Лабораторная работа №6
«Функции»

Задание №1
Собственные функции

Вариант 1

1. **Функция статистики числового списка**
   Напишите функцию analyze_numbers(lst), которая принимает список чисел и возвращает кортеж с минимальным, максимальным, суммарным и средним значениями.
   Особенности: Если список пустой, вернуть кортеж с четырьмя значениями None.

2. **Функция частотного анализа символов**
   Напишите функцию char_frequency(text), которая принимает строку и возвращает словарь, где ключи — уникальные символы (без учета регистра, игнорируя пробелы и пунктуацию), а значения — число их вхождений в строку.

3. **Функция фильтрации строк по длине с аргументом по умолчанию**
   Напишите функцию filter_long_words(words, min_length=5), которая принимает список строк и возвращает список слов, длина которых строго больше min_length. Если аргумент min_length не указан, использовать значение 5.

Вариант 2

1. **Функция произведения чисел с переменным числом аргументов**
   Напишите функцию multiply(*args), которая принимает неограниченное число числовых аргументов и возвращает их произведение. Если аргументы не переданы, функция должна вернуть None.

2. **Функция обработки именованных аргументов**
   Напишите функцию process_kwargs(**kwargs), которая принимает произвольное число именованных аргументов. Функция должна вернуть новый словарь, где каждый ключ преобразован в верхний регистр, а если значение является числом, то оно удвоено.

3. **Функция сортировки списка с указанием порядка**
   Напишите функцию custom_sort(numbers, order='asc'), которая принимает список чисел и необязательный параметр order. Если order равен 'asc' – вернуть список в возрастающем порядке, если 'desc' – в убывающем. При некорректном значении параметра – функция должна вернуть исходный список без изменений.

Вариант 3

1. **Функция удаления символов из строки**
   Напишите функцию remove_chars(text, chars), которая принимает строку и набор символов (в виде списка или множества) и возвращает новую строку, в которой все символы, входящие в chars, удалены (сохраните регистр остальных символов).

2. **Функция разделения списка слов на уникальные и повторяющиеся**
   Напишите функцию split_unique(words), которая принимает список слов и возвращает кортеж из двух списков: первый – список уникальных слов (без повторений), второй – список слов, которые встречались более одного раза. Для поиска использовать множества.

3. **Функция реверсирования каждого слова**
   Напишите функцию reverse_each_word(sentence), которая принимает строку (предложение) и возвращает новую строку, где в каждом слове порядок символов перевёрнут, но порядок слов в предложении остаётся прежним. Используйте методы split и join.

Вариант 4

1. **Функция создания словаря из двух списков**
   Напишите функцию lists_to_dict(keys, values), которая принимает два списка чисел (одинаковой длины) и возвращает словарь, где каждому элементу из первого списка соответствует элемент из второго. Если длины списков различаются, функция должна вернуть сообщение об ошибке или поднять исключение.

2. **Функция группировки слов по длине**
   Напишите функцию group_by_length(words), которая принимает список строк и возвращает словарь, где ключ — длина слова, а значение — список слов данной длины (отсортируйте слова в каждом списке по алфавиту).

3. **Функция преобразования числового списка**
   Напишите функцию transform_numbers(numbers), которая принимает список чисел и возвращает новый список, в котором чётные числа заменены на их квадрат, а нечётные – на их куб.

Вариант 5

1. **Рекурсивная функция вычисления факториала**
   Напишите рекурсивную функцию factorial(n), которая принимает неотрицательное целое число и возвращает его факториал. Если число отрицательное, функция должна вернуть сообщение об ошибке.

2. **Функция нахождения простых делителей**
   Напишите функцию prime_factors(n), которая принимает целое число и возвращает список его простых делителей (каждый делитель должен встречаться только один раз). Используйте вспомогательную функцию для проверки простоты.

3. **Функция суммирования и перемножения цифр числа**
   Напишите функцию digit_operations(n), которая принимает целое число и возвращает кортеж, где первый элемент – сумма цифр числа, а второй – произведение цифр. Преобразуйте число в строку для итерации по цифрам.

Вариант 6

1. **Функция проверки палиндрома**
   Напишите функцию is_palindrome(text), которая принимает строку и возвращает True, если строка является палиндромом (игнорируйте пробелы, знаки препинания и регистр), иначе – False.

2. **Функция фильтрации простых чисел из списка**
   Напишите функцию filter_primes(numbers), которая принимает список чисел и возвращает новый список, состоящий только из простых чисел. Обязательно реализуйте вспомогательную функцию is_prime(num).

3. **Функция разделения списка строк по средней длине**
   Напишите функцию split_by_average_length(strings), которая принимает список строк и возвращает кортеж из двух списков: первый – строки, длина которых больше средней длины всех строк, а второй – остальные.

Вариант 7

1. **Функция поиска подстроки в строке**
   Напишите функцию find_substring_positions(text, sub), которая принимает строку и подстроку и возвращает список позиций (индексов) первого символа всех вхождений подстроки в строку. Используйте цикл и метод find с указанием начального индекса.

