

План курса

Литература

- Б.В. Керниган, Д.М. Ричи. «Язык программирования C»
- Python books: <https://wiki.python.org/moin/PythonBooks>
- Бьерн Страуструп «Язык программирования C++»



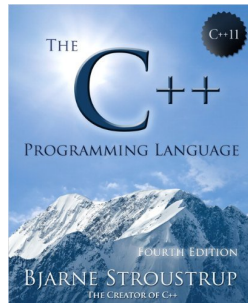
python™

» PythonBooks

» IntroductoryBooks » PythonBooks

There are a variety of books about Python. Here's a guide to them:

- » [IntroductoryBooks](#) (gentle overviews of the language)
- » [AdvancedBooks](#) (for when you don't want gentle)
- » [ReferenceBooks](#) (much information in a small space)
- » Specific applications:
 - » [GameProgrammingBooks](#)
 - » [NetworkProgrammingBooks](#)
 - » [GuiBooks](#)
 - » [JythonBooks](#)
 - » [ScientificProgrammingBooks](#)
 - » [SystemAdministrationBooks](#)
 - » [WebProgrammingBooks](#)
 - » [WindowsBooks](#)
 - » [XmlBooks](#)
 - » [ZopeBooks](#)
- » Books in languages other than English:



Лекции и практические задания

Лекции на GitHub

Лекции «внутри ОИЯИ»

Для практических занятий требуется:

- Компилятор `gcc` (GNU Compiler Collection) или `clang` («клэнг»)
- Python-3

Рекомендации для персонального компьютера

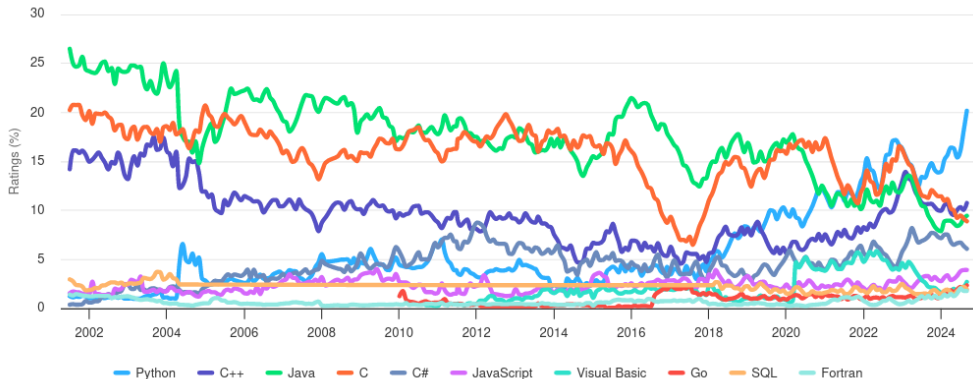
- ✓ Либо Unix подобная ОС: Linux, macOS ...
- ✓ Либо Windows и как одно из возможных решений:
 - `VirtualBox for Windows hosts`
 - `Ubuntu собранное для VirtualBox`, см. <https://www.osboxes.org/faqs/>
- 👁️ Всё это займет ~10GB

Место Python/C/C++ среди других языков

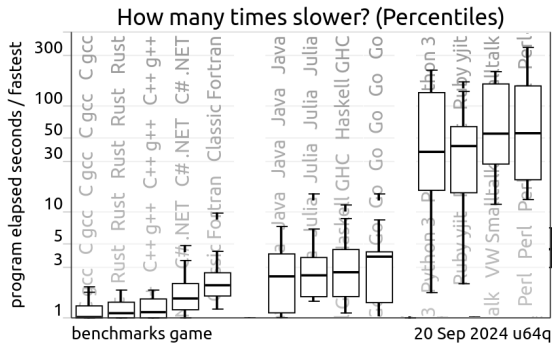
«Индекс популярности» языков

TIOBE Programming Community Index

Source: www.tiobe.com



«Тест производительности» языков (Benchmarks Game):



- несколько программ для «типичных проблем»



тесты не предскажут будет ли именно ваша программа работать быстрее на «языке X»!

