**Section 14: Advanced Python Modules**

**11.03.**

**104. Python Collections Module**

Counter gives a sort of dictionary for counting the appearances for elements in a list/string

This Counter class has methods such most\_common to retrieve the most common x elements

The defaultdict doesn’t throw an error when an unexisting key is accessed

Instead based on a rule the key is added with a value

We define the insertion rule as defaultdict(lambda …)

There also are named tuples that have a name and whose elements can be accessed by strings

The elements can also be accessed with indexes

We define one as namedtuple(‘tuple\_name’, [‘attr1’, ‘attr2’, …])

**105. Python OS Module**

os commands work regardless of the operating system

In Jupyter notebook we can use pwd to print the working directory

With os we can do the same by using os.getcwd()

All files and folders inside a folder can be printed with os.listdir() (by default is working dir)

shutil is on top of os module and can be used to move, copy and delete files

shutil.move(‘source\_path/file’, ‘dest\_path’)

shutil.copy(‘source\_path/file’, ‘dest\_path’)

For removing files we can use multiple methods:

os.unlink(path) – deletes a file

os.rmdir(path) – deletes a folder if is empty

shutil.rmtree(path) – deletes all files and folders (even if they aren’t empty)

To be safety we can use send2trash.send2trash(‘path\_file’)

With the first 3 methods the files/folders cannot be retrieved (backed-up)

To walk in a tree of files/folders with a depth-first search we can use os.walk(path)

for folder, sub\_folders, files in os.walk(path)

**14.03.**

**106. Python Datetime Module**

In the datetime package there are time, date and datetime objects

For time objects we can specify hour, minute, second, etc

In this object we can then access minute etc

For date objects we can use .today()

Also in date objects various attributes can be accessed

Datetime objects include both date and time

In each type of object we can replace parts with .replace(element\_name=new\_value)

Naturally we can perform operations on them and then to access attributes in the result

**107. Python Math and Random Modules**

With help(math) we can check all the available methods

Methods as floor, ceil, log in every base, radians to degree or vice versa conversions, trig. func.

There are constants as pi, e or values as inf, nan

Round is very strange because it isn’t rounding the numbers as expected

For example round(4.5) is 4, but round(5.5) is 6 -> helps in removing the bias estimate

It is doing this to mediate the round up for all the numbers

Random package has functionalities as:

random.randint(a, b) gives a random integer in the interval [a, b]

For reproducible result we can use random.seed(value)

Get a random number from a list: random.choice(list)

Get a list of random numbers with replacement: random.choices(population=list, k=val\_no)

Get a list of random numbers without replacement: random.sample(population=list, k=val\_no)

Shuffle a list in place: random.shuffle(list)

Get floating point numbers from various distributions:

random.uniform(a=val, b=val)

random.gauss(mu=val, sigma=val)

**15.03.**

**108. Python Debugger**

We can debug notebooks or scripts with pdb package

By using pdb.set\_trace() we pause at that point the execution an we can query variables

In VS Code we can use the embedded debugger for both scripts and notebooks

In this way all the variables and libraries can be checked

Also breaking points can be placed

**112. Timing Your Python Code**

There are multiple ways to time a code

The first one is to use time.time() for start and end, and then to compute end – start

This isn’t very precise for very fast code

The second approach would be to use timeit module

timeit.timeit(statement, setup, number\_of\_repetitions)

This function returns the total time for running the statement for n time

Is important that we need to pass the statement that is executed, but as a docstring

The setup is run only once and includes the function definition that is executed

The third option is to use in a jupyter notebook at the top of the cell %%timeit

In this way the code in the cell is executed for 100000 times

%%timeit -n number\_of\_repetitions

**113. Zipping and Unzipping files with Python**

With zipfile package we can create archives in which we can place files

Then we can extract them one by one

Is important to recall that we initially need to create an empty archive

comp\_file = zipfile.ZipFile(archive\_name, ‘w’)

comp\_file.write(file\_name, compression\_type)

comp\_file.close()

If we want to extract the data:

zip\_obj = zipfile.ZipFile(archive\_name, ‘r’)

zip\_obj.extractall(folder\_name)

The shutil package can be used to zip and unzip entire folders

shutil.make\_archive(output\_name, ‘zip’, dir\_to\_zip)

shutil.unpack\_archive(dir\_to\_zip, unzipped\_dir\_name, ‘zip’)

**16.03.**

**109. Python Regular Expressions Part One**

In Python is a dedicated library called re

There are multiple methods:

Re.search(pattern, string) that will return an object if there’s a match or None

On that object we can call .start(), .end() or .group() to get the matched string

Re.findall(pattern, string) returns a list with all the matched strings

To find multiple matched we can use re.finditer(pattern, string)

**110. Python Regular Expressions Part Two**

**Character identifiers:**

\d – digit

\w – alphanumeric (includes the \_)

\s – white space

\D, \W and \S are the negation of \d, \w, \s

**Quantifiers:**

+ one or more times

\* zero or more times

? zero or none

{x} exactly x times

{x, y} x to y times

{x,} x or more times

For a regular expression pattern we need to use r before defining it

The pattern that we are searching can be compiled before

In this way we can then retrieve parts of the found strings that match with parts of the pattern

compiled\_pattern = re.compile(r’(\d{3})-(\d{4)’)

results = re.search(compiled\_pattern, text)

We access parts of results, with results.group(index) knowing that the index starts from 1

**111. Python Regular Expressions Part Three**

**Additional regex syntax:**

| as or -> cat|dog -> it finds cat or dog

. -> is a wild card and it matched any character

^ -> finds patterns that appear at the beginning of the text

$ -> finds patterns that appear at the end of the text

If we use ^ inside [], for example [^!.?] it means anything except!.?

With [] we can define groups, and after a group a repetition can be specified as + or \*