Язык формирует наш способ мышления и определяет, о чем мы можем мыслить
— *Б.Л. Ворф*

Сравнение Python и PascalABC.NET

Михалкович С.С.
Южный федеральный университет,
Институт математики, механики и компьютерных наук
miks@sfedu.ru

Доклад на третьей Всероссийской научно-методической конференции
«Использование системы программирования PascalABC.NET
в обучении программированию»
(Ростов-на-Дону, 12-13 ноября 2021 г.)

Общая характеристика

Python

- Python возник в 1991 г.
- Среды программирования: IDLE (встроенная), WinG IDE, PyCharm, VS Code. Только PyCharm и VS Code обеспечивают неплохие подсказки по коду
- Высокоуровневость, краткость и понятность кода
- Использование в разработке ПО (**первый** язык в промышленном рейтинге TIOBE 2021)
- Интерпретатор, низкая скорость выполнения программ
- Динамическая типизация: пропускает многие ошибки, которые не пропускает компилятор
- Кроссплатформенный
- Распространенные библиотеки в области машинного обучения, нейронных сетей, больших данных и их визуализации

PascalABC.NET

- PascalABC.NET возник в 2007 г., активное развитие – с 2015 г.
- Среда программирования – PascalABC.NET. Автоформатирование, подсказки по коду
- Русскоязычный интерфейс и сообщения об ошибках
- Краткость и понятность кода, возможность писать высокоуровневый и низкоуровневый код
- Содержит языковые концепции ряда современных языков программирования
- Опирается на платформу .NET, полностью совместим на уровне промежуточного языка с языком C# (пятый язык в рейтинге TIOBE 2021)
- Компилятор кроссплатформенный, IDE только под Windows. Под Linux надо использовать внешние менее удобные IDE

Использование в обучении

Python

- Использование в обучении в школах и университетах США, Европы и России
- Некоторые концепции неудобны для обучения: динамическая типизация (спорно), ошибки на этапе выполнения, а не компиляции (начинающие допускают больше скрытых ошибок в программах)
- Нерусифицированная среда IDLE, нерусифицированные сообщения об ошибках
- Хороший выбор для ЕГЭ по информатике
- Хороший выбор для использования в олимпиадах по программированию (первый или второй язык)

PascalABC.NET

- Использование в обучении в школах и университетах России
- Компилятор отлавливает большинство ошибок до запуска программы, что хорошо для начинающих
- Русскоязычный интерфейс и сообщения об ошибках
- Хороший выбор для ЕГЭ по информатике
- Хороший выбор для использования в олимпиадах по программированию (третий язык)

Как сравниваются языки Python и PascalABC.NET?

На PascalABC.NET приводится код в стиле Python, хотя на PascalABC.NET можно эффективно писать, используя различные стили

Болтовня ничего не стоит. Покажите мне код.

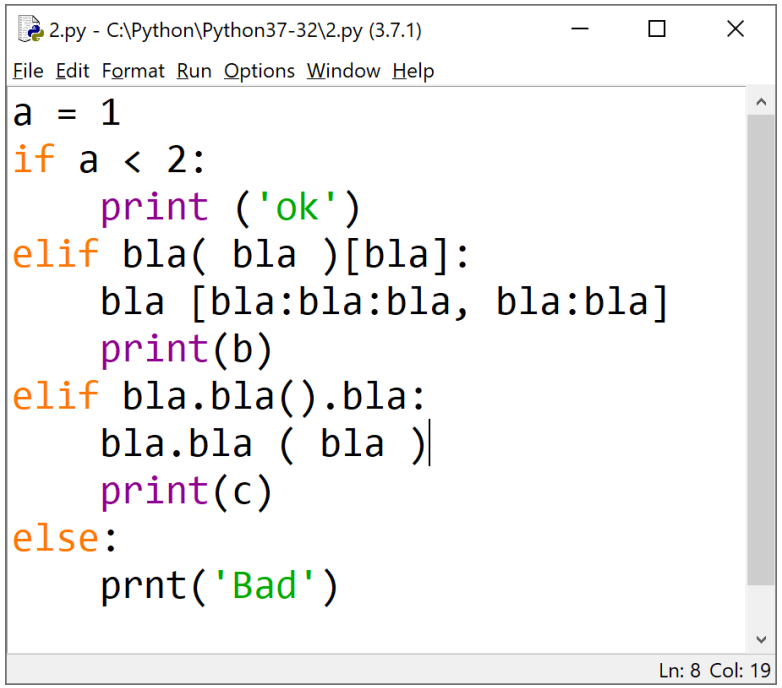
— *Linus Torvalds*

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)



PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

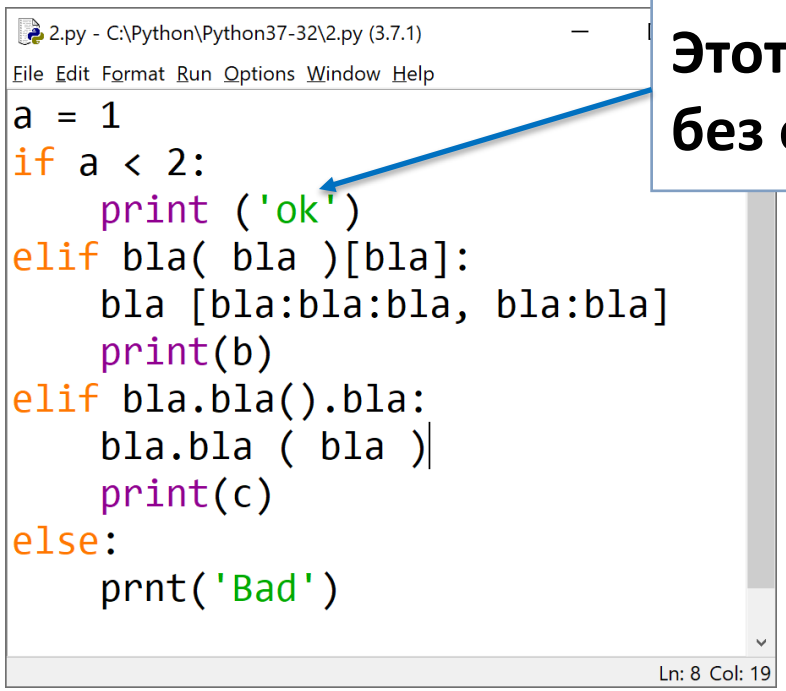
```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)



Этот код сработает без ошибок ☹

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

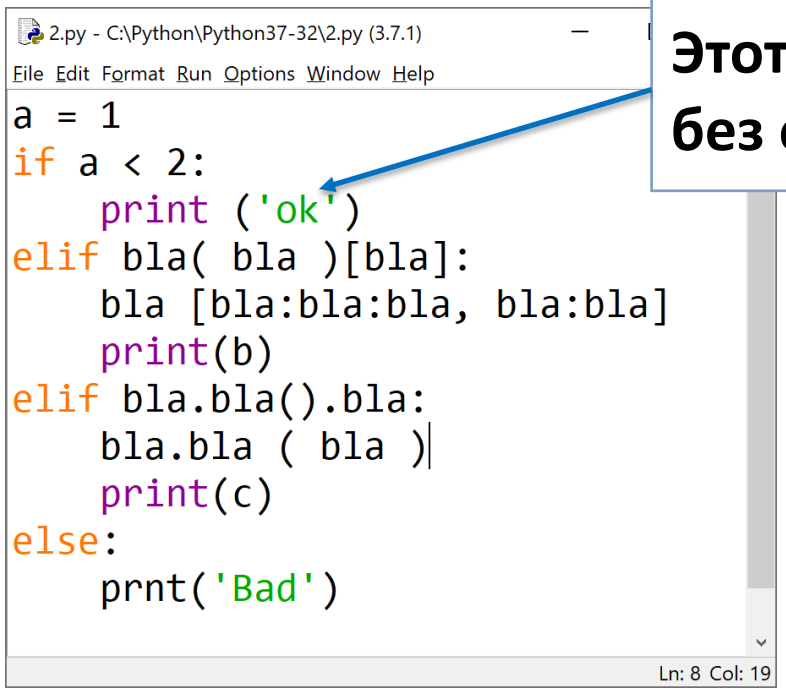
```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)



