

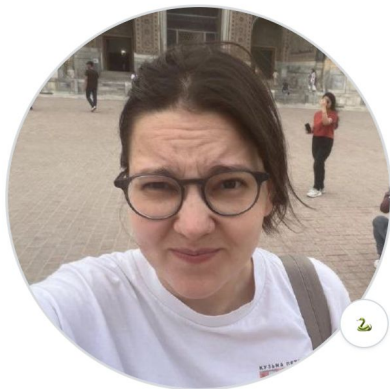
# Кодогенерация: как компьютеры учатся писать код за нас

или мы их учим..

```
mov    rax, 0x02000004  
mov    rdi, 1  
mov    rsi, output  
mov    rdx, dataSize
```

Юлия Волкова, CodeScoring

# О спикере



**Iuliia Volkova**  
xnuinside

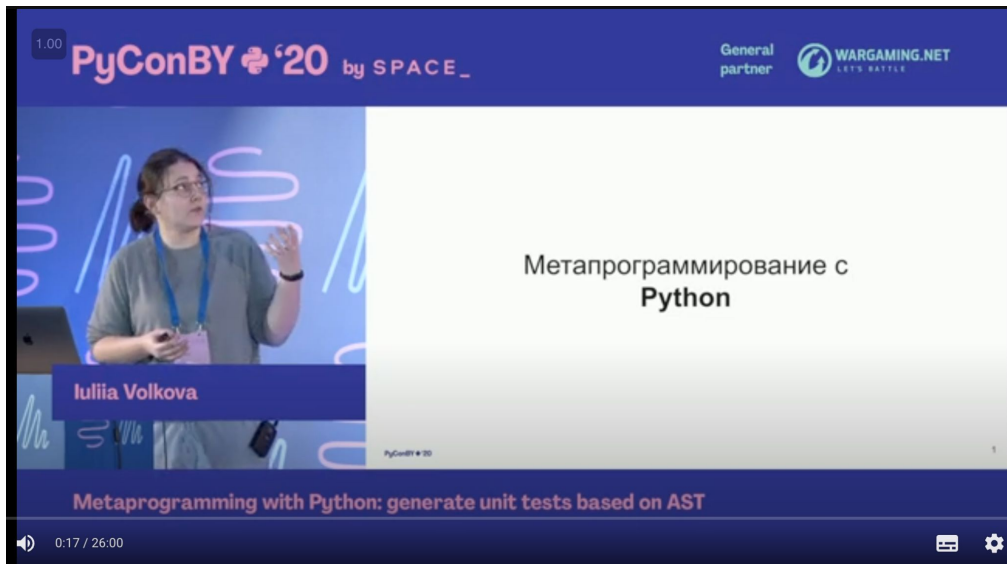
Руководитель отдела сбора и  
анализа данных в CodeScoring

>>> <https://github.com/xnuinside/>

---

8+ лет с Python, SQL; 15+ лет в  
разработке

# Почему я



## Pycon BY 2020

Метапрограммирование  
с Python

[https://www.youtube.com/watch?v=JZ07\\_AAquvY](https://www.youtube.com/watch?v=JZ07_AAquvY)

# Почему я

**TINKOFF** **pycon.ru**

## КОДОГЕНЕРАЦИЯ

- Конвертер DDL в модели: легко
- Генерация unit тестов для Python кода: сложно
- Автоматическая интерпретация "эээ ну мне чтобы вот так же, но чуть по другому" - ...

12:47 / 39:37

## Pycon Russia 2021

Разработка на Python. А можно еще быстрее?

[https://www.youtube.com/watch?v=nD4Y-yM\\_HRs](https://www.youtube.com/watch?v=nD4Y-yM_HRs)

# Агенда

Юля, начни с  
дисплеймера

- что вообще такое кодогенерация
- какие виды бывают
- когда полезно и осуществимо / когда не очень
- история
- поговорим про другие языки программирования
- довольно очевидное будущее

# Что такое вообще эта ваша кодогенерация

генератор

```
def generate_code():  
    print_str = "print('Print something')"  
    return print_str  
  
def write_output(content: str) -> None:  
    with open('generated.py', 'w+') as f:  
        f.write(content)  
  
def main():  
    write_output(generate_code())  
  
main()
```



результат

```
generated.py ×  
meetup_snippets > generated.py  
1 print('Print something')
```

# Кодогенератор - это

Программа, создающая исходный код **другой программы**

# Чуть более практичный пример

## SQL DDL

```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```



## Pydantic

```
class Material(BaseModel):  
    id: int  
    title: str  
    description: Optional[str]  
    link: str  
    type: Optional[MaterialType]  
    additional_properties: Optional[Json]  
    created_at: Optional[datetime.datetime]  
    updated_at: Optional[datetime.datetime]
```



# Чуть более практичный пример

## SQL DDL

```
CREATE TABLE "material" (  
    "id" SERIAL PRIMARY KEY,  
    "title" varchar NOT NULL,  
    "description" text,  
    "link" varchar NOT NULL,  
    "type" material_type,  
    "additional_properties" json,  
    "created_at" timestamp DEFAULT (now()),  
    "updated_at" timestamp  
);
```



## SQLAlchemy

```
class Material(Base):  
    tablename = "material"  
  
    id = sa.Column(sa.Integer(), autoincrement=True)  
    title = sa.Column(sa.String(), nullable=False)  
    description = sa.Column(sa.Text())  
    link = sa.Column(sa.String(), nullable=False)  
    type = sa.Column(sa.Enum(MaterialType))  
    additional_properties = sa.Column(JSON())  
    created_at = sa.Column(sa.TIMESTAMP(), server_default=sa.text('now()'))  
    updated_at = sa.Column(sa.TIMESTAMP())
```

# Как это работает?

```
CREATE TABLE "material" (  
  "id" SERIAL PRIMARY KEY,  
  "title" varchar NOT NULL,  
  "description" text,  
  "link" varchar NOT NULL,  
  "type" material_type,  
  "additional_properties" json,  
  "created_at" timestamp DEFAULT (now()),  
  "updated_at" timestamp  
);
```

```
class Material(Base):  
    tablename = "material"  
  
    id = sa.Column(sa.Integer(), autoincrement=True, primary_key=True)  
    title = sa.Column(sa.String(), nullable=False)  
    description = sa.Column(sa.Text())  
    link = sa.Column(sa.String(), nullable=False)  
    type = sa.Column(sa.Enum(MaterialType))  
    additional_properties = sa.Column(JSON())  
    created_at = sa.Column(sa.TIMESTAMP(), server_default=sa.text('now()'))  
    updated_at = sa.Column(sa.TIMESTAMP())
```

# Популярные библиотеки в Python

- 1) [https://docs.pydantic.dev/latest/integrations/datamodel\\_code\\_generator/](https://docs.pydantic.dev/latest/integrations/datamodel_code_generator/)

Open API / Json Schema -> Pydantic Models

- 2) <https://github.com/koxudaxi/fastapi-code-generator>

Open API -> Fast API код

- 3) <https://github.com/agronholm/sqlacodegen>

SQLAlchemy модели из существующей БД

- 4) <https://github.com/openapi-generators/openapi-python-client>

Open API -> python client

- 5) <https://github.com/danielgtaylor/python-betterproto>

Куда без gRPC и protobuf

# Общая идея

Понятная схема,  
DSL, ограниченная грамматика



Генерируем код

И это конечно же  
только первый вид

двигаем дальше



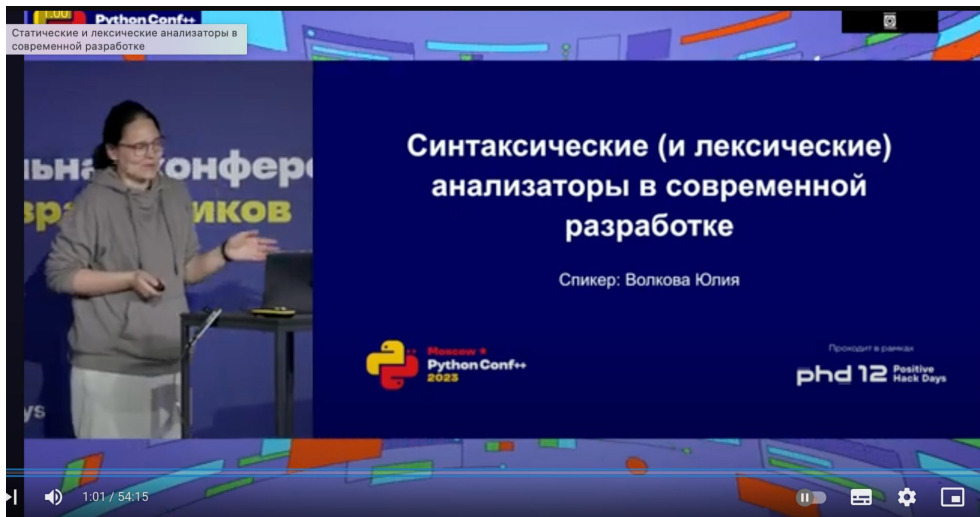
# textX

<https://github.com/textX/textX>

From a **single language description (grammar)**, textX will build a parser and a meta-model (a.k.a. abstract syntax) for the language. See the docs for the details.

The logo for textX, where the word "text" is in a black sans-serif font and the "X" is a large, stylized green letter.

Более подробно можно посмотреть тут:



## Moscow Python 2023:

Статические и лексические  
анализаторы в современной  
разработке

[https://www.youtube.com/watch?v=mh-eLNIMz3M&ab\\_channel=PositiveEvents](https://www.youtube.com/watch?v=mh-eLNIMz3M&ab_channel=PositiveEvents)

# Как интерпретируется код?

Ваш текст  
(.py модуль)

