

NovaData LMS: создание programming-заданий

Целевая аудитория: авторы курсов и методисты

Документ описывает создание заданий типа `programming` в LMS: какие поля заполнять, как устроены тест-кейсы и чем отличаются режимы `run_tests` и `stdin_stdout`.

Связанные документы:

- `README_DOCUMENTATION.md` - индекс комплекта документации и рекомендуемый порядок чтения.
- `PROJECT_DOCUMENTATION.md` - общая техническая документация проекта.
- `API_CONTRACTS.md` - API-контракты LMS/checkers.
- `DEPLOYMENT_GUIDE.md` - инструкция по развертыванию.
- `ENVIRONMENT_VARIABLES.md` - схема переменных окружения.
- `OPERATIONS_RUNBOOK.md` - регламент эксплуатации.
- `SQL_TASK_AUTHOR_GUIDE.md` - руководство по созданию SQL-заданий.

1. Назначение programming-заданий

Programming-задание используется, когда студент должен отправить программный код, а LMS автоматически проверяет решение через `code_checker`.

Поддерживаются два основных формата задач:

- `function mode` - студент реализует функцию или метод;
- `stdin/stdout mode` - студент пишет программу, которая читает `stdin` и печатает `stdout`.

Режим определяется автоматически по формату первого тест-кейса.

2. Где создается задание

В интерфейсе автора:

1. Открыть курс.
2. Перейти к структуре курса.

3. Открыть нужный модуль и урок.
4. Добавить шаг типа `programming`.
5. Открыть редактирование `programming`-шага.
6. Заполнить форму и сохранить.

После создания шага LMS создает связанную запись `StepContentProgramming`.

3. Основные поля формы

Поле	Обязательность	Описание
Условие задачи	Да	Текст задачи для студента. Поддерживает форматирование через CKEditor
Язык программирования	Да	Язык решения
Окружение	Да	Runtime/sandbox profile
Версия языка	Нет	Например 3.14 , 21 , 2.12
Имя функции/метода	Для function mode	Имя функции, которую должен реализовать студент
Тестовые случаи	Да	JSON-массив тестов
Стартовый код	Нет	Заготовка, которую увидит студент
Решение преподавателя	Нет	Эталонное решение для автора/методиста
Таймаут выполнения	Да	Ограничение времени, секунды
Лимит памяти	Да	Ограничение памяти, МБ
Баллы	Да	Баллы за задание
Максимум попыток	Да	Сколько раз студент может отправить решение
Подсказка	Нет	Показывается студенту
Объяснение решения	Нет	Пояснение правильного подхода

4. Поддерживаемые языки и окружения

В модели LMS указаны языки:

- Python;
- Java;
- Scala;
- C++;
- C#;
- Go;
- JavaScript;
- TypeScript;
- R;
- SQL.

Фактическая поддержка зависит от `sandbox images` в `code_checker` . На момент документации в `code_checker` реально описаны профили:

- Python base;
- Python PySpark;
- Python sklearn;
- Python TensorFlow;
- Python PyTorch;
- Java base;
- Java Spark;
- Scala Spark;
- C++;
- C#;
- Go;
- Haskell;
- R;
- Rust.

Перед публикацией задачи нужно убедиться, что выбранная пара `language + environment + version` есть в `code_checker` .

5. Выбор режима проверки

LMS определяет режим проверки по первому тест-кейсу:

Формат первого input	Режим	Что проверяется
<code>{"args": [...]}</code>	<code>run_tests</code>	Функция или метод
строка	<code>stdin_stdout</code>	Вся программа через <code>stdin/stdout</code>

Все тест-кейсы внутри одной задачи должны использовать один формат. Нельзя смешивать `function mode` и `stdin/stdout mode` в одной задаче.

6. Function mode

Используйте `function mode`, если студент должен реализовать конкретную функцию или метод.

Требования:

- заполнить имя функции/метода ;
- в тестах использовать `input.args` ;
- `args` должен быть массивом;
- каждый тест должен содержать `input` и `output` .

Пример тестов:

```
[
  {
    "input": {
      "args": [2, 3]
    },
    "output": 5
  },
  {
    "input": {
      "args": [0, 0]
    },
    "output": 0
  }
]
```

Пример стартового кода Python:

```
def sum_numbers(a, b):  
    # Напишите решение здесь  
    pass
```

Пример решения:

```
def sum_numbers(a, b):  
    return a + b
```

7. Stdin/stdout mode

Используйте stdin/stdout mode, если студент должен написать полноценную программу.

Требования:

- поле `Имя функции/метода` можно оставить пустым;
- `input` должен быть строкой;
- `output` должен быть строкой;
- переносы строк нужно явно указывать как `\n` ;
- `stdout` сравнивается с ожидаемым выводом.

