



Программирование на Python

Введение, настройка окружения, переменные и
функции

Преподаватель курса

- Швайковский Виктор Сергеевич;
- с 2020 года в разработке;
- Python Backend Developer;
- TeamLead команды разработки (20 человек);
- учу 
- решаю задачи на 



@viktor_shv



viktor@shvajkovskij.ru

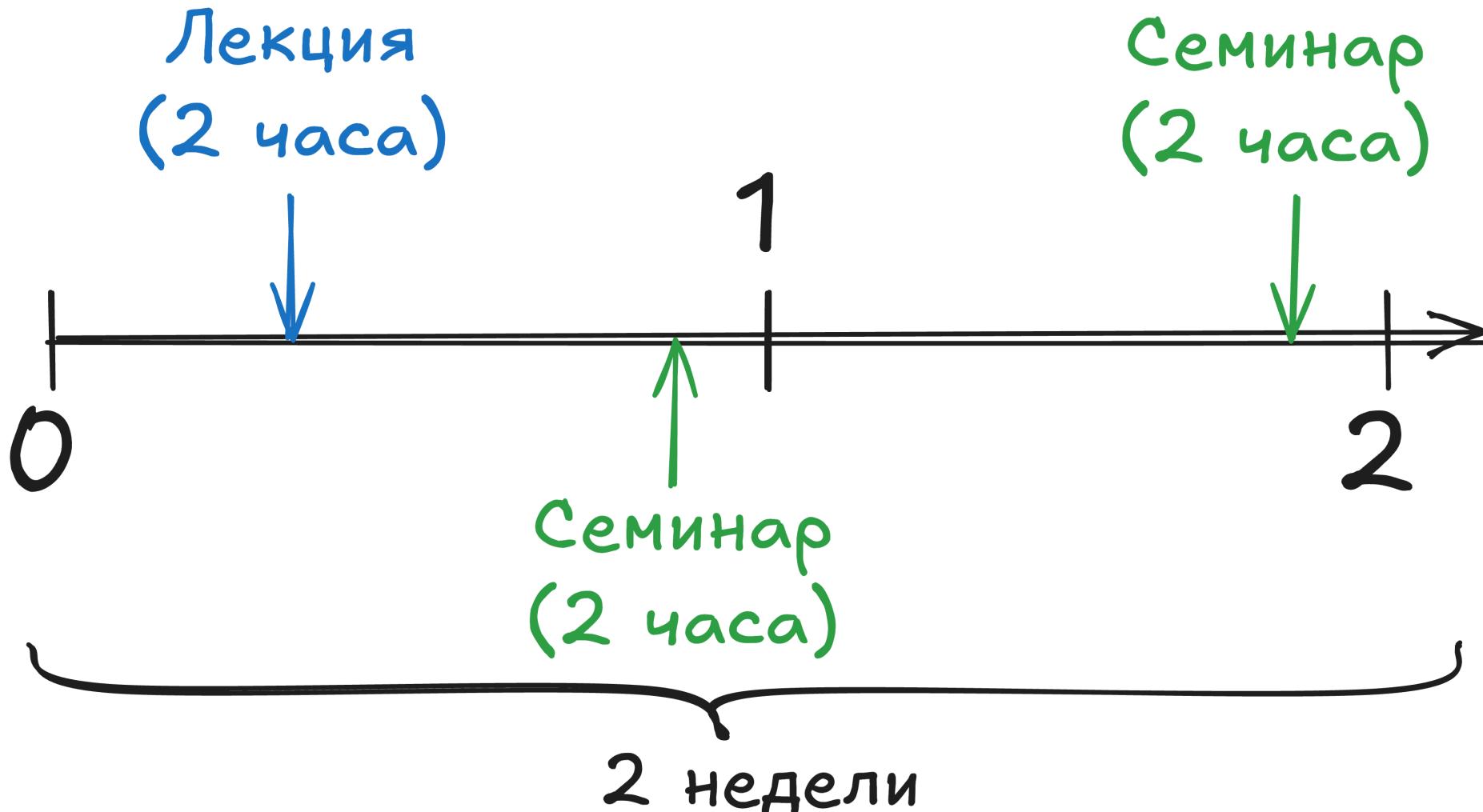
Цели курса

1. **Дать прочный фундамент:** освоить синтаксис, структуры данных и ключевые парадигмы Python.
2. **Научить решать реальные задачи:** применять полученные знания для обработки и анализа данных с помощью библиотек NumPy и Pandas.
3. **Привить культуру разработки:** уверенно использовать Git, настроить профессиональную среду разработки (IDE) и писать чистый код.

Программа курса

- 8 лекций (16 час.)
 - 16 практических занятий (32 час.)
 - сдача и защита домашних заданий
 - экзамен
1. Введение, настройка окружения, переменные и функции.
 2. Условные конструкции, строки, циклы.
 3. Списки, словари, кортежи, множества.
 4. Работа с файлами, обработка исключений, тестирование.
 5. Объектно-ориентированное программирование
 6. Итераторы, генераторы, декораторы.
 7. Основы NumPy.
 8. Основы Pandas.

Таймлайн курса



Критерии оценки

1. Домашние задания (до 80 баллов):

- 8 домашних заданий (ДЗ) по 10 баллов
- баллы за ДЗ начисляются только после успешной устной защиты
- ДЗ, код которого отправлен, но устная защита не пройдена, оценивается в 0 баллов и не считается сданным

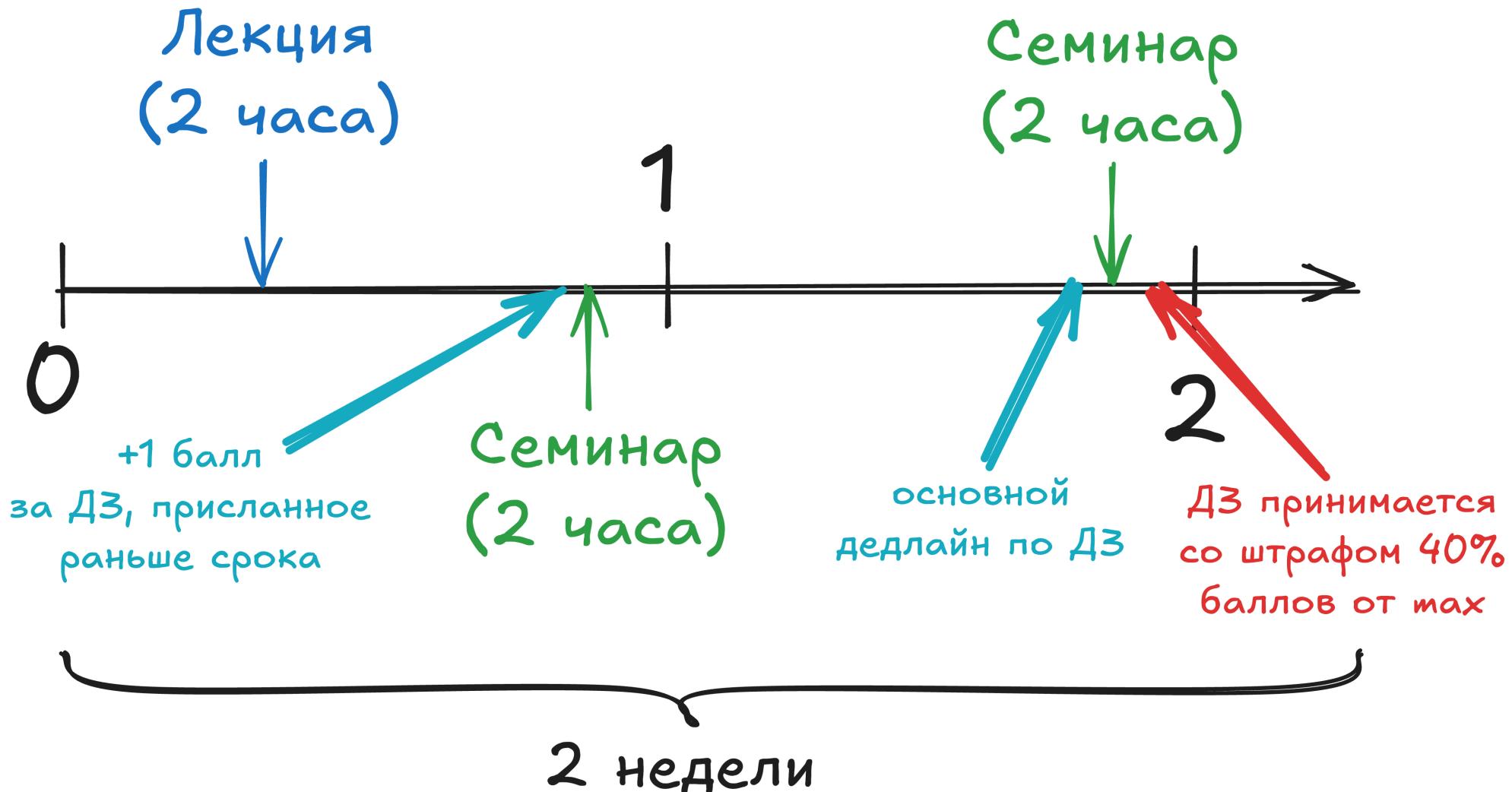
2. Бонусы (до 16 баллов):

