# Лабораторная работа 3.3. Работа с массивами данных. Вычислительная сложность алгоритмов

**Цель работы**: научиться понимать природу и разновидности ошибок, возникающих в программном обеспечении, а также исследовать подходы к их поиску, отладке и обработке. Освоить обработку исключений в Python и научиться писать надёжный код с минимизацией ошибок.

**Задачи работы**:

1. Изучить основные известные примеры ошибок в программном обеспечении и их последствия.

2. Разобраться в типах ошибок: синтаксических, логических и ошибок времени выполнения.

3. Освоить методы поиска ошибок и отладки программ.

4. Изучить подходы к обработке ошибок, включая использование конструкции **try-except** в Python.

5. Научиться возбуждать исключения вручную с использованием **raise** и работать с утверждениями (**assert**).

6. Научиться правильно использовать исключения и утверждения в коде, оценить их преимущества и недостатки.

7. Решить практические задачи по обработке ошибок и написанию кода с учётом возможных исключительных ситуаций.

**Примечание**

При выполнении заданий используйте заготовки решений: -> [Репозиторий](https://github.com/BosenkoTM/Python-Programming-A-101/tree/main/labs/lab_03-02).

<https://github.com/BosenkoTM/Python-Programming-A-101/tree/main/labs/lab_03-02>

* Изучить теоретические основы раздела[Вычислительная сложность алгоритмов](https://www.yuripetrov.ru/edu/python/ch_06.html)

<https://www.yuripetrov.ru/edu/python/ch_06_01.html>

**Теоретические основы**

**3.3.1** Известные ошибки в ПО

История программирования знает множество примеров, когда ошибки в коде привели к серьёзным последствиям:

**3.3.1.1** 1962 г.: ракета Маринер-1.

Ошибка в навигационном программном обеспечении ракеты NASA привела к отклонению от курса и её разрушению.

**3.3.1.2** 1985 г.: аппарат лучевой терапии Therac-25.

Ошибка в программировании аппарата привела к фатальным дозам облучения пациентов.

**3.3.1.3** 1991 г.: ЗРК Patriot.

Сбой таймера в ПО привёл к потере точности расчётов и катастрофическим последствиям в войне в Персидском заливе.

**3.3.1.4** 2000 г.: проблема 2000 года (Y2K).

Массовый сбой систем ожидался из-за неправильного представления года двумя цифрами. Хотя последствий удалось избежать, проблема показала важность правильной работы с данными.

**3.3.1.5** 2009-2011 гг.: отзыв автомобилей Toyota.

Ошибка в программном обеспечении управления педалью акселератора привела к сотням аварий.

**3.3.1.6** 2012 г.: проблема с Nasdaq во время IPO Facebook.

Ошибка в программном обеспечении Nasdaq, связанная с некорректной обработкой большого количества заявок на покупку акций, привела к задержке IPO Facebook. В результате трейдеры потеряли сотни миллионов долларов, и репутация биржи была подорвана.

**3.3.1.7** 2013 г.: сбой в системе Healthcare.gov.

После запуска веб-портала для программы медицинского страхования США система оказалась неспособной справиться с нагрузкой из-за множества ошибок в программировании. Это привело к миллионам недовольных пользователей и необходимости масштабной доработки сайта.

**3.3.1.8** 2016 г.: взлом голосового ассистента Alexa.

Из-за уязвимости в голосовом помощнике Amazon Alexa злоумышленники могли получить доступ к личной информации пользователей. Эта ошибка подняла вопросы безопасности интернет-устройств и алгоритмов искусственного интеллекта.

**3.3.1.9** 2018 г.: сбой в ПО Boeing 737 MAX.

Ошибка в системе управления полётом (MCAS) стала причиной двух авиакатастроф Boeing 737 MAX, унесших жизни 346 человек. Проблема заключалась в некорректной работе программного обеспечения, которое без предупреждения активировало управление самолётом. Это привело к глобальной приостановке эксплуатации Boeing 737 MAX.

**3.3.1.10** 2021 г.: сбой в облачных сервисах Facebook.

Из-за ошибки конфигурации в системах маршрутизации данные Facebook, Instagram и WhatsApp стали недоступны для миллиардов пользователей по всему миру. Это привело к остановке бизнес-процессов, зависящих от этих сервисов, и репутационным потерям компании.

**3.3.1.11** 2022 г.: Ошибка в криптовалютной сети Solana

Из-за ошибки в алгоритме обработки транзакций сеть Solana оказалась парализована более чем на 17 часов. Это привело к падению стоимости криптовалюты и подорвало доверие к платформе.

**3.3.1.12** 2023 г.: Уязвимость в системе искусственного интеллекта ChatGPT

В марте 2023 года была выявлена утечка данных из-за ошибки в работе кэширования. Это привело к доступу пользователей к истории чатов других людей, что вызвало вопросы к безопасности таких систем и их управления конфиденциальностью.

**3.3.1.13** 2024 г.: проблема в ПО беспилотного автомобиля Tesla.

Сбой в обновлении программного обеспечения автопилота Tesla привёл к тому, что машины некорректно реагировали на препятствия, что вызвало несколько мелких аварий. Это усилило давление регуляторов на производителей беспилотных технологий.

**3.3.1.14** 2025 г.: сбой в глобальной сети управления энергией.

Ошибка в распределённой системе управления энергией (умные сети) привела к масштабному отключению электроэнергии в нескольких странах. Основная причина — неверная обработка данных из-за переполнения буфера. Событие стало важным сигналом для переоценки безопасности инфраструктурного ПО.

