

Справочник языка

Все команды от переменных и типов до ядра без ОС

Tachbit 1.3 · ноябрь 2026

Каждый пример с блоком «Вывод» собран и запущен этой версией компилятора — вывод настоящий.

Содержание

1. Как запускать
2. Программа и запись кода
3. Переменные и типы
4. Действия и сравнения
5. Условия
6. Циклы
7. Списки: list
8. Словарь: dict
9. Текст
10. Функции
11. Свои типы: obj и enum
12. Ошибки: fail, try, or, else err
13. Модули
14. Числа, время, случайность
15. Файлы, папки, система
16. Интернет и JSON
17. Окно и рисование
18. Все ядра и потоки
19. Скорость: слова, которые пишешь сам
20. Без ОС: своё ядро
21. Что ловит компилятор до запуска
22. Все слова языка от A до Z

1 Как запускать

> Команда `tb` ставится один раз: на Mac и Linux `./tools/install.sh`, на Windows `powershell -ExecutionPolicy Bypass -File tools\install.ps1`. После каждого `git pull` запусти установку снова.

Команды tb

Все команды работают из любой папки.

`tb new имя`	новый проект: папка имя/ и в ней main.tb
`tb start имя [слова]`	собрать проект и запустить; без имени — проект в текущей папке; можно дать файл: tb start hello.tb
`tb run файл.tb [слова]`	сделать программу и сразу запустить (программа кладётся в ~/.tachbit/run)
`tb build файл.tb`	сделать программу рядом с файлом (на Windows — файл.exe)
`tb check файл.tb`	только найти ошибки, ничего не собирать
`tb c файл.tb`	показать C-код, который получился
`tb tree файл.tb`	показать дерево программы
`tb tokens файл.tb`	показать токены (слова, на которые разрезан файл)
`tb bare файл.tb`	сделать ядро без ОС: файл.elf для QEMU (Linux и Mac)
`tb app файл.tb [--own]`	программа для Omnium (файл.oapp); --own — свой машинный код, без LLVM
`tb lib файл.tb`	библиотека для C: libфайл.a и файл.h (функции с for c)
`tb ll файл.tb`	показать LLVM IR
`--llvm / --c`	к build и run: собрать через LLVM (по умолчанию, если есть clang) или через C
`tb lang en / tb lang ru`	сообщения компилятора по-английски или по-русски
`tb version`	какая версия стоит

Переменные окружения

Пишутся перед программой: `TB_CORES=1 ./prog`.

`TB_CORES=N`	сколько ядер брать для on all cores (1 — удобно для сравнения)
`TB_GC=0`	выключить уборку памяти
`TB_GC_STATS=1`	печатать каждую уборку: сколько памяти нужно, сколько было, сколько мс
`TB_GC_STRESS=N`	уборка на каждом N-м выделении памяти (очень медленно, для поиска ошибок)
`TB_FPS_FREE=1`	окно без ограничения 60 кадров в секунду (для замеров)
`TB_GPU=0`	рисовать процессором, без видеокарты
`TB_GPU_INFO=1`	сказать при запуске, чем рисуем
`TB_SOUND=0`	без звука; TB_SOUND_FILE=out.wav — звук в файл
`TB_LIB=папка`	где искать библиотеки для use
`TB_MEM_CHECK=1`	строгая проверка памяти в danger
`NO_COLOR=1`	ошибки компилятора без цвета

2 Программа и запись кода

func main()

Программа всегда начинается с `func main()`. Блоки задаются отступом в 4 пробела, скобок { } нет. Комментарий — `//` до конца строки.

```
// моя первая программа
func main()
    print("Привет, Tachbit!")   // печать
```

ВЫВОД

```
Привет, Tachbit!
```

print и show

`print(a, b, ...)` печатает всё через пробел и переводит строку. Списки, dict и obj тоже печатаются. `show(...)` — то же без перевода строки.

```
func main()
    show("раз ")
    show("два ")
    print("три")
    print(1, 2.5, true, "текст", [1, 2, 3])
```

ВЫВОД

```
раз два три
1 2.5 true текст [1, 2, 3]
```

3 Переменные и типы

Переменная

`имя = значение`. Тип Tachbit понимает сам по значению. Имя внутри блока не может повторять имя снаружи — так нельзя случайно спрятать переменную.

```
func main()
    age = 20            // int
    price = 9.5         // dec
    name = "Али"        // text
    ok = true           // bool
    print(name, age, price, ok)
```

ВЫВОД

```
Али 20 9.5 true
```

Простые типы

`int`	целое число, 64 бита: от −9 223 372 036 854 775 808 до 9 223 372 036 854 775 807; выход за край — ошибка с номером строки
`dec`	дробное число (double, 64 бита)
`text`	текст в UTF-8, русские буквы работают
`bool`	true или false

