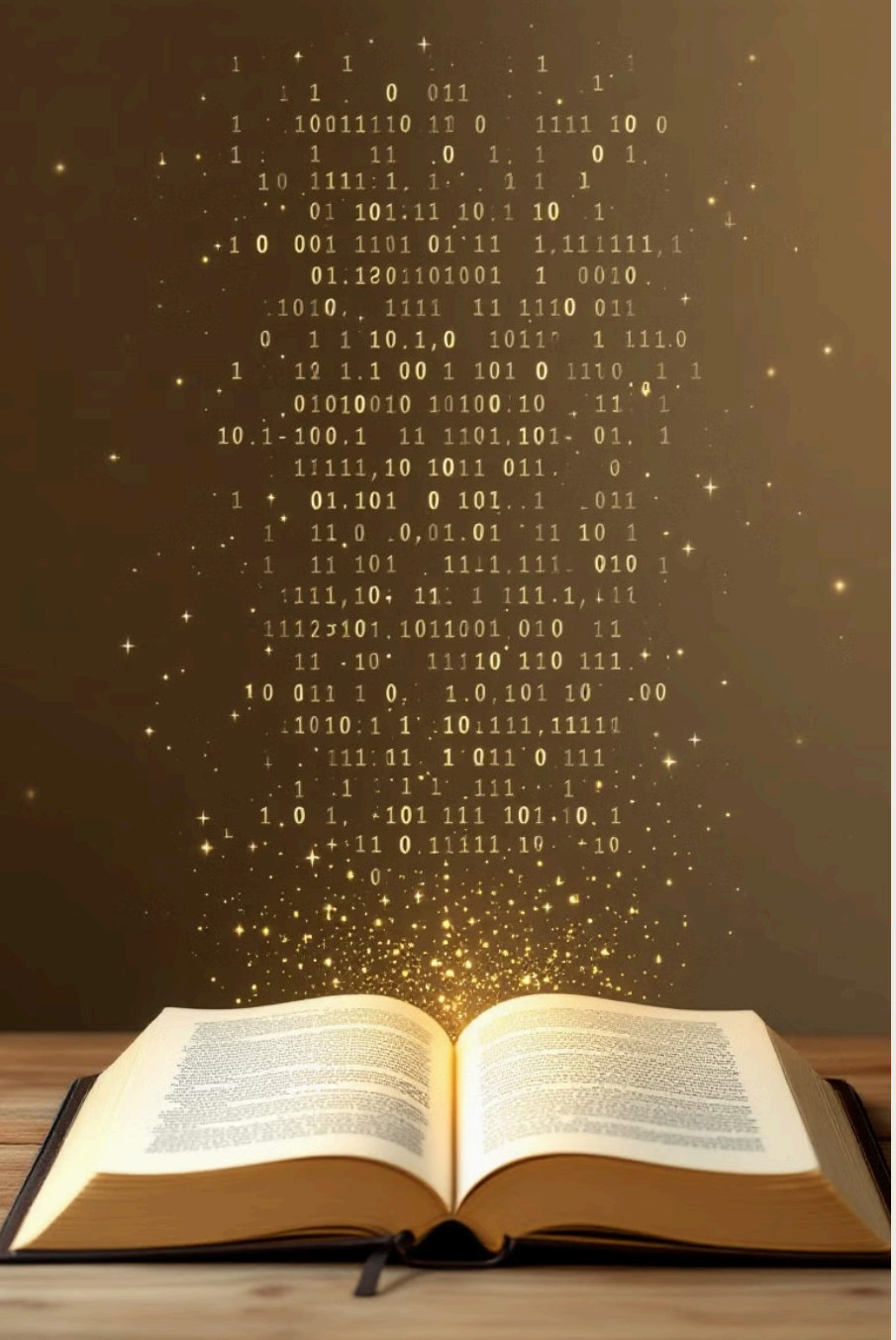


Процедуры, Функции и Рекурсия в Информатике

Добро пожаловать на урок, посвященный процедурам, функциям и рекурсии. Сегодня мы изучим важные концепции, которые помогут вам в программировании. Мы рассмотрим, как разбивать код на модули и как функции могут вызывать сами себя.

