

Fundamentals of Programming for Artificial Intelligence

Session 04
String & Text File

Instructors:

Dr. Lê Thanh Tùng

Dr. Nguyễn Tiến Huy

Content

- 1 String
- 2 Text file

1. String

String

- A string is a sequence of characters
- Using single quotes or double quotes to represent a string in Python

```
# create a string using double quotes
string1 = "Python programming"

# create a string using single quotes
string1 = 'Python programming'
```

Access String Characters

- Indexing: positive and negative index (like List)

```
greet = 'hello'  
  
# access 1st index element  
print(greet[1]) # "e"
```

```
greet = 'hello'  
  
# access 4th last element  
print(greet[-4]) # "e"
```

- Slicing

```
greet = 'Hello'  
  
# access character from 1st index to 3rd index  
print(greet[1:4]) # "ell"
```

Strings are immutable

- That means the characters of a string cannot be changed

```
message = 'Hola Amigos'  
message[0] = 'H'  
print(message)
```

```
Traceback (most recent call last):  
  File "<string>", line 2, in <module>  
TypeError: 'str' object does not support item assignment
```

- However, we can assign the variable name to a new string

```
message = 'Hola Amigos'  
  
# assign new string to message variable  
message = 'Hello Friends'  
  
prints(message); # prints "Hello Friends"
```

Multiline String

- We can also create a multiline string in Python by using triple quotes

```
# multiline string  
message = """  
Never gonna give you up  
Never gonna let you down  
"""  
  
print(message)
```

Output

```
Never gonna give you up  
Never gonna let you down
```

String operations

- "+": concatenate strings

```
greet = "Hello, "  
name = "Jack"  
  
# using + operator  
result = greet + name  
print(result)  
  
# Output: Hello, Jack
```

- "%":
formatting

```
print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

```
My name is Zara and weight is 21 kg!
```

- "*": Repetition

```
s = "120"  
print(s*2)  
  
#output: 120120
```


String comparison

Declare the string variable:

```
fruit1 = 'Apple'
```

String is

- Case-sensitive
- A (65) < a (97)

Operator	Code	Output
Equality	<code>print(fruit1 == 'Apple')</code>	True
Not equal to	<code>print(fruit1 != 'Apple')</code>	False
Less than	<code>print(fruit1 < 'Apple')</code>	False
Greater than	<code>print(fruit1 > 'Apple')</code>	False
Less than or equal to	<code>print(fruit1 <= 'Apple')</code>	True
Greater than or equal to	<code>print(fruit1 >= 'Apple')</code>	True

String comparison

String comparison is based on character comparison which depends on ASCII code

```
string1 = "Abrar"
string2 = "Ahmed"
string3 = "ABCD"
string4 = "ABCD"

if string1 <= string2:
    print(string1, " is smaller ", string2, " is greater")
if string2 >= string4:
    print(string4, " is smaller ", string2, " is greater")
if string3 == string4:
    print(string3, " is equal to ", string4)
if string1 != string3:
    print(string1, " is not equal to ", string3)
```

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i

Basic Operations

<u>isalnum()</u>	Returns True if all characters in the string are alphanumeric
<u>isalpha()</u>	Returns True if all characters in the string are in the alphabet
<u>isascii()</u>	Returns True if all characters in the string are ascii characters
<u>isdecimal()</u>	Returns True if all characters in the string are decimals
<u>isdigit()</u>	Returns True if all characters in the string are digits
<u>isidentifier()</u>	Returns True if the string is an identifier
<u>islower()</u>	Returns True if all characters in the string are lower case
<u>isnumeric()</u>	Returns True if all characters in the string are numeric
<u>isprintable()</u>	Returns True if all characters in the string are printable
<u>isspace()</u>	Returns True if all characters in the string are whitespaces
<u>isupper()</u>	Returns True if all characters in the string are upper case

Basic Operations

Convert string into lower, upper:

lower() Converts a string into lower case

capitalize() Converts the first character to upper case

upper() Converts a string into upper case

```
message = 'python is fun'

# convert message to uppercase
print(message.upper())

# Output: PYTHON IS FUN
```

```
message = 'PYTHON IS FUN'

# convert message to lowercase
print(message.lower())

# Output: python is fun
```

```
sentence = "i love PYTHON"

# converts first character to uppercase and others to lowercase
capitalized_string = sentence.capitalize()

print(capitalized_string)

