TACHBIT 1.3 · СПРАВОЧНЫЙ КУРС

Язык Tachbit от первой строки до своего ядра

Все слова, команды и встроенные функции языка: что делает, как пишется, проверенный пример с настоящим выводом и задание. 28 уроков и полный указатель.

`func` `if` `each` `count` `list` `dict` `obj` `enum` `match` `fail`
`try` `link` `background` `channel` `on all cores` `temp` `danger`
`shape` `use c` `window_open` `draw_mesh` `play_sound` `tb bare`

Версия от 25 ноября 2026 · Ядро языка заморожено с 1.2, в 1.3 добавлены только новые слова · Каждый пример проверен компилятором Tachbit 1.3

Как устроена эта книга

Первый курс («Как думает программа») учил логике: как устроены приложения, игры и системы. Эта книга — про сам язык. В каждом уроке одна тема и все её слова: что слово делает, как пишется и где ошибиться. Код набирайте руками, запускайте `tb run main.tb` и сравнивайте вывод с книгой.

Как читать уроки. Сверху — цель и список слов урока. Потом объяснения с примерами. Под каждым примером — его настоящий вывод: все примеры запущены компилятором Tachbit 1.3. Примеры с окном, звуком и своим ядром проверены компилятором, но запускать их нужно на своём компьютере. В рамке «Запомнить» — правила, на которых чаще ошибаются. В конце — задание.

В конце книги три указателя: все слова языка, все встроенные функции и частые сообщения компилятора.

Содержание

	Часть 1. Основа языка — Значения, решения, циклы, текст, списки, словари, функции, свои типы, ошибки.
1	Первая программа
2	Переменные и типы
3	Сравнения и решения
4	Повторение: циклы
5	Текст
6	Списки
7	Словари: dict
8	Функции
9	Свои типы: obj
10	Ярлыки: link
11	Варианты: enum и match
12	Ошибки: fail, try, or, else err
	Часть 2. Настоящие программы — Модули, числа и время, файлы, сеть, память.
13	Модули и библиотеки
14	Числа, время и случайность
15	Файлы, папки, ввод
16	Интернет: HTTP, JSON и свой сервер
17	Память
	Часть 3. Несколько ядер и скорость — Фон, замки, каналы, все ядра, SIMD.
18	Работа в фоне: background, shared, lock
19	Каналы, move и все ядра
20	Скорость одним словом
	Часть 4. Окна, игры, звук — 2D, 3D, звук, кнопки и поля.
21	Окно и 2D-рисование
22	3D
23	Звук
24	Приложения с кнопками: lib/ui

	Часть 5. Система — Память руками, байты, C, своё ядро, команды.
25	Железо: danger, память руками, shape, zone
26	Связь с C и системой
27	Своё ядро: tb bare
28	Команды tb и настройки
	Указатели: слова языка · встроенные функции · сообщения компилятора

1 Первая программа

Часть 1

Цель: Понять, из чего состоит файл .tb и как его запустить.

`func` `main` `print` `show` `//`

Файл и запуск

Программа на Tachbit — это текстовый файл с расширением `.tb`. Работа начинается с функции `main`. Всё, что относится к функции, пишется ниже с отступом в 4 пробела. Отступ здесь не украшение, а часть языка: он показывает, где блок начинается и где кончается. Фигурных скобок, `then` и `end` в Tachbit нет.

```
// всё после двух косых черт — комментарий, компьютер его не читает
func main()
    print("Привет, Tachbit!")
    print("2 + 3 =", 2 + 3)       // print пишет через пробел и переводит строку
    show("без ")                  // show — то же, но без перевода строки
    show("перевода")
    print("")
```

ВЫВОД

```
Привет, Tachbit!
2 + 3 = 5
без перевода
```

Команды терминала

`tb run файл.tb` собирает программу и сразу запускает. `tb check файл.tb` только ищет ошибки, ничего не запуская. `tb build файл.tb` кладёт готовую программу рядом с файлом. Полный список команд — в последнем уроке.

```
tb run hello.tb        # собрать и запустить
tb check hello.tb      # только проверить
tb build hello.tb      # сделать программу hello рядом с файлом
```

Запомнить

- Одна программа — одна `main` в главном файле.
- Отступ — ровно 4 пробела на уровень. Табы лучше не смешивать с пробелами.
- Ошибки компилятор объясняет по-русски и с номером строки ещё до запуска.

Задание. Напишите программу, которая печатает ваше имя, город и год рождения тремя строками, а четвёртой строкой — сколько вам лет в 2026 году (посчитайте вычитанием).

2 Переменные и типы

Часть 1

Цель: Научиться хранить данные: числа, дроби, текст и да/нет.

`int` `dec` `text` `bool` `true` `false` `const` `=` `+= -= *= /=` `%`

Четыре основных типа

Переменная — это имя для значения. Тип Tachbit узнаёт сам по первому значению. Типов четыре: `int` (целое число), `dec` (дробь), `text` (текст) и `bool` (`true` или `false`). Тип переменной потом не меняется: если в `x` лежит число, текст туда положить нельзя.

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

    total = int 0             // тип можно написать явно
    rate = dec 0              // дробь, пока 0
    note = text               // пустой текст ""
    const SHOP = "Нектар"     // const — нельзя менять
    print(total, rate, note, SHOP)
```

ВЫВОД

```
Алия 20 9.5 true
0 0  Нектар
```

Арифметика

Плюс, минус, умножение, деление и остаток `%`. У целых деление отбрасывает дробную часть: `7 / 2` даёт 3. Нужна дробь — пусть одно из чисел будет `dec`. Короткие записи `+=`, `-=`, `*=`, `/=` меняют переменную на месте.

```
func main()
    print(7 / 2, 7 % 2, 7.0 / 2)
    coins = 10
    coins += 5            // coins = coins + 5
    coins *= 2
    coins -= 1
    print(coins)
    print(2 + 3 * 4, (2 + 3) * 4)    // сначала * и /, потом + и -
```

ВЫВОД

```
3 1 3.5
29
14 20
```

Запомнить

- Имя переменной начинается с буквы; занятые слова (`if`, `int`, `bool` …) брать нельзя.
- Слишком большое целое (больше 9 223 372 036 854 775 807) — ошибка с номером строки, а не тихая порча числа.
- Имя внутри блока не может повторять имя снаружи: так нельзя случайно спрятать переменную.

Задание. Заведите переменные: цена товара, количество и скидка в процентах. Посчитайте сумму без скидки, размер скидки и итог. Итог посчитайте и в `int`, и в `dec` и сравните.

3 Сравнения и решения

Часть 1

Цель: Научить программу выбирать, что делать.

`if` `else if` `else` `and` `or` `not` `== != < > <= >=`

if, else if, else

`if` проверяет условие. Если оно верно, выполняется блок под ним. `else if` — следующая проверка, если первая не прошла. `else` — всё остальное. Сравнения: `==` равно, `≠` не равно, `< > ⩽ ⩾`.

```
func grade(score: int)
    if score ⩾ 90
        return "отлично"
    else if score ⩾ 70
        return "хорошо"
    else if score ⩾ 50
        return "сойдёт"
    else
        return "мало"

func main()
    print(grade(95), grade(72), grade(10))
```

ВЫВОД

```
отлично хорошо мало
```

and, or, not

Условия соединяются словами: `and` — оба верны, `or` — хотя бы одно, `not` — наоборот. Скобки помогают читать сложные условия.

```
func main()
    age = 20
    money = 300
    banned = false
    if age ⩾ 18 and money ⩾ 100 and not banned
        print("можно купить")
    if money < 50 or banned
        print("нельзя")
    else
        print("денег хватает и не в бане")
    can = age ⩾ 18 and not banned       // условие можно сохранить в bool
    print(can)
```

ВЫВОД

```
можно купить
денег хватает и не в бане
true
```

Запомнить

- После `if` не нужны ни скобки, ни двоеточие, ни `then`.
- `=` кладёт значение, `==` сравнивает.
- Текст тоже сравнивается: `"a" < "b"`, `name == "Алия"`.

Задание. Напишите функцию `ticket(age: int)`, которая возвращает цену билета: до 7 лет — 0, до 18 — 100, от 65 — 80, остальным — 200. Проверьте пять возрастов.

4 Повторение: циклы

Часть 1

Цель: Повторять действия, не копируя код.

`repeat … times` `repeat while` `count … from … to` `each … in` `skip` `stop`

Четыре вида циклов

`repeat N times` — ровно N раз. `repeat while условие` — пока условие верно. `count i from A to B` — счётчик от A до B, оба конца входят. `each x in список` — по каждому элементу.

```
func main()
    repeat 3 times
        show("ура ")
    print("")

    n = 1
    repeat while n < 100
        n *= 3
    print("первая степень 3 после 100:", n)

    count i from 1 to 5
        show(i * i, " ")
    print("")

    each fruit in ["яблоко", "груша", "слива"]
        print("-", fruit)
```

ВЫВОД

```
ура ура ура
первая степень 3 после 100: 243
1  4  9  16  25
- яблоко
- груша
- слива
```

skip и stop

`skip` — сразу к следующему шагу цикла. `stop` — выйти из цикла совсем.

