Przydatne funkcje

Systemy liczbowe

Zliczanie jedynek w zapisie binarnym liczby

In [33]:
```python
def count_bin_ones(n: int) -> int:
    count = 0
    while n:
        n, digit = divmod(n, 2)
        count += digit
    return count
```

In [34]:
```python
count_bin_ones(123)
```

Out[34]: 6

Konwersja liczb naturalnych z systemu dziesiętnego na wskazany

In [35]:
```python
def to_base_init() -> "'to_base' function":
    digit_string = '0123456789abcdefghijklmnopqrstuvwxyz'

    def to_base(num: int, base: int) -> str:
        if not 2 <= base <= 36:
            raise ValueError(f'Wrong base passed. Expected integer from range 2 to 36, got {base}.)')

        digits = []
        while num:
            num, digit = divmod(num, base)
            digits.append(digit_string[digit])
        return ''.join(digits[::-1]) or '0'

    return to_base


to_base = to_base_init()
```

In [36]:
```python
to_base(89743198, 31)
```

Out[36]: '345d6r'

Konwersja liczb naturalnych z systemu dziesiętnego na LICZBĘ w systemie od 2 do 10

In [37]:
```python
def to_base(num: int, base: int) -> int:
    if not 2 <= base <= 10:
        raise ValueError(f"Cannot convert number to base {base}. Supported bases: 2-10.")
    if base == 10:
        return num

    result = 0
    mul = 1
    while num:
        num, digit = divmod(num, base)
        result += mul * digit
        mul *= 10
    return result
```

In [38]:
```python
print(to_base(4565, 7))
```

```
16211
```

Konwersja liczb zmiennoprzecinkowych na system binarny

In [39]:
```python
def to_bin(n: int) -> str:
    dgts = []
    while n:
        n, dgt = divmod(n, 2)
        dgts.append(dgt)
    return ''.join(str(n) for n in dgts[::-1])


def to_bin_float(n: float) -> (str, str):
    # We make use of the limited Pythons's precision of floats and iterate till the remainder is not equal to
    fixed_point = int(n // 1)
    fixed_point_bin = to_bin(fixed_point)
    n -= fixed_point
    floating_point = []
    while n:
        dgt, n = divmod(n*2, 1)
        floating_point.append(str(int(dgt)))
    return fixed_point, ''.join(floating_point)
```

In [40]:
```python
to_bin_float(.1)
```

Out[40]: (0, '0001100110011001100110011001100110011001100110011001101')

Konwersja liczby ze wskazanego systemu liczbowego na dziesiętny

In [41]:
```python
def to_decimal_init() -> "'to_base' function":
    digits_dict = {digit: value for value, digit in enumerate('0123456789abcdefghijklmnopqrstuvwxyz')}

    def to_decimal(num: str, base: int) -> int:
        if not 2 <= base <= 36:
            raise ValueError(f'Wrong base passed. Expected integer from range 2 to 36, got {base}.)')

        result = 0
        num = num.lower()
        for power in range(len(num)):
            digit = num[-power-1]
            result += digits_dict[digit] * base ** power
        return result

    return to_decimal


to_decimal = to_decimal_init()
```

In [42]:
```python
to_decimal('xa', 23), to_decimal('GaReK', 36), to_decimal('PythonIsAwesome', 28)
```

Out[42]: (769, 27375932, 4779432462672532216566)

Obliczanie silni

Oblicznie silni (iteracyjnie)

In [43]:
```python
def factorial(n: int) -> int:
    if n < 0:
        raise ValueError('Cannot calculate factorial of negative numbers')

    res = 1
    for i in range(2, n+1):
        res *= i
    return res
```

In [44]:
```python
factorial(5)
```

Out[44]: 120

Obliczanie silni (rekurencyjnie)

In [45]:
```python
def factorial(n: int) -> int:
    return 1 if n <= 1 else n * factorial(n-1)
```

In [46]:
```python
factorial(7)
```

Out[46]: 5040

Obliczanie silni (rekurencyjnie z cachowaniem wartości z użyciem słownika)

In [47]:
```python
def factorial(n: int, *, cache={}) -> int:
    if n <= 1:
        return 1
    if n not in cache:
        cache[n] = n * factorial(n-1, cache=cache)
        print(cache)  # REMOVE ME
    return cache[n]
```

