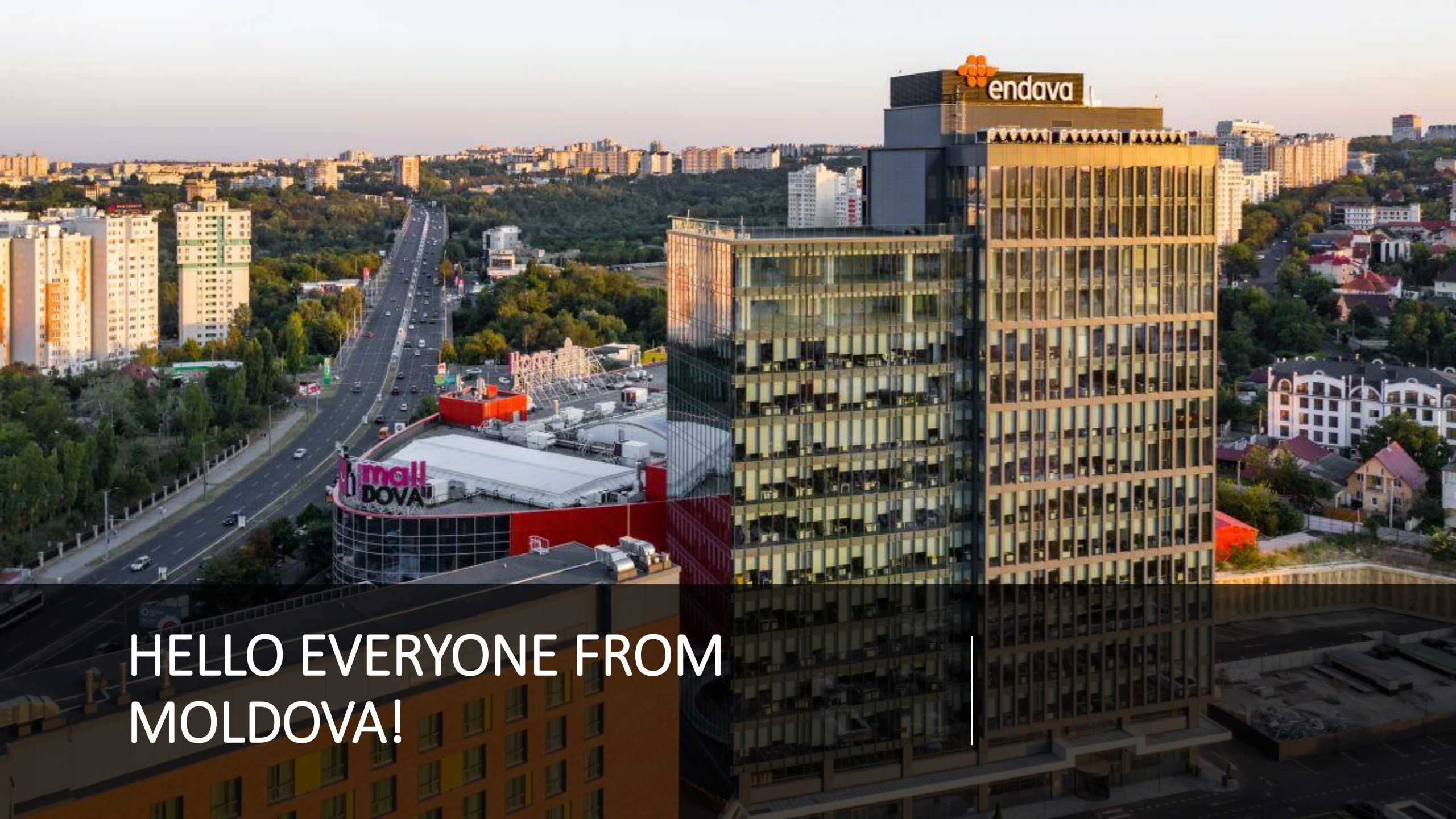
PYTHON PART 3

SCHOOL OF DEVOPS, FEBRUARY 2022

DMITRII DIACIKOVSKII

DEVOPSENGINEER

dmitrii.diacikovskii@endava.com



HELLO EVERYONE FROM MOLDOVA!

