Лекция 3. Структура языка программирования.

Элементы языка. Символы. Ключевые слова и идентификаторы. Фундаментальные типы данных. Модификаторы и спецификаторы типов. Переменные. Константы. Примеры простых программ.

# **Введение**

Язык программирования является основным инструментом разработчика для создания программного обеспечения. Понимание его структуры и элементов — ключевой шаг на пути к освоению программирования. В этой статье мы рассмотрим структуру языка программирования, его основные элементы, такие как символы, ключевые слова и идентификаторы. Эти фундаментальные компоненты образуют основу синтаксиса и семантики языка, определяя правила и возможности, которые разработчики используют для написания программ.

Далее, мы погрузимся в фундаментальные типы данных, модификаторы и спецификаторы типов, которые обеспечивают гибкость и контроль при работе с различными видами данных. Типы данных играют важную роль в управлении памятью и эффективным выполнением программ, а модификаторы и спецификаторы позволяют уточнять и расширять их возможности. Мы также обсудим переменные и константы, которые являются основными строительными блоками для хранения и обработки данных в программах.

Наконец, для лучшего понимания теоретических аспектов, мы приведем примеры простых программ, иллюстрирующие использование рассмотренных элементов и структур. Эти примеры помогут закрепить знания и показать практическое применение теоретических концепций в реальных сценариях программирования.

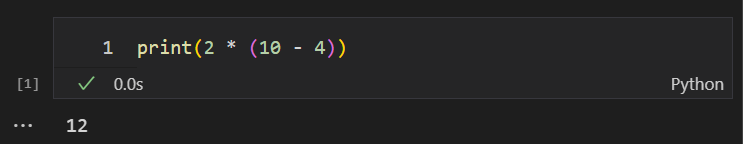
# **Структура программы на языке Python**

Итак, рассмотрим, из чего состоят программы на языке Python. Любая программа на языке Python состоит из **модулей**. Модуль на языке Python представляет собой ряд связанных между собой операций. Модули сохраняются в отдельных файлах с расширением **\*.py**. Сохранённые модули можно использовать в создаваемых программах. Сначала мы будем разрабатывать довольно простые программы, которые практически всегда будут состоять всего из одного модуля.

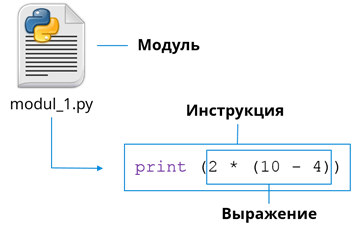
Модули, в свою очередь, состоят из более простых структурных единиц. В модулях содержится код на языке Python, состоящий из **инструкций**.  Инструкции представляют собой указания компьютеру. Они определяют, какие операции выполнит компьютер с данными. Инструкции в языке Python делятся на простые и составные. Простые инструкции описываются одной строкой кода, составные же – содержат вложенные инструкции.

Инструкции могут содержать **выражения**. И если инструкции определяют, какие действия будут выполнены над информацией, то выражения в составе инструкций определяют, над какими именно данными будут выполнены действия, описанные в инструкции.

Вместе с языком Python поставляется множество стандартных модулей, которые предоставляют программисту большое количество инструментов и возможностей для написания самых разных программ.

Рассмотрим пример. Находясь в интерактивном режиме среды разработки, создадим новый файл, в котором запишем несколько инструкций. Сохраним его под именем *modul\_1.py*. Запишем в файле инструкцию вывода print (2 \* (10 – 4)). Сохраним файл и запустим инструкцию на выполнение. В главном окне среды разработки было выведено значение записанного нами выражения, то есть 12.

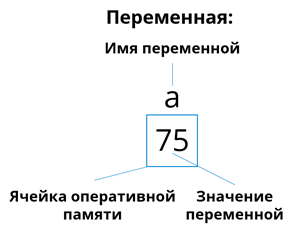
В описанном нами примере, файл modul\_1.py является модулем, строка, записанная нами в этом файле, является инструкцией, а математическое выражение, записанное в скобках, является выражением.



Рассмотрим, что такое операции. Операциями в языках программирования называются любые действия над операндами. Операндами называются некоторые данные. Для примера рассмотрим операции, используемые в описанном нами математическом выражении. Всего их две: разность и умножение. Операндами для операции разности являются числа: 10 и 4. Операндами для операции умножения являются число 2 и разность чисел 10 и 4. Установив скобки в этом выражении, мы определили порядок выполнения операций. Сначала будет выполнена операция разности, записанная в скобках, после чего её результат будет использован в качестве операнды при выполнении умножения. Приоритет выполнения операций в языке Python соответствует математическому.

