«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники (ФПИиКТ)

Лабораторная работа

Исследование протоколов, форматов обмена информацией и языков разметки документов

Вариант 32

Выполнил

Григорьев Даниил Александрович

Группа 3116

Принял В.В. Пономарев

Санкт-Петербург 2024

Содержание

[Задание 3](#_Toc177734059)

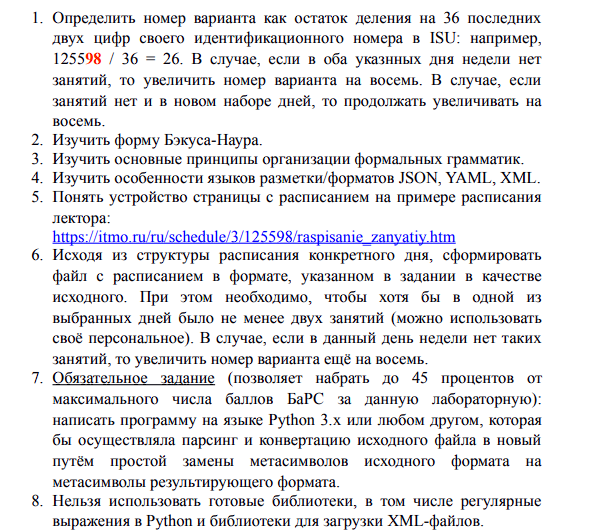
[Основные этапы вычисления 4](#_Toc177734060)

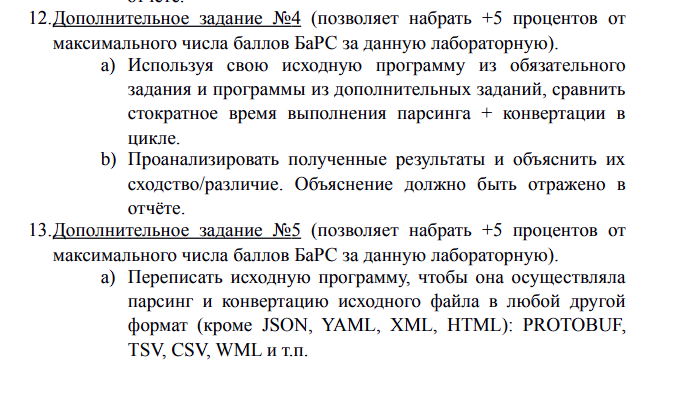
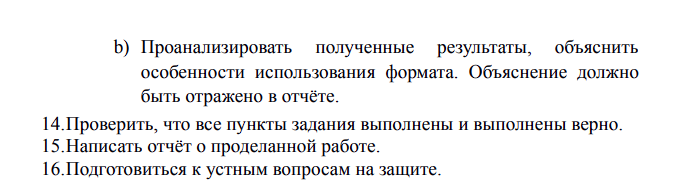
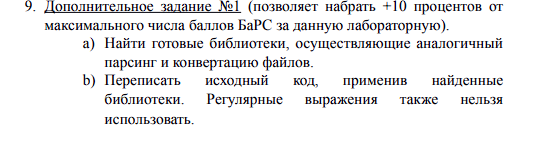
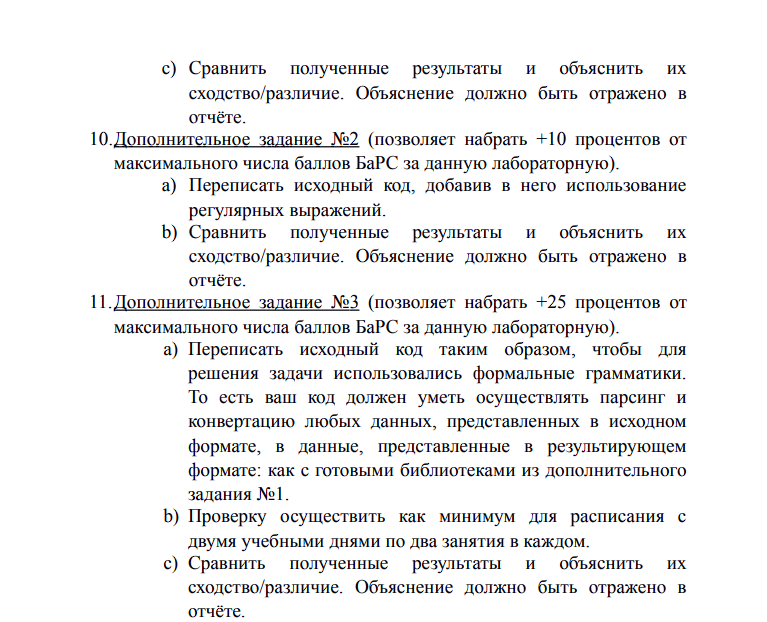
[Код программы для дополнительного задания №1: 7](#_Toc177734061)