Пример тестов:

```
[  
  {  
    "input": "2 3\n",  
    "output": "5\n"  
  },  
  {  
    "input": "-1 10\n",  
    "output": "9\n"  
  }  
]
```

Пример стартового кода Python:

```
a, b = map(int, input().split())  
# выведите ответ
```

Пример решения:

```
a, b = map(int, input().split())
print(a + b)
```

8. PySpark/DataFrame задачи

Для DataFrame-задач используется function mode, а DataFrame передается как специальный объект в `args`.

Пример:

```
[
  {
    "input": {
      "args": [
        {
          "__type__": "dataframe",
          "data": [
            [1, "Alice", 100],
            [2, "Bob", 200]
          ],
          "columns": ["id", "name", "score"]
        }
      ]
    },
    "output": []
  }
]
```

Требования:

- объект DataFrame должен содержать `__type__ = "dataframe"` ;
- поле `data` должно быть массивом строк данных;
- поле `columns` должно быть массивом имен колонок.

9. Required и forbidden constructs

Форма содержит скрытые JSON-поля для:

- `required_constructs` ;
- `forbidden_constructs` ;
- `additional_files` .

Эти поля используются для дополнительных требований к коду. В текущей документации основной сценарий для автора - тест-кейсы. Если в интерфейсе доступны элементы управления конструкциями, их нужно заполнять как JSON.

Пример `required constructs`:

```
{
  "all": ["For"],
  "any": ["If", "While"],
  "not": ["import os"]
}
```

10. Настройки выполнения

Рекомендуемые значения:

Тип задачи	Timeout	Memory
Простая Python/C++/Go задача	15-30 сек	256-512 МБ
Java/C# задача	20-60 сек	512 МБ
PySpark/Spark задача	60-180 сек	1-2 ГБ
ML-задача	60-180 сек	1-2 ГБ

Фактические ограничения также зависят от профиля `sandbox` в `code_checker/app/config.py` .

11. Правила составления тестов

Рекомендуется:

- добавить минимум 3-5 тестов;
- включить базовый положительный кейс;
- добавить граничные значения;
- добавить пустые/нулевые значения, если они допустимы;

- проверить отрицательные числа, большие числа, строки с пробелами;
- для `stdin/stdout` явно контролировать переносы строк;
- не смешивать разные форматы тестов.

Не рекомендуется:

- делать единственный тест;
- использовать случайные данные без фиксированного `expected output`;
- требовать ввод из файлов, если runner этого не поддерживает;
- использовать сеть в решении студента;
- полагаться на порядок элементов, если задача допускает несколько правильных порядков, без явного описания этого в проверке.

12. Чек-лист перед публикацией

Перед публикацией programming-задания проверьте:

- условие задачи понятно студенту;
- выбран язык и окружение, которые реально поддерживает `code_checker` ;
- версия языка соответствует `sandbox image`;
- тесты являются валидным JSON;
- все тесты используют один режим;
- для `function mode` заполнено `method_name` ;
- для `stdin/stdout mode` `input` и `output` являются строками;
- стартовый код не содержит готового решения;
- эталонное решение проходит тесты;
- выставлены баллы и число попыток;
- `timeout` и `memory limit` соответствуют сложности задачи;
- подсказка и объяснение не раскрывают решение раньше времени.

13. Типовые ошибки автора

Ошибка	Последствие	Как исправить
Смешаны <code>input.args</code> и строковый <code>input</code>	Форма не сохранится	Использовать один формат тестов
Не заполнен <code>method_name</code> для <code>function mode</code>	Проверка не сможет вызвать функцию	Указать имя функции

Ошибка	Последствие	Как исправить
output в stdin/stdout не строка	Форма не сохранится	Записать expected output в кавычках
Нет переносов строк \n	stdout может не совпасть	Явно указать \n
Выбран неподдерживаемый language/environment	Проверка упадет	Сверить с профилями code_checker
Слишком маленький timeout	Корректные решения падают по времени	Увеличить timeout
Слишком мало тестов	Неполная проверка	Добавить граничные кейсы

14. Пример полной задачи

Задача: написать функцию `is_even`, которая возвращает `true`, если число четное.

Поля:

Язык: `python`

Окружение: `base`

Версия: `3.14`

Имя функции: `is_even`

Баллы: `5`

Максимум попыток: `3`

Timeout: `15`

Memory: `256`

Тесты:

```
[
  {
    "input": {
      "args": [2]
    },
    "output": true
  },
  {
    "input": {
      "args": [3]
    },
    "output": false
  },
  {
    "input": {
      "args": [0]
    },
    "output": true
  }
]
```

Стартовый код:

```
def is_even(n):
    pass
```

Решение:

```
def is_even(n):
    return n % 2 == 0
```