In [48]:
```python
print(factorial(10))
print(factorial(15))
print(factorial(8))  # No more calculations required as we have already calculated 8!
```

```
{2: 2}
{2: 2, 3: 6}
{2: 2, 3: 6, 4: 24}
{2: 2, 3: 6, 4: 24, 5: 120}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800}
3628800
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800, 11: 39916800}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800, 11: 39916800, 12: 479001600}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800, 11: 39916800, 12: 479001600, 13:
6227020800}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800, 11: 39916800, 12: 479001600, 13:
6227020800, 14: 87178291200}
{2: 2, 3: 6, 4: 24, 5: 120, 6: 720, 7: 5040, 8: 40320, 9: 362880, 10: 3628800, 11: 39916800, 12: 479001600, 13:
6227020800, 14: 87178291200, 15: 1307674368000}
1307674368000
40320
```

Obliczanie silni (rekurencyjnie z cachowaniem wartości z użyciem dekoratora)

In [49]:
```python
def memoized(fn):
    cache = {}  # Cached values will be stored here

    def inner(arg):
        if arg not in cache:
            cache[arg] = fn(arg)
        return cache[arg]

    return inner


@memoized
def factorial(n: int) -> int:
    return 1 if n <= 1 else n * factorial(n-1)
```

In [50]:
```python
factorial(50)
```

Out[50]: 30414093201713378043612608166064768844377641568960512000000000000

Ostatnia niezerowa cyfra silni

In [51]:
```python
def last_non_zero_factorial_digit_init():
    last_factorials_digits = 1, 2, 6, 4
    last_pow2_digits = 2, 4, 8, 6

    def last_non_zero_factorial_digit(num: int) -> int:
        if num < 5:
            return last_factorials_digits[max(num-1, 0)]

        remainder = num//5
        pow2 = last_pow2_digits[remainder % 4 - 1]
        mul = 1
        for n in range(remainder*5+1, num+1):
            mul = (mul * n) % 10

        return (last_non_zero_factorial_digit(remainder) * pow2 * mul) % 10

    return last_non_zero_factorial_digit


last_non_zero_factorial_digit = last_non_zero_factorial_digit_init()
```

In [52]:
```python
last_non_zero_factorial_digit(38712974173298231209412)
```

Out[52]: 4

Ciąg Fibonacciego

Szybkie obliczanie wyrazów ciagu Fibonacciego

In [53]:
```python
from math import sqrt

def fib_init():
    sqrt_5 = sqrt(5)
    mul = 1 / sqrt_5
    a = (1+sqrt_5)/2
    b = (1-sqrt_5)/2

    def fib(n):
        return int(mul*(a**n - b**n))

    return fib


fib = fib_init()
```

In [54]:
```python
fib(10)
```

Out[54]: 55

Obliczanie wskazanej liczby początkowych wyrazów ciągu Fibonacciego

In [55]:
```python
def fibs(n: int) -> list:
    values = [1, 1]
    if n <= 0:
        return []
    if n == 1:
        return [1]
    if n == 2:
        return values

    for _ in range(n-2):
        values.append(values[-1] + values[-2])

    return values
```

In [56]:
```python
fibs(10)
```

Out[56]: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

Obliczanie kolejnych wyrazów ciągu Fibonacciego nie większych niż przekazany

In [57]:
```python
def fibs_2(max_num: int) -> list:
    if max_num <= 0:
        return []

    values = [1, 1]
    while True:
        new_fib = values[-1] + values[-2]
        if new_fib > max_num:
            break
        values.append(new_fib)

    return values
```

In [58]:
```python
fibs_2(100)
```

Out[58]: [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

Wyznaczanie następnej liczby większej od podanej, której nie jest sumą spójnego podciągu Fibonacciego

In [59]:
```python
def generate_fibs(bound: int) -> list:
    fibs = [1, 1]
    while fibs[-1] < bound:
        fibs.append(fibs[-1] + fibs[-2])
    return fibs


def get_next_matching_num(num: int) -> int:
    fibs = generate_fibs(num)
    while True:
        i, j = 0, 1
        curr_sum = fibs[0]
        num += 1

        if num > fibs[-1]:
            fibs.append(fibs[-1] + fibs[-2])

        while i < j < len(fibs):
            if curr_sum == num:
                break
            if curr_sum < num:
                curr_sum += fibs[j]
                j += 1
            else:
                curr_sum -= fibs[i]
                i += 1
        else:
            return num
```