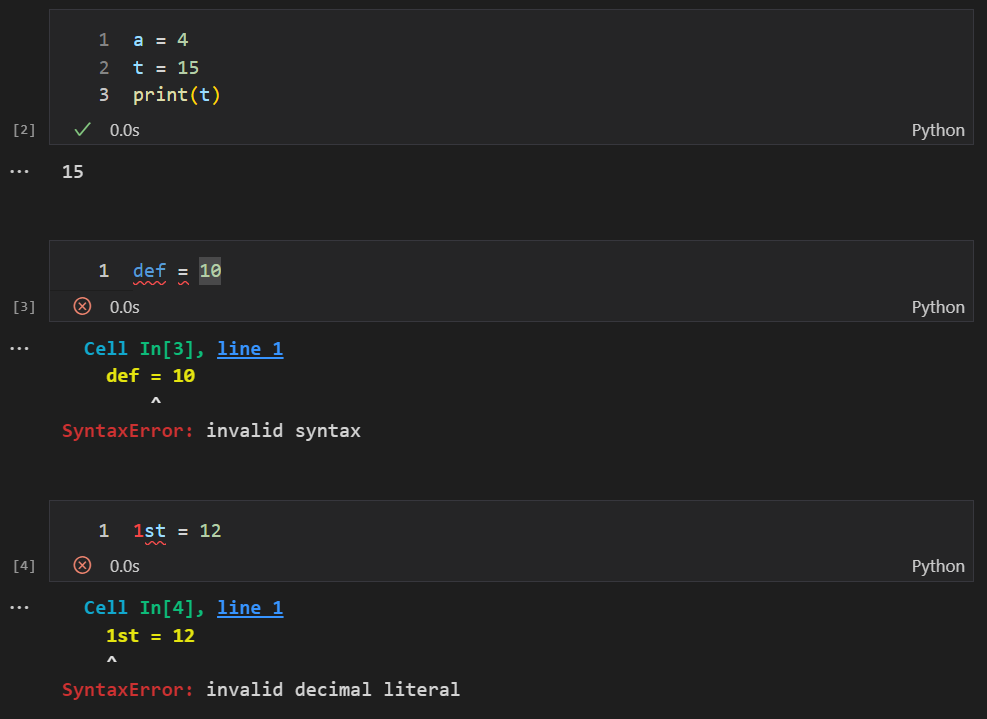
# **Типы данных, переменные, константы**

Операндами для операций могут быть литералы, выражения и переменные. В описанном нами примере операндами разности являются литералы, то есть числа, которые записаны при написании кода, а для операции умножения – литерал, то есть число 2, а также выражение, то есть разность чисел 10 и 4. Операндами могут быть и переменные. Независимо от языка программирования, переменной называется именованная область оперативной памяти, в которой хранится информация определённого типа. Механизм связи между переменной и данными, которые она содержит, может отличаться в зависимости от языка программирования. Пока просто запомним, что данные, сохранённые в переменной, связаны с некоторым именем и могут быть вызваны по этому имени.

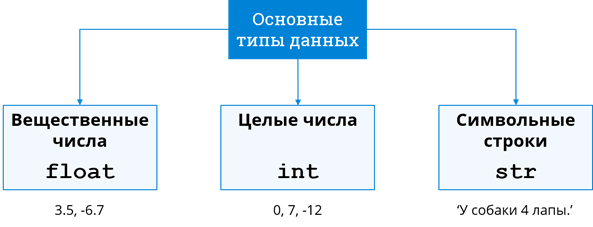


Данные, которые соответствуют переменной в языке Python, могут быть определены с помощью инструкции присваивания. Инструкция присваивания в языке Python записывается с помощью знака равенства, слева от которого находится имя переменной, а справа – её значение. То есть, чтобы переменной а присвоить значение 4, мы должны записать строку кода: а = 4.

Рассмотрим пример. В среде разработки языка Python, в интерактивном режиме, присвоим переменной t значение 15, после чего убедимся, что операция выполнена. Для этого запишем строку кода: t = 15 и нажмём клавишу Enter. Операция присваивания уже была выполнена – убедимся в этом. Для этого запишем инструкцию print (t) для вывода значения переменной t на экран. Таким образом, мы убедились, что переменная t имеет значение 15. Важно запомнить, что переменную стоит называть осмысленно, её имя не должно совпадать со служебными словами языка или содержать служебные символы; имя переменной не может начинаться с цифры.