Переменная с типом

Когда начинаешь с нуля или с пустого, тип пишется явно: `int 0`, `dec 0`, `text` (пустой текст), `bool`.

```
func main()
    total = int 0
    avg = dec 0
    log = text
    flag = bool
    total += 5
    avg = 2.5
    log += "ok"
    print(total, avg, log, flag)
```

ВЫВОД

```
5 2.5 ok false
```

const

Значение, которое нельзя менять. Попытка изменить — ошибка ещё при сборке.

```
func main()
    const name = "Tachbit"
    print(name)
```

ВЫВОД

```
Tachbit
```

Числа 0x и деление

`0xFF` — число в шестнадцатеричной записи. `7 / 2` у целых даёт 3 (остаток отбрасывается), `%` — остаток. Нужна дробь — возьми dec: `7.0 / 2`.

```
func main()
    print(0xFF, 7 / 2, 7 % 2, 7.0 / 2)
```

ВЫВОД

```
255 3 1 3.5
```

Превращения типов

Текст + число склеиваются сами. `to_int` и `to_dec` превращают текст в число и могут сломаться (в тексте не число), поэтому с ними `try`, `or` или `else err`.

```
func main()
    n = to_int("42") or 0
    d = to_dec("2.5") or 0.0
    bad = to_int("abc") or -1
    print("n = " + n, d * 2, bad, round(2.6))
```

ВЫВОД

```
n = 42 5 -1 3
```

4 Действия и сравнения

Арифметика

`+ - * /`	сложить, вычесть, умножить, разделить
`%`	остаток от деления
`x += 1, -=, *=, /=`	изменить переменную
`a + b` у текста	склеить; числа, dec и bool внутри превращаются в цифры сами

Сравнения и логика

`== !=`	равно, не равно (у текста — по буквам)
`< > <= >=`	меньше, больше; у текста — по алфавиту: "a" < "b"
`and or not`	и, или, не
`true false`	истина и ложь

Пример

```
func main()
    a = 10
    b = 3
    a += b
    print(a, a > 12 and b < 5, not (a == 13), "a" < "b")
```

ВЫВОД

```
13 true false true
```

5 Условия

if / else if / else

```
func main()
    t = 25
    if t > 30
        print("жарко")
    else if t > 15
        print("тепло")
    else
        print("холодно")
```

ВЫВОД

```
тепло
```

match

Разбор по вариантам: по `enum`, `int` и `text`. В одной строке можно перечислить несколько значений через запятую. `else` — всё остальное. У `enum` без `else` компилятор требует разобрать все варианты.

```
func main()
    code = 404
    match code
        200
            print("ок")
        404, 410
            print("нет страницы")
        else
            print("другое")
```

ВЫВОД

```
нет страницы
```

6 Циклы

count

Счётчик. Последнее число тоже входит.

```
func main()
    count i from 1 to 5
        show(i + " ")
    print("")
```

ВЫВОД

```
1 2 3 4 5
```

each

Пройти по списку, по буквам текста, по ключам dict или по каналу.

```
func main()
    each n in [3, 8, 1]
        show(n + " ")
    each ch in "abc"
        show(ch + "-")
    print("")
```

ВЫВОД

```
3 8 1 a-b-c-
```

repeat

`repeat N times` — N раз. `repeat while условие` — пока условие верно.

```
func main()
    repeat 3 times
        show("ку ")
    x = 100
    repeat while x > 10
        x = x / 2
    print(x)
```

ВЫВОД

```
ку ку ку 6
```

skip и stop

`skip` — перейти к следующему шагу. `stop` — выйти из цикла.

```
func main()
    count i from 1 to 10
        if i % 2 == 0
            skip
        if i > 7
            stop
        show(i + " ")
    print("")
```

ВЫВОД

```
1 3 5 7
```

7 Списки: list

Создать и менять

`[3, 8, 1]` — готовый список, `list text` — пустой. Номера с 0. Номер за пределами останавливает программу с понятной ошибкой, а не портит память.

```
func main()
    nums = [3, 8, 1]
    names = list text
    names.add("Аня")
    names.add("Бек")
    nums[0] = 5
    last = nums.pop()
    print(nums, last, len(names), names[1])
    sort(names)
    nums.clear()
    print(len(nums))
```

ВЫВОД

```
[5, 8] 1 2 Бек
0
```

Всё про список

`list int, list text, list Item`	пустой список нужного типа
`list list int`	список списков
`xs[i], xs[i] = v`	взять и положить по номеру
`xs.add(v)`	положить в конец
`xs.pop()`	забрать последний (у пустого — остановка с ошибкой)
`xs.clear()`	сделать пустым, память отдаётся сразу
`len(xs)`	сколько элементов
`sort(xs)`	расставить по порядку на месте (int, dec или text)
`join(xs, ", ")`	склеить list text в один текст
`each x in xs`	пройти по всем