In [60]:
```python
get_next_matching_num(10)
```

Out[60]: 14

Liczby pierwsze

Prosty algorytm na sprawdzanie, czy liczba naturalna jest pierwsza

In [61]:
```python
from math import sqrt


def is_prime(num: int) -> bool:
    if num in (2, 3):
        return True
    if not num % 2 or not num % 3 or num < 2:
        return False

    for div in range(5, int(sqrt(num))+1, 6):
        if not num % div or not num % (div + 2):
            return False
    return True
```

In [62]:
```python
is_prime(137)
```

Out[62]: True

Algorytm do generowania liczb pierwszych nie większych niż wskazana

In [63]:
```python
def sieve_of_eratosthenes(n: int) -> list:
    primes = []
    to_skip = set()

    for val in range(2, n+1):
        if val not in to_skip:
            primes.append(val)
            to_skip.update(range(val*val, n+1, val))

    return primes
```

In [64]:
```python
sieve_of_eratosthenes(71)
```

Out[64]: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71]

Liczby względnie pierwsze

In [65]:
```python
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a


def are_relative_primes(a, b):
    return gcd(abs(a), abs(b)) == 1
```

In [66]:
```python
are_relative_primes(-123, 55)
```

Out[66]: True

Liczby złożone

Sprawdzanie, czy liczba jest złożona

In [67]:
```python
from math import sqrt


def is_composite(num: int) -> bool:
    if num < 4:
        return False
    if not num % 2 or not num % 3:
        return True

    for div in range(5, int(sqrt(num))+1, 6):
        if not num % div or not num % (div + 2):
            return True
    return False
```

In [68]:
```python
is_composite(567)
```

Out[68]: True

Dzielniki liczby

Obliczanie wszystkich dzielników liczby naturalnej (krótszy zapis)

In [69]:
```python
from math import sqrt

def all_factors(n: int) -> set:
    step = n % 2 + 1  # If a number is prime, skip even values as they cannot be factors of an odd number
    return {
        factor for div in range(1, int(sqrt(n))+1, step) if not n % div
        for factor in (div, n//div)
    }
```

In [70]:
```python
all_factors(120)
```

Out[70]: {1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120}

Obliczanie wszystkich dzielników liczby naturalnej (dłuższy zapis)

In [71]:
```python
from math import sqrt

def all_factors(n: int) -> set:
    step = n % 2 + 1  # If a number is prime, skip even values as they cannot be factors of an odd number
    result = set()
    for i in range(1, int(sqrt(n))+1, step):
        div, mod = divmod(n, i)
        if not mod:
            result.add(div)
            result.add(i)
    return result
```

In [72]:
```python
all_factors(120)
```

Out[72]: {1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 20, 24, 30, 40, 60, 120}

Obliczanie czynników pierwszych liczby naturalnej

In [73]:
```python
from math import sqrt

def prime_factors(n: int) -> list:
    factors = []

    # Check if a number is valid
    if n < 4:
        return factors

    n_cp = n  # For prime numbers cases
    # Handle case when a number given is even to reduce a number of iterations
    if n % 2 == 0:
        factors.append(2)
        while not n % 2:
            n //= 2

    # Check if any of odd numbers not greater than a square root of a number divides this number
    for div in range(3, int(sqrt(n))+1, 2):
        if not n % div:
            factors.append(div)
            n //= div

            while not n % div:
                n //= div

            if n == 1:
                return factors

    # For prime numbers cases
    if n != n_cp and n > 1:
        factors.append(n)
    return factors
```

In [74]:
```python
prime_factors(0)
```

Out[74]: []

In [75]:
```python
prime_factors(1024)
```

Out[75]: [2]

In [76]:
```python
prime_factors(437)
```

Out[76]: [19, 23]

Największy wspólny dzielnik i najmniejsza wspólna wielokrotność

Największy wspólny dzielnik 2 liczb całkowitych (rekurencyjne)

