GeekBrains

# Структурная и процедурная парадигмы

Урок 2

Курс “Парадигмы программирования и языки парадигм”

# Содержание урока

# План курса

1. Введение и основные типы парадигм
2. Структурное и процедурное программирование
3. ООП
4. Функциональное программирование
5. Логическое программирование

# План урока

- 📌 Структурное программирование
  - С goto или без
  - История структурного программирования
  - Примеры языков
- 📌 Процедурное программирование
  - История процедурного программирования
  - Примеры процедурных языков
- 📌 Примеры кода
  - Countif
  - Сортировка пузырьком
- 📌 Преимущества и недостатки
- 📌 3 реальных кейса

# Структурное программирование

# Термины и определения

💡 **Структурное программирование (Structured Programming)**
императивный стиль, который основан на последовательном исполнении хорошо структурированных “блоков” без goto.

💡 **Утверждение / Последовательность (Sequence)**
однократное выполнение следующей написанной операции.

💡 **Ветвление / Условный оператор (Conditional)**
однократное выполнение любого количества операций внутри ветвления после проверки условия.

💡 **Цикл (Loop)**
многократное исполнение любого количества операций внутри цикла до тех пор, пока заранее определённое условие выполняется.

# Чем отличается

Основное отличие структурного программирования от императивного заключается в том, что программа представляет собой набор блоков строго без использования оператора “goto”.

Пример:

```
#include <iostream>
using namespace std;

int main() {
    int a = 5;
    LOOP:do {
        if (a < 20) {
            a = a + 1;
            goto LOOP;
        }
        cout << "value of a: " << a << endl;
    }
}
```

# С goto или без

1

**Использование GOTO**

+ В некоторых случаях, может быть получена наиболее эффективная **реализация алгоритма** и наименьшая **длина программы**
+ При грамотном использовании, оптимальное **использование ресурсов**

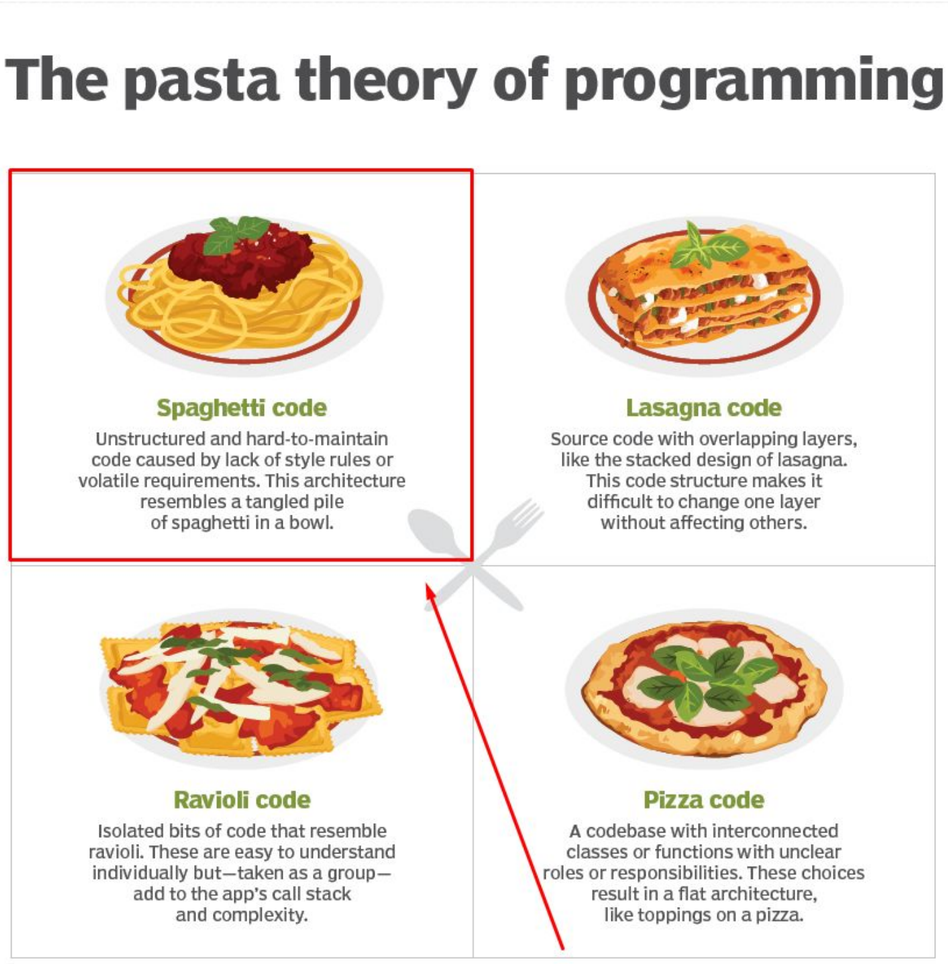
- Возможная **запутанность** (спагетти код): неочевидная последовательность исполнения программы
- Вероятность возникновения ошибок