Этот код сработает без ошибок ☹

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

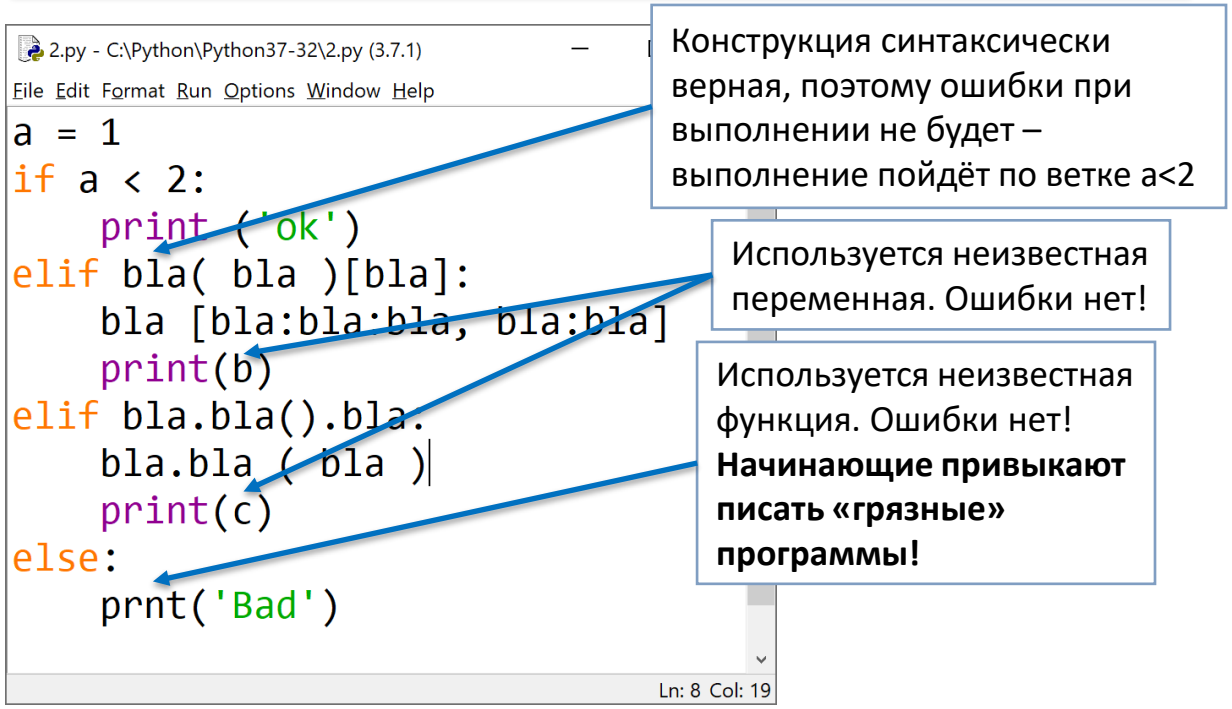
Этот код компилятор не пропустит из-за многочисленных ошибок ☺

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)



Конструкция синтаксически верная, поэтому ошибки при выполнении не будет – выполнение пойдёт по ветке a<2

Используется неизвестная переменная. Ошибки нет!

Используется неизвестная функция. Ошибки нет! **Начинающие привыкают писать «грязные» программы!**

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

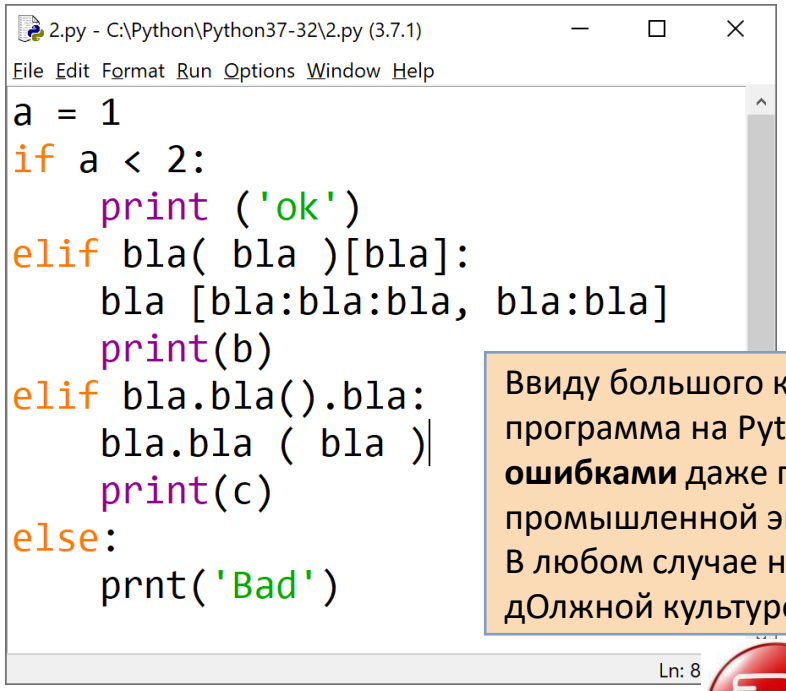
```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)



Ввиду большого количества веток программа на Python может остаться **с ошибками** даже после тестирования (при промышленной эксплуатации программы!). В любом случае начинающие не обладают дОлжной культурой тестирования

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)

2.py - C:\Python\Python37-32\2.py (3.7.1)

File Edit Format Run Options Window Help

```
a = 1
if a < 2:
    print ('ok')
elif bla( bla )[bla]:
    bla [bla:bla:bla, bla:bla]
    print(b)
elif bla.bla().bla:
    bla.bla ( bla )
    print(c)
else:
    prnt('Bad')
```

Ln: 8

Ввиду большого количества веток программа на Python может остаться с ошибками даже после тестирования (при промышленной эксплуатации программы!). В любом случае начинающие не обладают должной культурой тестирования

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

Компилятор выведет ошибку на первом же bla:
Неизвестное имя 'bla'

Диагностика ошибок

Python

Перед запуском проверяется соответствие грамматике.

Контроль области видимости переменных и контроль типов осуществляются на этапе выполнения (для обучения **это плохо** – лучше чтобы как можно больше ошибок отлавливались автоматически)

2.py - C:\Python\Python37-32\2.py (3.7.1)

File Edit Format Run Options Window Help

```
a = 1
if a < 2:
    print ('ok')
elif bla( bla )[bla]:
    bla [bla:bla:bla, bla:bla]
    print(b)
elif bla.bla().bla:
    bla.bla ( bla )
    print(c)
else:
    prnt('Bad')
```

Ln: 8

Ввиду большого количества веток программа на Python может остаться с ошибками даже после тестирования (при промышленной эксплуатации программы!). В этом – **огромный недостаток интерпретаторов** (**плохо для обучения**)

PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

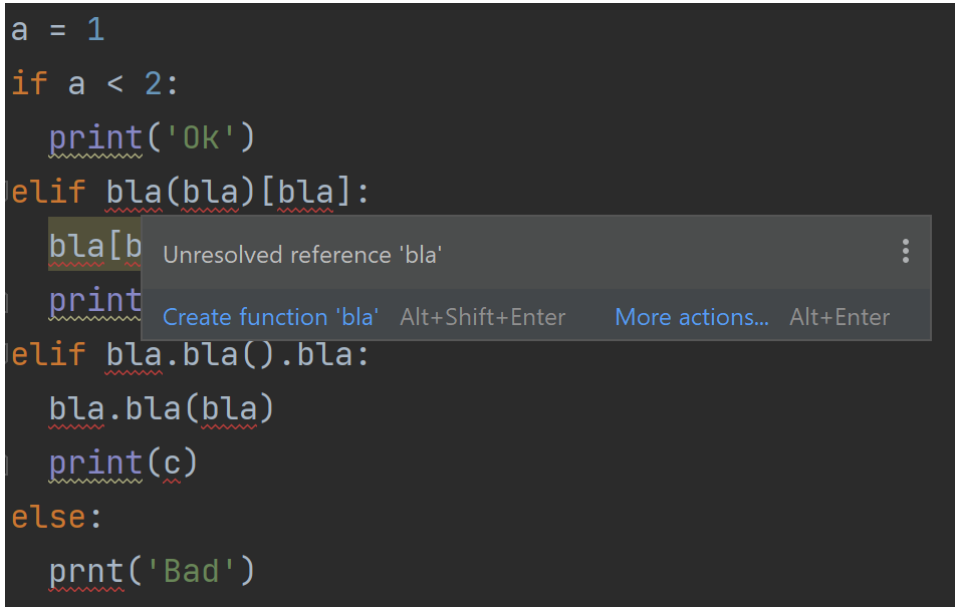
Компилятор будет выводить ошибки пока вы не исправите последнюю – только после этого программа будет запущена

В этом – **огромное преимущество компиляторов** как для обучения, так и для промышленного использования!

Диагностика ошибок

Python

Современные Python-среды, такие как PyCharm, содержат статический анализатор кода, позволяющий подчёркивать часть таких ошибок. Однако диагностика в таких анализаторах неполная и всецело зависит от среды программирования (в PyCharm предупреждения одни, в VS Code - другие). Так, PyCharm **почти всегда** подчеркивает не определенные ранее переменные:



PascalABC.NET

Перед запуском компилятор проверяет соответствие грамматике, осуществляет контроль области видимости переменных и контроль типов.

Множество ошибок компиляции препятствует написанию неправильных программ на раннем этапе и приучает к строгой культуре кода (**это хорошо**)