In [77]:
```python
def calc_gcd(a, b):
    return a if not b else calc_gcd(b, a % b)

def gcd(a: int, b: int) -> int:
    return calc_gcd(abs(a), abs(b))
```

In [78]:
```python
gcd(960, 216)
```

Out[78]: 24

Największy wspólny dzielnik 2 liczb całkowitych (nierekurencyjne)

In [79]:
```python
def gcd(a: int, b: int) -> int:
    a = abs(a)
    b = abs(b)
    while b:
        a, b = b, a % b
    return a
```

In [80]:
```python
gcd(960, 216)
```

Out[80]: 24

Największy wspólny dzielnik 2 i więcej liczb całkowitych (Konieczne jest użycie którejś z powyższych funkcji)

In [81]:
```python
def gcd_many_values(*nums: int) -> int:
    if len(nums) < 2:
        raise ValueError(f'Expected at least 2 values, got {len(nums)}')

    res = nums[0]
    i = 1
    while i < len(nums) and res != 1:
        res = gcd(res, nums[i])
        i += 1
    return res
```

In [82]:
```python
gcd_many_values(785, 50, 15)
```

Out[82]: 5

Najmniejsza wspólna wielokrotność 2 liczb naturalnych (konieczne jest użycie którejś z funkcji gcd)

In [83]:
```python
def calc_lcm(a: int, b: int) -> int:
    return a * b // gcd(a, b)


def lcm(a: int, b: int) -> int:
    if a < 0 or b < 0:
        raise ValueError(f'Cannot calculate lowest common multiple of negative numbers')
    return calc_lcm(a, b)
```

In [84]:
```python
lcm(5, 13)
```

Out[84]: 65

Najmniejsza wspólna wielokrotność dla 2 lub więcej liczb naturalnych

In [85]:
```python
def lcm_many_values(*nums: int) -> int:
    if len(nums) < 2:
        raise ValueError(f'Expected at least 2 values, got {len(nums)}')

    res = nums[0]
    i = 1
    while i < len(nums):
        res = lcm(res, nums[i])
        i += 1
    return res
```

In [86]:
```python
lcm_many_values(7, 3, 5)
```

Out[86]: 105

Pozostałe liczbowe

Sprawdzanie, czy liczba jest doskonała

In [87]:
```python
def is_perfect_number(n: int) -> bool:
    if n <= 0:
        raise ValueError('Non-positive number cannot be interpreted as positive number')
    factors = all_factors(n)
    factors.remove(n)
    return sum(factors) == n
```

In [88]:
```python
is_perfect_number(496)
```

Out[88]: True

Liczby zaprzyjaźnione

In [89]:
```python
def are_numbers_friendly(a: int, b: int) -> bool:
    if a == b:
        return False

    a_factors = all_factors(a)
    a_factors.remove(a)
    b_factors = all_factors(b)
    b_factors.remove(b)

    return sum(a_factors) == b and sum(b_factors) == a
```

In [90]:
```python
are_numbers_friendly(284, 220)
```

Out[90]: True

Obliczanie wartości liczby PI

In [91]:
```python
from math import sqrt

def calc_pi(prec: int = 50) -> float:
    value = 1
    multiplier = sqrt(.5)
    for _ in range(prec):
        value *= multiplier
        multiplier = sqrt(.5 + .5 * multiplier)

    return 2 / value
```

In [92]:
```python
calc_pi()
```

Out[92]: 3.141592653589794

Obliczanie wartości liczby e

In [93]:
```python
def calc_e(prec: int = 20) -> float:
    fact = 1
    e = 0
    for i in range(1, prec+1):
        e += 1/fact
        fact *= i
    return e
```

In [94]:
```python
calc_e()
```

Out[94]: 2.7182818284590455

Palindrom

Sprawdzanie, czy sekwencja jest palindromem

In [95]:
```python
def is_palindrome(seq: 'any sequence') -> bool:
    for i in range(len(seq)//2):
        if seq[i] != seq[-i-1]:
            return False
    return True
```

In [96]:
```python
is_palindrome('12321'), is_palindrome('1221'), is_palindrome('1'), is_palindrome('123421')
```

Out[96]: (True, True, True, False)

