

## CSI31 Lecture 10

**Topics:** *Chapter 4. Computing with strings*

4.1 String data type

4.2 Simple string processing

4.3 Strings, lists and sequences

**HW# 8** (due date: Wednesday, October 29th):

page 118 programming exercise 4.

Take care of the cases when a user inputs a number  $> 100$  or a negative number, or inputs a character. The later one can be done with exception handling.

## 4.1 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)  
>>> type(string2)
```

### How to input strings:

use another input function: *raw-input*

– doesn't evaluate the expression the user enters.

#### Example:

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

```
main()
```

when run the program: (if we input name Caroline)

```
Good day, Caroline
```

The old way won't work properly:

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

```
main()
```

we will get *NameError*. This program will work only if the user inputs the name in single or double quotes.  
"Caroline" or 'Caroline'

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>]`

Type the following in the interactive window:

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

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

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

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

*positive indexing*: from the left end to right end

*negative indexing*: from the right end to the left end

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

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

It is also possible to access a contiguous sequence of characters ([substring](#)):

syntax:

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

Continue working with the string variable phrase (`phrase = 'It is sunny today'`):

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

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

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

### Operations with strings:

concatenation:	+	example: string1+string2
repetition:	*	example: string1*string2
length of a string:	<code>len</code>	example: len(string1)

iteration through characters: `for <var> in <string>`  
example: for i in 'Hello John'

see example program: [string\\_operations.py](#)

## 4.2 Simple string processing

Let's write a program that will be printing out the months name given the number of the month. In other words, the user will enter the number of the month, and our program will output the months name.

Let's recall the months names and their numbers:

1	January	7	July
2	February	8	August
3	March	9	September
4	April	10	October
5	May	11	November
6	June	12	December

How to write a program? What Python's tools to use?

1. Use if-elif-else
2. Use strings

1. Use if-elif-else statements.

*Algorithm:*

```
input the number of the month (n)
if n=1    output January
elif n=2  output February
elif n=3  output March
...
elif n=12 output December
else      you inputed the wrong number of the month
```

## 2. Use strings

the longest months name is September: 9 characters

Store the names of months in one string (each months name should occupy 9 slots):

```
months='January February March April May June July August SeptemberOctober  
November December '
```

*Algorithm:*

input the number of a month (n)

output the slice/piece of the months string (9 characters long, starting from  $((n-1)*9)^{\text{th}}$  position)

see program [months\\_string.py](#)

## 4.3 Strings, Lists, and Sequences

We can work with lists of strings, which are also a kind of sequence.

Recall the syntax for lists: **[list of elements separated by commas]** (for example: [1,2,3,4,5,6])

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

Type the following in the interactive window:

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

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

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

```
>>> list_of_grades[0]
```

```
>>> list_of_grades[3]
```

```
>>> list_of_grades[2:4]
```

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

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:

see program [months\\_lists.py](#)

! 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"]