8 Словарь: dict

dict

Пары «ключ → значение». Ключ — `text` или `int`. Порядок при `each` и печати — как добавляли.

```
func main()
    phones = {"Али": "555", "Вали": "777"}
    counts = dict text int
    each w in split("да нет да да", " ")
        counts[w] += 1
    print(phones["Али"], counts["да"], counts.has("может"))
    phones.remove("Вали")
    print(phones.keys(), counts.values(), len(counts))
    each k in counts
        print(k, counts[k])
```

ВЫВОД

```
555 3 false
[Али] [3, 1] 2
да 3
нет 1
```

Всё про dict

`dict text int`	пустой словарь: тип ключа, тип значения
`{"a": 1}`	готовый словарь
`d[k], d[k] = v, d[k] += 1`	взять, положить, изменить (нет ключа при += — начинает с 0)
`d.has(k)`	есть ли ключ
`d.remove(k)`	убрать ключ
`d.keys(), d.values()`	списки ключей и значений
`d.clear(), len(d)`	очистить, сколько пар
`each k in d`	пройти по ключам

9 Текст

Буквы и куски

`s[i]` — знак по номеру (с 0), `cut(s, a, b)` — с a-го по b-й знак включительно, `len(s)` — длина. Всё это считает буквы, русские тоже: `len("чай")` = 3, `s[0]` у русского слова — целая первая буква. Байты UTF-8 нужны редко (файлы, сеть): для них `byte_len`, `byte_at`, `byte_char`.

```
func main()
    s = "Tachbit"
    print(s[0], cut(s, 1, 3), len(s), len("чай"))
    print(code("A"), char(66), "чай"[1], byte_len("чай"))
```

ВЫВОД

```
T ach 7 3
65 B a 6
```

Функции для текста

`len(s)`	сколько букв (русская буква — одна)
`cut(s, a, b)`	кусок с a-й по b-ю букву
`split(s, ",")`	разрезать в list text
`join(xs, ", ")`	склеить list text
`find(s, "x")`	номер первого вхождения или −1
`replace(s, "a", "b")`	заменить все
`upper(s), lower(s)`	БОЛЬШИЕ / маленькие буквы
`trim(s)`	убрать пробелы по краям
`starts_with(s, x), ends_with(s, x)`	начинается / кончается на
`code("A") → 65, char(66) → "B"`	буква ↔ номер
`to_int(s), to_dec(s)`	текст в число (может сломаться)
`byte_len(s), byte_at(s, i), byte_char(n)`	байты UTF-8: сколько байт, байт по номеру, текст из одного байта

Пример

```
func main()
    s = "  чай, кофе, сок  "
    parts = split(trim(s), ", ")
    print(parts, find(s, "кофе"), upper("tachbit"))
    print(replace("a-b-c", "-", "+"), starts_with("tachbit", "tach"), join(parts, "/"))
```

ВЫВОД

```
[чай, кофе, сок] 7 TACHBIT
a+b+c true чай/кофе/сок
```

10 Функции

func и return

Параметры пишутся с типом. Тип ответа Tachbit понимает сам. Вызывать можно по порядку или с именами — тогда порядок любой.

```
func area(w: int, h: int)
    return w * h

func main()
    print(area(3, 4), area(h: 3, w: 5))
```

ВЫВОД

```
12 15
```

gives: именованные выходы

`gives` называет выходы. Их может быть несколько. Выход-список уже пустой, его можно сразу наполнять, и `return` в конце не нужен. Ответы можно принять в имена, в ячейки списка и в поля obj: `xs[0], p.y = f()`.

```
func divide(a: int, b: int) gives q: int, r: int
    q = a / b
    r = a % b

func evens(n: int) gives xs: list int
    count i from 1 to n
        if i % 2 == 0
            xs.add(i)

func main()
    q, r = divide(17, 5)
    print(q, r, evens(10))
```

ВЫВОД

```
3 2 [2, 4, 6, 8, 10]
```

link: ярлык

Ярлык — второе имя для того же места. Меняешь ярлык — меняется оригинал. Объекты и списки и так передаются ярлыками: функция меняет тот же список.

```
obj Hero
    hp: int

func hit(link hp, damage: int)
    hp -= damage

func main()
    h = Hero(hp: 100)
    hit(h.hp, 30)
    life: link h.hp
    life += 5
    print(h.hp)
```

ВЫВОД

```
75
```

11 Свои типы: obj и enum

obj

Свой тип с качествами. Пишется снаружи функций. Пропущенные качества будут 0, "" или пустой список.

```
obj User
    name: text
    age: int
    tags: list text

func main()
    u = User(name: "Али", age: 20)
    u.age += 1
    u.tags.add("admin")
    print(u.name, u.age, u.tags)
    print(u)
```