2. **Функция вычисления среднего значения через args*
   Напишите функцию average_value(*numbers), которая принимает переменное число числовых аргументов и возвращает их среднее значение. Если аргументы не переданы, вернуть 0.

3. **Функция суммирования списков из словаря**
   Напишите функцию sum_dict_values(data), которая принимает словарь, где значения – списки чисел, и возвращает новый словарь, где для каждого ключа значение – сумма чисел в соответствующем списке. Реализуйте это с помощью словарного включения.

Вариант 8

1. **Функция составления акронима**
   Напишите функцию make_acronym(words), которая принимает список слов и возвращает строку, состоящую из первых букв каждого слова (акроним). Используйте цикл и операцию конкатенации.

2. **Функция преобразования формата даты**
   Напишите функцию reformat_date(date_str), которая принимает дату в формате "dd-mm-yyyy" и возвращает строку с датой в формате "yyyy/mm/dd". Используйте методы split и форматирование строк.

3. **Функция группировки пар (имя, возраст)**
   Напишите функцию group_by_age(pairs), которая принимает список кортежей вида (name, age) и возвращает словарь, где ключ — возраст, а значение — список имён с данным возрастом.

Вариант 9

1. **Функция разделения списка на чётные и нечётные**
   Напишите функцию separate_even_odd(numbers), которая принимает список чисел и возвращает кортеж из двух списков: первый – список чётных, второй – список нечётных чисел.

2. **Функция повторения символов в строке**
   Напишите функцию repeat_chars(text, n), которая принимает строку и целое число n и возвращает новую строку, где каждый символ исходной строки повторяется n раз.

3. **Функция построения частотного словаря для слов**
   Напишите функцию word_frequency(text), которая принимает строку, очищает её от знаков препинания, приводит к нижнему регистру, разбивает на слова и возвращает словарь, где каждому слову соответствует число его вхождений.

Вариант 10

1. **Функция проверки совершенности числа**
   Напишите функцию is_perfect(n), которая принимает целое число и возвращает True, если число совершенное (сумма его собственных делителей равна самому числу), иначе – False. Используйте цикл для поиска делителей.

2. **Функция объединения двух списков в кортежи**
   Напишите функцию zip_lists(list1, list2), которая принимает два списка одинаковой длины и возвращает список кортежей, где каждый кортеж состоит из элементов, стоящих на одинаковых позициях. Если длины списков различаются – функция должна вывести сообщение об ошибке.

3. **Функция преобразования чисел по порогу**
   Напишите функцию process_numbers(numbers, threshold=0), которая принимает список чисел и необязательный параметр threshold. Функция должна возвращать новый список, в котором каждое число больше threshold заменяется на сумму его цифр (полученную путём преобразования числа в строку), а остальные числа остаются без изменений.

Задание №2
Рекурсия

Вариант 1

1. **Рекурсивное вычисление факториала**
   Напишите функцию factorial(n), которая принимает неотрицательное целое число n и возвращает его факториал. Если n < 0, функция должна вернуть сообщение об ошибке.

2. **Рекурсивное вычисление n-го числа Фибоначчи**
   Напишите функцию fibonacci(n), которая возвращает n-е число Фибоначчи (считаем, что первые два числа равны 0 и 1). Учтите базовые случаи и обеспечьте корректную работу для n ≥ 0.

3. **Рекурсивный разворот строки**
   Напишите функцию reverse_str(s), которая принимает строку s и возвращает новую

строку с символами в обратном порядке. Решите задачу без использования срезов и встроенных функций реверса.

Вариант 2

1. **Рекурсивное суммирование элементов вложенного списка**
   Напишите функцию recursive_sum(lst), которая принимает список, элементы которого могут быть как числами, так и вложенными списками, и возвращает сумму всех чисел внутри структуры.
2. **Рекурсивное выравнивание (flatten) вложенного списка**
   Напишите функцию flatten(lst), которая принимает вложенный список (произвольной глубины) и возвращает плоский список со всеми элементами.
3. **Рекурсивное вычисление глубины вложенного списка**
   Напишите функцию max_depth(lst), которая возвращает максимальную глубину вложенности элементов в списке. Например, для [1, [2, [3, 4]], 5] функция должна вернуть 3.

Вариант 3

1. **Рекурсивный поиск всех ключей во вложенном словаре**
   Напишите функцию find_keys(d), которая принимает словарь, значения которого могут быть другими словарями, и возвращает список всех ключей, встречающихся на любом уровне вложенности.
2. **Рекурсивное извлечение значений по заданному ключу**
   Напишите функцию extract_values(d, key), которая принимает вложенный словарь d и искомый ключ key, возвращая список всех значений, ассоциированных с этим ключом на всех уровнях.
3. **Рекурсивное подсчитывание общего количества ключей**
   Напишите функцию count_keys(d), которая возвращает общее число ключей во вложенном словаре (учитывая все уровни вложенности).