Pobieranie danych

Tworzenie tablicy 2-wymiarowej

In [97]:
```python
def create_matrix(rows: int, columns: int, *, fill_with=0) -> '2-dimensional list':
    return [[fill_with]*columns for _ in range(rows)]
```

In [98]:
```python
create_matrix(5, 3)
```

Out[98]: [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]

Wypełnianie tablicy 2-wymairowej

In [99]:
```python
def fill_matrix(matrix: list):
    for i in range(len(matrix)):
        while True:
            row = input().split()
            if len(row) == len(matrix[i]):
                break
        for j, val in enumerate(row):
            matrix[i][j] = int(val)
```

In [100…
```python
matrix = create_matrix(5, 3)
# fill_matrix(matrix)
```

In [101…
```python
matrix
```

Out[101… [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]

Tworzenie i wypełnianie tablicy 1-wymiarowej

In [103…
```python
n = int(input())
t = [int(input()) for _ in range(n)]
```

```
10
7
9
12
40
11
-13
72
18
-11
0
```

In [104…
```python
t
```

Out[104… [7, 9, 12, 40, 11, -13, 72, 18, -11, 0]

Listy (tablice)

Podciągi w tablicy 1-wymiarowej

Najdłuższy spójny podciąg arytmetyczny

In [105…
```python
def longest_common_arithmetic_subsequence(seq: 'sequence of numbers') -> int:
    if len(seq) < 2:
        return 0

    max_length = curr_length = 2
    for i in range(1, len(seq)-1):
        if seq[i-1] + seq[i+1] == 2*seq[i]:
            curr_length += 1
            if curr_length > max_length:
                max_length = curr_length
        else:
            curr_length = 2
    return max_length
```

In [106…
```python
longest_common_arithmetic_subsequence((1, 2, 3, 5, 4, 3, 2, 1))
```

Out[106… 5

Najdłuższy spójny podciąg geometryczny

In [107…
```python
def longest_common_geometric_subsequence(seq: 'sequence of numbers') -> int:
    if len(seq) < 2:
        return 0

    max_length = curr_length = 2
    for i in range(1, len(seq)-1):
        if seq[i-1] * seq[i+1] == seq[i]**2:
            curr_length += 1
            if curr_length > max_length:
                max_length = curr_length
        else:
            curr_length = 2
    return max_length
```

In [108…
```python
longest_common_geometric_subsequence((1, 2, 4, 16, 64, 256))
```

Out[108… 4

Najdłuższy spójny podciąg, będący palindromem

In [109…
```python
def palindrome_length(seq: 'sequence of numbers', left_idx: int, right_idx: int) -> int:
    while 0 <= left_idx and right_idx < len(seq) and seq[left_idx] == seq[right_idx]:
        left_idx -= 1
        right_idx += 1
    return right_idx - left_idx - 1


def longest_commmon_palindrome_subsequence(seq: 'sequence of numbers') -> int:
    max_length = 0
    start_idx = 0

    # Loop only while it is still possible to find the longest subsequence
    while len(seq)-1 - start_idx > max_length//2:
        # For palindromes of odd elements counts
        odd_length = palindrome_length(seq, start_idx, start_idx)
        # For palindromes of even elements counts
        even_length = palindrome_length(seq, start_idx, start_idx+1)

        curr_length = max(odd_length, even_length)

        if curr_length > max_length:
            max_length = curr_length

        start_idx += 1

    return max_length
```

In [110…
```python
longest_commmon_palindrome_subsequence([1, 2, 1, 6, 7, 4, 5, 6, 5, 4, 2, 1, 4, 4, 1, 2])
```

Out[110… 6

Najdłuższy spójny podciąg rosnący o największej sumie wartości, która jest równa sumie indeksów miejsc, na których stoją

In [111…
```python
def longest_common_increasing_indices_values_sum_subsequence(seq: 'sequence of numbers') -> int:
        max_length = current_length = 1
        sum_indices = 0
        sum_values = prev_val = seq[0]

        for i in range(1, len(seq)):
            if seq[i] > prev_val:
                current_length += 1
                sum_indices += i
                sum_values += seq[i]
                if current_length > max_length and sum_indices == sum_values:
                    max_length = current_length
            else:
                prev_val = sum_values = seq[i]
                current_length = 1
                sum_indices = i

        return 0 if max_length == 1 else max_length
```