```
##
var a := 1;
if a < 2 then
  Print('Ok')
else if bla(bla)[bla] then begin
  bla[bla:bla:bla, bla:bla];
  Print(b)
end
else if bla.bla().bla then
begin
  bla.bla(bla);
  Print(c)
end
else Prnt('Bad')
```

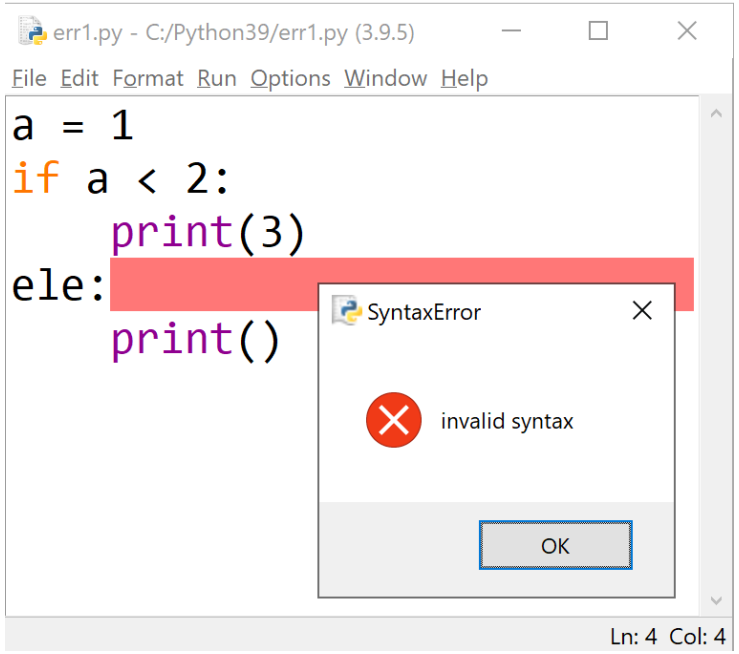
Компилятор будет выводить ошибки пока вы не исправите последнюю – только после этого программа будет запущена

В этом – **огромное преимущество компиляторов** как для обучения, так и для промышленного использования!

Сообщения об ошибках

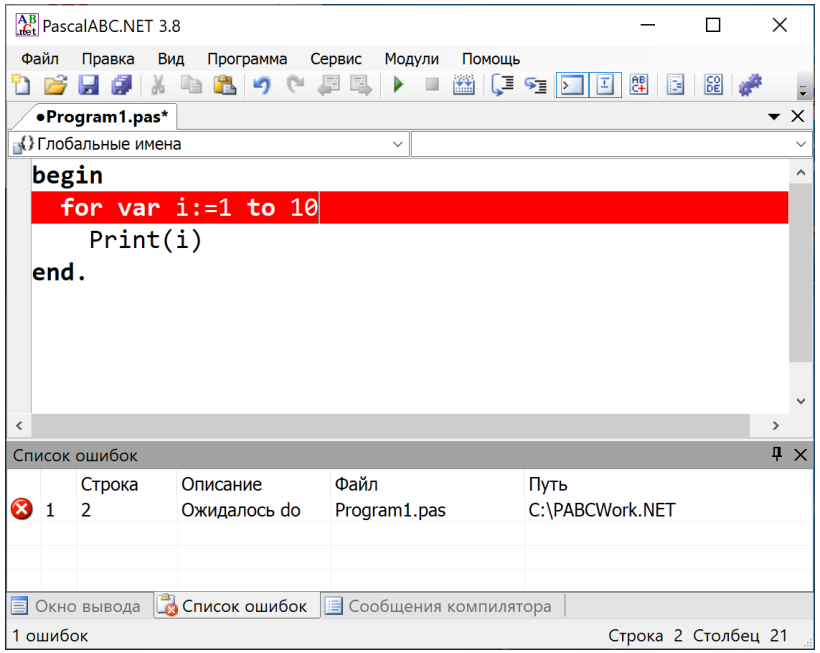
Python

Сообщения о синтаксических ошибках в Python – абсолютно **неинформативные**. В основном ошибка одна и та же – «**invalid syntax**» («неверный синтаксис» или «что-то не так» :)



PascalABC.NET

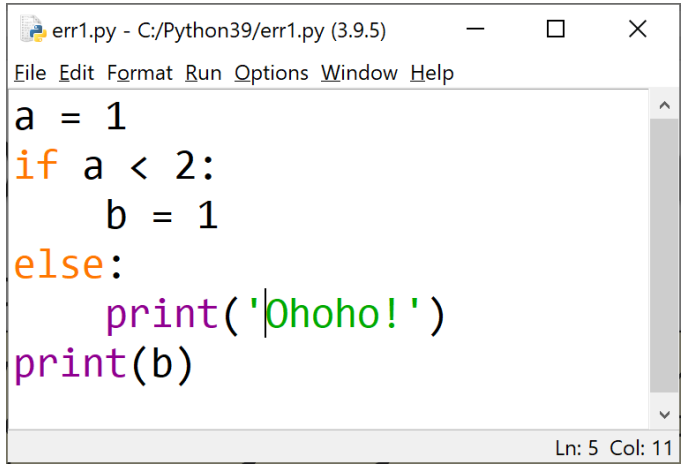
Сообщения об ошибках в PascalABC.NET – русифицированные, **крайне информативные**, различные в разных ситуациях и в простых программах позволяют быстро разобраться начинающему, что именно не так:



Ошибки, связанные с видимостью переменных

Python

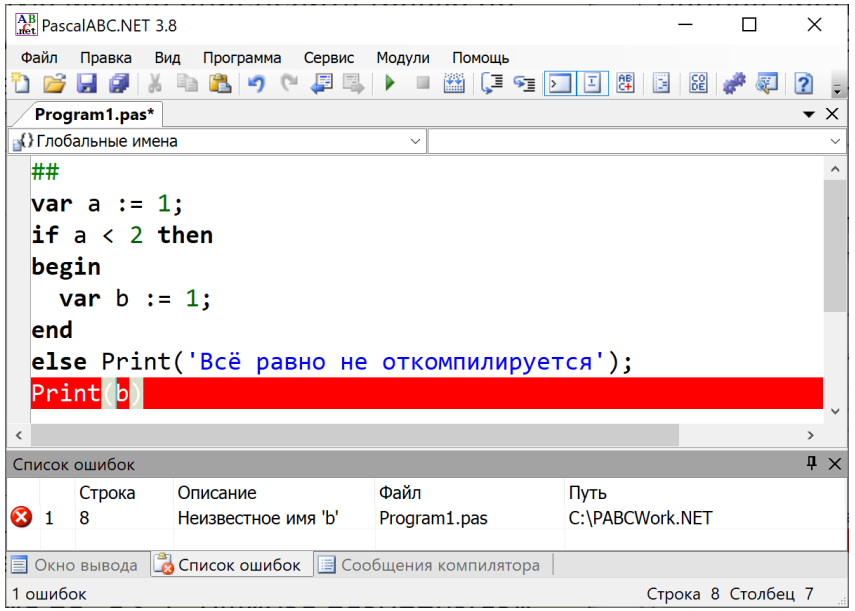
В Python переменную можно определить только по одной из веток. Если выполнение пойдёт именно по этой ветке, ошибка не будет замечена.



Данная программа **проработает успешно**. Однако при изменении условия на «a < 1» ошибка наконец-таки будет диагностирована, но на этапе выполнения

PascalABC.NET

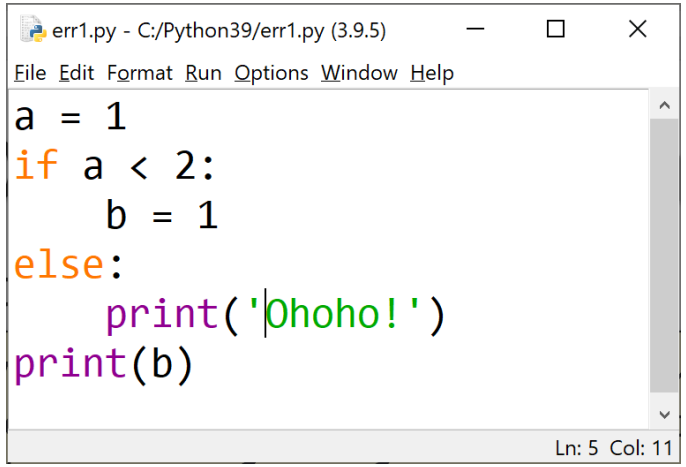
Данная программа **не откомпилируется** в PascalABC.NET. Если попытаться определить переменную только по одной ветке, то мы получим ошибку компиляции: переменная b, определенная в блоке, не видна при выходе из блока:



Ошибки, связанные с видимостью переменных

Python

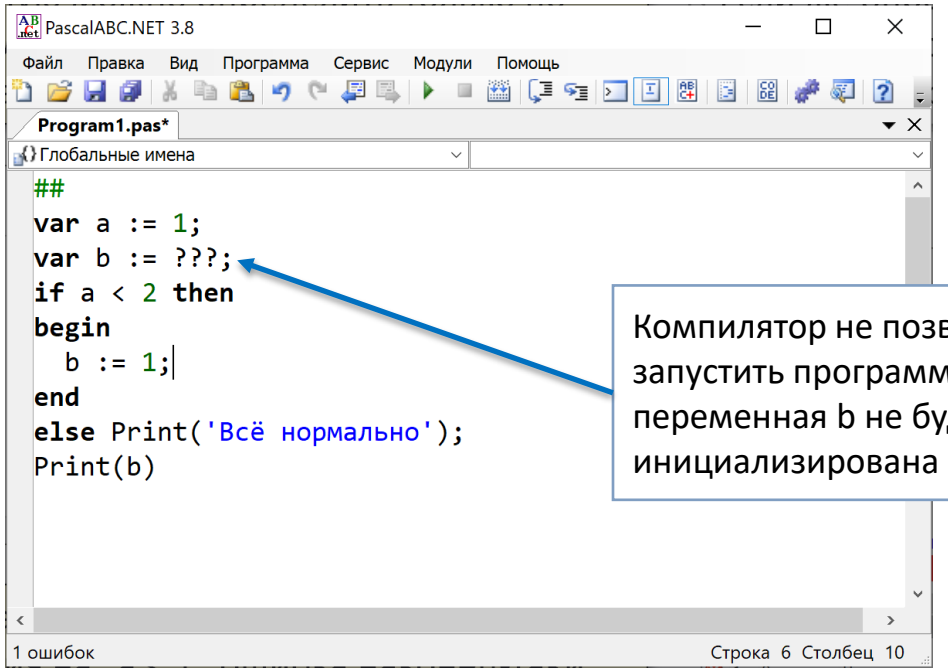
В Python переменную можно определить только по одной из веток. Если выполнение пойдёт именно по этой ветке, ошибка не будет замечена.



Данная программа **проработает успешно**. Однако при изменении условия на «a < 1» ошибка наконец-таки будет диагностирована, но на этапе выполнения.

PascalABC.NET

Если же определить переменную b до оператора if, то возникнет вопрос – каким значением инициализировать b, и пока мы его не напишем, программа **не откомпилируется.**



Компилятор не позволит запустить программу пока переменная b не будет здесь инициализирована значением

Время выполнения программ

Python

Т.к. Python является интерпретатором, время выполнения программ может быть очень большим.

PascalABC.NET

Компилятор генерирует эффективный код, сравнимый по быстродействию с кодом на C#, C++

Задача. Вычислить $\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{i\cdot j}$