[Заключение 8](#_Toc177734062)

[Список использованных источников 9](#_Toc177734063)

Задание



Основные этапы вычисления

Основное задание:

def parse\_input\_file():

file = open("Informatics/fourth\_lab/main\_task/input.json", "r", encoding="utf-8")

yaml\_data = file.read()

*# Заменяем ключи на YAML-формат (отступы и двоеточия)*

yaml\_data = yaml\_data.replace("{", "").replace("}", "")

yaml\_data = yaml\_data.replace("[", "").replace("]", "")

yaml\_data = yaml\_data.replace(",", "")

yaml\_data = yaml\_data.replace(":", ":\n ")

yaml\_data = yaml\_data.replace("\"", "") *# Убираем кавычки*

yaml\_data = yaml\_data.replace("\n\n", "\n") *# Убираем лишние пустые строки*

lines = yaml\_data.split("\n")

*# print(lines)*

yaml\_lines = []

for line in lines:

if line.strip() == "":

continue

indent\_level = line.find(line.strip()) *# находим уровень отступа*

yaml\_lines.append(" " \* indent\_level + line.strip())

final\_var = ""

reserved\_words = ["date", "first\_day", "second\_day", "subject", "lesson", "lecture", "time", "room", "place", "schedule"]

print(yaml\_lines)

for line in yaml\_lines:

exist\_flag = False

for i in reserved\_words:

if i in line:

final\_var += "\n" + line

exist\_flag = True

break

if exist\_flag == False:

final\_var += " " + line.strip()

output\_file = open("Informatics/fourth\_lab/main\_task/output.yaml", "w", encoding="utf-8")

output\_file.write(final\_var)

output\_file.close()

def main():

parse\_input\_file()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Пример работы:

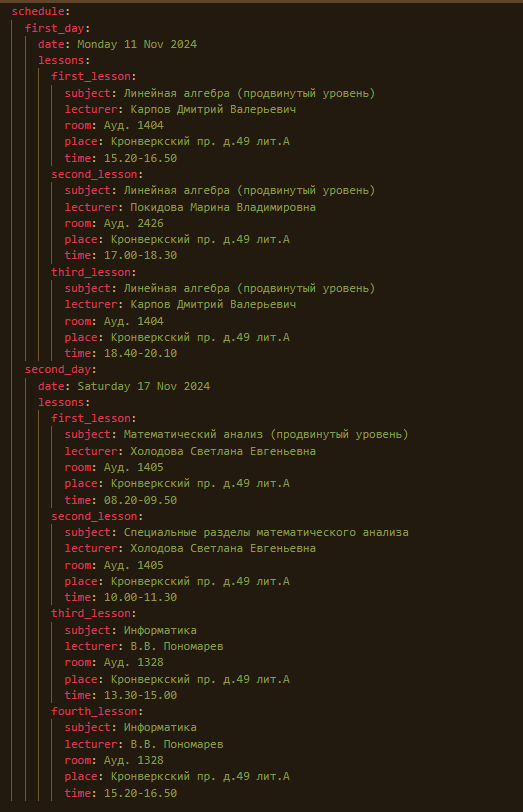
Входной файл input.json:

Первая половина файла



Вторая половина файла

Выходной файл после работы алгоритма:



Дополнительное задание №1

Код алгоритма:

import json

import yaml *# используем pyyaml*

input\_file = open("Informatics/fourth\_lab/first\_option\_task/input.json", "r", encoding="utf-8").read()

json\_data = json.loads(input\_file) *# парсим json из строки*

yaml\_data = yaml.dump(json\_data, default\_flow\_style=False, allow\_unicode="True") *# конвертируем json в yaml*

output\_file = open("Informatics/fourth\_lab/first\_option\_task/output.yaml", "w", encoding="utf-8")

output\_file.write(yaml\_data)

output\_file.close()

Входные файлы для обязательного задания и дополнительного задания №1 ничем не отличаются. В выходных файлах глобальных отличий нет, единственное отличие – в выходном коде после выполнения алгоритма дополнительного задания №1 “ключи” сортируются по названию в алфавитном порядке.

Дополнительное задание №2

Код алгоритма:

import re

def parse\_input\_file():

file = open("Informatics/fourth\_lab/second\_option\_task/input.json", "r", encoding="utf-8")

yaml\_data = file.read()

yaml\_data = re.sub(r'["}],', "", yaml\_data)

yaml\_data = re.sub(r'[{}"]', "", yaml\_data)

lines = yaml\_data.split("\n")

*# print(lines)*

yaml\_lines = []

for line in lines:

if line.strip() == "":

continue

indent\_level = line.find(line.strip()) *# находим уровень отступа*

yaml\_lines.append(" " \* (indent\_level-1) + line.strip())

output\_file = open("Informatics/fourth\_lab/second\_option\_task/output.yaml", "w", encoding="utf-8")

output\_file.write("\n".join(yaml\_lines))

output\_file.close()

def main():

parse\_input\_file()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Входные файлы для обязательного задания и дополнительного задания №2 ничем не отличаются. В выходных файлах обязательного и дополнительного задания №2 отличий нет.

Дополнительное задание №3

Код алгоритма:

class JSONParser: *# парсер json формата из строки: строка -> json*

def \_\_init\_\_(self, text):

self.text = text

self.index = 0

def parse(self): *#старт парсинга: пропускаем пробелы -> обрабатываем не пробельный символ -> пропускаем пробелы -> возвращаем json*

self.skip\_whitespace()

value = self.parse\_value()

self.skip\_whitespace()

return value

def parse\_value(self): *# обрабатываем не пробельный символ (если пробел, то пропускаем и работаем с не пробельным символом)*

self.skip\_whitespace()

char = self.current\_char()

if char == '"':

return self.parse\_string()

elif char == '{':

return self.parse\_object()

elif char == '[':

return self.parse\_array()

elif char.isdigit() or char == '-':

return self.parse\_number()

elif self.text[self.index:self.index+4].lower() == "true":

self.index += 4

return True

elif self.text[self.index:self.index+5].lower() == "false":

self.index += 5

return False

elif self.text[self.index:self.index+4].lower() == "none":

self.index += 4

return None

else:

print("Че ты сюда вписал вообще, кроме цифр/чисел/строк/словарей/списков/true/false/none ничего не принимается")

def parse\_object(self): *# работаем с объектом (словарь)*

obj = {}

self.index += 1 *# Skip '{'*

self.skip\_whitespace()

while self.current\_char() != '}':

key = self.parse\_string()

self.skip\_whitespace()

if self.current\_char() != ':':

print("Переделывай свой json: ожидался символ : после ключа словаря")

self.index += 1 *# Skip ':'*

self.skip\_whitespace()

value = self.parse\_value()

obj[key] = value

self.skip\_whitespace()

if self.current\_char() == ',':

