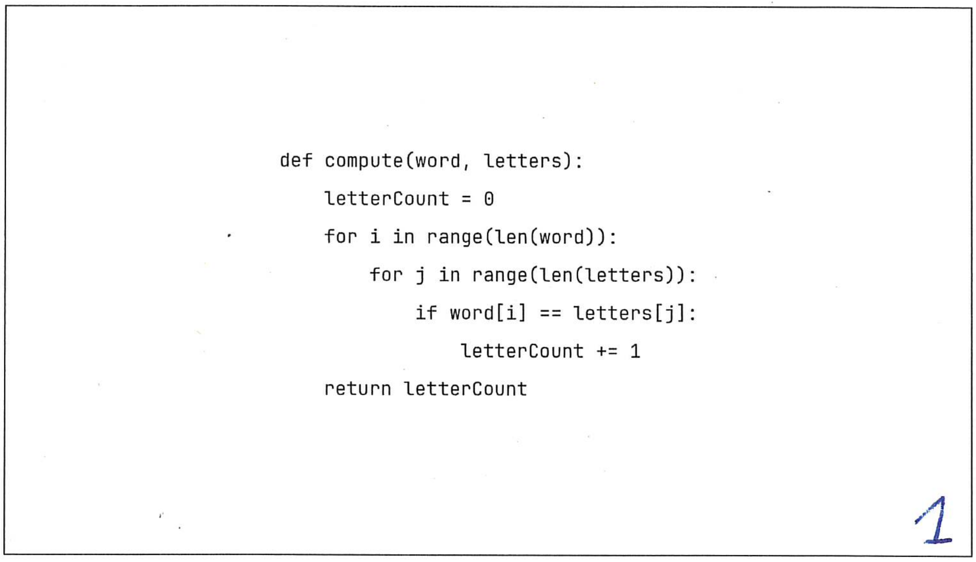
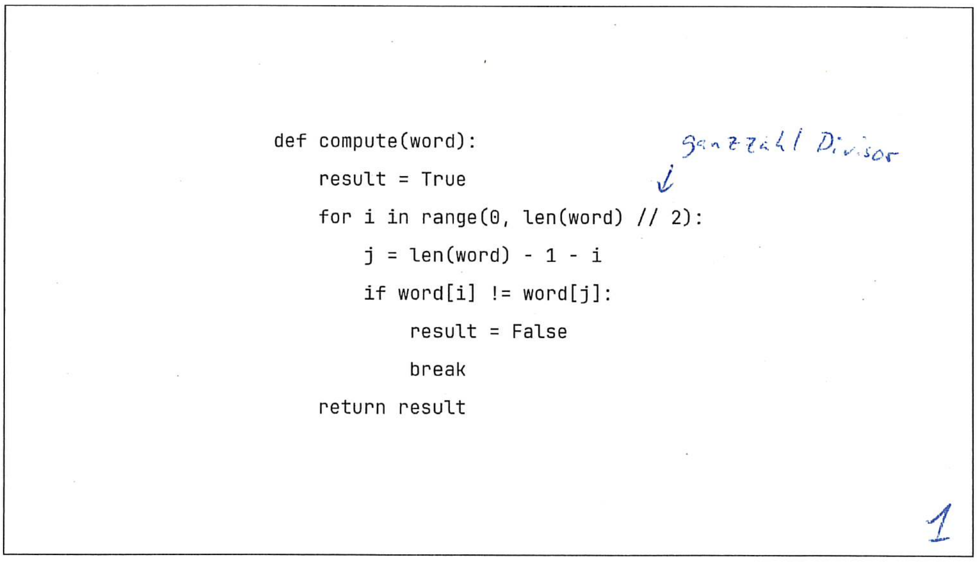
25574



```
def compute(word, letters):
    letterCount = 0
    for i in range(len(word)):
        for j in range(len(letters)):
            if word[i] == letters[j]:
                letterCount += 1
    return letterCount
```