# Output: I love python
```

Some important functions

- split(): https://www.w3schools.com/python/ref_string_split.asp
- join(): https://www.w3schools.com/python/ref_string_join.asp
- strip(): https://www.w3schools.com/python/ref_string_strip.asp
- find(): https://www.w3schools.com/python/ref_string_find.asp
- startswith(): https://www.w3schools.com/python/ref_string_startswith.asp
- rstrip(): https://www.w3schools.com/python/ref_string_rstrip.asp
- zfill(): https://www.w3schools.com/python/ref_string_zfill.asp
- index(): https://www.w3schools.com/python/ref_string_index.asp
- count(): https://www.w3schools.com/python/ref_string_count.asp
- endswith(): https://www.w3schools.com/python/ref_string_endswith.asp

Exercise

- Count all letters, digits, and special symbols from a given string

```
Input string= P@yn2at&dfsdfd&^^^&^^&^#i5ve
```

```
total counts of chars, Digits, and symbols
```

```
Chars = 14 Digits = 2 Symbol = 12
```

Exercise

- Given two strings, s1 and s2. Write a program to create a new string s3 by appending s2 in the middle of s1

Given:

```
s1 = "Ault"  
s2 = "Kelly"
```

Expected Output:

```
AuKellylt
```

Exercise

- Arrange string characters such that lowercase letters should come first

Given:

```
str1 = PyNaTive
```

Expected Output:

```
yaivePNT
```


Exercise

- Removal all characters from a string except integers

Given:

```
str1 = 'I am 25 years and 10 months old'
```

Expected Output:

```
2510
```

Exercise

- Remove empty strings from a list of strings

Given:

```
str_list = ["Emma", "Jon", "", "Kelly", None, "Eric", ""]
```

Expected Output:

```
Original list of sting  
['Emma', 'Jon', '', 'Kelly', None, 'Eric', '']
```

```
After removing empty strings  
['Emma', 'Jon', 'Kelly', 'Eric']
```

Exercise

- Split a string on hyphens

Given:

```
str1 = Emma-is-a-data-scientist
```

Expected Output:

```
Displaying each substring
```

```
Emma
```

```
is
```

```
a
```

```
data
```

```
scientist
```

Exercise

- Write a program that reads a string from the user. If all the characters in the string are hexadecimal digits, print out the corresponding decimal value. If not, print out an error message
- 1F => 31
- 1G => "Error"
- 1f => 31

Exercise

- Write a function to count the occurrence of all words (non-case sensitive) in the string. Known that each word is separated by a space

Exercise

- You should write functions so that print out the meaningful integer number in s. Known that this number is separated by the other characters and maybe starts with many "0". Return -1 if there is not a number in s.

Sample input and output:

Input: abc0001vcs234as

Output for marking: 0001234 => 1234 (eliminate "0" in the beginning of number)

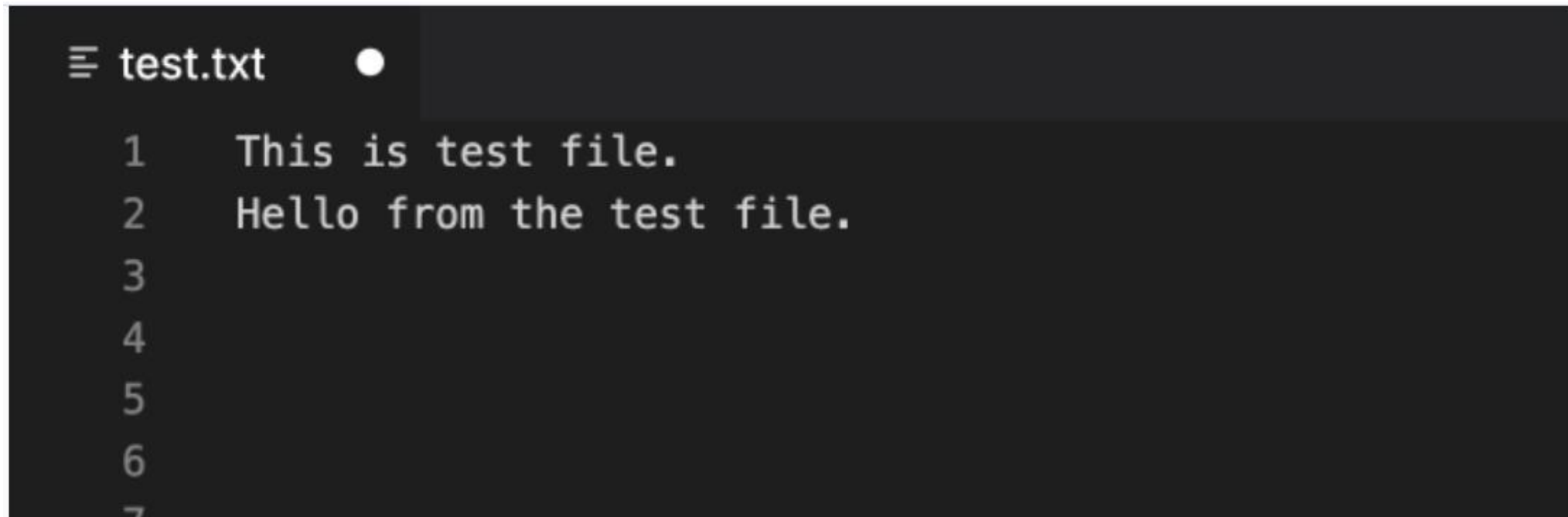
OUTPUT:

1234|

2. Text File

Text File

- Python provides basic functions and methods necessary to manipulate files by default.
- You can do most of the file manipulation using a **file** object.



