Типы — залог здоровья кода

Демяненко Я.М.

Южный федеральный университет,
Институт математики, механики и компьютерных наук

demyana@sfedu.ru

Доклад на пятой Всероссийской научно-методической конференции
«Использование системы программирования PascalABC.NET
в обучении программированию»
(Ростов-на-Дону, 28-29 марта 2025 г.)

Что понимается под здоровьем кода (Code Health)

нет строгого определения

- поддерживаемость
- читаемость
- стабильность
- простота кода

- простота
- удобочитаемость
- тестируемость
- строительная способность

Нет синтаксических ошибок
Нет семантических ошибок
Результат корректен, предсказуем и стабилен

Типизация языков

Языки программирования по типизации принято делить на два больших лагеря:

типизированные
C/C++
C#
JScript
PascalABC.Net
PHP
Python
Scala
…

нетипизированные (бестиповые)
Ассемблер
Forth
Brainfuck
…

Категории типизированных языков

Статическая типизация

При **статической типизации** переменная связывается с типом **в момент объявления** переменной

Статическая типизация означает, что типы переменных определяются **на этапе компиляции**

Динамическая типизация

При динамической типизации переменная связывается с типом **в момент присваивания** значения

Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов

Динамическая типизация означает, что типы определяются **во время выполнения**

Языки статической / динамической типизации

Статическая:

- **C/C++**
- **C#**
- **PascalABC.Net**
- **Java**
- Fortran
- Haskell
- TypeScript
- Flow
- Standard ML
- Elm
- Idris
- Agda
- Scala, Ocaml
- F#
- Rust
- Visual Prolog
- …

Динамическая:

- Smalltalk
- **Python**
- Objective-C
- Ruby
- PHP
- Perl
- **JavaScript**
- Лисп

Некоторые статически типизированные языки позже получили возможность также использовать динамическую типизацию при помощи специальных подсистем

Сильная / слабая типизация (строгая / нестрогая)

Сильная типизация выделяется тем, что язык **не позволяет смешивать** в выражениях различные типы и **не выполняет автоматические неявные преобразования**, например нельзя вычесть из строки множество.

Языки со **слабой типизацией выполняют** множество **неявных преобразований** автоматически, даже если может произойти потеря точности или преобразование неоднозначно.

Сильная
Java
Python
Haskell
Lisp

Слабая
C/C++
PascalABC.Net
JavaScript
Visual Basic
PHP

Преимущества статической типизации

Статическая типизация даёт **самый простой машинный код**, поэтому она удобна для языков, дающих **исполняемые файлы** операционных систем или JIT-компилируемые промежуточные коды

Многие **ошибки исключаются** уже **на стадии компиляции**, поэтому статическая типизация **хороша для** написания **сложного, но быстрого кода**

А еще это очень хорошо для школьников

Преимущества динамической типизации

Динамическая типизация предлагает **гибкость** и **удобство для быстрого прототипирования** и разработки

Динамическая типизация позволяет **уменьшить сложность кода**

Вряд ли для школьников актуально прототипирование на этапе обучения

Недостатки — продолжение достоинств (статическая)

предлагает большую **безопасность на этапе компиляции**, предотвращая многие типы ошибок, которые могут быть не очевидны в динамически типизированных языках до момента выполнения программы

обеспечивает **строгую проверку типов** и может способствовать созданию более **надежного** и **производительного кода**

сэкономит время на отладку и тестирование, особенно в больших и сложных проектах

анализаторы подсказывают о возможных ошибках

Для школьников это хорошо

некоторое увеличение объёма кода (необходимо явно указывать типы переменных)

Школьники лучше освоят типы

требует время на этап компиляции и сборки программы перед её запуском

Школьники еще не пишут большие программы

строгие требования к типам могут **ограничивать** возможности для **динамического изменения** поведения программы

усложнение работы **со слабоструктурированными** данными, например, с данными в формате JSON, структура которых может меняться

Недостатки — продолжение достоинств (динамическая)

позволяет переменным "переодеваться" в разные типы данных на лету, во время выполнения программы. Это делает **код более гибким**