```
def write_output(content: str) -> None:
    with open('generated.py', 'w+') as f:
        f.write(content)
```



# Как интерпретируется код?

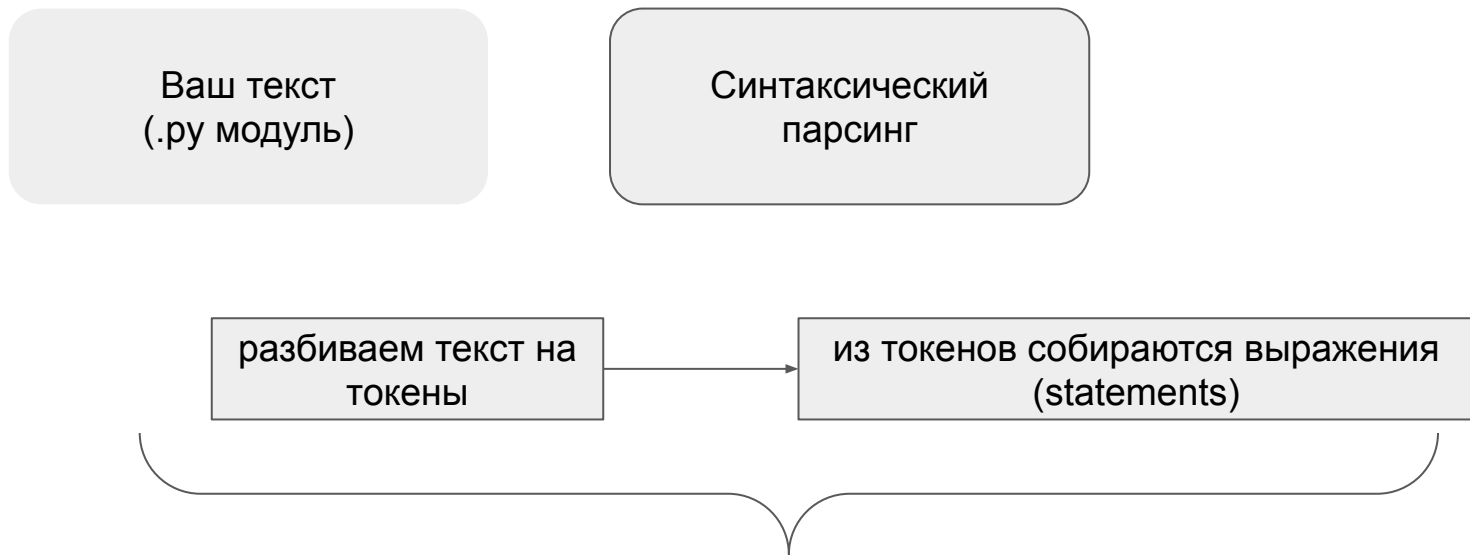
Ваш текст  
(.py модуль)

Синтаксический  
парсинг

разбиваем текст на  
токены

```
def write_output(content: str) -> None:  
    with open('generated.py', 'w+') as f:  
        f.write(content)
```

# Как интерпретируется код?



# Грамматика (пример)

<https://github.com/python/cpython/blob/main/Grammar/python.gram>

```
main ▾ cpython / Grammar / python.gram
Blame 1564 lines (1341 loc) · 69.8 KB · ⓘ
2
3     simple_stmts[asdl_stmt_seq*]:
4         | a=simple_stmt !';' NEWLINE { (asdl_stmt_seq*)_PyPegen_singleton_seq(p, a) } # Not needed, there for speedup
5         | a[asdl_stmt_seq*]=';' .simple_stmt+ [';'] NEWLINE { a }
6
7     # NOTE: assignment MUST precede expression, else parsing a simple assignment
8     # will throw a SyntaxError.
9     simple_stmt[stmt_ty] (memo):
10         | assignment
11         | &"type" type_alias
12         | e=star_expressions { _PyAST_Expr(e, EXTRA) }
13         | &'return' return_stmt
14         | &('import' | 'from') import_stmt
15         | &'raise' raise_stmt
16         | &'pass' pass_stmt
17         | &'del' del_stmt
18         | &'yield' yield_stmt
```

# Как интерпретируется код?

Ваш текст  
(.py модуль)

Синтаксический  
парсинг

(AST) Абстрактное  
синтаксическое дерево

```
import ast

print(ast.dump(ast.parse('x = 1'), indent=4))
```

---

```
Module(
  body=[
    Assign(
      targets=[
        Name(id='x', ctx=Store()),
      ],
      value=Constant(value=1)))
```

textX: придумали свой язык запросов

**client** с **id** диапазон от **200000** до **3000000**

**order** с **created\_at** от **2025-10-10**

# textX

```
1  from textx import metamodel_from_str
2
3  # Грамматика без явного использования terminal
4  grammar = """
5  Model:
6  |   conditions+=Condition*
7  ;
8
9  Condition:
10 |   subject=Subject 'c' field='id' 'диапазон' 'от' min=INT 'до' max=INT
11 |   | subject=Subject 'c' field='created_at' 'от' date=DATE
12 ;
13
14 Subject:
15 |   'client'
16 |   | 'order'
17 ;
18
19 INT: /\d+;/
20 DATE: /\d{4}-\d{2}-\d{2}/;
21 ID: /[a-zA-Z_][a-zA-Z0-9_]*;/
22 """
23
24 # Создаем метамодель
25 metamodel = metamodel_from_str(grammar)
```

textX: выход

