Intro to Computer Science
CS-UH 1001, Spring 2022

Lecture 4 – String Methods, Sequences, Lists

Today's Lecture

- More string methods
- Sequences
- Lists
- List indexing
- List methods

Recap

- The input(message) function is used to get data from the user
  - You can give a message as an argument to prompt the user what input you expect
  - Everything is read and returned as a string
  - If you want to input numbers then you have to type cast, e.g. int() or float()

Recap

- String indexing: string[index]
  - Always starts from 0
  - Can be positive
  - Can be negative
- String slicing: string[start:end]
  - end is always excluded
  - You can also define step size [start:end:step]

- len() returns string length
- ASCII
  - ord() and chr()
- Comparison operators
  - ==, !=

Recap

- Some string methods
  - .upper()
  - .lower()
  - .capitalize()
  - .title()

- All string methods return a value (string)! They do not change the string variable!

Replacing Substrings

- Replacing a substring in a string:

```
>>> substring = string.replace(old, new, max)
```

- Arguments:
  - old: substring to be replaced
  - new: new substring to replace the old substring
  - max: number of replacements (optional)

- It returns a copy of the string in which the occurrences of old have been replaced with new

Example: Replacing Substrings

```
>>> name = ‘This is not an advertisement!’

>>> name.replace(‘is’, ‘was’)
>>> print(name)
‘This is not an advertisement!’

>>> new_name = name.replace(‘is’, ‘was’)
>>> print(new_name)
‘Thwas was not an advertwasement!’
```

How to replace only the word ‘is’?

```
>>> new_name = name.replace(‘ is ’, ’ was ’)
>>> print(new_name)
‘This was not an advertisement!’
```

Finding Substrings

- Finding a substring in a string:
  >>> index = string.find(sub, start, end)

- Arguments:
  - sub: substring to find
  - start: start index (inclusive) for the search (optional)
  - end: end index (exclusive) for the search (optional)

- It returns:
  - the lowest (positive) index of the first character where sub is found within the string
  - -1 if no match was found

Example: Finding Substrings

```
>>> name = ‘python’
>>> print(name.find(‘p’))
0
>>> print(name.find(‘th’))
2
>>> print(name.find(‘th’,3))
-1
>>> print(name.find(‘th’,0,3))
-1
```

How about: `print(name.find(‘th’,-4,-1))`

```
2
```

Counting String Occurrences

- Counting how many times a substring is present in a string:

  occurrences = string.count(sub, start, end)

- Arguments:
  - sub: substring to count
  - start: start index (inclusive) for the count (optional)
  - end: end index (exclusive) for the count (optional)

