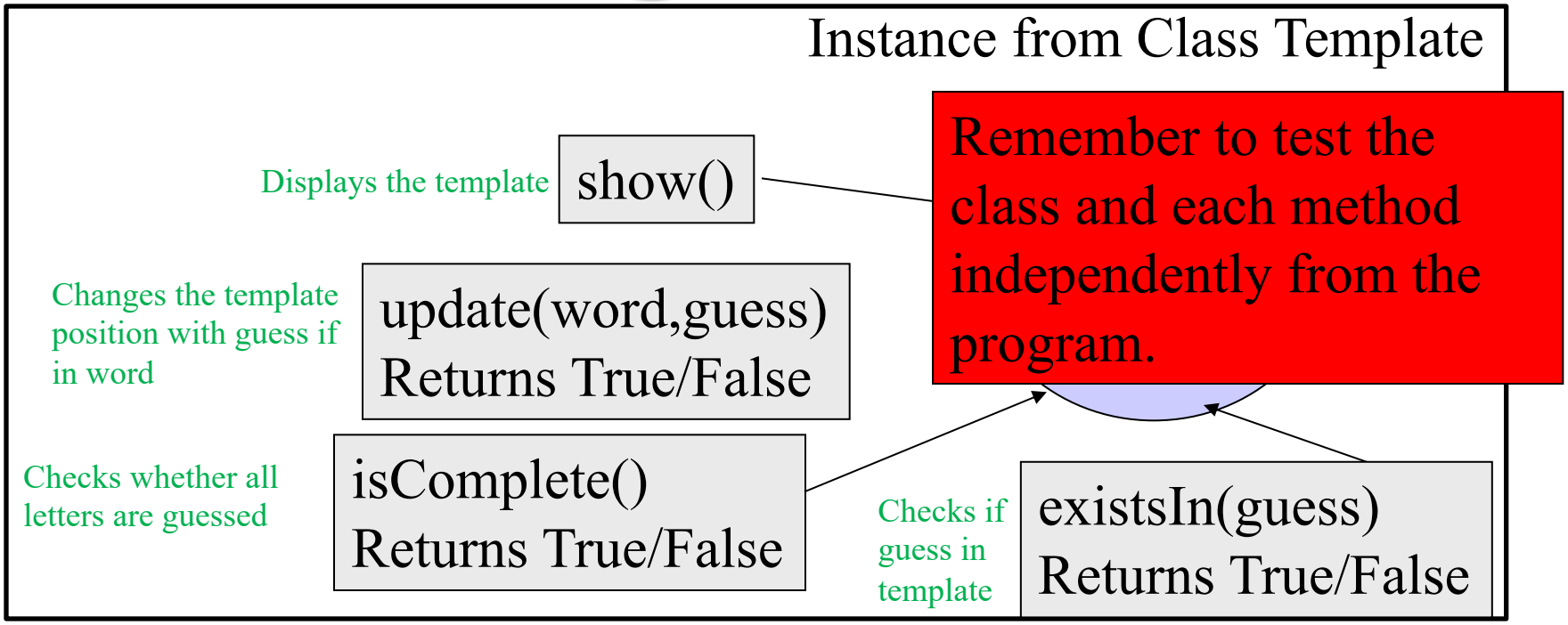
Tells me if character is in word.

Remember to test the class and each method independently from the program.

```
def show(self):
    line = ""    #string to be displayed
    for i in range(len(self.dashes)):
        line += self.dashes[i] + " "
    print (line)
```