self.index += 1 *# Skip ','*

self.skip\_whitespace()

elif self.current\_char() != '}':

print("Переделывай свой json: ожидались символы , или }")

self.index += 1 *# Skip '}'*

return obj

def parse\_array(self): *# работаем со списком*

arr = []

self.index += 1 *# Skip '['*

self.skip\_whitespace()

while self.current\_char() != ']':

value = self.parse\_value()

arr.append(value)

self.skip\_whitespace()

if self.current\_char() == ',':

self.index += 1 *# Skip ','*

self.skip\_whitespace()

elif self.current\_char() != ']':

print("Переделывай свой json, он выполнен не по формату: нужны , или ]")

self.index += 1 *# Скипаем ']'*

return arr

def parse\_string(self): *# работаем со строкой*

self.index += 1 *# Скипаем открытие строки '"'*

start = self.index

while self.current\_char() != '"':

if self.current\_char() == '\\':

self.index += 2 *# Скипаем символ \*

else:

self.index += 1

result = self.text[start:self.index]

self.index += 1 *# Скипаем закрытие строки '"'*

return result

def parse\_number(self): *# работаем с числами*

start = self.index

if self.current\_char() == '-':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

if self.current\_char() == '.':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

if self.current\_char() in 'eE':

self.index += 1

if self.current\_char() in '+-':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

return float(self.text[start:self.index])

def current\_char(self): *# текущий символ*

if self.index < len(self.text):

return self.text[self.index]

return ''

def skip\_whitespace(self): *# скипаем пробелы (прибавляем к рассматриваемому индексу единицу)*

while self.index < len(self.text) and self.text[self.index].isspace():

self.index += 1

def json\_to\_yaml(data, indent = 0, tire = 0): *# конвертер json в yaml*

yaml = ""

if isinstance(data, dict):

for key, value in data.items():

if tire != 1 or key != list(data.keys())[0]:

if isinstance(value, dict):

yaml += "\n"

elif isinstance(value, list):

yaml += "\n"

elif isinstance(value, str):

yaml += "\n"

else:

yaml += "\n"

yaml += indent \* " " + key + ":" + json\_to\_yaml(value, indent+2,0)

elif key == list(data.keys())[0]and tire == 1:

yaml += key + ":" + json\_to\_yaml(value, indent+2,0)

elif isinstance(data, list):

for value in data:

yaml += "\n" + (indent-2) \* " " + "- " + json\_to\_yaml(value, indent, 1)

elif isinstance(data, str):

yaml += (f' "{data.strip()}"')

else:

yaml += " " + str(data).strip()

return yaml

*# Чтение JSON-файла, разбор и преобразование в YAML*

def main():

with open("Informatics/fourth\_lab/third\_option\_task/example0.json", "r", encoding="UTF-8") as file:

json\_text = file.read()

parser = JSONParser(json\_text)

parsed\_data = parser.parse()

yaml\_output = json\_to\_yaml(parsed\_data)

yaml\_output = yaml\_output.lstrip("\n")

with open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/third\_option\_task/output.yaml", "w", encoding = "UTF-8") as file:

file.write(yaml\_output)

print("JSON успешно конвертирован в YAML и сохранён в 'output.yaml'.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Дополнительное задание №4

Код алгоритма:

import time

import json

import yaml

import os

import re

def main\_task():

input\_file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/input.json", "r", encoding="utf-8")

output\_file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/output.yaml", "w", encoding="utf-8")

yaml\_data = input\_file.read()