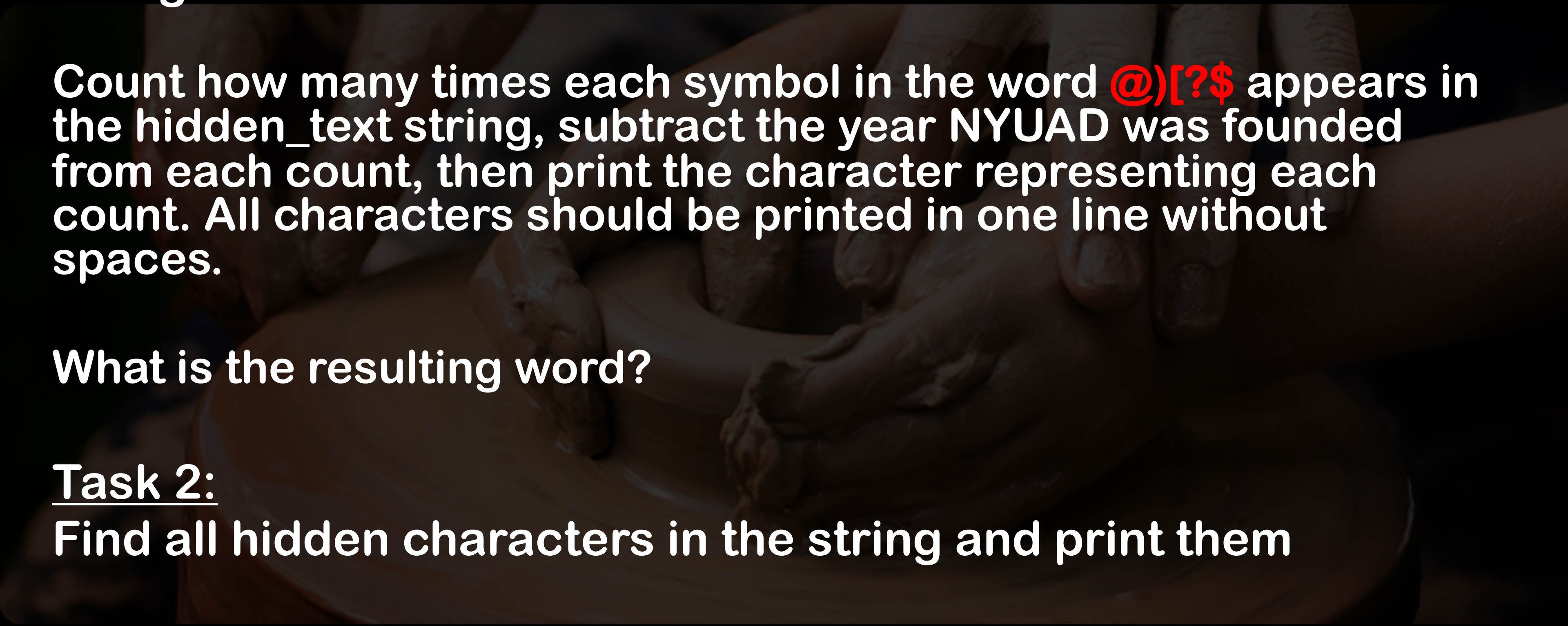
- It returns the number of occurrences of sub in the string

Example: Counting String Occurrences

```
>>> name = ‘banana’
>>> print(name.count(‘a’))
3
>>> print(name.count(‘p’))
0
>>> print(name.count(‘na’))
2
>>> print(name.count(‘a’, 3))
2
>>> print(name.count(‘a’, 0, 3))
1
```

Breakout session I: String methods

Download ex_4.1.py file from Brighspace and write a program to do the following tasks:



Task 1:

Translate the encoded word @)[?$ using the hidden_text string as follows:

Count how many times each symbol in the word @)[?$ appears in the hidden_text string, subtract the year NYUAD was founded from each count, then print the character representing each count. All characters should be printed in one line without spaces.

What is the resulting word?

Task 2:

Find all hidden characters in the string and print them

```
print(“Let’s take a 90 min break!”.replace(“90”, “15”))
```

Sequences and Lists

Sequences

- A sequence is an object that holds multiple items of data
  - it stores the data one after the other

- In Python, there are several types of sequences
  - String: sequence of characters
  - Lists: sequence of items of any data type

Lists

- Lists are used to store multiple items in a single variable

- Lists can contain
  - items from the same data type
  - items from different data types

- Lists are mutable (changeable)
  - List items can be modified after they have been created
  - List items can be added or removed from the list during runtime

Lists

- Lists are created using square brackets [ ]

```
>>> empty_list = [ ]
>>> odd_numbers = [1, 3, 5, 7, 9]
```

List items are (visually) separated by commas (,)

- Lists allow duplicate values
- List items are ordered*

*ordered != sorted

Examples: Lists

```
>>> odd_numbers = [1, 3, 5, 7, 9]
```

odd_numbers	1	3	5	7	9

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’, ‘Robb’]
```

names	Jon	Sansa	Arya	Robb

```
>>> mixed_list = [‘Jon’, 1, 2.5, ‘a’, True]
```

mixed_list	Jon	1	2.5	a	True

Overloaded Operators on Lists

- Similar to strings you can:
  - concatenate lists using the + operator
  - apply repetition using the * operator

- Examples:

```
>>> list1 = [1, 2]
>>> list2 = [3, 4]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4]
```

```
>>> list1 = [1, 2]
>>> num = 3
>>> list2 = list1 * num
>>> print(list2)
[1, 2, 1, 2, 1, 2]
```

List Indexing and Slicing

- List items are indexed

```
>>> numbers = [1, 2, 3, 4, 5]
>>> print(numbers[1])
2
```

0	1	2	3	4
1	2	3	4	5

- Slicing works too!