Рассмотрим, какие типы данных используются в языке программирования Python. Во время написания программ мы будем использовать всего четыре основных типа данных: *целые числа* – int, *вещественные числа* – float, булев тип данных и *символьные строки* – str. К целым числам относятся числа без дробной части, например: 0, 7, - 12 и т. д. К вещественным числам относятся числа с дробной частью, например: 3.5, - 6.7 и так далее… К символьным строкам относятся любые последовательности символов. При вводе они заключаются в двойные или одинарные кавычки, булев тип данных - логические значения **True** (истина) и **False** (ложь).



Всех встроенных типов данных больше:

* Числовые – целые, вещественные, комплексные числа. Примечание: для максимально точных расчетов с десятичными числами в Python используют модуль decimal (тип данных Decimal), а для операций с рациональными числами (дробями) – модуль fractions (тип данных Fraction).
* Булевы – логические значения True (истина) и False (ложь).
* Строковые – последовательности символов в кодировке Unicode.
* NoneType – нейтральное пустое значение, аналогичное null в других языках программирования.
* Последовательности – списки, кортежи, диапазоны.
* Словари – структура данных типа «ключ: значение».
* Множества – контейнеры, содержащие уникальные значения. Подразделяются на изменяемые set и неизменяемые frozenset множества.
* Байтовые типы – bytes (байты), bytearray (изменяемая байтовая строка), memoryview (предоставление доступа к внутренним данным объекта).

В таблице приведены примеры и определения встроенных типов данных:

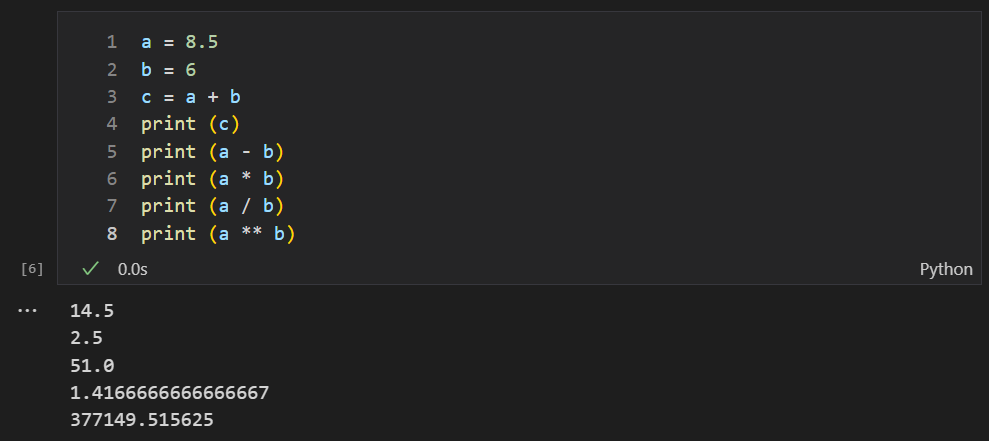
|  |  |  |  |
| --- | --- | --- | --- |
| Тип данных | Значение | Определение в Python | Вариант использования |
| Целые числа | -3, -2, -1, 0, 1, 2, 3 | int | a = int(input()) |
| Вещественные числа | -1.5, -1.1, 0.6, 1.7 | float | a = float(input()) |
| Комплексные числа | −5i, 3+2i | complex | a = complex(input()) |
| Булевы значения | True, False | True, False | flag = True |
| NoneType | None | None | a = None |
| Строка | 'abracadabra' | str | a = str(5) |
| Список | [1, 2, 3], ['a', 'b', 'c'] | list | a = list(('a', 'b', 'c')) |
| Кортеж | ('red', 'blue', 'green') | tuple | a = tuple(('red', 'blue', 'green')) |
| Изменяемое множество | {'black', 'blue', 'white'}, {1, 3, 9, 7} | set | a = set(('black', 'blue', 'white')) |
| Неизменяемое множество | {'red', 'blue', 'green'}, {2, 3, 9, 5} | frozenset | a = frozenset((2, 5, 3, 9)) |
| Диапазон | 0, 1, 2, 3, 4, 5 | range | a = range(6) |
| Словарь | {'color': 'red', 'model': 'VC6', 'dimensions': '30x50'} | dict | a = dict(color='red', model='VC6', dimensions='30x50') |
| Байты | b'\x00\x00\x00' | bytes | a = bytes(3) |
| Байтовая строка | (b'\x00\x00') | bytearray | a = bytearray(2) |
| Просмотр памяти | 0x1477a5813a00 | memoryview | a = memoryview(bytes(15)) |

Те из вас, кто до этого использовал другие языки программирования, например Pascal, знают, что, прежде чем использовать переменную в программе, её нужно объявить. При этом, за переменной закрепляется ячейка оперативной памяти, а также указывается тип данных, которые будут на ней храниться. Возможно, многие из вас обратили внимание на то, что при написании кода для демонстрации работы оператора присваивания, мы не объявляли переменную Тэ, прежде чем присвоить ей значение. Так происходит потому, что в языке Python переменная объявляется автоматически перед первым использованием. Тогда у многих из вас может возникнуть вопрос: «Как определяется тип переменной? Ведь его мы тоже не указывали». В отличие от всё того же языка Pascal, в языке Python используется динамическая типизация. Что это означает? В языке Python тип переменной определяется автоматически, в зависимости от присвоенного ей значения, а так как значение переменной в ходе исполнения программы может изменяться, то и тип переменной также изменяется вслед за значением.

