ILP 2023 – W5S1+
Project (End)

Matthieu DE MARI – Singapore University of Technology and Design

Survey and course feedback

- You should have received a survey link, asking for your feedback about this summer school.

- If not done already, take a minute to fill it up!

Project

Let us continue with activities from the project (previous lecture!)

ILP 2023 – W5S2-3
Everything about strings

Matthieu DE MARI – Singapore University of Technology and Design

Outline (Week5, Session2-3 – W5S2-3)

- The string type (and similarities with the list type)
- Indexing, slicing, and traversing a string
- Updating a string and unmutability
- Membership and find
- Booleans on strings, sorting
- Printing and formatting
- Strings <-> lists conversions
- Splitting, character checking
- (More stuff, RTFM!)

From lists to strings

- Lists and strings behave in a very similar way.
- **Lists:** multiple **elements** between **brackets**, separated by commas
- **Strings:** multiple **characters** (~elements) between **single** or **double quotes (‘, “)**, without any separators.

- Strings are **roughly the same thing as a list of characters**.

string = “Hello”

equivalent_list = [“H”, “e”, “l”, “l”, “o”]

- **Great news:** This means that most functions, methods and operators working on lists will work on strings as well!

Length of a string

- The **length** of a string is simply defined as the number of characters in the string.
- This is obtained, as before via the **len()** function.
- **Spaces count as characters!**

```
1  a_string = "This is a string."
2  print(a_string)
3  print(len(a_string))
```

```
This is a string.
17
```

Using triple quotes and string formatting (\)

Need to use the single or double quote symbol in your strings?

1. Replace the single or double quotes at the beginning and end of the string **with three of these symbols (“”” or ‘’’)**.
2. Use a **backslash symbol** (\) before a single or double quote in a string.

You can also use some string formatting

- **\n** to start a new line (~ pressing the Enter key),
- and **\t** for tabulating (~ pressing the Tab key)

Using triple quotes and string formatting (\)

```
1  # A terrible computer science joke
2  joke1 = "Bob was heading to the supermarket."
3  joke2 = 'As he was leaving, his wife told him:'
4  joke3 = ''' "While you're out, pick up some eggs!" '''
5  joke4 = "He never came back. \"I\'m really sad\", said the wife."
6  joke5 = "\nAlso, the store eventually ran out of eggs."
7  joke6 = "\t The End."
8  print(joke1)
9  print(joke2 + joke3)
10 print(joke4, joke5)
11 print(joke6, end = "(badum tsss!)")
```

```
Bob was heading to the supermarket.
As he was leaving, his wife told him: "While you're out, pick up some eggs!"
He never came back. "I'm really sad", said the wife.
Also, the store eventually ran out of eggs.
         The End.(badum tsss!)
```

The + and * operators on strings

Strings behave as lists.

- For this reason, the + operator will **concatenate two strings,**
- And the * operator will **repeat and concatenate** a string **n times**, with **n** being an **integer positive value**.

```
1  list1 = [1, 2, 3]
2  list2 = [4, 5, 6, 7]
3  n = 3
4  new_list = list1*n + list2
5  print(new_list)
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3, 4, 5, 6, 7]
```

```
1  str1 = "pika"
2  str2 = "chu"
3  n = 3
4  new_string = str1*n + str2
5  print(new_string)
```

```
pikapikapikachu
```

Indexing and slicing a string

As with lists, you can

- **index** a string, by using the bracket notation [] on a list. It returns the **character** at the index position.
- **slice** a string, by using the bracket notation [] on a list. It returns the **substring** with specified indexes.

```
1  # Calling/indexing a string as in lists
2  a_string = "This is a string."
3  print(a_string[0])
4  print(a_string[-2])
```

```
T
g
```

```
1  # Slincing a string is fine as well
2  a_string = "This is a string."
3  print(a_string[5:12])
4  print(a_string[-6:-1:2])
```

```
is a st
tig
```

Unmutable types

- A major difference between lists and strings, however, is that strings are **unmutable**.