считается, позволяет **уменьшить сложность** кода

предлагает гибкость и удобство для **быстрого прототипирования** и разработки

позволяет разработчикам **сосредоточиться** на **логике**, а не на строгом определении типов данных

требует от программиста **большей внимательности**, чтобы избежать ошибок

Для школьников это плохо

возрастает риск появления **ошибок**, связанных с неправильным типом данных

Для школьников это плохо

вредит производительности программ, т. к. отсутствие информации о типе **не позволяет** компилятору **эффективно оптимизировать** код

средам разработки становится **сложнее анализировать** то, как код будет исполняться, заранее предупреждать о возможных ошибках

Для школьников это плохо

Рассмотрим «взрослый» пример

Представьте, что вы пишете программу для учета товаров на складе. В один момент вам нужно обработать данные о количестве товаров, которые могут быть представлены как в виде чисел, так и в виде текста, например, "десять" или "10".

В языках с динамической типизацией, таких как Python, вам не нужно заранее указывать, что переменная будет только числом или только текстом.

Это позволяет с легкостью обрабатывать разные типы данных без дополнительного кода для проверки и преобразования типов.

Реализация на Python

```
def count_items(items):
    total = 0
    for item in items:
        if isinstance(item, str):
            if item.isdigit():
                total += int(item)  # Преобразуем текст в число, если это возможно
            else:
                print(f"Не могу преобразовать '{item}' в число!")
        else:
            total += item
    return total

# Список товаров может содержать как числа, так и текст
items = [5, "10", "десять", 7, "3"]
print(f"Общее количество товаров: {count_items(items)}")
```

```
Не могу преобразовать  десять  в число! Общее количество товаров:  25
```

Реализация на PascalABC.Net

```
function count_items(items: array of Objects):integer;
begin
  result := 0;
  foreach var item in items do
    if item is integer then
      result += item as integer
    else
    try
      result += item.tointeger;//Преобразуем текст в число, если это возможно
    except
      print('Не могу преобразовать ',item,' в число!');
    end;
end;
begin
  //var items := [5, '10', 'десять', '7', '3'];
  var items:=readlines('t.txt');
  print('Общее количество товаров: ',count_items(items))
end.
```

```
Не могу преобразовать  десять  в число! Общее количество товаров:  25
```

Система RTTI (Runtime Type Identification)

Механизм динамической идентификации типа данных в языках со статической типизацией
Впервые появился в языке C++

RTTI основана на способности системы сообщать о динамическом типе объекта и предоставлять информацию об этом типе во время выполнения (в отличие от времени компиляции)

Начинать в школе с таких примеров не рекомендуется

Выбор между динамической и статической типизацией

Зависит от множества факторов, включая
- специфику проекта
- требования к безопасности
- требования к производительности
- предпочтения команды разработчиков

Динамическая типизация предоставляет гибкость и удобство для быстрого прототипирования и разработки

Статическая типизация обеспечивает строгую проверку типов и может способствовать созданию более надежного и производительного кода

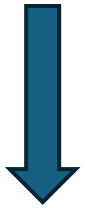
Школьнику нужна строгая проверка типов

Переход между языками у обучающихся

Студенты ИММиКН ЮФУ

ФИИТ

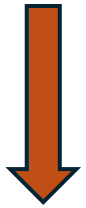
Первый курс: PascalABC.Net



Второй курс: C++

ПМИ

Первый курс: Python



Второй курс: C++

Сложности с восприятием типов

Чем грозит школьникам старт с динамической типизации

Динамическая типизация позволяет переменным "переодеваться" в разные типы данных на лету, во время выполнения программы. Это делает код более гибким

Динамическая типизация требует от программиста бОльшей внимательности, чтобы избежать ошибок

Школьники **Внимание**

Внимание **Школьников**

Школьники стремятся решать задачи побыстрее
Отсюда повышенная невнимательность

Типы есть … Но

Но **динамически** типизированные языки не только **не требуют указывать тип** переменных, но и **не определяют** его сами **до** момента **получения** конкретных **значений при запуске**.