```
client с id диапазон от 200000 до 3000000
```

```
Subject: client, Field: id, Min: 200000, Max: 3000000
```

```
AST for query 1:
```

```
{'subject': 'client', 'field': 'id', 'min': 200000, 'max': 3000000, 'date': None  
<textx:Model instance at 0x1018aa660>}
```

# Генераторы парсеров

- 1) <https://github.com/textX/textX>
- 2) <https://wwwantlr.org/>
- 3) <https://tree-sitter.github.io/tree-sitter/>
- 4) <https://eclipse.dev/Xtext/>
- 5) <https://github.com/pyparsing/pyparsing>
- 6) и тд

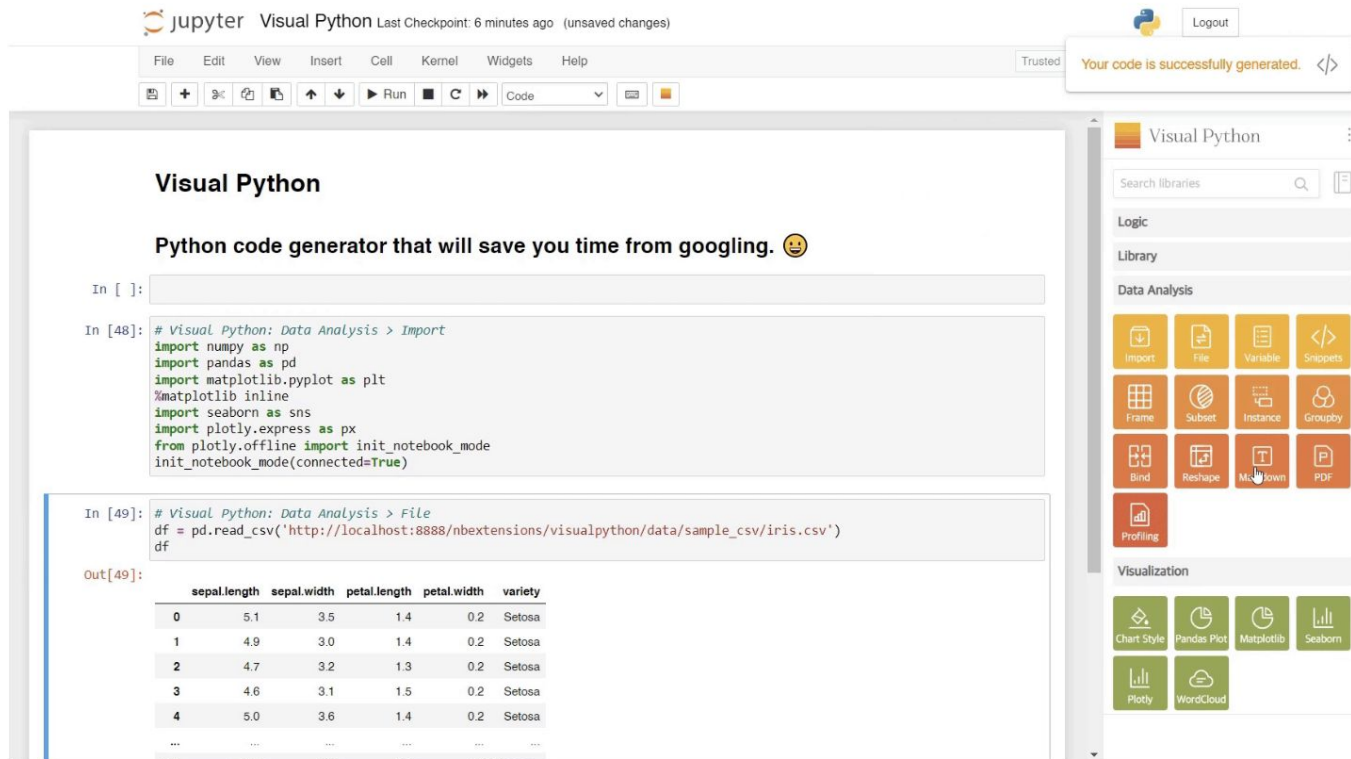


# Форматинг / fix линтеров - это тоже кодогенерация

```
>> ruff check $(diff) --fix-only
```

```
>> black python_file.py
```

# Визуальное программирование



The screenshot displays the Visual Python Jupyter environment. At the top, the header shows 'jupyter Visual Python' and 'Last Checkpoint: 6 minutes ago (unsaved changes)'. A menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with icons for file operations and execution is visible. A notification banner states 'Your code is successfully generated.' with a code icon.

The main workspace is titled 'Visual Python' and contains the text: 'Python code generator that will save you time from googling. 😊'. Below this, two code cells are shown:

```
In [ ]:
```

```
In [48]: # Visual Python: Data Analysis > Import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
from plotly.offline import init_notebook_mode
init_notebook_mode connected=True
```

```
In [49]: # Visual Python: Data Analysis > File
df = pd.read_csv('http://localhost:8888/nbextensions/visualpython/data/sample_csv/iris.csv')
df
```

The output of the second cell is a table:

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...

The right sidebar, titled 'Visual Python', contains a search bar and several categories of tools:

- Logic**
- Library**
- Data Analysis**
  - Import
  - File
  - Variable
  - Snippets
  - Frame
  - Subset
  - Instance
  - Groupby
  - Bind
  - Reshape
  - Mask/Down
  - PDF
  - Profiling
- Visualization**
  - Chart Style
  - Pandas Plot
  - Matplotlib
  - Seaborn
  - Plotly
  - WordCloud

<https://github.com/visualpython/visualpython>

# Кодогенерация по областям/назначению

Форматеры  
кода

Генерация  
boilerplate кода

Кодогенерация в  
компиляторах  
(Intermediate  
Representation как  
пример)

Генерация по  
спецификации (Open  
API, любой DSL)

Транспайлинг (с одного языка  
в другой / с одного вида  
моделей в другой)

Препроцессоры

Визуальные  
генераторы кода

Макросы,  
шаблонизаторы

# Транспайлинг (конечно же не только JS)

**Babel**

ES6+ → ES5

**TypeScript**

TypeScript → JavaScript

**CoffeeScript**

CoffeeScript → JavaScript

**Svelte**

Svelte → JavaScript

**Elm**

Elm → JavaScript

**ReasonML / ReScript**

Reason → JavaScript

# Транспайлинг (в Python)

<b>Cython</b>	Python → C
<b>Nuitka</b>	Python → C/C++ → бинарник
<b>Transcrypt</b>	Python → JavaScript
<b>Pyjs</b>	Python → JavaScript
<b>py-backwards</b>	Python 3.6 → Python 2.7

# Препроцессоры

```
#include <iostream>          // Подключаем стандартный ввод-вывод

#define PI 3.14159           // Заменит PI на 3.14159
#define SQUARE(x) ((x)*(x)) // Макрос с аргументом
#define DEBUG                // Задаём флаг DEBUG

int main() {
    double radius = 5.0;
    double area = PI * SQUARE(radius);

#ifdef DEBUG
    std::cout << "[DEBUG] radius = " << radius << std::endl;
    std::cout << "[DEBUG] area = " << area << std::endl;
#endif

    std::cout << "Area of circle: " << area << std::endl;
    return 0;
}
```

# Препроцессоры

```
#include <iostream>      // Подключаем стандартный
                          // ВВОД-ВЫВОД


#define PI 3.14159        // Заменит PI на 3.14159
#define SQUARE(x) ((x)*(x)) // Макрос с аргументом
#define DEBUG             // Задаём флаг DEBUG

int main() {
    double radius = 5.0;
    double area = PI * SQUARE(radius);

#ifdef DEBUG
    std::cout << "[DEBUG] radius = " << radius <<
std::endl;
    std::cout << "[DEBUG] area = " << area << std::endl;
#endif

    std::cout << "Area of circle: " << area <<
std::endl;
    return 0;
}
```

... тут будет код модуля iostream ...