```
>>> print(numbers[2:4])
[3, 4]
>>> print(numbers[1:2])
[2]
```

- Note: Slicing always returns a List!

Strings vs. Lists

Strings

- name = ‘python’

name	p	y	t	h	o	n

- Strings are defined using single or double quotation
- Indexing/slicing
  - name[0] → ‘p’
  - name[2:4] → ‘th’

Lists

- numbers = [1, 2, 3, 4, 5]

numbers	1	2	3	4	5

- Lists are defined by items inside [ ]; items are separated by commas
- Indexing/slicing:
  - numbers[0] → 1
  - numbers[2:4] → [3, 4]

Strings vs. Lists
or
Immutability vs. Mutability

- Strings are immutable:

```
>>> name = ‘python’
>>> name[0] = ‘b’        Type Error
```

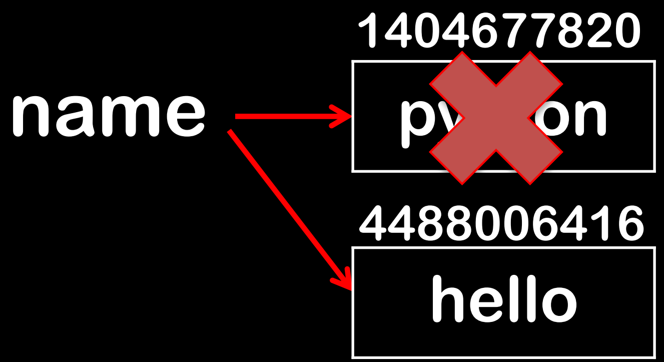
- Lists are mutable:

```
>>> even_numbers = [1, 4, 6, 8]
>>> even_numbers[0] = 2
>>> print(even_numbers)
[2, 4, 6, 8]
```

Immutability vs. Mutability

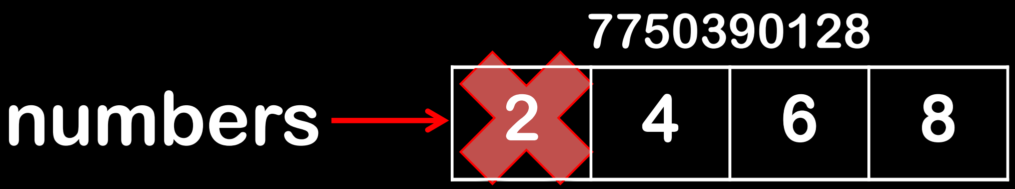
- Strings:

```
>>> name = 'python'
>>> print(id(name))
1404677820
>>> name = 'hello'
>>> print(id(name))
4488006416
```



- Lists:

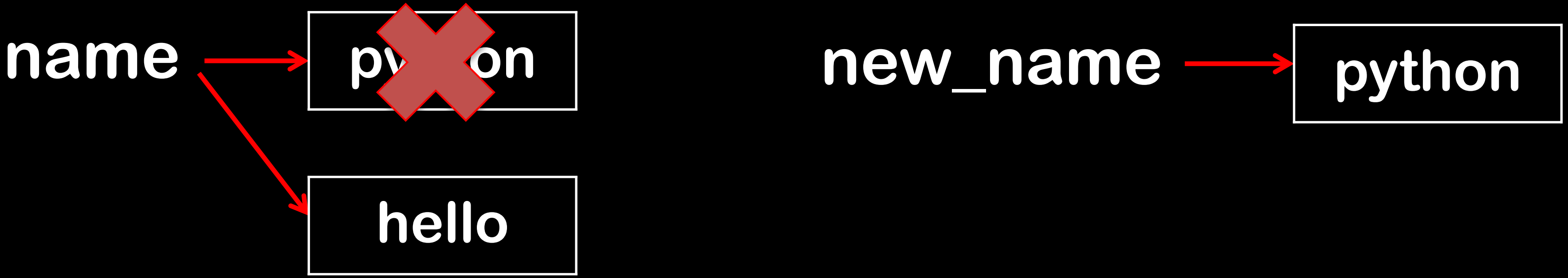
```
>>> numbers = [1, 4, 6, 8]
>>> print(id(numbers))
7750390128
>>> numbers[0] = 2
>>> print(id(numbers))
7750390128
```



Copying Strings