Например, функция в Python

```
def f(x, y):
    return x + y
```

может складывать
- два целых числа
- склеивать строки
- склеивать списки
- и т. д.

А что именно функция делает мы сможем понять, когда запустим программу.

Поэтому иногда говорят, что значения в динамических языках обладают типом, но переменные и функции — нет.

Неожиданные результаты

Python относится к языкам с сильной типизацией

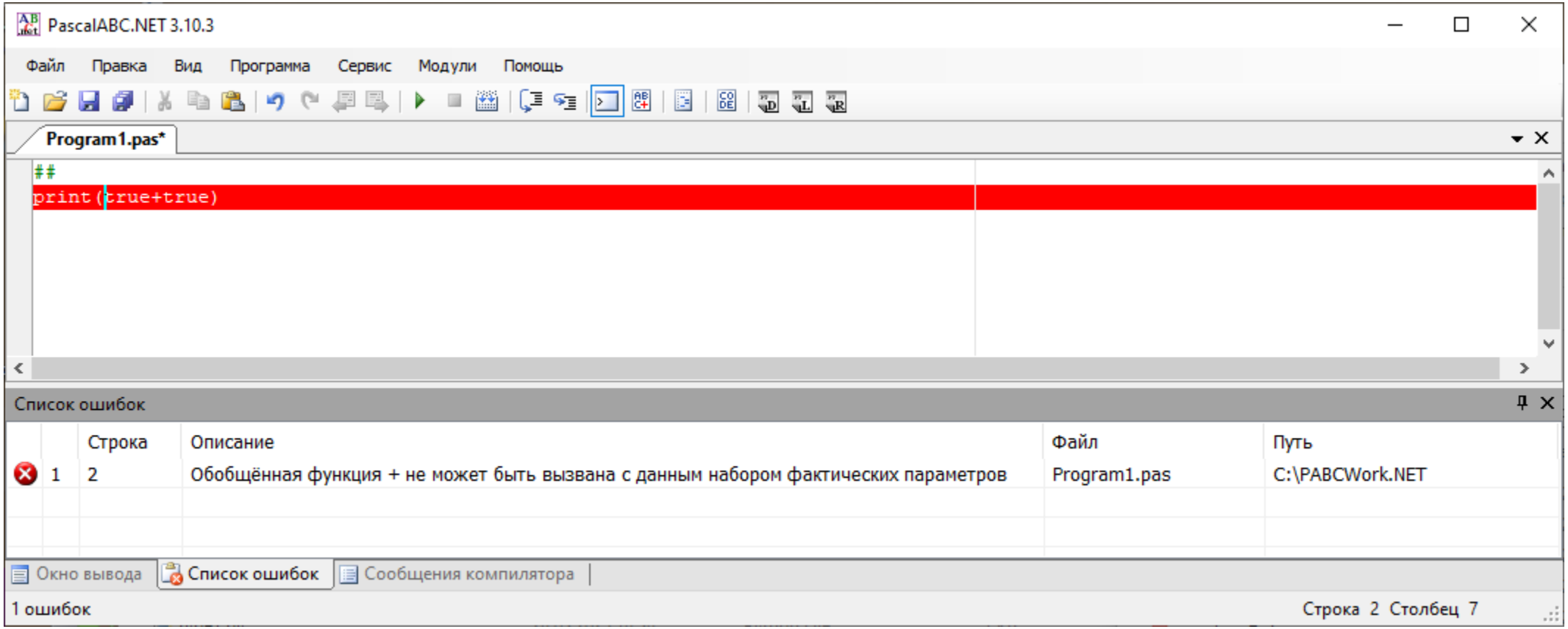
Однако результат сложения двух boolean

print(True+True)

```
====== RESTART: E:/Конфереции и публикации/PABC.NET/PABC.NET 2025/bool.py ======
2
>>>
```