```
int main() {
    double radius = 5.0;
    double area = 3.14159 * ((x)*(x));

    std::cout << "Area of circle: " << area <<
std::endl;
    return 0;
}
```

Когда кодогенерация  
это хорошо?





# Попытка в оценку сложности

код - читайте как "вход" для  
кодогенератора

$C(L)$  = Сложность анализа кода на языке  $L$

$$C(L) \propto f(S, T, G, D)$$

# Попытка в оценку сложности

$$C(L) \propto f(S, T, G, D)$$

**S** — **семантическая мощность** языка (например, поддержка исключений, динамики, first-class функций, метапрограммирования)

**T** — **число токенов** в языке (лексем, ключевых слов, операторов, специальных конструкций)

**G** — **грамматическая мощность** (зависит от количество продукций, разветвлённость правил (количество альтернатив в одном правиле, рекурсивная глубина)

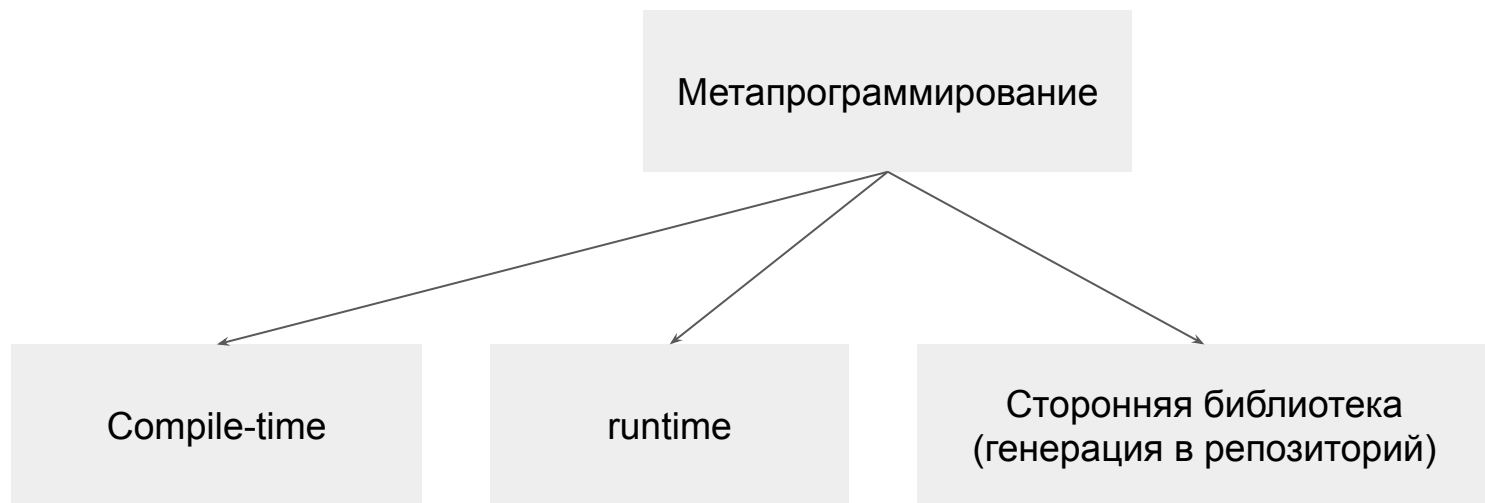
**D** — **динамичность языка** (например, eval, dynamic typing, monkey patching, runtime dispatch)

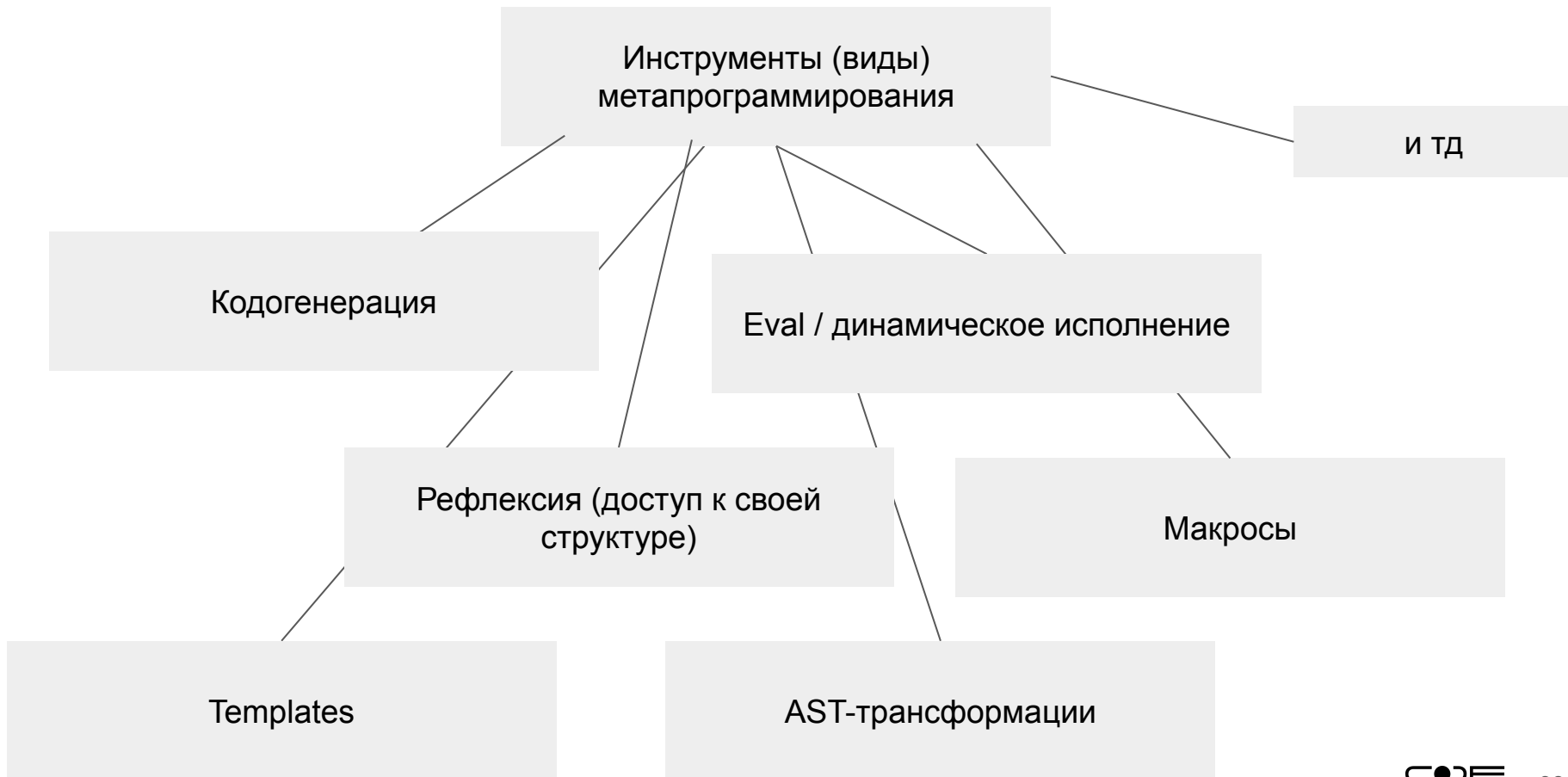
Натуральный язык - **невозможная**  
задача для алгоритмической  
кодогенерации (поэтому LLM ...)

# Метапрограммирование

Метапрограммирование - это создание программ, которые порождают другие программы

## По времени исполнения





# Метапрограммирование, но не кодогенерация

```
# Исходная функция  
def say_hello():  
    print("Hello")
```

# Метапрограммирование, но не кодогенерация

```
# Получаем AST этой функции
source = inspect.getsource(say_hello)
tree = ast.parse(source)

# Модификатор AST: заменяем строку "Hello" на "Goodbye"
class ModifyPrintVisitor(ast.NodeTransformer):
    def visit_Expr(self, node):
        if (isinstance(node.value, ast.Call) and
            isinstance(node.value.func, ast.Name) and
            node.value.func.id == "print" and
            isinstance(node.value.args[0], ast.Constant) and
            node.value.args[0].value == "Hello"):

            node.value.args[0] = ast.Constant(value="Goodbye")
        return node

# Применяем модификацию
modified_tree = ModifyPrintVisitor().visit(tree)
ast.fix_missing_locations(modified_tree)
```



# Метапрограммирование, но не кодогенерация

```
# Компилируем и выполняем модифицированную функцию
code = compile(modified_tree, filename="<ast>", mode="exec")
namespace = {}
exec(code, namespace)

# Вызываем изменённую функцию
namespace['say_hello']()
```

```
(meetup-snippets-py3.13) uliavolkova@MacBook-Pro-
Goodbye
(meetup-snippets-py3.13) uliavolkova@MacBook-Pro-
```

# Гораздо попроще

есть еще `eval`,  
`literal_eval` и тд

```
# Создаём строку с кодом функции
code = """
def greet(name):
    print(f"Привет, {name}!")
"""