1

Figure 1: countLettersMN

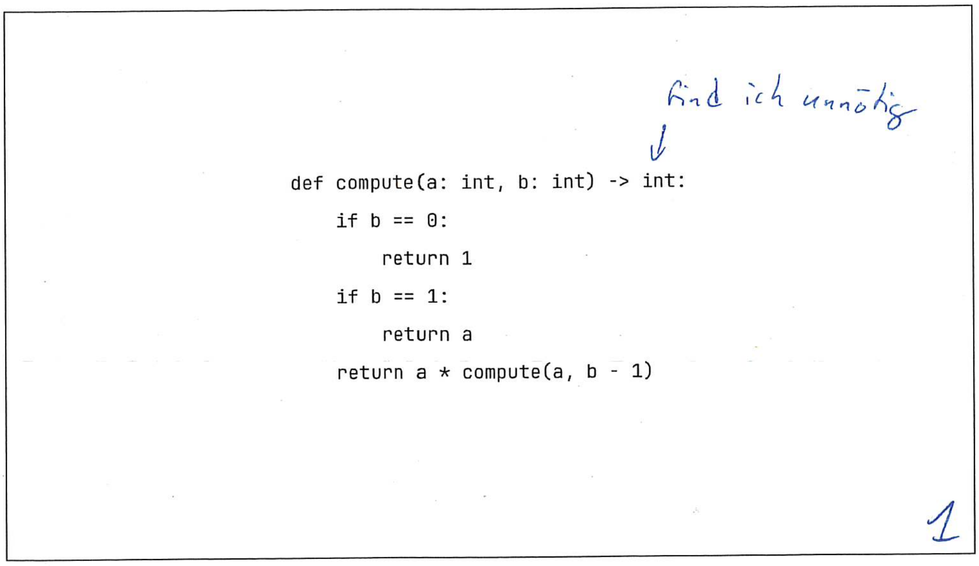


ganzzahl Divisor ↓

```
def compute(word):
    result = True
    for i in range(0, len(word) // 2):
        j = len(word) - 1 - i
        if word[i] != word[j]:
            result = False
            break
    return result
```

1

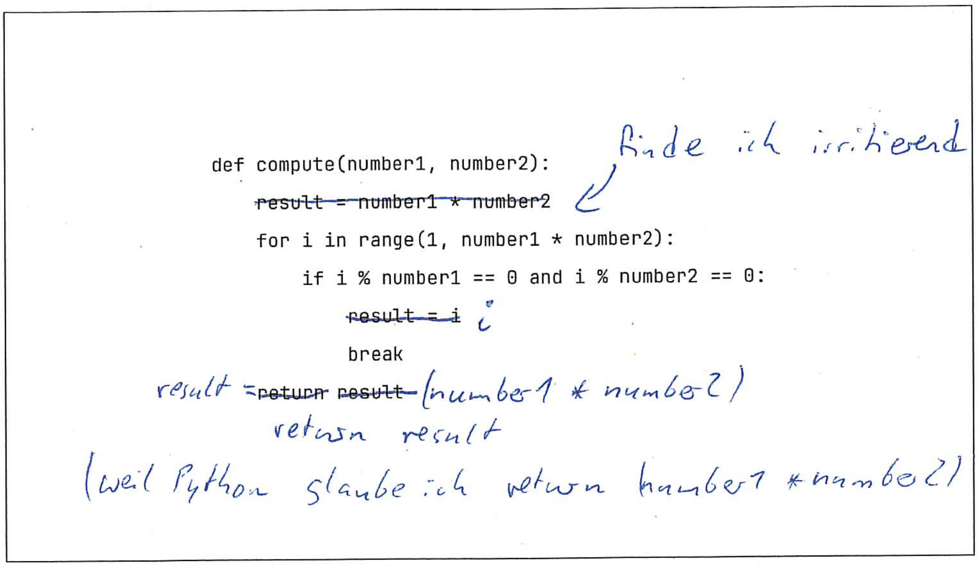
Figure 2: palindromeMN



```
                                              find ich unnötig
                                              ↓
def compute(a: int, b: int) -> int:
    if b == 0:
        return 1
    if b == 1:
        return a
    return a * compute(a, b - 1)
```

1

Figure 3: powerMT

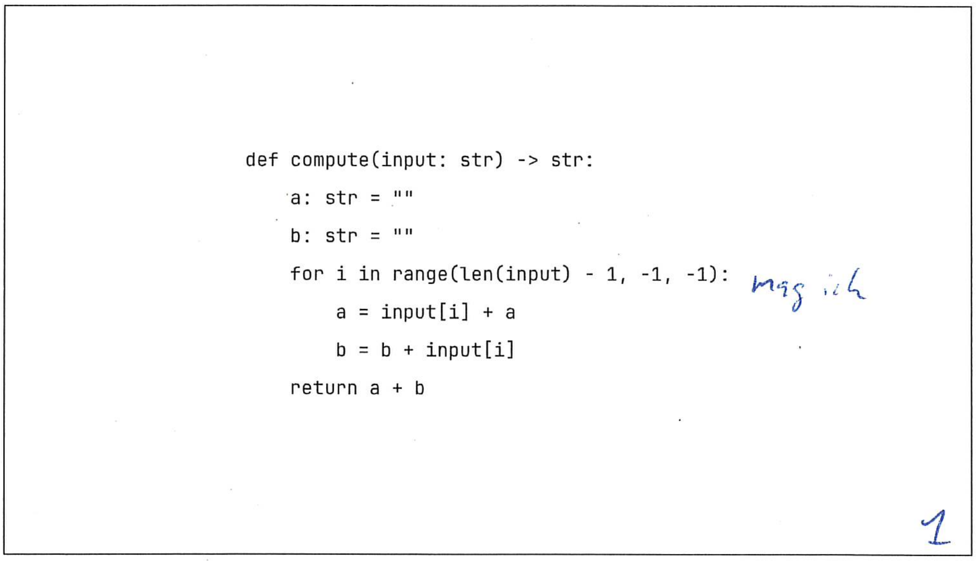


```
                                         finde ich irritierend
def compute(number1, number2):
    ~~result = number1 * number2~~  ↙
    for i in range(1, number1 * number2):
        if i % number1 == 0 and i % number2 == 0:
            ~~result = i~~ i
            break
result = ~~return result~~ (number1 * number2)
    return result
(weil Python glaube ich return number1 * number2) nicht mag
```