Как же определить, какой тип имеет переменная на данный момент? Для этого в языке Python есть функция type. Посмотрим, как она работает. В среде разработки языка Python, в интерактивном режиме, сначала присвоим переменной a значение 4, после чего запишем инструкцию type (a). На экран был выведен результат выполнения инструкции. В угловых скобках записано слово type, после которого в одинарных кавычках следует тип переменной – int. Теперь присвоим переменной a значение -5.7. В языке Python дробная часть числа отделяется от целой точки. Теперь снова запишем инструкцию type (a). Как видим, тип переменной a изменился на float. Теперь присвоим переменной a символьную строку. Для этого, после знака равенства, в двойных или одинарных кавычках запишем любую последовательность символов, например, слово «Привет». Теперь снова запишем инструкцию type (a). Тип переменной a изменился на str. Для того, чтобы задать число с нулевой дробной частью как вещественное, достаточно просто указать его дробную часть равной нулю. Присвоим переменной a значение 4.0, после чего запишем инструкцию type (a). Как видим, тип переменной a стал float. Важно запомнить, что значения строкового типа не могут употребляться в одном выражении со значениями одного из числовых типов, иначе результатом попытки выполнения инструкции будет сообщение об ошибке.

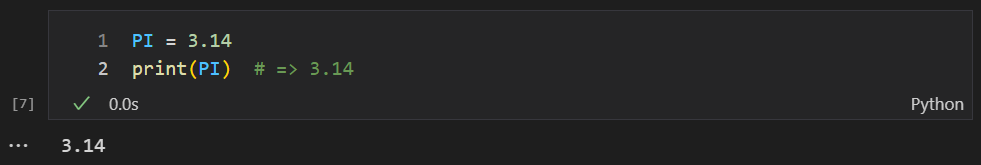
Решим задачу. Написать модуль, в котором переменным a и b присваиваются значения соответственно равные 8.5 и 6, а переменной c – значение суммы a и b. После чего выводятся на экран значения переменной c, разности a и b, их произведения, частного, а также ab.

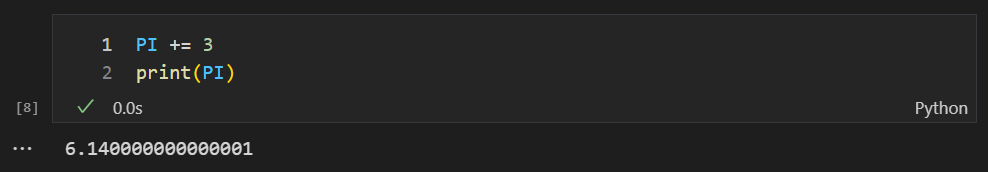
Создадим файл модуля, после чего сохраним его. Начнём написание инструкций. В начале запишем инструкцию для присваивания переменной a значения 8.5. После этого, запишем инструкцию для присваивания переменной b значения 6, а также инструкцию присваивания переменной c значения суммы a и b. Дальше напишем инструкцию print, после которой будут следовать пустые скобки. Скопируем эту инструкцию четыре раза (по количеству выражений, результаты которых необходимо вывести). В первой инструкции print в скобках запишем c, во второй – a – b, в третьей – a \* b, в четвёртой – a / b и в пятой – a \*\* b.

После того как все инструкции записаны, сохраним модуль и запустим его на выполнение. В главном окне среды разработки в пяти строках было выведено пять чисел – результаты перечисленных операций. Модуль работает правильно. Задача решена.

Обратим внимание на то, как выполняются инструкции присваивания в написанном нами модуле. При выполнении первой инструкции сначала выделяется ячейка оперативной памяти для переменной a, после чего проверяется литерал 8.5. Это вещественное число, поэтому тип переменной становится float. Далее в ячейку оперативной памяти заносится значение литерала 8.5. Вторая инструкция выполняется так же, с той лишь разницей, что литерал 6 является целым числом и тип переменной b будет int. При выполнении третьей инструкции присваивания сначала из оперативной памяти извлекаются значения переменных a и b, после чего вычисляется значение их суммы. Далее для переменной c выделяется ячейка оперативной памяти. Так как результатом операции сложения является вещественное число, тип переменной c становится float. В созданную ячейку оперативной памяти заносится результат суммы, то есть четырнадцать целых пять десятых.