Вариант 4

1. **Рекурсивное генерирование всех перестановок списка**
   Напишите функцию permute(lst), которая принимает список и возвращает список всех возможных перестановок элементов. Решите задачу с использованием рекурсии.
2. **Рекурсивное генерирование сочетаний заданной длины**
   Напишите функцию combinations(lst, k), которая возвращает список всех комбинаций из k элементов списка lst.
3. **Рекурсивное построение степенного множества (power set)**
   Напишите функцию power_set(lst), которая принимает список и возвращает список всех его подмножеств (мощность множества).

Вариант 5

1. **Рекурсивное возведение числа в степень**
   Напишите функцию power(x, n), которая принимает число x и целое неотрицательное число n и возвращает результат вычисления xnx^nxn с использованием рекурсии.

2. **Рекурсивное вычисление НОД (алгоритм Евклида)**
   Напишите функцию gcd(a, b), которая принимает два целых числа и рекурсивно вычисляет их наибольший общий делитель.

3. **Рекурсивное вычисление суммы цифр числа**
   Напишите функцию sum_digits(n), которая принимает неотрицательное целое число n и возвращает сумму его цифр, используя рекурсию (преобразование числа в строку не допускается).

Вариант 6

1. **Рекурсивная проверка палиндрома**
   Напишите функцию is_palindrome(s), которая проверяет, является ли строка s палиндромом (игнорируйте пробелы, знаки препинания и регистр) с использованием рекурсии.

2. **Рекурсивное удаление гласных из строки**
   Напишите функцию remove_vowels(s), которая принимает строку и рекурсивно возвращает новую строку без гласных букв (a, e, i, o, u, и их аналоги, без учёта регистра).

3. **Рекурсивный подсчёт вхождений символа в строке**
   Напишите функцию count_char(s, ch), которая принимает строку и символ, и с помощью рекурсии возвращает количество вхождений ch в s.

Вариант 7

Представьте бинарное дерево в виде вложенных кортежей:
(значение, левое_поддерево, правое_поддерево); пустое поддерево обозначается как None.

1. **Рекурсивное вычисление суммы узлов бинарного дерева**
   Напишите функцию tree_sum(tree), которая возвращает сумму всех значений в дереве.

2. **Рекурсивный поиск значения в бинарном дереве**
   Напишите функцию tree_contains(tree, target), которая возвращает True, если в дереве присутствует значение target, и False иначе.

3. **Рекурсивное вычисление высоты бинарного дерева**
   Напишите функцию tree_height(tree), которая возвращает высоту дерева (количество уровней, где высота пустого дерева равна 0).

Вариант 8

1. **Рекурсивное вычисление пересечения списка множеств**
   Напишите функцию recursive_intersection(sets), которая принимает список множеств и рекурсивно возвращает их пересечение.
2. **Рекурсивное вычисление объединения списка множеств**
   Напишите функцию recursive_union(sets), которая принимает список множеств и возвращает их объединение, используя рекурсию.
3. **Рекурсивное вычисление симметрической разности списка множеств**
   Напишите функцию recursive_sym_diff(sets), которая принимает список множеств и возвращает их симметрическую разность (поэлементное применение операции «исключающее ИЛИ» для множеств).

Вариант 9

1. **Рекурсивное подсчитывание элемента в произвольной вложенной структуре**
   Напишите функцию count_occurrences(structure, target), которая принимает произвольную вложенную структуру (которая может состоять из списков, кортежей и множеств) и возвращает количество вхождений элемента target.
2. **Рекурсивное нахождение максимального элемента во вложенной структуре**
   Напишите функцию find_max(structure), которая принимает вложенную структуру, содержащую числа, и возвращает максимальное число (структура может быть комбинацией списков, кортежей и множеств).
3. **Рекурсивное выравнивание (flatten) произвольной вложенной структуры**
   Напишите функцию flatten_structure(structure), которая преобразует вложенную структуру (состоящую из списков, кортежей, множеств) в плоский список всех элементов.

Вариант 10

1. **Рекурсивное преобразование вложенного словаря в плоский**
   Напишите функцию flatten_dict(d, parent_key='', sep='.'), которая принимает вложенный словарь и возвращает новый «плоский» словарь, где ключи – это объединённые через разделитель sep ключи всех уровней.
2. **Рекурсивное подсчитывание общего количества ключей в вложенном словаре**
   Напишите функцию total_keys(d), которая принимает вложенный словарь и возвращает общее число ключей на всех уровнях вложенности.
3. **Рекурсивное инвертирование вложенного словаря**
   Напишите функцию invert_nested_dict(d), которая на каждом уровне заменяет ключи и значения местами (при условии, что значения являются хэшируемыми) и возвращает новый словарь с инвертированными парами на всех уровнях.

Задание №3
Функции высших порядков

Вариант 1

1. **Функция «apply_twice»**
   Напишите функцию:

```
def apply_twice(f, x):
  """
  Принимает функцию f и значение x, возвращает результат f(f(x)).
  """
```

Проверьте работу функции на примере: передайте функцию, которая увеличивает число на 3, и значение 4.