2

Figure 4: leastCommonMultipleMN

25574

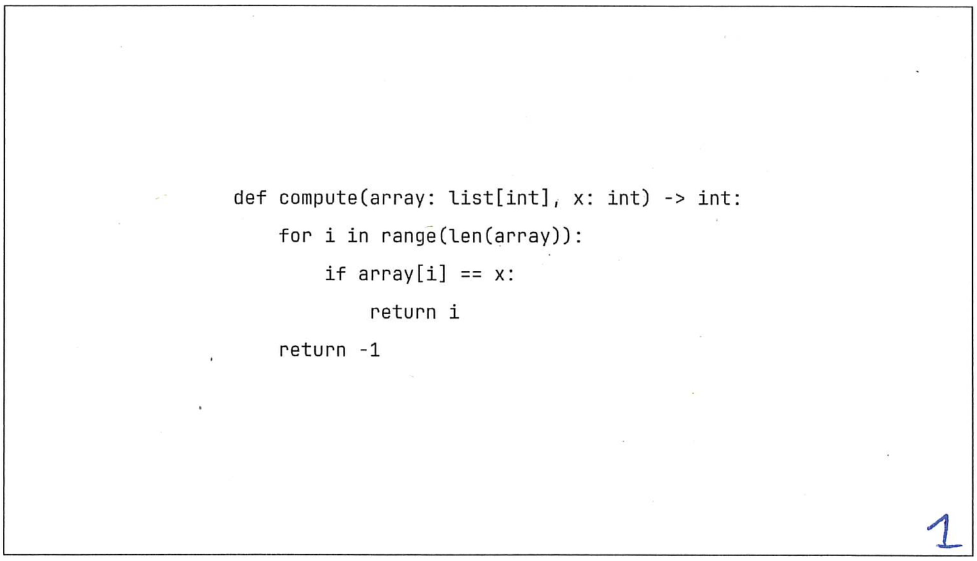


```
def compute(input: str) -> str:
    a: str = ""
    b: str = ""
    for i in range(len(input) - 1, -1, -1):
        a = input[i] + a
        b = b + input[i]
    return a + b
```

mag ich

1

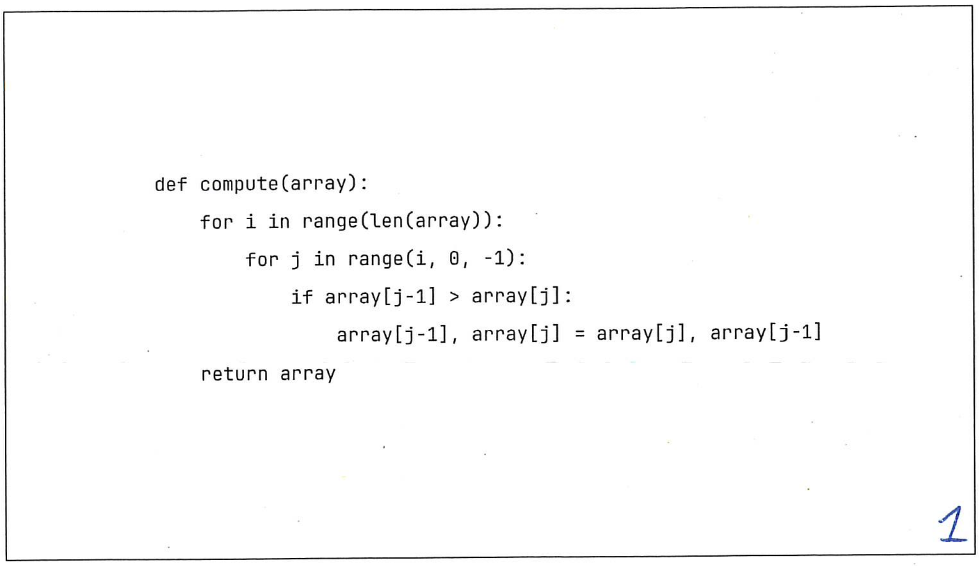
Figure 5: forwardBackwardMT



```
def compute(array: list[int], x: int) -> int:
    for i in range(len(array)):
        if array[i] == x:
            return i
    return -1
```