- **Unmutable** means that you can not modify parts of it, by using the bracket notation, as you would with lists.

```
1 # Lists are mutable and can be updated
2 a_list = [1, 4, 7, 9]
3 a_list[2] = 11
4 print(a_list)
```

```
[1, 4, 11, 9]
```

```
1 # Strings are unmutable
2 # (i.e. cannot be changed once defined)
3 a_string = "Your GPA is 2"
4 a_string[-1] = 5 # Does not work
```

```
---------------------------------------------------------
TypeError                                 Traceback (most r
<ipython-input-3-736e16edf069> in <module>
      2 # (i.e. cannot be changed once defined)
      3 a_string = "Your GPA is 2"
----> 4 a_string[-1] = 5 # Does not work

TypeError: 'str' object does not support item assignment
```

```
1 # Strings are unmutable
2 # (i.e. cannot be changed once defined)
3 a_string = "Your GPA is 2"
4 a_string[-1] = "5" # Stop it we said
```

```
---------------------------------------------------------
TypeError                                 Traceback (most r
<ipython-input-4-a2c24ccda094> in <module>
      2 # (i.e. cannot be changed once defined)
      3 a_string = "Your GPA is 2"
----> 4 a_string[-1] = "5" # Stop it we said

TypeError: 'str' object does not support item assignment
```

Replacing parts of a string

Instead, the strings come with a **replace()** method.

- Its two parameter are **strings s1** and **s2** and the method is applied on a third **string s3**.
- The output is a new string, consisting of the third string s3, where the substring **s1** has been replaced with **s2**.
- The **replace()** method replaces **all occurrences** of **s1** in **s3** with **s2**.

```
1 a_mistake = "British cuisine is the best."
2 fixed = a_mistake.replace("British", "French")
3 print(fixed)
```

```
French cuisine is the best.
```

Traversing a string

Because strings are roughly the same things as lists, they can be traversed using a **for** loop,

- Element-wise,

```
1  # Traversing a string
2  # (element-wise)
3  a_string = "Hello"
4  for character in a_string:
5      print("---")
6      print(character)
```

```
---
H
---
e
---
l
---
l
---
o
```

Traversing a string

Because strings are roughly the same things as lists, they can be traversed using a **for** loop,

- Element-wise,
- Index-wise,

```
1  # Traversing a string
2  # (index-wise)
3  a_string = "Hello"
4  for index in range(len(a_string)):
5      print("---")
6      print(a_string[index])
```

```
---
H
---
e
---
l
---
l
---
o
```

Traversing a string

Because strings are roughly the same things as lists, they can be traversed using a **for** loop,

- Element-wise,
- Index-wise,
- Index- and element-wise with enumerate.

```
1  # Traversing a string
2  # (enumerate)
3  a_string = "Hello"
4  for index, character in enumerate(a_string):
5      print("---")
6      print(index, character)
```

```
---
0 H
---
1 e
---
2 l
---
3 l
---
4 o
```

Traversing a string

Because strings are roughly the same things as lists, they can be traversed using a **for** loop,

- Element-wise,
- Index-wise,
- Index- and element-wise with enumerate.
- (Zip, etc. also works!)

```
1  # Traversing a string
2  # (enumerate)
3  a_string = "Hello"
4  for index, character in enumerate(a_string):
5      print("---")
6      print(index, character)
```

```
---
0 H
---
1 e
---
2 l
---
3 l
---
4 o
```

The in keyword

- As with lists, the **in** keyword can be used to check if a character or a string appears in another string.
- It returns a Boolean set to **True** or **False** accordingly.

```
1  # Membership keyword: in
2  a_string = "Pikachu"
3  bool1 = "k" in a_string
4  print("bool1: ", bool1)
5  bool2 = "chu" in a_string
6  print("bool2: ", bool2)
7  bool3 = "swag" in a_string
8  print("bool3: ", bool3)
```

```
bool1:  True
bool2:  True
bool3:  False
```

Activity 1 – Is it a palindrome

- A **palindrome** is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam, racecar, etc.
- There are also numeric palindromes, including date/time stamps using short digits 11/11/11 11:11 and long digits 02/02/2020.
- Sentence-length palindromes may be written when allowances are made for adjustments to capital letters, punctuation, and word dividers, such as "A man, a plan, a canal, Panama!".