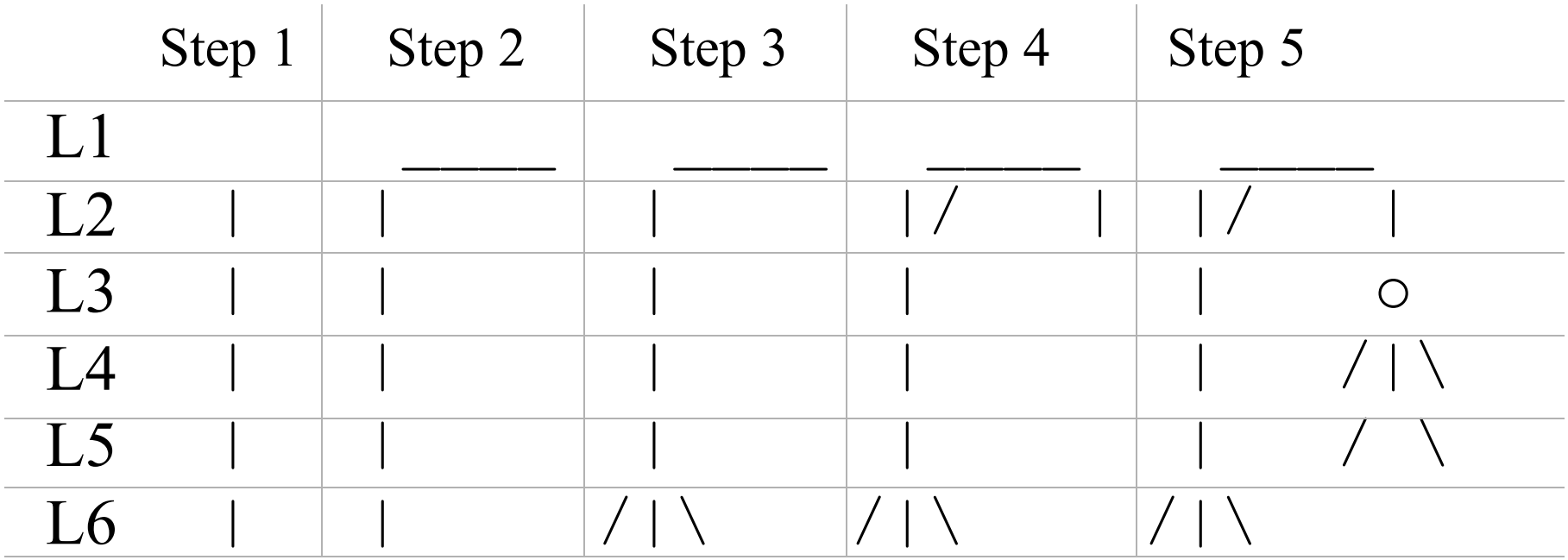
Create a template then show it.
Update the template then show it.
Check if it is full, etc.

Testing the class



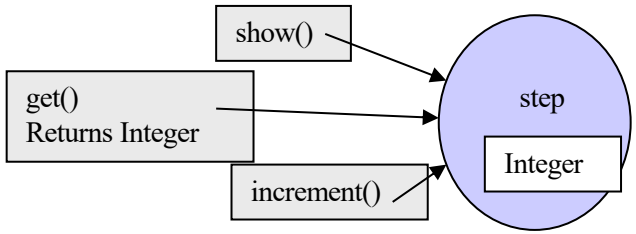
Create a template then show it.
Update the template then show it.
Check if it is full, etc.

Steps for the Gallows



- **L2** to **L6** are always at least "|"
- Except for Step 1 **L1** is always "_____"
- Above Step2 **L6** has a base "/|\"
- Above Step3 **L2** has reinforcement and rope "|/ |"
- In Step5 the man is hung

Implementation of classes with Python



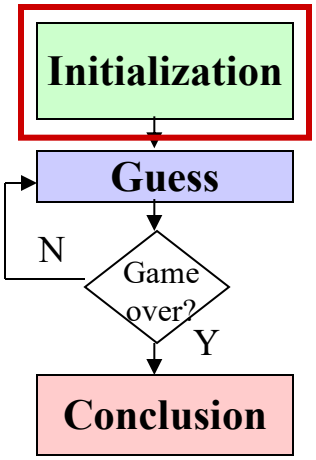

```
### class Gallows
class Gallows:
    def __init__(self):
        self.step = 0

    def increment(self):
        self.step +=1

    def get(self):
        return self.step
```

```
    def show(self):
        l1=l2=l3=l4=l5=l6=""
        if self.step>0:
          l2=l3=l4=l5=l6=" |"
        if self.step>1 :
          l1=" ____"
        if self.step>2:
          l6="/|\\"
        if self.step>3:
          l2=" |/  |"
        if self.step==5:
          l3=" |   o"
          l4=" |  /|\\"
          l5=" |  / \\"
        print (l1,l2,l3,l4,l5,l6,sep="\n")
```