AG by Andrej Gorbatniov

Цели урока

1 **Изучение процедур и функций**

Понимание, как создавать и использовать процедуры и функции.

2 **Знакомство с рекурсией**

Освоение рекурсивных алгоритмов.

3 **Применение рекурсии**

Решение задач с использованием рекурсии.

4 **Алгоритмическое мышление**

Развитие навыков разработки алгоритмов.

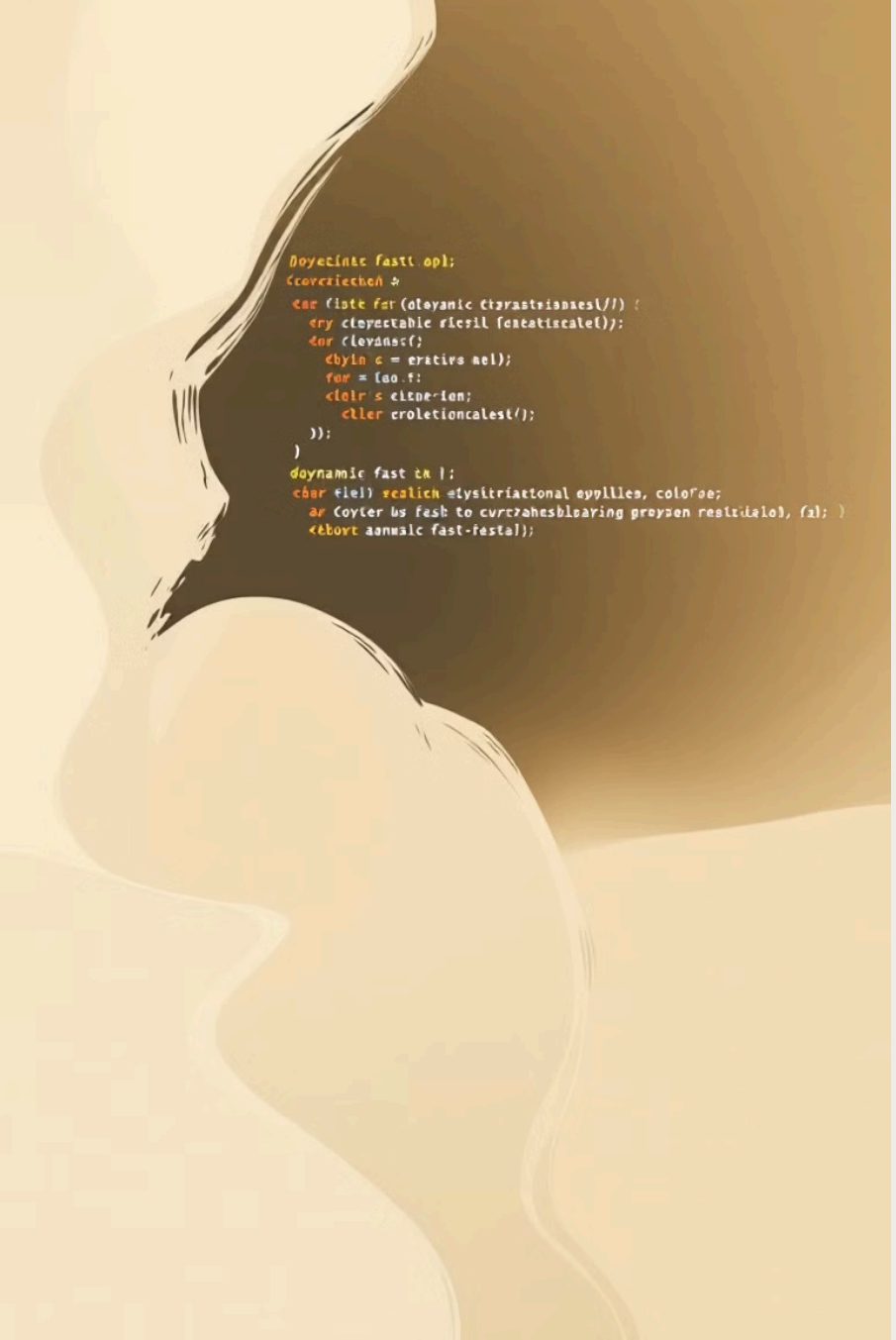
Модульное программирование

Модульность

Разделение программы на отдельные модули.