Activity 1 – Is it a palindrome

- A **palindrome** is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam, racecar, etc.
- There are also numeric palindromes, including date/time stamps using short digits 11/11/11 11:11 and long digits 02/02/2020.
- Sentence-length palindromes may be written when allowances are made for adjustments to capital letters, punctuation, and word dividers, such as "A man, a plan, a canal, Panama!".

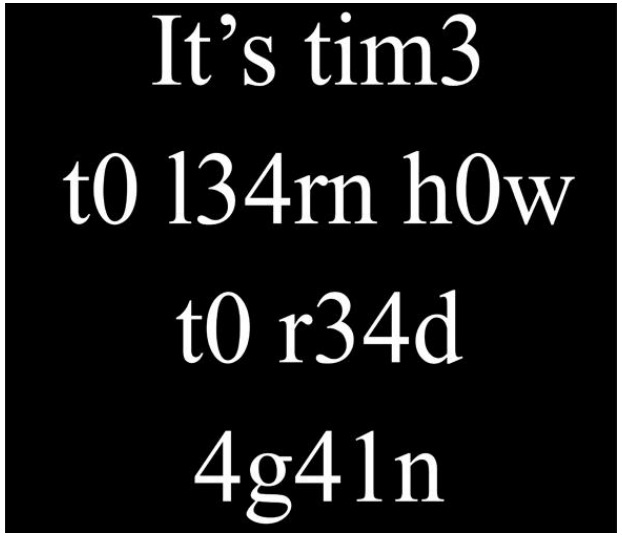
- Write a function **is_palindrome()**, which receives a **single word**, in a string variable **word** as its sole input parameter.
- It should return **True** if the string in question is a **palindrome**.
- And **False otherwise**.

Activity 2 – 1337 speak translators

Leet (or "1337") is a system of modified spellings used primarily on the Internet. It often uses character replacements in ways that play on the similarity of their glyphs via reflection or other resemblance.

Leet speak replaces the letters I with 1, the letters Z with 2, the letters E with 3, the letters A with 4, the letters S with 5, the letters T with 7, the letters B with 8, the letters O with 0.

For instance, the English sentence "HELLO THERE, HOW ARE YOU?" will write in leet speak as "H3LL0 7H3R3, H0W 4R3 Y0U?".



Activity 2 – 1337 speak translators

Leet (or "1337") is a system of modified spellings used primarily on the Internet. It often uses character replacements in ways that play on the similarity of their glyphs via reflection or other resemblance.