- бонусные баллы добавляются к вашим основным баллам и помогают достичь нужного порога
- за посещение лекций: +1 балл за лекцию (макс. +8)
- за скорость: +1 балл за ДЗ, отправленное до первого семинара по лекции (макс. +8)

3. Правила дедлайна:

- ДЗ, отправленное после дедлайна (после второго семинара по лекции), может получить максимум 6 баллов из 10

Дедлайны курса



Критерии оценки

1. Оценка "Отлично" (5) автоматом:

- **условие №1:** сданы и защищены 8 ДЗ (включая NumPy и Pandas).
- **условие №2:** ваша общая сумма баллов – 72 или выше ($\geq 90\%$ от основы).

2. Оценка "Хорошо" (4) автоматом:

- **условие №1:** сданы и защищены минимум 6 ДЗ (включая NumPy и Pandas).
- **условие №2:** ваша общая сумма баллов – 64 или выше ($\geq 80\%$ от основы).

3. Допуск к экзамену:

- **условие №1:** сданы и защищены минимум 4 домашних задания.
- **условие №2:** ваша общая сумма баллов – от 40 до 63 ($\geq 50\%$ от основы).
- *выполнение этих условий дает вам право подтвердить знания на экзамене для получения оценки за курс*

Для студентов с опытом в Python

Если вы уже уверенно программируете на Python и считаете, что владеете темами курса, вы можете подтвердить свои знания, чтобы получить оценку "Отлично" (5) досрочно.

- **Формат подтверждения:**

- Техническое собеседование со мной длительностью 30-40 минут.

- **Что будет на собеседовании:**

- Собеседование состоит из двух частей, имитирующих реальное интервью на позицию Junior Python Developer

- **Code Review вашего проекта:**

- Вы предоставляете ссылку на ваш GitHub-репозиторий с проектом (учебным или личным), который вы считаете показательным.
- Вы должны быть готовы объяснить любую часть кода, принятые архитектурные решения и ответить на вопросы по нему.

- **Live-coding и теория:**

- Решение одной-двух практических задач "с нуля" (на онлайн-платформе или в IDE).
- Устные вопросы по ключевым темам курса: структуры данных (списки, словари, множества), принципы ООП, работа с файлами, основы NumPy/Pandas.

- **Как записаться:**

- Если вы хотите пройти собеседование, напишите мне на почту или в telegram до **12.09.2025**. Также можно записаться после текущей лекции.

Telegram channel



<https://t.me/+NDkPxKEKwMI1M2Uy>

GitHub Repository



[https://github.com/viktor-shvajkovskij/python-
basic-course-fall-2025](https://github.com/viktor-shvajkovskij/python-basic-course-fall-2025)

Процесс сдачи домашних заданий

- зарегистрируйтесь в GitHub <https://github.com>
- создайте приватный репозиторий с названием {surname}-{name}-python-basic-course-fall-2025
- добавьте viktor-shvajkovskij в Collaborators своего репозитория
- склонируйте репозиторий себе локально на компьютер
- создайте ветку homework-n, где n – номер домашнего задания
- в этой ветке добавьте решение в папку homework-n в формате homework_01.ipynb
- отправьте ветку в удаленный репозиторий
- создайте pull request из созданной ветки в ветку main
- укажите viktor-shvajkovskij в поле Reviewers вашего pull request

Пример сдачи домашнего задания

- `git clone https://github.com/ViktorShv95/shvajkovskij-viktor-python-basic-course-fall-2025.git`
- `git checkout -b homework-01`
- `mkdir homework-01`
- `touch homework-01/homework_01.ipynb`
- `git add homework-01/homework_01.ipynb`
- `git commit -m "Add homework_01.ipynb"`
- `git push -u origin homework-01`
- создать pull request в интерфейсе GitHub

Установка Python на Windows

- Поскольку программа установки предельно проста, главная задача на этом этапе – правильно выбрать версию для загрузки. Проверьте заранее, какая версия вам нужна – для 32 или для 64 бит – и какая версия Windows у вас установлена. Затем выполните следующие действия:
- Скачивание установщика.
 - Перейдите на [официальный сайт Python](#).
 - Нажмите на кнопку "Download Python" – установщик автоматически предложит версию, подходящую для вашей системы (например, для Windows 10 или Windows 11).
- Установка.
 - Запустите скачанный файл .exe.
 - На первом экране отметьте галочку "Add Python to PATH" (добавление Python в переменную PATH), чтобы избежать дополнительных настроек.
 - Нажмите "Install Now" для стандартной установки или "Customize Installation" для настройки путей установки.
- Проверка установки.
 - Откройте командную строку (CMD) и введите следующую команду:
 - `python --version`
 - Если Python установлен, появится информация о версии.
- Установка pip.
 - pip автоматически устанавливается с Python. Для проверки выполните команду:
 - `pip --version`
- На Windows Python по умолчанию устанавливается в папку `C:\Users\имя_пользователя\AppData\Local\Programs\Python`. Папка создается автоматически при установке Python. Можно проверить ее наличие, чтобы дополнительно убедиться, что Python установлен на вашем компьютере.

Установка Python на MacOS

- Установить Python на macOS можно через официальный установщик или при помощи системы Homebrew, которая упрощает и облегчает управление программами в этой операционной системе. Выполните следующие действия:
- Установка Homebrew. Установите [Homebrew](#), если его еще нет. В терминале выполните команду:
 - `brew install python`
 - Это установит последнюю версию Python 3.
- Скачивание официального установщика Python. Если вы предпочитаете использовать установщик, скачайте его с [сайта Python](#). Запустите скачанный .pkg файл и следуйте инструкциям.
- Проверка установки. В терминале введите:
 - `python3 --version`
- MacOS использует Python 2 для системных задач, поэтому нужно использовать `python3`. После завершения установки Python будет добавлен в системные пути.