Преимущества

- Упрощение разработки и отладки
- Повторное использование кода
- Лучшая читаемость
- Командная работа



Процедуры

Определение

Функции без возврата значения.

Пример

```
def print_hello():
    print('Hi')
```

Вызов

```
print_hello()
```

Функции

Определение

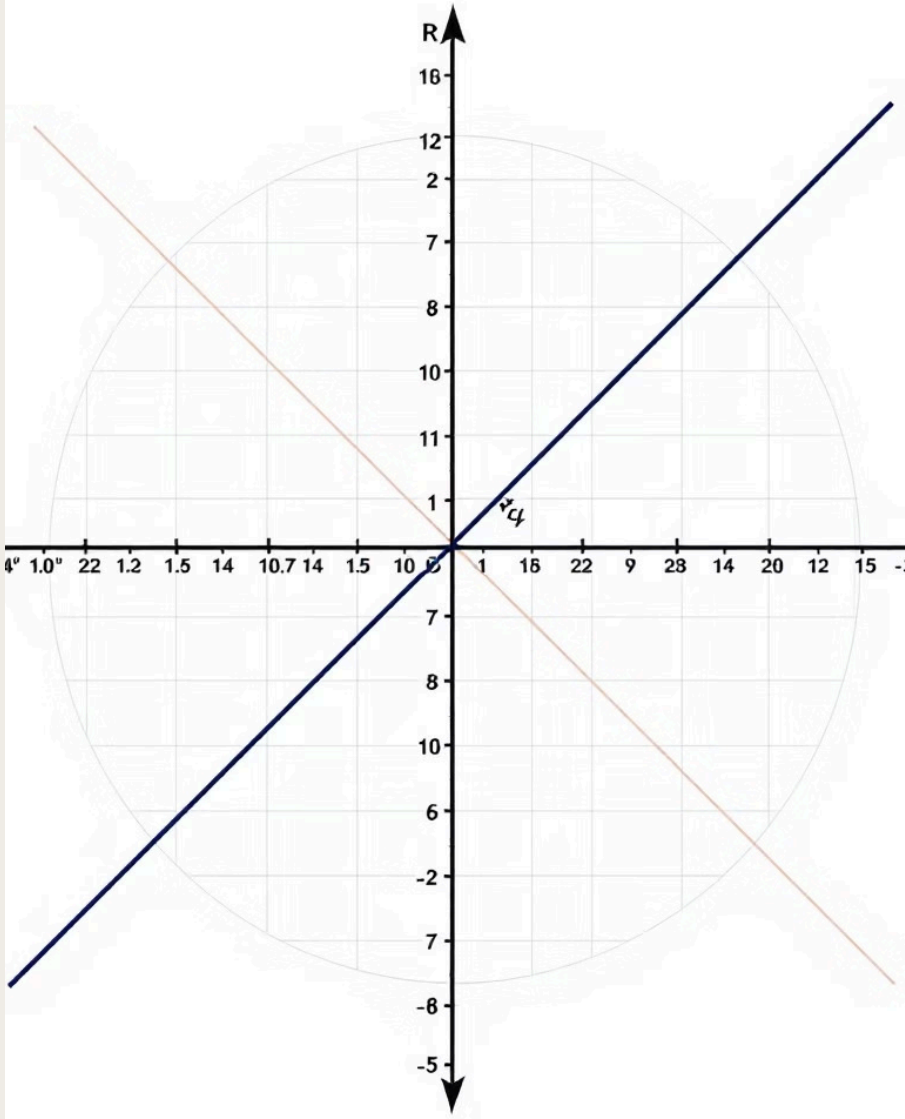
Подпрограммы, возвращающие результат.

Пример

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

Использование

```
total = sum_numbers(5, 3)
```



Параметры и переменные

1

Формальные параметры

В объявлении функции.

