Лабораторная работа 8

Функции высшего порядка и обработка последовательностей

В этой лабораторной функции будут не только выполнять вычисления: их можно сохранять в переменных, передавать другим функциям и получать как результат работы функции. Списки не используйте. Там, где это существенно, условие прямо называет нужный инструмент.

Раздел 1. Распаковка и функции для обработки последовательностей

Задача 1. Цена с учётом скидки и налога

Опишите функцию `final_cost(price, discount_percent, tax_percent)`, возвращающую итоговую цену: сначала к базовой цене применяется скидка, затем начисляется налог на уменьшенную цену. Три вещественных числа вводить в одну строку через пробел. Поместить ввод данных прямо в вызов функции `final_cost`. Используйте распаковку результата `map` с помощью оператора `*`.

In [ ]:
```
def final_cost(price, discount_percent, tax_percent):
    ...
```

Задача 2. Текст в коды символов

Опишите функцию `text_to_codes(text)`, которая возвращает строку с десятичными кодами символов исходного текста, разделёнными точкой с запятой. Например, для `A!` результат — `65;33`. Используйте `map`, `ord`, `str` и `join`; циклы не используйте.

In [ ]:
```
def text_to_codes(text):
    return ...
```

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

In [5]:
```
assert text_to_codes('A!') == '65;33'
assert text_to_codes('a') == str(ord('a'))
assert text_to_codes('H3ll0') == '72;51;108;108;48'
```

Задача 3. Коды символов в текст

Опишите функцию `codes_to_text(codes)`, выполняющую обратное преобразование. Строка `codes` непустая и содержит корректные десятичные коды символов, разделённые точкой с запятой. Используйте `map`, `int`, `chr` и `join`; циклы не используйте.

In [ ]:
```
def codes_to_text(codes):
    return ...
```

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

In [7]:
```
assert codes_to_text('65;33') == 'A!'
assert codes_to_text(str(ord('я'))) == 'я'
assert codes_to_text('72;51;108;108;48') == 'H3ll0'
```

Задача 4. Очистка имени пользователя

Опишите функцию `clean_login(text)`. Она должна перевести строку в нижний регистр и оставить только латинские буквы, цифры и символ `_`. Используйте `filter` и `lambda`; цикл не используйте. Считайте, что буквы во входной строке только латинские.

In [ ]:
```
def clean_login(text):
    ...
```

Задача 5. Проверка шестнадцатеричного токена

Опишите функцию `is_hex_token(token)`, возвращающую `True`, если строка имеет длину от 4 до 16, состоит только из шестнадцатеричных цифр `0` – `9`, `A` – `F`, `a` – `f`, содержит хотя бы одну цифру и хотя бы одну букву. Для проверок символов используйте `all`, `any` и `map`; циклы не используйте.

In [ ]:
```
def is_hex_token(token):
    ...
```

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

In [12]:
```
assert is_hex_token('A3F09C')
assert not is_hex_token('ABCDEF')
assert not is_hex_token('12G4')
```

Задача 6. Последовательные скидки

Опишите функцию `price_after_discounts(price, *discounts)`, применяющую к цене все скидки по очереди. Например, скидки 10% и 20% уменьшают цену `1000` до `720`, а не до `700`. Для последовательного применения скидок используйте `reduce` и `lambda`; циклы не используйте. Цена и хотя бы одна скидка вводятся **отдельно**.

In [ ]:
```python
from functools import reduce

def price_after_discounts(price, *discounts):
    return ...
```

Задача 7. Короткий числовой отпечаток текста

Опишите функцию `short_hash(text)`, вычисляющую число по правилу: для каждого следующего символа новое значение равно `(старое_значение * 31 + код_символа) % 1_000_000_007`. Начальное значение равно 0. Это упрощённый вариант полиномиального хеша; он не предназначен для защиты паролей. Используйте `reduce` и `lambda`; циклы не используйте.

In [ ]:
```python
#from functools import reduce

def short_hash(text):
    return ...
```

Раздел 2. Функция как аргумент

Задача 8. Таблица значений функции

Опишите функцию `print_table(start, end, action, name)`. Она должна вывести значения функции `action` для всех целых чисел от `start` до `end` включительно; `name` используется в пояснении вывода к каждой строке. В основной программе в качестве `name` передайте текст `Квадрат`, а в качестве `action` передайте анонимную функцию для вычисления квадрата числа. Затем отдельно выведите кубы чисел.