Краткая история стандартов языка C

- **C89** – первый официальный стандарт C: ANSI 1989
(C90 – first ISO version, C95 – bug-fix, minor version)
- **C99** – ISO версия 1999 года: **улучшена поддержка численных расчетов**
- **C17** – текущая стабильная версия, bug-fix для C11
- **C23** – предварительная версия, ожидается принятия в октябре 2024

Поддержка компиляторами «последних версий»

- **GCC:** gcc-12.3 по умолчанию C17, C23(**-std=c2x**) частично с gcc-9
- **clang:** C17 с версии clang-7, C23(**-std=c23**) частично с clang-9
- **Microsoft Visual:** C11(std:c11) и C17(std:c17) начиная с VS 2019

Пример: сумма ряда $\cos 0^\circ + \cos 1^\circ + \dots + \cos 90^\circ$

```
#include <stdio.h>
#include <math.h>
int main() {
    const double gr2rad = M_PI/180; /* degrees to radians */
    double sum=0;
    for(int i = 0; i <= 90; i++) { // C99!
        sum += cos(i*gr2rad);
    }
    char* math_fmt = u8" 90°\n Σ cos(i) = %f\ni=0°\n"; // UTF8: C11!
    printf(math_fmt,sum);
}
```

- Файл: `sum_cos.c` (`.c` – стандартное расширение для C)
- Компиляция: `clang -std=gnu11 sum_cos.c -lm`
 - `-std=gnu11` – использовать стандарт языка C11
 - `-lm` – «подключить» математическую библиотеку

 Выполнение: `./a.out`

90°
Σ cos(i) = 57.794325
i=0°

Обзор языка

Имена переменных (идентификаторы)

- Все переменные должны быть декларированы до их использования
- Имена состоят из букв, цифр и «подчёркиваний»; цифра не должна быть первой
- ПРОПИСНЫЕ и строчные буквы разные!
- Число значимых символов: 31(63) внешние(внутренние) имена
- Имена должны отличаться от служебных слов

```
int i,j,k;           // uninitialized variables
char let_A = 'A';
double pi = 3.1415926;
int CamelCaps = 1;   // горбатый регистр
long int Very_long_names_like_this_do_not_make_sense = 0L;
```

Основные типы данных (следующие две лекции)

char	—	один байт, в котором находится один символ
int	—	целое, размер зависит от используемой машины
float	—	число с плавающей точкой одинарной точности
double	—	число с плавающей точкой двойной точности

«Традиционные» комментарии в C

```
/*  
    commented  
    lines  
*/
```

Однострочные комментарии, начиная с C99

```
// one-line comments as in C++  
c = 1 + d /* - 123 */ + k; // совмещение комментариев
```


Оператор sizeof()

`sizeof(obj)` — возвращает размер объекта `obj` в «байтах»

1 байт \equiv `sizeof(char)`

X86-64

- short int = 2
- int = 4
- long int = 8
- long long int = 8
- void* = 8
- float = 4
- double = 8
- long double = 16

ARM64

- short int = 2
- int = 4
- long int = 8
- long long int = 8
- void* = 8
- float = 4
- double = 8
- long double = 8

$\text{sizeof}(\text{long long}) \geq \text{sizeof}(\text{long}) \geq \text{sizeof}(\text{int}) \geq \text{sizeof}(\text{short})$
 $\text{sizeof}(\text{long double}) \geq \text{sizeof}(\text{double}) \geq \text{sizeof}(\text{float})$

Область видимости (scope)

Область видимости объекта (переменной) — часть программы в которой этот объект можно использовать

Глобальная область видимости

Переменные объявленные вне функций доступны в любом месте программы

- все функции по умолчанию глобальные
- глобальные переменные

Локальная область видимости

Внутри блока между { и }:

- переменная существует от места объявления до конца блока
- параметры функции являются локальными переменными в теле функции

Пример: глобальные и локальные переменные

```
int f_counter = 0; /* global variable */
double a = 1.;    /* other global variable */

double function(double x) {
    f_counter++;
    double a = 10;    /* local variable */
    return ret = fabs(x-a); /* |x-a| */
}

int main() {
    function(a); /* using global a */
    int a = 5;   /* local variable a */
    function(a); /* using local a */
    printf("function has been called %d times\n", f_counter);
}
```

Операторы

Присваивание

```
a = b;
```

```
c = d = e = f = 0;
```

```
2 = k; /* Ошибка! */
```

☞ **Lvalue (left-value)** — то, что может стоять в левой части оператора присваивания

Арифметические унарные операторы: $-$, $+$

```
b = -a; /* меняет знак на противоположный */
```

```
c = +a; /* не выполняет никаких действий */
```

Бинарные операторы: +, −, *, / и % (операция деления по модулю)

```
int a = 1, b = 2, c = 3;
a = b + c;           /* a=5 */
b = a / c;           /* b=1 */
c = a % 3;           /* c=2 */
```

👉 Операция % только для целых типов

«Гибридные» операторы: $e1 \odot= e2$; то же, что $e1 = (e1) \odot (e2)$;

```
counter_i += 10;      /* counter_i = counter_i + 10; */
i %= j;               /* i = i % j; i,j - int! */
a *= x + 1;           /* a = a * (x+1); */
a + b *= 2;           /* Ошибка! В левой части должно стоять Lvalue */
val[i1+i2] *= 2;      /* Правильно: val[i1+i2]=val[i1+i2]*2 */
```