OUR PLANS

Unicode / ASCII / UTF8
Working with files
Access mode
With structure
Reading from files and writing to files
Exceptions handling
Useful python libraries
Import
Jinja2


A 10 MEG HARD DRIVE FROM THE 1970'S
imgflip.com

UNICODE

```
[ec2-user@ip-172-31-5-17 ~]$ echo 333 > tmp.txt
[ec2-user@ip-172-31-5-17 ~]$ cat tmp.txt
333
[ec2-user@ip-172-31-5-17 ~]$ xxd -b tmp.txt
00000000: 00110011 00110011 00110011 00001010                333.
[ec2-user@ip-172-31-5-17 ~]$
```

00000000: - It is not interesting for us
00110011 - This is the true value for the first digit 3
00110011 - This is the true value for the second digit 3
00110011 - This is the true value for the third digit 3
00001010 - This is the end of line control character. We don't even see it
333 - xxd shows what we expected to see

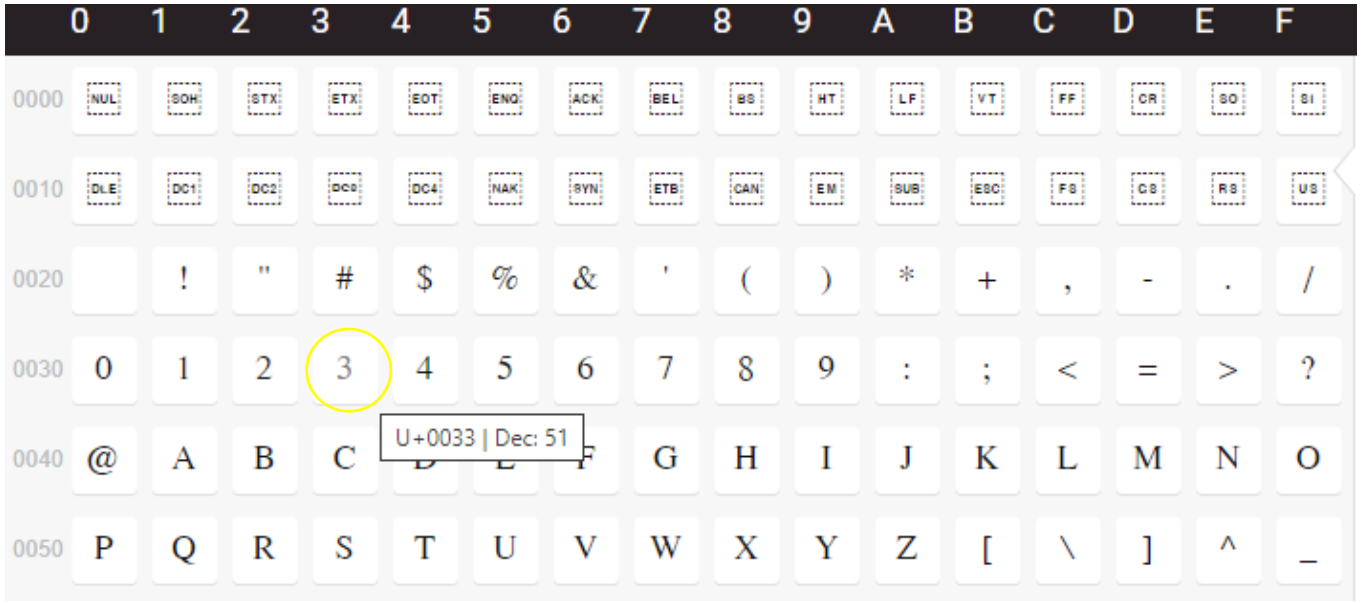
UNICODE

00110011 - This is byte.