1

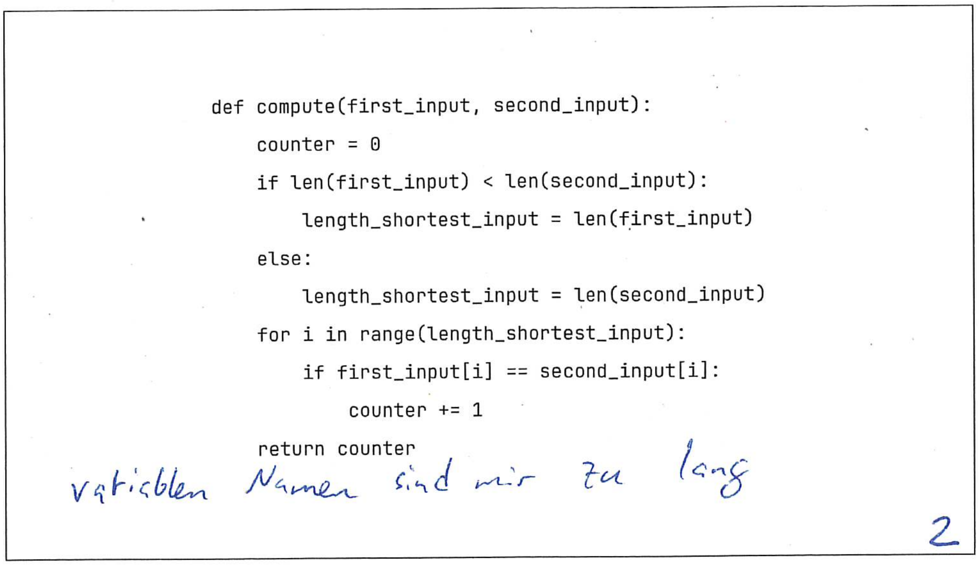
Figure 6: linearSearchMT



```
def compute(array):
    for i in range(len(array)):
        for j in range(i, 0, -1):
            if array[j-1] > array[j]:
                array[j-1], array[j] = array[j], array[j-1]
    return array
```

1

Figure 7: bubbleSortMN

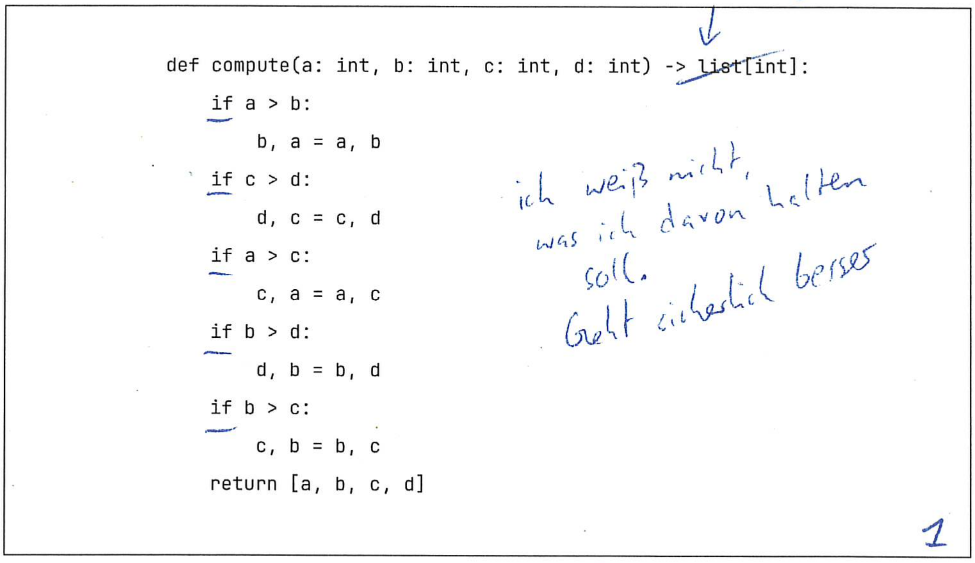


```
def compute(first_input, second_input):
    counter = 0
    if len(first_input) < len(second_input):
        length_shortest_input = len(first_input)
    else:
        length_shortest_input = len(second_input)
    for i in range(length_shortest_input):
        if first_input[i] == second_input[i]:
            counter += 1
    return counter
```