```
≡ test.txt ●  
1 This is test file.  
2 Hello from the test file.  
3  
4  
5  
6  
7
```


Open a file

- Before you can read or write a file, you have to open it using the *open()* function.

`file = open(file_name [, access_mode][, buffering])`

- **access_mode** – read, write, append etc (r,w,a,...)
- **buffering** –
 - buffering = 0, no buffering takes place
 - buffering = 1, line buffering is performed while accessing a file.
 - buffering > 1, buffering action is performed with the indicated buffer size.
 - buffering < 0, the buffer size is the system default (default behavior).

Access Mode

Mode	Description
<code>r</code>	Open a file for reading. (default)
<code>w</code>	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
<code>x</code>	Open a file for exclusive creation. If the file already exists, the operation fails.
<code>a</code>	Open a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
<code>t</code>	Open in text mode. (default)
<code>b</code>	Open in binary mode.
<code>+</code>	Open a file for updating (reading and writing)

Closing Files

- The `close()` method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.
- Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the `close()` method to close a file.

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name

# Close opened file
fo.close()
```

Attributes related to file object

Sr.No.	Attribute & Description
1	file.closed Returns true if file is closed, false otherwise.
2	file.mode Returns access mode with which file was opened.
3	file.name Returns name of the file.
4	file.softspace Returns false if space explicitly required with print, true otherwise.

Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result –

```
Name of the file:  foo.txt
Closed or not :   False
Opening mode :    wb
Softspace flag :  0
```

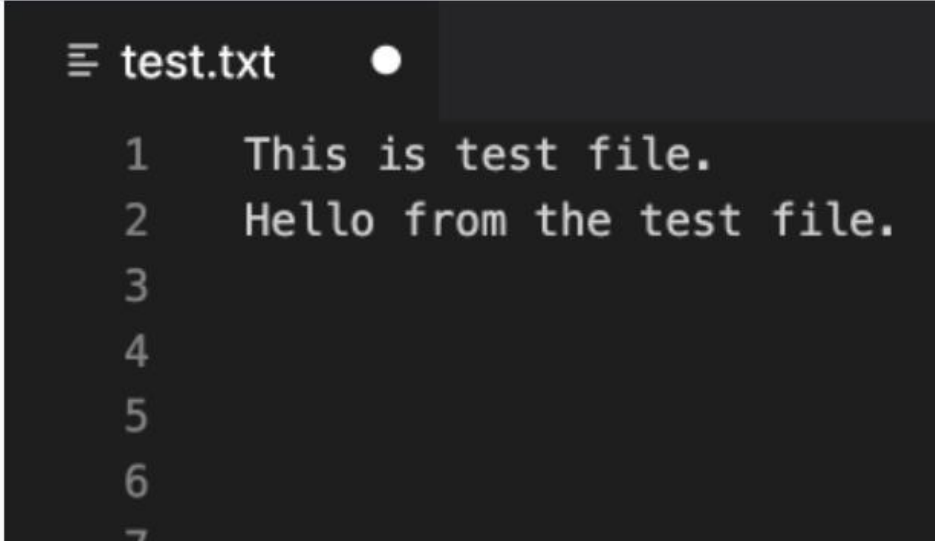
Example

```
# open a file
file1 = open("test.txt", "r")

# read the file
read_content = file1.read()
print(read_content)
```

Output

```
This is a test file.
Hello from the test file.
```



```
test.txt
1 This is test file.
2 Hello from the test file.
3
4
5
6
7
```

Exception Handling in Files

- If an exception occurs when we are performing some operation with the file, the code exits without closing the file
- Using try/except/...

```
try:
    file1 = open("test.txt", "r")
    read_content = file1.read()
    print(read_content)

finally:
    # close the file
    file1.close()
```

with ... open Syntax

- In Python, we can use the **with...open** syntax to automatically close the file

```
with open("test.txt", "r") as file1:  
    read_content = file1.read()  
    print(read_content)
```


Reading Content from Files

Method	Argument
<code>read([n])</code>	Reads and returns <code>n</code> bytes or less (if there aren't enough characters to read) from the file as a string. If <code>n</code> not specified, it reads the entire file as a string and returns it.
<code>readline()</code>	Reads and returns the characters until the end of the line is reached as a string.
<code>readlines()</code>	Reads and returns all the lines as a list of strings.