**Константы в Python**

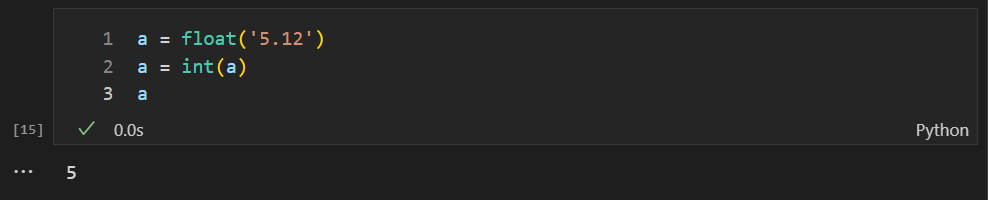
Некоторые данные никогда не меняются — например, математические постоянные. Возьмем для примера число π. Оно всегда равно 3.14 и не может измениться. Чтобы обратиться к подобным данным, в Python используют константы:

Константа создается так же, как переменная. Разница только в том, что константы принято именовать заглавными буквами и с \_ в качестве разделителя между словами. Константа, как и переменная, может использоваться в любом выражении. По своим свойствам константа не отличается от переменной и может быть изменена:

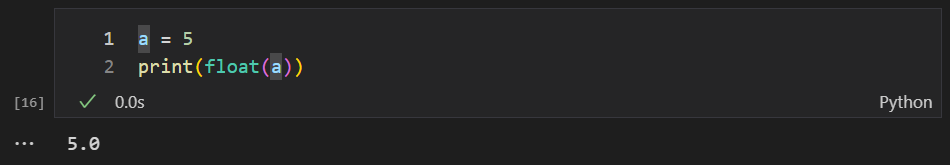
**Преобразование типов данных**

Python позволяет на лету изменять типы данных. Это может оказаться очень полезным при решении тренировочных и практических задач.

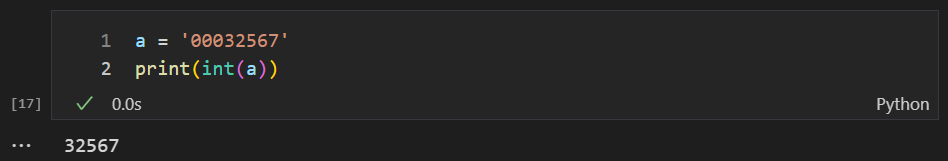
**Округление вещественного числа:**

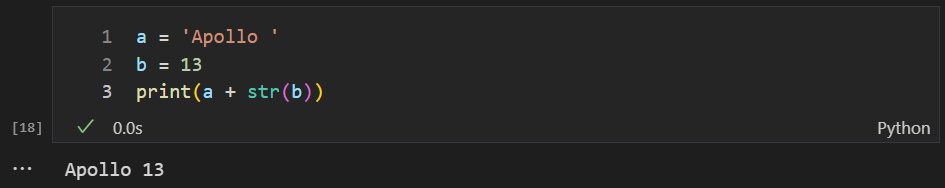


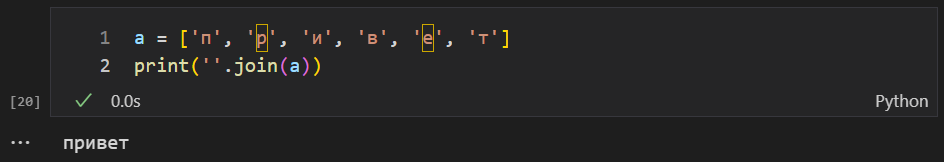
**Преобразование целого числа в вещественное:**