Установка Python на Linux

- Установка Python на Linux может зависеть от вашего дистрибутива. Рассмотрим основные методы.
- **1. Ubuntu/Debian.** Выполните команду:
 - sudo apt update
 - sudo apt install python3 python3-pip
- **2. Fedora.** Используйте:
 - sudo dnf install python3
- **3. Arch Linux.** Введите команду:
 - sudo pacman -S python
- **4. Другие дистрибутивы:** воспользуйтесь менеджером пакетов вашей системы или скачайте исходники с python.org и установите вручную.
- **Проверка установки**
 - В терминале выполните:
 - python3 --version

IDE (Integrated Development Environment)

Visual Studio Code <https://code.visualstudio.com>

1. Бесплатный и быстрый
2. Стандарт индустрии
3. Универсальность
4. Гибкость и расширяемость
 - ms-python.python
 - ms-toolsai.jupyter

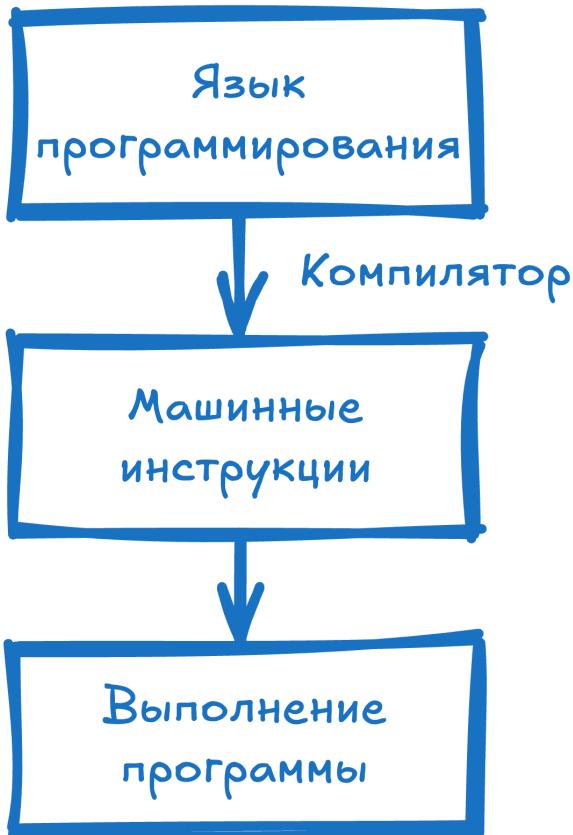
Язык программирования Python

- высокоуровневый
- интерпретируемый язык программирования
- динамическая строгая типизация
- поддержка ООП
- высокая модульность

Компилируемый vs Интерпретируемый

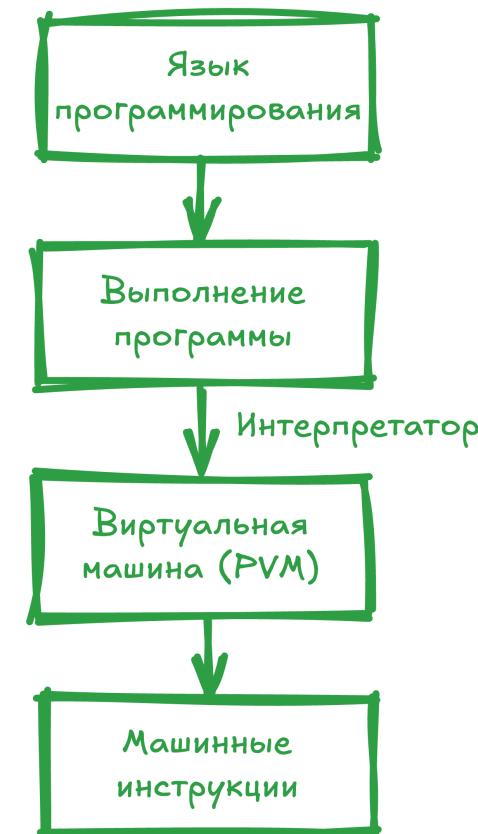
Компилируемый

(C, C++, Go, COBOL)



Интерпретируемый

(Python, PHP, Ruby)

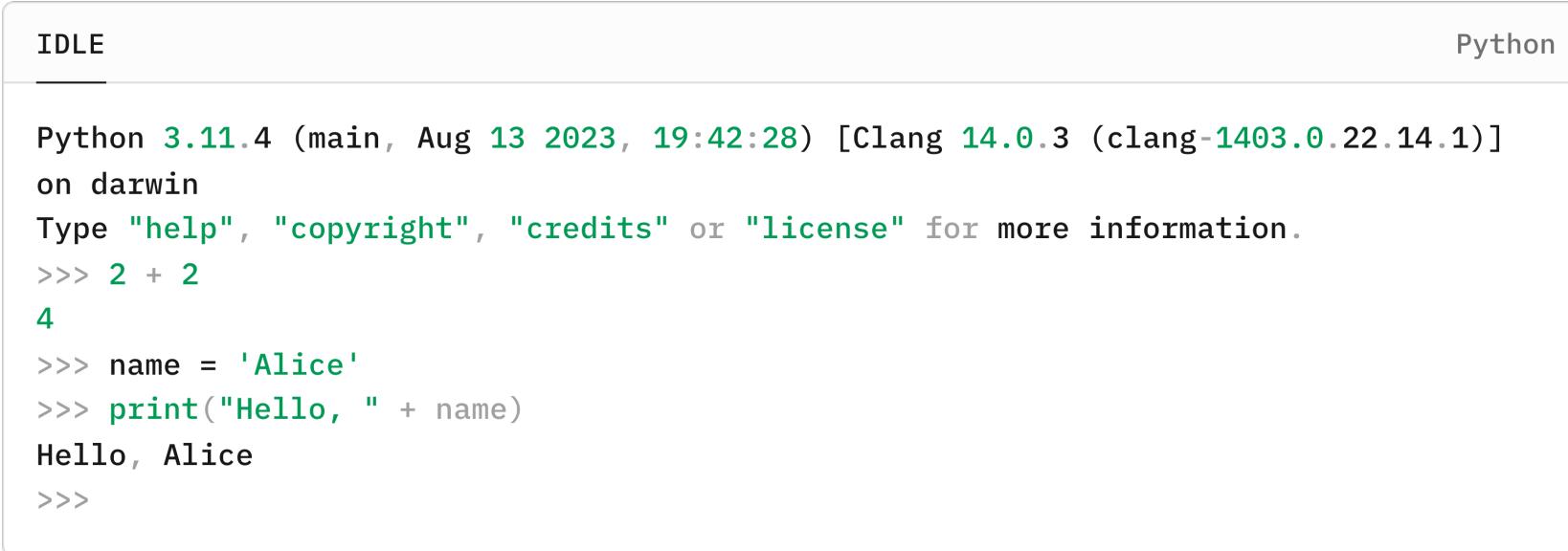


Язык программирования Python

- эталонная реализация языка CPython
<https://github.com/python/cpython>
- интеграция с C/C++, если возможностей Python недостаточно
- автоматическая сборка мусора
- кроссплатформенность

Python в качестве калькулятора

- IDLE (Integrated Development and Learning Environment)
- `python`



```
IDLE Python 3.11.4 (main, Aug 13 2023, 19:42:28) [Clang 14.0.3 (clang-1403.0.22.14.1)]  
on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 2 + 2  
4  
>>> name = 'Alice'  
>>> print("Hello, " + name)  
Hello, Alice  
>>>
```

Скрипты Python

- *.py
- python hello.py

hello.py Python

```
print('Hello, World!')
```

output Python

```
(base) → python-basic-course-fall-2025 git:(main) python hello.py
Hello, World!
```

БЛОКНОТЫ. Jupyter Notebooks

- *.ipynb
- jupyter notebook

The screenshot shows a Jupyter Notebook interface with three code cells:

- Cell 1:** Contains the code `2 + 2`. The output is `4`. Status: 0.0s.
- Cell 2:** Contains the code `print('Hello, Masters!')`. The output is `Hello, Masters!`. Status: 0.0s.
- Cell 3:** Contains the code `a, b = 2, 1000
a ** b`. The output is `107150860718626732094842504906000181056140481170553360744375038837035105112493612249319`. Status: 0.0s.