```
func main()
    count i from 1 to 20
        if i % 2 == 0
            skip              // чётные пропускаем
        if i > 9
            stop              // дальше 9 не идём
        show(i, " ")
    print("")
```

ВЫВОД

```
1  3  5  7  9
```

Запомнить

- `count` умеет считать и вниз: `count i from 5 to 1` .
- `each` работает со списками, текстом (по буквам), `dict` (по ключам) и каналами.
- Бесконечный цикл: `repeat while true` , выход из него — `stop` или `return` .

Задание. Выведите таблицу умножения на 7 (от 7×1 до 7×10). Потом найдите первое число больше 1000, которое делится и на 7, и на 13.

5 Текст

Цель: Склеивать, резать, искать и менять текст.

`+` `len` `s[i]` `cut` `find` `replace` `split` `join` `upper` `lower` `trim` `starts_with` `ends_with` `to_int` `to_dec` `code` `char` `byte_len` `byte_at` `byte_char`

Склейка и буквы

Текст склеивается через `+`, числа при этом сами становятся текстом. `len` считает буквы (а не байты), `s[0]` — первая буква, `cut(s, от, до)` — кусок, буквы считаются с нуля.

```
func main()
    name = "Алия"
    hello = "Привет, " + name + "! Тебе " + 20 + " лет."
    print(hello)
    print(len(name), name[0], name[len(name) - 1])
    print(cut("Tachbit", 0, 3))          // буквы с 0-й по 3-ю
    each ch in "кот"
        show("[", ch, "]")
    print("")
```

ВЫВОД

```
Привет, Алия! Тебе 20 лет.
4 А я
Tach
[ к ][ о ][ т ]
```

Поиск и замена

Для работы с текстом есть готовые функции. Все они возвращают новый текст и не меняют старый.

```
func main()
    s = "  Мама мыла раму  "
    t = trim(s)                          // убрать пробелы по краям
    print("[" + t + "]")
    print(upper(t), lower(t))
    print(find(t, "мыла"), find(t, "кот"))    // номер буквы или -1
    print(replace(t, "раму", "окно"))
    print(starts_with(t, "Мама"), ends_with(t, "!"))
    parts = split("a,b,c", ",")
    print(parts, join(parts, " + "))
```

ВЫВОД

```
[Мама мыла раму]
МАМА МЫЛА РАМУ мама мыла раму
5 -1
Мама мыла окно
true false
[a, b, c] a + b + c
```

Числа и коды букв

`to_int` и `to_dec` превращают текст в число. Они могут не справиться (в тексте не число), поэтому их зовут с запасным значением через `or` (подробнее в уроке 12). `code` даёт номер буквы, `char` — букву по номеру.

```
func main()
    a = to_int("42") or 0
    b = to_int("сорок") or -1
    d = to_dec("2.5") or 0.0
    print(a + 1, b, d * 2)
    print(code("A"), char(66), code("п"), char(1087))
```

ВЫВОД

```
43 -1 5
65 B 1087 п
```

Запомнить

- `len`, `s[i]`, `cut`, `find` считают буквы: `len("привет")` = 6, смайлик — одна буква.
- Байты нужны редко (файлы, сеть): `byte_len`, `byte_at`, `byte_char`.
- `s += x` дописывает в конец быстро, без лишних копий.

Задание. Пользователь ввёл `" иванов иван "`. Сделайте из этого `"Иванов И."`: уберите пробелы, разрежьте по пробелу, сделайте первую букву заглавной, от имени оставьте одну букву.

6 Списки

Цель: Хранить много значений под одним именем.

`[ ]` `list T` `list list T` `.add` `.pop` `.clear` `len` `sort` `x[i]`

Создать, взять, изменить

Список пишется в квадратных скобках. Номера идут с нуля. Пустой список создаётся с типом: `list int`, `list text`. Номер за краем списка останавливает программу с понятной ошибкой.

```
func main()
    nums = [3, 8, 1]
    print(nums[0], len(nums))
    nums[0] = 5
    nums.add(7)                     // в конец
    last = nums.pop()               // взять и убрать последний
    print(nums, last)

    names = list text               // пустой список текста
    names.add("Вали")
    names.add("Алия")
    sort(names)
    print(names)

    grid = list list int            // список списков
    grid.add([1, 2])
    grid.add([3, 4])
    print(grid[1][0])
    nums.clear()
    print(len(nums))
```

ВЫВОД

```
3 3
[5, 8, 1] 7
[Алия, Вали]
3
0
```

Пройти и собрать итог

Чаще всего по списку идут через `each`, а нужен номер — через `count`.

```
func main()
    scores = [12, 40, 7, 33]
    total = 0
    best = scores[0]
    each s in scores
        total += s
        best = max(best, s)
    print("сумма", total, "лучший", best)
    count i from 0 to len(scores) - 1
        show(i, ":", scores[i], "  ")
    print("")
```

ВЫВОД

```
сумма 92 лучший 40
0 : 12   1 : 40   2 : 7   3 : 33
```

Запомнить

- Список передаётся в функцию как ярлык: функция меняет тот же список, а не копию.
- `sort` сортирует числа и текст на месте.
- `.clear()` делает список пустым и сразу отдаёт его память.

Задание. Есть список оценок. Посчитайте среднее (в `dec`), сколько оценок выше среднего, и создайте новый список только из оценок не ниже 50.

7 Словари: dict

Часть 1

Цель: Находить значение по ключу, как в телефонной книге.

`{ }` `dict K V` `d[k]` `.has` `.remove` `.keys` `.values` `.clear` `len` `each k in d`

Ключ и значение

`dict` хранит пары «ключ → значение». Ключ — `text` или `int`. Пустой словарь: `dict text int` (ключ текст, значение число).

```
func main()
    phones = {"Алия": "555-12", "Вали": "555-34"}
    print(phones["Алия"])
    phones["Тимур"] = "555-56"          // добавить или заменить
    print(phones.has("Вали"), phones.has("Олег"))
    phones.remove("Вали")
    print(len(phones), phones.keys())
    each name in phones                 // по ключам, в порядке добавления
        print(name, "→", phones[name])
```

ВЫВОД

```
555-12
true false
2 [Алия, Тимур]
Алия → 555-12
Тимур → 555-56
```

Счётчик слов

Самый частый приём: посчитать, сколько раз встретилось каждое слово.

```
func main()
    text_in = "кот пёс кот мышь кот пёс"
    counts = dict text int
    each w in split(text_in, " ")
        if not counts.has(w)
            counts[w] = 0
        counts[w] += 1
    print(counts)
    print(counts.values())
```

ВЫВОД

```
{кот: 3, пёс: 2, мышь: 1}
[3, 2, 1]
```

Запомнить

- Взять ключ, которого нет, — ошибка; сначала проверьте `.has(k)`.
- Убирать ключи во время `each` по этому же словарю нельзя: компилятор не даст.
- Порядок ключей при печати и `each` — как добавляли.

Задание. Сделайте словарь «товар → цена» на 5 товаров и список покупок. Посчитайте сумму покупок; если товара нет в словаре, напечатайте «нет такого: ...» и пропустите его.

8 Функции

Цель: Разбить программу на кусочки с именами.

`func` `return` `gives` `имя: тип` `вызов по имени`

Параметры и return

У каждого параметра пишется тип. `return` отдаёт ответ и сразу выходит из функции. Тип ответа Tachbit узнаёт сам. При вызове можно назвать параметры по имени: тогда порядок не важен и код читается лучше.

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

func greet(name: text, n: int)
    repeat n times
        print("привет,", name)

func main()
    print(area(3, 4))
    print(area(h: 10, w: 2))         // по именам — в любом порядке
    greet("Алия", 2)
```

ВЫВОД

```
12
20
привет, Алия
привет, Алия
```

Несколько ответов: gives

Если ответов несколько, их называют после `gives`. Внутри функции они — обычные переменные. Принять их можно сразу в несколько имён, в ячейки списка или в поля obj (`p.x, p.y = f()`).

```
func divide(a: int, b: int) gives whole: int, rest: int
    whole = a / b
    rest = a % b

func main()
    q, r = divide(17, 5)
    print("17 = 5 *", q, "+", r)
    res = [0, 0]
    res[0], res[1] = divide(9, 4)     // ответы сразу в ячейки списка
    print(res)
```

ВЫВОД

```
17 = 5 * 3 + 2
[2, 1]
```

Функция зовёт сама себя

Это называется рекурсия. Нужна остановка, иначе функция будет звать себя вечно; Tachbit тогда скажет «стек переполнен».

```
func fact(n: int)
    if n ⩽ 1
        return 1
    return n * fact(n - 1)

func main()
    print(fact(10))
```

ВЫВОД

```
3628800
```

Запомнить

- Один ответ — `return` ; несколько — `gives` ; не получилось — `fail` (урок 12).
- Списки и obj приходят в функцию как ярлыки: изменения видны снаружи. Числа и текст копируются.
- Порядок функций в файле не важен: `main` может стоять и первой.