We can convert it from binary to decimal and hexadecimal:

DEC: 00110011 -> 51
HEX: 00110011 -> 33

https://unicode-table.com/en/



endava

ASCII

2^7=128

The last bit is used to control errors

All numbers and Latin alphabet

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

endava

UTF-8

UTF-8 is the layer between the bits on the disk and the character in the Unicode table.
This layer does not affect the first 127 characters in the Unicode table. The entire Latin alphabet is placed in this range. UTF-8 encoding was invented for other alphabets.

```
[ec2-user@ip-172-31-5-17 ~]$ echo П > tmp.txt
[ec2-user@ip-172-31-5-17 ~]$ cat tmp.txt
П
[ec2-user@ip-172-31-5-17 ~]$ xxd -b tmp.txt
00000000: 11010000 10011111 00001010                    ...
```

00000000: - It is not interesting for us
11010000 - This is the first part of the letter П in UTF-8
10011111 - This is the second part of the letter П in UTF-8
00001010 - This is the end of line control character
… - П isn't visible, because xxd is very simple.
It doesn't know how to decode these all sorts of UTF-8. The Latin alphabet fits into the first 127 Unicode characters, and they don't need to be decoded at all ¯_(ツ)_/¯

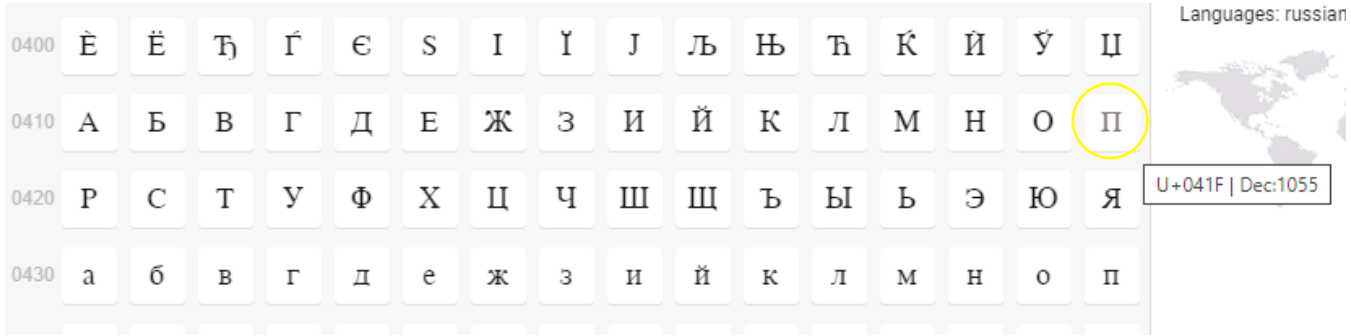
UTF-8

11010000 10011111

It is necessary to throw out 3 high bits at the first byte and 2 high bits at the second byte for decoding:
xxx --- xx
xxx10000 xx011111

Combine it:
0000010000011111

DEC: 10000011111 -> 1055
HEX: 10000011111 -> 41F



UNICODE / ASCHII / UTF8

I showed you the reverse process of obtaining a character from the repository.
The direct process of writing a symbol to the repository is absolutely identical. Any text editor known to you, after you clicked the button with the symbol on the keyboard, gets the number of this symbol in the Unicode table and encodes this number through the UTF-8 encoding, and only then these bytes get to disk.

https://en.wikipedia.org/wiki/UTF-8

Summary:
In storage always lie 0s and 1s.

UNICODE / ASCHII / UTF8

```python
# ord returns an integer for the specified Unicode character representing its code position
print(ord('3'))
# 51
print(ord('П'))
# 1055

# chr returns specified Unicode character for the integer representing its code position
print(chr(51))
# 3
print(chr(1055))
# П

print(chr(ord('a')))
# a
```

WORKING WITH FILES

Python3 works with files in much the same way as described above. Yes, in detail, the difference is huge, but the basic concept is identical. Reading bytes from a file, they are decoded from UTF-8 and converted to Unicode characters. Writing data to a file, Python3 encodes Unicode characters to UTF-8 and writes the received bytes to a file.