Toolbar buttons include: Создать (Create), + Code, + Markdown, and a vertical ellipsis.

Арифметические операторы

Арифметическая операция	Оператор
Сложение	+
Вычитание	-
Умножение	*
Деление	/
Целая часть от деления	//
Остаток от деления	%
Возведение в степень	**

Операторы сравнения

Операция сравнения	Оператор
Равно	$==$
Не равно	$!=$
Меньше	$<$
Больше	$>$
Меньше или равно	\leq
Больше или равно	\geq

Операторы присваивания

Операция присваивания	Оператор
Присвоить	=
Прибавить и присвоить	+=
Вычесть и присвоить	-=
Умножить и присвоить	*=
Разделить и присвоить (результат – float)	/=
Взять остаток и присвоить	%=
Разделить нацело и присвоить	//=
Возвести в степень и присвоить	**=

Ключевые слова

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Именование переменных

"Код читают гораздо чаще, чем пишут"

1. Жесткие правила:

1. Допустимые символы: имена могут содержать только буквы (a-z, A-Z), цифры (0-9) и знак подчеркивания (_).
2. Нельзя начинать с цифры:
 - user1
 - 1user
3. Нельзя использовать ключевые слова: вы не можете назвать переменную class, def, for, if и т.д. Ваша IDE обычно подсвечивает их другим цветом.
4. Регистр имеет значение: user и User – это две разные переменные.

Соглашение о стиле (PEP 8)

Стиль	Пример	Где используется в Python
<code>snake_case</code>	<code>user_name, total_price</code>	 Наш стандарт для переменных и функций
<code>camelCase</code>	<code>userName, totalPrice</code>	 (Часто используется в JavaScript)
<code>PascalCase</code>	<code>UserName, TotalPrice</code>	 Только для именования Классов (class UserProfile)

Практические примеры

Плохо

```
x = "Viktor"
```

```
l = [1, 2, 3]
```

```
usr_nm
```

```
UserName
```

```
is_active_boolean
```

Хорошо

```
user_name = "Viktor"
```

```
user_ids = [1, 2, 3]
```

```
user_name
```

```
user_name
```

```
is_active
```

Почему?

Имя должно отражать суть данных, а не быть загадкой.

Будьте конкретны. Не list, а user_ids. Не data, а active_clients.

Не экономьте на буквах.
Читаемость важнее краткости.

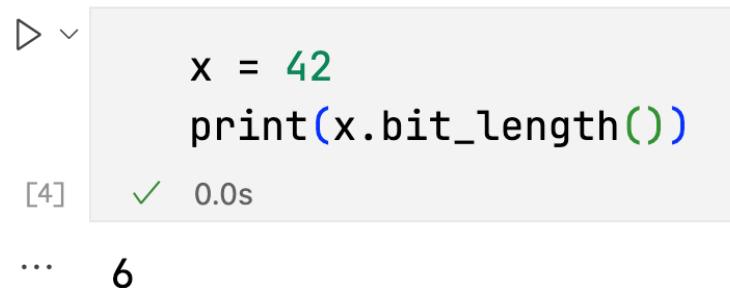
Следуйте стандарту snake_case.
PascalCase зарезервирован для классов.

Не добавляйте тип в имя.
Аннотация типов (: bool) сделает это за вас.

Основные типы объектов

1. Программы состоят из модулей;
2. Модули содержат операторы;
3. Операторы содержат выражения;
4. Выражения создают и обрабатывают объекты;

В python всё является объектом



A screenshot of a Python code editor showing a code cell. The code is:

```
x = 42
print(x.bit_length())
[4]    ✓  0.0s
```

The output shows a green checkmark and the time "0.0s". Below the code cell, there is a ellipsis "... 6" followed by a small number "6".

Числа. Целые числа int

```
a = 42
b = 5
print("a = {}, b = {}".format(a, b)) # форматирование строки методом format
print(type(a))
✓ 0.0s
```

```
a = 42, b = 5
<class 'int'>
```

Числа. Числа с плавающей точкой float

```
print(f"a / b = {a / b}")
print(type(a / b))
```

✓ 0.0s

```
a / b = 8.4
<class 'float'>
```

```
c = 3.1415
d = c ** .5
print(f"c = {c}, d = {d}")
print(f"d = {d:.2f}") # форматирование вывода f-строкой
```

✓ 0.0s

```
c = 3.1415, d = 1.772427713617681
d = 1.77
```

Числа. Комплексные числа

complex

```
e = complex(42, 13)  
print(type(e))
```

✓ 0.0s

<class 'complex'>

```
f = 42 + 13j  
g = 42 + 13J  
print(e == f == g)
```

✓ 0.0s

True

```
imaginary_unit = 1j  
print(imaginary_unit**2)
```

✓ 0.0s

(-1+0j)

```
print(f"{{f.real=}}, {{f.imag=}}")
```

✓ 0.0s

f.real=42.0, f.imag=13.0

Булевый тип bool

- True / False

```
flag = True
print(type(flag))
✓ 0.0s
<class 'bool'>
```

```
print(13 < 42)
✓ 0.0s
```

True

```
print(f"True = {int(True)}")
print(f"False = {int(False)}")
print(isinstance(flag, int)) # True, т.е. булевые значения реализованы как целые числа
✓ 0.0s
True = 1
False = 0
True
```

Булевые операторы

- логическое И – бинарный `and`

```
print(False and False)
print(False and True)
print(True and False)
print(True and True)
```

✓ 0.0s

False
False
False
True

- логическое ИЛИ – бинарный `or`

```
print(False or False)
print(False or True)
print(True or False)
print(True or True)
```