2. **Пользовательский map**
   Напишите функцию:

```
def custom_map(func, iterable):
  """
  Принимает функцию и итерируемый объект.
  Возвращает список, в котором к каждому элементу применена функция func.
  """
```

Реализуйте функцию с помощью цикла (или рекурсии) и протестируйте на списке чисел.

3. **Пользовательский filter с опцией инверсии**
   Напишите функцию:

```
def custom_filter(predicate, iterable, invert=False):
  """
  Принимает предикат, итерируемый объект и булев флаг invert.
  Если invert=False, возвращает список элементов, для которых predicate(element) True;
  иначе – список элементов, для которых predicate(element) False.
  """
```

Проверьте работу на списке строк, фильтруя, например, по длине.

Вариант 2

1. **Функция-композитор (compose)**
   Напишите функцию:

```
def compose(f, g):
  """
```

Принимает две функции f и g, возвращает новую функцию h,

такую что h(x) = f(g(x)).

"""

Используйте её для создания функции, которая сначала берет абсолютное значение, а затем возводит в квадрат.

2. **Функция «apply_if»**
   Напишите функцию:

```
def apply_if(func, predicate):
  """
  Принимает функцию и предикат.
  Возвращает новую функцию, которая применяет func к аргументу только если predicate(arg) возвращает True,
  иначе возвращает аргумент без изменений.
  """
```

Проверьте её на преобразовании строки (например, применять преобразование только если строка не пустая).

3. **Функция «pipe»**
   Напишите функцию:

```
def pipe(functions):
  """
  Принимает список функций и возвращает новую функцию, которая последовательно применяет их к значению.
  """
```

Протестируйте, создав цепочку преобразований числа (например, умножение, прибавление, возведение в степень).

Вариант 3

1. **Функция «filter_map»**
   Напишите функцию:

```
def filter_map(predicate, func, iterable):
  """
  Сначала фильтрует iterable по предикату, затем применяет func к оставшимся элементам.
  Возвращает список результатов.
  """
```

Проверьте на списке чисел: отберите только четные и возведите их в квадрат.

2. **Реализация reduce для суммирования**
   С использованием functools.reduce напишите функцию:

```
def reduce_sum(numbers, start=0):
  """
  Возвращает сумму чисел из списка с начальным значением start.
  """
```

Проверьте работу на примере списка.

3. **Замыкание: Функция-множитель**
   Напишите функцию:

```
def make_multiplier(factor):
  """
  Принимает число factor и возвращает новую функцию,
  которая умножает свой аргумент на factor.
  """
```

Проверьте создание функции, которая умножает число на 5.

Вариант 4

1. **Декоратор «time_it»**
   Напишите декоратор:

```
def time_it(func):
  """
  Декорирует функцию, выводя время её выполнения.
  """
```

Примените его к функции, которая выполняет сложные вычисления (например, суммирование большого списка).

2. **Декоратор мемоизации (cache)**
   Напишите декоратор:

```
def memoize(func):
  """
  Декорирует функцию, кэшируя её результаты.
  """
```

Протестируйте его на рекурсивной функции вычисления факториала или чисел Фибоначчи.

3. **Декоратор «retry»**
   Напишите декоратор:

def retry(n):

"""

Декоратор, который повторяет вызов функции n раз, если возникает исключение.

"""

Примените его к функции, которая может случайно выбрасывать исключение.

Вариант 5

1. **Функция «do_n_times»**
   Напишите функцию:

def do_n_times(func, n):

"""

Возвращает новую функцию, которая при вызове выполнит func n раз и вернет последний результат.

"""

Проверьте на функции, которая, например, печатает значение.

2. **Частичное применение с functools.partial**

   С использованием functools.partial создайте функцию, которая суммирует список чисел с фиксированным начальным значением (например, 10). Опишите процесс частичного применения.

3. **Функция «apply_recursive»**
   Напишите функцию:

def apply_recursive(func, n):

"""

Возвращает новую функцию, которая является композицией функции func, примененной n раз.

То есть, если f = apply_recursive(func, 3), то f(x) = func(func(func(x))).

"""

Проверьте работу на функции, которая прибавляет 2.

Вариант 6

1. **Функция «conditional»**
   Напишите функцию:

def conditional(predicate, action):

"""

Принимает предикат и функцию-действие.

Возвращает новую функцию, которая применяет action к аргументу, если predicate(arg) истинно, иначе возвращает arg.

"""

Протестируйте на числах, например, увеличивая число только если оно положительное.

2. **Функция «zip_with»**
   Напишите функцию:

```
def zip_with(binary_func, list1, list2):
```

"""

Принимает бинарную функцию и два списка.

Возвращает новый список, где каждый элемент – результат применения binary_func к парам элементов.

"""

Проверьте работу для сложения элементов двух списков.