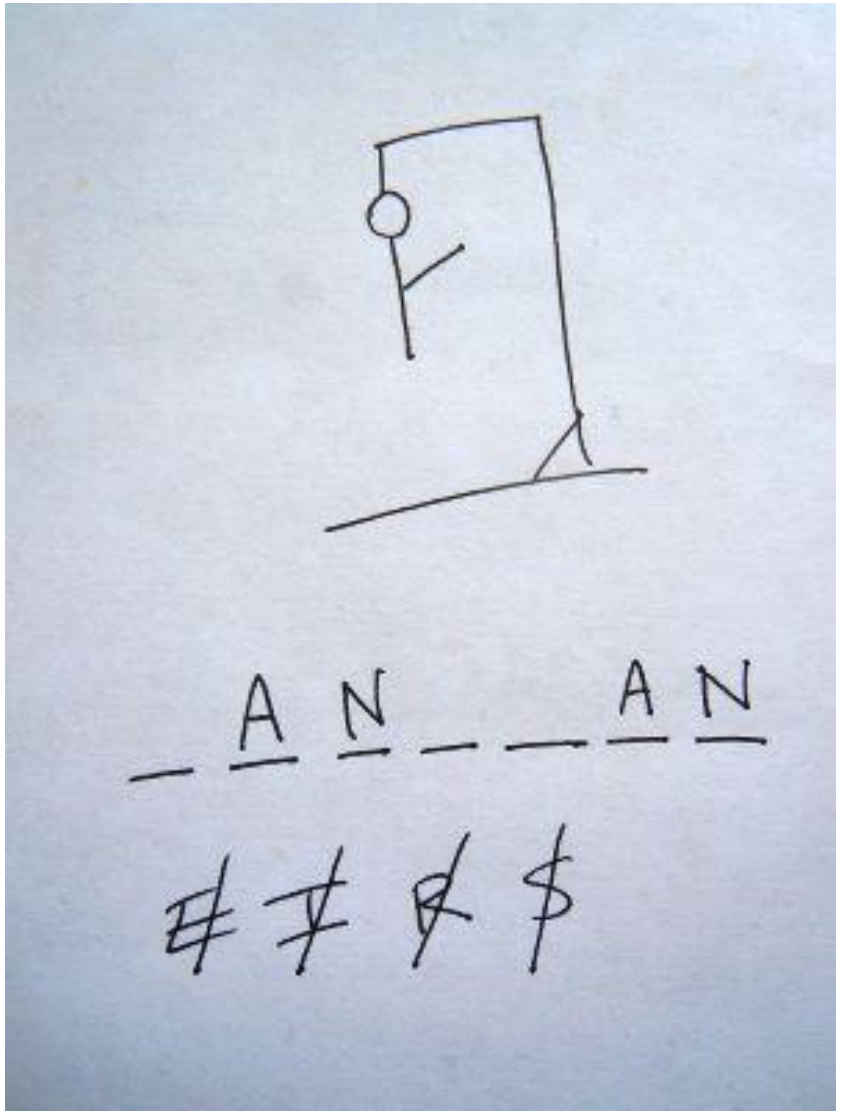
Leet speak replaces the letters I with 1, the letters Z with 2, the letters E with 3, the letters A with 4, the letters S with 5, the letters T with 7, the letters B with 8, the letters O with 0.

For instance, the English sentence "HELLO THERE, HOW ARE YOU?" will write in leet speak as "H3LL0 7H3R3, H0W 4R3 Y0U?".

- Write a function **eng_to_leet().**
- It receives a single parameter, **eng_str**, which corresponds to a sentence in English, written in capital letters.
- It returns the equivalent sentence in leet speak.
- Do so by modifying the letters with numbers, by following the rules we defined earlier.

Activity 3 – Hangman

The hangman is a game where one player thinks of a word, phrase or sentence and the other(s) tries to guess it by suggesting letters within a certain number of guesses.



Activity 3 – Hangman

Write a function **hangman()**, which receives four arguments:

- hidden_word: a hidden word that the player must guess. Normally, this word would be hidden from the player, but we will show it for now, as a way to check our function hangman() works as intended.
- proposed_letter: a single letter string, corresponding to a letter, which the player just proposed.
- letters_list: a list of letters, containing the previous propositions made by the user.

This function returns three arguments:

- revealed_word: The revealed word consists of the hidden_word, where the letters in the parameters proposed_letter and letters_list are shown. the other letters are replaced with star (*) characters.

For instance, if the parameters are

- hidden_word = "bulbazaur",
- proposed_letter = "u",
- and letters_list = ["e", "a", "r"] are passed to the function hangman(),

Then it will return revealed_word = "*u**a*aur". Note that all occurrences of the proposed letters should be displayed.

Boolean comparisons on strings

- As you have seen earlier, we can use **==** (resp. **!=**) to check if two strings are identical (resp. different).
- The comparison operators (**>**, **<**, **>=**, **<=**) can also be used.
- These comparison operators compare the alphabetical order between strings.

```
1  string1 = "Matt"
2  string2 = "Chris"
3  string3 = "Oka"
4  string4 = "Matt"
5  bool1 = (string1 == string2)
6  print("bool1: ", bool1)
7  bool2 = (string1 == string4)
8  print("bool2: ", bool2)
9  bool3 = (string1 > string2)
10 print("bool3: ", bool3)
11 bool4 = (string1 <= string3)
12 print("bool4: ", bool4)
```

```
bool1:  False
bool2:  True
bool3:  True
bool4:  True
```

Strings formatting with format()

