Лабораторная работа 6

Функции и составные алгоритмы

В задачах используйте уже написанные функции, если условие прямо на это указывает.

Задача 1. Проверка простоты

Опишите функцию `is_prime(number)`, возвращающую для целого положительного `number` значение `True`, если число простое, и `False` в противном случае. Используйте цикл `for`, `break` и `else` у цикла.

In [1]:
```
from math import isqrt

def is_prime(number):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [11]:
```
assert is_prime(97)
assert not is_prime(1)
assert not is_prime(100)
```

Задача 2. Следующее простое число

Используя функцию `is_prime`, опишите функцию `next_prime(number)`, возвращающую первое простое число, большее `number`.

In [ ]:
```
def next_prime(number):
    ...
```

Задача 3. Количество простых до границы

Опишите функцию `count_primes(limit)`, возвращающую количество простых чисел от 2 до `limit` включительно.

In [ ]:
```
def count_primes(limit):
    ...
```

Задача 4. Наибольший общий делитель

Опишите функцию `gcd(a, b)`, возвращающую НОД двух натуральных чисел алгоритмом Евклида (без рекурсии!).

In [ ]:
```
def gcd(a, b):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [16]:
```
assert gcd(54, 24) == 6
```

Задача 5. Наименьшее общее кратное

Опишите функцию `lcm(a, b)`, возвращающую НОК двух натуральных чисел. Используйте функцию `gcd`.

In [ ]:
```
def lcm(a, b):
    ...
```

Задача 6. Сумма собственных делителей

Опишите функцию `sum_proper_divisors(number)`, возвращающую сумму положительных делителей натурального числа `number`, не считая само число.

In [ ]:
```
def sum_proper_divisors(number):
    ...
```

Задача 7. Совершенное число

Опишите функцию `is_perfect(number)`, возвращающую `True`, если целое число равно сумме своих собственных делителей. Используйте функцию из предыдущей задачи.

In [ ]:
```
def is_perfect(number):
    ...
```

Задача 8. Сокращение дроби

Опишите функцию `reduce_fraction(numerator, denominator)`, возвращающую сокращённые числитель и знаменатель. Используйте `gcd`. Числитель и знаменатель считать целыми.

In [ ]:
```
def reduce_fraction(numerator, denominator):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [41]:
```
assert reduce_fraction(12, 18) == (2, 3)
```

Задача 9. Целая степень

Опишите функцию `power_int(base, exponent)`, возводящее вещественное `base` в целую степень `exponent`. Учтите нулевую и отрицательную степень.

In [ ]:
```
def power_int(base, exponent):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов. Если ожидаемое значение равно нулю или очень близко к нему, добавьте в `isclose` абсолютную погрешность `abs_tol`.

In [42]:
```
from math import isclose

assert isclose(power_int(2, -3), 0.125, rel_tol = 1e-9)
```

Задача 10. Сумма положительных аргументов

Опишите функцию `sum_positive(*numbers)`, возвращающую сумму положительных среди переданных ей целых чисел. Вызовите её с несколькими числовыми аргументами.

In [ ]:
```
def sum_positive(*numbers):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [44]:
```
assert sum_positive(-2, 5, 0, 7, -1) == 12
```

Задача 11. Все ли числа делятся

Опишите функцию `all_divisible_by(divisor, *numbers)`, возвращающую `True`, если каждое переданное целое число делится на `divisor`.

In [ ]:
```
def all_divisible_by(divisor, *numbers):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [46]:
```
assert all_divisible_by(3, 6, 9, 12)
assert not all_divisible_by(3, 6, 10, 12)
```

Задача 12. Минимум и максимум аргументов

Опишите функцию `min_max(*numbers)`, возвращающую минимум и максимум среди переданных вещественных чисел (хотя бы одно число есть). Не используйте встроенные `min` и `max`.

In [ ]:
```
def min_max(*numbers):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [51]:
```
mn, mx = min_max(7.5, -2.23, 15.67, 0.01, 9.04)
assert isclose(mn, -2.23, rel_tol = 1e-9)
assert isclose(mx, 15.67, rel_tol = 1e-9)
```

Задача 13. Количество и сумма цифр

Опишите функцию `digit_count_and_sum(number)`, возвращающую количество цифр и сумму цифр положительного целого числа `number`. Распакуйте возвращённые значения в основной программе в переменные, а затем выведите их.

In [53]:
```
def digit_count_and_sum(number):
    ...
```

```
Количество цифр: 5, сумма цифр: 20
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [54]:
```
assert digit_count_and_sum(50702) == (5, 14)
```

Задача 14. Добавление цифры слева

Используя функцию `digit_count_and_sum` из прошлой задачи, опишите функцию `add_left_digit(digit, number)`, добавляющую цифру `digit` слева к положительному целому числу `number` и возвращающую полученное число. Считайте, что $1 \leqslant digit \leqslant 9$.

In [ ]:
```
def add_left_digit(digit, number):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [56]:
```
assert add_left_digit(7, 205) == 7205
```

Задача 15. Число Армстронга

Положительное число называется *числом Армстронга*, если оно равно сумме своих цифр, возведённых в степень, равную количеству цифр. Например, $153 = 1^3 + 5^3 + 3^3$. Используя функцию `digit_count_and_sum` из задачи 13 для нахождения количества цифр, опишите булеву функцию `is_armstrong(number)`.

In [ ]:
```
def is_armstrong(number):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [58]:
```
assert is_armstrong(153)
assert is_armstrong(9474)
assert not is_armstrong(154)
```

В следующих задачах используется функция `is_leap_year(year)`, разобранная в лабораторной работе 5. Она приведена ниже как готовая вспомогательная функция, а не как новая задача. Не забудьте запустить ячейку с данной функцией.

In [59]:
```
def is_leap_year(year):
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)
```

Задача 16. Количество дней в месяце

Используя `is_leap_year`, опишите функцию `month_days(month, year)`, возвращающую количество дней в месяце с номером `month` года `year`. Используйте `match/case` и не используйте списки.

In [ ]:
```
def month_days(month, year):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [61]:
```
assert month_days(12, 2026) == 31
assert month_days(2, 2026) == 28
assert month_days(2, 2000) == 29
assert month_days(4, 2026) == 30
```

Задача 17. Предыдущая дата

Используя `month_days`, опишите функцию `prev_date(day, month, year)`, которая по **правильной** дате возвращает предыдущую дату: новый день, месяц и год.

In [ ]:
```
def prev_date(day, month, year):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [63]:
```
assert prev_date(15, 4, 1994) == (14, 4, 1994)
assert prev_date(1, 3, 2024) == (29, 2, 2024)
assert prev_date(1, 1, 2026) == (31, 12, 2025)
```

Задача 18. Следующая дата

Используя `month_days`, опишите функцию `next_date(day, month, year)`, которая по **правильной** дате возвращает следующую дату: новый день, месяц и год.

In [ ]:
```
def next_date(day, month, year):
    ...
```

Пример проверки. Запустите ячейку с функцией, затем эту ячейку. Допишите еще несколько тестов.

In [65]:
```
assert next_date(28, 2, 2023) == (1, 3, 2023)
assert next_date(28, 2, 2024) == (29, 2, 2024)
assert next_date(31, 12, 2025) == (1, 1, 2026)
```

In [ ]:
```

```