```
>>> name  = ‘python’
>>> new_name  = name
>>> name = ‘hello’

>>> print(name)
‘hello’
>>> print(new_name)
‘python’
```



Copying Lists

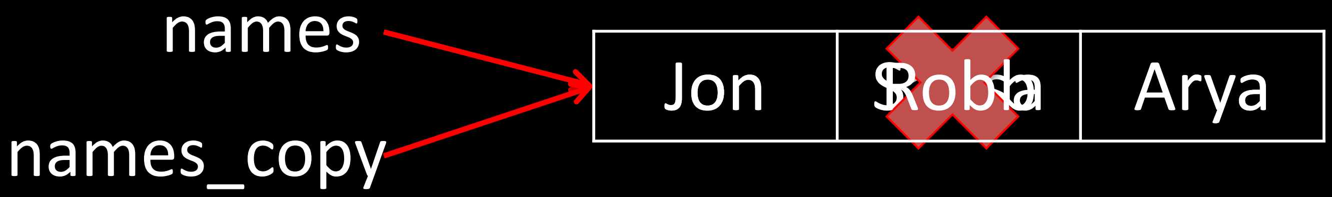
- Since Lists are mutable, list variables only store a reference to the object

- If you want to copy a list, you need to copy its items

- Just assigning the list to a new variable will **not** create a copy of the list
  - Both variables are pointing to the same memory location where the list is stored

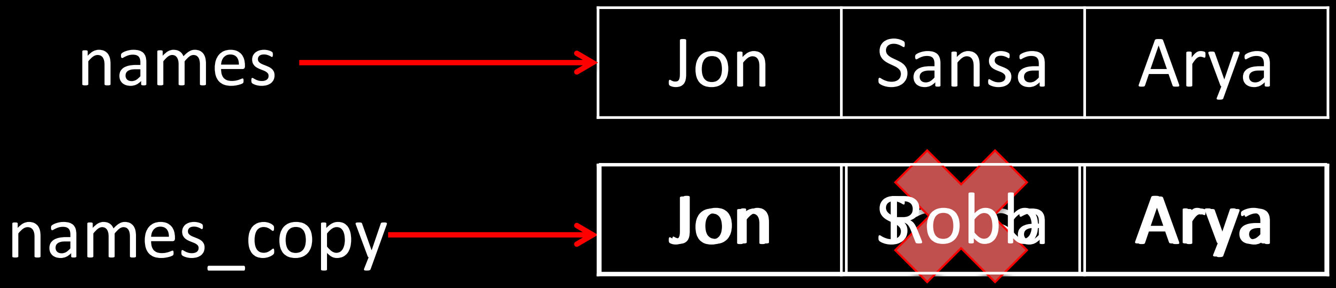
Example: Copying Lists

```
>>> names = ['Jon', 'Sansa', 'Arya']
>>> names_copy = names
>>> names_copy[1] = 'Robb'
>>> print(names)
['Jon', 'Robb', 'Arya']
```



How to copy Lists then?

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> names_copy = [] + names
>>> names_copy[1] = ‘Robb’
>>> print(names)
[‘Jon’, ‘Sansa’, ‘Arya’]
```



Lists Length

- You can check how many items are in a list using the len() function:

```
>>> names = ['Jon', 'Sansa', 'Robb']
>>> len(names)
3

>>> names = []
>>> len(names)
0
```

List Methods

Note: Since lists are mutable, most list methods can alter the content of the list. They do NOT return the altered list

Index method

- The index(item) method returns the index of the first element in the list that matches the item in the argument
  - An error is given if the item is not found in the list

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> print(names.index(‘Sansa’))
1

>>> print(names.index(‘Cersei’))
>>> ValueError: ‘Cersei’ is not in list
```

The in Operator with Lists

- Remember the in operator for strings?
  - Checks if a substring exists in another string
  - Use the in operator with lists to check if an item is present in the list or not