In [112…
```python
t = [34, 23, 12, 2313, 341, 123, 14, 93, 83, 61,
     82, 283, 278, 2, 5, 21, 30, 68, 123, 93, 1,
     2, 60, 90, 123, 342, 12, 42, 32, 231]
longest_common_increasing_indices_values_sum_subsequence(t)
```

Out[112… 4

Spójny podciąg o zadanej długości i największej sumie

In [113…
```python
def max_sum_consistent_subsequence(seq: 'sequence of numbers', max_length: int) -> int:
    max_sum = 0

    for i in range(len(seq)):  # Index of the subsequence's beginning
        curr_sum = 0
        for j in range(i, min(i + max_length, len(seq))):  # Currently summed value's index
            curr_sum += seq[j]

            if curr_sum > max_sum:
                max_sum = curr_sum

    return max_sum
```

In [114…
```python
t = [34, 23, 12, 2313, 341, 123, 14, 93, 83, 61,
     82, 283, 278, 2, 5, 21, 30, 68, 123, 93, 1,
     2, 60, 90, 123, 342, 12, 42, 32, 231]
max_sum_consistent_subsequence(t, 5)
```

Out[114… 2884

Tablice 2-wymiarowe (algorytmy różne)

Wypełnianie 2-wymiarowej kwadratowej listy po spirali (tu kolejnymi liczbami)

In [115…
```python
def fill_matrix_spiral(matrix: [[int]]):
    if len(matrix) != len(matrix[0]):
        raise ValueError('Wrong dimensions of the matrix passed. Number of rows differs from a number of column

    num = 0
    def next_int():
        nonlocal num
        num += 1
        return num

    end_idx = len(matrix)-1
    center_idx = len(matrix) // 2
    for begin_idx in range(center_idx):
        for i in range(begin_idx, end_idx):
            matrix[begin_idx][i] = next_int()
        for i in range(begin_idx, end_idx):
            matrix[i][-begin_idx - 1] = next_int()
        for i in range(end_idx, begin_idx, -1):
            matrix[-begin_idx - 1][i] = next_int()
        for i in range(end_idx, begin_idx, -1):
            matrix[i][begin_idx] = next_int()
        end_idx -= 1

    if len(matrix) % 2:
        matrix[center_idx][center_idx] = next_int()
```

In [116…
```python
t = create_matrix(6, 6)  # I use function declared above
fill_matrix_spiral(t)
```

In [117…
```python
t
```

Out[117…
```
[[1, 2, 3, 4, 5, 6],
 [20, 21, 22, 23, 24, 7],
 [19, 32, 33, 34, 25, 8],
 [18, 31, 36, 35, 26, 9],
 [17, 30, 29, 28, 27, 10],
 [16, 15, 14, 13, 12, 11]]
```

Sumowanie wartości w wierszu

In [118…
```python
def row_sum(matrix: '2-dimensional sequence of numbers', row_idx: int):
    return sum(matrix[row_idx])
```

In [119…
```python
row_sum(t, 3)  # See t matrix above
```

Out[119… 155

Sumowanie wartości w kolumnie

In [120…

```python
def col_sum(matrix: '2-dimensional sequence of numbers', col_idx: int):
    sum_ = 0
    for row_idx in range(len(matrix)):
        sum_ += matrix[row_idx][col_idx]
    return sum_
```

In [121…

```python
row_sum(t, -1)  # See t matrix above
```

Out[121… 81

Sumowanie wartości w wierszu i kolumnie jednocześnie z wyrzuceniem wartości w punkcie przecięcia

In [122…

```python
def row_col_sum(matrix: '2-dimensional sequence of numbers', row_idx: int, col_idx: int):
    sum_ = -2 * matrix[row_idx][col_idx]

    # Get sum of values in the specified row
    for i in range(len(matrix[row_idx])):
        sum_ += matrix[row_idx][i]

    # Get sum of values in the specified column
    for i in range(len(matrix)):
        sum_ += matrix[i][col_idx]

    return sum_
```

In [123…

```python
row_col_sum(t, 1, -1)
```

Out[123… 154

Sumowanie wartości po przekątnej (z lewego górnego narożnika do prawego dolnego)