2

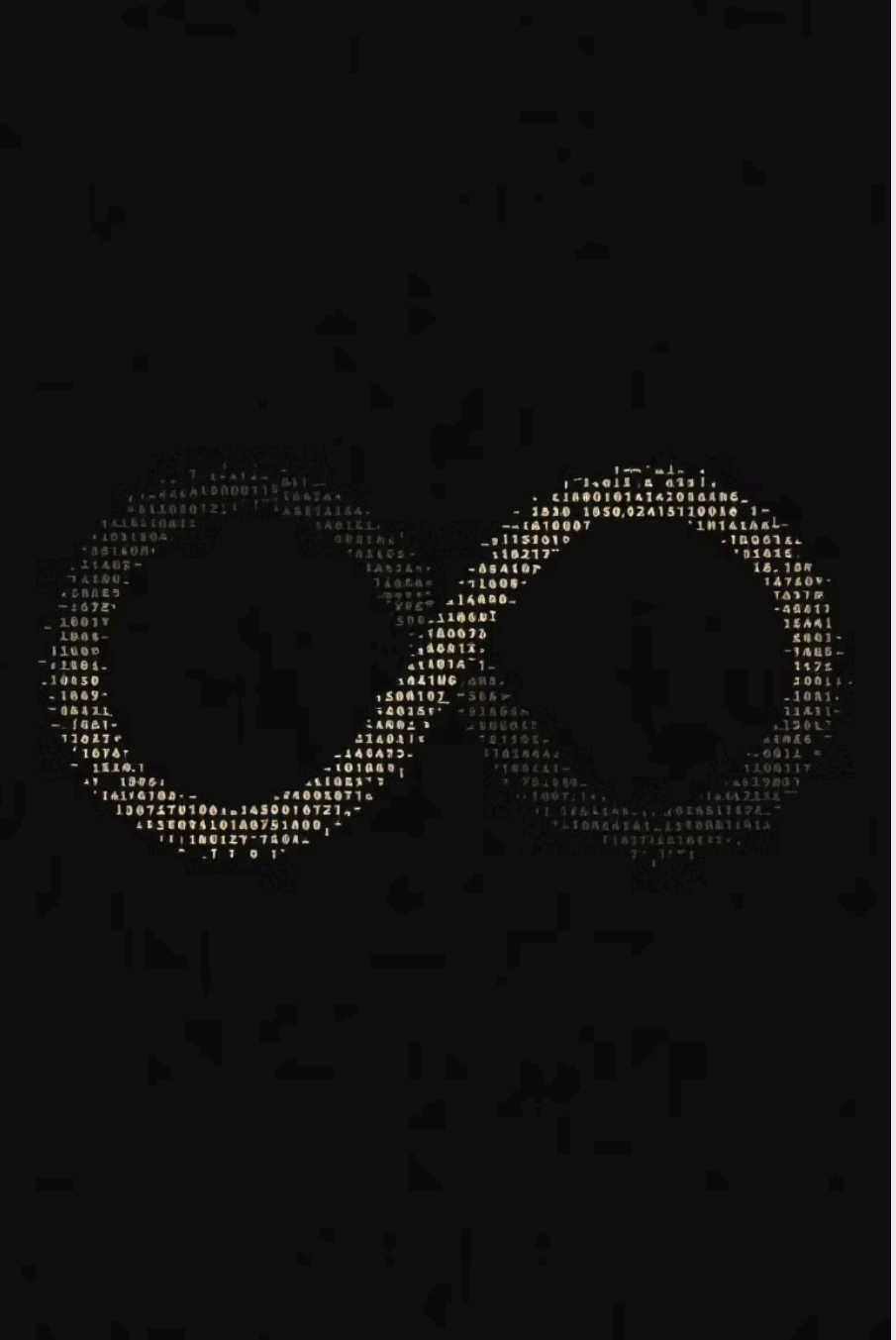
Фактические параметры

При вызове функции.

3

Локальные переменные

Действуют внутри функции.



Что такое рекурсия?

Рекурсия

Метод, когда функция вызывает себя.

Пример

“Чтобы понять рекурсию, нужно сначала понять рекурсию”.

Важно

Обязательно наличие базового случая.

Структура рекурсивного решения

Базовый случай

Условие выхода из рекурсии.

Рекурсивный случай

Сведение к подзадаче.

Рекурсивный вызов

Функция вызывает себя.

Пример — факториал

Определение

n! = n × (n-1) × (n-2) × ... × 1.