Не самый лучший пример для школьников

В PascalABC.Net над boolean операция плюс не определена



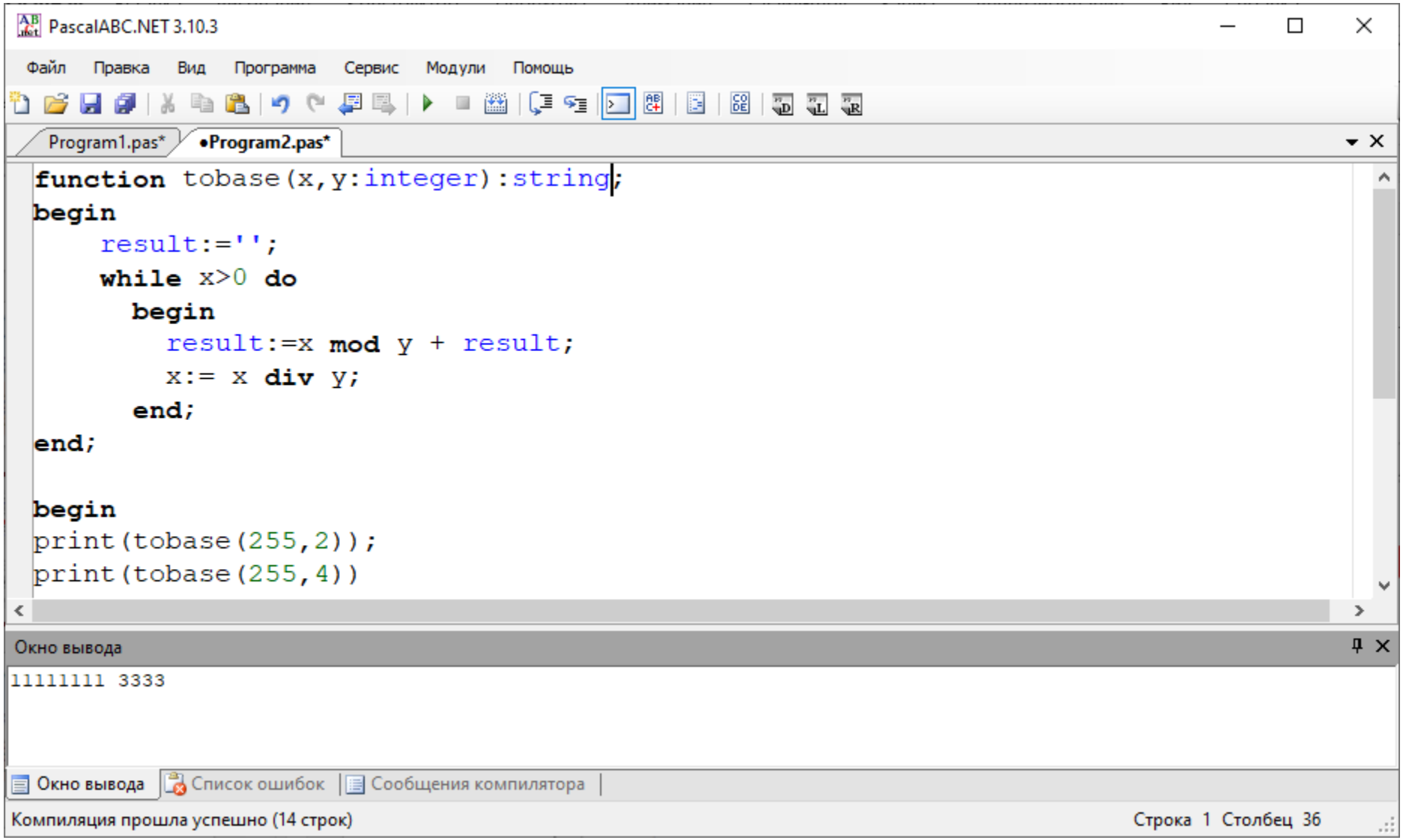
```
def tobase(x,y):
   s=''
   while x>0:
      s=x%y+s
      x//=y
   return s

print(tobase(255,2))
print(tobase(255,4))
```

Легко допустить ошибку

Язык с сильной динамической типизацией **не поддерживает неявное преобразование** типов

```
Traceback (most recent call last):
  File "E:/Конфереции и публикации/PABC.NET/PABC.NET 2025/type1.py", line 8, in <module>
    print(tobase(255,2))
  File "E:/Конфереции и публикации/PABC.NET/PABC.NET 2025/type1.py", line 4, in tobase
    s=x%y+s
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>>
```