In [ ]:
```python
def print_table(start, end, action, name):
    ...

# Квадраты


# Кубы
```

Задача 9. Подсчёт по условию

Опишите функцию `count_by_condition(start, end, condition)`, возвращающую количество целых чисел от `start` до `end` включительно, для которых функция `condition` возвращает `True`. Проверьте функцию: подсчитайте числа, которые делятся на введённый делитель и не оканчиваются цифрой 0. Условие передайте через `lambda`.

In [ ]:
```python
def count_by_condition(start, end, condition):
    ...
```

Задача 10. Итерационный поиск квадратного корня

Опишите функцию `iterate(value, count, rule)`, которая `count` раз применяет функцию `rule` к значению `value` и возвращает результат. С её помощью приблизьте квадратный корень положительного числа `N` методом Ньютона. Начальное приближение — `N`, правило перехода: `(current + N / current) / 2`. Передайте правило как `lambda`-функцию. Число итераций вводится пользователем.

In [ ]:
```
from math import isclose

def iterate(value, count, rule):
    ...
```

Пример проверки. Здесь достаточно относительной погрешности: ожидаемое значение не близко к нулю. Допишите еще несколько тестов.

In [21]:
```
root_of_two = iterate(2.0, 5, ???) # вставьте подходящую лямбду на место ???
assert isclose(root_of_two, 2 ** 0.5, rel_tol=1e-9)
```

Раздел 3. Локальные функции, фабрики и конвейеры

Задача 11. Допустимые символы и скобки выражения

Опишите функцию `is_simple_expression(expression)`. Она возвращает `True`, если непустая строка содержит только цифры, пробелы, символы `+`, `-`, `*`, `/`, круглые скобки, а скобки правильно сбалансированы: их количество не становится отрицательным при просмотре слева направо и в конце равно нулю. Полную математическую корректность выражения проверять не нужно.

Внутри функции определите локальную функцию `is_allowed(symbol)` для проверки допустимого символа. Используйте эту локальную функцию вместе с `all` и `map`.

In [ ]:
```
def is_simple_expression(expression):
    ...
```

Задача 12. Самый частый значимый символ

Дана строка, содержащая хотя бы один непробельный символ. Найдите самый частый непробельный символ. При равной частоте выберите тот, который встречается раньше. Сначала отфильтруйте пробельные символы с помощью `filter`, затем примените `max` с параметром `key=text.count`. Здесь `text.count` передаётся встроенной функции как функция, задающая критерий выбора.

In [ ]:
```

```

Задача 13. Фабрика калибровки датчика

Опишите функцию `make_calibrator(raw_start, raw_end, value_start, value_end)`. Она возвращает функцию `calibrate(raw_value)`, которая линейно переводит показание датчика из диапазона `raw_start..raw_end` в физическую величину диапазона `value_start..value_end`. Например, датчик с показаниями от 0 до 10 может соответствовать температуре от -50 до 50.

Вложенная функция должна использовать параметры внешней функции. Значения `raw_start` и `raw_end` различны.

In [ ]:
```
def make_calibrator(raw_start, raw_end, value_start, value_end):
    ...
```

Пример проверки. Для вещественного результата используйте `isclose`. Допишите еще несколько тестов.

In [28]:
```
from math import isclose

sensor_to_temperature = make_calibrator(0, 10, -50, 50)
assert isclose(sensor_to_temperature(2.5), -25.0, rel_tol=1e-9)
assert isclose(sensor_to_temperature(10), 50.0, rel_tol=1e-9)
```

Задача 14. Фабрика конвейеров преобразований

Опишите функцию `make_pipeline(*steps)`, возвращающую функцию `pipeline(value)`. Полученная функция должна последовательно применять к `value` все функции из `steps` и возвращать итог. Вложенная функция здесь полезна: она хранит набор заранее настроенных шагов обработки и может многократно применяться к разным значениям.

Проверьте свою функцию на строке: удалите пробелы по краям, переведите текст в верхний регистр и добавьте в конце `!`.

In [ ]:
```
def make_pipeline(*steps):
    ...
```

Пример проверки. Допишите еще несколько тестов.

In [30]:
```
prepare = make_pipeline(str.strip, str.upper, lambda text: text + '!')
assert prepare('  готово ') == 'ГОТОВО!'
```

Задача 15. Конвейер для метки статьи

Используйте `make_pipeline` из предыдущей задачи, чтобы создать функцию `to_slug`. Она должна: удалить пробелы по краям, перевести строку в нижний регистр, заменить каждый пробел на дефис и удалить все символы, кроме букв, цифр и дефиса.

Для последнего шага используйте `filter` и `lambda`. Примените созданную функцию к введённому тексту.

In [ ]:
```
to_slug = make_pipeline(...)
```