- We have seen how the **format()** method could be used to insert the values of some variables inside a string. It can operate with indexes or even keywords.
- Some arguments can also be passed to the placeholders {}, but this is out of the scope of this class.
- https://docs.python.org/3/library/string.html#formatstrings

```
1  # Minimal format
2  print("Your {} is now level {}. Well done {}!".format("Pikachu", 15, "Pikachu"))
3  # Indexed placeholders
4  print("Your {1} is now level {0}. Well done {1}!".format(15, "Pikachu"))
5  # Keyword arguments
6  print("Your {name} is now level {level}. Well done {name}!".format(name = "Pikachu", level = 15))
```

```
Your Pikachu is now level 15. Well done Pikachu!
Your Pikachu is now level 15. Well done Pikachu!
Your Pikachu is now level 15. Well done Pikachu!
```

F-strings

- The latest versions of Python (3.7+) came with **F-strings:** the **F-strings** are strings, but they are preceded by the letter **F**, to indicate that it contains some placeholders **{}**.
- In F-strings, you can directly pass some variable names **inside** the placeholders, which is simpler than using **format()**.

```
1  # F-strings
2  name = "Pikachu"
3  level = 15
4  fstring = F"Your {name} is now level {level}."
5  print(fstring)
6  print(type(fstring))
```

```
Your Pikachu is now level 15.
<class 'str'>
```

Lists <-> Strings conversions

- Because lists and strings are roughly the same thing, it is sometimes convenient to convert one into the other.
- **Str -> List:** easily done with **list()**, which gives a list of characters.

```
1  # String to list conversion
2  so_noisy = "PIKAPI!"
3  print(so_noisy)
4  noise_as_list = list(so_noisy)
5  print(noise_as_list)
```

```
PIKAPI!
['P', 'I', 'K', 'A', 'P', 'I', '!']
```

Lists <-> Strings conversions

- **List -> Str:** done with the **join()** method, often applied on an empty string “”.

- Applying it on something else than an empty string leads to… interesting results.

```
1  # List of strings joining
2  print(noise_as_list)
3  noise_as_string = ''.join(noise_as_list)
4  print(noise_as_string)
5  # List of strings joining
6  HELL_NO = "PIKA".join(noise_as_list)
7  print(HELL_NO)
```

```
['P', 'I', 'K', 'A', 'P', 'I', '!']
PIKAPI!
PPIKAIPIKAKPIKAAPIKAPPIKAIPIKA!
```

Splitting a string into a list of strings

- If needed you can split a large string into a list of smaller ones, with the **split()** method.
- By default, it returns a list of the different words in the string.
- You can also specify a substring to use as a separator.
- Typically convenient for dates (14/10/2020), hours (14:52:26), etc.

```
1  # Splitting a string
2  so_annoying = "PIKA PIKA PIKA? PIKAPI PIKA PIKA PI!"
3  print(so_annoying)
4  print(so_annoying.split())
5  # Splitting a string
6  my_birthday = "02-Mar-1989"
7  print(my_birthday)
8  print(my_birthday.split("-"))
```

```
PIKA PIKA PIKA? PIKAPI PIKA PIKA PI!
['PIKA', 'PIKA', 'PIKA?', 'PIKAPI', 'PIKA', 'PIKA', 'PI!']
02-Mar-1989
['02', 'Mar', '1989']
```

Activity 4 - Create password from sentence

A **passphrase** is a sentence which can be used to create passwords that are difficult to crack and easy to memorize.

The user simply needs to create a memorable sentence containing multiple words.

Later on, the user will assemble a password by alternating the first letter of each word in the passphrase and the number of characters in it. In addition, add the special characters when they appear.

Activity 4 - Create password from sentence

A **passphrase** is a sentence which can be used to create passwords that are difficult to crack and easy to memorize.

The user simply needs to create a memorable sentence containing multiple words.

Later on, the user will assemble a password by alternating the first letter of each word in the passphrase and the number of characters in it. In addition, add the special characters when they appear.

For instance, the sentence

"I used to be an adventurer like you until I took an arrow to the knee“

can be used to generate the following password

"I1u4t2b2a2a10l4y3u5I1t4a2a5t2t3k4“

Activity 4 - Create password from sentence

A **passphrase** is a sentence which can be used to create passwords that are difficult to crack and easy to memorize.