3. **Рекурсивная функция «unfold»**
   Напишите функцию:

```
def unfold(seed, predicate, func, next_func):
```

"""

Генерирует список, начиная с seed.

Пока predicate(seed) истинно, добавляет seed в список, затем seed обновляется функцией next_func.

Примените func к seed перед добавлением (если необходимо).

"""

Используйте unfold для генерации последовательности до определённого условия.

Вариант 7

1. **Функция каррирования (curry)**
   Напишите функцию:

```
def curry(func):
```

"""

Преобразует функцию, принимающую два аргумента, в каррированную функцию.

"""

Протестируйте на функции сложения двух чисел.

2. **Функция раскаррирования (uncurry)**
   Напишите функцию:

```
def uncurry(curried_func):
```

"""

Принимает каррированную функцию и возвращает функцию, принимающую два аргумента одновременно.

"""

Проверьте работу, раскаррировав ранее созданную функцию.

3. **Функция «compose_all»**
   Напишите функцию:

```
def compose_all(*functions):
  """
  Принимает произвольное число функций и возвращает их композицию,
  такую что результат вычисляется справа налево: f(g(h(x))).
  """
```

Проверьте, составив цепочку преобразований для числа.

Вариант 8

1. **Функция «map_filter_reduce»**
   Напишите функцию:

```
def map_filter_reduce(iterable, map_func, filter_pred, reduce_func, initial):
  """
  Комбинирует операции map, filter и reduce.
  Сначала применяет map_func к каждому элементу, затем фильтрует по filter_pred,
  и, наконец, сводит результаты с помощью reduce_func, начиная с initial.
  """
```

Протестируйте на списке чисел: например, сначала умножьте, затем оставьте только четные, а потом просуммируйте.

2. **Рекурсивная функция «chain»**
   Напишите функцию:

```
def chain(functions, value):
  """
  Рекурсивно применяет список функций к значению.
  """
```

Проверьте, создав цепочку преобразований для строки или числа.

3. **Функция «flatten_and_apply»**
   Напишите функцию:

def flatten_and_apply(nested_list, func):

"""

Принимает вложенный список (любая глубина) и функцию.

Выравнивает список и применяет func к каждому элементу, возвращая новый плоский список.

"""

Протестируйте на вложенном списке чисел.

Вариант 9

1. **Функция «apply_with_logging» (декоратор)**
   Напишите декоратор:

def apply_with_logging(func):

"""

Возвращает новую функцию, которая при каждом вызове выводит входные аргументы и результат.

"""

Примените к любой функции (например, факториалу) и проверьте логирование.

2. **Функция «filter_out_none»**
   Напишите функцию:

def filter_out_none(iterable):

"""

Возвращает список элементов из iterable, исключая все элементы, равные None.

Используйте функцию filter.

"""

Проверьте на списке с None-значениями.

3. **Рекурсивная функция «find_first»**
   Напишите функцию:

def find_first(predicate, iterable):

"""

Рекурсивно возвращает первый элемент из iterable, удовлетворяющий predicate,

или None, если такого элемента нет.

"""

Проверьте работу на списке чисел.

Вариант 10

1. **Функция «transform_dict»**
   Напишите функцию:

```
def transform_dict(d, transform_func):
  """
  Принимает словарь d и функцию transform_func, которая принимает пару (key, value) и возвращает (new_key, new_value).
  Возвращает новый словарь с преобразованными ключами и значениями.
  """
```

Протестируйте, например, преобразуя все ключи в верхний регистр и удваивая значения.

2. **Функция «merge_dicts» через reduce**
   Напишите функцию:

```
from functools import reduce


def merge_dicts(*dicts):
  """
  Принимает любое число словарей и объединяет их, суммируя значения для одинаковых ключей.
  """
```

Проверьте работу на примере нескольких словарей с числовыми значениями.

3. **Функция «apply_to_dict_values»**
   Напишите функцию:

```
def apply_to_dict_values(d, func):
  """
  Принимает словарь и функцию, возвращает новый словарь, где к каждому значению применена func.
  """
```

Протестируйте на словаре, где значения – числа (например, возведите каждое число в квадрат).

Задание №4
Декораторы

Вариант 1

1. **Декоратор измерения времени выполнения (time_it)**
   Напишите декоратор time_it, который измеряет время выполнения декорируемой

функции и выводит его вместе с именем функции. Протестируйте декоратор на функции, суммирующей большое количество чисел (например, используя цикл или генератор).

2. **Декоратор мемоизации (cache)**
   Реализуйте декоратор memoize, который кэширует результаты вызова функции для заданных аргументов. Примените его к рекурсивной функции вычисления n-го числа Фибоначчи и сравните скорость работы с некэшированной версией.

Вариант 2

1. **Декоратор повторных попыток (retry)**
   Создайте декоратор retry, принимающий параметр n (число попыток). Он должен повторно вызывать функцию до n раз, если во время вызова возникает исключение. Проверьте декоратор на функции, которая случайным образом генерирует исключение.