Операторы ++ и -- (инкремент/декремент)

++/-- увеличивает/уменьшает величину на единицу

- Префиксные: ++i; --j; (оператор перед переменной)

j = ++i;	=>	i = i + 1;
		j = i;

- Постфиксные: i++; j--; (оператор после переменной)

j = i++;	=>	j = i;
		i = i + 1;

Запрещено или надо избегать:

j = (i*10)++;	/* ERROR: i*10 is not lvalue */
(i++)++;	/* ERROR: i++ is not lvalue */
++i*i++;	/* WARNING: operation on 'i' may be undefined */
f(++i,++i);	/* WARNING: a more general case of the previous */

Операции отношения и логические операции

- Операции отношения: `>=` `>` `<=` `<`
- Операции равенства и неравенства: `==` `!=`
- Отрицание (унарный оператор): `!`
- Логические связки: `&&` («И») `||` («ИЛИ»)
- В C понятие «логический тип» отсутствует. Вместо этого используются целочисленные переменные: `0` – ложь (`FALSE`), `1` – истина (`TRUE`)

```
if( n%2 == 0 && n%3 == 0 ) {  
    printf(" %d divisible by 6 without a remainder\n",n);  
}
```

```
if( b=fun(a) ) { /* it's not a bug */  
    printf(" fun(%d) returns a non-zero value %d\n",a,b);  
}
```

Обратите внимание

- $0 < j < 10$ то же самое, что $(0 < j) < 10$ и это всегда TRUE
☞ Правильная запись: $0 < j \ \&\& \ j < 10$
- Результаты логических выражений можно использовать как обычные целые числа:

```
int minus_one_in_power(int i) { /*  $(-1)^i$  */  
    return -1 + 2*(i%2 == 0);  
}
```
- Часть логической связки с $\&\&$, $||$ может не выполняться
☞ Так делать не надо:

```
if( (a >= 0 &\& a <= 10) &\& ++b > 0 ) {...}
```


Условный (троичный) оператор:

`(condition) ? (expression-true) : (expression-false) ;`

- 1 Вычисляется `condition`
- 2 Если условие истинно (не ноль), то вычисляется `expression-true`, в противном случае `expression-false`
- 3 Вычисленное значение возвращается, значит оператор может стоять в правой части присваивания

```
printf("the number %i are %s\n ",i,(i%2) ? "odd" : "even");
```

```
int max(int a, int b) {  
    return (a > b) ? a : b;  
}
```

Массивы и циклы

Набор из 100 чисел 'double' расположенных последовательно:

```
double a[100];          /* a[0],a[1] ... a[99] */
```

- Доступ по индексу:

```
a[10] = 1.2;
```

```
a[11] = (a[10])++;
```

```
printf(" a[10]=%g a[11]=%g\n",a[10],a[11]); // a[10]=2.2 a[11]=1.2
```

- Нумерация индексов начинается с нуля
- Выход за пределы массива не проверяется

```
a[100] = 0.; // can be run-time error
```

Цикл for

```
for(ExprIni; Cond; ExprIter) {  
    statement;  
}
```

почти то же, что и конструкция

```
ExprIni;        // initial stmt.  
while(Cond) {    // condition  
    statement;   // loop body  
    ExprIter;    // «iteration»  
}
```

```
for(i = 0; i < 10; i++) {...}
```

- ❶ Инициализация перед выполнением цикла: $i = 0$;
- ❷ Условие продолжения цикла (первая инструкция цикла): $i < 10$;
- ❸ Тело цикла: {...}
- ❹ Изменение счётчика (последняя инструкция цикла): $i++$;

Стандартное выражение для обработки массива $a[n]$:

```
for (i = 0; i < n; i++) {  
    a[i] = ...;  
}
```

Цикл `for`, продолжение ...

- Любое из трёх выражений в `for` можно оставить пустым, но разделители (`;;`) всегда должны присутствовать
- «Пустое условие» в `for` равносильно бесконечному циклу

Примеры организации бесконечного цикла:

```
for (;;) {...} /* с циклом for */
```

```
while(1) {...} /* с циклом while: 1 == true */
```

- 👉 Для выхода из бесконечного цикла используют операторы перехода или `return`

C99 допускает декларацию переменных в первом поле `for`

```
for ( int i = 0; i < n; i++ ) { ... }
```

- 👉 Область видимости этой переменной: **только внутри цикла**

Операторы перехода

Оператор `break`

- 1 В цикле: немедленно прекращает выполнение **текущего** цикла
- 2 В операторе `switch`: осуществляет переход на конец оператора `switch`