The user simply needs to create a memorable sentence containing multiple words.

Later on, the user will assemble a password by alternating the first letter of each word in the passphrase and the number of characters in it. In addition, add the special characters when they appear.

For instance, the sentence

"I used to be an adventurer like you until I took an arrow to the knee“

can be used to generate the following password

"I1u4t2b2a2a10l4y3u5I1t4a2a5t2t3k4“

Write a function **generate_password()**, which receives a passphrase as its only parameter, and returns a single output, consisting of a string generated according to the method we described.

Characters types check in strings

You can check for characters types in a string with 3 methods.

- **isalpha()** returns **True** is characters are letters only; and **False** otherwise.
- **isdigit()** returns **True** if the characters are digits only; and **False** otherwise.
- **isalnum()** returns **True** if the characters are letters and digits only; and **False** otherwise.

```
1  # The isalpha() method
2  my_first_name = "Matt"
3  my_last_name = "De Mari"
4  print(my_first_name.isalpha())
5  print(my_last_name.isalpha())
```

```
True
False
```

```
1  # The isdigit() method
2  my_SG_phone = "00000000"
3  my_FR_phone = "+33 0000000000"
4  print(my_SG_phone.isdigit())
5  print(my_FR_phone.isdigit())
```

```
True
False
```

```
1  # The isalnum() method
2  my_age = "31years" # Sigh
3  my_email = "best_teacher_in_SUTD@sutd.edu.sg"
4  print(my_age.isalnum())
5  print(my_email.isalnum())
```

```
True
False
```

Activity 5 – Password checker

Write a function **password_check()**, which receives a single parameter, password, consisting of a string variable.

This function returns a boolean with value True if:

- the password has at least 8 characters,
- the password contains both letters and numbers,
- and the password contains at least one special character (i.e. a non-number, non-letter character).

Otherwise, the function returns False.

As usual, RTFM!

- Strings are one of the most furnished types in Python, with many more methods.

- Read more about strings, here!

https://docs.python.org/2/library/string.html

Conclusion

- The string type (and similarities with the list type)
- Indexing, slicing, and traversing a string
- Updating a string and unmutability
- Membership and find
- Booleans on strings, sorting
- Printing and formatting
- Strings <-> lists conversions
- Splitting, character checking
- (More stuff, RTFM!)

The find() method

- The strings come with an additional method, called **find()**, which looks for a character or substring inside another string.
- It returns the index of the first appears of said character or substring inside the string.
- If the character or substring does not appear, it returns -1.

```
1  # Find method in strings
2  a_string = "Pikapikapikachu"
3  print(a_string.find("k"))
4  print(a_string.find("p"))
5  print(a_string.find("chu"))
6  print(a_string.find("swag"))
```

```
2
4
12
-1
```

Sorting strings and lists of strings

- We have seen earlier how the **sorted()** function could be used to sort a list of strings alphabetically.

- It can also be applied on a single string. This produces a list of characters, in alphabetical order.

```
1  # Sorting strings and lists of strings
2  string1 = "BDEFGAC"
3  print(sorted(string1))
4  strings_list = ["Matt", "Chris", "Oka"]
5  print(sorted(strings_list))
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G']
['Chris', 'Matt', 'Oka']
```

Additional: functions, strip-cleaning a string

- Strip-cleaning a string removes the extra spaces, if any, at the beginning and end of a string.

- If a substring is passed to the **split()** method, it will remove the occurrences of these extra characters at the beginning and end of the string.

```
1  # Stripping a string
2  the_ISS = "      This string is floating in spaces     "
3  print(the_ISS)
4  print(the_ISS.strip())
5  # Stripping a string
6  allergic_to_stars = "***COUGH**COUGH******"
7  print(allergic_to_stars)
8  print(allergic_to_stars.strip("*"))
```

```
      This string is floating in spaces
This string is floating in spaces
***COUGH**COUGH******
COUGH**COUGH
```

As usual, RTFM!

- Strings are one of the most furnished types in Python, with many more methods.

- Read more about strings, here!

https://docs.python.org/2/library/string.html