Задание. Напишите `min_max(xs: list int) gives lo: int, hi: int` и `is_prime(n: int)` . Выведите все простые числа до 100 и самое маленькое и большое из них.

9 Свои типы: obj

Часть 1

Цель: Собрать несколько полей в одну вещь.

`obj` `Тип(поле: …)` `x.поле` `list Тип`

Описать и создать

`obj` описывает свой тип: какие у него качества и каких они типов. Создают его по имени типа, качества называют по имени. Пропущенные качества будут 0, пустым текстом или пустым списком.

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

func main()
    u = User(name: "Алия", age: 20)
    u.age += 1
    u.tags.add("админ")
    print(u.name, u.age, u.tags)
    print(u)                          // obj печатается целиком

    team = list User
    team.add(u)
    team.add(User(name: "Вали", age: 17))
    each p in team
        if p.age ⩾ 18
            print(p.name, "взрослый")
```

ВЫВОД

```
Алия 21 [админ]
User(name: Алия, age: 21, tags: [админ])
Алия взрослый
```

obj — это ярлык

Переменная с obj указывает на сам объект. Если отдать его в функцию или положить во второе имя, меняется тот же объект.

```
obj Hero
    hp: int

func hit(h: Hero, dmg: int)
    h.hp -= dmg

func main()
    a = Hero(hp: 100)
    b = a                 // b и a — один и тот же герой
    hit(b, 30)
    print(a.hp)
```

ВЫВОД

```
70
```

Запомнить

- Тип объекта пишется с большой буквы (так принято).
- obj можно класть в списки и dict, а качество может быть списком или другим obj.
- Пустой (не созданный) объект при обращении останавливает программу с понятной ошибкой.

Задание. Сделайте `obj Item` (название, цена, количество) и `obj Cart` со списком товаров. Напишите функции `add(cart, item)` и `total(cart)`.

10 Ярлыки: link

Цель: Менять чужую переменную через другое имя.

`link` `имя: link место` `func f(link x, …)`

link на качество или элемент

`link` — это ярлык: другое имя для того же места в памяти. Изменили ярлык — изменилось само место. Удобно, когда до значения длинный путь.

```
obj Hero
    hp: int
    gold: int

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

    xs = [1, 2, 3]
    first: link xs[0]
    first = 99
    print(xs)
```

ВЫВОД

```
75
[99, 2, 3]
```

link в параметре

Если функция должна поменять число вызывающего, параметр делают ярлыком.

```
func heal(link hp, amount: int)
    hp += amount
    if hp > 100
        hp = 100

func main()
    hp = 70
    heal(hp, 50)
    print(hp)
```

ВЫВОД

```
100
```

Запомнить

- Пока жив ярлык на элемент списка, сам список нельзя растить (`.add`): память списка могла бы переехать. Компилятор это проверяет.
- Ярлык на один элемент `list bool` взять нельзя: там биты, у бита нет адреса.

Задание. Напишите `swap(link a, link b)` , которая меняет два числа местами, и проверьте на двух переменных и на двух элементах списка.

11 Варианты: enum и match

Часть 1

Цель: Описать вещь, которая бывает в одном из нескольких состояний.

`enum` `Тип.вариант` `match` `else`

enum

`enum` перечисляет все варианты. Значение пишется как `Тип.вариант`. `print` печатает имя варианта.

```
enum Status
    new
    paid
    sent

func main()
    s = Status.new
    print(s)
    s = Status.paid
    if s == Status.paid
        print("оплачен")
```

ВЫВОД

```
new
оплачен
```

match

`match` разбирает значение по вариантам: у каждого свой блок. Несколько вариантов можно перечислить через запятую. Если вариант забыт, компилятор назовёт его по имени. `match` работает и с числами, и с текстом; там нужен `else`.

```
enum Status
    new
    paid
    sent

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

func day_name(n: int)
    match n
        6, 7
            return "выходной"
        else
            return "будний"

func main()
    print(label(Status.new), label(Status.sent))
    print(day_name(3), day_name(7))
```

ВЫВОД

```
новый в работе
будний выходной
```

Запомнить

- `match` по `enum` должен разобрать все варианты или иметь `else`.
- `enum` можно класть в obj, списки и посылать по каналам.

Задание. Сделайте `enum Light` (red, yellow, green) и функцию `next(l: Light)`, которая возвращает следующий цвет светофора. Прогоните светофор 6 шагов.

12 Ошибки: fail, try, or, else err

Цель: Сообщать о неудаче и решать, что делать с ней.

`fail` `try` `or` `else err` `err`

fail и три способа поймать

`fail "причина"` заканчивает функцию ошибкой. Такую функцию нельзя вызвать просто так: компилятор попросит решить, что делать при ошибке. `or значение` — подставить запасное. `else err` — свой блок, в `err` лежит текст причины. `try` — передать ошибку выше (в `main` программа остановится и напечатает причину).

```
func parse_age(s: text)
    n = to_int(s) or -1
    if n < 0
        fail "не число: " + s
    if n > 150
        fail "слишком много: " + s
    return n

func main()
    a = parse_age("20") or 0
    b = parse_age("сто") or 0            // запасное значение
    print(a, b)
    c = parse_age("200") else err
        print("ошибка:", err)
        c = 0
    print(c)
```

ВЫВОД

```
20 0
ошибка: слишком много: 200
0
```

try внутри функции

`try` полезен, когда функция сама не знает, что делать с ошибкой, и отдаёт её тому, кто её вызвал.

```
func parse_age(s: text)
    n = to_int(s) or -1
    if n < 0
        fail "не число: " + s
    return n

func sum_ages(a: text, b: text)
    x = try parse_age(a)       // сломалось — sum_ages тоже ломается
    y = try parse_age(b)
    return x + y

func main()
    print(sum_ages("20", "30") or -1)
    r = sum_ages("20", "abc") else err
        print("не вышло:", err)
        return
    print(r)
```

ВЫВОД

```
50
не вышло: не число: abc
```

Запомнить

- Сломаться могут: `read_file`, `write_file`, `input`, `to_int`, `to_dec`, `window_open`, `load_image`, сеть и ваши функции с `fail` или `try`.
- `try` можно и без присваивания: `try write_file("a.txt", s)`.
- Деление на ноль, номер за краем списка, переполнение — не `fail`, а авария: программа останавливается с номером строки.

Задание. Напишите `withdraw(balance: int, amount: int)`: ошибка, если сумма ≤ 0 или больше баланса, иначе новый баланс. Вызовите её тремя способами: с `or`, с `else err` и из другой функции через `try`.

13 Модули и библиотеки

Цель: Разложить программу по файлам и брать готовое.

`use` `modpub` `modpri` `модуль.функция` `lib/ui` `lib/physics` `lib/anim`

Свой модуль

`use money` подключает файл `money.tb` из той же папки. Функции модуля зовут через его имя. `modpub` делает функцию видимой другим файлам; без него (или с `modpri`) функция видна только своему файлу.

```
// файл money.tb
modpub func with_tax(price: int)
    return price + fee(price)

func fee(price: int)          // видна только внутри money.tb
    return price * 12 / 100

// файл main.tb
use money

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

Готовые библиотеки

С Tachbit идут библиотеки в папке `lib/`: `ui` (кнопки, поля, списки, окна внутри программы), `physics` (тела, столкновения, лучи), `anim` (плавность, ключи, кости). `use` ищет файл сначала рядом с программой, потом в папке `TB_LIB`, потом в `~/.tachbit/lib`.

```
use anim

func main()
    count i from 0 to 4
        t = i * 0.25
        print(t, anim.ease("inout", t))
```

ВЫВОД

```
0 0
0.25 0.0625
0.5 0.5
0.75 0.9375
1 1
```

Запомнить

- В модуле не нужна `main`; `tb check` модуля проверит его функции.
- Имя модуля = имя файла без `.tb`.
- Новые возможности после заморозки ядра 1.2 появляются именно как библиотеки.

Задание. Вынесите функции корзины из урока 9 в файл `cart.tb` и подключите его из `main.tb`. Сделайте одну функцию `modpri` и проверьте, что из `main` её не видно.

14 Числа, время и случайность

Часть 2

Цель: Готовые функции для чисел и времени.