✓ 0.0s

False
True
True
True

- логическое НЕ – унарный `not`

```
print(not False)
print(not True)
```

✓ 0.0s

True
False

None

```
null_variable = None  
print(type(None))
```

✓ 0.0s

<class 'NoneType'>

```
bool(None)
```

✓ 0.0s

False

ФУНКЦИИ

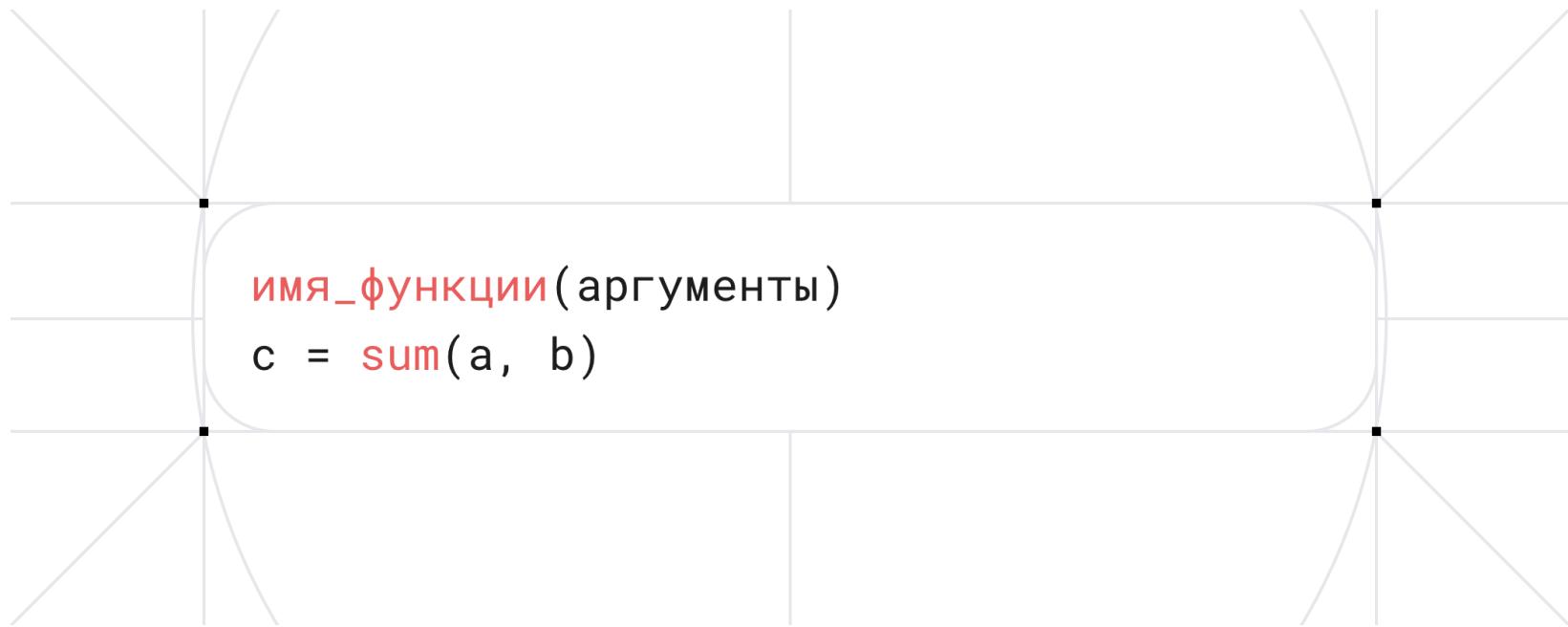
- имя функции – название, с помощью которого можно вызывать функцию в коде;
- аргументы – значения, которые функция принимает на вход. Это поле может быть пустым;
- тело функции – набор инструкций, которые выполняются при вызове;
- результат – значения, которые функция возвращает при завершении работы.

ФУНКЦИИ

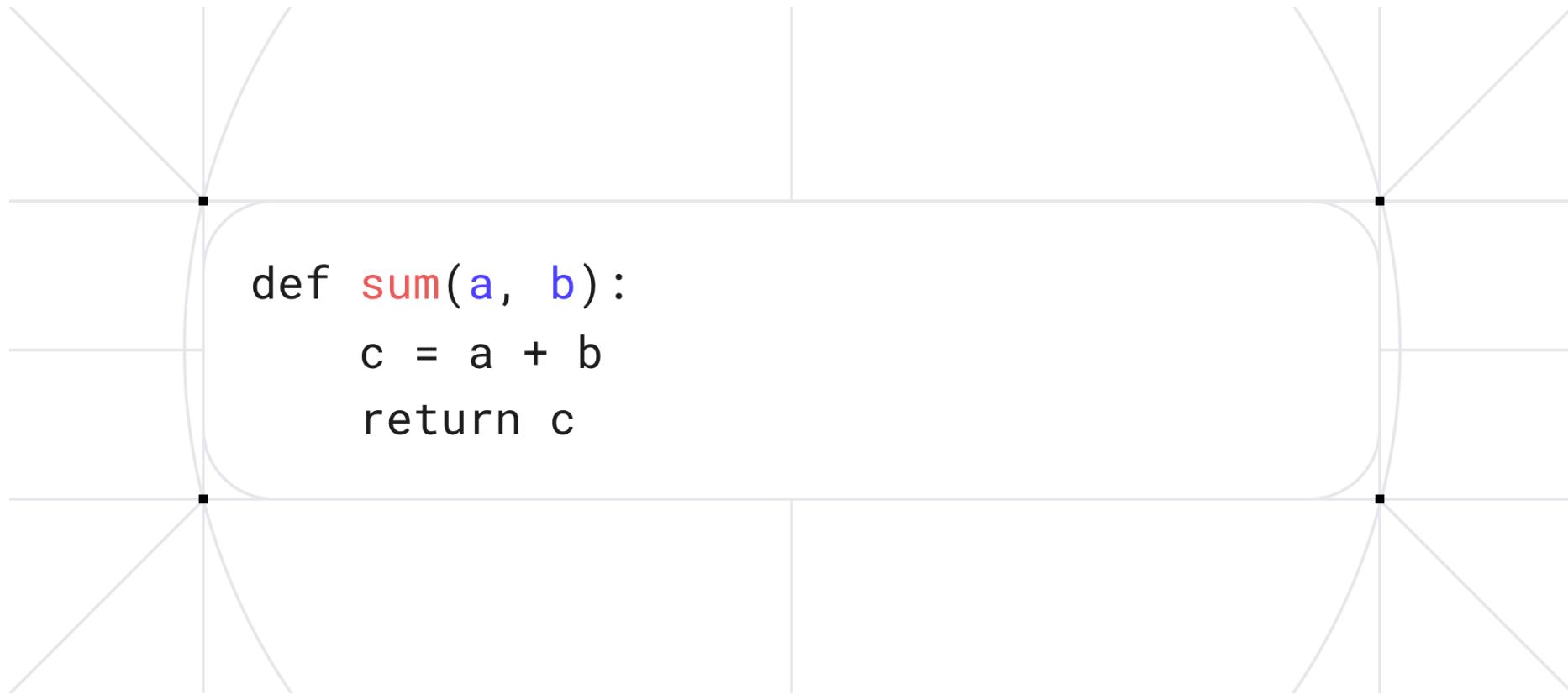
```
def имя_функции (аргументы):  
    тело_функции  
    return результат
```

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

Вызов функции



Локальная область видимости



Область объемлющей функции

```
def make_counter():
    # Объявляем переменную count в объемлющей функции
    count = 0

    def counter():
        # Указываем, что count находится в объемлющей функции
        nonlocal count
        count += 1
        return count

    return counter

# Создаём счётчик
call_counter = make_counter()

# Пример использования счётчика
print(call_counter()) # Вывод: 1
print(call_counter()) # Вывод: 2
print(call_counter()) # Вывод: 3
```

Глобальная область

```
# Глобальная переменная, которая обозначает количество сделанных тортов
cake_count = 10

def modify_cake():
    global cake_count
    # Изменяем значение глобальной переменной
    cake_count = 15

modify_cake()
print(cake_count) # Вывод: 15
```

Аргументы функций

1. Позиционные

- передаются в строго определённом порядке
- значения сопоставляются с параметрами по позиции: первому параметру соответствует первый аргумент, второму – второй и так далее.

2. Именованные

- передаются с указанием имени аргумента
- это позволяет не соблюдать порядок, указанный в определении функции, – если явно указывать имена.