variablen Namen sind mir zu lang

2

Figure 8: commonCharsMN



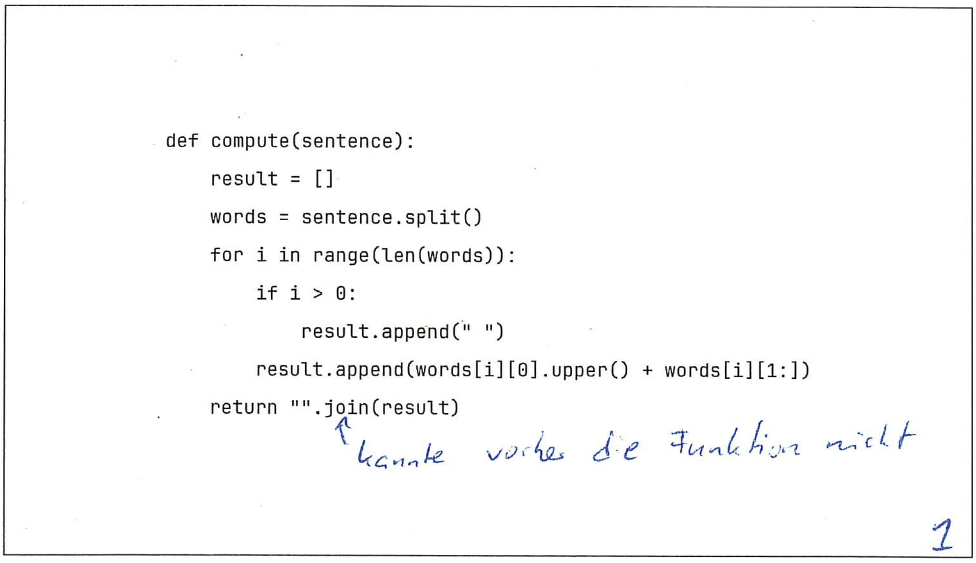
unnötig

```
def compute(a: int, b: int, c: int, d: int) -> ~~list~~[int]:
    if a > b:
        b, a = a, b
    if c > d:
        d, c = c, d
    if a > c:
        c, a = a, c
    if b > d:
        d, b = b, d
    if b > c:
        c, b = b, c
    return [a, b, c, d]
```

ich weiß nicht, was ich davon halten soll. Geht sicherlich besser

1

Figure 9: unrolledSortMT

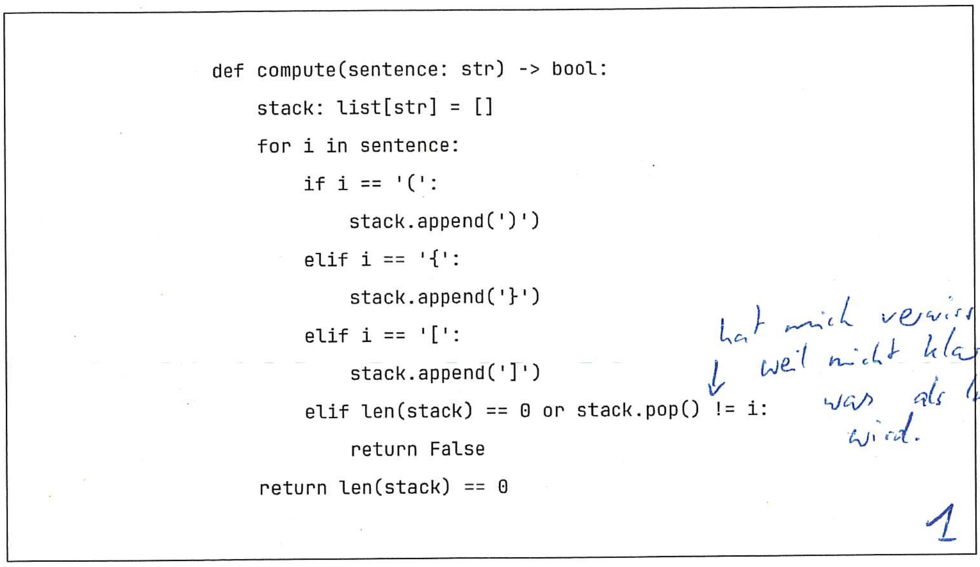


```
def compute(sentence):
    result = []
    words = sentence.split()
    for i in range(len(words)):
        if i > 0:
            result.append(" ")
        result.append(words[i][0].upper() + words[i][1:])
    return "".join(result)
```

kannte vorher die Funktion nicht

1

Figure 10: capitalizeFirstLetterMN

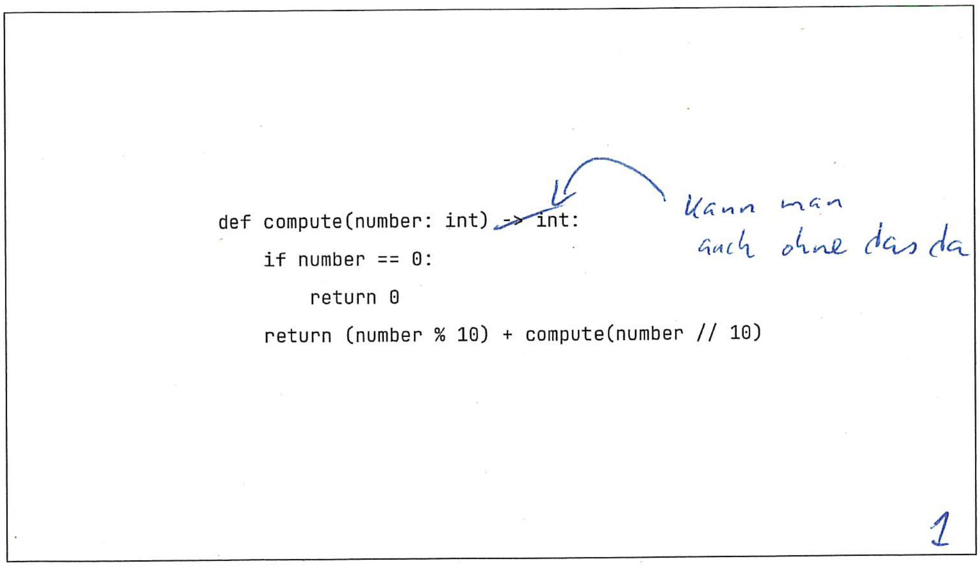


```
def compute(sentence: str) -> bool:
    stack: list[str] = []
    for i in sentence:
        if i == '(':
            stack.append(')')
        elif i == '{':
            stack.append('}')
        elif i == '[':
            stack.append(']')
        elif len(stack) == 0 or stack.pop() != i:
            return False
    return len(stack) == 0
```