`round` `abs` `min` `max` `sqrt` `sin` `cos` `tan` `asin` `acos` `atan` `atan2` `pow` `exp` `log` `floor` `ceil` `pi` `random` `time_ms` `wait_ms` `date_now` `unix_time` `bit_and` `bit_or` `bit_xor` `bit_not` `shift_left` `shift_right` `0x`

Числа

Округлить, модуль, меньшее и большее, корень. `min` и `max` работают с двумя числами. Для игр и графики есть углы и степени: `sin cos tan asin acos atan atan2` (углы в радианах, `pi()` = 3.14159...), `pow` (степень), `exp`, `log`, `floor` (вниз до целого), `ceil` (вверх).

```
func main()
    print(round(2.6), round(2.4), abs(-7), min(3, 9), max(3, 9), sqrt(16.0))
    print(floor(2.7), ceil(2.1), pow(2.0, 10.0), round(pi() * 100))
    print(round(sin(pi() / 2)), round(cos(0.0)), round(atan2(1.0, 1.0) * 180 / pi()))
    print(bit_and(12, 10), bit_or(12, 10), bit_xor(12, 10), shift_left(1, 4), 0xFF)
```

ВЫВОД

```
3 2 7 3 9 4
2 3 1024 314
1 1 45
8 14 6 16 255
```

Время и случайность

`time_ms()` — миллисекунды от старта (для замеров), `wait_ms` — подождать, `date_now()` — дата и время текстом, `unix_time()` — секунды с 1970 года. `random(a, b)` — случайное целое, оба конца входят.

```
func main()
    t0 = time_ms()
    wait_ms(50)
    spent = time_ms() - t0
    print(spent ⩾ 50)
    d = random(1, 6)
    print(d ⩾ 1 and d ⩽ 6)
    print(len(date_now()), unix_time() > 1700000000)
```

ВЫВОД

```
true
true
19 true
```

Запомнить

- `round` отдаёт `int`. Дробь из целого: умножьте на `1.0`.
- `0x`-числа и битовые функции работают везде, не только в `danger`.

Задание. Сделайте игру «угадай число»: программа загадывает `random(1, 100)`, вы вводите числа (`input()`, урок 15), она отвечает «больше» или «меньше» и в конце говорит число попыток и сколько секунд прошло.

15 Файлы, папки, ввод

Часть 2

Цель: Сохранять данные между запусками и спрашивать пользователя.

`read_file` `write_file` `file_exists` `file_size` `list_dir` `is_folder` `make_folder` `move` `remove` `input` `args` `env` `run` `exit`

Файлы

`write_file` записывает текст в файл целиком, `read_file` читает его. Обе могут сломаться (нет прав, нет файла), поэтому их зовут с `try`, `or` или `else err`.

```
func main()
    try write_file("notes.txt", "купить хлеб\nпозвонить маме\n")
    s = try read_file("notes.txt")
    each line in split(trim(s), "\n")
        print("•", line)
    print(file_exists("notes.txt"), file_size("notes.txt"))
    missing = read_file("нет-такого.txt") or "(пусто)"
    print(missing)
```

ВЫВОД

```
• купить хлеб
• позвонить маме
true 50
(пусто)
```

Папки

Для папок: `list_dir` (имена, у папок `/` в конце), `is_folder`, `make_folder`, `move` (переместить или переименовать), `remove` (файл или пустая папка).

```
func main()
    try make_folder("box")
    try write_file("box/a.txt", "A")
    try move("box/a.txt", "box/b.txt")
    names = try list_dir("box")
    print(names, is_folder("box"))
    try remove("box/b.txt")
    try remove("box")
    print(file_exists("box"))
```

ВЫВОД

```
[b.txt] true
false
```

Ввод, слова запуска, окружение

`input()` читает строку с клавиатуры (может сломаться, если ввод кончился). `args()` — список слов, написанных после имени файла: `tb run main.tb один два`. `env("HOME")` — переменная окружения. `run("команда")` запускает команду терминала и отдаёт код выхода. `exit(код)` завершает программу.

```
func main()
    show("Как тебя зовут? ")
    name = input() or "гость"
    print("Привет,", name)
    print("слов после файла:", len(args()))
    print(len(env("HOME")) > 0)
    code_out = run("true")
    print("код выхода:", code_out)
    exit(0)
```

ВЫВОД (ВВЕЛИ: АЛИЯ)

```
Как тебя зовут? Привет, Алия
слов после файла: 0
true
код выхода: 0
```

Запомнить

- Путь пишется от папки, где запустили программу, или полный: `/Users/имя/…` .
- `\n` в тексте — перевод строки, `\t` — табуляция, `\"` — кавычка.

Задание. Сделайте «дневник»: программа читает `diary.txt` (если его нет — пустой), печатает старые записи, спрашивает новую через `input()` и дописывает её с датой `date_now()` .

16 Интернет: HTTP, JSON и свой сервер

Часть 2

Цель: Получать данные из сети и связывать программы.

`http_get` `json_get` `json_text` `net_listen` `net_port` `net_accept` `net_connect` `net_send` `net_receive` `net_close`

Страница и JSON

`http_get(адрес)` отдаёт текст страницы. `json_get(текст, путь)` достаёт значение по пути: ключи и номера через точку. `json_text(s)` делает из текста строку JSON в кавычках, чтобы собрать свой JSON.

```
func main()
    j = "{\"users\": [{\"name\": \"Алия\", \"age\": 20}, {\"name\": \"Вали\"}]}"
    name = try json_get(j, "users.0.name")
    age = try json_get(j, "users.0.age")
    second = try json_get(j, "users.1")
    print(name, age)
    print(second)
    print("{\"msg\": " + json_text("он сказал \"да\"") + "}")
    // page = try http_get("https://example.com")   — нужен интернет
```

ВЫВОД

```
Алия 20
{"name": "Вали"}
{"msg": "он сказал \"да\""}
```

Свой сервер и клиент

`net_listen(порт)` открывает сервер (0 — любой свободный, узнать какой: `net_port`). `net_accept` ждёт клиента. `net_connect(адрес, порт)` подключается. `net_send` и `net_receive` передают текст, `net_close` закрывает. Пустой ответ `""` значит, что другая сторона закрыла связь.

```
func serve(srv: int)
    c = net_accept(srv) else err
        return -1
    msg = net_receive(c) else err
        return -1
    net_send(c, "эхо: " + msg) else err
        return -1
    net_close(c)
    return 0

func main()
    srv = try net_listen(0)
    port = net_port(srv)
    s = background serve(srv)
    c = try net_connect("127.0.0.1", port)
    try net_send(c, "привет")
    answer = try net_receive(c)
    print(answer)
    net_close(c)
    print(s.wait)
```

ВЫВОД

```
эхо: привет
0
```

Запомнить

- Все сетевые слова, кроме `net_port` и `net_close`, могут сломаться: нет сети, порт занят, сервер не ответил.
- Чтобы сервер обслуживал многих, принимайте клиентов в цикле и отдавайте каждого в `background` (урок 18).

Задание. Сделайте сервер-калькулятор: клиент шлёт `"3 + 4"`, сервер отвечает `"7"`. Проверьте его своим клиентом в той же программе.

17 Память

Цель: Понять, кто и когда отдаёт память, и где можно помочь.

`temp` `.clear()` `TB_GC_STATS` `TB_GC`

Всё само

В Tachbit не нужно писать `free`. Память шага цикла и функции компилятор отдаёт сам, сразу. То, что живёт дольше, убирает уборщик памяти: он сам находит объекты, на которые больше никто не смотрит. Программа, которая работает неделями, не растёт по памяти.

```
func main()
    keep = dict int text
    count i from 1 to 300000
        order = "заказ №" + i + " на сумму " + (i % 1000)
        keep[i % 100] = order          // хранится только последние 100
    print(len(keep), keep[7])
```

ВЫВОД

```
100 заказ №299907 на сумму 907
```

temp: временная память руками

Блок `temp` говорит: всё, что создано внутри, после блока не нужно. Память берётся большими кусками и отдаётся разом в конце — это очень быстро. Наружу можно вынести только числа.

```
func main()
    total = 0
    repeat 10 times
        temp
            lines = list text
            count i from 1 to 10000
                lines.add("строка " + i)
            each s in lines
                total += len(s)
    print(total)
```

ВЫВОД

```
1088940
```

Запомнить

- `.clear()` у списка и dict сразу отдаёт их память.
- `TB_GC_STATS=1 tb run файл.tb` печатает каждую уборку; `TB_GC=0` выключает уборщик.
- Внутри `temp` нельзя `return`, `fail`, `try`.

Задание. Возьмите пример с заказами и запустите его с `TB_GC_STATS=1`. Сколько было уборок? Потом поменяйте 100 на 100 000 и сравните.

18 Работа в фоне: background, shared, lock

Цель: Делать несколько дел одновременно.

`background` `.wait` `.crashed` `shared` `lock` `unlock`

background и .wait

`background f(...)` запускает функцию в отдельном потоке (на другом ядре процессора) и сразу отдаёт «номерок». `.wait` ждёт, пока функция закончит, и забирает ответ. Пока фон работает, `main` делает своё.

```
func sum_to(a: int, b: int)
    s = 0
    count i from a to b
        s += i
    return s

func main()
    x = background sum_to(1, 5000000)
    y = background sum_to(5000001, 10000000)
    z = sum_to(1, 10)                // main тем временем считает своё
    print(x.wait + y.wait, z)
```

ВЫВОД

```
50000005000000 55
```

shared и lock

`shared` — переменная, общая для всей программы. Если её меняют из фона, это делают внутри `lock имя` : тогда в один момент блок выполняет только кто-то один, и сложения не теряются. `unlock имя` отпускает замок раньше конца блока.

```
shared hits = int 0

func work(n: int)
    repeat n times
        lock counter
            hits += 1
    return 0

func main()
    a = background work(100000)
    b = background work(100000)
    print(a.wait + b.wait, hits)
```

ВЫВОД

```
0 200000
```

> **Запомнить**
>
> - В `background` можно передать только `int` , `dec` , `bool` , `text` , каналы и списки через `move` .
> - Компилятор сам ловит: `shared` без замка, замок внутри того же замка, два замка в разном порядке.
> - `a.crashed` — упала ли задача (в своей ОС Omnium падение задачи не роняет систему).