Implementation with Python

```
### hangman word game
myfile = open('wordfile','r')
myWord = myfile.readline()
myfile.close()
incorrect = []
gameOver = False
myWord=myWord.rstrip()
myTemplate = Template(len(myWord))
hangman = Gallows()
```

- Initialization
  - Open file;
  - Read one word W from file;
  - Close file;
  - Check length of W
  - Initialize template
  - Initialize hangman
  - Initialize game_over ← false
  - Initialize wrong_guesses ← []

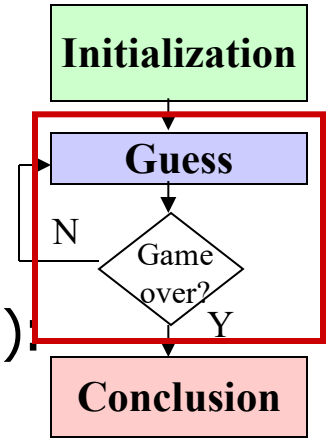


Notice only 1st word is read. A better solution is needed

Implementation with Python

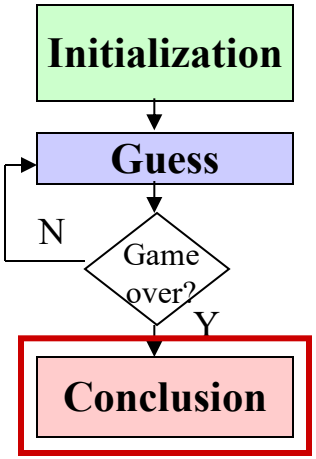
```
while not gameOver:
    guess = input('Guess a letter >')
    # ignore guess if it is already exist in the template
    # ignore guess if it exists in the incorrect list
    while(guess in incorrect or myTemplate.existsIn(guess)):
        guess = input('Guess a letter >')
    # Proceed to update once guess is valid
    updated = myTemplate.update(word,guess)
    myTemplate.show()
    if updated == False:
        incorrect.append(guess)
        hangman.increment()
        incorrect = sorted(incorrect)
        print(','.join(incorrect))
        hangman.show()
    if myTemplate.isComplete() or hangman.get() == 5:
        gameOver = True
```

Repeated Guess
  Read Character C from keyboard
  If Exists(C, W)
    Update(template,C)
    Display template
  Else
    hangman.step ← hangman.step + 1
    Display template
    Display hangman
    Append (C, wrong_guesses)
  game_over ← (hangman.step=5) or (template.full)



Implementation with Python

```
if myTemplate.isComplete():
    print ("Well Done!")
    print ("The Word was indeed " + myWord)
else:
    print (out + "R.I.P.!")
    print ("The Word was " + myWord)
```