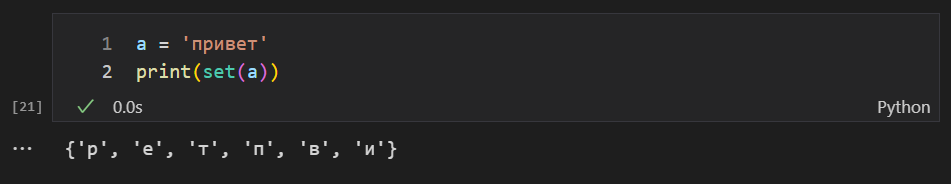
****

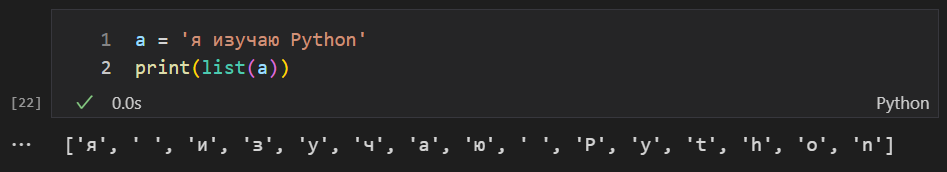
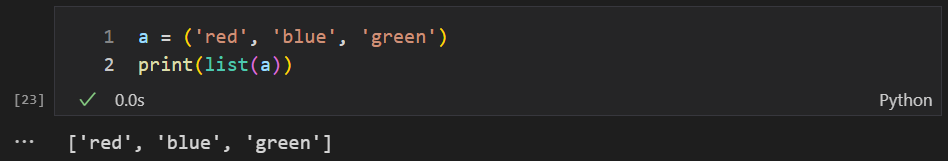
**Преобразование строки в число и вывод числа без ведущих нулей:**

**Сложение строки и числа:**

**Преобразование списка в строку:**

**Преобразование строки в множество:**

**Преобразование строки в список:**

**Преобразование кортежа в список:**

Разумеется, преобразование типов данных происходит в определенных пределах – строку, содержащую буквенные и специальные символы, нельзя сделать числом, а превращение списка, строки, множества или кортежа в словарь потребует дополнительных манипуляций, которые мы изучим позже.

# **Ключевые слова и идентификаторы. Модификаторы и спецификаторы типов**

Ключевые слова — это часть языка, зарезервированные слова в Python. Ключевое слово нельзя использовать в качестве имени переменной, функции или любого другого идентификатора. Они нужны для синтаксиса и структуры языка Python.

Ключевые слова в Python чувствительны к регистру. Все, кроме True, False и None, пишутся в нижнем регистре.

|  |  |
| --- | --- |
| Ключевое слово | Описание |
| False | Ложь |
| True | Истина |
| None | «Пустой» объект |
| and | Логическое *И* |
| with / as | Менеджер контекста |
| assert условие | Возбуждает исключение, если условие ложно |
| break | Выход из цикла |
| class | Пользовательский тип, состоящий из методов и атрибутов |
| continue | Переход на следующую итерацию цикла |
| def | Определение функции |
| del | Удаление объекта |
| elif | В противном случае, если |
| else | Иначе |
| except | Перехватить исключение |
| finally | Вместе с инструкцией try, выполняет инструкции независимо от того, было ли исключение или нет |
| for | Цикл for |
| from | Импорт нескольких функций из модуля |
| global | Позволяет сделать значение переменной, присвоенное ей внутри функции, доступным и за пределами этой функции |
| if | Если |
| import | Импорт модуля |
| in | Проверка на вхождение |
| is | Проверка на идентичность |
| lambda | Определение анонимной функции |
| nonlocal | Позволяет сделать значение переменной, присвоенное ей внутри функции, доступным в объемлющей инструкции |
| not | Логическое *НЕ* |
| or | Логическое *ИЛИ* |
| pass | Ничего не делающая конструкция |
| raise | Возбудить исключение |
| return | Вернуть результат |
| try | Выполнить инструкции, перехватывая исключения |
| while | Цикл while |
| yield | Определение функции-генератора |

Идентификатор — имя некоторого объекта в программе, являющееся его уникальным признаком, позволяющим отличать его от других объектов.

Идентификатор обязательно есть у каждой переменной, функции, объекта и т.п. Идентификаторы в Питоне не ограничены по длине и чувствительны к регистру. То есть А и а — это разные имена переменных или функций.

В идентификаторах допустимы только символы от "A" до "Z" в верхнем и нижнем регистре, подчеркивание "\_" и, кроме первого символа идентификатора, цифры от "0" до "9".

Например, \_\_aAb12\_as111\_1\_4\_5 — корректный идентификатор, а 1z — некорректный, т.к. начинается с цифры.

Для идентификаторов на Питоне версии 3 можно использовать не только ASCII символы, но и Unicode. На практике это означает, что переменные можно называть по-русски.

**Правила записи идентификаторов**

1. В записи идентификаторов можно использовать латинские буквы в нижнем регистре (от a до z), в верхнем регистре (от A до Z), цифры (от 0 до 9) и символ подчеркивания \_. Например, допустимыми будут такие идентификаторы: myClass, var\_1 и print\_this\_to\_screen.

2. Идентификатор не может начинаться с цифры. 1variable не может быть идентификатором, а вот variable1 — допустимое имя.

3. Ключевые слова нельзя использовать в качестве идентификаторов.