Задание. Посчитайте простые числа до 2 000 000 в четырёх `background` по кускам и сравните время (`time_ms`) с подсчётом в одном потоке.

19 Каналы, move и все ядра

Цель: Передавать данные между потоками и раздавать цикл всем ядрам.

`channel T` `.send` `.receive` `.close` `.ready` `each x in ch` `move` `on all cores`

channel

Канал — очередь между потоками. `.send(x)` кладёт в конец, `.receive()` берёт из начала и ждёт, если пусто. `.close()` — больше писать не будут. `each x in ch` берёт, пока канал не закрыт и не пуст. `.ready()` — есть ли что взять без ожидания.

```
func worker(jobs: channel int, results: channel text)
    each n in jobs
        results.send("готово " + n * n)
    return 0

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())
    print(w.wait)
```

ВЫВОД

```
готово 1
готово 4
готово 9
0
```

move и on all cores

`move` отдаёт список в фон целиком, без копии; после этого старое имя пустое. `on all cores` раздаёт цикл `count` или `each` всем ядрам процессора. Общий итог копят в `shared` через `+=`.

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

shared found = int 0

func is_prime(n: int)
    if n < 2
        return false
    d = 2
    repeat while d * d ⩽ n
        if n % d == 0
            return false
        d += 1
    return true

func main()
    data = list int
    count i from 1 to 1000
        data.add(i)
    a = background total(move data)
    print(a.wait)
    count n from 1 to 100000 on all cores
        if is_prime(n)
            found += 1
    print(found)
```

ВЫВОД

```
500500
9592
```

Запомнить

- Внутри `on all cores` данные снаружи только читаются; компилятор следит даже через другие имена и функции.
- `TB_CORES=1 ./программа` — запустить на одном ядре для сравнения.
- По каналу идут `int`, `dec`, `bool`, `text` и `enum`.

Задание. Сделайте конвейер из трёх потоков: первый шлёт числа 1..20, второй возводит их в квадрат, третий складывает. Связь — двумя каналами.

20 Скорость одним словом

Цель: Слова, которые ускоряют горячие циклы.

`by vec` `by columns` `each piece in … by N` `together`

by vec и by columns

`each x in nums by vec` складывает несколько чисел одной командой процессора (SIMD). `obj … by columns` хранит список объектов колонками: все `x` подряд, все `mass` подряд. Это быстрее, когда цикл читает одно-два качества из многих.

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

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)

    parts = list Particle
    count i from 1 to 100
        parts.add(Particle(x: i * 1.0, mass: 2.0))
    m = 0.0
    each p in parts
        m += p.mass
    print(m)
```

ВЫВОД

```
333833500
200
```

Куски и together

`each piece in data by N` идёт кусками по N элементов, чтобы кусок помещался в быстрый кэш процессора; `piece` — окно в сам список. `together` просит память принести несколько значений одновременно, а не по очереди.

```
func main()
    data = list int
    count i from 1 to 10000
        data.add(i)
    sum = 0
    each piece in data by 4096
        each v in piece
            sum += v
    print(sum)

    ages = {"Алия": 20, "Вали": 17}
    together
        a = ages["Алия"]
        b = ages["Вали"]
        x = data[9999]
    print(a, b, x)
```

ВЫВОД

```
50005000
20 17 10000
```

Запомнить

- Эти слова не меняют смысл программы, только скорость. Сначала пишите просто, потом ускоряйте то, что медленно.
- Многое компилятор делает сам: «заказ наперёд», горячие и холодные качества obj, склейка текста за один раз, `list bool` битами.

Задание. Замерьте `time_ms` сумму квадратов 10 000 000 чисел обычным `each` и с `by vec`.

21 Окно и 2D-рисование

Цель: Открыть окно, рисовать и слушать клавиши и мышь.

`window_open` `window_alive` `clear_window` `rgb` `rgba` `draw_pixel` `draw_rect` `draw_round` `draw_circle` `draw_line` `draw_triangle` `draw_text` `key_down` `mouse_x` `mouse_y` `mouse_down` `mouse_wheel` `mouse_back` `mouse_forward` `typed_text` `pad_count` `pad_down` `pad_axis` `load_image` `draw_image` `draw_image_alpha` `image_w` `image_h` `image_scaled` `screen_image` `load_font` `draw_font` `text_width` `frame_ms` `fps` `set_fps` `window_w` `window_h` `window_sharp` `window_scale` `window_new` `window_use` `window_shown` `window_close` `window_fullscreen` `mouse_lock` `mouse_dx` `mouse_dy` `set_cursor` `clipboard_get` `clipboard_set` `blur_rect` `draw_shadow`

Самая маленькая игра

`window_open` открывает окно. `repeat while window_alive()` — кадр за кадром: показывает нарисованное, принимает клавиши и держит частоту экрана. Точка (0, 0) — левый верхний угол, `y` растёт вниз. Цвет — `rgb(красный, зелёный, синий)` от 0 до 255.

```
func main()
    try window_open(400, 300, "Привет")
    x = 200.0
    repeat while window_alive()
        dt = frame_ms() / 1000            // секунды с прошлого кадра
        if key_down("left")
            x -= 200 * dt
        if key_down("right")
            x += 200 * dt
        clear_window(rgb(20, 20, 30))
        draw_rect(0, 250, 400, 50, rgb(60, 140, 80))
        draw_circle(round(x), 230, 20, rgb(47, 209, 191))
        draw_line(0, 0, mouse_x(), mouse_y(), rgba(255, 255, 255, 100))
        draw_text(10, 10, "Стрелки двигают круг", 1, rgb(255, 255, 255))
        if mouse_down()
            draw_pixel(mouse_x(), mouse_y(), rgb(255, 0, 0))
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Картинки, шрифты, ввод

Картинки PNG, JPEG, GIF и BMP: `load_image` → номер, `draw_image`. Шрифты TTF: `load_font`, `draw_font`, `text_width`. Набранные буквы — `typed_text()`, колёсико — `mouse_wheel()`, геймпад — `pad_down` и `pad_axis`.

```
func main()
    try window_open(640, 360, "Картинки")
    hero = load_image("hero.png") or -1
    font = load_font("/System/Library/Fonts/Helvetica.ttc") or -1
    name = ""
    repeat while window_alive()
        name += typed_text()
        clear_window(rgb(240, 240, 245))
        if hero ⩾ 0
            draw_image(hero, 20, 20)
        if font ⩾ 0
            draw_font(font, 20, 300, "Имя: " + name, 24, rgb(30, 30, 40))
        if pad_count() > 0 and pad_down(0, "a")
            draw_round(400, 40, 200, 80, 16, "all", rgb(255, 95, 162))
        set_cursor("hand")
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Запомнить

- Клавиши: `left right up down space enter escape backspace tab shift cmd ctrl alt`, `f1`…`f12`, буквы и цифры. `cmd` = Command на Mac и Ctrl на Linux/Windows.
- `frame_ms()` — длина прошлого кадра: умножайте скорость на него, и игра идёт одинаково при любой частоте.
- Ещё окна: `chat = try window_new(300, 200, "Чат")`, `window_use(chat)` — рисовать в нём, `window_use(0)` — снова в главном, `window_shown`, `window_close`.
- Полный экран — `window_fullscreen(true)`. Мышь для шутера: `mouse_lock(true)`, сдвиг мыши — `mouse_dx()` и `mouse_dy()`.
- Ещё: `window_w/h`, `window_sharp` (Retina), `set_fps`, `fps`, `clipboard_get/set`, `blur_rect`, `draw_shadow`, `draw_image_alpha`, `image_scaled`, `screen_image`.

Задание. Сделайте игру «лови звёзды»: звезда падает сверху в случайном месте, корзина внизу двигается стрелками, считайте очки и выводите их текстом.

22 3D

Цель: Сетки из треугольников, камера, свет и картинки на гранях.