# Выполняем строку как обычный Python-код
exec(code)

# Теперь функция реально существует и её можно вызвать
greet("Юля")
```

```
(meetup-snippets-py3.13) uliavolkova@MacBook-Pro-Ulia meetup_snippets % python meetup_s
Привет, Юля!
(meetup-snippets-py3.13) uliavolkova@MacBook-Pro-Ulia meetup_snippets %
```

# LISP - Царь метапрограммирования (и его младший брат Clojure)

```
(defmacro while (var &body body)
  `(do ()
      ((not ,var))
      ,@body))
```

```
(defun increment-until-five ()
  (let ((x 0))
    (while (< x 5)
      (progn
        (setf x (+ x 1))
        (format t "x is: ~A~%" x))))))

(increment-until-five)
```

# LISP - макросы

```
(defun increment-until-even ()  
  (let ((x 1))  
    (while (oddp x)  
      (progn  
        (setf x (+ x 1))  
        (format t "x is event: ~A~%" x))))))  
  
(increment-until-even)
```

# Инструменты метапрограммирования в языках

Язык	Макросы	Рефлексия / introspection	Генерация кода	Доступ к AST / синтаксису	Eval / исполнение кода	Прочее
Python	❌ (но есть Jinja2, Мако для шаблонов)	✅ <code>getattr</code> , <code>type</code> , <code>inspect</code>	✅ через <code>exec</code> , AST, Jinja	✅ <code>ast</code> модуль	✅ <code>eval</code> , <code>exec</code>	Метаклассы, декораторы, codegen из DSL
C++	✅ <code>Templates</code> , <code>constexpr</code> , <code>macro</code>	⚠️ ограниченная	✅ TMP, генерация через шаблоны	⚠️ AST через Clang API	❌	SFINAE, Concepts, template metaprogramming
Rust	✅ <code>macro_rules!</code> , procedural macros	⚠️ ограниченная	✅ <code>build.rs</code> + макросы	✅ через <code>syn</code> , <code>quote</code>	❌	Crate-based генерация, derive macros
Java	❌ (нет "настоящих" макросов)	✅ <code>Reflection</code> API	⚠️ Аннотации + APT	⚠️ Только в compile-time tools	❌	Annotation Processing, bytecode tools
JavaScript	❌ (нет макросов), но шаблоны в TS	✅ <code>typeof</code> , proxies	✅ динамическ и из строки	⚠️ AST через Babel / TS API	✅ <code>eval</code> , <code>new Function()</code>	Метапрограммировани е через прокси / Proxy

# Инструменты метапрограммирования в языках

Язык	Макросы	Рефлексия / introspection	Генерация кода	Доступ к AST / синтаксису	Eval / исполнение кода	Прочее
Lisp (Common)	✓ <code>defmacro</code> , <code>macrolet</code> , <code>reader macros</code>	✓ полный доступ к структуре	✓ AST = код = данные	✓ homoiconicity	✓ <code>eval</code>	Является эталоном гомоиконного языка
C#	✗ (но есть Source Generators)	✓ <code>System.Reflection</code>	✓ Roslyn Source Generators	✓ через Roslyn	✓ <code>Expression.Compile()</code>	Partial classes, Expression Trees
Go	✗	⚠ Ограниченная через <code>reflect</code>	⚠ Gen-tools (go generate)	⚠ нет AST API напрямую	✗	<code>go/ast</code> пакеты, но нет встроенной генерации
Julia	✓ Макросы ( <code>@macro</code> )	✓ <code>typeof</code> , <code>fieldnames</code> , <code>eval</code>	✓ AST и код как данные	✓ через <code>Meta</code>	✓ <code>eval</code>	гомоиконный, мощные макросы и introspection
Scala	✓ Macros, inline	✓ <code>TypeTags</code> , <code>Reflection</code>	✓ Compiler plugins, macros	✓ через Quasiquotes	⚠ ограничено	Metaprogramming в Scala 3 через <code>inline</code>

К истории !



# Сначала был ассемблер

```
BITS 64
GLOBAL _start

SECTION .data align=4
msg1 db "Привет, мир 1", 10
len1 equ $ - msg1

msg2 db "Привет, мир 2", 10
len2 equ $ - msg2

SECTION .text align=4
_start:
    ; print msg1
    mov     rax, 0x2000004
    mov     rdi, 1
    lea     rsi, [rel msg1]
    mov     rdx, len1
    syscall

    ; print msg2
    mov     rax, 0x2000004
    mov     rdi, 1
    lea     rsi, [rel msg2]
    mov     rdx, len2
    syscall

    ; exit
    mov     rax, 0x2000001
    xor     rdi, rdi
    syscall
```

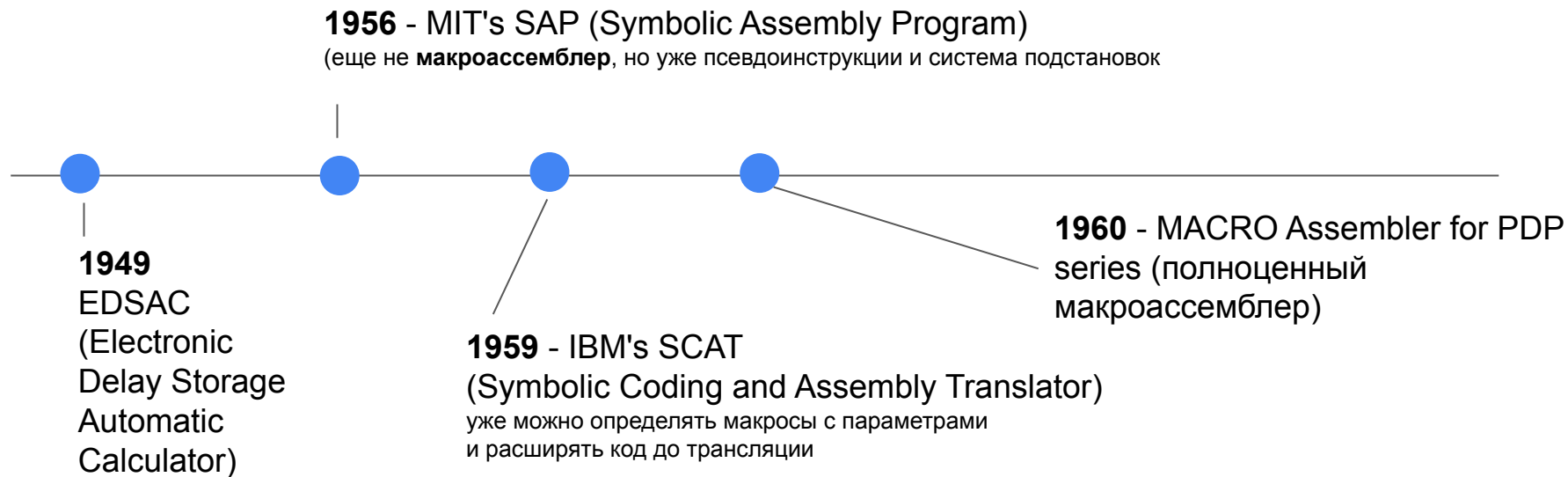
mov rax, 1



48 C7 C0 01 00 00 00



# Основные вехи развития (50-60е): ассемблеры



# Макросы

```
BITS 64
GLOBAL _start

SECTION .data align=4
msg1 db "Привет, мир 1", 10
len1 equ $ - msg1

msg2 db "Привет, мир 2", 10
len2 equ $ - msg2

SECTION .text align=4
_start:
    ; print msg1
    mov     rax, 0x2000004
    mov     rdi, 1
    lea     rsi, [rel msg1]
    mov     rdx, len1
    syscall

    ; print msg2
    mov     rax, 0x2000004
    mov     rdi, 1
    lea     rsi, [rel msg2]
    mov     rdx, len2
    syscall

