Challenges 3

December 15, 2025

1 Task 1 - Enumerate, Tuple unpacking

Implement a function `student_seats` which takes a list of matriculation numbers, seats per row and space between students. It should then output where each student is seated Expected output:

```
students_seat([1337, 2337, 3337, 4337], 10, 4)
```

```
Student 1337 sits in 0, seat 0
Student 2337 sits in 0, seat 4
Student 3337 sits in 0, seat 8
Student 4337 sits in 1, seat 2
```

- Hint: use the `enumerate` function which provides you a tuple of `(index, value)`
- Hint 2: try to use tuple-unpacking in the for-loop head
- Hint 3:

```
row = (i * space_between) // seats_per_row
seat = (i * space_between) % seats_per_row
```

[2]:
```python
def students_seat(mat: list, seats_per_row: int, space_between: int) -> None:
    for i, mat_nr in enumerate(mat):
        row = (i * space_between) // seats_per_row
        seat = (i * space_between) % seats_per_row
        print(f"Student {mat_nr} sits in {row}, seat {seat}")

students_seat([1337, 2337, 3337, 4337], 10, 4)
```

```
Student 1337 sits in 0, seat 0
Student 2337 sits in 0, seat 4
Student 3337 sits in 0, seat 8
Student 4337 sits in 1, seat 2
```

2 Task 2: List Comprehensions

- Implement a function `get_even_and_divisible(n, div)` which returns all even numbers between 2 and n (inclusive) and are also divisible by `div`
- You must use a comprehension

[ ]:
```python
def get_even_and_divisible(n: int, div: int) -> list:
    pass
```

```
assert get_even_and_divisible(100, 7) == [14, 28, 42, 56, 70, 84, 98]
assert get_even_and_divisible(100, 5) == [10, 20, 30, 40, 50, 60, 70, 80, 90,␣
 ↪100]
```

3 Task 2b: Translate into comprehension

[ ]:
```
def frequency_analysis(input_text: str) -> dict:
    result = {}
    for c in set(input_text):
        result[c] = input_text.count(c)
    return result

assert frequency_analysis("AABBCCC") == {'A': 2, 'C': 3, 'B': 2}
```

[4]:
```
def frequency_analysis(input_text: str) -> dict:
    return {c: input_text.count(c) for c in set(input_text)}
assert frequency_analysis("AABBCCC") == {'A': 2, 'C': 3, 'B': 2}
```

4 Task 3: Read and Sort

- Implement a function read_and_sort, which reads the file sortme.txt and returns list of lines (without new lines at the end!!), but sorted

[9]:
```
def read_and_sort() -> list:
    with open("sortme.txt") as fh:
        lines = [line.strip() for line in sorted(fh.readlines())]
        return lines
    pass

assert read_and_sort()[-1] == 'fdd8c3b26e1172a8d7e05c8817ef3ef7'
assert read_and_sort()[0] == '01ca0557576d7b0f9131a6560508b822'
```

[7]:
```
def read_and_sort() -> list:
    with open("sortme.txt") as fh:
        lines = [line.strip() for line in fh.readlines()]
        return sorted(lines)


assert read_and_sort()[-1] == 'fdd8c3b26e1172a8d7e05c8817ef3ef7'
assert read_and_sort()[0] == '01ca0557576d7b0f9131a6560508b822'
```

[8]:
```
sorted?
```

```
Signature: sorted(iterable, /, *, key=None, reverse=False)
Docstring:
```

```
Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the
reverse flag can be set to request the result in descending order.
Type:      builtin_function_or_method
```

5 Task 4: Exceptions

- Implement a function factorial, which raises a RuntimeError with a message "Not defined for negative numbers" if called with negative numbers

[10]:
```python
def factorial(n: int) -> int:
    if n < 0:
        raise RuntimeError("Not defined for negative numbers")
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result


assert factorial(5) == 120
try:
    factorial(-1)
    assert False #  should never be reached
except RuntimeError as e:
    assert str(e) == 'Not defined for negative numbers'
except Exception as e:
    assert False
```

6 Task 5a - Integrity anyone?

- Implement a function `integrity(message: str, digest: str) -> bool` which for given message checks if the SHA1 sum of the message is the one that was passed to the function as `digest`
- Hint: recall how to transform a str to bytes...

[11]:
```python
import hashlib

def integrity(message: str, digest: str) -> bool:
    return hashlib.sha1(message.encode()).hexdigest() == digest
    pass

assert integrity("Hello World", "0a4d55a8d778e5022fab701977c5d840bbc486d0") is␣
 ↪True
assert integrity("hello World", "0a4d55a8d778e5022fab701977c5d840bbc486d0") is␣
 ↪False
```

7 Task 5b - JSON find the key

- Implement a function `parse_and_return` that parses it first parameter as JSON and returns the value if a given key exists (and None otherwise)

[12]:
```
from json import loads

def parse_and_return(json_str: str, key: str) -> str | None:
    return loads(json_str).get(key)


assert parse_and_return('{"ben": "stock@cispa.de", "sebastian":␣
 ↪"sebi@somethingvienna.at"}', 'ben') == 'stock@cispa.de'
assert parse_and_return('{"ben": "stock@cispa.de", "sebastian":␣
 ↪"sebi@somethingvienna.at"}', 'marius') is None
```

[ ]:
```

```