`mesh_new` `mesh_add` `mesh_add_uv` `mesh_image` `draw_mesh` `draw_mesh_rot` `camera` `sunlight` `point_light` `spot_light` `light_off` `fog` `shadows` `mesh_smooth` `mesh_shine` `mesh_glow` `mesh_alpha` `mesh_clear` `mesh_count` `mesh_lod` `mesh_far` `mesh_free` `bone_add` `bone_box` `bone_turn` `mesh_shader` `shader_set` `gpu func` `draw_triangle`

Куб, камера и свет

`mesh_new()` — пустая сетка, `mesh_add` добавляет в неё треугольник (x вправо, y вверх, z вперёд). `draw_mesh(m, x, y, z, поворот)` рисует сетку в мире, `draw_mesh_rot` — с наклоном и креном. `camera` — где стоит глаз и куда смотрит. `sunlight` включает солнце, `point_light` — лампу (до 8 ламп), `fog` — туман, `shadows` — тени. У сетки есть свои настройки: `mesh_smooth` (плавный свет), `mesh_shine` (блеск), `mesh_glow` (светится сама), `mesh_alpha` (стекло, вода). Считает видеокарта.

```
func quad(m: int, x1: dec, y1: dec, z1: dec, x2: dec, y2: dec, z2: dec, x3: dec, y3: dec, z3: dec, x4:
dec, y4: dec, z4: dec, c: int)
    mesh_add(m, x1, y1, z1, x2, y2, z2, x3, y3, z3, c)
    mesh_add(m, x1, y1, z1, x3, y3, z3, x4, y4, z4, c)

func main()
    try window_open(640, 480, "Куб")
    m = mesh_new()
    quad(m, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, rgb(230, 80, 80))
    quad(m, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, rgb(80, 200, 120))
    quad(m, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0, rgb(80, 140, 240))
    glass = mesh_new()
    quad(glass, -1.0, -1.0, 0.0, -1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, -1.0, 0.0, rgb(120, 200, 255))
    mesh_alpha(glass, 0.4)                // стекло: видно, что за ним
    mesh_shine(m, 0.6)                    // блеск, как у лака
    camera(0, 2, -6, 0, 0, 0)
    sunlight(0.5, 1, -0.3, 0.25)          // солнце сверху сбоку
    point_light(0, -3, 2, -2, rgb(255, 180, 90), 8)   // тёплая лампа номер 0
    fog(rgb(15, 18, 28), 8, 20)           // туман от 8 до 20
    shadows(20)                           // тени в квадрате 20 × 20
    turn = 0.0
    repeat while window_alive()
        turn += 60 * frame_ms() / 1000
        clear_window(rgb(15, 18, 28))
        draw_mesh(m, 0, 0, 0, turn)
        draw_mesh_rot(m, 3, 0, 0, turn, 30, 0)
        draw_mesh(glass, -3, 0, -1, 0)
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Картинка на сетке

`mesh_image(m, картинка)` рисует сетку картинкой. `mesh_add_uv` — треугольник, у каждого угла своё место на картинке (u, v от 0 до 1).

```
func main()
    try window_open(640, 480, "Пол")
    tex = try load_image("grass.png")
    floor = mesh_new()
    mesh_image(floor, tex)
    mesh_add_uv(floor, -5, 0, -5, 0, 0, -5, 0, 5, 0, 4, 5, 0, 5, 4, 4)
    mesh_add_uv(floor, -5, 0, -5, 0, 0, 5, 0, 5, 4, 4, 5, 0, -5, 4, 0)
    camera(0, 3, -8, 0, 0, 0)
    repeat while window_alive()
        clear_window(rgb(150, 200, 255))
        draw_mesh(floor, 0, 0, 0, 0)
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Запомнить

- Ближнее закрывает дальнее само (глубина). Задние грани закрытых сеток не рисуются.
- Готовые модели из Blender и других программ: `use model`, потом `md = try model.load("hero.glb")`, `model.build(md)`, `model.draw(md, x, y, z, 0, 0, 0)`. Анимации из файла: `model.play(md, "walk", секунды)`.
- Кости: `bone_add`, `bone_box`, `bone_turn` — одна сетка становится персонажем, который сгибается.
- Свои шейдеры пишутся на самом Tachbit: `gpu func`, потом `mesh_shader(m, "имя")`. Готовые (вода, огонь, небо, мультяшный, pbr) — в `use shaders`.
- Большой мир: `mesh_lod` (вдали проще), `mesh_far` (дальше не рисовать), `mesh_free` (отдать память), подгрузка кусков — `use stream`.
- `TB_GPU_INFO=1` — сказать при запуске, чем рисуем; `TB_GPU=0` — рисовать процессором.
- Примеры: `examples/basics/cube.tb`, `shader_scene.tb`, `view_model.tb`, игры `examples/apps/corridor`, `football`, `pbr_room`, `big_world`.

Задание. Постройте из кубов лестницу из 5 ступенек и облетите её камерой по кругу: камера в точке `(8 * cos(a), 4, 8 * sin(a))`, смотрит в центр, угол `a` растёт каждый кадр. Повесьте над лестницей лампу.

23 Звук

Цель: Звуки, музыка и звук в 3D.

`load_sound` `play_sound` `loop_sound` `play_sound_at` `loop_sound_at` `play_tone` `stop_sound` `sound_playing` `sound_volume` `sound_pitch` `move_sound` `master_volume` `listener`

Играть звуки

`load_sound` загружает WAV и отдаёт номер звука. `play_sound` играет один раз, `loop_sound` — по кругу; оба отдают номер голоса, которым потом управляют. `play_tone(частота, мс)` — простой писк.

```
func main()
    try window_open(300, 200, "Звук")
    step = try load_sound("step.wav")
    music = try load_sound("music.wav")
    m = loop_sound(music)
    sound_volume(m, 0.4)
    master_volume(0.8)
    repeat while window_alive()
        if key_down("space")
            v = play_sound(step)          // номер голоса
        if key_down("t")
            beep = play_tone(440, 150)
        if key_down("s")
            stop_sound(m)
        clear_window(rgb(30, 30, 40))
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Звук в мире

`play_sound_at(звук, x, y, z)` ставит звук в 3D: громкость по расстоянию, сторона, эффект Доплера. `listener` — где уши (обычно те же числа, что у `camera`). `move_sound` двигает звучащий голос.

```
func main()
    try window_open(300, 200, "3D-звук")
    car = try load_sound("engine.wav")
    v = loop_sound_at(car, -20, 0, 5)
    x = -20.0
    listener(0, 0, 0, 0, 0, 1)
    repeat while window_alive()
        x += 10 * frame_ms() / 1000
        move_sound(v, x, 0, 5)
        sound_pitch(v, 1.0)
        if not sound_playing(v)
            stop
        clear_window(rgb(30, 30, 40))
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Запомнить

- Свой микшер: 64 голоса, без щелчков, ограничитель на выходе.
- `TB_SOUND_FILE=out.wav` — писать звук в файл, `TB_SOUND=0` — без звука.

Задание. Добавьте звук в игру «лови звёзды»: писк при поимке, тон пониже при промахе и тихую музыку по кругу.

24 Приложения с кнопками: lib/ui

Часть 4

Цель: Собрать окно с кнопками, полями и списками.

`use ui` `ui.start` `ui.frame` `ui.end_frame` `ui.button` `ui.btn` `ui.edit` `ui.lbl` `ui.hbox` `ui.vbox` `ui.done` `ui.check` `ui.window`

Немедленное рисование

Библиотека `ui` написана на самом Tachbit. Каждый кадр программа заново зовёт `ui.button`, и функция сразу рисует кнопку и отвечает, нажали ли её. Хранить кнопки нигде не нужно. `hbox` / `vbox` раскладывают элементы без координат.

```
use ui

func main()
    try window_open(520, 300, "Заметки")
    u = ui.start()
    notes = list text
    draft = ""
    repeat while window_alive()
        ui.frame(u)
        clear_window(u.bg)
        ui.pad(u, 16)
        ui.heading(u, "Мои заметки")
        ui.hbox(u, 32, 8)
        draft = ui.edit(u, "draft", draft, "новая заметка")
        if ui.btn_main(u, "Добавить") and draft =/= ""
            notes.add(draft)
            draft = ""
        ui.done(u)
        each n in notes
            ui.lbl(u, "• " + n)
        ui.end_frame(u)
```

ПРОВЕРКА

```
tb check: ошибок нет. Откроется окно — запускайте на своём компьютере.
```

Запомнить

- Что есть: кнопки, галочки, поля ввода с выделением и буфером обмена, списки, прокрутка, меню, вкладки, ползунки, подсказки, перетаскивание, окна внутри программы, вопрос «да/нет», тёмная и светлая тема.
- Пример большого приложения — файловый менеджер `examples/apps/files`.

Задание. Сделайте список дел: поле ввода, кнопка «Добавить», у каждой задачи галочка «сделано» и кнопка «удалить». Сохраняйте список в файл при выходе.

25 Железо: danger, память руками, shape, zone

Цель: Работать с адресами, байтами и регистрами, как в C.

