Python: Strings

1405 Instructor: Ruiqing (Sam) Cao

Sequence Types: String (str)

- Enclosing in either double (`""`) or single (`''`) quote **`"Hello world!"`** and **`'Hello world!'`** *completely equivalent*

- Almost *anything you type on the keyboard* can be part of a str: letters (A-z,å,ñ,...), digits, special characters, white space

- Python *does not have a Character type* in contrast to C or Java; a character are simply stored as a `str` type of length 1

String Comparison

- String comparisons follow **lexicographic order**
- This means `st1 < st2` if `st1` appears before `st2` in the dictionary using the ASCII table (where 0-9 are before A-Z, and A-Z are before a-z; special characters are more complex, but check ASCII table)
- For example, the following expressions all evaluate to **`True`**

```
'ZZ'<'a'
'111'<'A'
'2024-12-30'<'2025-01-02'
```

String Creation, Indexing, & Slicing

- *Create* a string

  `st="Hello world!"` or `st='Hello world!'`

- String *indexing* (accessing a character at a position)

  `st[4]` → `'o'` (position *starts from 0*)

- String *slicing* (extract a substring)

  `st[6:11]` → `'world'` (from *begin position* to *end position -1*)

In Python, indexes *Start at 0* (not 1) !

Smartest people ranked:

12. You
11. Can't
10. Rank
9. Them
8. Because
7. People
6. Are
5. All
4. Smart
3. In
2. Different
1. Ways
0. *Programmers*

String Length & Concatenation

- String *length* (number of characters in it)

```
len('Hello world!')
```
→ **12**

- *Combine* a few strings (using operator “+”)

```
'Hell'+'o '+'world!'
```
→ **'Hello world!’**

- *Repeat* the same string multiple times

```
3*'dare'
```
→ **'daredaredare'**

Check for Substring

First, create a string and store it: `st = 'Hello world!'`

- Substring *existence*

```
'world' in st → True
```

- Substring at the *beginning*

```
st.startswith('Hell') → True
```

- Substring at the *end*

```
st.endswith('!') → True
```

Replace Substrings

- *Replace* all the substrings that equals the indicated pattern
  `'i tzee tznow'.replace('tz','s')` -> `'i see snow'`

For instance, remove all the white spaces from a string `st`

`st.replace(' ','')`

String Splitting & Joining

- *Split* a string separated by a token into a list

```
'i see  snow'.split(' ')→ ['i','see','','snow']
```

If no token given, split by white space and ignore empty string

```
'i see  snow'.split()→ ['i','see','snow']
```

- *Join* elements in a list into a string (reversal of split)

```
' '.join(['i','see','','snow'])→ 'i see  snow'
```

→ The token can be any string (not necessarily `' '`)

String Trimming

- *Remove* leading & trailing *whitespaces*

```
'  hello  '.strip()  → 'hello'
```

- Remove specific *characters*

```
'  hello  '.strip(' ho')  → 'ell'  (remove ' ', 'h', 'o')
```

- Remove *leading characters* (*to the left*) only

```
'  hello  '.lstrip(' ho')  → 'ello  '
```

- Remove *trailing characters* (*to the right*) only

```
'  hello  '.rstrip(' ho')  → '  hell'
```

String Lowercasing & Uppercasing

- Convert string to *lowercase*

```
'Hello world!'.lower()  → 'hello world!'
```

- Convert string to *uppercase*

```
'Hello world!'.upper()  → 'HELLO WORLD!'
```

String Encoding

- Sometimes when you read strings from large files, you need to *encode* them (the common choice is `'utf-8'`, occasionally `'latin-1'` if you are dealing with older files or systems)

```
encoded = st.encode('utf-8')
```

- *Decode* back an encoded string

```
decoded = encoded.decode('utf-8')
```

String Formatting (f-strings)

- *F-strings* (formatted string literals) is a very useful way to *include variables and expressions* directly within a string
  - Requires Python 3.6 or later version

- How it works: prefixed with the letter f or F, " " or ' ' to enclose the string, and embeds variables and expression inside curly braces { } within the string

- **Best option** for string formatting in Python (better than older methods): flexible, efficient, and user-friendly

String Formatting (f-strings)

- Embedding variables and expressions

For instance, here are some variables

```
a=1.1
b=2
C='duMMY'
```

Let’s see how to use f-strings to embed them and their expressions

```
f"a is {a}" → 'a is 1.1'
f"a+b equals {a+b}" → 'a+b equals 3.1'
f"lowercase C is {C.lower()}" → 'lowercase C is dummy'
```

String Formatting (f-strings)

- Printing large numeric values can be ugly – formatting them appropriately helps (no need to show pi=3.141592653589793 if only the first two decimal places are necessary 3.14)

Some very common use cases

➢Print only a few decimal places: `pi=3.141592653589793` is `float`

`f"pi equals: {pi:.2f}"` → `f"pi equals: {pi:.2f}"`

➢Add leading zeros to same length (e.g., region code) : `jj=680` is `int`

`f"Jonkoping {jj:04}"` → `'Jonkoping 0680'`

➢Print the thousand separators: `bignum=1000234.5` is `float`

`f"big number is {bignum:,}"` → `'big number is 1,000,234.5'`

Exercises: Strings

- Write a Python program to:

1) Ask the user to input a word.
2) Remove leading and trailing white spaces and converts the word to all lowercase.
3) Check if the word is a palindrome (reads the same forwards and backwards).
4) Print True or False.

- Write a Python program to:

1. Input an English sentence (e.g., “Hello world!” and converts to all lowercase.
2. Replace all non-alphabetic characters with white spaces. Splits sentence by white space into **a list** of strings.
3. Ask the user to input a search term.
4. Check if the term (case insensitive) is in the list and prints an appropriate message.
5. Create another list containing only words with more than 5 letters from the list