2. **Декоратор логирования вызова функции (logger)**
   Напишите декоратор, который перед вызовом функции выводит в консоль её имя и входные аргументы, а после — результат вызова. Проверьте работу на нескольких функциях с различными типами аргументов.

Вариант 3

1. **Параметризованный декоратор, модифицирующий результат**
   Создайте декоратор multiply_result(factor), который принимает параметр factor и возвращает декоратор, умножающий возвращаемое значение функции на этот множитель. Примените его к функции, возвращающей число.

2. **Декоратор с использованием functools.wraps**
   Напишите декоратор, который модифицирует поведение функции (например, выводит сообщение до и после вызова) и обязательно сохраняет метаданные (имя, docstring) исходной функции через functools.wraps. Проверьте, что после декорирования имя и документация функции не изменились.

Вариант 4

1. **Декоратор контроля типов аргументов**
   Реализуйте декоратор type_check, который проверяет типы переданных аргументов в соответствии с аннотациями функции. Если какой-либо аргумент имеет неверный тип, декоратор должен выбрасывать исключение TypeError. Протестируйте его на функции с несколькими аргументами разных типов.

2. **Декоратор кэширования с подсчетом вызовов**
   Напишите декоратор, который кэширует результаты функции и одновременно ведёт счет вызовов (например, добавляет атрибут call_count к функции). Примените его к рекурсивной функции (например, для вычисления факториала) и проверьте корректность работы.

Вариант 5

1. **Декоратор, разрешающий вызов функции только один раз (once)**
   Реализуйте декоратор once, который позволяет выполнить декорируемую функцию только при первом вызове, а последующие вызовы возвращают результат первого. Протестируйте его на функции, которая генерирует случайное число.
2. **Декоратор обработки исключений с возвратом значения по умолчанию**
   Напишите декоратор handle_error(default), который перехватывает все исключения, возникающие при выполнении функции, и возвращает значение default. Проверьте работу на функции, которая может делить на ноль.

Вариант 6

1. **Декоратор, логирующий вызовы в файл**
   Создайте декоратор, который записывает в файл (или имитирует запись, выводя на экран с пометкой «FILE LOG») имя функции, аргументы и результат её работы, а также время вызова. Протестируйте на нескольких функциях.
2. **Параметризованный декоратор, активирующий логирование по флагу**
   Реализуйте декоратор debug_log(enabled), который при enabled=True выводит лог сообщений (например, входные аргументы и результат), а при False — ничего не делает. Примените его к функции, которая обрабатывает список чисел.

Вариант 7

1. **Декоратор повторения до успешного результата**
   Напишите декоратор repeat_until_success(max_attempts), который повторяет вызов функции до тех пор, пока функция не вернет истинное значение или не исчерпает max_attempts. Протестируйте на функции, которая случайно генерирует истину/ложь.
2. **Декоратор трансформации результата**
   Создайте декоратор, который проверяет тип возвращаемого значения: если функция возвращает число, декоратор возводит его в квадрат, иначе возвращает значение без изменений. Протестируйте на функциях, возвращающих разные типы данных.

Вариант 8

1. **Декоратор, превращающий функцию в каррированную**
   Напишите декоратор curry, который преобразует функцию, принимающую два аргумента, в каррированную функцию (т.е. функцию, принимающую один аргумент и возвращающую функцию для второго аргумента). Проверьте работу на простой функции сложения.
2. **Декоратор, объединяющий несколько функций (compose_all)**
   Реализуйте декоратор или функцию высшего порядка compose_all, которая принимает несколько функций и возвращает их композицию (вычисляется справа налево). Протестируйте, создав цепочку преобразований для строки или числа.

Вариант 9

1. **Декоратор, повторяющий выполнение с задержкой**
   Создайте декоратор retry_with_delay(n, delay), который повторяет вызов функции до n раз с задержкой delay секунд между попытками, если функция выбрасывает исключение. Используйте модуль time.sleep для задержки. Протестируйте на функции, генерирующей исключения.

2. **Декоратор для объединения аргументов (zip_with)**
   Напишите декоратор, который изменяет поведение функции так, чтобы она принимала два списка, а затем применяла переданную бинарную функцию к парам элементов из этих списков. Например, функция суммирования должна возвращать список сумм. Используйте этот декоратор для создания функции, объединяющей два списка.

Вариант 10

1. **Декоратор проверки и преобразования возвращаемого значения**
   Напишите декоратор ensure_numeric, который проверяет, что возвращаемое значение функции является числом. Если нет — пытается преобразовать его к числу (например, с помощью float()), а при неудаче выбрасывает исключение. Протестируйте на функции, которая может возвращать строку или число.

2. **Декоратор, изменяющий поведение функции на основе количества вызовов**
   Реализуйте декоратор switch_behavior(threshold, func1, func2), который до достижения количества вызовов threshold использует функцию func1 для обработки входных данных, а после — переключается на func2. Протестируйте, создав две разные реализации одной логики (например, сложение чисел) и отслеживая переключение.