`danger` `new_mem` `free_mem` `grow_mem` `copy_mem` `fill_mem` `read8…64` `write8…64` `address_of` `asm` `shape` `at` `u8…u64` `i8…i64` `bits` `bytes` `big` `zone_new` `zone_take` `zone_used` `zone_clear` `zone_free`

danger и память руками

Слова, которыми можно испортить память, работают только внутри блока `danger`. Так сразу видно опасные места. `new_mem` берёт блок байт (заполнен нулями), `read/write 8/16/32/64` читают и пишут числа нужного размера, `free_mem` отдаёт блок.

```
func main()
    danger
        buf = try new_mem(64)
        write32(buf, 7)
        write32(buf + 4, 1000)
        fill_mem(buf + 8, 0xFF, 4)
        copy_mem(buf + 16, buf, 8)
        print(read32(buf + 16), read32(buf + 20), read8(buf + 9))
        buf = try grow_mem(buf, 128)
        free_mem(buf)
```

ВЫВОД

```
7 1000 255
```

shape: раскладка байтов

`shape` описывает, какие поля лежат в памяти по порядку и какого они размера. `Тип at адрес` накладывает раскладку на память, и поля читаются как обычные `int`. Так описывают регистры устройств, заголовки пакетов и записи на диске.

```
shape Header
    kind: u8
    flags: bits 4
    ver: bits 4
    length: u16 big
    id: u32

func main()
    print(Header.size)
    danger
        a = try new_mem(Header.size)
        h = Header at a
        h.kind = 2
        h.ver = 3
        h.length = 513
        h.id = 70000
        print(h.kind, h.ver, h.length, h.id, read8(a + 2), read8(a + 3))
        free_mem(a)
```

ВЫВОД

```
8
2 3 513 70000 2 1
```

zone: много кусков, одна уборка

Зона выдаёт куски памяти очень быстро и отдаёт их все сразу: удобно для кадра игры или одного запроса сервера.

```
func main()
    danger
        z = try zone_new(4096)
        count i from 1 to 100
            p = zone_take(z, 24)
            write64(p, i)
        print(zone_used(z))
        zone_clear(z)
        print(zone_used(z))
        zone_free(z)
```

ВЫВОД

```
3200
0
```

Запомнить

- Размеры полей: `u8 u16 u32 u64`, `i8 … i64`, `bits N`, `bytes N`, `big` — старший байт первым.
- `TB_MEM_CHECK=1` ловит чтение после `free_mem`, выход за край и забытые блоки — со строками.
- Внутри `danger` числа при переполнении заворачиваются, как в железе.

Задание. Опишите `shape` для записи «товар»: код `u32`, цена `u32`, количество `u16`, флаг «есть на складе» `bits 1`. Положите 10 записей подряд в одну память и посчитайте сумму.

26 Связь с C и системой

Цель: Звать библиотеки C, быть библиотекой для C, звать ядро ОС.

`use c` `c.функция` `for c` `tb lib` `syscall` `i32 u64 f32 …`

use c

Блок `use c` описывает функции C, которые хотите звать. Библиотеку называют в кавычках (`use c "sqlite3"`) или берут свой файл `.c` . Без имени — стандартная библиотека C, вызов через `c.` . Вызов C — только в `danger` .

```
use c
    func abs(x: i32) gives i32
    func strlen(s: text) gives u64

func main()
    danger
        print(c.abs(-42), c.strlen("Tachbit"))
```

ВЫВОД

```
42 7
```

for c, tb lib и syscall

Функция с `for c` видна из C под своим именем; `tb lib файл.tb` собирает `libфайл.a` и заголовок `файл.h` . `syscall(номер, …)` зовёт ядро Linux или Mac напрямую.

```
func add(a: int, b: int) for c
    return a + b

func main()
    danger
        pid = syscall(39)          // getpid на Linux x86-64 (на Mac номер другой)
        print(pid > 0)
    print(add(2, 3))
```

ВЫВОД

```
true
5
```

Запомнить

- Типы для C: `int i8 i16 i32 i64 u8 u16 u32 u64 dec f32 bool text` .
- Текст из C сразу копируется в память Tachbit.
- В своей ОС Omnium вместо syscall — каналы к службам.

Задание. Подключите `use c` функцию `getenv(name: text) gives text` и сравните её ответ с `env("HOME")` .

27 Своё ядро: tb bare

Цель: Программа, которая загружается без операционной системы.

`tb bare` `on timer` `on keyboard` `on interrupt N` `read_key` `uptime_ms` `yield` `port_in8…32` `port_out8…32` `page_new` `page_free` `interrupts_on` `interrupts_off` `tb app`

Ядро на Tachbit

`tb bare файл.tb` делает 64-битное ядро для x86-64: оно загружается само (QEMU или GRUB), без macOS и Linux. Внутри работают `print`, списки, obj, `background` (это настоящие задачи с переключением по таймеру). Функции с `on timer`, `on keyboard`, `on interrupt N` зовёт сам процессор.

```
shared ticks = int 0

func on_tick() on timer          // 1000 раз в секунду
    ticks += 1

func on_key() on keyboard        // при каждом нажатии
    k = read_key()

func main()
    print("ядро запущено")
    wait_ms(100)
    print("тиков:", ticks, "время:", uptime_ms())
    danger
        write16(0xB8000, 0x0E41)       // жёлтая A в угол экрана
        status = port_in8(0x64)        // порт клавиатуры
        interrupts_on()
```

Запуск

Нужен QEMU. Пример ядра — `examples/kernel/main.tb`, своя ОС Omnium — папка `omnium/`. `tb app файл.tb` собирает программу для Omnium (`--own` — своим машинным кодом, без LLVM).

```
brew install qemu                    # Mac, один раз
tb bare kernel.tb                    # → kernel.elf
qemu-system-x86_64 -kernel kernel.elf -serial stdio
./tools/kernel.sh examples/kernel/main.tb --text
./omnium/run.sh                      # своя ОС Omnium
```

Запомнить

- Только в ядре: `port_in/out`, `page_new/free`, `interrupts_on/off`, `uptime_ms`, `read_key`, `yield`, `module_*`, `kernel_symbol`, `call_address`.
- Падение задачи `background` в ядре роняет только её; `a.crashed` говорит, упала ли.
- Проверено в QEMU; на настоящем компьютере пока не запускалось.

Задание. Запустите `./tools/kernel.sh examples/kernel/main.tb` и найдите в коде, где ядро обрабатывает клавиши. Добавьте свою команду, которая печатает время работы ядра.

28 Команды tb и настройки

Цель: Всё, что можно сделать из терминала.

`tb new` `tb start` `tb run` `tb build` `tb check` `tb c` `tb ll` `tb bare` `tb app` `tb lib` `tb version` `tb lang`

Команды

Короткие команды для проектов и полные для отдельных файлов.

```
tb new game                  # новый проект: папка game/ с main.tb
tb start game [слова]        # собрать проект и запустить
tb run файл.tb [слова]       # собрать и запустить файл
tb build файл.tb             # программа рядом с файлом
tb check файл.tb             # только найти ошибки
tb c файл.tb                 # показать получившийся C
tb ll файл.tb                # показать LLVM IR
tb bare файл.tb              # ядро без ОС
tb app файл.tb [--own]       # программа для Omnium
tb lib файл.tb               # библиотека для C
tb version                   # какая версия
tb lang en / tb lang ru      # сообщения компилятора по-английски или по-русски
--llvm / --c                 # к build и run: через LLVM или через C
```

Переменные окружения

Пишутся перед командой: `TB_GPU_INFO=1 tb run game.tb`.

```
TB_CORES=1        # все ядра → одно (сравнить скорость)
TB_GC_STATS=1     # печатать уборки памяти;  TB_GC=0 — без уборки
TB_MEM_CHECK=1    # строгая проверка памяти в danger
TB_GPU=0          # рисовать процессором;  TB_GPU_INFO=1 — чем рисуем
TB_FPS_FREE=1     # не ждать экран (замеры)
TB_SOUND=0        # без звука;  TB_SOUND_FILE=out.wav — звук в файл
TB_LIB=папка      # где искать библиотеки для use
TB_UI_FONT=шрифт.ttf   # шрифт для lib/ui
```

Запомнить

- VS Code: расширение Tachbit подсвечивает код и подчёркивает ошибки при сохранении.
- Полный справочник — `docs/guide.md` в репозитории.

Задание. Создайте проект `tb new hello`, запустите его через `tb start hello`, потом посмотрите `tb c hello/main.tb`: сравните, во что превратились ваши 5 строк.

Указатель 1. Слова языка

Слово	Что делает	Где
`func`	объявить функцию	урок 8
`return`	отдать ответ и выйти	урок 8
`gives`	несколько ответов функции	урок 8
`if / else if / else`	решение	урок 3
`and / or / not`	соединить условия	урок 3
`true / false`	да / нет	урок 2
`const`	значение, которое нельзя менять	урок 2
`int dec text bool`	типы	урок 2
`repeat N times`	N раз	урок 4
`repeat while`	пока верно	урок 4
`count i from A to B`	счётчик	урок 4
`each x in …`	по элементам	урок 4
`skip / stop`	следующий шаг / выйти из цикла	урок 4
`list T / [ ]`	список	урок 6
`dict K V / { }`	словарь	урок 7
`obj`	свой тип	урок 9
`link`	ярлык на место	урок 10
`enum`	варианты	урок 11
`match`	разбор по вариантам	урок 11
`fail`	закончить функцию ошибкой	урок 12
`try`	передать ошибку выше	урок 12
`or`	запасное значение при ошибке	урок 12
`else err`	свой блок на ошибку	урок 12
`use`	подключить модуль	урок 13
`modpub / modpri`	видно другим файлам / только своему	урок 13
`temp`	временная память блока	урок 17
`background / .wait / .crashed`	работа в фоне	урок 18
`shared`	общая переменная	урок 18
`lock / unlock`	замок	урок 18
`channel T`	очередь между потоками	урок 19
`move`	отдать список целиком	урок 19
`on all cores`	цикл на всех ядрах	урок 19
`by vec`	SIMD-сумма	урок 20
`obj … by columns`	хранить колонками	урок 20
`each piece in … by N`	кусками	урок 20
`together`	несколько чтений сразу	урок 20
`danger`	блок опасных слов	урок 25
`shape / at`	раскладка байтов	урок 25
`u8…u64 i8…i64 bits bytes big`	поля shape и типы C	урок 25

`use c`	звать C	урок 26
`for c`	видно из C	урок 26
`on timer / on keyboard / on interrupt N`	функция прерывания (ядро)	урок 27
`//`	комментарий	урок 1

Указатель 2. Встроенные функции

✱ — может сломаться: зовите с `try`, `or` или `else err`.