hat mich verwirrt, weil nicht klar war, was als Input kommen wird.

1

Figure 11: validParenthesesMT



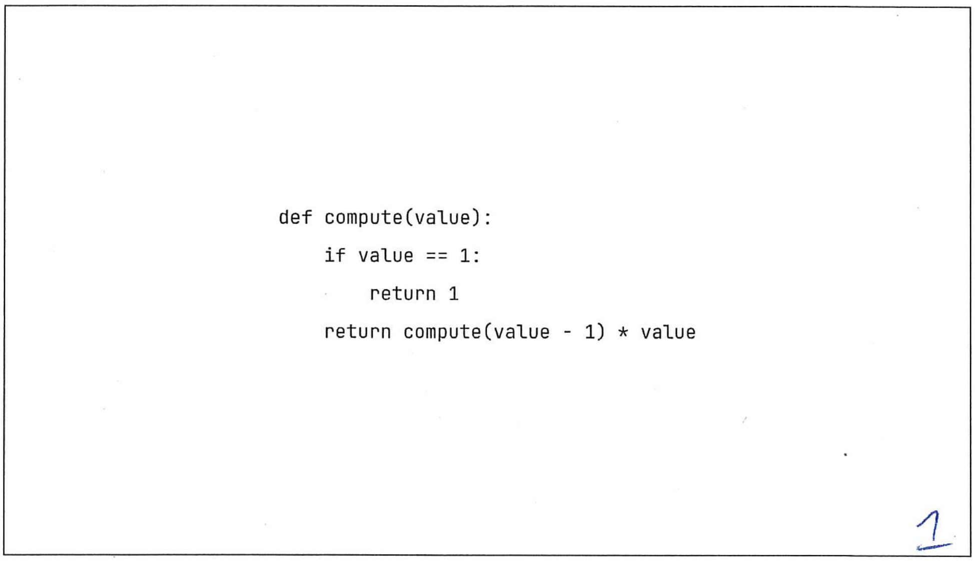
```
def compute(number: int) -> int:
    if number == 0:
        return 0
    return (number % 10) + compute(number // 10)
```

Kann man auch ohne das da

1

Figure 12: crossSumMT

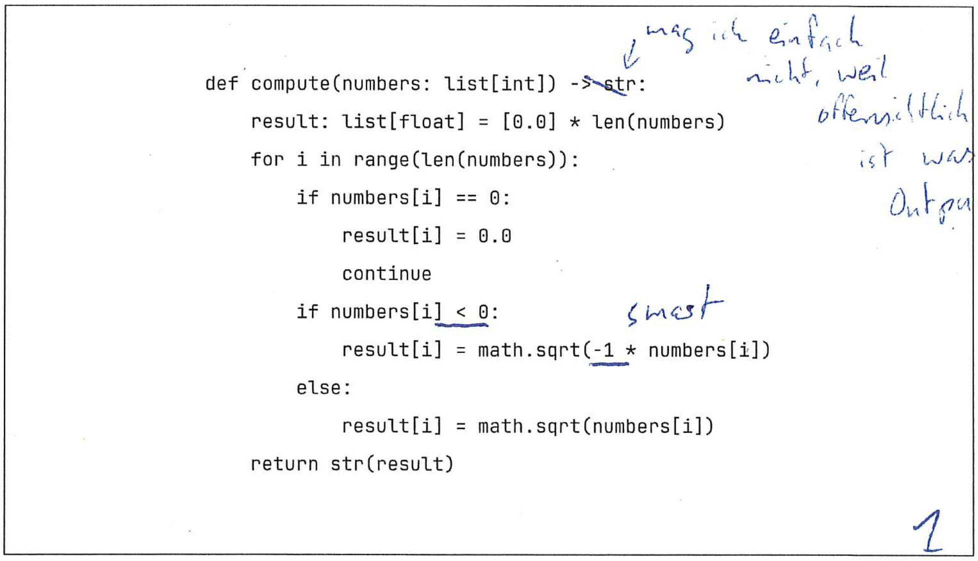
25574



```
def compute(value):
    if value == 1:
        return 1
    return compute(value - 1) * value
```