```
import time
start = time.time()
n = 20000
s = 0.0
for i in range(1,n+1):
  for j in range(1,n+1):
    s += 1.0/(i*j)
end = time.time()
print(s, end - start)
```

```
##
var n := 20000;
var s := 0.0;
for var i:=1 to n do
for var j:=1 to n do
  s += 1.0/(i*j);
Println(s,Milliseconds);
```

43,1 с 

0,37 с 

В 110 раз быстрее!

Процессор Intel Core I7-8700K CPU 3.7 ГГц, ОЗУ 16 Гб

Время выполнения программ 2

Python

PascalABC.NET

Задача. Вычислить количество простых чисел, меньших 100000 (неэффективный алгоритм)

```
import time
start = time.time()
bound = 100000
primes = 0
for i in range(2, bound):
    divisors = i-1
    while i % divisors != 0:
        divisors -= 1
    if divisors == 1:
        primes += 1
print(primes, time.time() - start)
```

```
##
var bound := 100000;
var primes := 0;
foreach var i in 2..bound-1 do
begin
  var divisors := i-1;
  while i mod divisors <> 0 do
    divisors -= 1;
  if divisors = 1 then
    primes += 1;
end;
Print(primes, Milliseconds/1000);
```

300 с 

6 с 

В 50 раз быстрее!

Процессор Intel Core I7-8700K CPU 3.7 ГГц, ОЗУ 16 Гб

Время выполнения программ 3

Python

PascalABC.NET

Задача. Алгоритм MergeSort (n=10000000)

```
import random
import time

def MergeSort(a, L, R):
  i = L; j = R
  x = a[(i + j) // 2]
  while i<=j:
    while a[i] < x: i += 1
    while a[j] > x: j -= 1
    if i <= j:
      t = a[i]; a[i] = a[j]; a[j] = t
      i += 1; j -= 1
  if i < R: MergeSort(a, i, R);
  if j > L: MergeSort(a, L, j);

n = 10000000
a = [random.randint(1,100) for k in range(n)]
start = time.time()
MergeSort(a,0,n-1)
print(time.time() - start)
```

```
## procedure MergeSort(a: array of integer; L, R: integer);
begin
  var i := L; var j := R;
  var x := a[(i + j) div 2];
  repeat
    while a[i] < x do i += 1;
    while a[j] > x do j -= 1;
    if i <= j then begin
      Swap(a[i], a[j]);
      i := i + 1; j := j - 1;
    end;
  until i > j;
  if i < R then MergeSort(a, i, R);
  if j > L then MergeSort(a, L, j);
end;

var n := 10000000;
var a := ArrRandomInteger(n,1,100);
MillisecondsDelta;
MergeSort(a, 0, N-1);
Write(MillisecondsDelta);
```

26 с

0.33 с

в 80 раз быстрее!

Процессор Intel Core I7-8700K CPU 3.7 ГГц, ОЗУ 16 Гб

Краткая форма записи программ

Python

Язык Python славится краткостью записи.

Переменные не надо описывать, вместо операторных скобок используются отступы:

```
a = int(input())
b = int(input())
if a < b:
    print(a, b)
else:
    print(b, a)
```


PascalABC.NET

Чтобы приблизиться к краткости языка Python, в PascalABC.NET введены краткие программы, начинающиеся с ## и позволяющие не писать операторные скобки **begin** – **end** для всей программы:

```
##
var a := ReadInteger;
var b := ReadInteger;
if a < b then
  Print(a,b)
else Print(b,a)
```

Это позволяет на PascalABC.NET писать короткие программы буквально в одну строку:

```
##
ReadString.ToIntegers.Order.Print
```

Отступы vs операторные скобки

Python

Язык Python вместо операторных скобок использует **отступы**, объединяющие группу операторов в **блок**. Отступы являются частью синтаксиса программы:

```
x = int(input())
s = 0
while x:
    s += x % 10
    x //= 10
```

Отступы создают лёгкость восприятия кода на Python.

В большинстве языков программирования вместо отступов используются операторные скобки, что создаёт у обучающихся проблемы при переходе на другой язык

PascalABC.NET

Язык PascalABC.NET для объединения нескольких операторов в блок использует **операторные скобки begin – end**:

```
## var x := ReadInteger;
var s := 0;
while x <> 0 do
begin
  s += x mod 10;
  x := x div 10
end;
```

Операторные скобки **утяжеляют программу** – код на PascalABC.NET выглядит более тяжеловесным

Операторные скобки **встречаются в большинстве современных языков программирования**

Определение переменной

Python

Для того чтобы определить переменную, в Python достаточно присвоить ей значение нужного типа

Эта компактность и понятность записи позволяет легко использовать язык Python непрофессионалам и школьникам:

```
a = 5
b = 3.14
password = 'Python'
lst = [1,2,3]
```

PascalABC.NET

В PascalABC.NET определение переменной тоже максимально компактно, но следует использовать ключевое слово **var**. Это лишь немного сложнее Python

Тип переменной при этом выводится автоматически по типу присваиваемого значения:

```
##
var a := 0;
var b := 3.14;
var password := 'PascalABC.NET';
var lst := |1,2,3|;
```

При описании можно указать тип явно – например, когда мы инициализируем другим типом:

```
var r: real := 1;
```

Динамическая и статическая типизация

Python

Тип переменной в Python может меняться после каждого присваивания. Такая типизация называется динамической. **Динамическая типизация** позволяет писать гибкие программы.

Но при использовании динамической типизации могут возникнуть неожиданные **проблемы области видимости**, когда например по разным веткам переменная получает значения разных типов:

```
s = 0
x = int(input())
if x < 2:
    a = 5
else:
    a = 'строка'
s += a
```

В этой строке – причина возможной ошибки времени **выполнения**

В этой строке произойдёт (или не произойдёт в зависимости от значения x) ошибка времени **выполнения**

PascalABC.NET

В PascalABC.NET тип переменной задаётся в момент её описания (с ключевым словом **var**) и не меняется при дальнейшем выполнении программы.

Это позволяет компилятору легко проверять типы и выдавать ошибки типизации до запуска программы.