You must follow a certain sequence of operations when working with files:

- Opening a file using the open () method
- Reading a file using the read () method or writing to a file using the write () method
- Closing a file with the close () method

WORKING WITH FILES

The file object has the following attributes:

file.closed returns true if the file is closed and false otherwise
file.mode returns the mode of access to the file, while the file should be open
file.name returns file name

To start working with a file, it must be opened using the open () function:

open (file_path, mode)

Where:
file_path represents the path to the file; it can be either absolute or relative
mode sets file open mode

ACCESS MODE

Access mode	Description
r	Opens a file in read-only mode. The pointer is at the beginning of the file.
rb	Opens a file for reading in binary format. The pointer is at the beginning of the file.
r+	Opens a file for reading and writing. The pointer is at the beginning of the file.
rb+	Opens a file for reading and writing in binary format. The pointer is at the beginning of the file.
w	Opens a file for writing only. The pointer is at the beginning of the file. Creates a file named file_name if one does not exist.
wb	Opens a file for writing in binary format. The pointer is at the beginning of the file. Creates a file named file_name if one does not exist.
w+	Opens a file for reading and writing. The pointer is at the beginning of the file. Creates a file named file_name if one does not exist.
wb+	Opens a file for reading and writing in binary format. The pointer is at the beginning of the file. Creates a file named file_name if one does not exist.
a	Opens a file to add information to the file. A pointer is at the end of the file. Creates a file named file_name if one does not exist.
ab	Opens a file to add in binary format. A pointer is at the end of the file. Creates a file named file_name if one does not exist.
a+	Opens a file for adding and reading. A pointer is at the end of the file. Creates a file named file_name if one does not exist.
ab+	Opens a file for adding and reading in binary format. A pointer is at the end of the file. Creates a file named file_name if one does not exist.

WORKING WITH FILES

After you finish working with the file, you must close it with the close () method. This method will free all resources associated with the file.

For example, open “/etc/passwd”:

```
f = '/etc/passwd'

print("Before open file")
file_object = open(f, 'a')

print("Before write file")
file_object.write("text")

print("Before close file")
file_object.close()
```

WORKING WITH FILES

When you open a file or while working with it you may encounter various exceptions: there is no access to it etc. Your program will fail in this case, and its execution will not reach the call to the close method, and accordingly the file will not be closed.

The output of the program will be as follows:

```
Before open file
Before write file
Traceback (most recent call last):
  File "C:/Users/ddiacikovskii/PycharmProjects/First/3.py", line 33, in <module>
    file_object.write("text")
io.UnsupportedOperation: not writable
```

OS MODULE

In this case, we must handle the exceptions (try … except) or use os module.

```
import os

f = '/etc/passwd'
if os.access(f, os.F_OK):
    print("File exists")
else:
    print("File not exists")
if os.access(f, os.R_OK):
    print("File can be read")
else:
    print("File can not be read")
if os.access(f, os.W_OK):
    print("File can be written")
else:
    print("File can not be written")
if os.access(f, os.X_OK):
    print("File can be executed")
else:
    print("File can not be executed")
```

WITH STRUCTURE

With structure creates a context manager that automatically closes the file when it’s finished working in it.

```
with open('/etc/passwd', 'r+') as f:
    for line in f:
        print(line)

    f.write("text")
```

You can perform all standard input / output operations while you are within the code block.
Upon completion of the block, as well as when an error occurs, the file descriptor will be closed automatically.