1

Figure 13: factorialMN



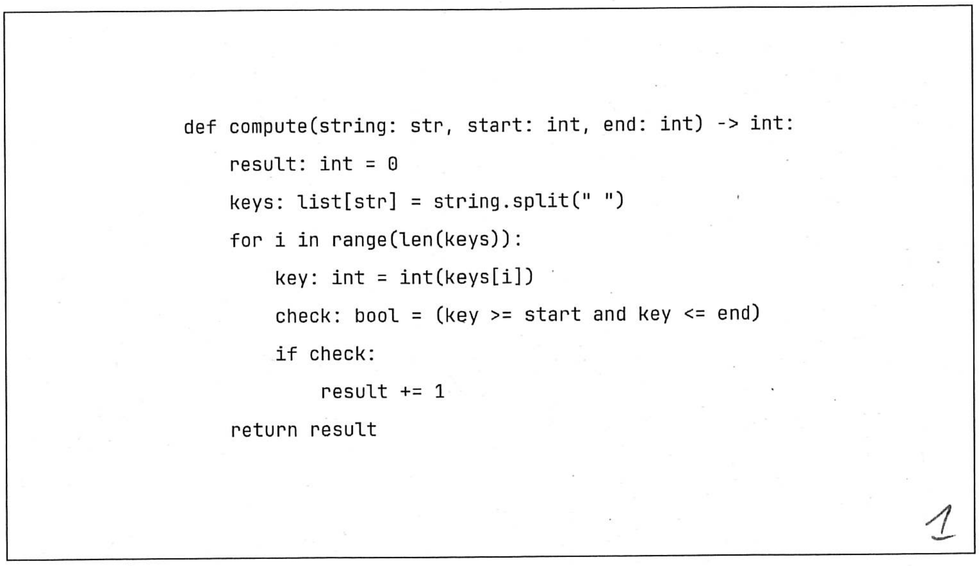
```
def compute(numbers: list[int]) -> str:
    result: list[float] = [0.0] * len(numbers)
    for i in range(len(numbers)):
        if numbers[i] == 0:
            result[i] = 0.0
            continue
        if numbers[i] < 0:
            result[i] = math.sqrt(-1 * numbers[i])
        else:
            result[i] = math.sqrt(numbers[i])
    return str(result)
```

mag ich einfach nicht, weil offensichtlich ist was der Output ist

smart

1

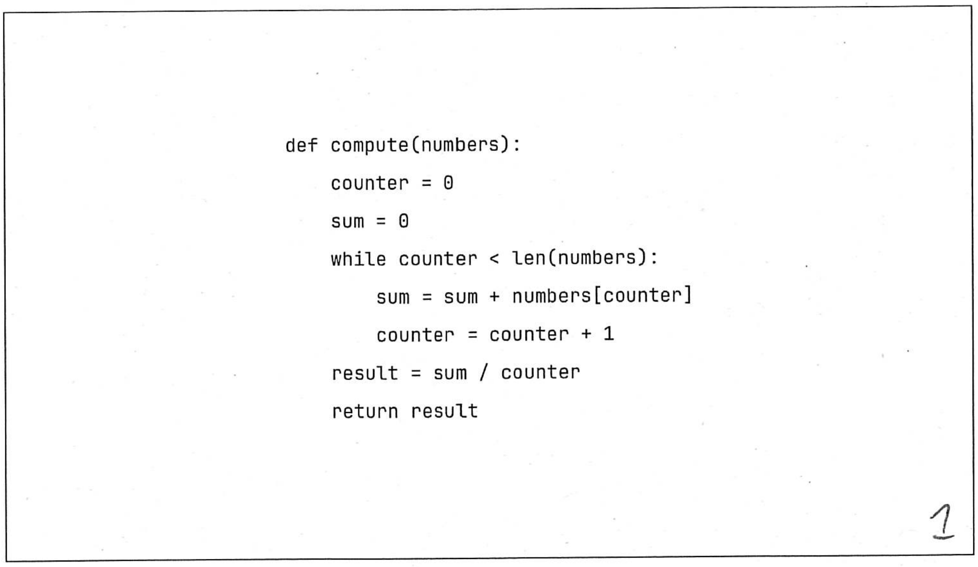
Figure 14: squareRootMT



```python
def compute(string: str, start: int, end: int) -> int:
    result: int = 0
    keys: list[str] = string.split(" ")
    for i in range(len(keys)):
        key: int = int(keys[i])
        check: bool = (key >= start and key <= end)
        if check:
            result += 1
    return result
```

1

Figure 15: countIntegerIntervalMT



```python
def compute(numbers):
    counter = 0
    sum = 0
    while counter < len(numbers):
        sum = sum + numbers[counter]
        counter = counter + 1
    result = sum / counter
    return result
```