```
##
var s := 0;
var x := ReadInteger;
var a: integer;
if x < 2 then
  a := 5
else
  a := 'ошибка';
s += a
```

Переменную a обязательно необходимо описать и указать её тип явно. Это позволит компилятору диагностировать ошибку до запуска программы

Ошибка **компиляции**: Нельзя преобразовать тип string к integer

Ввод

Python

В Python можно вводить только строки. Далее строки можно преобразовывать в тот тип, который мы реально вводим. Это сложно для начинающих:

```
x = int(input())
y = float(input())
s = input()
```

В Python ввод нескольких данных одного типа осуществляется разбиением строки на слова и затем применением к каждому функции преобразования. Это сложно для начинающих и они вызубривают такой код без понимания сути:

```
x,y = map(int,input().split())
a = [int(x) for x in input().split()]
```

PascalABC.NET

В PascalABC.NET для ввода стандартных типов предусмотрены различные функции. Это просто:

```
##
var x := ReadInteger;
var y := ReadReal;
var s := ReadString;
```

```
##
var x: integer;
var y: real;
var s: string;
Read(x,y,s);
```

Для ввода нескольких значений и для ввода массива значений предусмотрены специальные функции, позволяющие не считывать всю строку, а указывать, сколько элементов мы хотим считать. Это проще:

```
##
var (x,y) := ReadInteger2;
var a := ReadArrReal(5);
```

Ввод в цикле

Python

В Python если данные вводятся в одной строке, мы обязаны вначале прочитать строку и затем превратить её в список. Списки на раннем этапе – дополнительная сложность для начинающих.

Кроме того, трудно ввести ровно 10 значений если они вводятся в нескольких строках – проще ввести все в одной строке:

```
lst = [int(x) for x in input().split()]
count = 0
for x in lst:
    if x % 2 != 0:
      count += 1
print(count)
```

PascalABC.NET

В PascalABC.NET данные можно вводить в цикле. Это просто для начинающих.

Списки здесь абсолютно **не требуются**.

Кроме того, легко ввести ровно 10 значений, не думая о том, на скольких строках они вводятся:

```
##
var count := 0;
loop 10 do
begin
  var x := ReadInteger;
  if x mod 2 <> 0 then
    count += 1;
end;
Print(count);
```

Вывод

Python

В Python функция print разделяет данные пробелом, осуществляя после этого переход следующую строку

```
print(1,2)
print(3,4)
```

```
1 2
3 4
```

Это не всегда удобно, поэтому в функции print может использоваться параметр end, указывающий разделитель после вывода

```
for x in range(10):
  print(x,end = ' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

PascalABC.NET

В PascalABC.NET процедура Print также разделяет данные пробелом, но не переходит после этого на новую строку. Для перехода на новую строку используется Println. **Это проще**

```
##
Print(1,2);
Print(3,4);
```

```
1 2 3 4
```

```
##
Println(1,2);
Println(3,4);
```

```
1 2
3 4
```

При выводе значений в цикле просто используем Print

```
##
for var i:=0 to 9 do
  Print(i)
```

```
0 1 2 3 4 5 6 7 8 9
```

Вывод составных данных

Python

В Python функция print структурно выводит составные данные произвольного уровня вложенности

```
d = {'Иванов': ('Физика',[5,4]),
     'Попов': ('Химия',[4,4,3])}
print(d)
```

```
{'Иванов': ('Физика', [5, 4]), 'Попов': ('Химия', [4, 4, 3])}
```

PascalABC.NET

В PascalABC.NET процедура Print также выводит составные данные произвольного уровня вложенности

```
##
var x := Dict(('Иванов',('Физика',|5,4|)),
              ('Попов',('Химия',|4,4,3|)));
Print(x);
```

```
{(Иванов,(Физика,[5,4])),(Попов,(Химия,[4,4,3]))}
```

Большие наборы данных при выводе обрезаются

```
## Print(Arr(1..200))
```

```
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,4
6,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67
,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,
89,90,91,92,93,94,95,96,97,98,99,100,...]
```

Форматные и интерполированные строки

Python

В Python 3.6+ **форматные строки** начинаются с символа **f** и используются для упрощения вывода. Выражения в {} в форматной строке заменяются на их значения:

```
a, b = 2, 3
print(f"{a} + {b} = {a+b}")
```

```
2 + 3 = 5
```

В форматной строке можно указывать ширину поля вывода и количество знаков после точки для типа float:

```
for i in range(1,4):
  print(f"{i:3} {1/i:8.4f}")
```

```
  1   1.0000
  2   0.5000
  3   0.3333
  4   0.2500
```

PascalABC.NET

В PascalABC.NET форматные строки называются **интерполированными строками** и начинаются с символа $.

```
## var (a,b) := (2,3);
Println($'{a} + {b} = {a+b}');
```

```
2 + 3 = 5
```

В форматной строке можно указывать ширину поля вывода и количество знаков после точки для типа real:

```
## foreach var i in 1..4 do
     Println($'{i,3} {1/i,8:f4}')
```

```
  1   1.0000
  2   0.5000
  3   0.3333
  4   0.2500
```

Операция возведения в степень

Python

В Python операция возведения в степень имеет вид a ** b. При этом тип результата определяется крайне просто (напоминаем, что в Python – все целые длинные):

- если a и b – целые, b > 0, то результат – целый
- если a или b – типа float, то и результат – типа float (может произойти переполнение)

Использование длинных целых приводит к **падению производительности**, поэтому для небольших целых возведение в степень в Python работает медленно:

```
2 ** 3
```

PascalABC.NET

В PascalABC.NET также имеется операция возведения в степень a ** b. Однако статическая типизация накладывает свои ограничения:

- если a и b – целые или вещественные, то результат – вещественный (может произойти переполнение)
- если a: BigInteger, b – целое >= 0, то результат – BigInteger
- если a: BigInteger, b – целое < 0, то – ошибка времени выполнения
- если a: BigInteger, b – вещественное или наоборот, то – ошибка компиляции

Циклы

Python

В Python простейший цикл for записывается с использованием весьма сложной конструкции range(), **не очевидной начинающим**: это **функция**, возвращающая **последовательность**

```
for x in range(1,11):
  print(x)
```

```
1 2 3 4 5 6 7 8 9 10
```

Цикл «повторить 10 раз» записывается так:

```
for x in range(10):
  print('*')
```

PascalABC.NET

В PascalABC.NET цикл for **более понятен начинающему**

```
##
for var x := 1 to 10 do
  Print(x);
```

```
1 2 3 4 5 6 7 8 9 10
```

Цикл «повторить 10 раз» записывается проще:

```
##
loop 10 do
  Write('*');
```

Длинные целые

Python

В Python все целые – длинные.

Ниже приводится алгоритм вычисления произведения первых n натуральных чисел (n!)

```
n = 100
p = 1
for i in range(1,n+1):
    p *= i
print(p)
```

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

PascalABC.NET

В PascalABC.NET также имеется тип длинных целых BigInteger. По умолчанию целые константы имеют тип integer (для эффективности), поэтому **тип переменной p необходимо указывать явно**

```
## var n := 100;
var p: BigInteger := 1;
for var i := 1 to n do
  p *= i;
Print(p);
```

Можно короче:
var p := 1bi;
Окончание bi означает, что целая константа имеет тип BigInteger

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

Можно еще короче:

```
## Range(1bi,100bi).Product.Print
```

Диапазоны и операция in

Python

В Python отсутствуют специальные значения-диапазоны, а попадание в диапазон описывается двойным неравенством:

```
x = 5
if 3 <= x <= 7:
  print('Попадание в диапазон')
```

Аналогичный циклу справа цикл в Python записывается менее понятно:

```
for x in range(1,10):
  print(x)
```

PascalABC.NET

В PascalABC.NET введены значения-диапазоны a..b, позволяющие эффективно и понятно проверять попадание элемента в диапазон:

```
##
var x := 5;
if x in 3..7 then
  Print('Попадание в диапазон')
```

В PascalABC.NET диапазоны можно также использовать в заголовке цикла foreach:

```
##
foreach var x in 1..9 do
  Print(x)
```

Условная операция

Python

В Python условная операция выглядит так

```
x = 5
y = 0 if x < 5 else 1
```

«Запутанной условная операция в Python выглядит» (магистр Йода)

Но конечно дело привычки

PascalABC.NET

В PascalABC.NET условная операция выглядит так же как и условный оператор (это привычно)

```
##
var x := 5;
var y := if x < 0 then 0 else 1;
```

Как альтернатива имеется также условная операция в стиле C:

```
var y := x < 0 ? 0 : 1;
```

Кортежи и распаковка в переменные

Python

В Python кортежи активно используются для объединения данных в одно целое. Извлечение данных из кортежа называется распаковкой

Скобки при этом можно либо использовать либо нет

```
t = 1, 2
a, b = t
```

```
a, b = b, a
(a, b) = (b, a)
```

Распаковывать при этом можно не только кортежи, но и списки. Можно также распаковывать вложенные структуры и использовать расширенную распаковку

```
a, (b, c) = (1, (2, 3))
print(a,b,c)
a, *b, c = [1, 2, 3, 4]
print(a,b,c)
```

```
1 2 3
1 [2, 3] 4
```

PascalABC.NET

В PascalABC.NET кортежи также активно используются. Скобки при этом обязательны

```
##
var t := (1,2);
var (a,b) := t;
(a,b) := (b,a);
```

Распаковывать можно также массивы и последовательности. Расширенной распаковки нет

```
##
var a := ArrRandom(10);
var (x,y) := a;
var (min,min1) := a.Distinct.Order;
```