Аргументы функций

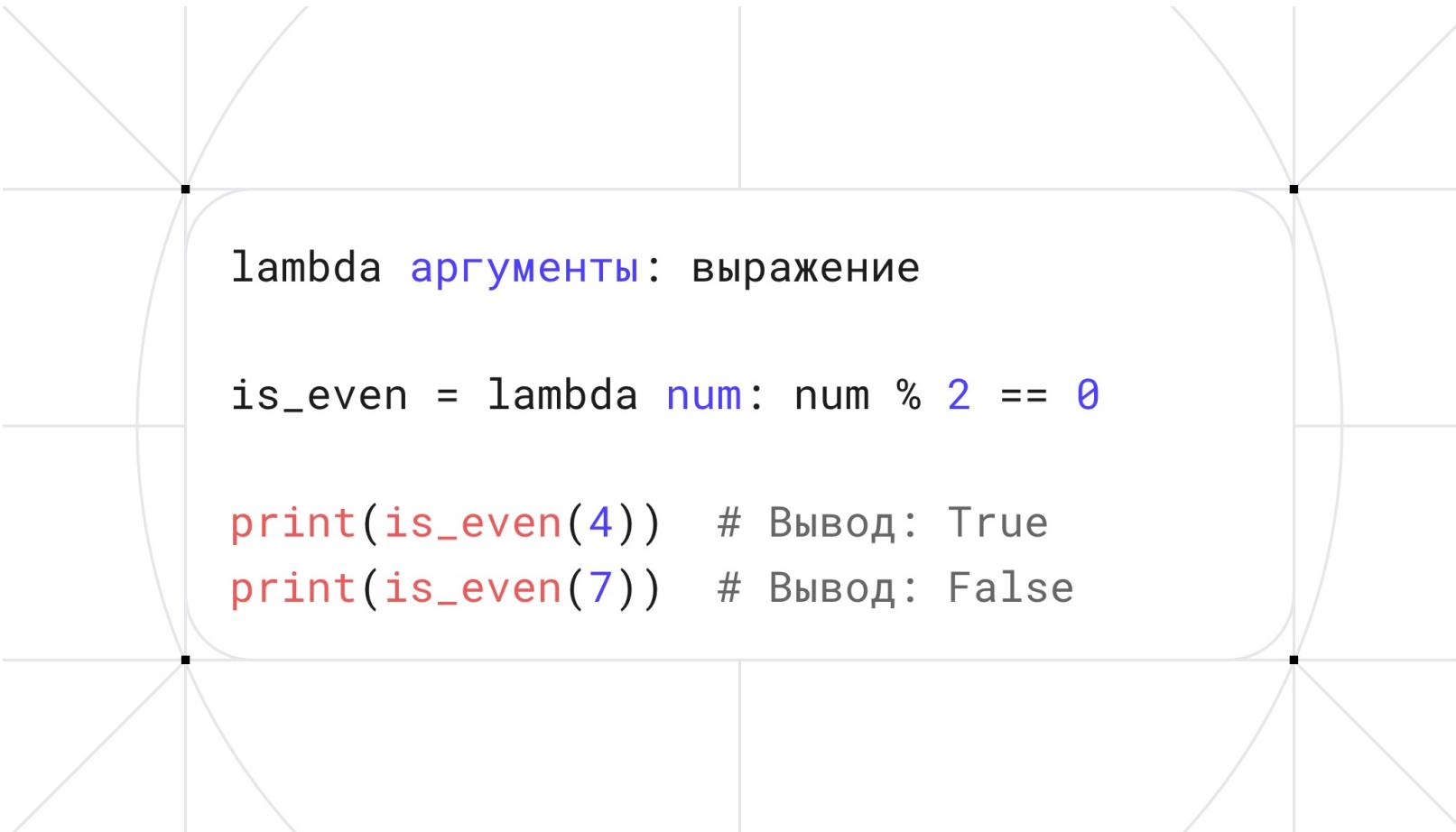
- аргументы переменной длины (*args и **kwargs)
- необязательные аргументы (параметры по умолчанию)

```
def greet(name = 'Посетитель', message = 'Привет, '):  
    print(message, name)
```

```
# Вызов функции без указания значения по умолчанию  
greet() # Вывод: Привет, Посетитель
```

```
# Вызов функции с изменённым значением по умолчанию  
greet('Артём') # Вывод: Привет, Артём
```

Lambda-функции



lambda аргументы: выражение

```
is_even = lambda num: num % 2 == 0
```

```
print(is_even(4)) # Вывод: True
```

```
print(is_even(7)) # Вывод: False
```

Процедуры и функции: различия

- **Процедура** - фрагмент кода, который выполняет определённую задачу или действие, но **ничего не возвращает**.
- **Функция** - тоже фрагмент кода, который выполняет определённую задачу или действие. Но она **возвращает результат** с помощью ключевого слова `return`.

ИТОГ ФУНКЦИИ

- Функции в Python объявляют с помощью ключевого слова `def`, за которым следует имя функции, круглые скобки для аргументов и двоеточие. Тело функции пишется с отступом.
- Python поддерживает несколько видов аргументов в функциях, включая позиционные, именованные, аргументы со значением по умолчанию, а также переменное количество аргументов с помощью `*args` и `**kwargs`.
- Локальные переменные видны только внутри функции, глобальные доступны во всём коде, а вложенные функции имеют доступ к переменным друг друга.
- Функции могут возвращать значения с помощью ключевого слова `return`. Если оно отсутствует, то функция возвращает `None`.
- Помимо стандартных функций, Python поддерживает создание анонимных функций с помощью ключевого слова `lambda`, они могут содержать лишь одно выражение.
- В Python нет явного различия между функциями и процедурами.

Динамическая типизация

```
a = 0                      # a - переменная типа int
print(a, type(a))
a = 0.0                     # теперь a - переменная типа float
print(a, type(a))
a = "zero"                   # теперь a - переменная типа str
print(a, type(a))
a = [0, 0.0, "zero"] # теперь a - переменная типа list
print(a, type(a))
```

✓ 0.0s

```
0 <class 'int'>
0.0 <class 'float'>
zero <class 'str'>
[0, 0.0, 'zero'] <class 'list'>
```