ВЫВОД

```
Али 21 [admin]
User(name: Али, age: 21, tags: [admin])
```

obj ... by columns

Тот же obj, но список таких объектов лежит в памяти колонками: все x подряд, все mass подряд. Цикл, который читает одно качество, идёт в разы быстрее. `.pop()` у такого списка нет; `add` кладёт копию.

```
obj P by columns
    x: dec
    mass: dec

func main()
    ps = list P
    count i from 1 to 3
        ps.add(P(x: i * 1.0, mass: 2.0))
    total = dec 0
    each p in ps
        total += p.mass
    print(total)
```

ВЫВОД

```
6
```

enum

Набор вариантов. Значение — `Тип.вариант`. `print` печатает имя варианта, `==` сравнивает.

```
enum Status
    new
    paid
    sent

func label(s: Status) gives r: text
    match s
        new
            return "новый"
        paid, sent
            return "в работе"
    return ""

func main()
    s = Status.paid
    print(s, label(s), s == Status.paid)
```

ВЫВОД

```
paid в работе true
```

12 Ошибки: fail, try, or, else err

fail

`fail "текст"` — функция заканчивается ошибкой. Такую функцию нельзя вызвать просто так: компилятор попросит решить, что делать с ошибкой.

`x = f() else err`	свой блок на ошибку ниже с отступом; в err — текст ошибки
`x = f() or 0`	при ошибке взять запасное значение
`x = try f()`	передать ошибку выше; в main программа остановится с текстом ошибки

Пример

Сломаться могут: `read_file`, `write_file`, `input`, `to_int`, `to_dec`, `list_dir`, `make_folder`, `move`, `remove`, `window_open`, `http_get`, `json_get` и твои функции, где есть `fail` или `try`.

```
func buy(left: int, many: int)
    if many > left
        fail "осталось только " + left
    return left - many

func main()
    a = buy(10, 3) else err
        print("не вышло:", err)
    b = buy(2, 9) else err
        print("не вышло:", err)
    c = buy(2, 9) or 0
    print(a, c)
```

ВЫВОД

```
не вышло: осталось только 2
7 0
```

13 Модули

use, modpub, modpri

`use money` подключает `money.tb` из той же папки. `modpub` — функцию видно другим файлам, `modpri` (или без слова) — только своему. Вызов: `money.with_tax(x)` или просто `with_tax(x)`. `main` — только в главном файле.

```
// money.tb
modpub func with_tax(sum: int)
    return sum + tax(sum)

modpri func tax(sum: int)
    return sum * 12 / 100

// main.tb
use money

func main()
    print(money.with_tax(1000))
```

14 Числа, время, случайность

Функции

`abs(x)`	модуль числа
`min(a, b), max(a, b)`	меньшее и большее
`round(x)`	округлить dec до int
`sqrt(x)`	квадратный корень
`sin, cos, tan (x)`	синус, косинус, тангенс; угол в радианах
`asin, acos, atan (x), atan2(y, x)`	угол по числу; atan2 — угол направления на точку
`pi()`	3.14159...
`pow(x, n), exp(x), log(x)`	степень, e в степени x, натуральный логарифм
`floor(x), ceil(x)`	вниз и вверх до целого
`random(a, b)`	случайное целое, оба конца входят
`time_ms()`	миллисекунды (для замеров)
`wait_ms(n)`	подождать n мс
`date_now()`	"2026-11-17 12:30:00" — время компьютера
`unix_time()`	секунды с 1970 года

Пример

```
func main()
    print(abs(-7), min(3, 9), max(3, 9), round(2.6), sqrt(16.0))
    print(floor(2.7), ceil(2.1), pow(2.0, 10.0), round(sin(pi() / 2)), round(atan2(1.0,
1.0) * 180 / pi()))
    d = random(1, 6)
    print(d >= 1 and d <= 6, unix_time() > 0, len(date_now()))
```

ВЫВОД

```
7 3 9 3 4
2 3 1024 1 45
true true 19
```

15 Файлы, папки, система

Функции

`read_file("a.txt")`	прочитать весь файл в текст (может сломаться)
`write_file("a.txt", s)`	записать текст в файл (может сломаться)
`file_exists(p)`	есть ли файл или папка
`file_size(p)`	размер в байтах или −1
`is_folder(p)`	это папка?
`list_dir(p)`	имена внутри; у папок в конце /
`make_folder(p)`	создать папку
`move(откуда, куда)`	переместить или переименовать
`remove(p)`	удалить файл или пустую папку
`input()`	строка с клавиатуры (может сломаться: ввод закончился)
`args()`	list text — слова после tb run файл.tb
`env("HOME")`	переменная окружения ("" если нет)
`run("команда")`	запустить команду терминала, ответ — код выхода
`exit(код)`	закончить программу с кодом

Пример

```
func main()
    try write_file("note.txt", "строка 1\nстрока 2\n")
    s = try read_file("note.txt")
    print(len(split(trim(s), "\n")), file_exists("note.txt"), file_size("note.txt"))
    try remove("note.txt")
    print(file_exists("note.txt"))
```

ВЫВОД

```
2 true 30
false
```

16 Интернет и JSON

Функции

`http_get` работает через curl. `json_get` берёт значение по пути: ключи и номера через точку; строка приходит без кавычек, объект и список — куском JSON.

