Program Arguments, Processes and Regular Expressions

Solve the following exercises and upload your solutions to Moodle until the specified due date. Make sure to use the *exact filenames* that are specified for each individual exercise. Unless explicitly stated otherwise, you can assume correct user input and correct arguments. You are *not allowed* to use any concepts and modules that have not yet been presented in the lecture.

Important Information!

We are running automated tests to aid in the correction and grading process, and deviations from the expected outputs lead to a significant organizational overhead, which we cannot handle in the majority of the cases due to the high number of submissions.

1. Please try to *exactly match the output* given in the examples (naturally, the input can be different). Feel free to copy the output text from the assignment sheet, and then change it according to the exercise task.

For example, if the exercise has an output of
`Number of cables: XYZ`
(where `XYZ` is some user input), do not write
`The number of cables: XYZ`
(additional `The` and lowercase `n`) or
`Number of cables:XYZ`
(missing space after the colon).

2. Furthermore, please don't have any lines of code that will be automatically executed when importing your module (except for what is asked by the exercise) as this will break our automated tests. Always execute your module before submitting it to verify this!

For example, if you have some code to test your program and reproduce the example outputs, either comment/remove these lines or move them to a function.

Exercise 1 – Submission: `a9_ex1.py` 25 Points

Write a program where a user can interactively search for patterns in a file with the module `re`. The program should work as follows (see the example program execution below for more details on how the command line interface (CLI) must look like):

1. The user must enter the input file to be parsed. Use `"Enter file name: "` for the CLI. Check, if the file exists. If not raise `ValueError(f"'<input_file>' is not a file")` where `<input_file>` is the file name previously entered by the user.
2. The user must enter the character encoding to use for accessing the input file or press the *Enter* key (results in an empty string) in which case the encoding should be set to `utf-8`. Use `"Enter character encoding or press ENTER for default (utf-8): "` for the CLI.
3. The user must enter a regular expression pattern or press the *Enter* key (results in an empty string) to exit. For each pattern, the `input_file` is parsed, and every full (e.g., in the case of groups) pattern match is extracted to a list and printed to the console as `f"<pattern>: <list_of_matches>"` where `<pattern>` is the regular expression pattern provided by the user and `<list_of_matches>` a list of the extracted pattern matches. Use `"Enter pattern or press ENTER to exit: "` for the CLI.
4. Repeat step 3. until the user presses the *Enter* key to exit.

Example program execution (assuming the current working directory contains `a9_ex1_data.txt`)[1]:

```
Enter file name: a9_ex1_data.txt
Enter character encoding or press ENTER for default (utf-8):
Enter pattern or press ENTER to exit:  (d..d)
(d..d): ['deed', 'de d']
Enter pattern or press ENTER to exit: \d{3}
\d{3}: ['780', '222', '341', '445', '100']
Enter pattern or press ENTER to exit: \d{ab12.c}
\d{ab12.c}: []
Enter pattern or press ENTER to exit: d[^ ]+
d[^ ]+: ['d23g780nb', 'deed', 'de', 'ddd32']
Enter pattern or press ENTER to exit:
```

[1] In green inputs from the console.

Exercise 2 – Submission: `a9_ex2.py` 25 Points

Write a function `extract_emails(text: str) -> list[str]` that extracts valid email addresses from a `text` and returns the addresses as a list of strings. Valid email addresses are defined in the following way `<username>@<subdomain>.<top-level domain>` with[2]:

- `username`: 1 or more characters. Allowed characters include both upper- and lower-case letters (from `a` to `z`), numbers and the following special characters: `"."`, `"-"`, `"_"`, `"%"` and `"+"`.
- `subdomain`: 1 or more characters. Allowed characters include both upper- and lower-case letters (from `a` to `z`) and the following special characters: `"."`, and `"-"`.
- `top-level domain`: 2 or more characters. Allowed characters only include upper- and lower-case letters (from `a` to `z`).

Also, check that the pattern starts with either a whitespace or newline character.

Use the `re` module and regular expressions to solve this task. You are *not allowed* to use any manual string handling such as splitting, replacing, substring checking, converting to lowercase, etc.

Example program execution:

```
t = """
Here are some email addresses:
john.doe@example.com, alice_smith123@gmail.com, ABC+@a-b-c.aBc,
contact@company.org, and info@sub.domain.co.uk.
Some invalid email addresses are:
john@, @example.com, user@domain, us/er@email.com,
invalid@domain.f and invalid.email@invalid@domain.com.
"""
print(extract_emails(t))
```

Example output:

```
['john.doe@example.com', 'alice_smith123@gmail.com', 'ABC+@a-b-c.aBc',
'contact@company.org', 'info@sub.domain.co.uk']
```

Hints:

- You can use https://regex101.com/ to quickly test your regular expressions.
- It is possible to create a single regular expression that can extract emails as specified above.

[2] This is our definition of a valid email. In real-world applications several other special characters might be allowed, and other restrictions might be applied.