2

**Запрет GOTO**

+ Программа четко **структурирована** и состоит из блоков кода
+ Наглядность исполнения (отсутствие **спагетти-кода**)

- Код может стать **сложным**: больше строк, чем могло бы быть
- Иногда невозможно получить оптимальный алгоритм




Название источника изображения

# История

Структурное программирование было разработано как ответ на проблему неструктрированного (“спагетти”) кода

- 1966 год - доказательство полноты структурного программирования, статья “**Flow Diagrams, Turing Machines And Languages With Only Two Formation Rules**”
- 1968 год - обоснование “вредности” goto, статья “**Go To Considered Harmful**”
- 1969 год - Логика Хоара, представлена статья “**An Axiomatic Basis for Computer Programming**”
- 1970 год - Язык Pascal Николасом Виртом

# Примеры языков программирования

**с GOTO**

- C, C#, C++
- Pascal
- Perl
- PHP
- Basic
- Go

**без GOTO**

- Python
- Java
- Ruby
- Swift
- Kotlin

# Процедурное программирование

# Термины и определения

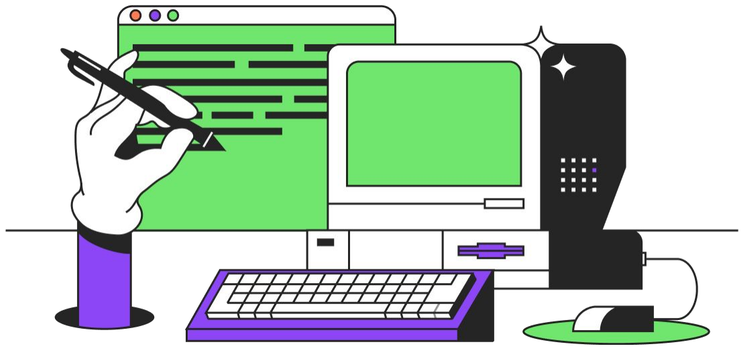
💡 **Процедурное программирование (Procedural Programming)**
императивный стиль, который заключается в процессе написания последовательности команд и вызовов процедур (можно с goto и без структуры)

💡 **Подпрограмма (Subroutine)**
последовательность команд, выделенная в отдельный блок кода, который после объявления можно вызывать по имени. Бывает двух типов: **процедура** и **функция**.

💡 **Процедура (Procedure)**
подпрограмма, результат выполнения которой является выполнение операция, из которых она состоит

💡 **Функция (Function)**
подпрограмма, результатом выполнения которой является возврат значения

💡 **Метод (Method)**
подпрограмма (функция или процедура), которая принадлежит классу (Объектно-ориентированное программирование).