```
>>> names = [‘Jon’, ‘Sansa’, ‘Arya’]
>>> ‘Jon’ in names
>>> True

>>> ‘Cercei’ in names
>>> False
```

Adding Items to Lists

- The append(item) method is used to dynamically add an item to a list during runtime
- The item appended is added to the end of the list
- The append(item) method modifies the list
  - it does NOT return a new list

- Example:

```
>>> numbers = [1, 2, 3]
>>> numbers.append(4)
>>> print(numbers)
[1, 2, 3, 4]
```

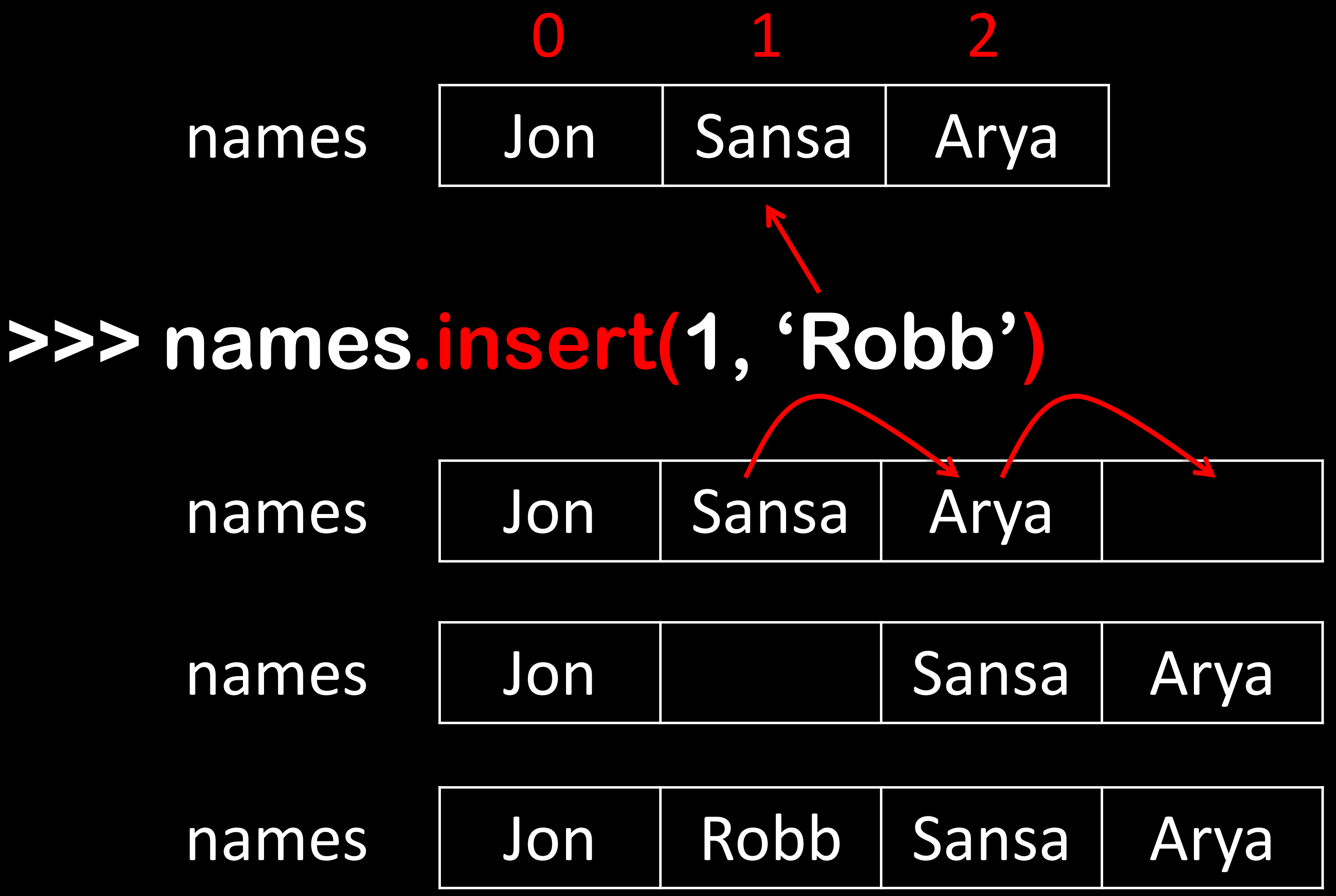
Never do:

```
>>> numbers = numbers.append(4)
>>> print(numbers)
None
```

Inserting Items Into Lists

- The insert(index, item) method
  - inserts an item to the list at a specific index
  - does NOT return the new list

Example: Inserting Items



```
>>> names.insert(1, ‘Robb’)
```

Example: Inserting Items

	0	1	2
names	Jon	Sansa	Arya
	-3	-2	-1

- **What will happen if you use an invalid index?**

```
>>> names.insert(50, ‘Robb’)
```

names	Jon	Sansa	Arya	Robb

- **What will happen if you use a negative index?**

```
>>> names.insert(-3, ‘Robb’)
```

names	Robb	Jon	Sansa	Arya

Other Useful List Methods and Functions

Method	Description
.sort()	Sort the items within the list in ascending order (from lower value to upper value)
.reverse()	Reverse the order of the items in the list
.count(item)	Counts how may times an item appears in the list
min (myList)	returns the element with the minimum value in the list
max (myList)	returns the element with the maximum value in the list

You can find more list methods here: https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Convert Strings to Lists

- Remember type casting?
  int(), float() and str()?

- A string can be changed into a list using the list() type cast:

```
>>> name = ‘python’
>>> my_list = list(name)
>>> print(my_list)
[‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’]
```

Converting Lists to Strings

- How about a string of words?

```
>>> sentence = ‘I love python’
>>> print(list(sentence))
[‘I’, ‘ ‘, ‘l’, ‘o’, ‘v’, ‘e’, ‘ ‘, ‘p’, ‘y’, ‘t’, ‘h’, ‘o’, ‘n’]
```

Splitting Strings

- The split(separator) method splits a string

string.split(separator)

- The separator argument is optional; by default “ “
- It returns a list of strings

```
>>> sentence = ‘I love python’
>>> print(sentence.split())
    [‘I’, ‘love’, ‘python’]
```

Splitting Strings

- You can also define the separator

```
>>> sentence = ‘I-love-python’
>>> sentence.split(‘-’)   [‘I’, ‘love’, ‘python’]
>>> sentence.split(‘o’)    [‘I-l’, ‘ve-pyth’, ‘n’]
>>> sentence.split(‘love’)   [‘I-’, ‘-python’]
>>> sentence.split()          [‘I-love-python’]
```

Lists to Strings

- The join(list) method does the opposite of the split() method
- It joins all items in the list into one string