WRITING TO FILES

To open a text file for recording, you must use w (overwrite) or a (add) mode.
Then, the write (str) method is used for writing, into which the recorded string is passed.
Only strings are recorded, therefore if you need to write data of any other types you must first convert it to a string.

```
with open('sometext.txt', 'w') as f:
    f.write("Hello")

with open('sometext.txt', 'a') as f:
    f.write("\nWorld\n")

with open('sometext.txt', 'a') as f:
    print('Hello Python!', file=f)
    f.write(str(42))
```

READING FROM FILES

readline () - reads one line from a file
read () - reads the entire contents of the file in one line
readlines () - reads all lines of a file into a list

Despite the fact that we do not explicitly use the readline () method to read each line, when iterating over a file, this method is automatically called to get each new line. To avoid unnecessary wrapping on another line, end = '' is passed to the print function.

```
with open('sometext.txt', 'r') as f:
    for line in f:
        print(line, end='')
```

READING FROM FILES

Explicitly call the readline () method to read lines:

```
with open('sometext.txt', 'r') as f:
    line_1 = f.readline()
    print(line_1, end='')
    line_2 = f.readline()
    print(line_2)
```

Or:

```
with open('sometext.txt', 'r') as f:
    line = f.readline()
    while line:
        print(line, end='')
        line = f.readline()
```

Or:

```
with open('sometext.txt', 'r') as f:
    content = f.read()
    print(content)
```

Or:

```
with open('sometext.txt', 'r') as f:
    contents = f.readlines()
    line_1 = contents[0]
    line_2 = contents[1]
```

READING FROM FILES

What if file encoding doesn't match UTF-8? For example WINDOWS-1251.
In this case, we can explicitly specify the encoding using the encoding parameter.

```
with open('sometext.txt', 'w', encoding='WINDOWS-1251') as f:
    f.write("АБВГ")

with open('sometext.txt', encoding='WINDOWS-1251') as f:
    text = f.read()
    print(text)
```

WORKING IN BINARY MODE

Python will not convert bytes greater than 127 to characters if you work with a file in binary mode. Python will read their 16-decimal representation. The read () method will return not just a string, but a bytes string.

```
with open('sometext.txt', 'w', encoding='UTF-8') as f:
    f.write("42Пfs")

with open('sometext.txt', 'rb+') as f:
    text = f.read(1)
    i = 1
    while text:
        print("{} byte :".format(i))
        print("\tbyte string like a character = {}".format(text))
        print("\thex = {}".format(text.hex()))
        print("\tbin = {}".format(bin(int(text.hex(), 16))[2:].zfill(8)))
        text = f.read(1)
        i = i + 1
```

```
# 1 byte :
#  byte string like a character = b'4'
#  hex = 34
#  bin = 00110100
# 2 byte :
#  byte string like a character = b'2'
#  hex = 32
#  bin = 00110010
# 3 byte :
#  byte string like a character = b'\xd0'
#  hex = d0
#  bin = 11010000
# 4 byte :
#  byte string like a character = b'\x9f'
#  hex = 9f
#  bin = 10011111
# 5 byte :
#  byte string like a character = b'f'
#  hex = 66
#  bin = 01100110
# 6 byte :
#  byte string like a character = b's'
#  hex = 73
#  bin = 01110011
```

RANDOM ACCESS

read () method reads data sequentially by default.
For random access to the file there is a seek function:

seek (offset [, whence])

Where:
offset - offset in bytes relative to the beginning of the file
whence - defaults to zero, indicates that the offset is taken relative to the beginning of the file

```
with open('sometext.txt', 'r+') as f:
    f.write('python')
    f.seek(3) # Go to 3-rd byte from beginning
    print(f.read(1))
# h
```

TRY-EXCEPT

Exceptions are when a technical error has occurred in the program.

For example:

- When you open a file and you do not have permission to do so. The program crashes.
- You are accessing the 10th element of the list in which there is no 10th element. The program crashes.
- etc.