1

Figure 16: arrayAverageMN

25574

```
def compute(input: int) -> bool:
    flag: bool = True
    for i in range(2, (input // 2) + 1):
        if input % i == 0:
            flag = False
    return flag
```

1

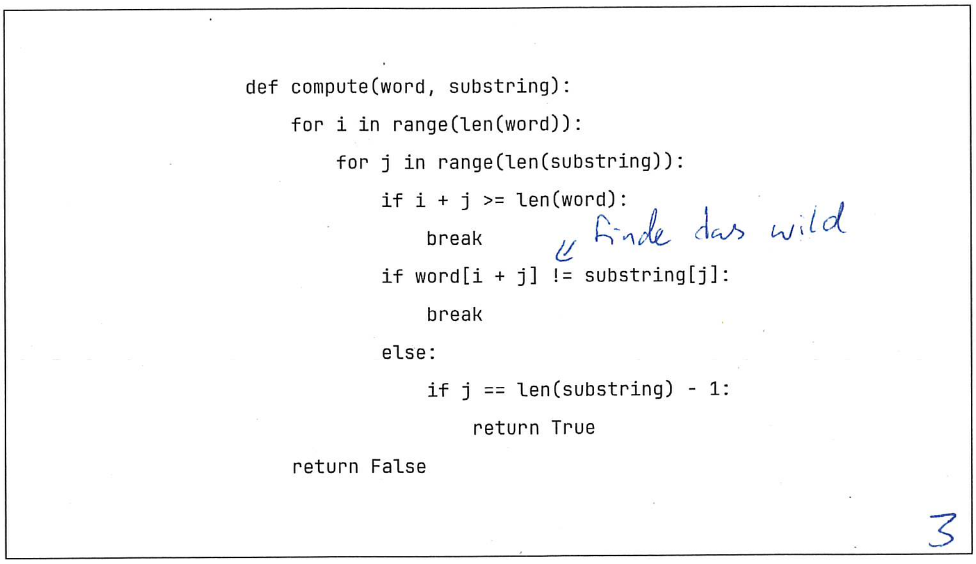
Figure 17: primeMT

```
def compute(number):
    if number == "0":
        return 0
    if number == "1":
        return 1
    if number[-1] == "0":
        return 2 * compute(number[:-1])
    if number[-1] == "1":
        return 1 + 2 * compute(number[:-1])
    return -1
```

wusste nicht, was das heißt. Beim Bsp Input war dann aber klar was das macht

4

Figure 18: binaryToDecimalMN

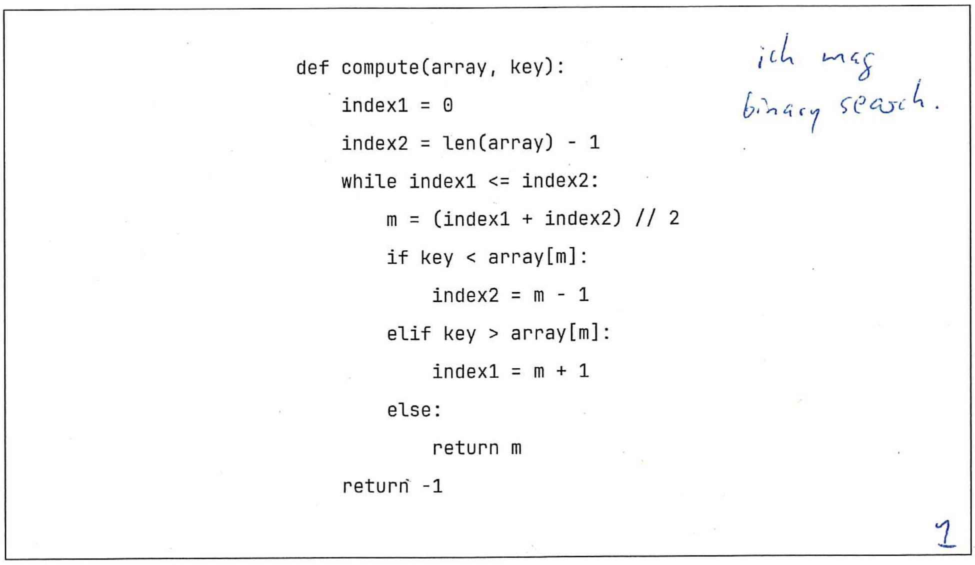


```
def compute(word, substring):
    for i in range(len(word)):
        for j in range(len(substring)):
            if i + j >= len(word):
                break
            if word[i + j] != substring[j]:
                break
            else:
                if j == len(substring) - 1:
                    return True
    return False
```

finde das wild

3

Figure 19: containsSubstringMN



```
def compute(array, key):
    index1 = 0
    index2 = len(array) - 1
    while index1 <= index2:
        m = (index1 + index2) // 2
        if key < array[m]:
            index2 = m - 1
        elif key > array[m]:
            index1 = m + 1
        else:
            return m
    return -1
```

ich mag binary search.

1

Figure 20: binarySearchMN