# Синонимы

Процедурное и императивное программирование - не одно и то же

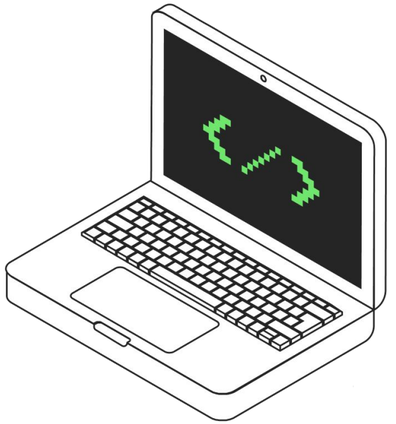
- Императивная парадигма - это общее название стиля описывающего последовательность операций, оно не обязательно является процедурным
- Процедурная парадигма - это императивная парадигма в рамках которой обязательно используются процедуры
- Вы можете писать в императивном стиля и не использовать ни процедурный ни структурный стиль
- Вы не можете писать в процедурном стиле, не используя императивный, т.к. процедурный стиль является видом императивного

# История

Процедурная парадигма - первая парадигма программирования

- Машинный код
- Ассемблер
- Fortran - 1957 год

Кстати, оператор goto в языке Fortran был. Но в более поздних процедурных языках, таких как Pascal, стал вытесняться структурной парадигмой.



# Примеры языков программирования

**Поддержка структурного стиля**

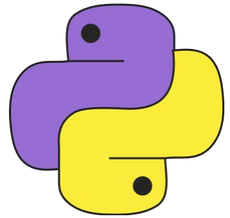
- C, C#, C++
- Python
- Java
- Pascal
- HTML

**Поддержка процедурного стиля**

- C, C#, C++
- Python
- Java
- Pascal
- Assembly
- Basic

# Примеры кода

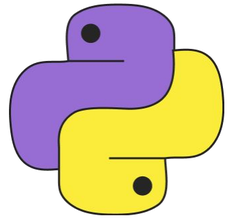
# Count IF



# Count IF - структурная реализация

Рассмотрим следующую задачу. Необходимо реализовать программу, которая:

**Получает на вход**: список li и число n
**Возвращает**: количество элементов в списке li равных этому заданному числу n

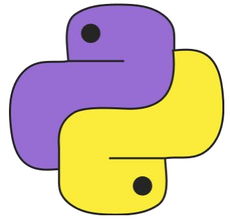


# Count IF - структурная реализация

Рассмотрим следующую задачу. Необходимо реализовать программу, которая:

**Получает на вход**: список li и число n
**Возвращает**: количество элементов в списке li равных этому заданному числу n

```python
## Structured Programming
input_li = input('Enter list li: ') # [1,2,3,4,5,6,6,5,5,1,2,3,5]
input_n = input('Enter number n: ') # 6
counter = 0
for el in input_li:
    if el == input_n:
        counter += 1
print(counter)                      # 2
```



# Count IF - процедурная реализация

Все то же самое, кроме того, что почти вся содержательная часть программы обернута в функцию countif и вызывается в 11 строке

```
 1 ## Procedural Programming
 2 input_li = input('Enter list li: ') # [1,2,3,4,5,6,6,5,5,1,2,3,5]
 3 input_n = input('Enter number n: ') # 6
 4 def countif(li, n):
 5     counter = 0
 6     for el in li:
 7         if el == n:
 8             counter += 1
 9     return counter
10
11 countif(input_li, input_n)          # 2
12
```