Списки и массивы

Python

В Python списки как правило используются в качестве массивов. Для инициализации последовательностью значений используются []:

```
a = [1,2,3,4,5]
```

Для инициализации по данному правилу или для ввода с клавиатуры используются генераторы списков:

```
a = [2*i+1 for i in range(10)]
b = [randint(1,50) for i in range(10)]
c = [int(input()) for i in range(10)]
```

Для заполнения используются также операции + и *

```
a = [0]*10;
b = [1]*5 + [2]*5;
```

PascalABC.NET

В PascalABC.NET массивы инициализируются набором значений с использованием || (к сожалению, [] заняты под множества) или функции Arr

```
## var a := |1,2,3,4,5|;
var a1 := Arr(1..5);
```

Для инициализации по данному правилу используются функции-генераторы **ArrGen**, для ввода с клавиатуры – ReadArr, а для генерации случайного - ArrRandom

```
## var a := ArrGen(10,i->2*i+1);
var b := ArrRandomInteger(10,1,50);
var c := ReadArrInteger(10);
```

Для заполнения также используются операции + и *

```
a = |0|*10;
b = |1|*5 + |2|*5;
```

Списки и массивы 2

Python

В Python для списков можно использовать операции добавления, удаления, вставки

```
a = [1,2,33,4,5]
a.append(10)
a.remove(33)
a.insert(1,66)
a.pop(0)
print(a)
```

[66, 2, 4, 5, 10]

PascalABC.NET

В PascalABC.NET для добавления, удаления, вставки следует использовать тип List<T>

```
## var a := Lst(1,2,33,4,5);
a.Add(10);
a.Remove(33);
a.Insert(1,66);
a.RemoveAt(0);
Print(a)
```

[66, 2, 4, 5, 10]

Списки и массивы 3

Python

В Python типичные операции с массивами, такие как нахождение суммы или минимума, являются внешними встроенными функциями.

```
from random import randint
a = [randint(1,50) for i in range(10)]
print(a)
print(sum(a),len(a),max(a),min(a))
```

Чтобы найти среднее или индекс минимального, следует либо писать неэффективный код, либо подключить модуль numpy, предварительно установив его, и использовать методы mean, argmin

```
print(sum(a)/len(a),a.index(max(a))) # неэфф.
import numpy as np
print(a.mean(),a.argmax())
```

PascalABC.NET

В PascalABC.NET все указанные характеристики массива являются методами и записываются по точке

```
## var a := ArrRandomInteger(10);
Println(a);
Println(a.Sum,a.Length,a.Min,a.Max);
Println(a.Average); // среднее значение
```

В PascalABC.NET у массивов есть встроенный метод «индекс максимального»:

```
## a.IndexMax.Print
```


Количество уникальных элементов

Python

В Python принято вычислять **количество уникальных элементов** списка, преобразуя его во множество, которое не может содержать несколько одинаковых элементов

```
from random import randint
a = [randint(1,100) for i in range(100)]
print(len(a))
print(len(set(a)))
```

```
100
65
```

При подключении модуля numpy можно воспользоваться методом unique, аналогичном Distinct

```
import numpy as np
print(len(np.unique(a)))
```

```
65
```

PascalABC.NET

В PascalABC.NET для определения количества уникальных элементов в последовательности, используются оба описанных в Python метода. Метод последовательностей Distinct аналогичен методу unique в Python и не требует подключения внешних пакетов.

```
## var a := ArrRandomInteger(100);
a.Count.Println;
a.Distinct.Count.Println
```

Второй метод, хотя и не так эффективен, тоже может использоваться:

```
HSet(a).Count.Println
```

Срезы

Python

В Python для списков и строк определены срезы, в том числе с шагом

```
a = [1,2,3,4,5]; s = 'ABCDE'
print(a[2:]); print(a[::2]); print(s[::2])
```

```
[3, 4, 5]
[1, 3, 5]
ACE
```

Срезы можно использовать в левой части присваивания

```
a[:2] = a[1:3]; print(a)
```

```
[2, 3, 3, 4, 5]
```

Срезы можно совмещать с отрицательными индексами

```
a = [1,2,3,4,5]; print(a[1:-1])
```

```
[2, 3, 4]
```

PascalABC.NET

В PascalABC.NET срезы определены для массивов, списков List<T> и строк

```
## var a := |1,2,3,4,5|; var s := 'ABCDE';
Print(a[2:]); Print(a[::2]); Print(s[::2]);
```

```
[3,4,5] [1,3,5] ACE
```

Срезы также можно использовать в левой части присваивания

```
a[:2] := a[1:3]; Print(a)
```

```
[2, 3, 3, 4, 5]
```

Вместо отрицательных индексов в PascalABC.NET используется конструкция ^1 – первый элемент с конца (такая же конструкция используется в C#)

```
a = [1,2,3,4,5]; Print(a[1:^1])
```

```
[2, 3, 4]
```

Лямбда-выражения

Python

Лямбда-выражения в Python записываются крайне **громоздко**, поэтому редко используются в обучении

```
a = [1,2,3,4,5]
sq = map(lambda x: x * x, a)
fsq = filter(lambda x: x % 2 == 0, sq)
srt = sorted(fsq, reverse = True)
for x in srt:
    print(x,end=' ')
```

По причине громоздкости лямбда-выражений вместо этого кода в Python используются генераторы списков

PascalABC.NET

В PascalABC.NET лямбда-выражения крайне **легковесны**, что позволяет записать данный алгоритм компактно и понятно

```
##
var a := Arr(1..5);
a.Select(x -> x * x)
  .Where(x -> x.IsEven)
  .OrderDescending.Println;
```

Генераторы списков

Python

Генераторы списков – мощная конструкция ряда языков программирования (в т.ч. Python) для генерации списков. Список генерируется по более простой последовательности, при этом элементы можно **преобразовывать** и **фильтровать**

```
a = [int(input()) for x in range(10)]
a2 = [x*x for x in a if x % 2 == 0]
```

Несмотря на широкую распространенность, генераторы списков имеют ряд **недостатков**, которые мы покажем в дальнейших примерах

PascalABC.NET

В PascalABC.NET вместо генераторов списков используются операции Select и Where с последовательностями. Условия фильтрации и преобразования необходимо записывать в виде лямбда-выражений

```
##
var a := ReadSeqInteger(10);
var a2 := a.Where(x->x.IsEven).Select(x->x*x);
```

Преимущество заключается в том, что операций с последовательностями – несколько десятков, и их легко комбинировать друг с другом, используя точечную нотацию

Генераторы списков – примеры

Python

Если необходимо **только отфильтровать** значения, то синтаксис становится **громоздким**: приходится повторять тривиальную конструкцию x for x in a

```
a = [12,3,5,2,41,92,28,35]
print([x for x in a if x % 2 == 0])
```

Если необходимо **вначале преобразовать**, используя сложную функцию f, **а затем отфильтровать** по условию, то используем код:

```
b = [f(x) for x in a]
print([y for y in b if y % 2 == 0])
```

В Python функция sum – внешняя, что хорошо воспринимается как известная математическая запись

```
print(sum(2*i+1 for i in range(10)))
```

$$\sum_{i=0}^{9}(2i+1)$$

PascalABC.NET

Если необходимо **только отфильтровать** значения, то мы используем только метод **Where**:

```
##
var a = |12,3,5,2,41,92,28,35|;
a.Where(x->x.IsEven).Print
```

Если необходимо **вначале преобразовать**, используя сложную функцию f, **а затем отфильтровать**, то вначале применяется метод Select, потом Where:

```
a.Select(x->f(x)).Where(y->y.IsEven).Print
```

В PascalABC.NET функция Sum является методом последовательности, что позволяет использовать точечную нотацию с **подсказкой** по точке в IDE

```
(0..9).Select(i -> 2*i+1).Sum.Print
```

Итерируемые объекты и последовательности

Python

В Python итерируемые объекты позволяют получать элементы один за другим и не хранят в памяти все значения

```
seq = (x*x for x in range(10))
print(seq)
for x in seq:
    print(x, end=' ')
```

```
<generator object <genexpr> at 0x000001B811B8F4A0>
0 1 4 9 16 25 36 49 64 81
```

Чтобы вывести все элементы, используется цикл for по итерируемому объекту
Итерируемый объект **одноразовый**: **второй проход** по нему ничего не возвращает (неожиданно!)

```
for x in seq:
    print(x, end=' ')
```

ничего не выводится!