*# Заменяем метасимволы для приведения json к yaml формату*

yaml\_data = yaml\_data.replace('",', "").replace('},', "").replace('"', "")

yaml\_data = yaml\_data.replace("{", "").replace("}", "")

yaml\_data = yaml\_data.replace("[", "").replace("]", "")

lines = yaml\_data.split("\n") *# список строк, с которым будем работать*

yaml\_lines = [] *# результирующий список строк*

first = -1

for line in lines:

if line.strip() == "":

continue

elif first == -1:

first = line.find(line.strip())

indent\_level = line.find(line.strip()) *# находим уровень отступа путем нахождения индекса первой встречи ключа в строке line*

yaml\_lines.append(" " \* (indent\_level-first) + line.strip().strip("\t"))

output\_file.write("\n".join(yaml\_lines))

output\_file.close()

def first\_option():

input\_file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/input.json", "r", encoding="utf-8")

output\_file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/output.yaml", "w", encoding="utf-8")

input\_file = input\_file.read()

json\_data = json.loads(input\_file) *# парсим json из строки*

yaml\_data = yaml.dump(json\_data, default\_flow\_style=False, allow\_unicode="True") *# конвертируем json в yaml*

output\_file.write(yaml\_data)

output\_file.close()

def second\_option():

file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/input.json", "r", encoding="utf-8")

yaml\_data = file.read()

yaml\_data = re.sub(r'["}],', "", yaml\_data)

yaml\_data = re.sub(r'[{}"]', "", yaml\_data)

lines = yaml\_data.split("\n")

yaml\_lines = []

for line in lines:

if line.strip() == "":

continue

indent\_level = line.find(line.strip()) *# находим уровень отступа*

yaml\_lines.append(" " \* (indent\_level-1) + line.strip())

output\_file = open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/output.yaml", "w", encoding="utf-8")

output\_file.write("\n".join(yaml\_lines))

output\_file.close()

*#######*

class JSONParser: *# парсер json формата из строки: строка -> json*

def \_\_init\_\_(self, text):

self.text = text

self.index = 0

def parse(self): *#старт парсинга: пропускаем пробелы -> обрабатываем не пробельный символ -> пропускаем пробелы -> возвращаем json*

self.skip\_whitespace()

value = self.parse\_value()

self.skip\_whitespace()

return value

def parse\_value(self): *# обрабатываем не пробельный символ (если пробел, то пропускаем и работаем с не пробельным символом)*

self.skip\_whitespace()

char = self.current\_char()

if char == '"':

return self.parse\_string()

elif char == '{':

return self.parse\_object()

elif char == '[':

return self.parse\_array()

elif char.isdigit() or char == '-':

return self.parse\_number()

elif self.text[self.index:self.index+4].lower() == "true":

self.index += 4

return True

elif self.text[self.index:self.index+5].lower() == "false":

self.index += 5

return False

elif self.text[self.index:self.index+4].lower() == "none":

self.index += 4

return None

else:

print("Че ты сюда вписал вообще, кроме цифр/чисел/строк/словарей/списков/true/false/none ничего не принимается")

def parse\_object(self): *# работаем с объектом (словарь)*

obj = {}

self.index += 1 *# Skip '{'*

self.skip\_whitespace()

while self.current\_char() != '}':

key = self.parse\_string()

self.skip\_whitespace()

if self.current\_char() != ':':

print("Переделывай свой json: ожидался символ : после ключа словаря")

self.index += 1 *# Skip ':'*

self.skip\_whitespace()

value = self.parse\_value()

obj[key] = value

self.skip\_whitespace()

if self.current\_char() == ',':

self.index += 1 *# Skip ','*

self.skip\_whitespace()

elif self.current\_char() != '}':

print("Переделывай свой json: ожидались символы , или }")

self.index += 1 *# Skip '}'*

return obj

def parse\_array(self): *# работаем со списком*

arr = []

self.index += 1 *# Skip '['*

self.skip\_whitespace()

while self.current\_char() != ']':

value = self.parse\_value()

arr.append(value)

self.skip\_whitespace()

if self.current\_char() == ',':

self.index += 1 *# Skip ','*

self.skip\_whitespace()

elif self.current\_char() != ']':

print("Переделывай свой json, он выполнен не по формату: нужны , или ]")