In [124…

```python
def diagonal_tl_br_sum(matrix: '2-dimensional sequence of numbers', row_idx: int, col_idx: int):
    row_idx = row_idx if row_idx > 0 else len(matrix) + row_idx
    col_idx = col_idx if col_idx > 0 else len(matrix[0]) + col_idx

    i = -min(row_idx, col_idx)
    sum_ = 0
    while row_idx+i < len(matrix) and col_idx+i < len(matrix[0]):
        c = (row_idx+i, col_idx+i)
        sum_ += matrix[row_idx+i][col_idx+i]
        i += 1

    return sum_
```

In [125…

```python
diagonal_tl_br_sum(t, 2, -1)
```

Out[125… 36

Sumowanie wartości po przekątnej (z prawego górnego narożnika do lewego dolnego)

In [126…

```python
def diagonal_tr_bl_sum(matrix: '2-dimensional sequence of numbers', row_idx: int, col_idx: int):
    row_idx = row_idx if row_idx > 0 else len(matrix) + row_idx
    col_idx = col_idx if col_idx > 0 else len(matrix[0]) + col_idx

    i = min(row_idx, len(matrix)-1 - col_idx)
    sum_ = 0
    while row_idx-i < len(matrix) and col_idx+i < len(matrix[0]):
        c = (row_idx-i, col_idx+i)
        sum_ += matrix[row_idx-i][col_idx+i]
        i -= 1

    return sum_
```

In [127…

```python
diagonal_tr_bl_sum(t, 2, -1)
```

Out[127… 76

Sumowanie wartości na krzyż (po obu przekątnych; konieczne jest użycie powyższych dwóch funkcji)

In [128…

```python
def diagonal_sums(matrix: '2-dimensional sequence of numbers', row_idx: int, col_idx: int):
    tl_br_sum = diagonal_tl_br_sum(matrix, row_idx, col_idx)
    tr_bl_sum = diagonal_tr_bl_sum(matrix, row_idx, col_idx)
    return tl_br_sum + tr_bl_sum - matrix[row_idx][col_idx]  # We calculated one value 2 times
```

In [129…

```python
diagonal_sums(t, 4, -3)
```

Out[129… 176

Suma otaczających elementów (obwódka z 8 elementów)

In [130…

```python
def get_surrounding_sum(matrix: '2-dimensional sequence of numbers', row_idx: int, col_idx: int):
    sum_ = -matrix[row_idx][col_idx]
    for i in range(row_idx-1, row_idx+2):
        for j in range(col_idx-1, col_idx+2):
            sum_ += matrix[i][j]
    return sum_
```

In [131…

```python
get_surrounding_sum(t, 4, 3)  # Uważaj na ujemne wartości dla punktów na krawędzi matrycy (zwrócą złą wartość)
```

Out[131… 192

Spójny podciąg o największej sumie i zadanej długości

In [132…

```python
def max_sum_consistent_subsequence(matrix: '2-dimensional sequence of numbers', max_length: int = 10) -> int:
    max_sum = 0

    # Search in rows
    for row_idx in range(len(matrix)):
        for i in range(len(matrix[row_idx])):  # Index of the sequence's beginning
            curr_sum = 0
            for j in range(i, min(i + max_length, len(matrix[row_idx]))):  # Currently summed value's index
                curr_sum += matrix[row_idx][j]

                if curr_sum > max_sum:
                    max_sum = curr_sum

    # Search in columns
    for col_idx in range(len(matrix[0])):
        for i in range(len(matrix)):  # Index of the sequence's beginning
            curr_sum = 0
            for j in range(i, min(i + max_length, len(matrix))):  # Currently summed value's index
                curr_sum += matrix[j][col_idx]

                if curr_sum > max_sum:
                    max_sum = curr_sum

    return max_sum
```

In [133…

```python
max_sum_consistent_subsequence(t, 3)
```

Out[133… 102

Linearyzacja 2-wymiarowej tablicy (listy)

In [134…

```python
def f(matrix: '2-dimensional sequence of numbers'):
    # This function works with non-square matrices as well as with square ones
    rows, cols = len(matrix), len(matrix[0])
    length = rows*cols  # A length of linearized matrix
    for i in range(length):
        val1 = matrix[i//cols][i%cols]
        for j in range(i+1, length):
            val2 = matrix[j//cols][j%cols]
            print(val1, val2)  # REMOVE ME
```