PascalABC.NET

В PascalABC.NET последовательности имеют тип **sequence of** T. Последовательности позволяют получать элементы один за другим и не хранят (вообще говоря) в памяти все значения. Последовательностями являются массивы, списки, множества, словари.

```
## var seq := SeqGen(10,i->i*i);
Println(seq);
seq.Println;
```

```
[0,1,4,9,16,25,36,49,64,81]
0 1 4 9 16 25 36 49 64 81
```

Для вывода последовательности можно пользоваться как процедурой Print, так и методом Print – это можно делать много раз в отличие от Python

Функции all и any

Python

Функции all и any в Python определены для списков и других итерируемых объектов с элементами логического типа

```
a = [True,True,True]
print(all(a)) # все элементы = True
a = [True,False,True]
print(any(a)) # есть элемент = True
```

Чтобы проверить, что **все числа в списке чётные**, надо список спроектировать на **список логических значений** и после этого применить all (это **усложнено**)

```
print(all(x % 2 == 0 for x in a))
```

Это – **итерируемый объект**. Но скобки для него можно не писать – их заменяют скобки вызова функции (!)

PascalABC.NET

В PascalABC.NET функции All и Any являются методами последовательностей. Для проверки того, что **все числа в массиве чётные**, функции All надо передать условие вида **x -> x.IsEven**, дословно читающееся как «**x такое что x – чётное**». Это **проще** и естественнее:

```
## var a := |3,1,5,17,99|;
a.All(x -> x.IsEven).Print;
```

Другой пример:

```
a.Any(x -> x in 10..20).Print;
```

Сортировка

Python

В Python сортировка списка **по месту** (меняется исходный список) осуществляется методом списка sort. Для сортировки в обратном порядке используется параметр reverse = True:

```
a = [randint(1,100) for i in range(10)]
a.sort(); a.sort(reverse = True)
```

Нестандартный критерий сортировки можно указать в параметре key:

```
pp = [('Сидоров',20),('Петров',22),
  ('Попов',20),('Иванов',22)]
pp.sort(key = lambda p: p[0]) # по фамилии
print(pp)
pp.sort(key = lambda p: (p[1],p[0]))
print(pp)
```

```
(Иванов,22) (Петров,22) (Попов,20) (Сидоров,20)
(Попов,20) (Сидоров,20) (Иванов,22) (Петров,22)
```

PascalABC.NET

В PascalABC.NET сортировка массива **по месту** (меняется исходный массив) осуществляется внешней функцией Sort или SortDescending (обратный порядок):

```
## var a := ArrRandomInteger(10);
Sort(a); SortDescending(a);
```

Можно указать нестандартный критерий сортировки в виде функции проекции на ключ сортировки

```
var pp := |('Сидоров',20),('Петров',22),
  ('Попов',20),('Иванов',22)|;
Sort(pp, p->p[0]); // по фамилии
pp.Println;
Sort(pp, p->(p[1],p[0])); pp.Println;
```

```
(Иванов,22) (Петров,22) (Попов,20) (Сидоров,20)
(Попов,20) (Сидоров,20) (Иванов,22) (Петров,22)
```

проекция персоны p на фамилию p[0]

Составной ключ в виде кортежа: по возрасту, потом по фамилии

Ещё раз: лямбда-выражения в PascalABC.NET выглядят существенно менее громоздко

Сортировка 2

Python

В Python функция sorted возвращает **новый список**. Она также имеет параметр reverse и параметр key:

```
a = [('Сидоров',20),('Петров',22),
  ('Попов',20),('Иванов',22)]
print(sorted(a, key = lambda p: p[0]))
print(sorted(a, key = lambda p: (p[1],p[0])))
```

```
(Сидоров,20) (Попов,20) (Петров,22) (Иванов,22)
(Попов,20) (Сидоров,20) (Иванов,22) (Петров,22)
```

Сортировка простых данных осуществляется без параметра key:

```
a = [3,1,55,17,9]
print(sorted(a))
print(sorted(a, reverse = True))
```

```
[1, 3, 9, 17, 55]
[55, 17, 9, 3, 1]
```

PascalABC.NET

В PascalABC.NET сортировка любой последовательности (в т.ч. массива) осуществляется с помощью метода OrderBy, параметром которого является проекция элемента на ключ сортировки. Метод ThenBy сортирует уже отсортированную последовательность по дополнительному ключу

```
## var a := |('Сидоров',20),('Петров',22),
  ('Попов',20),('Иванов',22)|;
a.OrderByDescending(p->p[0]).Println;
a.OrderBy(p->p[1]).ThenBy(p->p[0]).Println;
```

```
(Сидоров,20) (Попов,20) (Петров,22) (Иванов,22)
(Попов,20) (Сидоров,20) (Иванов,22) (Петров,22)
```

Для сортировки простых данных служат методы Order и OrderDescending:

```
## var a := |1,9,3,6,4|;
a.Order.Println;
a.OrderDescending.Println;
```

Матрицы – создание, вывод

Python

В Python матрица представляется как список списков:

```
a = [[1,2,3],[4,5,6]]
print(a)
```

```
[[1, 2, 3], [4, 5, 6]]
```

Доступ к элементам:

```
a[0][1] = 0
a[-1][-1] = 77
print(a)
```

```
[[1, 0, 3], [4, 5, 77]]
```

Необходимо дополнительно установить пакет numpy – он не входит в стандартную поставку

Чтобы получить дополнительный сервис, следует преобразовать список списков в массив numpy:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a)
```

```
[[1 2 3]
 [4 5 6]]
```

PascalABC.NET

В PascalABC.NET матрицы описываются так:

```
## var a: array [,] of integer;
```

Как обычно, многочисленные создающие функции позволяют использовать автовыведение типа:

```
## var a := Matr(2,3,|1,2,3,4,5,6|);
a.Println(3);
```

```
  1  2  3
  4  5  6
```

При выводе можно указывать ширину столбцов. Вывод осуществляется в виде таблицы (!)

Можно инициализировать одномерными массивами для лучшего структурирования элементов:

```
## var a := Matr(|1,2,3|,|4,5,6|);
```

Доступ к элементам:

```
a[0,1] := 0; a[^1,^1] := 77; a.Println;
```

```
  1   0   3
  4   5  77
```

Матрицы – заполнение

Python

В Python для заполнения матриц одним значением можно воспользоваться списковым включением

```
a = [[66]*3 for i in range(2)]
# a = [[66]*3]*2 – так нельзя!!!
print(a)
```

```
[[66, 66, 66], [66, 66, 66]]
```

Используя модуль numpy, заполнить матрицу одним значением и случайными значениями можно так:

```
import numpy as np
a = np.full((2,3),66)
print(a)
a = np.random.random((2,3))
print(a)
```

```
[[66 66 66]
 [66 66 66]]
[[0.15324882 0.41766248 0.32899827]
 [0.6084343  0.64118821 0.39340361]]
```

PascalABC.NET

Заполнение значением 66

В PascalABC.NET имеется ряд функций для заполнения матриц: MatrFill для заполнения одинаковыми элементами, MatrRandom для заполнения случайными

```
## var a := MatrFill(2,3,66);
var b := MatrRandomReal(3,4,0,99);
b.Println
```

Диапазон от 0 до 99

```
  58.94  13.61  53.77  68.20
  64.34  45.12  89.12  72.93
  13.15  67.36  31.26  48.60
```

По умолчанию для вывода вещественных используется 7 позиций и 2 позиции под дробную часть

При выводе матрицы вещественных можно указать количество позиций под число и под дробную часть

```
b.Println(9,4);
```

```
  35.4533  35.3281  97.7690  56.3820
  49.1106  97.3045  30.6958  44.6754
  27.5467  64.8031  72.0013  39.6183
```


Матрицы – заполнение по правилу

Python

В Python для заполнения матриц (списков списков) по правилу можно воспользоваться вложенными генераторами списков

```
a = [[(i+1)*(j+1) for i in range(3)] \
      for j in range(3)]
print(a)
```

```
[[1, 2, 3], [2, 4, 6], [3, 6, 9]]
```

PascalABC.NET

В PascalABC.NET для заполнения матрицы по некоторому правилу используется функция MatrGen

```
var c := MatrGen(3,3,(i,j)->(i+1)*(j+1));
c.Println
```

```
  1   2   3
  2   4   6
  3   6   9
```

Матрицы – строки и столбцы

Python

В Python для работы со строками и столбцами матрицы лучше использовать модуль numpy:

```
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a[2,:])
print(a[:,0])
```

```
[7 8 9]
[1 4 7]
```

Для строк и столбцов можно вычислять характеристики:

```
print(a[2,:].sum())
mins = [a[:,x].min() for x in range(3)]
print(mins)
```

```
24
[1, 2, 3]
```

PascalABC.NET

В PascalABC.NET можно получить все элементы определенной строки или столбца матрицы, используя методы Row и Col:

```
## var a := Matr(|1,2,3|,|4,5,6|,|7,8,9|);
a.Row(2).Println;
a.Col(0).Println;
```

```
7 8 9
1 4 7
```

Вместо Row и Col можно использовать срезы матриц:

```
a[2,:].Println;
a[:,0].Println;
```

Для строк и столбцов легко вычислять характеристики:

```
a.Row(2).Sum.Println;
var mins := ArrGen(3, j -> a.Col(j).Min);
```

Одномерный массив минимумов в столбцах матрицы

Строки

Python

PascalABC.NET

Строки в Python и PascalABC.NET имеют много общего. По умолчанию строки в Pascal индексируются с 1, но в PascalABC.NET это можно изменить с помощью директивы компилятора {$zerobasedstrings}