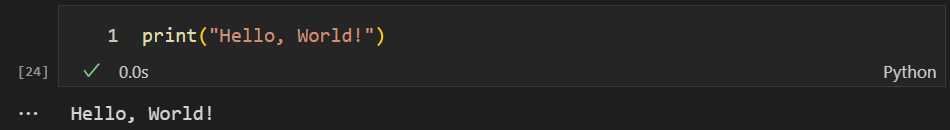
global = 1

4. При записи идентификатора нельзя использовать специальные символы, например: ! , @ , # , $ , % и т. д.

5. Идентификатор может быть любой длины.

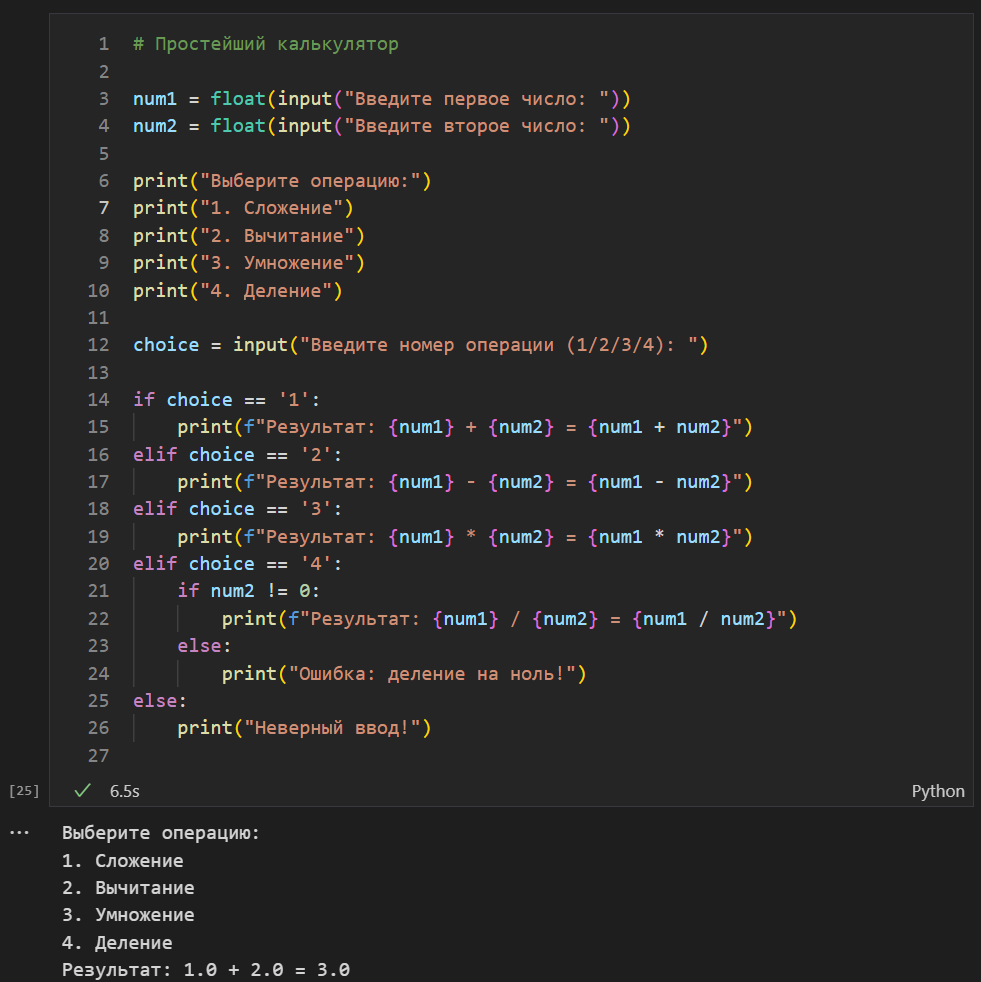
# **4. Примеры простых программ**

Первая программа, которую пишет большинство начинающих программистов, это "Hello, World!". Она выводит на экран текстовое сообщение и служит для демонстрации базового синтаксиса языка.

Эта программа состоит из одной строки кода. Функция print() используется для вывода текста на экран. В Python нет необходимости указывать типы переменных или использовать сложные структуры для такой простой задачи, что делает язык доступным для новичков.