Exercise 3 – Submission: `a9_ex3.py` 25 Points

Write a program that can approximate the Euler–Mascheroni constant which is defined as

$$\gamma = \lim_{n\to\infty}\left(-\log n + \sum_{k=1}^{n}\frac{1}{k}\right) \approx 0.577\,215\,665.$$

Your program should work as follows:

- Create a function `sum_of_fractions(start: int, end: int) -> float` that computes the following sum: $\sum_{k=start}^{end} 1/k$.
- Create the main entry point of your script with `if __name__ == "__main__"`. There use the `argparse` module to retrieve the following (keyword) command line arguments:
  - Number of processes: Long/short form: `processes` / `p`; type: `int`; optional; default: 1
  - Number of terms in the sum: Long = short form: `n`; type: `int`; optional; default: 1000

  Set up a `multiprocessing.Pool` with `p` processes, where each process invokes the above function `sum_of_fractions`. As `sum_of_fractions` requires multiple outputs as input, create a list of arguments with `p` tuples containing appropriate `start` and `end` for each process. Each process will do a part of the total sum for ∼`n/p` terms.
- Print the following output at the end of the program with a precision of 9 digits for the `gamma`: `"Euler-Mascheroni constant approximation (<n> terms): <gamma>"`

Hints:

- In the equation "log" corresponds to the natural logarithm (base e). This is the convention in many programming languages (Python included). You can use `math.log` to compute it.

Example command line arguments:

```
-n 10
-n 1_000_000_000 -p 4
```

Example output:

```
Euler-Mascheroni constant approximation (10 terms): 0.626383161
Euler-Mascheroni constant approximation (1000000000 terms): 0.577215665
```

Exercise 4 – Submission: `a9_ex4.py` 25 Points

Write a program where a user can interactively execute arbitrary programs with arguments. Read the following command line arguments with `argparse`

- Long form: `program`; short form: `p`; type: `str`; required

  Name of the program to be executed.

- Long form: `args`; short form: `a`, type: `list` with ≥ 1 `str` items; optional; default: `[]`

  Arguments to be passed to the program.

- Long form: `timeout`; short form: `t`, type: `float`; optional; default: `60`

  Timeout in seconds for the execution of the program.

The specified program with the arguments (if any) must be executed using the `subprocess` module. Specify the subprocess to use the `"utf-8"` encoding. To visualize this step in the console, print the strings:

- `"Running program '<program>' without any arguments with a <timeout>s timeout"` in case there are no arguments for the specified program `<program>`.
- `"Running program '<program>' with arguments <args> with a <timeout>s timeout"` in case there are some arguments `<args>` for the specified program `<program>`. `<args>` is simply the string representation of the list of arguments collected earlier (see example output below).

If the program does not exist or runs beyond the timeout catch the corresponding exceptions:

- If the program does not exist, catch the raised `FileNotFoundError` exception and print the following warning message: `"The specified program '<program>' could not be found"`, where `<program>` is the program name.
- If the timeout is reached, catch the raised `subprocess.TimeoutExpired` exception and print the following warning message: `"The program execution timed out"`.

In all other cases, the specified program is executed normally. Once the specified program has finished, print its return/status code (`"The '<program>' finished with exit code <code>"`, where `<code>` is this return/status code).

Also, print the output, but only if it is not empty.

- For non-empty error output, print `"The '<program>' produced the following error output:"` followed by the error output (on a new line).
- For non-empty standard output, print `"The '<program>' produced the following standard output:"` followed by the standard output (on a new line).

Example command line arguments:

```
--args hello
```

Example output (automatically generated by `argparse` due a required argument missing):

```
usage: a9_ex3.py [-h] -p PROGRAM [-a [ARGS ...]] [-t TIMEOUT]
a9_ex3.py: error: the following arguments are required: -p/--program
```

Example command line arguments:

```
--program pythn
```

Example output:

```
Running program 'pythn' without any arguments with a 60s timeout
The specified program 'pythn' could not be found
```

Example command line arguments:

```
-p python --timeout 10
```

Example output:

```
Running program 'python' without any arguments with a 10s timeout
The program execution timed out
```

Example command line arguments:

```
-p python --args a9_ex3.py
```

Example output:

```
Running program 'python' with arguments ['a9_ex3.py'] with a 60s timeout
The 'python' finished with exit code 0
The 'python' produced the following standard output:
Euler-Mascheroni constant approximation (1000 terms): 0.577715582
```

Example command line arguments[3]:

```
-p python -a a9_ex3.py "-x 9"
```

Example output:

```
Running program 'python' with arguments ['a9_ex3.py', '-x 9'] with a 60s timeout
The 'python' finished with exit code 2
The 'python' produced the following error output:
usage: a9_ex3.py [-h] [-p PROCESSES] [-n N]
a9_ex3.py: error: unrecognized arguments: -x 9
```

[3]The quotation marks are need because without them the call will try to interpret `-x` as part of `a9_ex4.py`'s CLI.