read()

- The **read()** method returns the specified number of bytes from the file.
Default is -1 which means the whole file

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str
# Close open file
fo.close()
```

This produces the following result –

```
Read String is : Python is
```

- Passed parameter is the number of bytes to be read from the opened file.
- Starts reading from the beginning of the file
- Default - read as much as possible, maybe until the end of file.

readlines()

- Return all lines in the file, as a list where each line is an item in the list object

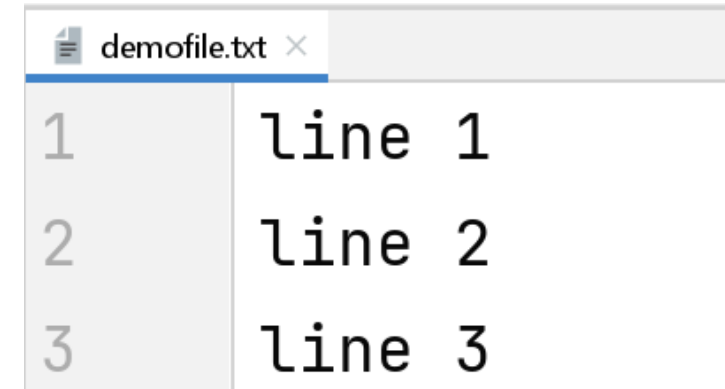
Syntax

```
file.readlines(hint)
```

Parameter Values

Parameter	Description
<i>hint</i>	Optional. If the number of bytes returned exceed the hint number, no more lines will be returned. Default value is -1, which means all lines will be returned.

```
f = open("demofile.txt", "r")
lines = f.readlines()
print(type(lines)) # <class 'list'>
print(lines) # ['line 1\n', 'line 2\n', 'line 3\n']
```

A screenshot of a text editor window titled 'demofile.txt'. The window displays three lines of text: 'line 1', 'line 2', and 'line 3', each on a new line. The lines are numbered 1, 2, and 3 on the left side of the editor.

1	line 1
2	line 2
3	line 3

readline()

Syntax

```
file.readline(size)
```

Parameter Values

Parameter	Description
<i>size</i>	Optional. The number of bytes from the line to return. Default -1, which means the whole line.

```
f = open("demofile.txt", "r")
line = f.readline()
print(type(line)) # <class 'str'>
print(line) # line 1
```

demofile.txt ×	
1	line 1
2	line 2
3	line 3

The readline() method

Example: Reading file line by line

```
1 >>>
2 >>> f = open("poem.txt", "r")
3 >>>
4 >>> while True:
5 ...     line = f.readline()
6 ...     if not line:
7 ...         break
8 ...     print(line)
9 ...
10 The caged bird sings
11 with a fearful trill
12 of things unknown
13 but longed for still
14 >>>
```

Writing to Files in Python

- There are two things we need to remember while writing to a file
 - If we try to open a file that doesn't exist, a new file is created
 - If a file already exists, its content is erased, and new content is added to the file (without append mode) with mode "w"

Method	Description
<code>write(s)</code>	Writes the string <code>s</code> to the file and returns the number characters written.
<code>writelines(s)</code>	Writes all strings in the sequence <code>s</code> to the file.

write()

- The `write()` method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.
- The `write()` method does not add a newline character ('\n') to the end of the string

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opened file
fo.close()
```

writelines()

- Writes the items of a list to the file

```
f = open("demofile3.txt", "a")
f.writelines(["\nSee you soon!", "\nOver and out."])
f.close()
```

```
#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

```
Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!
See you soon!
Over and out.
```


Exercise

- Retrieve a list of employees from the input.txt file and generate an employee list accordingly. If two employees share the same email abbreviation, assign them numbers in ascending order after the abbreviation.
- For instance, if there are two employees named Le Thanh Tung and Le Thu Tung with the email abbreviation "Ittung," Le Thanh Tung is listed first, so no number is required. Le Thu Tung, who appears later, will be assigned a number beginning from 2, resulting in the email Ittung2@fit.hcmus.edu.vn.
- The input and output examples for this task are given in the input.txt and out1a.txt files, respectively

Exercise

- Write a Python program to merge multiple text files into a single output file, sorting the contents in alphabetical order and removing duplicates. Assume that each input file contains one word per line, and the output file should also contain one word per line.

Function	Description
<code>os.listdir()</code>	Returns a list of all files and folders in a directory
<code>os.scandir()</code>	Returns an iterator of all the objects in a directory including file attribute information
<code>pathlib.Path.iterdir()</code>	Returns an iterator of all the objects in a directory including file attribute information

THANK YOU
for YOUR ATTENTION