**3.3.2** Определение и разновидности ошибок

**3.3.2.1** Синтаксические ошибки

Ошибки, связанные с нарушением правил синтаксиса языка программирования. Например:

print("Hello world) # Пропущена кавычка

**3.3.2.2** Логические (семантические) ошибки

Код выполняется, но результат не соответствует ожидаемому. Пример:

def add(a, b):

return a - b # Ошибка: вместо сложения происходит вычитание

**3.3.2.3** Ошибки времени выполнения

Ошибки, возникающие при выполнении программы, например деление на ноль:

x = 10 / 0 # ZeroDivisionError

**3.3.2.4** Недокументированное поведение

Поведение программы, не предусмотренное документацией. Может быть источником нестабильности.

**3.3.3** Поиск ошибок и отладка программы

* Ручная проверка кода.
* Использование отладчика. Например, встроенный отладчик в IDE.
* Добавление временных отладочных выводов. Использование **print()** для вывода переменных и этапов выполнения программы.
* Логирование. Применение модуля **logging** для отслеживания выполнения кода.

**3.3.4** Подходы к обработке ошибок

* Игнорирование ошибок. Приводит к нестабильной работе.
* Жёсткие проверки (**if-else**). Увеличивает сложность кода.
* Использование исключений. Гибкий подход, позволяющий отделить основной код от обработки ошибок.

**3.3.5** Обработка исключений в Python

**3.3.5.1** Понятия исключения

**Исключение (exception)** — это событие, которое возникает при выполнении программы и прерывает её нормальный ход.

**3.3.5.2** Конструкция **try-except**

Для обработки исключений используется блок **try-except**:

try:

x = 10 / 0

except ZeroDivisionError:

print("Деление на ноль недопустимо!")

**3.3.5.3** Вызов исключений (**raise**)

Исключения можно создавать вручную:

def divide(a, b):

if b == 0:

raise ValueError("Деление на ноль недопустимо!")

return a / b

**3.3.5.4** Особенности обработки исключений внутри функций

Исключения, возникающие в функции, можно перехватывать как внутри функции, так и за её пределами:

def divide(a, b):

return a / b

try:

result = divide(10, 0)

except ZeroDivisionError:

print("Ошибка деления на ноль!")

**3.3.5.5** Утверждения (**assert**)

Утверждения используются для проверки логических условий:

x = 5

assert x > 0, "x должно быть положительным"

3.3.5.6 Исключения или утверждения?

- Исключения используются для обработки ошибок, которые могут возникнуть в рабочей программе.

- Утверждения применяются для отладки и проверки внутренних условий разработки.

**Ход работы**

## [Тема Простая отладка программ](https://www.yuripetrov.ru/edu/python/ch_07_02.html#id6)

Для задач 3.3.1-3.3.5 приведено условие задачи и ее решение (ошибочное).

Для каждой задачи:

* используя визуальный просмотр кода, простое журналирование, проверку PEP8 или отладчик в IDE:
  + разберитесь в алгоритме решения;
  + найдите номера строк, в которых есть ошибки, и укажите их в строке документации;
* исправьте ошибки, минимально изменив код (как правило, заменив строчку на строчку); ошибки исправляются по ходу программы, сверху вниз.

Пример.

Исходная программа:

*"""*

*Ошибки (номера строк через пробел): !!!*

*"""*

**def** even\_mult(nums):

*"""Вернуть произведение четных чисел в списке 'nums'."""*

p = 0

**for** item **in** nums:

**if** item % 2 != 0:

p \*= item

**return** p

Исправленная программа:

*"""*

*Ошибки (номера строк через пробел): 8 10*

*"""*

**def** even\_mult(nums):

*"""Вернуть произведение четных чисел в списке 'nums'."""*

p = 1

**for** item **in** nums:

**if** item % 2 == 0:

p \*= item

**return** p

**№ 3.3.1 - № 3.3.5**

## [Тема Обработка исключений и утверждения](https://www.yuripetrov.ru/edu/python/ch_07_02.html#id7)[¶](https://www.yuripetrov.ru/edu/python/ch_07_02.html#id5)

Программы 3.3.6-3.3.11 написаны верно, однако содержат места потенциальных ошибок.

Для каждой задачи:

* найдите потенциальные источники ошибок (укажите номера строк в строке документации);
* используя конструкцию **[try](https://www.yuripetrov.ru/edu/python/ch_07_01.html" \l "id0" \o "try)** добавьте в код обработку соответствующих исключений.

Пример.

Исходная программа:

*"""*

*Ошибки (номера строк через пробел): !!!*

*"""*

**def** avg(a, b):

*"""Вернуть среднее геометрическое чисел 'a' и 'b'.*

*Параметры:*

*- a, b (int или float).*

*Результат:*

*- float.*

*"""*

**return** (a \* b) \*\* 0.5

a = float(input("a = "))

b = float(input("b = "))

c = avg(a, b)

print("Среднее геометрическое = **{:.2f}**".format(c))

Исправленная программа:

*"""*

*Ошибки (номера строк через пробел): 15 18 19*

*"""*

**def** avg(a, b):

*"""Вернуть среднее геометрическое чисел 'a' и 'b'.*

*Параметры:*

*- a, b (int или float).*

*Результат:*

*- float.*

*Исключения:*

*- ValueError: вычисление не возможно.*

*"""*

**if** a \* b >= 0:

**return** (a \* b) \*\* 0.5

**else**:

**raise** **ValueError**("Невозможно определить среднее геометрическое "

"введенных чисел.")

**try**:

a = float(input("a = "))

b = float(input("b = "))

c = avg(a, b)

print("Среднее геометрическое = **{:.2f}**".format(c))

**except** **ValueError** **as** err:

print("Ошибка:", err, ". Проверьте введенные числа.")

**except** **Exception** **as** err:

print("Ошибка:", err)

**№ 3.3.6 - № 3.3.11**