Рекурсивное определение

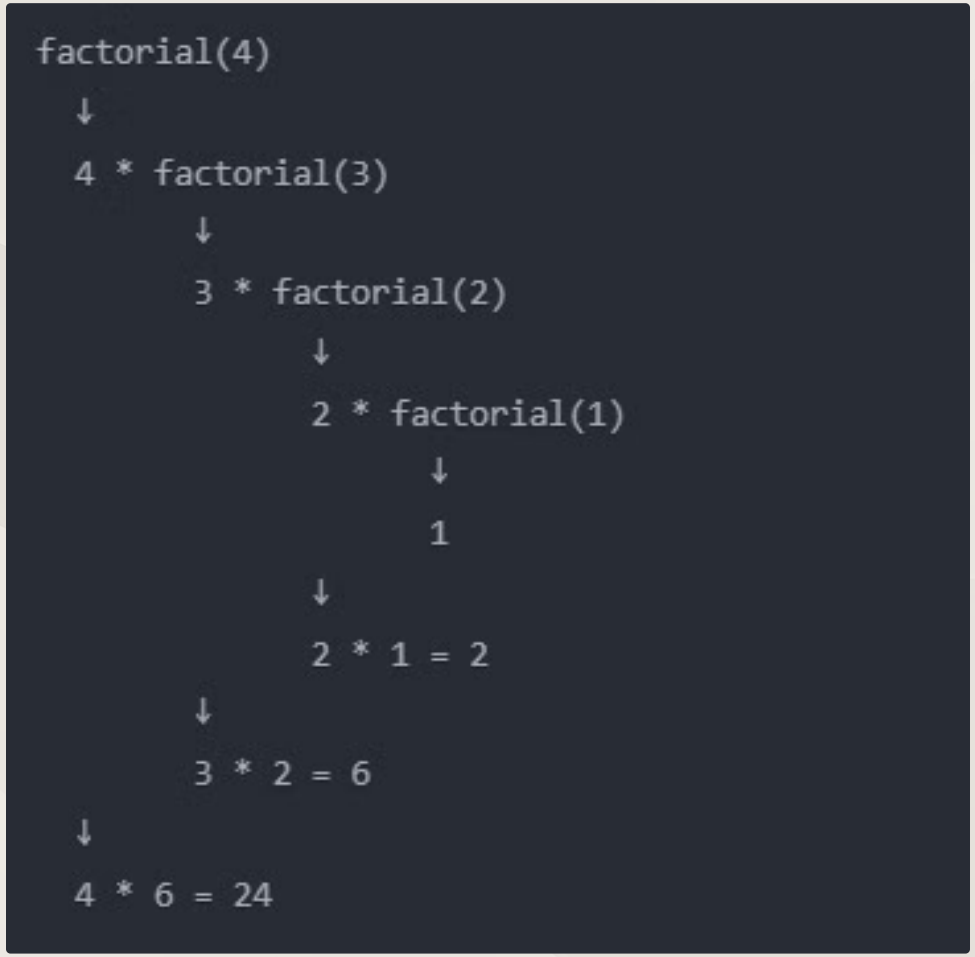
0! = 1 (базовый случай).

n! = n × (n-1)! (рекурсивный случай).

Код

```
def factorial(n):
  if n <= 1:
    return 1
  else:
    return n * factorial(n-1)
```

Пример выполнения factorial(4)



