

Lecture 11

Topics: *Chapter 5. Computing with strings*

5.1 The String Data Type

5.2 Simple String Processing

5.3 Lists as Sequences

5.1 String data type

Text is represented in programs by the *string* data type.

Think of string as a sequence of characters (symbols).

```
>>> string1='Hello'  
>>> string2='Hello'  
>>> type(string1)  
<class 'str'>  
>>> type(string2)  
<class 'str'>
```

5.1 String data type

Inputting strings:

Use input function: `input`

- Doesn't evaluate the expression the user enters.
(use `raw_input` in Python 2)

Example:

```
def main()
    f_name = input('Enter your name, please:')
    print('Good day, ', f_name)
```

```
main()
```

If we run the program and type in name Caroline, then we'll get:

`Good day, Caroline`

5.1 String data type

String is a sequence of characters/symbols, thus we can access the individual characters/symbols through *indexing*:

I	T		I	S		S	U	N	N	Y		T	O	D	A	Y
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

indexing is used in string expressions to access a specific character position in the string.

Syntax:

```
<string>[<expression>]
```

5.1 String data type

Type the following in the interactive window:

```
>>> phrase = 'It is sunny today'
```

```
>>> phrase[0]
```

```
'I'
```

```
>>> phrase[2]
```

```
'i'
```

```
>>> phrase[10]
```

```
'y'
```

Positive indexing: from the left end to the right end

Negative indexing: from the right end to the left end

```
>>> phrase[-1]
```

```
>>> phrase[-5]
```

5.1 String data type

It is also possible to access a contiguous sequence of characters (**substring**) using the colon operator (:)

Syntax:

```
<string>[<start>:<end>]
```

```
>>> phrase = 'It is sunny today'
```

```
>>> phrase[3:10]
```

```
'is sunn'
```

```
>>> phrase[:3]
```

```
'It '
```

```
>>> phrase[11:]
```

```
' today'
```

NOTE: negative numbers count from the end:

```
>> phrase[11:-2]
```

```
' tod'
```

5.1 String data type

There is also a **double colon operator (::)**:

Syntax:

```
<string>[<start>:<end>:<step>]
```

```
>>> phrase = 'It is sunny today'  
>>> phrase[0:10:2]  
'I ssn'
```

Or backwards!

```
>>> phrase[::-1]  
'yadot ynnus si tI'
```

In all slots, when empty, it means the default value:

```
0 for <start>,  
The length of the string for <end>  
1 for <step>
```

5.1 String data type

More operations with strings:

Concatenation: **+** Example: `string1+string2`

Repetition: ***** Example: `string1*string2`

Length of a string: **len** Example: `len(string1)`

Iteration through characters: **for <var> in <string>**
Example: `for i in 'Hello John'`

In a Python shell, write
`string1 = "Whatever you like"`
`for ch in string 1:`
 `print(ch)`

5.2 Simple string processing

Let's write a program that will be printing out the name of the month in short form.

In other words, the user will enter the number of the month, and our program will output the month in short form.

Recall the months names and their numbers:

1	Jan	7	Jul
2	Feb	8	Aug
3	Mar	9	Sep
4	Apr	10	Oct
5	May	11	Nov
6	Jun	12	Dec

5.2 Simple string processing

Let's write a program that will be printing out the name of the month in short form.

In other words, the user will enter the number of the month, and our program will output the month in short form.

Recall the months names and their numbers:

1	Jan	7	Jul
2	Feb	8	Aug
3	Mar	9	Sep
4	Apr	10	Oct
5	May	11	Nov
6	Jun	12	Dec

What Python's tools to use? **1.** Use if-elif-else **or** **2.** Use strings

5.2 Simple string processing

1. Use if-elif-else statements.

Algorithm:

```
input the number of the month (n)
if n=1: output Jan
elif n=2: output Feb
elif n=3: output Mar
...
elif n=12: output Dec
else : wrong number of the month
```

5.2 Simple string processing

2. Use strings

```
months='JanFebMarAprMayJunJulAugSepOctNovDec'
```

Algorithm:

Input the number of a month (n)

Output the slice/piece of the months string:

3 characters long,

Starting from $((n-1)*3)^{\text{th}}$ position

For example:

To get Jan (n=1), `print(months[0:3])`

To get Feb (n=2), `print(months[4:6])`

To get Mar (n=3), `print(months[7:9])`

(what is the pattern?)

5.3 Lists as Sequences

List data type in Python represents a sequence of elements, which are values of any Python data type.

We can work with *lists of strings*, *list of integers*, *list of values of mixed data type*, etc.