self.index += 1 *# Скипаем ']'*

return arr

def parse\_string(self): *# работаем со строкой*

self.index += 1 *# Скипаем открытие строки '"'*

start = self.index

while self.current\_char() != '"':

if self.current\_char() == '\\':

self.index += 2 *# Скипаем символ \*

else:

self.index += 1

result = self.text[start:self.index]

self.index += 1 *# Скипаем закрытие строки '"'*

return result

def parse\_number(self): *# работаем с числами*

start = self.index

if self.current\_char() == '-':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

if self.current\_char() == '.':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

if self.current\_char() in 'eE':

self.index += 1

if self.current\_char() in '+-':

self.index += 1

while self.current\_char().isdigit():

self.index += 1

return float(self.text[start:self.index])

def current\_char(self): *# текущий символ*

if self.index < len(self.text):

return self.text[self.index]

return ''

def skip\_whitespace(self): *# скипаем пробелы (прибавляем к рассматриваемому индексу единицу)*

while self.index < len(self.text) and self.text[self.index].isspace():

self.index += 1

def json\_to\_yaml(data, indent = 0, tire = 0): *# конвертер json в yaml*

yaml = ""

if isinstance(data, dict):

for key, value in data.items():

if tire != 1 or key != list(data.keys())[0]:

if isinstance(value, dict):

yaml += "\n"

elif isinstance(value, list):

yaml += "\n"

elif isinstance(value, str):

yaml += "\n"

else:

yaml += "\n"

yaml += indent \* " " + key + ":" + json\_to\_yaml(value, indent+2,0)

elif key == list(data.keys())[0]and tire == 1:

yaml += key + ":" + json\_to\_yaml(value, indent+2,0)

elif isinstance(data, list):

for value in data:

yaml += "\n" + (indent-2) \* " " + "- " + json\_to\_yaml(value, indent, 1)

elif isinstance(data, str):

yaml += (f' "{data.strip()}"')

else:

yaml += " " + str(data).strip()

return yaml

def third\_option\_task():

with open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/input.json", "r", encoding="UTF-8") as file:

json\_text = file.read()

parser = JSONParser(json\_text)

parsed\_data = parser.parse()

yaml\_output = json\_to\_yaml(parsed\_data)

yaml\_output = yaml\_output.lstrip("\n")

with open("C:/Users/danie/OneDrive/Desktop/main folder/ITMO\_files/Informatics/fourth\_lab/fourth\_option\_task/output.yaml", "w", encoding = "UTF-8") as file:

file.write(yaml\_output)

*####*

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = time.time()

for i in range(100):

main\_task()

end\_time = time.time()

print(f"Время выполнения файла основного задания:{end\_time-start\_time:.6f} секунд")

start\_time = time.time()

for i in range(100):

first\_option()

end\_time = time.time()

print(f"Время выполнения файла доп задачи 1: {end\_time-start\_time:.6f} секунд")

start\_time = time.time()

for i in range(100):

second\_option()

end\_time = time.time()

print(f"Время выполнения доп задачи 2: {end\_time-start\_time:.6f} секунд")

start\_time = time.time()

for i in range(100):

third\_option\_task()

end\_time = time.time()

print(f"Время выполнения доп задачи 3: {end\_time-start\_time:.6f} секунд")

Заключение

Я научился парсить json и конвертировать его в yaml

Вопросы

Форма Бэкуса-Наура – это формальная система описания синтаксиса, в которой одни синтаксические группы определяются через другие.