Результат: 24

Числа Фибоначчи

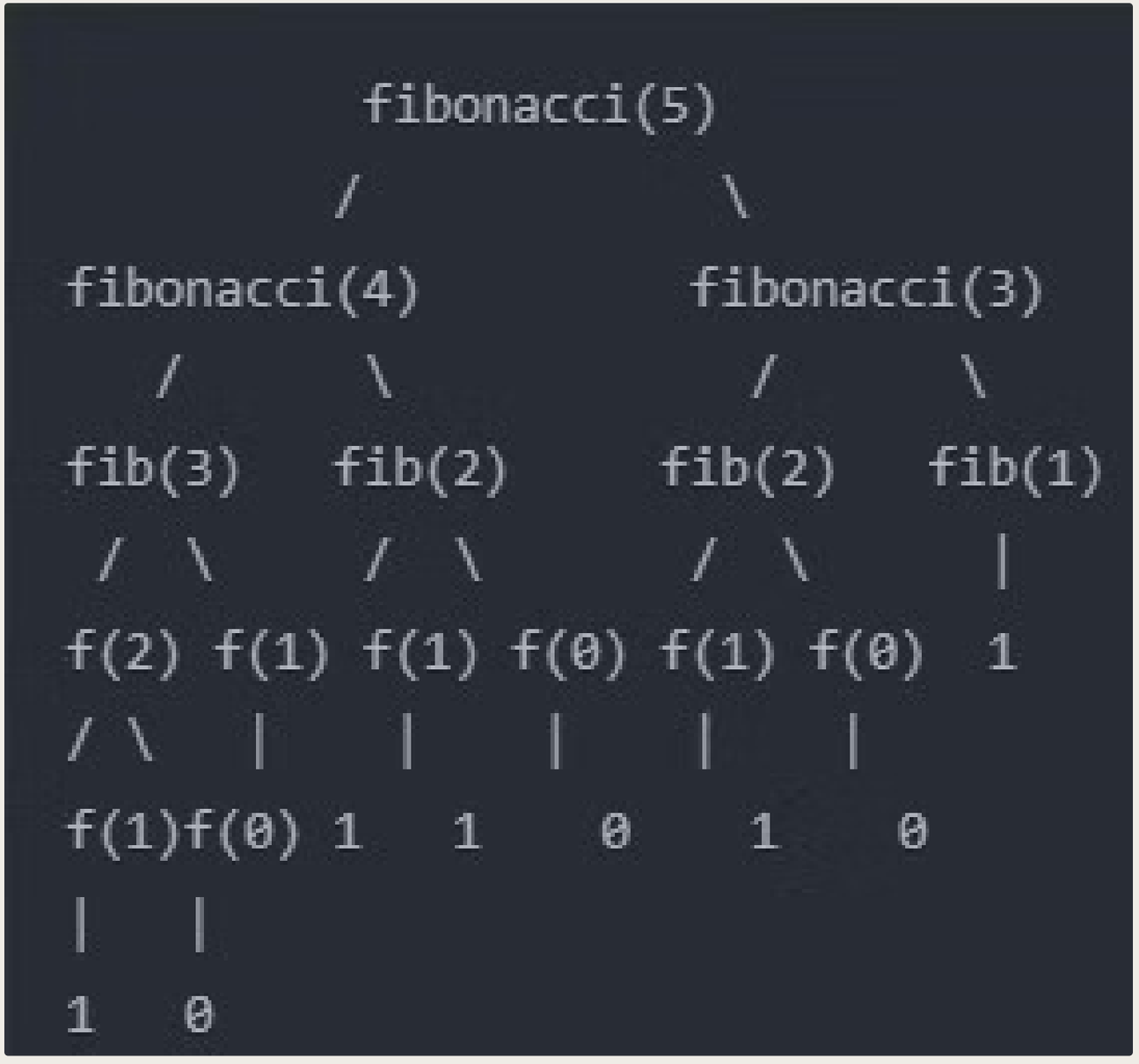
Числа Фибоначчи: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

Рекурсивное определение:

- F(0) = 0, F(1) = 1 (базовые случаи)
- F(n) = F(n-1) + F(n-2) для n > 1 (рекурсивный случай)

```
def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Пример выполнения fibonacci



Задачи для рекурсивного решения

- Вычисление суммы цифр числа
- Возведение в степень
- Алгоритм Евклида (НОД)
- Генерация перестановок
- Обход дерева или графа
- Ханойские башни

Преимущества и недостатки рекурсии

- ✓ Позволяет писать лаконичный и понятный код.
- ✓ Упрощает реализацию сложных алгоритмов (например, обход деревьев).
- ⚠ Может вызывать переполнение стека вызовов (если глубина рекурсии слишком большая).
- ⚠ Иногда работает медленнее, чем итеративный подход.

Практические задания

1. Написать рекурсивную функцию для вычисления факториала
2. Реализовать рекурсивную функцию для вычисления n-го числа Фибоначчи.
3. Дополнительное задание: написать рекурсивную функцию для нахождения суммы цифр числа.

Домашнее задание

1. Написать рекурсивную функцию для проверки, является ли строка палиндромом