    ; exit
    mov     rax, 0x2000001
    xor     rdi, rdi
    syscall
```



```
BITS 64
GLOBAL _start

SECTION .data align=4
msg1 db "Привет, мир 1", 10
len1 equ $ - msg1

msg2 db "Привет, мир 2", 10
len2 equ $ - msg2

%macro print 2
    mov     rax, 0x2000004 ; syscall: write
    mov     rdi, 1 ; stdout
    lea     rsi, [rel %1]
    mov     rdx, %2
    syscall
%endmacro

SECTION .text align=4
_start:
    print msg1, len1
    print msg2, len2

    mov     rax, 0x2000001 ; syscall: exit
    xor     rdi, rdi
    syscall
```

# Основные вехи развития (50-60е)



1957 -  
FORTRAN

# Основные вехи развития

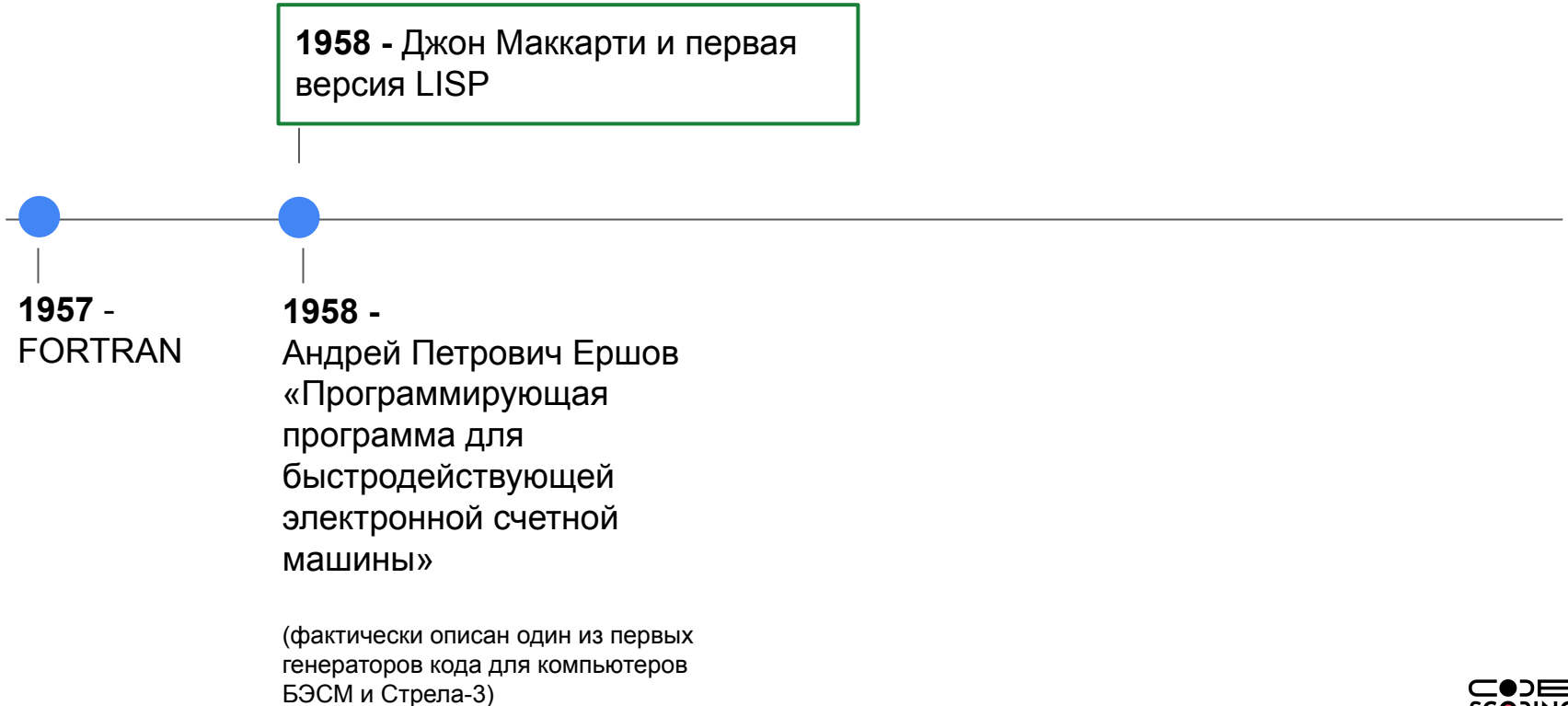


**1957 -**  
FORTRAN

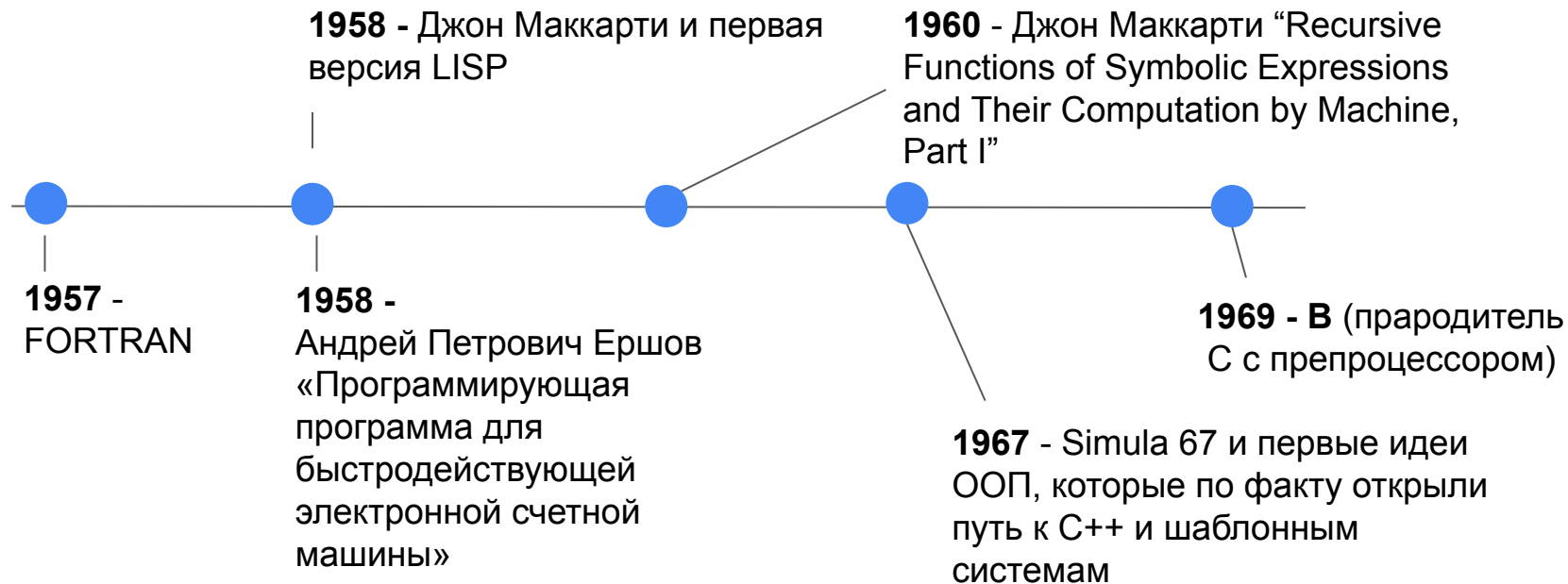
**1958 -**  
Андрей Петрович Ершов  
«Программирующая  
программа для  
быстродействующей  
электронной счетной  
машины»

(фактически описан один из первых  
генераторов кода для компьютеров  
БЭСМ и Стрела-3)

# Основные вехи развития



# Основные вехи развития



# 70e

**1972** — Создан язык C: препроцессор `cpr` становится массовым механизмом "грубоватой" кодогенерации.

**1974** — MacLisp: развивается система макросов, позднее появятся `reader macros`, `defmacro`.

**1977** — Появляется Yacc: генератор парсеров, создающий C-код из грамматик — классический DSL → `codegen` инструмент.

**1979** — Язык ML (Meta Language) с сильной типизацией и `pattern matching` → база для метапрограмм в функциональной парадигме.

# 80e

**1983** — Создан C++ — вводит шаблоны (templates) и перегрузку как первые формы "типовой" метапрограммы.

**1984** — Common Lisp официально стандартизируется: поддержка макросов, макросов макросов, eval-when.

**1985** — Создание Perl: поддержка генерации кода через строки, шаблоны, интерпретацию.

**1989** — ANSI C (C89): препроцессор официально стандартизирован.



# 90e

**1991** — Создан Python: динамический язык с eval, exec, функциями первого класса, позднее — metaclass.

**1994** — Template Haskell (экстеншен Haskell 98): появляется как форма compile-time метапрограммирования.

**1995** — Появляются PHP, JavaScript — активно используют строки как исполняемый код, eval.

**1997** — Boost C++ запускает template metaprogramming в промышленных масштабах.

**1998** — Стандарт C++98 включает шаблоны, constexpr, и SFINAE как ядро метапрограммирования.

00e

**2000** — Проект LLVM: создание IR (intermediate representation), который активно используется в генерации кода.

**2001** — Появляется Jinja (шаблонизатор Python) — генерация HTML, YAML, Python и т.д.

**2003** — Clang начинает использовать LLVM как backend с фазой codegen.

**2004** — Появляется Rust в Mozilla: позже предложит procedural macros и macro\_rules!.

# 10e

**2010** — Начинается бурное развитие template-based DSL codegen: Swagger/OpenAPI, Protobuf, GraphQL codegen.

**2012** — Запуск Hypothesis (Python): property-based генерация тестов — автоматическое покрытие кода

**2014** — Rust 1.0: полноценная система макросов и процедурных макросов как система метапрограммирования.

**2015** — Запуск TypeScript: создаёт JS-код из типизированного супермножества, став популярным транспайлером.

