1. FILESYSTEM - OPEN() http://www.diveintopython3.net/files.html

open(**file**, **mode**='r', **buffering**=-1, **encoding**=None, **errors**=None, **newline**=None, **closefd**=True)

1A: ACCESSING FILE - OPENING AND CLOSING STREAM

(test.txt consists of text “text text text”)

getting file to variable:

```
a_file = open("C:/temp/test.txt")
print(a_file)
# <_io.TextIOWrapper name='C:/temp/test.txt' mode='r' encoding='cp1252'>
```

reading content:

```
a_file = open("C:/temp/test.txt")
content = a_file.read()
print(content) # text text text
```

closing stream:

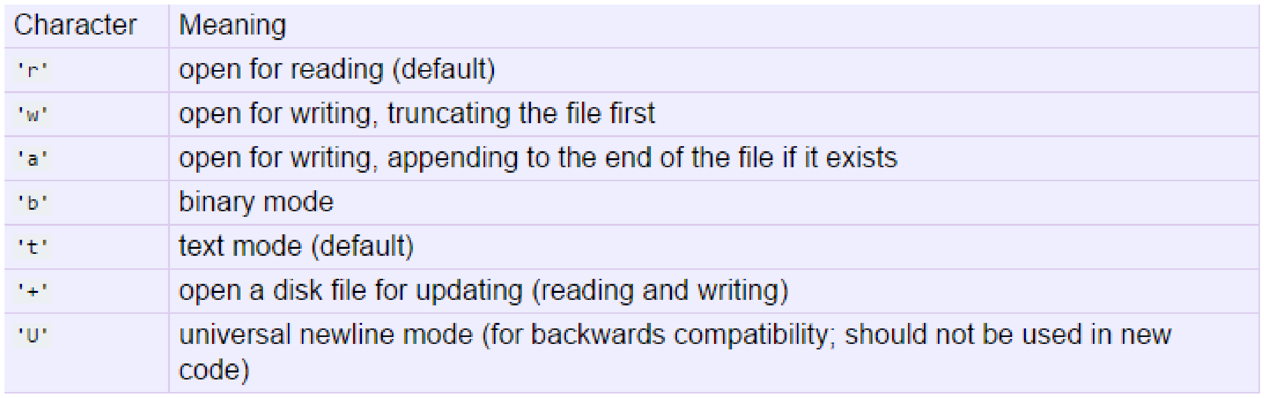
```
a_file = open("C:/temp/test.txt")
a_file.close()
content = a_file.read()
print(content) # ValueError: I/O operation on closed file.
```

1B: ENCODING

```
a_file = open('C:/temp/test.txt', encoding='cp1252')
content = a_file.read()
print(content) # ctštžcštýté=

a_file = open('C:/temp/test.txt', encoding='ascii')
content = a_file.read()
print(content)
# UnicodeDecodeError: 'ascii' codec can't decode byte 0x9a in position 2: ordinal not in
range(128)
```

1C: MODE - WRITING, APPENDING



Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)
`'U'`	universal newline mode (for backwards compatibility; should not be used in new code)

http://stackoverflow.com/a/23566951/1130598

writing to file:

```
a_file = open('C:/temp/test.txt', 'w') # mode has to be changed to 'w'
a_file.write('hi there, I am your new text') # text is not 'hi there, I am your new text'
a_file.seek(0) # will be discussed later
print(a_file.read()) # io.UnsupportedOperation: not writable
```

writing and reading files:

```
a_file = open('C:/temp/test.txt', 'r+') # mode changed to 'r+'
a_file.write('hi there, I am your new text')
a_file.seek(0) # will be discussed later
print(a_file.read()) # hi there, I am your new text
```

appending to file:

- append adds to existing file or create one if it doesnt exist
- write truncates existing file and writes then

```
# write mode
for i in range(0,5):
    a_file = open('C:/temp/test.txt', 'w')
    a_file.write(str(i))
# file is now 4

# append mode
for i in range(0,5):
    a_file = open('C:/temp/test.txt', 'w')
    a_file.write(str(i))
# file is now 1234

a_file = open('C:/temp/test2.txt', 'a+') # test2.txt doesnt exist
a_file.write('hello I am new here')
# now there is a test2.txt file with 'hello I am new here'
```

1D: WITH - BETTER WAY TO OPEN FILE

with closes the stream automatically - its much more safier

```
with open("C:/temp/test.txt", 'w+') as a_file:
    a_file.write('text text')

# and file is closed now
```

1E: CURSOR POSITION

position of stream

tell() - returns position

```
with open("C:/temp/test.txt", 'w+') as a_file:
print(a_file.tell()) # 0
a_file.write('text') # write changes position
print(a_file.tell()) # 4
```

seek(offset, from_what = 1) - changes position to offset

```
with open("C:/temp/test.txt", 'w+') as a_file:
    print(a_file.tell()) # 0
    a_file.write('text') # write changes position

    print (a_file.read())# ''

    a_file.seek(0)
    print (a_file.read())# ' text'

    print(a_file.tell()) # 4
    a_file.seek(2)
    print (a_file.read()) # 'tx'
```

from_what argument has 3 possibilities:

SEEK_SET or 0 – start of the stream (the default)
SEEK_CUR or 1 – current stream position; offset may be negative
SEEK_END or 2 – end of the stream; offset is usually negative

read(size) - read has an argument - max size of read text

```
with open("C:/temp/test.txt", 'w+') as a_file:
    a_file.write('text text text text text text') # write changes position
    a_file.seek(0)
    print(a_file.read(6)) # text t
```

1F: LINES

Creating new line in write() - **/n** does the magic:

```
with open("C:/temp/test.txt", 'w+') as a_file:
    for i in range(0,10):
        a_file.write('line' + str(i) + ': blablabla\n')

    a_file.seek(0)
    print(a_file.read())
```

Reading file line by line:

```
with open("C:/temp/LICENSE_PYTHON.txt", 'r') as a_file:
    for line in a_file:
        print(line)
```

readline(limit = -1) - reads to the end of line

```
with open("C:/temp/test.txt", 'r+') as a_file:
    print(a_file.readline()) # line0: blablabla
    a_file.seek(75)
    print(a_file.readline()) # e4: blablabla
```

readlines(hint = -1) - Read and return a list of lines from the stream. hint can be specified to control the number of lines read: no more lines will be read if the total size (in bytes/characters) of all lines so far exceeds hint.

```
with open("C:/temp/test.txt", 'r+') as a_file:
    print(a_file.readlines())
    # ['line0: blablabla\n', 'line1: blablabla\n', 'line2: blablabla\n', 'line3: blablabla\n', 'line4: blablabla\n', 'line5: blablabla\n', 'line6: blablabla\n', 'line7: blablabla\n', 'line8: blablabla\n', 'line9: blablabla\n']
```

1G: BINARY MODE

```
with open("C:/temp/test.jpg", 'rb+') as a_file:
    for line in a_file:
        print(line)
```

2. EXCEPTIONS http://www.diveintopython3.net/your-first-python-program.html#exceptions

- indication that something went wrong

2A: MOTIVATION

- wrong user input
- preventing expected errors
- handling error messages
- preventing code crashes

2B: TRY EXCEPT BLOCK

try - check if code is valid
except - if code is not valid

```
# without using try except block
print(4/0) # ZeroDivisionError: division by zero
print('next lines') # not printing anything, code is crashed

# using try except block
try:
    print(4/0)
except:
    print('math error') # this is printed

print('next lines') # this is printed also, code is still working


# preventing wrong input
number = input('select a number: ')

try:
    number + 5
    print(number + 5)
except:
    print('input is not a number')


# index() returns error if element is not in list
a_list = [5, 1, 6, 7, 3, 2]

for i in range(1,7):
    print('index of ',i, 'is', a_list.index(i))

# index of 1 is 1
# index of 2 is 5
# index of 3 is 4
# ValueError: 4 is not in list

# with try-except
a_list = [5, 1, 6, 7, 3, 2]

for i in range(1,7):
    try:
        print('index of ',i, 'is', a_list.index(i))
    except:
        print(i, 'is not in list')

# index of 1 is 1
# index of 2 is 5
# index of 3 is 4
# 4 is not in list
# index of 5 is 0
# index of 6 is 2
```

2C: ELSE

else- used with try and except, else will be evaluated if there is no error

```
try:
    number = int(input('enter a number:' ))
except:
    print('this is not a number')
else:
    print('this is a number')
```

2D: FINALLY

finally - used with try and except, code inside finally block is executed in any case

```
try:
    number = int(input('enter a number:' ))
except:
    print('this is not a number')
else:
    print('this is a number')
finally
    print('thank you for using our program')
```

2E: RAISE

raise - exits the code with error message
all error classes https://docs.python.org/3.3/library/exceptions.html#concrete-exceptions

```
# raising default error message
a_list = [5, 1, 6, 7, 3, 2]
```

```
for i in range(1,7):
    try:
        print('index of ',i, 'is', a_list.index(i))
    except:
        print(i, 'is not in list')
        raise # ValueError: 4 is not in list



# raising defined error message
a_list = [5, 1, 6, 7, 3, 2]

for i in range(1,7):
    try:
        print('index of ',i, 'is', a_list.index(i))
    except:
        print(i, 'is not in list')
        raise AssertionError('hi there, this is a custom error message')
```

raise could be used outside of try-except block

```
lucky_number = input('whats your lucky number?:' )
if lucky_number.find('7') == -1:
    raise Exception ('this is not a lucky number')
else:
    print('this is a lucky number')
```

3. BREAK (FOR CYCLES)

break - ends the iteration

```
rainfall_months = [10, 120, 150, 200, 210, 268, 272, 281, 295, 330, 354, 389]
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec']

rainfall_required = 270
month_required = str()

for mi in range(len(rainfall_months)):
    if rainfall_months[mi] > rainfall_required:
        month_required = months[mi]
        break

print(month_required) # Jul
```