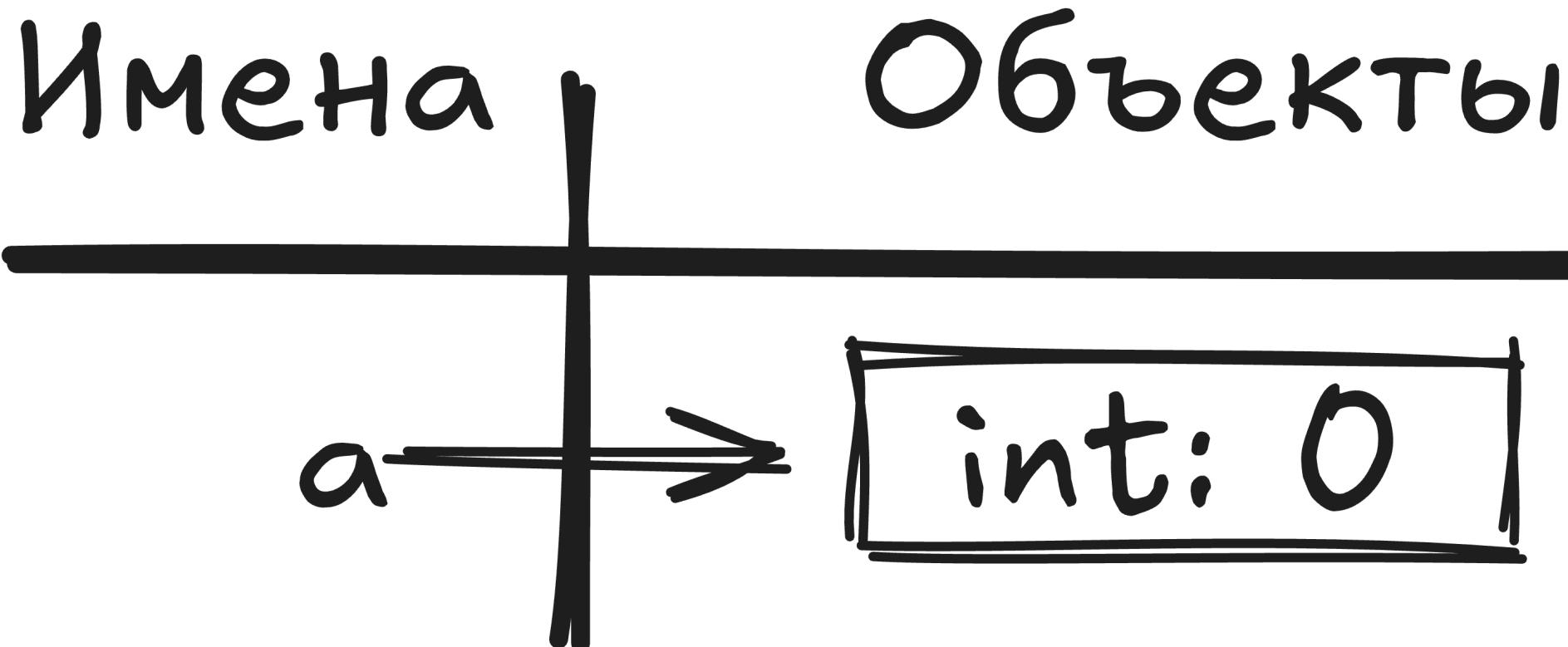
Динамическая типизация

Первая инструкция `a = 0.`

1. Сначала вычисляется значение выражения справа:
создается объект целочисленного
типа `int` содержащий в себе значение `0.`
2. Так как имени `a` до этого объявлено не было, то
такое имя создаётся и связывается с объектом
справа.

```
a = 0          # a - переменная типа int
print(a, type(a))
✓ 0.0s
0 <class 'int'>
```

Динамическая типизация



Динамическая типизация

Вторая инструкция `a = 0.0.`

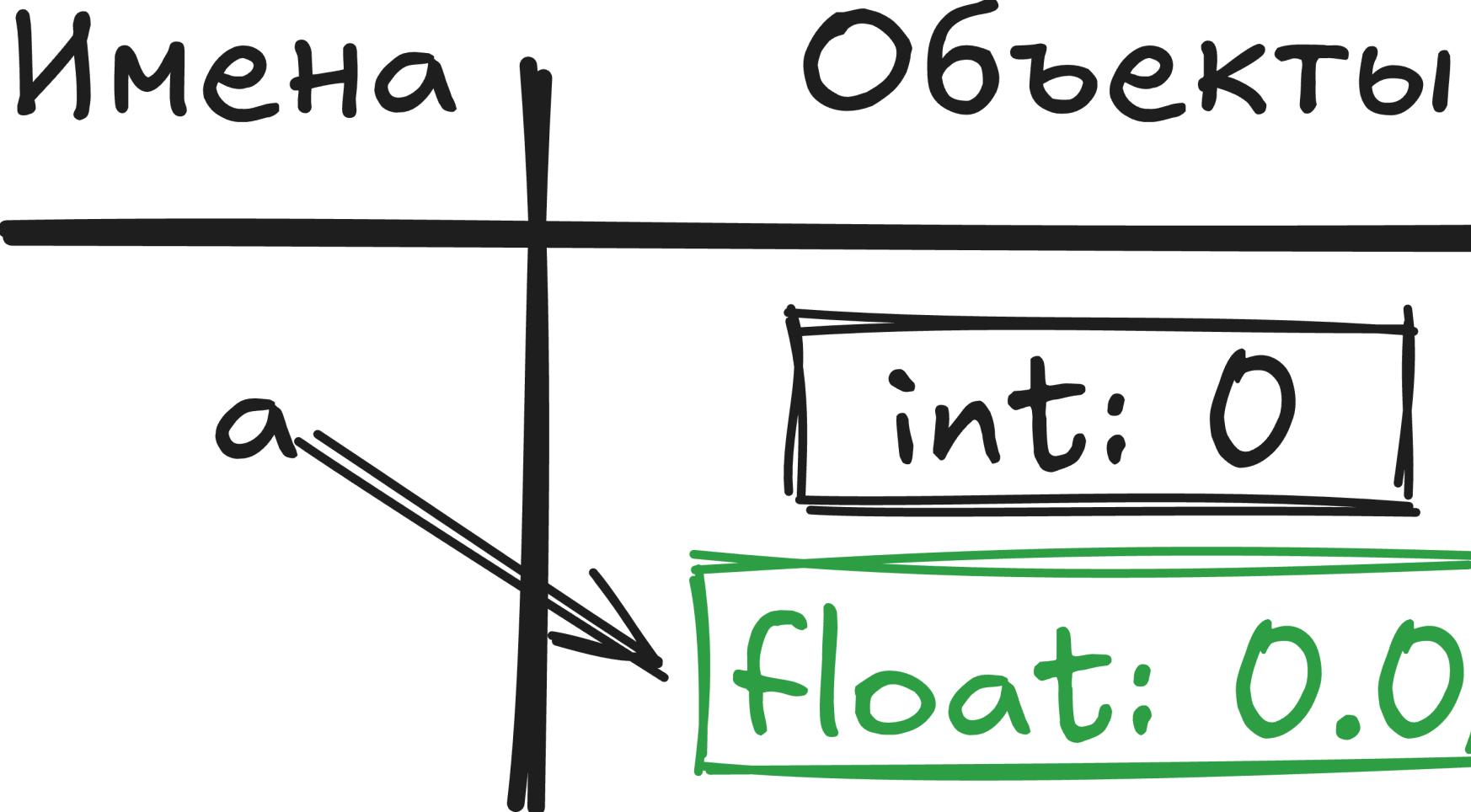
1. Создаётся объект типа `float`, содержащий значение `0.0.`
2. Так как имя `a` уже существует, то оно связывается с новым объектом.

```
a = 0.0          # теперь a - переменная типа float
print(a, type(a))
```

✓ 0.0s

```
0.0 <class 'float'>
```

Динамическая типизация



Динамическая типизация

Третья инструкция `a="zero"`.

1. Создаётся объект строкового типа `str`, содержащий значение `"zero"`.
2. Существующее имя `a` связывается с новым объектом, который имеет другой тип.