```
string.join(list)
```

- It returns a string by joining all list elements, separated by the string
- Note: list must contain string items!

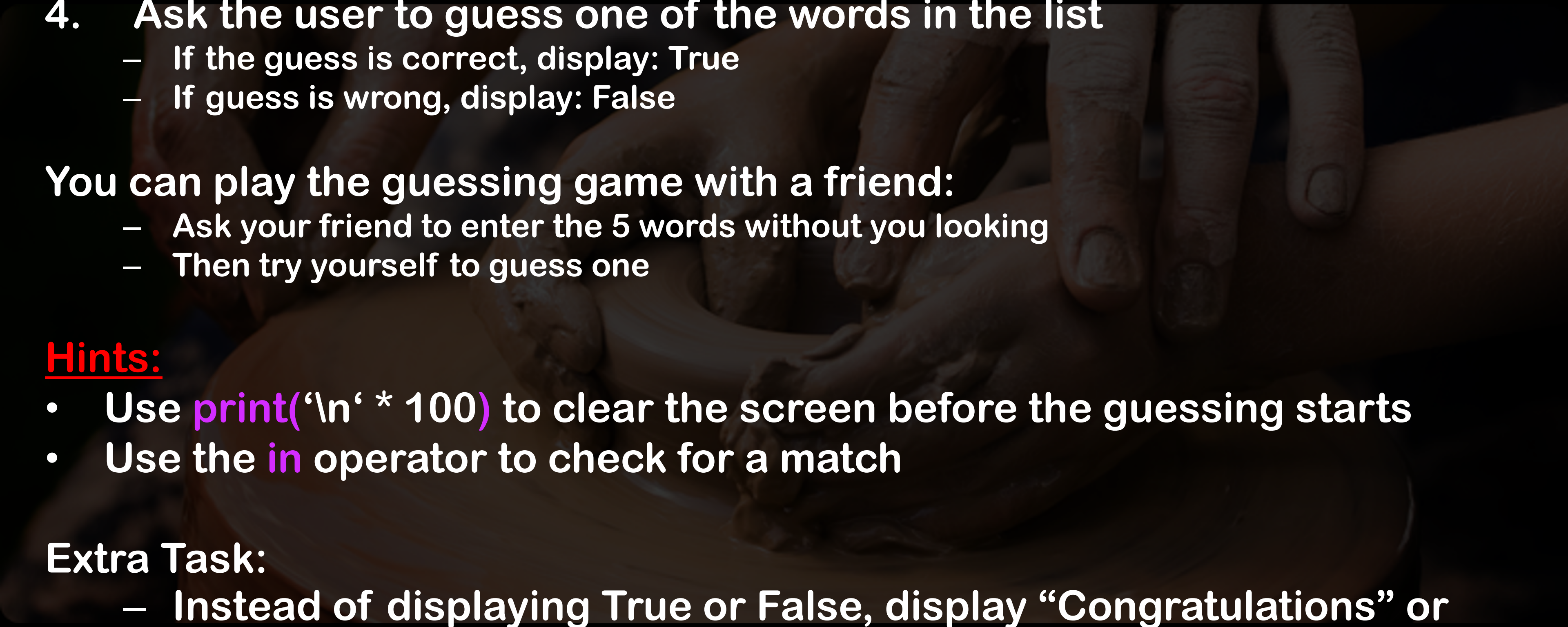
Example: Lists to Strings

```
>>> separator = “ “
>>> word_list = [‘I’, ‘love’, ‘python’]
>>> joined_string = separator.join(word_list)
>>> print(joined_string)
I love python

>>> separator = “_”
>>> joined_string = separator.join(word_list)
>>> print(joined_string)
I_love_python
```

Breakout session II:
Lists

Guessing game (ex_4.2.py)



Write a program that

1. Asks the user to input 4 words separated by comma (all with the same theme, e.g. fruits)
2. Create a list out of all 4 words
3. Then ask the user for one more word. Add that word to the beginning of the list
4. Ask the user to guess one of the words in the list
   - If the guess is correct, display: True
   - If guess is wrong, display: False

You can play the guessing game with a friend:

- Ask your friend to enter the 5 words without you looking
- Then try yourself to guess one

Hints:

- Use print('\n' * 100) to clear the screen before the guessing starts
- Use the in operator to check for a match

Extra Task:

- Instead of displaying True or False, display "Congratulations" or "Sorry, wrong guess"
- Remember: A True has the value of 1, whereas a False has the value of 0. Use them as indices to print the two strings above

Next Class

- More list methods
- Logical operators
- Decision structures

Removing an Item From a List

- If you want to remove an item from a list, there are three different ways to do this:

  – list.remove(item) removes the first occurrence of the item within the list (item is NOT returned!)

  – del list[index] removes the item at the specific index from the list

  – list.pop() removes the last item from the list and returns it

    - list.pop(index) removes an item at the specific index from the list and returns it

Example: remove() Method

	0	1	2	3
names	Jon	Sansa	Arya	Robb

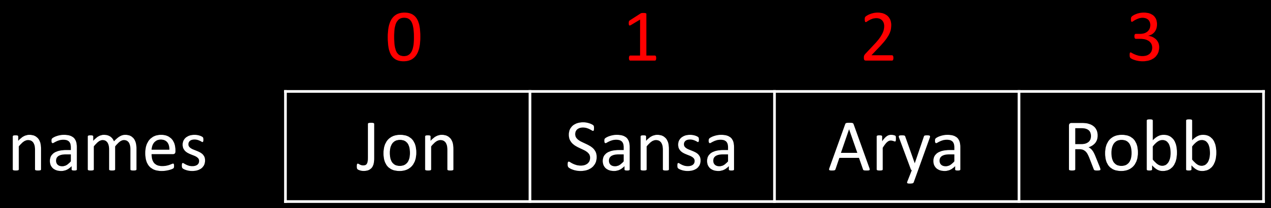
```
>>> names.remove('Sansa')
```

names	Jon	~~Sansa~~	Arya	Robb
names	Jon	Arya	Robb	

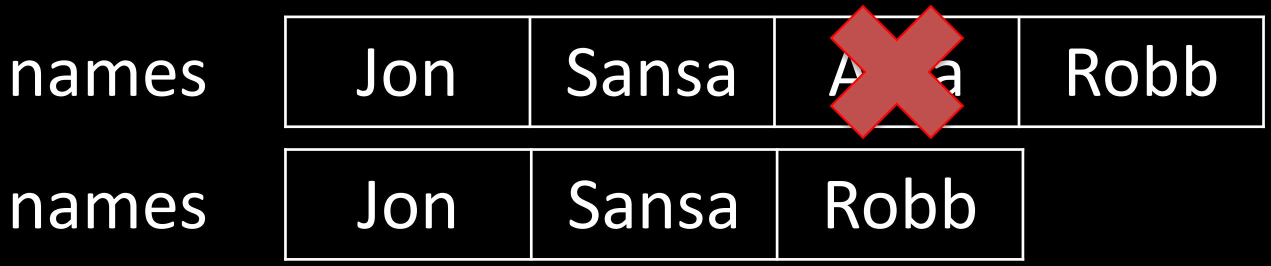
Never do:

```
>>> names = names.remove("Sansa")
>>> print(names)
None
```

Example: del Statement

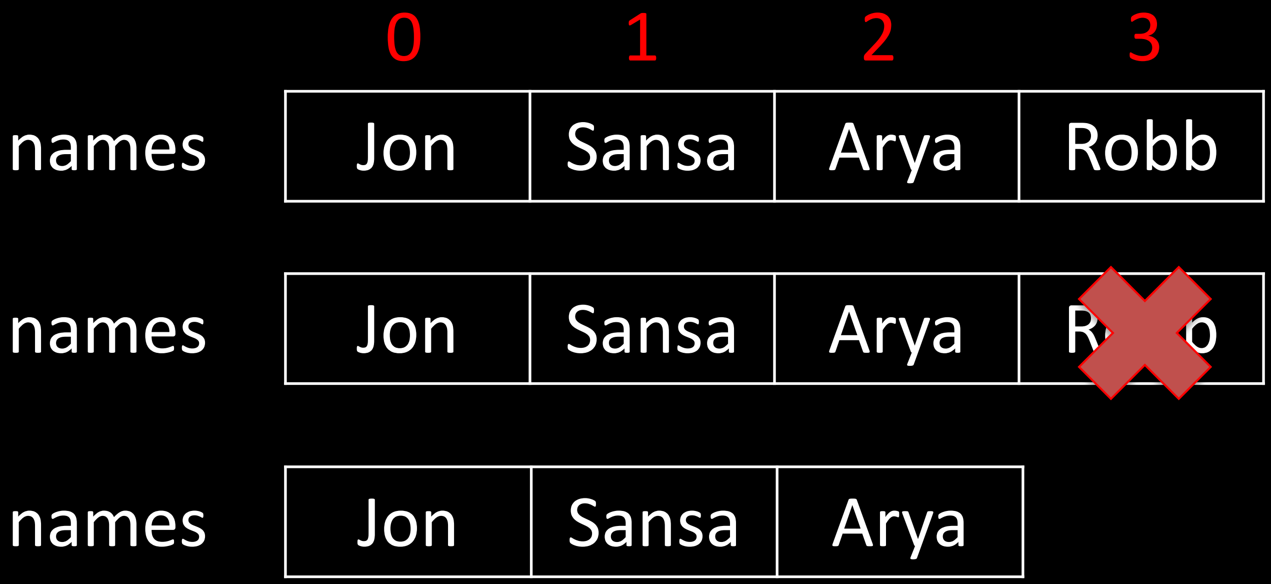


```
>>> del names[2]
```



```
>>> del names[5]        IndexError
```

Example: pop() Method



```
>>> name = names.pop()
>>> print(name)
Robb
```