`http_get("https://...")`	текст страницы (может сломаться)
`json_get(json, "users.0.name")`	значение по пути (может сломаться)
`json_text(s)`	текст как строка JSON в кавычках
`net_listen(порт)`	свой сервер: слушать порт (может сломаться); 0 — любой свободный
`net_port(s)`	какой порт слушаем
`net_accept(s)`	дождаться, кто подключится (может сломаться)
`net_connect("host", порт)`	подключиться к серверу (может сломаться)
`net_send(c, текст), net_receive(c)`	отправить / получить кусок (может сломаться)
`net_close(c)`	закрыть связь

Пример

```
func main()
    j = "{\"users\": [{\"name\": \"Ann\", \"age\": 30}]}"
    name = try json_get(j, "users.0.name")
    age = try json_get(j, "users.0.age")
    print(name, age, json_text("say \"hi\""))
```

ВЫВОД

```
Ann 30 "say \"hi\""
```

17 Окно и рисование

Окно: на Mac — Cocoa, на Linux — X11, на Windows — Win32. Рисует видеокарта (Metal на Mac, Vulkan на Linux и Windows); нет видеокарты — процессор, кадр тот же. Точка (0, 0) — левый верхний угол, y растёт вниз.

Функции

`window_open(ш, в, "заголовок")`	открыть окно (может сломаться, с try)
`window_alive()`	следующий кадр: показать нарисованное, принять клавиши, 60 кадров/с; false — окно закрыли
`rgb(r, g, b)`	цвет, каждая часть от 0 до 255
`clear_window(цвет)`	залить всё окно
`draw_pixel(x, y, цвет)`	точка
`draw_rect(x, y, ш, в, цвет)`	прямоугольник
`draw_circle(x, y, радиус, цвет)`	круг
`draw_line(x1, y1, x2, y2, цвет)`	линия
`draw_text(x, y, "текст", размер, цвет)`	текст, буква 8×16, русский тоже
`key_down("left")`	нажата ли клавиша: left right up down space enter escape backspace tab shift, буквы, цифры
`mouse_x(), mouse_y(), mouse_down()`	мышь
`rgba(r, g, b, a)`	полупрозрачный цвет: a 255 — сплошной, 128 — наполовину
`draw_triangle(x1, y1, x2, y2, x3, y3, цвет)`	треугольник
`draw_round(x, y, ш, в, r, "all", цвет)`	скруглённый прямоугольник
`blur_rect, draw_shadow`	матовое стекло, мягкая тень
`frame_ms(), fps(), set_fps(n)`	длина прошлого кадра в мс (для скорости движения), кадров в секунду, своя частота
`window_w(), window_h()`	размер окна; окно можно тянуть мышью
`window_sharp(), window_scale()`	чёткое окно на Retina, во сколько раз больше точек
`window_new, window_use, window_shown, window_close`	ещё настоящие окна системы; window_use(0) — снова главное
`window_fullscreen(true)`	на весь экран
`mouse_wheel(), mouse_back(), mouse_forward()`	колёсико, боковые кнопки
`mouse_lock(true), mouse_dx(), mouse_dy()`	мышь для шутера: курсор спрятан, сдвиг с прошлого кадра

`typed_text()`	набранные буквы с прошлого кадра
`pad_count(), pad_down(n, "a"), pad_axis(n, "lx")`	геймпад: кнопки и стики
`clipboard_get(), clipboard_set(s), set_cursor("hand")`	буфер обмена, вид курсора
`load_image(p), draw_image(i, x, y)`	картинки PNG, JPEG, GIF, BMP (может сломаться)
`image_w, image_h, image_scaled, draw_image_alpha, screen_image`	размер, масштаб, мягкие края, снимок кадра
`load_font(p), draw_font(f, x, y, s, размер, цвет), text_width`	шрифты TTF

3D

Мир из сеток треугольников: x вправо, y вверх, z вперёд. Ближнее закрывает дальнее само.

`mesh_new(), mesh_add(m, 9 чисел, цвет)`	пустая сетка, треугольник в неё
`draw_mesh(m, x, y, z, поворот)`	нарисовать сетку в мире
`draw_mesh_rot(m, x, y, z, поворот, наклон, крен)`	с полным поворотом
`camera(x, y, z, куда_x, куда_y, куда_z)`	где глаз и куда смотрит
`sunlight(x, y, z, фон)`	солнце; sunlight(0, 0, 0, 0) — выключить
`point_light, spot_light, light_off, light_shadow`	лампы (до 8) и фонари, тени от ламп
`fog(цвет, откуда, докуда), shadows(размер)`	туман, тени от солнца
`mesh_image, mesh_add_uv`	картинка на сетке
`mesh_smooth, mesh_shine, mesh_glow, mesh_alpha`	плавный свет, блеск, свечение, стекло и вода
`bone_add, bone_box, bone_weight, bone_turn, bone_move`	кости: сетка сгибается
`mesh_clear, mesh_count, mesh_free`	стереть сетку, сколько треугольников, отдать память
`mesh_simplify, mesh_lod, mesh_far`	вдали проще, дальше не рисовать
`gpu func, mesh_shader, shader_set, screen_shader`	свои шейдеры на Tachbit
`use model, use shaders, use stream`	модели OBJ/glTF/GLB, готовые шейдеры, подгрузка большого мира

Звук

Свой микшер: до 64 звуков сразу, звук в 3D слышно слева и справа.