**Калькулятор**

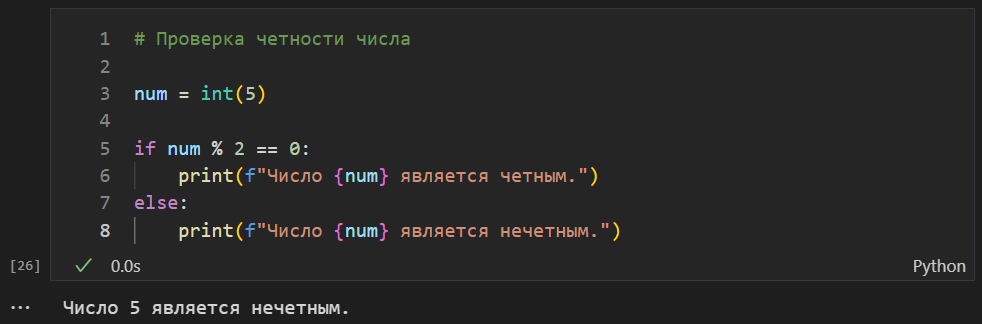
Простейший калькулятор — это следующая программа, которую часто создают начинающие программисты. Он принимает два числа и выполняет арифметические операции (сложение, вычитание, умножение и деление).

В этой программе используются функции input() для получения ввода от пользователя и float() для преобразования строки в число с плавающей точкой. Условия if-elif-else используются для выполнения различных операций в зависимости от выбора пользователя.

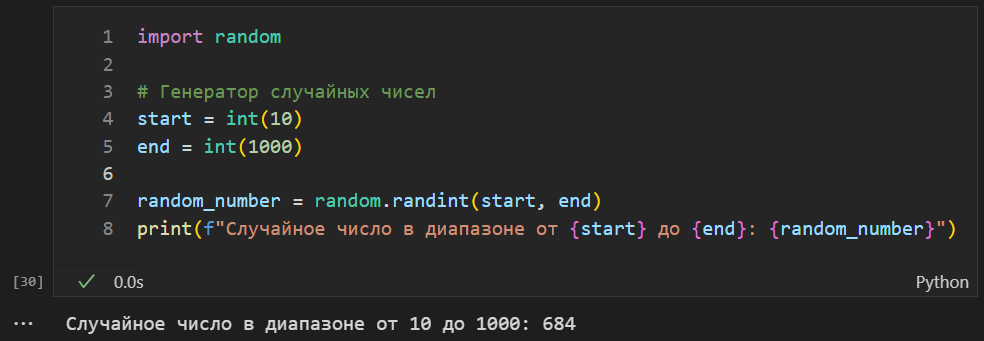
**Проверка четности числа**

Программа для определения, является ли число четным или нечетным, демонстрирует использование условий и операторов.

Здесь используется оператор % (остаток от деления) для проверки, делится ли число на 2 без остатка. В зависимости от результата, программа выводит соответствующее сообщение.

**Генератор случайных чисел**

Генератор случайных чисел демонстрирует работу с библиотекой

Функция random.randint() используется для генерации случайного целого числа в заданном диапазоне. Это полезно для создания случайных значений в различных приложениях, таких как игры или симуляции.

# **Заключение**

Понимание структуры и основных элементов языка программирования является фундаментом для эффективного написания кода и создания сложных программных решений. Рассмотренные в статье символы, ключевые слова, идентификаторы и фундаментальные типы данных обеспечивают базис, необходимый для построения логики и выполнения задач. Эти элементы образуют синтаксическую и семантическую основу языка, позволяя разработчикам выразить алгоритмы и структуры данных с необходимой точностью и ясностью.

Модификаторы и спецификаторы типов играют важную роль в расширении возможностей типов данных, предоставляя дополнительные инструменты для управления памятью и оптимизации производительности. Переменные и константы, будучи основными элементами хранения данных, позволяют организовывать и эффективно управлять информацией в программах. Использование этих компонентов в различных комбинациях дает возможность создавать мощные и гибкие приложения, удовлетворяющие самым разнообразным требованиям.

Примеры простых программ, представленные в статье, иллюстрируют практическое применение теоретических знаний, демонстрируя, как базовые элементы и структуры языка программирования могут быть использованы для решения реальных задач. Эти примеры служат мостом между теорией и практикой, помогая закрепить полученные знания и вдохновляя на дальнейшее изучение и совершенствование навыков программирования. В итоге, глубокое понимание структуры языка и его элементов открывает двери к эффективному и креативному программированию, позволяя создавать высококачественное программное обеспечение.

# **Список литературы**

1. https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-3-tipy-dannyh-preobrazovanie-i-bazovye-operacii-2022-10-14

2. https://videouroki.net/video/02-struktura-programmy-na-yazyke-python-operacii-peremennye-i-literaly-tipy-dannyh.html

3. https://codechick.io/tutorials/python/keywords-identifier

4. https://foxford.ru/wiki/informatika/klyuchevye-slova-i-identifikatory-v-python?utm\_referrer=https%3A%2F%2Fwww.google.com%2F

5. https://letpy.com/handbook/keywords/