**2017** — Первый релиз Pynguin: автоматическая генерация тестов по коду (стоит отметить, что инструмент до сих пор в стадии “эксперимента” и скорее всего без LLM так и останется в ней)

**2018** — C++20 разрабатывает концепции и constexpr-программы как частичный DSL.

# 20e

**2020** — OpenAI выпускает GPT-3, который уже может писать код.

**2021** — GitHub + OpenAI запускают Copilot — массовый инструмент генерации кода.

**2022** — Массовое распространение LLM-кодогенерации: Codex, Tabnine, Replit Ghostwriter.

**2023** — Появляются модели с поддержкой fine-tuned code instruction: StarCoder, CodeLlama, DeepSeek-Coder.

**2024** — Meta Llama 3, CodeGemma, Mistral 7B

**2025** — DeepSeek-R1, Mistral Medium 3, Devstral, Gemini 2.5 Pro / Flash, Claude 3.7 Sonnet и Claude Opus 4 / Sonnet 4 **и мы где-то тут с вами сейчас**

# Вайб-кодинг, LLM и современность

Вайб кодинг.

Хорошо, или плохо? Легко, или сложно? ...more

Show translation

⚡ VIBE CODING: The New Era of Developers Has Arrived

Ever coded with music on, AI beside you, and zero stress? ...more

# Вайб-кодинг, LLM и современность



Refrigeration.AI  
[View my services](#)

2w • Edited •

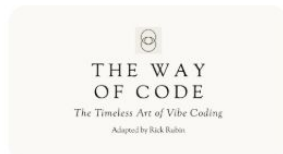
Вайб кодинг.

Хорошо, или плохо? Легко, или с

[Show translation](#)

Vibe coders are the punk rock revolution of software

When we recognize code as elegant, ...more



**Rick Rubin | The Way of Code: The Timeless Art of Vibe Coding**

[thewayofcode.com](http://thewayofcode.com)

⚡ VIBE CODING: The New Era of Developers Has Arrived

Ever coded with music on, AI beside you, and zero stress? ...more

**Repost**

**Send**

# Вайб-кодинг, LLM и современность



Refrigeration.AI  
[View my services](#)

2w • Edited • 🔄

Вайб кодинг.

Хорошо, или плохо? Легко, или с

Show translation

Vibe coders are the punk rock revolution of software

When we recognize code as elegant, ...more

The Vibe Coding Revolution: Separating Fact from Fiction

If you're into tech and active on social media, you've probably heard ...more

⚡ VIBE CODING: The New Era of

Ever coded with music on, AI bes



# VIBE CODING

Вайб кодинг.

Хорошо, или плохо? 🤔

Show translation

volution of software

nt, ...more

...more

⚡ VIBE CODING: The New Era

Ever coded with music on, AI bes

...code coding', where you fu  
...entials, and forget that the code ev  
...because the LLMs (e.g. Cursor Composer w Son  
getting too good. Also I just talk to Composer with SuperWhi  
barely even touch the keyboard. I ask for the dumbest things  
"decrease the padding on the sidebar by half" because I'm to  
find it. I "Accept All" always, I don't read the diffs anymore. W  
error messages I just copy paste them in with no comment, u  
fixes it. The code grows beyond my usual comprehension, I c  
really read through it for a while. Sometimes the LLMs can't f  
just work around it or ask for random changes until it goes av  
too bad for throwaway weekend projects, but still quite amu  
building a project or webapp, but it's not really coding - I just  
say stuff, run stuff, and copy paste stuff, and it mostly works

1:17 AM · Feb 3, 2025 · 3.3M Views



Что тут можно сказать..