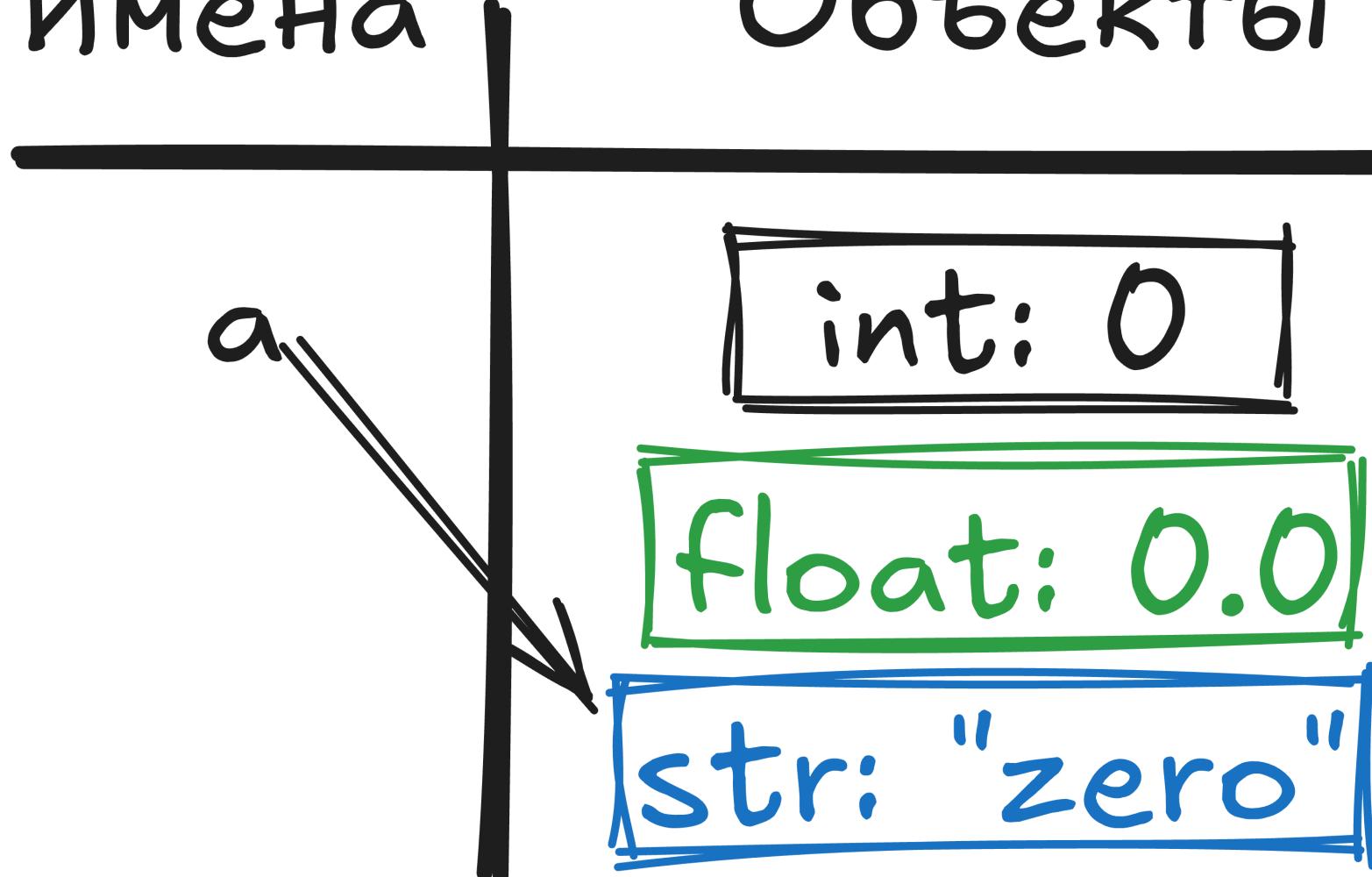
```
a = "zero"          # теперь a - переменная типа str
print(a, type(a))
```

✓ 0.0s

```
zero <class 'str'>
```

Динамическая типизация

Имена Объекты



Динамическая типизация

Четвертая инструкция `a = [0, 0.0, "zero"]`.

1. Создаётся три объекта: целочисленный объект `0`, действительнозначный объект `0.0` и строковый объект `"zero"`.
2. Создаётся объект спискового объект длины 3, ячейки которого связываются с объектами, созданными на предыдущем этапе.
3. Существующее имя `a` связывается с новым объектом, который имеет другой тип.

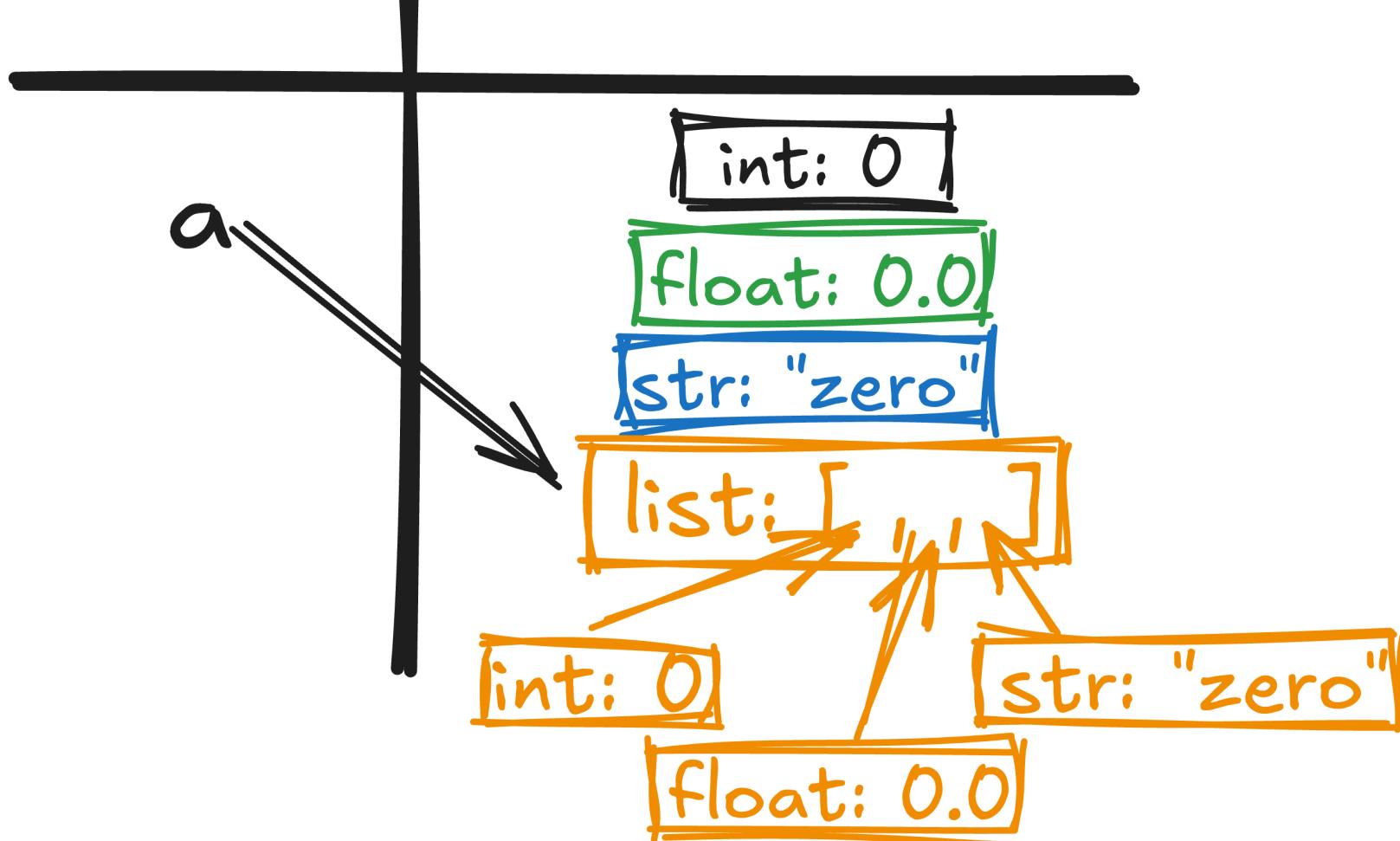
```
a = [0, 0.0, "zero"] # теперь a - переменная типа list
print(a, type(a))
```

✓ 0.0s

```
[0, 0.0, 'zero'] <class 'list'>
```

Динамическая типизация

Имена | Объекты



Утиная типизация

```
1 def f(x):  
2     return x + 1
```

```
1 def duck(x):  
2     x.swim()  
3     x.quack()  
4     return
```

Аннотация типов

```
1 def f(x: int):  
2     return x + 1
```

```
1 def f(x: int) -> int:  
2     return x + 1
```

Преимущества аннотации типов

1. Обнаружение ошибок
2. Документирование кода
3. Автозаполнение в IDE

Превью следующей лекции

- изменяемые и неизменяемый типы объектов
- создание и удаление объектов
- сборщик мусора
- условные конструкции
- строки
- ЦИКЛЫ.

Полезные материалы

- <https://docs.python.org/3/tutorial/index.html>
- <https://docs.jupyter.org/en/latest/install/notebook-classic.html>
- <https://code.visualstudio.com/docs/datascience/jupyter-notebooks>
- <https://peps.python.org/pep-0008/>