# Сортировка пузырьком

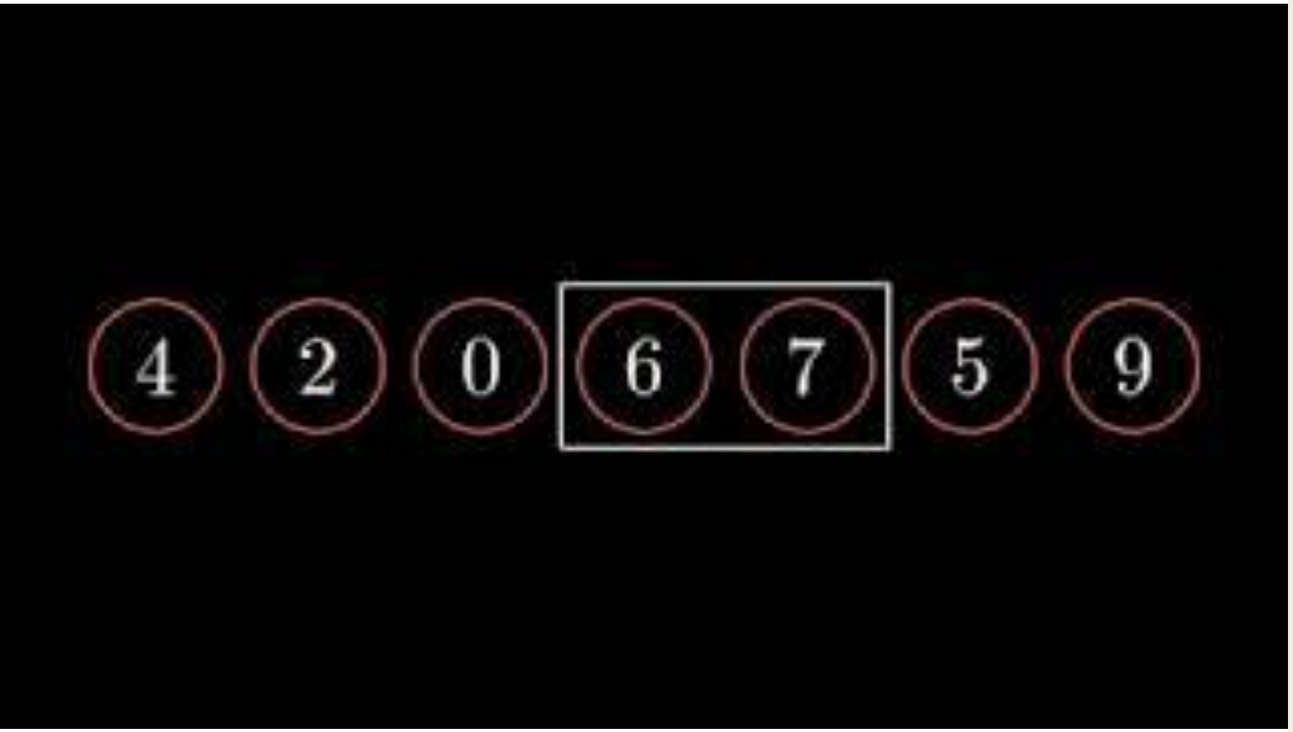
# Сортировка пузырьком

Задача сортировки массива:

- На вход подается массив
- Не выходе - массив необходимо упорядочить по признаку
- Элементы массив и признак для упорядочивания - могут быть разными
- Как правило:
  - элементы массив - числа
  - признак - порядок на числах


Задача сортировки может быть решена различными алгоритмами

Один из них - сортировка пузырьком.




```
1 for i in range(n):
2     for j in range(n - i - 1):
3         if arr[j] > arr[j + 1]:
4             swap(arr[j], arr[j + 1])
```

# Сортировка пузырьком

Разберем псевдокод этого алгоритма:

- line 1: Первый цикл нужен для того, чтобы проходов по массиву было столько, сколько элементов в массиве
- line 2: Второй цикл нужно, чтобы обеспечить собственно каждый проход по элементам массива
- line 3: Условный оператор (ветвление), который проверяет является ли элемент слева большем чем элемент справа
- line 4: Если да, меняем их местами

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Меняем элементы местами
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Example usage
my_array = [64, 25, 12, 22, 11]
sorted_array = bubble_sort(my_array)
print(sorted_array)
```

# Сортировка пузырьком

Разберем псевдокод этого алгоритма:

- line 1: Первый цикл нужен для того, чтобы проходов по массиву было столько, сколько элементов в массиве
- line 2: Второй цикл нужно, чтобы обеспечить собственно каждый проход по элементам массива
- line 3: Условный оператор (ветвление), который проверяет является ли элемент слева большем чем элемент справа
- line 4: Если да, меняем их местами

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Меняем элементы местами
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Example usage
my_array = [64, 25, 12, 22, 11]
sorted_array = bubble_sort(my_array)
print(sorted_array)
```

# Сортировка пузырьком

Какие парадигмы здесь используются?

- Императивная?
- Структурная?
- Процедурная?

```python
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                # Меняем элементы местами
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

# Example usage
my_array = [64, 25, 12, 22, 11]
sorted_array = bubble_sort(my_array)
print(sorted_array)
```

# Сортировка пузырьком

Какие парадигмы здесь используются?