## Claude 4

Claude Opus 4 is our most powerful model yet and the best coding model in the world, leading on SWE-bench (72.5%) and Terminal-bench (43.2%). It delivers sustained performance on long-running tasks that require focused effort and thousands of steps with the

# Про бенчмарки

- 1) SWE-bench - <https://github.com/SWE-bench/SWE-bench>
- 2) HumanEval - <https://github.com/openai/human-eval>
- 3) DS-1000 - <https://ds1000-code-gen.github.io>
- 4) MPBB (Mostly Basic Python Problems)  
<https://github.com/google-research/google-research/tree/master/mbpp>
- 5) APPS (Automated Programming Progress Standard) -  
<https://github.com/hendrycks/apps>
- 6) MultiPL-E - <https://github.com/nuprl/MultiPL-E>
- 7) Speedrunning Benchmark - <https://github.com/facebookresearch/llm-speedrunner>
- 8) и другие

# HumanEval

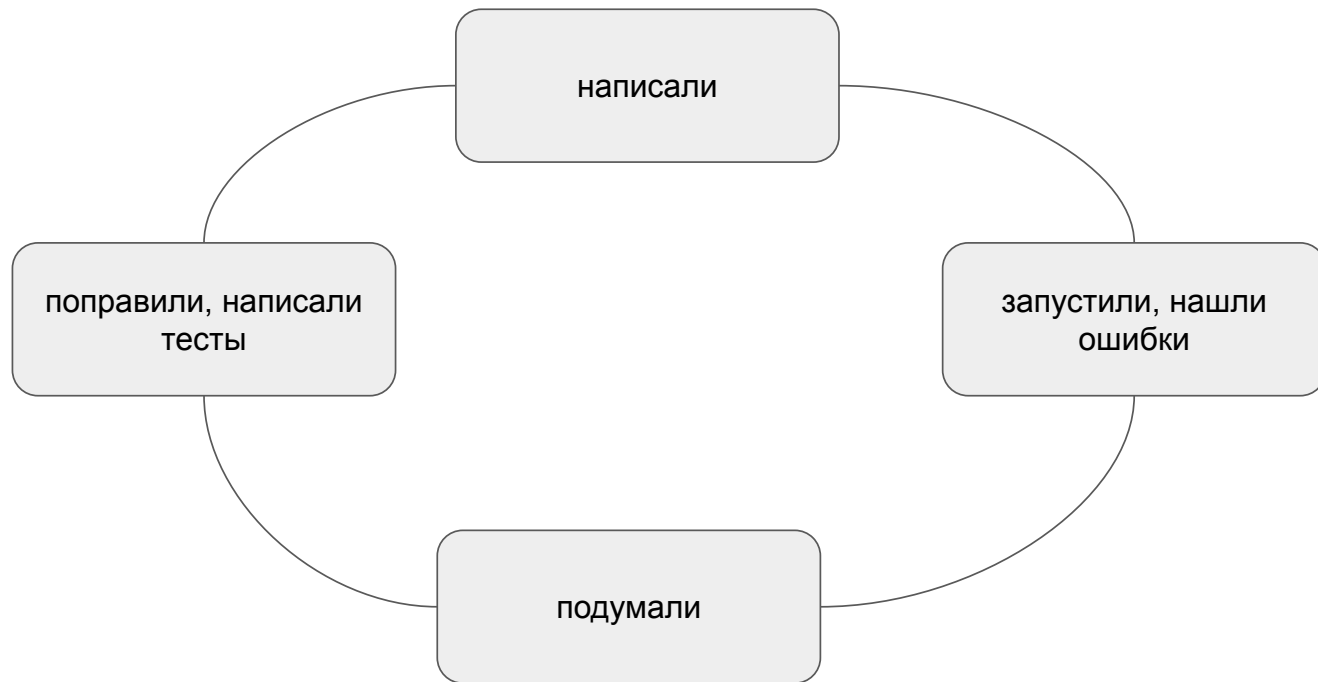
<b>GPT-4</b>	67,0%	Март 2023 г.
<b>Claude 2</b>	71,2%	Июль 2023 г.
<b>Code Llama 34B-Python</b>	53,7%	Август 2023 г.
<b>Phind-CodeLlama 34B v2</b>	73,8%	Август 2023 г.
<b>StarCoder (15B)</b>	40,8%	Май 2023 г.
<b>DeepSeek-Coder 33B</b>	75,0%	Январь 2024 г.
<b>Claude 3 Opus</b>	<b>84,9%</b>	<b>Март 2024 г.</b>

# SWE-bench

<b>GPT-4</b> (агент)	≈ <b>20%</b> (полный набор)	авг. 2024 г.	<a href="#">link</a>
<b>Claude 2</b> (без агентов)	<b>1,96%</b> (с ретривером BM25)	2024 г.	<a href="#">link</a>
<b>Claude 3.5 Sonnet</b> (агент)	<b>49%</b> (SWE-бенч Verified)	янв. 2025 г.	<a href="#">link</a>
<b>Code Llama</b> / др. открытые	~15–20% (лучшие агенты на 2024 г.)	2024 г.	<a href="#">link</a>

<https://github.com/SWE-bench/SWE-bench>

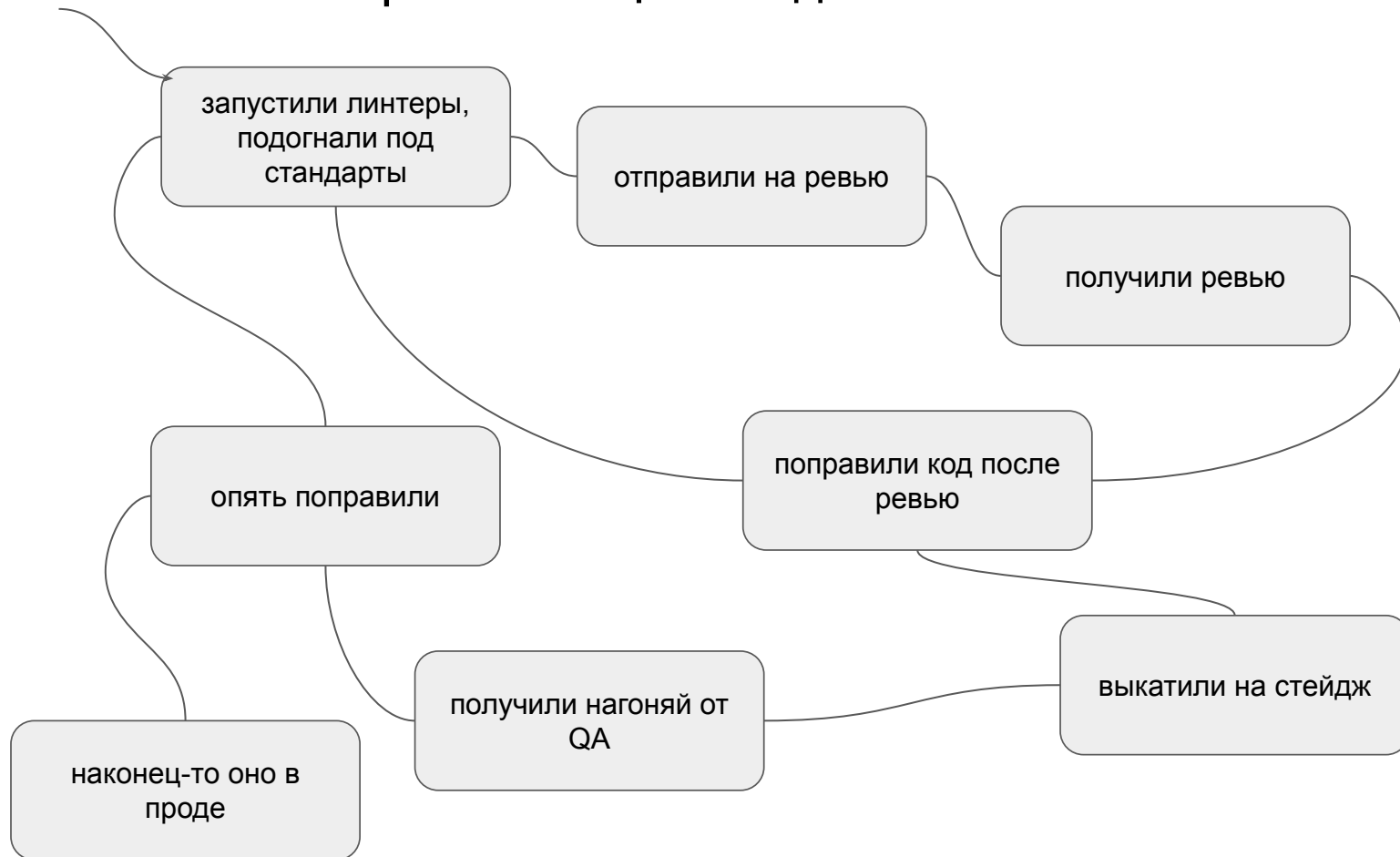
# Цикл написания работающего кода



# Цикл написания работающего кода



# Цикл написания работающего кода



# Подводя итоги

<b>1950-е</b>	Макроассемблеры, символика	Текстовая подстановка
<b>1960-е</b>	Макросы Lisp, DSL, парсеры	AST, eval, гомоиконность
<b>1970-е</b>	C и препроцессоры, Yacc	DSL → code, препроцессоры
<b>1980-е</b>	Шаблоны, OOP, TeX	Компилируемые шаблоны, TMP
<b>1990-е</b>	Рефлексия, templates, eval	Runtime + compile-time mix
<b>2000-е</b>	Шаблонизация YAML, спецификации	Генерация по DSL, IR, Model-Driven Architecture (MDA)
<b>2010-е</b>	AST, тестирование, transpilers	Property-based, AST codegen
<b>2020-е</b>	LLM, мультязычная генерация, пайплайны	Prompt2Code, AI DevTools, hybrid gen, No-code системы



# Вместо выводов



- история кодогенерации - это история всего программирования
- желание уменьшать ручную работу - это нормально
- оценивайте сложность при выборе инструмента

# Ссылки

- Recursive Functions of Symbolic Expressions

and Their Computation by Machine, Part I

[https://www.cs.tufts.edu/comp/150FP/archive/john-mccarthy/recursive.pdf?utm\\_source=chatgpt.com](https://www.cs.tufts.edu/comp/150FP/archive/john-mccarthy/recursive.pdf?utm_source=chatgpt.com)

- Ершов “Программирующая программа для быстродействующей электронной счетной машины”

[https://ershov.iis.nsk.su/ru/node/777610?utm\\_source=chatgpt.com](https://ershov.iis.nsk.su/ru/node/777610?utm_source=chatgpt.com)

Всем спасибо за внимание!

Вопросы?



слайды можно  
найти тут