`load_sound(p)`	загрузить WAV (может сломаться)
`play_sound(s), loop_sound(s), play_tone(гц, мс)`	играть один раз, по кругу, простой тон
`play_sound_at, loop_sound_at, move_sound, listener`	звук в 3D и где уши
`stop_sound, sound_playing, sound_volume, sound_pitch, master_volume`	управление

Пример

```
func main()
    try window_open(400, 300, "Привет")
    x = 200.0
    repeat while window_alive()
        dt = frame_ms() / 1000
        if key_down("left")
            x -= 200 * dt
        if key_down("right")
            x += 200 * dt
        clear_window(rgb(20, 20, 30))
        draw_circle(round(x), 150, 20, rgb(47, 209, 191))
        draw_text(10, 10, "Стрелки двигают круг", 1, rgb(255, 255, 255))
```

18 Все ядра и потоки

on all cores и shared

Слова в конце строки цикла — и он идёт сразу на всех ядрах. `shared` — общая переменная: каждое ядро копит своё через `+=` / `-=`, в конце всё складывается. Компилятор сам проверяет, что ядра не мешают друг другу: внешние переменные внутри только читаются, внешние списки и obj менять нельзя (свой элемент `x` в `each x in users on all cores` — можно), нельзя `return`, `fail`, `try`, `stop`.

```
func main()
    sales = list int
    count i from 1 to 1000000
        sales.add(i * 7919 % 1000)
    shared big = int 0
    each s in sales on all cores
        if s > 900
            big += 1
    print("крупных:", big)
```

ВЫВОД

```
крупных: 99000
```

background и .wait

Функция уходит работать в свой поток, ответ — номерок. `.wait` ждёт и забирает результат. Функция получает только int, dec, bool, text (или списки через move), без link и fail.

```
func sum_to(n: int)
    s = 0
    count i from 1 to n
        s += i
    return s

func main()
    a = background sum_to(1000)
    b = background sum_to(2000)
    print(a.wait + b.wait)
```

ВЫВОД

```
2501500
```

move

`background f(move xs)` — список уходит в фон целиком, без копии. После этого пользоваться `xs` нельзя, пока не дашь новое значение.

```
func total(xs: list int)
    s = 0
    each x in xs
        s += x
    return s

func main()
    data = list int
    count i from 1 to 100
        data.add(i)
    a = background total(move data)
    print(a.wait)
```

ВЫВОД

```
5050
```

channel

Очередь между потоками. `.send(x)` кладёт в конец, `.receive()` берёт из начала и ждёт, `.close()` — больше не пишут, `each x in ch` — пока не закрыт и не пуст. По каналу идут int, dec, bool, text, enum.

```
func worker(jobs: channel int, results: channel text)
    each n in jobs
        results.send("готово " + n)

func main()
    jobs = channel int
    results = channel text
    w = background worker(jobs, results)
    count i from 1 to 3
        jobs.send(i)
    jobs.close()
    count i from 1 to 3
        print(results.receive())
```

ВЫВОД

```
готово 1
готово 2
готово 3
```

lock и unlock

`lock имя` — блок, который в один момент выполняет только кто-то один. `unlock имя` отпускает раньше конца блока. Внутри lock нельзя return, fail, try. Компилятор проверяет, что замки везде берут в одном порядке. `shared` снаружи функций — общее число для всей программы (int, dec, bool).

```
shared total = int 0

func add_many(n: int)
    count i from 1 to n
        lock m
            total += 1
    return n

func main()
    a = background add_many(100000)
    b = background add_many(100000)
    done = a.wait + b.wait
    print(done, total)
```

ВЫВОД

```
200000 200000
```

19 Скорость: слова, которые пишешь сам

temp

Всё, что создано в блоке, удаляется разом в конце блока. Наружу можно менять только int, dec, bool.

```
func main()
    total = 0
    repeat 3 times
        temp
            lines = list text
            count i from 1 to 1000
                lines.add("строка " + i)
            each s in lines
                total += len(s)
    print(total)
```

ВЫВОД

```
29679
```

each ... by vec

SIMD: процессор складывает несколько чисел одной командой. Внутри внешние числа только копят через += или -=.

```
func main()
    nums = list int
    count i from 1 to 1000
        nums.add(i)
    total = 0
    each x in nums by vec
        total += x * x
    print(total)
```