```
s = 'ABCD'
s1 = 'BC'
print(s[0],s[-1])
# элементы менять нельзя! s[0]='x' -ошибка!
print(s.lower(),s.upper())
n = s.find(s1)
s = s.replace(s1,'XY')
# s.remove отсутствует
print(s1 in s)
print(s[:3], s[1:])  # срезы строк
i = int(s); s = str(i) # строка<->целое
s = 'первый второй третий'
ss = s.split() # список строк
' '.join(ss) # список в строку, разделитель ' '
```

Несколько путаная конструкция: разделитель ' ' указывается как объект, у которого есть метод join, принимающий список строк

```
{$zerobasedstrings}
## var s := 'ABCD';
var s1 := 'BC';
Print(s[0],s[^1]);
s[0]:='x'; // элементы менять можно!
Print(s.ToLower, s.ToUpper);
var n := s.IndexOf(s1);
s := s.Replace(s1,'XY');
s := s.Remove(3,1);
Println(s1 in s);
Println(s[:3], s[1:]); // срезы строк
var i := s.ToInteger; s := i.ToString;
s := 'первый второй третий';
var ss := s.ToWords;
s := ss.JoinToString;
```

Конструкция очевидная: последовательность строк имеет метод объединения всех строк в строку

Словари

Python

В Python словари представляют собой неупорядоченные наборы пар (ключ, значение) с быстрым поиском по ключу.

```
d = {'крокодил': 3,'бегемот': 2}
d['крокодил'] += 1
d['какаду'] = d.get('какаду',0) + 1;
del d['крокодил']
if 'крокодил' not in d:
    print('крокодил не ловится')
print(d)
for key,value in d.items():
    print(key,value)
```

```
крокодил не ловится
{'бегемот': 2, 'какаду': 1}
бегемот 2
какаду 1
```

PascalABC.NET

В PascalABC.NET словари Dictionary<Key, Value> представляют собой неупорядоченные наборы пар (ключ, значение) с быстрым поиском по ключу. Создающая функция словаря – Dict, в качестве параметров ей передается список кортежей.

```
## var d := Dict(('крокодил',3),('бегемот',2));
d['крокодил'] += 1;
d['какаду'] := d.Get('какаду') + 1;
d -= 'крокодил';
if 'крокодил' not in d then
  Println('крокодил не ловится');
d.Println;
foreach var kv in d do
  Println(kv.Key, kv.Value);
```

```
крокодил не ловится
(бегемот,2) (какаду,1)
бегемот 2
какаду 1
```

Текстовые файлы

Python

В Python продемонстрируем основные операции с текстовыми файлами: открытие на чтение, открытие на запись, закрытие, ввод-вывод, для закрытых файлов – переименование и удаление

```
import os
import os.path
f = open('a.txt','w')
n, r, s, b = 1, 3.14, 'Hello', True
f.write(f"{n} {r} {s} {b}")
f.close()
if os.path.exists('b.txt'):
    os.remove('b.txt')
os.rename('a.txt','b.txt')
with open('b.txt','r') as f:
    ss = f.readline().split();
n, r, s, b = int(ss[0]), float(ss[1]), ss[2], \
    bool(ss[3])
print(n, r, s, b)
```

При использовании with файл закрывается автоматически

```
1 3.14 Hello True
```

PascalABC.NET

В PascalABC.NET текстовые файлы имеют тип Text. Основные операции: открытие на чтение, открытие на запись, закрытие, ввод-вывод, для закрытых файлов – переименование и удаление

```
## var f: Text;
f := OpenWrite('a.txt');
f.Println(1,3.14,'Hello',True);
f.Close;
if FileExists('b.txt') then
  DeleteFile('b.txt');
f.Rename('b.txt');
var f1 := OpenRead('b.txt');
var n := f1.ReadInteger;
var r := f1.ReadReal;
var s := f1.ReadWord;
var b := f1.ReadBoolean;
f1.Close;
Print(n,r,s,b);
```

```
1 3.14 Hello True
```

Цикл по строкам текстового файла

Python

В Python цикл по строкам текстового файла выглядит следующим образом:

```
with open('6.py','r') as f:
    while True:
        s = f.readline()
        if not s:
            break
        s = s.rstrip()
         if 'open' in s:
             print(s)
```

Или так:

```
with open('6.py','r') as f:
    for s in f:
        s = s.rstrip()
        if 'open' in s:
            print(s)
```

Или так:

```
with open('6.py','r') as f:
     print(''.join(s for s in f if 'open' in s))
```

PascalABC.NET

В PascalABC.NET цикл по строкам текстового файла выглядит следующим образом:

```
## var f := OpenRead('a1.pas');
while not f.Eof do begin
  var s := f.ReadString;
  if 'begin' in s then
      Println(s);
end;
f.Close;
```

Можно рассматривать файл как последовательность строк, используя функцию ReadLines, которая автоматически открывает и закрывает файл

```
## foreach var s in ReadLines('a1.pas') do
  if 'begin' in s then
    Println(s);
```

Или то же одной строкой (one liner):

```
## ReadLines('a1.pas').Where(s -> 'begin' in s)
    .PrintLines
```

Функции

Python

В Python при описании функции **не надо описывать типы параметров** и типы возвращаемых значений. Такой код читается легко, но может приводить к ошибкам на этапе выполнения:

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

print(Add(2,3))
print(Add('12',4))
```

Ошибка выполнения (**плохо**)

Функция Add будет работать со всеми типами, для которых доступна операция сложения:

```
print(Add('2','3'))
```

PascalABC.NET

В PascalABC.NET существует возможность писать короткие определения функций и не указывать тип возвращаемого значения. Он выводится автоматически

```
##
function Add(a,b: integer) := a + b;

Print(Add(2,3));
Print(Add('2',3));
```

Ошибка компиляции (**хорошо**): Нельзя преобразовать тип char к integer

Функция Add не будет работать с типом char – для другого типа необходимо определить вторую (перегруженную) версию Add

```
function Add(a,b: char) := a + b;
```

Это – общая особенность языков со статической типизацией, делающая код более эффективным и позволяющая отлавливать ошибки на этапе компиляции

Функции 2

Python

Поскольку в Python не надо описывать типы параметров и типы возвращаемых значений при описании функции, код получается лёгким:

```
def DivsOn(x, divs):
  lst = []
  for d in divs:
    if x % d == 0:
      lst.append(d)
  return lst

print(DivsOn(36,[2,3,4,5,6,7]))
```

```
[2, 3, 4, 6]
```

Помним, что Python не защищён здесь от ошибок при выполнениии:

```
print(DivsOn(36,2)) # ошибка выполнения – плохо!
```

PascalABC.NET

В PascalABC.NET существует возможность писать короткие определения функций и не указывать тип возвращаемого значения. Он выводится автоматически

```
##
function DivsOn(x: integer; divs: List<integer>):
  List<integer>;
begin
  Result := new List<integer>;
  foreach var d in divs do
    if x mod d = 0 then
      Result.Add(d)
end;

Print(DivsOn(36,Lst(2,3,4,5,6
```

Все типы должны быть описаны. Это нужно для генерации эффективного кода

```
[2, 3, 4, 6]
```

При использовании неправильных типов произойдет ошибка на этапе компиляции:

```
print(DivsOn(36,2)) # ошибка компиляции - хорошо
```

Кеширование результатов функции

Python

В Python с версии 3.9 для кеширования результатов чистой функции используется декоратор @cache

```
import time
from functools import cache

@cache
def fib(n):
    if n < 3:
        return 1
    return fib(n - 1) + fib(n - 2)

s = time.time()
print(fib(42), time.time() - s)
```

```
267914296 0.0
```

Если закомментировать декоратор @cache, то вычисления будут продолжаться 38 секунд:

```
267914296 37.79281687736511
```

PascalABC.NET

В PascalABC.NET для кеширования результатов чистой функции используется атрибут [Cache]

```
##
[Cache]
function fib(n: integer): integer :=
  if n in 1..2 then 1
  else fib(n-1) + fib(n-2);

function fib1(n: integer): integer :=
  if n in 1..2 then 1
  else fib1(n-1) + fib1(n-2);

Println(fib(42),MillisecondsDelta/1000);
Println(fib1(42),MillisecondsDelta/1000);
```

```
267914296 0.002
267914296 9.252
```

Первое вычисление мгновенно, второе – 9 секунд

Выводы

Язык PascalABC.NET можно использовать:

- в стиле старого Паскаля (для поддержки старого кода)
- в стиле языка C# (полная совместимость с C# на уровне сборок, использование .NET-библиотек)
- в стиле языка Python (короткий синтаксис, короткие имена генераторов массивов, списков, последовательностей и проч.)

Несомненно, наиболее эффективно использовать PascalABC.NET
в стиле PascalABC.NET ☺