Задание №5
Основы функционального программирования

Вариант 1

1. **Обработка списка чисел с функциональными конструкциями**
   Задача:
   Напишите функцию process_numbers(numbers) с использованием map, filter и reduce, которая выполняет следующие шаги:

   - Фильтрует список, оставляя только простые числа (реализуйте вспомогательную функцию is_prime(n)).
   - Возводит оставшиеся числа в квадрат с помощью map.
   - Вычисляет произведение всех полученных чисел с помощью reduce.
   - Возвращает полученное произведение.

2. **Нахождение палиндромов в списке строк**
   Задача:
   Напишите функцию find_palindromes(words), которая принимает список строк и с использованием filter и lambda-функций возвращает список слов, являющихся

палиндромами (игнорируя регистр).
Подсказка: Строка считается палиндромом, если равна своей обратной (например, "Level" → level == level[::-1]).

Вариант 2

1. **Рекурсивное выравнивание (flatten) вложенного списка**
   Задача:
   Напишите рекурсивную функцию flatten(lst), используя функциональный стиль (с применением lambda-выражений, if-else внутри функции), которая принимает произвольный вложенный список и возвращает плоский список всех его элементов.

2. **Суммирование значений словаря с фильтрацией**
   Задача:
   Напишите функцию sum_filtered_values(d, threshold), которая принимает словарь с числовыми значениями и число threshold. С помощью функций высшего порядка (например, filter и reduce) функция должна:

   - Отфильтровать пары, где значение больше threshold.
   - Вернуть сумму отфильтрованных значений.

Вариант 3

1. **Композиция функций для обработки строк**
   Задача:
   Реализуйте функцию-композитор compose(f, g), которая принимает две функции и возвращает их композицию (функция, вычисляющая f(g(x))). Затем создайте функцию, которая:

   - Принимает строку, обрезает лишние пробелы, переводит её в нижний регистр и возвращает строку, в которой порядок слов перевёрнут (сохраняя порядок символов в каждом слове).
   - Используйте композицию ранее созданных функций для реализации.

2. **Функциональное вычисление факториала**
   Задача:
   Напишите функцию factorial(n), которая вычисляет факториал числа n с использованием рекурсии и функций высшего порядка (например, используя reduce для свёртки последовательности от 1 до n).
   Подсказка: Можно сначала создать список чисел с помощью range, а затем свернуть его через умножение.

Вариант 4

1. **Группировка чисел по чётности с использованием функций высшего порядка**
   Задача:
   Напишите функцию group_by_parity(numbers), которая принимает список чисел и с помощью filter (или dict comprehensions) делит их на две группы: четные и нечетные.

Функция должна возвращать словарь вида:
{ 'even': [...], 'odd': [...] }
где значения – списки чисел, отсортированные по возрастанию.

2. **Генерация последовательности Фибоначчи через рекурсию и map**
   Задача:
   Реализуйте функцию fibonacci(n), которая с помощью рекурсии и (при необходимости) функциональных конструкций возвращает список из первых n чисел Фибоначчи.
   Подсказка: Можно создать вспомогательную рекурсивную функцию, которая добавляет новое число к накопленному списку.

Вариант 5

1. **Обработка списка студентов с использованием map, filter и sorted**
   Задача:
   Напишите функцию top_students(students, threshold), которая принимает список словарей вида
   {'name': <строка>, 'score': <число>}
   и число threshold. С помощью функций высшего порядка:
   - Отфильтруйте студентов с оценкой не ниже threshold.
   - Верните список имён отобранных студентов, отсортированных по алфавиту.
2. **Реализация бинарного поиска в функциональном стиле**
   Задача:
   Напишите функцию binary_search(lst, target) с использованием рекурсии и функционального стиля (без использования циклов), которая ищет target в отсортированном списке lst и возвращает его индекс или -1, если элемент не найден.

Вариант 6

1. **Частотный анализ слов в тексте с использованием reduce**
   Задача:
   Напишите функцию word_count(text), которая принимает строку, приводит её к нижнему регистру, удаляет знаки препинания и с помощью функций высшего порядка (например, reduce) формирует словарь, где ключ – слово, а значение – число его вхождений.
2. **Генерация степенного множества (power set) с рекурсией**
   Задача:
   Реализуйте функцию power_set(s), которая принимает множество или список и с помощью рекурсии и функций высшего порядка возвращает список всех его подмножеств (power set).
   Подсказка: Подумайте о включении/исключении каждого элемента.

Вариант 7

1. **Функция-генератор линейного преобразования (каррирование)**
   Задача:

Напишите функцию make_linear_transform(a, b), которая возвращает новую функцию, принимающую число x и вычисляющую a * x + b. Используйте lambda-выражение для реализации.
Проверьте работу, создав, например, функцию, умножающую на 3 и прибавляющую 5.

2. **Вычисление скалярного произведения двух векторов с использованием map и reduce**
   Задача:
   Напишите функцию dot_product(v1, v2), которая принимает два списка чисел одинаковой длины и возвращает их скалярное произведение. Используйте функции map и reduce.