Функция	Что делает	Где
Вывод и ввод		
`print(a, b, …)`	вывести через пробел и перевести строку	урок 1
`show(…)`	вывести без перевода строки	урок 1
`input()`	строка с клавиатуры ✱	урок 15
`args()`	слова после имени файла	урок 15
Текст		
`len(s)`	сколько букв (у списка и dict — элементов)	урок 5
`cut(s, a, b)`	кусок с буквы a по b	урок 5
`find(s, x)`	номер буквы или -1	урок 5
`replace(s, a, b)`	заменить	урок 5
`split(s, sep) / join(xs, sep)`	разрезать / склеить	урок 5
`upper / lower / trim`	регистр, пробелы по краям	урок 5
`starts_with / ends_with`	начинается / кончается	урок 5
`code(s) / char(n)`	номер буквы / буква по номеру	урок 5
`to_int(s) / to_dec(s)`	текст в число ✱	урок 5
`byte_len / byte_at / byte_char`	работа с байтами	урок 5
Числа		
`round / abs / min / max / sqrt`	округлить, модуль, меньшее, большее, корень	урок 14
`sin / cos / tan / asin / acos / atan / atan2 / pi()`	углы (радианы)	урок 14
`pow / exp / log / floor / ceil`	степень, логарифм, вниз и вверх до целого	урок 14
`random(a, b)`	случайное целое	урок 14
`bit_and / bit_or / bit_xor / bit_not`	биты	урок 14
`shift_left / shift_right`	сдвиг битов	урок 14
`sort(xs)`	отсортировать список	урок 6
Время		
`time_ms() / wait_ms(n)`	миллисекунды / подождать	урок 14
`date_now() / unix_time()`	дата текстом / секунды с 1970	урок 14
Файлы и система		
`read_file / write_file`	прочитать / записать файл ✱	урок 15
`file_exists / file_size`	есть ли файл / размер	урок 15
`list_dir / is_folder / make_folder`	папки ✱	урок 15
`move(a, b) / remove(p)`	переместить / удалить ✱	урок 15
`env(name)`	переменная окружения	урок 15
`run(cmd) / exit(code)`	команда терминала / выйти	урок 15
Сеть		
`http_get(url)`	текст страницы ✱	урок 16
`json_get(j, path) / json_text(s)`	JSON ✱	урок 16
`net_listen / net_port / net_accept`	сервер ✱	урок 16

`net_connect / net_send / net_receive / net_close`	связь ✱	урок 16
Окно и ввод		
`window_open(w, h, title)`	открыть окно ✱	урок 21
`window_alive()`	следующий кадр	урок 21
`window_w / window_h / window_sharp / window_scale`	размер, Retina	урок 21
`window_new / window_use / window_shown / window_close`	ещё окна ✱	урок 21
`window_fullscreen`	полный экран	урок 21
`frame_ms / fps / set_fps`	время кадра и частота	урок 21
`key_down(name) / typed_text()`	клавиши / набранный текст	урок 21
`mouse_x / mouse_y / mouse_down / mouse_wheel / mouse_back / mouse_forward`	мышь	урок 21
`mouse_lock / mouse_dx / mouse_dy`	мышь для шутера	урок 21
`pad_count / pad_down / pad_axis`	геймпад	урок 21
`clipboard_get / clipboard_set / set_cursor`	буфер обмена, курсор	урок 21
2D		
`rgb / rgba`	цвет	урок 21
`clear_window`	залить окно	урок 21
`draw_pixel / draw_rect / draw_circle / draw_line / draw_triangle`	фигуры	урок 21
`draw_round / blur_rect / draw_shadow`	скругление, матовое стекло, тень	урок 21
`draw_text`	текст встроенным шрифтом	урок 21
`load_font / draw_font / text_width`	шрифты TTF ✱	урок 21
`load_image / draw_image / draw_image_alpha`	картинки ✱	урок 21
`image_w / image_h / image_scaled / screen_image / image_from_mem`	размер, масштаб, снимок	урок 21
3D		
`mesh_new / mesh_add / mesh_add_uv / mesh_image`	сетка	урок 22
`draw_mesh / draw_mesh_rot`	нарисовать сетку	урок 22
`camera / sunlight`	камера и солнце	урок 22
`point_light / spot_light / light_off / light_shadow`	лампы и фонари	урок 22
`fog / shadows`	туман и тени	урок 22
`mesh_smooth / mesh_shine / mesh_glow / mesh_alpha`	плавный свет, блеск, свечение, стекло	урок 22
`bone_add / bone_box / bone_weight / bone_turn / bone_move`	кости	урок 22
`mesh_clear / mesh_count / mesh_lod / mesh_far / mesh_free`	менять сетку, большие миры	урок 22
`gpu func / mesh_shader / shader_set / screen_shader`	свои шейдеры	урок 22
Звук		
`load_sound`	загрузить WAV ✱	урок 23
`play_sound / loop_sound / play_tone`	играть	урок 23

`play_sound_at / loop_sound_at / move_sound / listener`	звук в 3D	урок 23
`stop_sound / sound_playing / sound_volume / sound_pitch / master_volume`	управление	урок 23
Только в danger		
`new_mem / grow_mem / free_mem / copy_mem / fill_mem`	память руками	урок 25
`read8…64 / write8…64`	числа по адресу	урок 25
`address_of / asm`	адрес списка / команда процессора	урок 25
`zone_new / zone_take / zone_used / zone_clear / zone_free`	зоны	урок 25
`syscall(n, …)`	вызов ядра ОС	урок 26
Только в своём ядре		
`port_in8…32 / port_out8…32`	порты устройств	урок 27
`page_new / page_free`	страницы памяти	урок 27
`interrupts_on / interrupts_off`	прерывания	урок 27
`uptime_ms / read_key / yield`	время, клавиша, уступить	урок 27
`module_count / module_addr / module_size / kernel_symbol / call_address`	программы Omnium	урок 27

Ещё методы: у списка `.add .pop .clear`, у dict `.has .remove .keys .values .clear`, у канала `.send .receive .close .ready`, у фона `.wait .crashed`. Библиотеки `lib/ui.tb`, `lib/physics.tb`, `lib/anim.tb` описаны в `docs/guide.md`.

Указатель 3. Частые сообщения компилятора

Все эти сообщения автор книги получил сам, пока писал примеры. Компилятор пишет номер строки и почти всегда — как исправить.

Сообщение	Что значит и что делать
«это слово занято языком, возьми другое имя»	Имя переменной или параметра совпало со словом языка (`times` , `bool` , `size` у shape...). Переименуйте.
«результат вызова пропадает: сохрани его»	Функция что-то отдаёт (например, номер голоса у `play_sound`), а вы это выбросили. Напишите `v = …` .
«эта функция спрятана в своём модуле: поставь перед ней modpub»	Вы зовёте из другого файла функцию без `modpub` .
«ожидалось значение, а тут: try»	`try` нельзя писать прямо внутри `print(…)` и других выражений. Сначала `x = try f()` , потом `print(x)` .
«лишнее в конце строки: or»	`or` нужен запасной ответ, а у функции без ответа (`net_send`) его нет. Используйте `try` или `else err` .
«не то количество значений в вызове»	Не хватает или лишний аргумент. Компилятор рядом подскажет, как пишется вызов.
«функция может закончиться ошибкой: напиши try f(), f() or запасное или f() else err»	Функция с `fail` (или `read_file` , `to_int` ...) вызвана просто так.
«ожидался знак:)»	Чаще всего — лишнее в описании параметров (например, `link x: int` : у ярлыка тип не пишется).
«стек переполнен: функция зовёт себя слишком глубоко»	Рекурсия без остановки.
«номер за пределами списка / деление на ноль / переполнение int»	Аварии во время работы: программа останавливается и пишет строку.

Дальше. Полный справочник со всеми подробностями — `docs/guide.md` . Готовые программы для чтения — `examples/` (игры, «Файлы», 3D) и `omnium/` (своя ОС на Tachbit).