Testing

```
_ _ _ _ _ _ _ _ _

Guess a letter:
```

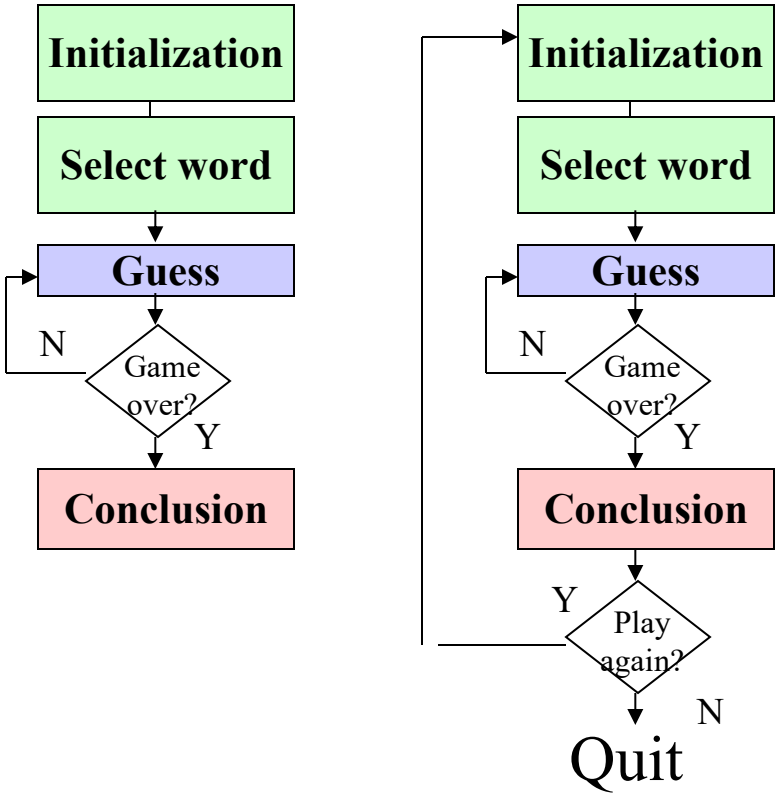
```
  _____
 |/    |
 |     o
 |    /|\
 |    / \
 /|\

c _ _ _ _ t e r

x, g, h, y, k,
R.I.P.!

The Word was computer
```

```
  _____
 |/    |
 |
 |
 |
 /|\

c _ _ _ _ t e r

x, g, h, y,
Guess a letter:
```

```
Well Done!
The word is indeed: computer
```

Repeating the game

- Ask whether the player wants to play again



Adding a Global Loop

```
infile = open('words.txt', 'r')
list_of_lines = infile.readlines()
infile.close()
play_again = True
while(play_again):
    # Choose a random line
    a_line = random.choice(list_of_lines)
    a_line = a_line.strip()
    # Separate the word and the hint
    word,hint = aline.split(',')
    print('HINT :', hint)
    play_one_round(word)
    reply = input('Do you want to play again?y/N')
    if reply.lower() != 'y':
        play_again = False
print("Thank you for playing Hangman")
```

```
infile = open('words.txt', 'r')
list_of_lines = infile.readlines()
infile.close()
play_again = True
while(play_again):
    # Choose a random line
    a_line = random.choice(list_of_lines)
    a_line = a_line.strip()
    # Separate the word and the hint
    word,hint = aline.split(',')
    print('HINT :' ,hint)
    play_one_round(word)
    reply = input('Do you want to play
   again?y/N')
    if reply.lower() != 'y':
        play_again = False
print("Thank you for playing Hangman")
```

Call a function

```
def play_one_round(word):
    gameOver = False
    incorrect = [ ]
    myTemplate = Template(word)
    hangman = Gallows()
    myTemplate.show()
    while not gameOver:
        guess = input('Enter a letter :')
        while guess in incorrect or
myTemplate.existsIn(guess) or not guess.isalpha():
            guess = input('Invalid guess! Enter letter again :')

        result = myTemplate.update(word,guess)
        myTemplate.show()
        if result == False:
            incorrect.append(guess)
            incorrect.sort()
            print(','.join(incorrect))
            hangman.increment()
            hangman.show()
        if hangman.get() == 5 or myTemplate.isComplete():
            gameOver = True
    if myTemplate.isComplete():
        print('Congratulations !')
    else:
        print('RIP')
        print('The word is ', word)
```

Improvements

1. Each line of the file contains a word and a hint. These are comma separated
   - E.g. Cat, animal
   - Display the hint with the template
2. Select randomly within the file
3. The list of wrong guesses should be sorted alphabetically
4. A wrong guess should not be entered again
5. Only valid input should be accepted.

```
horse,animal
orange,fruit
tree,living thing
bee,animal
keyboard,object
computer,object
machine,object
automobile,object
canada,country
…
```

More suggestions for output

Line 1
Line 2
Line 3
Line 4
Line 5
Line 6

```
 _____
|/    |
|          _ o _ _ u _ e r
|
|          a, i, h, b
/|\        Guess a letter
```

Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7

```
 _____
|/    |
|     o    _ o m _ u _ e r
|    /|\
|    / \   a, i, h, b, l
/|\          R.I.P.
             The word was "Computer"
```

```
 _____
|/    |
|
|
|
/|\

_ o _ _ u _ e r

a, i, h, b,
Guess a letter:
```

As opposed to