Exceptions:
https://docs.python.org/3/library/exceptions.html

Exception hierarchy:
https://docs.python.org/3/library/exceptions.html#exception-hierarchy

TRY-EXCEPT

C
LBYL = Look before you leap

Python
EAFP = Easier to ask for forgiveness than permission

TRY-EXCEPT

Principle:

```
try:
    # Trying to do an operation
    pass
except:
    # Running these steps if try fails
    pass
else:
    # Running these steps if try succeeds
    pass
finally:
    # Always running these steps
    pass
```

TRY-EXCEPT

Example:

```
a = 1
b = 0
try:
    c = a/b
    print(f"Printing 'a' from try {a}")

except ZeroDivisionError:
    print("Running these steps because try fails")

else:
    print("Running these steps because try succeeds")

finally:
    print("Always running these steps")
# Output:
# Running these steps because try fails
# Always running these steps
```

Can’t seem to fix that error?

endava

TRY-EXCEPT

Creating personal exceptions:

```
class MyException(BaseException):
    pass

trigger = 0
try:
    if trigger:
        raise(ZeroDivisionError("Just for fun"))
    else:
        raise(MyException("I am a small harmless exception"))

except ZeroDivisionError as err:
    print("I caught you")
    print(err)

except MyException as err:
    print("I caught you, MyException")
    print(err)
# I caught you, MyException
# I am a small harmless exception
```

TRY-EXCEPT

Another example:

```
class BackendCrashed(BaseException):
    pass

class InvalidProduct(BaseException):
    pass

class CartError(BaseException):
    pass

try:
    cart.push()
except BackendCrashed:
    return Response(status=500)
except InvalidProduct:
    return Response(status=400)
except CartError as e:
    sentry.log(e)
    return Response(status=500)
```

TRY-EXCEPT

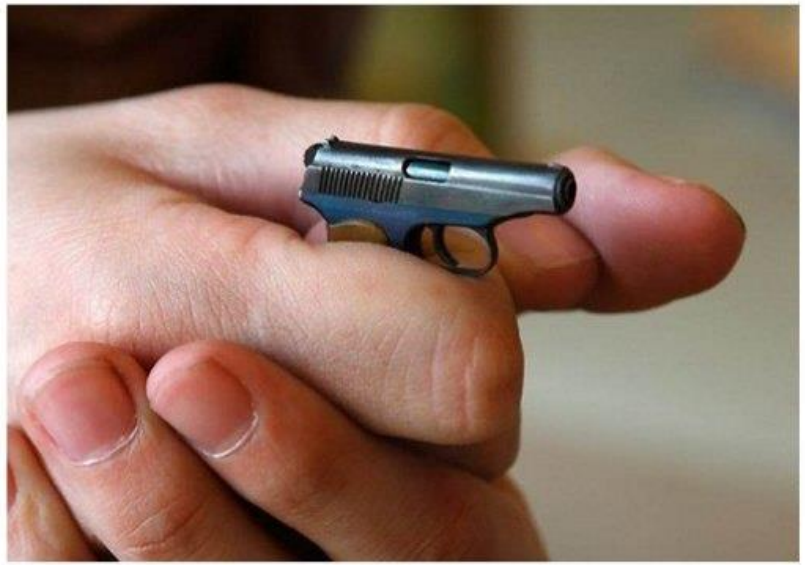
Bad example:

```
print("An important process works")
try:
    c = a+b
    print("Starting process...")
    make_magic()
    print("Generating result...")
    generate_result()
except Exception:
    print("No one saw anything, I keep working")
```

TRY-EXCEPT

```
def serverthread(clientsock, clientaddr, settings):
    while True:
        bin_data = clientsock.recv(settings['blksize'])
        if not bin_data:
            break
        udata = bin_data.decode()
        try:
            process_request(udata)
        except NotHTTPProtocol:
            logging.critical(f"Program received a non HTTP request:\n{udata})
        except ProcessRequestErr:
            logging.warning(f"Program use only GET/POST requests!\n{udata}")
        except CookieErr:
            logging.warning(f"Error with setting cookie!\n{udata}")
        except Exception:
            logging.critical(f"Unexpected execption\n{udata}")
        finally:
            clientsock.close()
```

