**1. Типы ошибок в коде**

**Синтаксическая ошибка**

**Определение:** Нарушение правил языка программирования, из-за которого код не может быть выполнен.  
**Пример:**

python

Copy

Download

if x == 5 *# Ошибка: пропущено двоеточие*

print("x равно 5")

**Логическая ошибка**

**Определение:** Код работает, но выдает неверный результат из-за неправильной логики.  
**Пример:**

python

Copy

Download

def is\_even(n):

return n % 2 != 0 *# Ошибка: должно быть `n % 2 == 0`*

**Ошибка в условии**

**Определение:** Неправильное условие в if, while или других управляющих конструкциях.  
**Пример:**

python

Copy

Download

if x > 10 and x < 5: *# Условие никогда не выполнится*

print("x в диапазоне")

**Ошибка обработки граничных случаев**

**Определение:** Код не учитывает крайние значения (например, пустой список, ноль, максимальное число).  
**Пример:**

python

Copy

Download

def divide(a, b):

return a / b *# Ошибка: не обрабатывается `b = 0`*

**Отклонение от алгоритма**

**Определение:** Код делает не то, что описано в требованиях.  
**Пример:**

python

Copy

Download

*# Алгоритм требует бинарного поиска, но используется линейный*

def find\_square(n):

for i in range(n): *# Должен быть бинарный поиск*

if i \* i == n:

return True

return False

**2. Методы поиска ошибок**

1. **Визуальный анализ кода** – проверка кода на соответствие алгоритму.
2. **Сравнение с описанием алгоритма** – пошаговая сверка логики.
3. **Ручное тестирование** – запуск функции на разных входных данных.

**Практическая часть: Анализ кода is\_perfect\_square(n)**

**1. Визуальный анализ и сравнение с алгоритмом**

(Предположим, что код студента выглядит так:)

python

Copy

Download

def is\_perfect\_square(n):

if n < 0:

return False

if n == 0 or n == 1:

return True

left, right = 2, n // 2

while left <= right:

mid = (left + right) // 2

if mid \* mid == n:

return True

elif mid \* mid < n:

left = mid + 1

else:

right = mid - 1

return False

**Найденные ошибки:**

* **Строка 5 (right = n // 2)** – **Ошибка граничного случая**.
  + **Почему:** Для n = 2 или n = 3 диапазон [2, n//2] становится [2, 1], и цикл не выполняется, хотя должен.
  + **Исправление:** right = n (или n // 2 + 1).

**2. Мнимые "ошибки"**

* **Строка 6 (while left <= right)** – выглядит подозрительно (может показаться, что < достаточно), но корректно, так как mid должен проверяться при left == right.

**3. Ручное тестирование**

| **Вход (n)** | **Ожидаемый** | **Фактический** | **Вывод** |
| --- | --- | --- | --- |
| -1 | False | False | ✅ |
| 0 | True | True | ✅ |
| 1 | True | True | ✅ |
| 2 | False | True\* | ❌ (Ошибка) |
| 4 | True | True | ✅ |
| 15 | False | False | ✅ |
| 16 | True | True | ✅ |
| 1000000 | True | True | ✅ |

**Выявленная ошибка:**

* При n = 2 возвращает True, хотя должно быть False.

**4. Исправленный код**

python

Copy

Download

def is\_perfect\_square(n):

if n < 0:

return False

if n == 0 or n == 1:

return True

left, right = 2, n *# Исправлено: right = n вместо n // 2*

while left <= right:

mid = (left + right) // 2

if mid \* mid == n:

return True

elif mid \* mid < n:

left = mid + 1

else:

right = mid - 1

return False

**Ключевые изменения:**

* Исправлен диапазон бинарного поиска (right = n вместо n // 2).

**5. Анализ и выводы**

* **Наиболее серьезная ошибка:** Неправильный диапазон бинарного поиска, из-за которого функция давала ложные срабатывания.
* **Чаще всего встречалась:** Ошибка граничных случаев.
* **Самый эффективный тест:** n = 2 (выявил проблему с диапазоном).

**Итог:**

* Корректная обработка граничных значений критически важна.
* Бинарный поиск требует точного задания диапазона.
* Тестирование на малых числах (n=2,3) помогает находить ошибки быстрее.