<определяемый символ> ::= <посл1> | <посл2>

Принципы организации формальных грамматик

Такие единицы есть:

Терминал – неизменяемый объект, существующий в словах языка

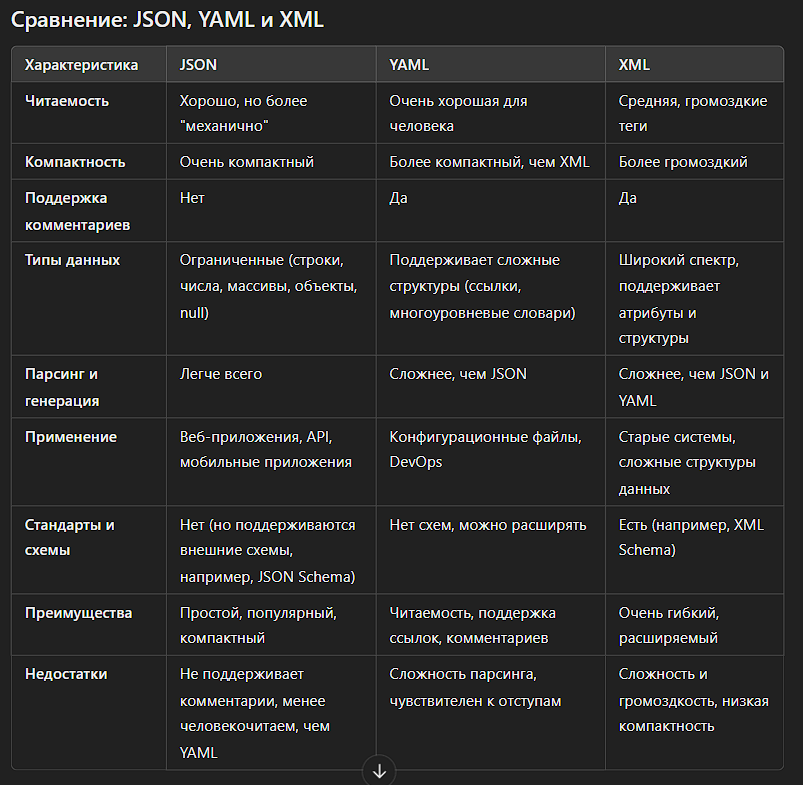
Нетерминал – объект, обозначающий сущность языка (формула, команда …)

Начальный объект

Правила (продукции):

<S> -> <…> -> <…> …

Формальная грамматика – способ описания формального языка



Markup – общее название языков разметки (HTML, latex, XML)

Markdown – упрощенный язык разметки для быстрого форматирования текста в HTML

Protobuf – протокол сериализации данных, предложенный Google, как альтернатива xml. Работает быстрее, поскольку осуществляется передача бинарных данных, оптимизированных под минимальный размер сообщения[

CSV – comma separated values – отделяются запятыми

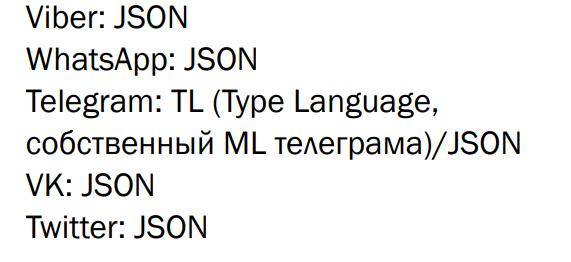
TSV – tab separated values – отделяются табуляцией

Разнообразие требований к данным, оптимизация, гибкость, упрощение работы с данными

< - начало тега, > - конец тега

Сериализация – процесс перевода данных в битовую последовательность

# - коммент



SVG — это формат и стандарт для описания двухмерной векторной графики с помощью XML. Scalable vector graphics – масштабируемая векторная графика

<a href="URL">Текст ссылки</a>

Объекты, массивы, строки, числа, null, true/false

Список использованных источников

1. [Балакшин, П. В. Информатика / П. В. Балакшин, В. В. Соснин, Е. А. Машина. – СПБ : Университет ИТМО, 2020. – 143 с.]
2. [ГОСТ 7.32-2017 «Отчет о научно-исследовательской работе. Структура и правила оформления» : дата введения 2017 25 09. – Москва : Стандартинформ, 2017. – 32 с.]
3. ГОСТ 7.1 — 2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления» : дата введения 01.07.2004. – Москва : ИПК Издательство стандартов, 2003. – 57 с.