Python with libraries



Python without libraries

OS LIBRARY

```
import os

# Create dir
os.mkdir('folder’)

# Delete dir
os.rmdir('folder’)

# Rename file rename(source, target)
os.rename('folder/old.txt', 'folder/new.txt’)

# Remove file
os.remove('folder/new.txt')
```

OS LIBRARY

```
import os

os.path.split(r'C:\temp\data.txt')
# ('C:\\temp', 'data.txt')
os.path.join(r'C:\temp', 'output.txt')
# C:\temp\output.txt

name = r'C:\temp\data.txt'
os.path.dirname(name)
# C:\temp
os.path.basename(name)
# data.txt
os.path.splitext(name)
# ('C:\\temp\\data', '.txt')
name.split(os.sep)
# ['C:', 'temp', 'data.txt']
```

OS LIBRARY

```
import os

name = r'C:\\temp\\public/files/index.html'
os.path.normpath(name)
# C:\temp\public\files\index.html
name = r'C:\temp\\sub\.\file.ext'
os.path.normpath(name)
# C:\temp\sub\file.ext

# Get full file name
os.path.abspath('.')
# C:\Users\ddiacikovskii\PycharmProjects\First
os.path.abspath('..')
# C:\Users\ddiacikovskii\PycharmProjects
os.path.abspath(r'./FacadeWebApi/web.config.bak')
# C:\Users\ddiacikovskii\PycharmProjects\First\FacadeWebApi\web.config.bak
```

STRING LIBRARY

```
import string

print(string.ascii_letters)
# abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
print(string.ascii_lowercase)
# abcdefghijklmnopqrstuvwxyz
print(string.ascii_uppercase)
# ABCDEFGHIJKLMNOPQRSTUVWXYZ
print(string.digits)
# 0123456789
print(string.printable)
# 0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
print(string.punctuation)
# !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
print(string.whitespace)
#  \t\n\r\x0b\x0c
```

IMPORT

What it's for?

- Code normalization
- Encapsulation
- Code structuring
- Pypi

IMPORT

- The module has no difference from the programs that we write on Python:

```
# File my_beauty_module.py
import math

def add(a, b):
    return a+b

def mul(a, b):
    return a*b

def log3(x):
    return math.log(x, 3)

if __name__ == "__main__":
    print("Hi, I'm a module! You should do import my_beauty_module, but not that")
```

A module is a file with the extension .py that contains a set of functions, variables, classes combined in meaning. The module should not contain the code initiating the program.

IMPORT

```python
# the contents of either the program or the module programandmodule.py

print(f"> I am the code inside the programandmodule.py file, and my name is: {__name__}")

if __name__ == "__main__":
    print("> I'm running as a program, I was launched from the command line ")
else:
    print("> I'm running as a module, I was imported somewhere")
```

IMPORT

```
# any other file
import programandmodule
```

> I am the code inside the programandmodule.py file, and my name is: programandmodule
> I'm running as a module, I was imported somewhere

```
# run programandmodule from command line
```

> I am the code inside the programandmodule.py file, and my name is: __main__
> I'm running as a program, I was launched from the command line

IMPORT

- Program:

```
# File module.py
print("I love Python")
```

- Module:

```
# File module.py
def make_magic():
    print("I love Python")
```

- Dangerous! We turn the module into a program:

```
# File module.py
def make_magic():
    print("I love Python")

make_magic()
```

IMPORT

Conclusion:

The program has a code and executes the logic (initiates the work) of this code immediately!

A module simply contains a set of code united in meaning, the LOGIC of which is not executed by itself.