Вариант 8

1. **Реализация универсальной функции-композиции (compose_all)**

   Задача:

Напишите функцию compose_all(*funcs), которая принимает произвольное число функций и возвращает их композицию (вычисляется справа налево: результат последней функции передается в предпоследнюю и так далее). Протестируйте композицию на цепочке преобразований для числа или строки.

Стандартные входные и выходные данные:

Вход:

funcs = [lambda x: x + 1, lambda x: x * 2, lambda x: x ** 3]

value = 2

Выход: 17 (порядок вычислений: 23=8→8×2=16→16+1=1723=8→8×2=16→16+1=17).

Необычные/критические данные:

Вход: funcs = [], value = 5.

Выход: 5 (нет функций для применения — возвращается исходное значение).

Вход: funcs = [lambda s: s.upper()], value = "hello".

Выход: "HELLO" (одна функция в композиции).

2. **Рекурсивное вычисление НОД через функции высшего порядка**

   Задача:

Напишите функцию gcd(a, b) для вычисления наибольшего общего делителя двух чисел с использованием рекурсии и функционального стиля (например, используя lambda-выражение для описания рекурсивного шага).

Стандартные входные и выходные данные:

Вход: a = 48, b = 18.

Выход: 6 (НОД(48, 18) = 6).

Необычные/критические данные:

Вход: a = 0, b = 0.

Выход: 0 (математически не определен, но функция возвращает 0).

Вход: a = -4, b = 6.

Выход: 2 (НОД для отрицательных чисел берется по модулю).

Вход: a = 123456789, b = 987654321.

Выход: 3 (НОД вычисляется корректно, но возможна ошибка переполнения стека для очень больших чисел из-за рекурсии).

Вариант 9

1. **Реализация пайплайна обработки данных**

Задача:
Напишите функцию *pipeline(value, funcs)*, которая принимает начальное значение и список функций, последовательно применяет их к значению и возвращает итоговый результат.
Используйте функцию *reduce* для свёртки списка функций.
Протестируйте, передав цепочку преобразований для строки или числа.

Стандартные входные и выходные данные:

Вход: value = 3, funcs = [lambda x: x + 2, lambda x: x * 5]

Выход: (3 + 2) * 5 = 25

Необычные/критические данные:

Вход: value = 5, funcs = []

Выход: 5 (нет функций для применения — возвращается исходное значение)

Вход: value = "test", funcs = [str.upper, lambda s: s[::-1]]

Выход: "TSET" (преобразование в верхний регистр и разворот строки)

2. **Фильтрация простых чисел с использованием функционального подхода**

Задача:

Напишите функцию *filter_primes(numbers)*, которая принимает список чисел и с помощью функции *filter* и *lambda*-выражения (а также встроенной или собственной функции проверки простоты) возвращает список только простых чисел.

Стандартные входные и выходные данные:

Вход: [2, 3, 4, 5, 9, 11]

Выход: [2, 3, 5, 11]

Необычные/критические данные:

Вход: [-5, 0, 1, 2]

Выход: [2] (отрицательные числа, 0 и 1 не считаются простыми)

Вход: []

Выход: [] (пустой список на входе)

Вход: [997]

Выход: [997] (большое простое число)

Вариант 10

1. **Рекурсивное вычисление суммы цифр числа (без преобразования в строку)**

Задача:

Напишите функцию *sum_digits(n)*, которая принимает неотрицательное целое число и рекурсивно вычисляет сумму его цифр, не используя преобразование числа в строку. Используйте функциональный подход (например, через рекурсивное свёртывание).

Стандартные входные и выходные данные:

Вход: 123

Выход: 6 (поскольку 1 + 2 + 3 = 6)

Необычные/критические данные:

Вход: 0

Выход: 0 (поскольку число состоит из одной цифры, и это 0)

Вход: 99999999999999999999999999999999999999999999999999 (очень большое число)

Выход: 450 (поскольку сумма цифр 9, повторяющихся 50 раз, равна 450)

2. **Генерация всех комбинаций элементов списка длины k с функциональным стилем**

Задача:

Напишите функцию *combinations(lst, k)*, которая принимает список и число *k* и возвращает список всех сочетаний из *k* элементов, используя рекурсию и функции высшего порядка.
Подсказка: Подумайте о разбиении задачи на включение/исключение первого элемента.

Стандартные входные и выходные данные:

Вход: *lst = [1, 2, 3], k = 2*

Выход: *[[1, 2], [1, 3], [2, 3]]*

Необычные/критические данные:

Вход: lst = [], k = 0

Выход: [[]] (единственная комбинация длины 0 — пустой список)

Вход: lst = [1, 2, 3, 4, 5], k = 5

Выход: [[1, 2, 3, 4, 5]] (единственная комбинация длины 5 — весь список)

Вход: lst = [1, 2, 3], k = 4

Выход: [] (невозможно создать комбинацию длины 4 из списка длины 3)