Все ок


PascalABC.NET 3.10.3
Файл Правка Вид Программа Сервис Модули Помощь
Program1.pas*
•Program2.pas*
function tobase(x,y:integer):string;
begin
result:='';
while x>0 do
begin
result:=x mod y + result;
x:= x div y;
end;
end;
begin
print(tobase(255,2));
print(tobase(255,4))
Окно вывода
11111111 3333
Окно вывода
Список ошибок
Сообщения компилятора
Компиляция прошла успешно (14 строк)
Строка 1 Столбец 36

Неожиданная ошибка

```
def tobase(x,y):
   s=''
   while x>0:
      s=str(x%y)+s
      x//=y
   return s

print(tobase(255,2))
print(tobase(255,4))
print(tobase(255,'4'))
```

Функцию отладили. Отработала дважды корректно.

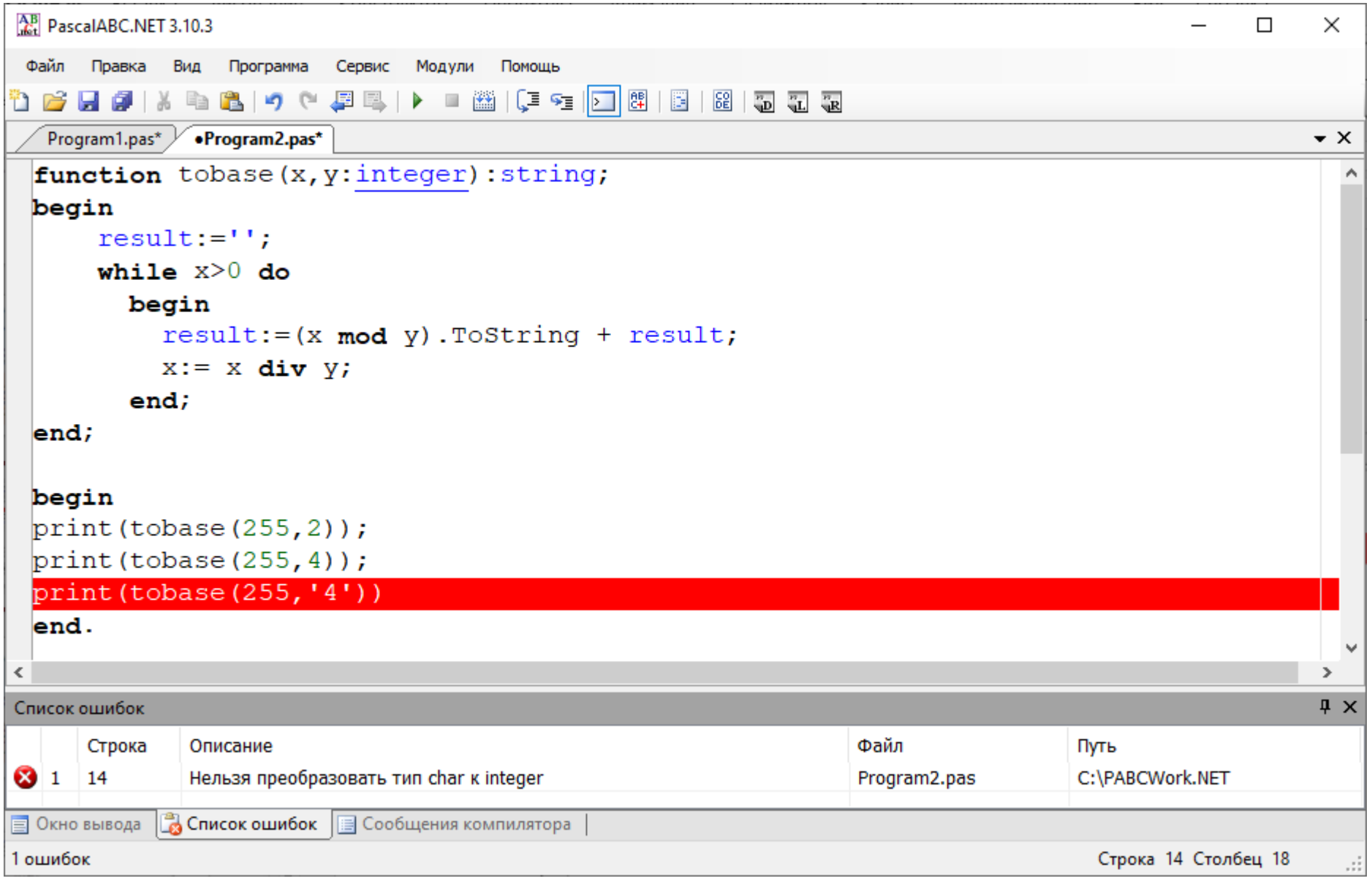
Отлаженную и работающую **функцию** вызвали с **некорректными данными**