- Императивная?
    - Да. Реализация сортировки написана пошагово, на уровне манипуляций с конкретными переменными.
- Структурная?
    - Да. Весь алгоритм реализован с помощью трех управляющих конструкций и без goto.
- Процедурная?
    - Да. Сортировка массива оформлена в виде функции.

# Преимущества и недостатки

# Структурная парадигма

1

**Преимущества**

- ➕ Легче искать: можно быстро найти нужную часть
- ➕ Легче поддерживать: изменение кода в одном блоке не повлияет на другие
- ➕ Легче тестировать: каждый блок можно протестировать отдельно
- ➕ Уменьшение вероятности ошибок: больше контроля при процессе разработки

2

**Недостатки**

- ➖ Возможная неэффективность: нет гарантий что структурная реализация оптимальна
- ➖ Нехватка абстракций: может приводить к дублированию кода
- ➖ Трудности при распараллеливании: структурный код будет распараллелить сложнее, чем процедурный
- ➖ Риск потери данных: все переменные скорее всего находятся в одной области видимости

# Процедурная парадигма

1

**Преимущества**

- ➕ Легче читать код: с помощью чтения “сверху - вниз”
- ➕ Многопоточность: исполнение процедуры легче распараллелить
- ➕ Повторное переиспользование кода: процедуры можно объявить один раз, а потом использовать сколько угодно
- ➕ Сообщество: быстрая взаимопомощь внутри сообщества программистов

2

**Недостатки**

- ➖ Разработка может длиться дольше
- ➖ Потеря данных внутри процедур и невоспроизводимость

# Признаки “выгодности”

1

**Процедурная**

\+ Когда необходимо решить задачу, которая может быть “удачно” разбита на более мелкие подзадачи

\+ Проект содержит части кода, используемые несколько раз

\+ Линейная структура проекта

— Большие объемы данных

— Сложная система или приложение не укладывающаяся в форму последовательности шагов и их результатов

2

**Структурная**

\+ Отдельные части кода не повторяются

\+ Стабильность и предсказуемость важнее скорости

\+ Необходимо, чтобы код был на ладони

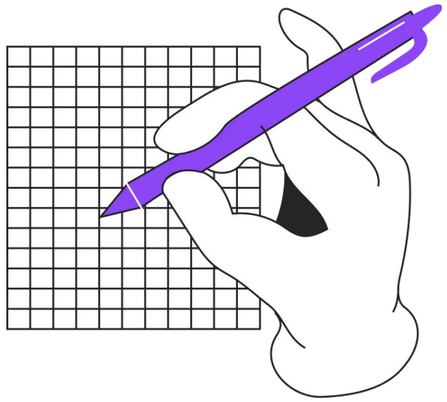
\+ Необходима быстрая разработка

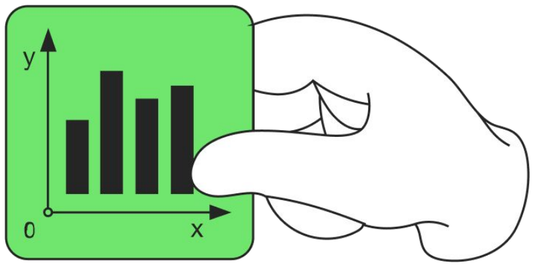
— Некоторые части планируется переиспользовать несколько раз

— Эффективность кода важнее читабельности и структуры

# Реальные кейсы

# Кейс №1. Построение дашбордов







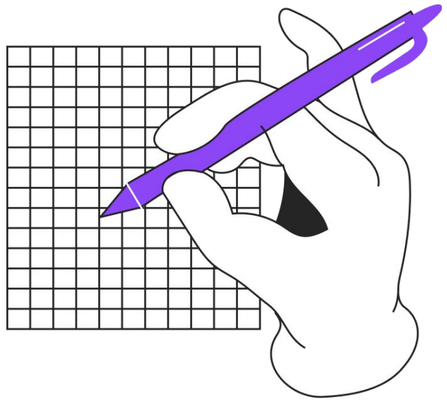
# Кейс №1. Построение дашбордов

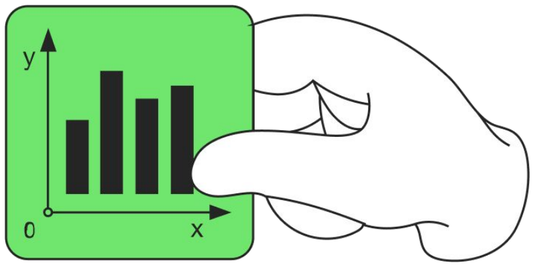
Вы программист и получили заказ от небольшой компании, в которой часть решений принимается на основе визуализации созданной в Excel-е. Процесс:

- Открытие таблицы в Excel
- Предобработка данных
- Построение графиков

Вам необходимо автоматизировать процесс.
Вопросы:

- Насколько важно применять **структурную парадигму**?
- Необходимо ли использовать **процедурное программирование**?







# Кейс №1. Построение дашбордов

Вам необходимо автоматизировать процесс.

Вопросы:

- Использовать **структурную парадигму**? - Скорее да:
  - Во-первых, у вас не будет доступа к goto
  - Во-вторых, читабельность важнее эффективности
- Использовать **процедурное парадигму**? - Возможно:
  - Собираюсь ли я переиспользовать части программы, которые сейчас пишу?
    - Да - используем процедурную
    - Нет - не стоит её использовать

# Кейс №2. Парсер валют

# Кейс №2. Парсер валют

Предположим, вас попросили написать программу, которая раз в 1 час находит и скачивает курс конвертации нескольких валют с сайта ЦБ.
Вопросы:

- Использовать структурную парадигму?
- Использовать процедурную парадигму?

Примерный псевдокод вашей программы:

```
Делать каждый час:
    Получить сайт ЦБ
    Для каждой из валют:
        Найти курс конвертации
        Сохранить этот курс
    Разорвать соединение
```

# Кейс №2. Парсер валют

- Использовать структурную парадигму?
  - Да. Не использовать структурную - нет смысла, goto тут точно не нужен

- Использовать процедурную парадигму?
  - Да. В данной программе будут составные шаги, которые удобно завернуть в процедуры и использовать каждый раз независимо.

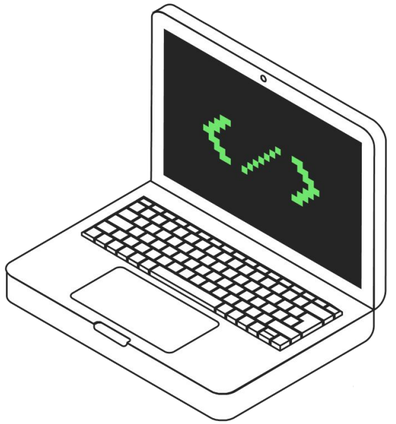
```
Делать каждый час:
    Получить сайт ЦБ
    Для каждой из валют:
        Найти курс конвертации
        Сохранить этот курс
    Разорвать соединение
```

# Кейс №3. Прототип

# Кейс №3. Прототип

Вас попросили реализовать алгоритм, который пока что неизвестно будет ли работать, и для того чтобы убедиться в этом, необходимо прогнать несколько тестов и получить прототип.

- Использовать процедурную парадигму?
    - Скорее нет. Вы будете в режиме быстрой разработки и не будете думать о дальнейшем использовании этого кода
- Использовать структурную парадигму?
    - Возможно. Зависит от вашего опыта оптимизации алгоритмов с помощью goto.



# Итоги лекции

# Подведение итогов

Сегодня мы подробно изучили **структурную** и **процедурную** парадигмы программирования.
Мы узнали:

- **Структурное программирование**
  С goto или без, история структурного программирования, примеры языков
- **Процедурное программирование**
  История процедурного программирования, примеры процедурных языков
- **Примеры кода**
  Пример с Countif, пример с сортировкой пузырьком
- **Преимущества и недостатки**
  Структурного и процедурного программирования
- **Реальные кейсы**
  Построение дашбордов, парсер валют, прототип

Если остались вопросы
Telegram: @alexlevinML

# Конец лекции
# Спасибо за внимание!