ВЫВОД

```
333833500
```

each piece in data by N

Работа кусками по N элементов, пока кусок в быстром кэше. `piece` — окно в data, не копия.

```
func main()
    data = list int
    count i from 1 to 10
        data.add(i)
    each piece in data by 4
        print(len(piece), piece[0])
```

ВЫВОД

```
4 1
4 5
2 9
```

together

Несколько независимых чтений `d[key]` и `xs[i]` идут к памяти одновременно. Переменные видны и после блока.

```
func main()
    ages = {"a": 1, "b": 2}
    nums = [10, 20, 30]
    together
        x = ages["a"]
        y = ages["b"]
        z = nums[2]
    print(x, y, z)
```

ВЫВОД

```
1 2 30
```

Что компилятор делает сам

Писать ничего не надо: память шага цикла и функции отдаётся сама; уборка памяти (1.1); адресный dict int; заказ памяти наперёд в циклах по плану; горячие и холодные качества obj; склеивание текста на месте; list bool по одному биту; списки int по 4 байта, если числа доказанно малы (1.1); проверка края списка один раз до цикла, где это можно доказать; на своём компьютере tb run собирает под твой процессор.

20 Без ОС: своё ядро

> `tb bare файл.tb` делает ядро (.elf), которое загружается без операционной системы (QEMU или GRUB). Внутри работают print, list, dict, obj, циклы, background.

Биты (везде)

`0xB8000`	шестнадцатеричное число
`bit_and(a, b), bit_or, bit_xor`	побитовые И, ИЛИ, исключающее ИЛИ
`bit_not(a)`	побитовое НЕ
`shift_left(a, n), shift_right(a, n)`	сдвиг битов

danger

Блок, где можно трогать железо. Снаружи этих слов — ошибка сборки. Внутри проверок нет совсем — так задумано.

`read8/16/32/64(адрес)`	прочитать память
`write8/16/32/64(адрес, число)`	записать в память
`address_of(список)`	адрес данных списка
`asm("cli")`	команда процессора как есть
`new_mem, grow_mem, free_mem, copy_mem, fill_mem`	память руками
`zone_new, zone_take, zone_used, zone_clear, zone_free`	зона: много кусков, одна уборка
`shape … at`	раскладка байтов (u8…u64, i8…i64, bits, bytes, big)
`syscall(n, …)`	прямой вызов ядра ОС
`use c / for c`	звать С / дать функцию С
`port_in8/16/32(порт), port_out8/16/32(порт, n)`	порты (только в ядре)
`page_new(), page_free(адрес)`	страница памяти 4 КБ (только в ядре)
`interrupts_on(), interrupts_off()`	прерывания (только в ядре)

Прерывания и функции ядра

`func f() on timer`	процессор зовёт её 1000 раз в секунду
`func f() on keyboard`	при каждом нажатии клавиши
`func f() on interrupt N`	прерывание 32..47
`uptime_ms()`	мс с запуска ядра
`wait_ms(n)`	подождать
`read_key()`	нажатая буква или ""
`yield()`	отдать процессор другой задаче
`background в ядре`	задача со стеком 64 КБ, переключение каждые 10 мс

Пример

```
shared ticks = int 0

func on_tick() on timer
    ticks += 1

func main()
    danger
        write16(0xB8000, 0x0E41)
    wait_ms(100)
    print("тиков:", ticks)
```

21 Что ловит компилятор до запуска

Защита

функция без ответа на ошибку	вызов f() с fail без try / or / else err
гонка на ядрах	функция меняет список снаружи on all cores, в том числе через другое имя
shared без замка	+= к общей переменной из background без lock
вечный замок	lock m внутри lock m; два замка в разном порядке
ярлык в пустоту	e: link xs[0], потом xs.add(...)
удаление ключей во время each	each k in d → d.remove(k)
забытый вариант	match по enum без всех вариантов и без else
спрятанное имя	имя внутри блока повторяет имя снаружи
move	второе имя у списка или использование после move

Во время работы

Деление на ноль, номер за пределами списка, нет ключа в dict, pop у пустого списка, слишком глубокая рекурсия — программа останавливается с понятным текстом и номером строки, а не портит память. Переполнение int тоже ловится. Честно не ловится: всё внутри danger (там числа заворачиваются) и переполнение в on all cores и by vec.