In [135…

```python
t = [
    [1, 2, 3],
    [4, 5, 6]
]
f(t)
```

```
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
```

Linearyzacja 3-wymiarowej tablicy (listy)

In [136…

```python
def f(matrix3d: '3-dimensional sequence of numbers'):
    levels, rows, cols = len(matrix3d), len(matrix3d[0]), len(matrix3d[0][0])
    level_size = rows*cols      # A size of a matrix in a field
    length = levels*level_size  # A length of linearized matrix
    for i in range(length):
        k1 = i%level_size  # The first pointer in the current level
        val1 = matrix3d[i//level_size][k1//cols][k1%cols]
        for j in range(i+1, length):
            k2 = j%level_size  # The second pointer in the current level
            val2 = matrix3d[j//level_size][k2//cols][k2%cols]
            print(val1, val2)  # REMOVE ME
```

In [137…

```python
t = [
    [
        [1, 2],
        [3, 4]
    ],
    [
        [5, 6],
        [7, 8]
    ]
]
f(t)
```

```
1 2
1 3
1 4
1 5
1 6
1 7
1 8
2 3
2 4
2 5
2 6
2 7
2 8
3 4
3 5
3 6
3 7
3 8
4 5
4 6
4 7
4 8
5 6
5 7
5 8
6 7
6 8
7 8
```

Algorytmy szachowe

Sprawdzanie, czy hetmany (królowe) się szachują (dla przekazanej listy koordynatów)

In [138…

```python
def check_if_queens_checkmate(num_rows, num_cols, coords):
    taken_rows = [0]*num_rows
    taken_cols = [0]*num_cols
    taken_tl_br_diagonal = [0]*(num_cols + num_rows)
    taken_tr_bl_diagonal = [0]*(num_cols + num_rows)

    for r, c in coords:
        if taken_rows[r] or taken_cols[c] or taken_tl_br_diagonal[r+c] or taken_tr_bl_diagonal[r-c]:
            return True  # Return True if found a queen that is in check
        else:
            taken_rows[r] = taken_cols[c] = taken_tl_br_diagonal[r+c] = taken_tr_bl_diagonal[r-c] = 1

    return False
```

In [139…

```python
t = [(0, 0), (1, 2), (2, 3), (5, 1)]
rows = 4
cols = 6
print(check_if_queens_checkmate(rows, cols, t))
```

```
True
```

Pozostałe

Generowanie 2-wymiarowej listy losowych liczb całkowitych

In [140…

```python
import random


def random_matrix(rows: int, cols: int, min_num: int, max_num: int) -> [[int]]:
    return [[random.randint(min_num, max_num) for _ in range(cols)] for _ in range(rows)]
```

In [141…

```python
N = 5
min_num, max_num = 0, 100
t = random_matrix(N, N, min_num, max_num)
print(*t, sep='\n')
```

```
[36, 93, 50, 1, 87]
[39, 38, 98, 93, 59]
[65, 65, 40, 34, 33]
[2, 90, 34, 30, 97]
[93, 82, 35, 12, 29]
```

Generowanie 1-wymiarowej listy losowych liczb całkowitych

In [142…

```python
import random


def random_list(length: int, min_num: int, max_num: int) -> [int]:
    return [random.randint(min_num, max_num) for _ in range(length)]
```

In [143…

```python
N = 25
min_num, max_num = 0, 10
t = random_list(N, min_num, max_num)
print(t)
```

```
[10, 7, 4, 7, 3, 5, 8, 2, 0, 0, 1, 8, 2, 6, 9, 2, 5, 1, 9, 1, 7, 9, 9, 8, 0]
```

Tworzenie zbioru cyfr, z jakich zbudowana jest liczba naturalna

In [144…

```python
def get_number_digits(num):
    digits = set()

    while num and len(digits) < 10:  # If a number has all possible digits, end the loop
        num, dgt = divmod(num, 10)
        digits.add(dgt)

    return digits
```

In [145…

```python
get_number_digits(2341)
```

Out[145… {1, 2, 3, 4}