Оператор `continue`

- Прерывает итерацию цикла и переходит к следующей итерации
👉 в цикле `for` переход на приращение `ExprIter`

```
for (i = 0; i < 5; i++) {  
    for (j = 0; j < 5; j++) {  
        if(j+i>2) break; // по j  
        printf("(%d,%d)",i,j);  
    }  
}
```

> (0,0)(0,1)(0,2)(1,0)(1,1)(2,0)

```
for (i = 0; i < 5; i++) {  
    for (j = 0; j < 5; j++) {  
        if(j+i<6) continue; // по j  
        printf("(%d,%d)",i,j);  
    }  
}
```


> (2,4)(3,3)(3,4)(4,2)(4,3)(4,4)

Оператор goto

```
int i = 0;  
LOOP:  
a += b[i];  
i++;  
if( i < 10 ) goto LOOP; // сравните с циклом do-while
```

Пояснения

- Для оператора `goto` всегда необходима метка – идентификатор с последующим двоеточием: `LOOP:`
- Управление передается на ту точку программы, где стоит метка
- Метка может находиться как до, так и после оператора `goto`

 Старайтесь не использовать `goto`

Старайтесь не использовать goto

- В структурном программировании **goto** рассматривают как крайне нежелательный оператор. *«Качество программного кода обратно пропорционально количеству операторов goto в нём»*
(E.W. Dijkstra "Go to statement considered harmful", 1968)
- «Спагетти-код» – программа содержащая много операторов **goto**

Оправданное применение:

Donald Knuth, "Structured Programming with go to Statements", 1974

```
for( i = 0; i < n; i++ )  
    for( j = 0; j < m; j++ )  
        if( matrix[i][j] == value ) {  
            printf("value %d found in cell (%d,%d)\n",value,i,j);  
            goto end_loop;  
        }  
printf("value %d not found\n",value);  
end_loop: ;
```

Дополнительные слайды

X86-64 vs X86-32

X86-64

- short int = 2
- int = 4
- long int = 8
- long long int = 8
- void* = 8
- float = 4
- double = 8
- long double = 16

X86-32

- short int = 2
- int = 4
- long int = 4
- long long int = 8
- void* = 4
- float = 4
- double = 8
- long double = 12

Оператор if - else

if - else используется при необходимости сделать выбор:

```
if (логическое выражение)
    оператор-1
else
    оператор-2
```

/* необязательная */
/* часть */

Пример: $z = \max(a, b)$

```
/* 1-st variant */
z = a;
if(b > z) z = b;
```

```
/* 2-nd variant */
if (a > b) {
    z = a;
} else {
    z = b;
}
```

Выбор из нескольких вариантов

Переключатель switch

```
switch (expression) {  
    case Const_Int_1:  
        statement;  
    case Const_Int_2:  
        statement;  
    ...  
    default:  
        statement;  
}
```

Сравните с if

```
if (expression-1) {  
    statement;  
} else if (expression-2) {  
    statement;  
} else if (expression-3) {  
    statement;  
} else {  
    statement;  
}
```

Особенности switch

- Значение `expression` должно быть целым (`int`, `char`)
- После перехода на `case` программа выполняется **до конца switch**
- Если выражение не совпадает ни с одним из `case`, переходит на `default`
- `default` может отсутствовать

Циклы

while

```
while (expression) {  
    statement;  
}
```

- цикл выполняется по тех пор пока **expression** не равно нулю
- затем выполняется следующий за **while** оператор

do - while

```
do {  
    statement;  
} while (expression);
```

- **statement** выполняется в любом случае
- вычисляется **expression** и цикл повторяется если результат не нулевой

Точка с запятой ; (*end of statement*)

```
x = 0;           // признак конца оператора
;               // пустой оператор
for(i=0; i<10; sum=i++); // тоже пустой оператор
x = sin(a)       // выражение можно
+               // разбить на
cos(b);          // несколько линий
```

Фигурные скобки { ... } (*compound statement*)

☞ Используются для объединения описаний и операторов в составной оператор или блок

```
if ( a < b ) {
    a++;
    b--;
}
```

Оператор последовательного вычисления (запятая ,)

`expression1 , expression2 , expression3 , ...`

- Выражения, разделённые запятой, вычисляются слева направо
- Возвращается значение правого (последнего) операнда

Пример:

```
for ( i = 1, j = 9; i <= 5; i++, j-- ) { ... }
```

Внимание

- запятые разделяющие переменные в описаниях

```
int i=1, j=0;
```

- запятые разделяющие аргументы функций

```
double f(int x, int y);
```

```
x += f(i, j);
```



Не имеют отношения к операции запятая!