IMPORT

- Importing the entire module and using it:

```
import my_beauty_module

if __name__ == "__main__":
    print(my_beauty_module.add(5, 8))
    print(my_beauty_module.log3(10))
```

- Or import module like an alias:

```
import my_beauty_module as mbm  # mbm – is alias

if __name__ == "__main__":
    print(mbm.add(5, 8))
    print(mbm.log3(10)
```

```
# File my_beauty_module.py
import math

def add(a, b):
    return a+b

def mul(a, b):
    return a*b

def log3(x):
    return math.log(x, 3)
```

IMPORT

- Also we can select what we want to import: from … import …

```
from my_beauty_module import add, log3

if __name__ == "__main__":
    print(add(5, 9))
    print(log3(10))
```

```
# File my_beauty_module.py
import math

def add(a, b):
    return a+b

def mul(a, b):
    return a*b

def log3(x):
    return math.log(x, 3)
```

IMPORT

- You can use import * for sketch projects (not production):

```
from my_beauty_module import *
```

- In other cases, import only what you need:

```
from abc import a, b, c
```

IMPORT

Why modules are needed?

In order to carry the code into separate files

Keep porridge of function descriptions, variables and executable code in one script is very bad

IRL you can always select some adequate piece of logic and put it into a separate function or module

Task: “stay warm in comfort”

- A bad programmer will simply dig a dugout and throw junk there.
- A good programmer builds a house, and put everything in its place.

Task: “change the bulb”

- A bad programmer will go to a turner, carve a ladder, look for nichrome, go to a factory, melt a bulb for a lamp, try to invent a lamp in a new way. He will bring it all home and try to replace the lamp.
- A good programmer will stand on a chair and screw in a new lamp that he bought in a store.

JINJA2

Jinja2 is a template engine for the Python programming language.

The work is divided into several stages: library import, template creation, template rendering.

Jinja is Flask's default template engine, and it is also used by Ansible, Trac, and Salt.
Ansible uses Jinja2 templating to enable dynamic expressions and access to variables and facts.

JINJA2

Python program for generating nginx.conf

```
from jinja2 import Template

env_dict = {
   '10.100.1.100':   'qa1',
   '10.100.1.101':   'qa2',
   '10.100.1.102':   'qa3',
   '10.100.2.100':   'sit1',
   '10.100.2.101':   'sit2',
   '10.100.2.102':   'sit3',
   '10.200.1.100':   'dev1',
   '10.200.1.101':   'dev2',
}

nginx_conf = open('nginx.j2').read()
template = Template(nginx_conf)
render = template.render(env_dict=env_dict)

with open('ngxin.conf', 'w+') as f:
   f.write(render)
```

nginx.j2 template file

```
{% for IP, ENV in env_dict.items() %}
server {
 listen 9090;
 server_name {{ ENV }}.mydomain.com;
 add_header X-Frame-Options SAMEORIGIN;
 add_header X-XSS-Protection "1; mode=block";
 add_header X-Content-Type-Options nosniff;

 location / {
  proxy_pass        http://{{ IP }}:8080;
  proxy_redirect    off;
  proxy_set_header   Host $host;
  proxy_set_header   X-Real-IP $remote_addr;
  proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header   X-Forwarded-Host $server_name;
 }
}

{% endfor %}
```

JINJA2

The same example, but using for loop in python program in case you need additional logic for each element in the dictionary (e.g., if/else statements)

```
from jinja2 import Template

env_dict = {
   '10.100.1.100':   'qa1',
   '10.100.1.101':   'qa2',
   '10.100.1.102':   'qa3',
   '10.100.2.100':   'sit1',
   '10.100.2.101':   'sit2',
   '10.100.2.102':   'sit3',
}

result = ''
nginx_conf = open('nginx.j2').read()
template = Template(nginx_conf)
for IP, ENV in env_dict.items():
   render = template.render(IP=IP, ENV=f"{ENV}.mydomain.com")
   result += render

with open('ngxin.conf', 'w+') as f:
   f.write(result)
```

```
server {
 listen 9090;
 server_name {{ ENV }};
 add_header X-Frame-Options SAMEORIGIN;
 add_header X-XSS-Protection "1; mode=block";
 add_header X-Content-Type-Options nosniff;

 location / {
  proxy_pass        http://{{ IP }}:8080;
  proxy_redirect     off;
  proxy_set_header   Host $host;
  proxy_set_header   X-Real-IP $remote_addr;
  proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header   X-Forwarded-Host $server_name;
 }
}
```