Причина. Язык с сильной динамической типизацией проверяет типы уже при выполнении

Для школьников это плохо

```
11111111
3333
Traceback (most recent call last):
  File "E:/Конференции и публикации/PABC.NET/PABC.NET 2025/type1.py", line 10, in <module>
    print(tobase(255,'4'))
  File "E:/Конференции и публикации/PABC.NET/PABC.NET 2025/type1.py", line 4, in tobase
    s=str(x%y)+s
TypeError: unsupported operand type(s) for %: 'int' and 'str'
>>>
```

Обнаружили ошибку на этапе компиляции



Решение, который я часто вижу у школьников

Даны два числа в восьмеричной системе счисления. Найти их сумму.

```
n=15
m=25

r=oct(n)[2:]
p=oct(m)[2:]

print(r+p)
print(oct(n+m)[2:])
```

```
## uses school;

var n:=15bi;
var m:=25bi;

var r:=n.tobase(8);
var p:=m.tobase(8);

println(r+p);
println((n+m).tobase(8))
```

Какой результат верный ?

1731 VS 50

Что же надеялись получить?

Прошу назвать типы переменных и значений

Плохо менять тип переменной в программе

```
a=15
a=oct(a)[2:]
print(a)
b=5
b=oct(b)[2:]
print(b)
print(a>b)
```

Проверить для двух чисел в восьмеричной системе счисления, что первое больше второго

```
====== RESTART: E:/Конфереции и публикации/PABC.NET/PABC.NET 2025/type3.py =====
17
5
False
>>>
```

Смена типов у переменных незаметно привела к странному результату 15 в восьмеричной системе не больше 5 в восьмеричной

Не меняйте типы у переменных

Одна функция для разных типов

```
def find(required_element, sequence):
    """Выполняет поиск элемента в последовательности."""
    for index, element in enumerate(sequence):
        if element == required_element:
            return index
    return - 1


print(find(2, [1, 2, 3]))  # Выведет: 1
print(find("c", ("a", "b", "c", "d")))  # Выведет: 2
```

Кажется, что в Python достаточно одной функции для разных типов

Типичная ошибка (невнимательно отнеслись к типам)

```
def find(required_element, sequence):
    """Выполняет поиск элемента в последовательности."""
    for index, element in enumerate(sequence):
        if element == required_element:
            return index
    return - 1


print(find(2, [1, 2, 3]))  # Выведет: 1
print(find("c", ("a", "b", "c", "d")))  # Выведет: 2

print(find(1, 1))  # Вызывает исключение TypeError
```

Недостатком динамической типизации является то, что она может привести к неожиданным ошибкам во время выполнения.

Например, **если** мы передадим **не итерируемый объект sequence** (который нельзя обойти с помощью for), мы получим ошибку.

Но мы узнаем об этом, когда выполнение программы дойдет до определенной строки с циклом for по этому объекту.

Типичные ошибки школьников

Что делать?

Прививать уважение к типам

Как?

Мои рекомендации для преподавателей при обучении программированию

- Обсуждать типы данных в задаче
- Обсуждать в явном виде типы используемых переменных
- Стараться избегать использования одной и той же переменной для значений разных типов