22 Все слова языка от A до Z

Ключевые слова

`and`	и
`background`	работа в своём потоке
`bool`	true / false
`by columns`	obj колонками
`by vec`	SIMD-цикл
`by N`	цикл кусками
`channel`	очередь между потоками
`const`	нельзя менять
`count ... from ... to`	счётчик
`danger`	блок для железа
`dec`	дробное число
`dict`	словарь
`each ... in`	по списку, тексту, dict, каналу
`else`	иначе; else err — свой ответ на ошибку
`enum`	набор вариантов
`fail`	закончить функцию ошибкой
`false`	ложь
`func`	функция
`gives`	именованные выходы функции
`if`	если
`in`	в (each)
`int`	целое число
`link`	ярлык
`list`	список
`lock`	замок
`match`	разбор вариантов
`modpri`	видно только в своём файле
`modpub`	видно другим файлам

`move`	отдать список без копии
`not`	не
`obj`	свой тип
`on all cores`	на всех ядрах
`on timer / keyboard / interrupt`	прерывание (ядро)
`or`	или; f() or 0 — запасное значение
`repeat ... times / while`	N раз / пока верно
`return`	отдать ответ
`shared`	общая переменная
`skip`	следующий шаг цикла
`stop`	выйти из цикла
`temp`	временная память блока
`text`	текст
`together`	чтения одновременно
`true`	истина
`try`	передать ошибку выше
`unlock`	отпустить замок раньше
`use`	подключить модуль
`shape … at`	раскладка байтов
`use c / for c`	связь с C
`gpu func`	шейдер на Tachbit

Встроенные функции

`abs`	модуль числа
`address_of`	адрес списка (danger)
`args`	слова запуска
`asm`	команда процессора (danger)
`bit_and / or / xor / not`	биты
`bone_add / bone_turn`	кости
`byte_len / byte_at / byte_char`	байты текста
`camera`	камера 3D
`char`	номер → буква
`clear_window`	залить окно
`code`	буква → номер
`cut`	кусок текста
`date_now`	дата и время
`draw_circle / line / pixel / rect / text`	рисование
`draw_triangle / draw_round`	треугольник, скругление
`ends_with`	кончается на
`env`	переменная окружения
`exit`	закончить программу
`file_exists`	есть ли файл
`file_size`	размер файла
`find`	найти в тексте
`floor / ceil`	вниз / вверх до целого
`frame_ms / fps / set_fps`	кадры
`http_get`	скачать страницу
`input`	строка с клавиатуры
`interrupts_on / off`	прерывания (ядро)
`is_folder`	это папка?
`join`	склеить список
`json_get`	значение из JSON

`json_text`	текст в JSON
`key_down`	клавиша нажата?
`len`	длина
`list_dir`	что в папке
`load_font / draw_font / text_width`	шрифты
`load_image / draw_image`	картинки
`load_sound / play_sound / loop_sound / play_tone`	звук
`lower`	маленькие буквы
`make_folder`	создать папку
`max`	большее
`mesh_new / mesh_add / draw_mesh / draw_mesh_rot`	3D-сетки
`mesh_shader / shader_set / screen_shader`	свои шейдеры
`min`	меньшее
`mouse_wheel / lock / dx / dy`	мышь
`mouse_x / y / down`	мышь
`move(a, b)`	переместить файл
`net_listen / net_accept / net_connect`	сеть: сервер и клиент
`net_send / net_receive / net_close / net_port`	сеть: обмен
`new_mem / free_mem / copy_mem`	память руками (danger)
`pad_count / pad_down / pad_axis`	геймпад
`page_new / page_free`	страница памяти (ядро)
`pi`	3.14159…
`port_in / port_out`	порты (ядро)
`pow / exp / log`	степень, экспонента, логарифм
`print`	печать
`random`	случайное число
`read8..64 / write8..64`	память (danger)

`read_file`	прочитать файл
`read_key`	буква (ядро)
`remove`	удалить файл
`replace`	заменить
`rgb`	цвет
`rgba`	полупрозрачный цвет
`round`	округлить
`run`	команда терминала
`shift_left / right`	сдвиг битов
`show`	печать без перевода строки
`sin / cos / tan / asin / acos / atan / atan2`	углы
`sort`	по порядку
`split`	разрезать текст
`sqrt`	корень
`starts_with`	начинается с
`sunlight / point_light / spot_light / fog / shadows`	свет, туман, тени
`syscall`	вызов ядра ОС (danger)
`time_ms`	миллисекунды
`to_dec`	текст → dec
`to_int`	текст → int
`trim`	убрать пробелы
`typed_text`	набранные буквы
`unix_time`	секунды с 1970
`upper`	БОЛЬШИЕ буквы
`uptime_ms`	мс с запуска (ядро)
`wait_ms`	подождать
`window_alive`	следующий кадр
`window_new / use / shown / close / fullscreen`	окна

`window_open`	открыть окно
`window_w / window_h`	размер окна
`write_file`	записать файл
`yield`	отдать процессор (ядро)
`zone_new / zone_take`	зоны (danger)

Методы

`xs.add(v)`	list: в конец
`xs.pop()`	list: забрать последний
`xs.clear()`	list и dict: очистить
`d.has(k)`	dict: есть ключ?
`d.remove(k)`	dict: убрать ключ
`d.keys(), d.values()`	dict: списки
`ch.send(x), ch.receive(), ch.close()`	channel
`a.wait`	background: дождаться ответа