Syntax: [list of elements separated by commas]

Examples: [1, 2, 3, 4, 5, 6]
[1, 'a', "Hi, how are you?", 132, 5.6]

All the string operations listed before are applicable to sequences (lists).

5.3 Lists as Sequences

Type the following in the interactive window:

```
>>> [1,5] + [2,8]
[1, 5, 2, 8]
```

```
>>> [1,5]*4
[1, 5, 1, 5, 1, 5, 1, 5]
```

```
>>> list_of_grades=['A', 'B', 'C', 'D', 'F']
```

```
>>> list_of_grades[0]
'A'
```

```
>>> list_of_grades[3]
'D'
```

```
>>> list_of_grades[2:4]
['C', 'D']
```

```
>>> len(list_of_grades)
5
```

5.3 Lists as Sequences

Lists are more general than strings: they can be sequences of arbitrary values, not just characters

```
myList=[1, ' 'January' ',2, ' 'February' ',3, ' 'Hello' ' ]
```

Using lists we can re-write our program for months and make it easier to retrieve months full names by their number:

```
months=["January", "February", "March", "April",  
"May", "June", "July", "August", "September",  
"October", "November", "December"]
```

Now it is easy!

```
n = eval(input("Enter the number of the month"))  
print(months[n-1])
```

see program [months_lists.py](#)

5.3 Lists as Sequences

! Lists are **mutable**, i.e. the value of an item in a list can be modified with an assignment statement.

Example:

```
myList=[1, ' 'Thank you' ',5]  
myList[2]=' 'Hello' '
```

the resulting list: [1,"Thank you","Hello"]

On the contrary, **strings are NOT mutable!**

Try to write in a shell:

```
string1 = "Whatever"  
string1[0] = "H"
```


5.3 Lists as Sequences

Matrices (lists of lists)

[Def] A **matrix** is a rectangular array of numbers.
A matrix with m rows and n columns is called **$m \times n$ matrix**.

Plural form: **matrices**

Notation: **$A_{m \times n}$**

Matrix with $m = n$ is called **square matrix**

$$A = \begin{bmatrix} 1 & 4 & -9 \\ 0 & 1 & 3 \\ 7 & -2 & 8 \\ 0 & -1 & 5 \end{bmatrix}$$

← *matrix with 4
rows and 3
columns*

5.3 Lists as Sequences

Matrices (lists of lists)

a_{ij} – element of matrix in row i and column j

$$A = \begin{matrix} & \begin{matrix} 1^{\text{st}} & 2^{\text{nd}} & 3^{\text{rd}} \end{matrix} \\ \begin{matrix} 1^{\text{st}} \\ 2^{\text{nd}} \\ 3^{\text{rd}} \\ 4^{\text{th}} \end{matrix} & \begin{bmatrix} 1 & 4 & -9 \\ 0 & 1 & 3 \\ 7 & -2 & 8 \\ 0 & -1 & 5 \end{bmatrix} \end{matrix}$$

5.3 Lists as Sequences

Matrices (lists of lists)

a_{ij} – element of matrix in row i and column j

$$A = \begin{array}{ccc|c} & \text{1}^{\text{st}} & \text{2}^{\text{nd}} & \text{3}^{\text{rd}} \\ \begin{array}{c} \text{1}^{\text{st}} \\ \text{2}^{\text{nd}} \\ \text{3}^{\text{rd}} \\ \text{4}^{\text{th}} \end{array} & \begin{bmatrix} 1 & 4 & -9 \\ 0 & 1 & 3 \\ 7 & -2 & 8 \\ 0 & -1 & 5 \end{bmatrix} & & \end{array}$$

$$a_{12} = 4$$

$$a_{21} = 0$$

$$a_{42} = -1$$

5.3 Lists as Sequences

Matrices in Python

$$A = \begin{bmatrix} 1 & 4 & -9 \\ 0 & 1 & 3 \\ 7 & -2 & 8 \\ 0 & -1 & 5 \end{bmatrix}$$

0th 1st 2nd
0th 1st 2nd 3rd

$$\begin{array}{l} a_{12} = 4 \\ a_{21} = 0 \\ a_{42} = -1 \end{array} \longrightarrow \begin{array}{l} a_{01} = 4 \\ a_{10} = 0 \\ a_{31} = -1 \end{array}$$

In Python interactive window:

```
>>> A = [ [1,4,-9], [0,1,3], [7,-2,8], [0,-1,5] ]
```

```
>>> A[0][1]
```

```
4
```

```
>>> A[1][0]
```

```
0
```