

Шпаргалка: переменные и типы данных

Кратко

Что	Как	О чём помнить
Задать переменную	<code><имя переменной> = <значение></code> Пример <code>password = 8</code>	Как называть переменные: Только цифры и латинские буквы. Имя начинается не с цифры. Вместо пробелов — нижние подчеркивания. Название отражает суть переменной. Слова написаны на английском. Всё с маленькой буквы.
Типы данных: <code>int</code> , <code>float</code> , <code>string</code>	<code>password = 8</code> <code>number = 5.4</code> <code>name = 'Акакий'</code> или <code>name = "Акакий"</code>	Для строк кавычки могут быть <code>'одинарными'</code> либо <code>"двойными"</code> . Можно и так, и так. Главное, чтобы открывающая и закрывающая были одинаковыми. С разными будет ошибка.
Разделить числа	<code>print(4 / 2)</code> напечатается 2.0 <code>print(4 // 2)</code> напечатается 2	Обычно если делят целые числа, используют <code>//</code> , чтобы получилось тоже целое число. А если дробные, используют <code>/</code> .

Сложить строки	<p>Через +:</p> <pre>print('Какая полезная ' + 'шпаргалка')</pre> <p>Через f-string:</p> <pre>name = 'шпаргалка' print(f'Какая полезная {name}')</pre>	<p>Для +. Оператор сложения не заменяется на пробел. Поэтому нужно ставить его перед кавычками, иначе слова слипнутся: 'Какая полезная '.</p> <p>У конкатенации есть краткая запись: +=.</p> <p>Для f-string. Если нужно вывести фигурные скобки текстом, окружи их ещё одной парой скобок. Это называется экранировать: print(f'Фигурные скобки: {{}}'). Выведет «Фигурные скобки: {}».</p> <p>Если значение переменной нужно вывести в кавычках, окружи ими фигурные скобки: "{name}".</p>
Повторить строку несколько раз	<pre>print(<что нужно повторить> * <сколько раз>)</pre> <p>Пример</p> <pre>la_la_land = 'la' print(la_la_land * 10)</pre> <p>получится <u>lalalalalalalalalala</u></p>	<p>Со сложением так не работает.</p> <p>Есть краткая запись *=.</p>
Найти остаток от деления	<pre>print(10 % 10) получится 0 print(10 % 7) получится 3</pre>	<p>Если число слева от оператора % меньше числа справа, остаток будет равен числу слева.</p> <pre>print(3 % 7) выдаст 3</pre>
Присваивание и операция: +=, -=, *=, /=, //= и %=	<pre>numbers = numbers - 12 ↓ numbers -= 12</pre>	
Число → строка	<pre>str(57.2) получится строка «57.2» str(57) получится строка «57»</pre>	

Строка → число	<code>int('5')</code> получится число 5 <code>float("57.2")</code> получится число 57.2	
Дробное число → целое	<code>int(2.72)</code> выдаст 2	<code>int()</code> не округляет числа по правилам арифметики, а просто отбрасывает дробную часть. Функция <code>int()</code> одинаково работает с положительными и отрицательными числами: <code>int(-3.14)</code> выдаст -3
Несколько преобразований сразу	<code>fraction = 1.5</code> <code>print("Целая часть = " + str(int(fraction)))</code> Вернётся «Целая часть = 1»	Важно не сбиться в количестве скобок: закрывающих должно быть столько же, сколько и открывающих Для операций с целыми и дробными числами приводить их к одному типу не нужно, тут Python разберётся
Напечатать что-то	<code>print('Какая полезная шпаргалка')</code> <code>print(12)</code>	Когда пишешь несколько аргументов, запятая между ними заменится на пробел. Такой код: <code>print('Все', 'автотесты', 'выполнены')</code> Выведет: Все автотесты выполнены Иногда пробелы ставят, чтобы код было удобнее читать.

Подробно

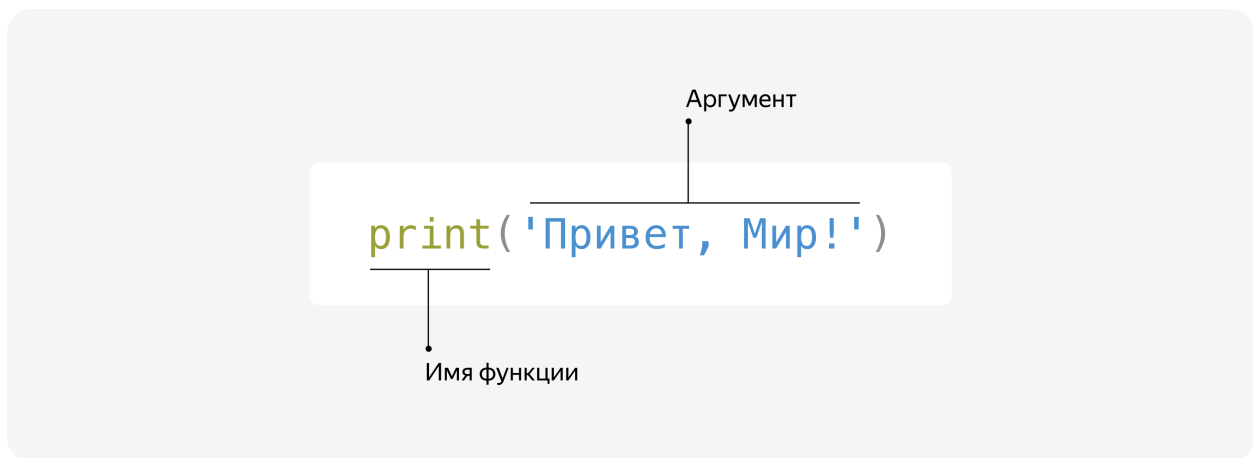
Функции

Функция — это команда, которая выполняет какое-то действие. Например, печатает текст или складывает числа.

Можно сказать, это такая маленькая подпрограмма в программе. Скажем, код проводит расчёты, а отдельные функции в нём — складывают, умножают и выводят результат.

Аргументы

В скобках у функции стоят **аргументы** — это то, что функция забирает и обрабатывает. В случае с `print()` это то, что тебе нужно напечатать.



Говорят, что аргументы **передаются** в функцию. А функция их **принимает**.

Комментарии в коде

Автоматизаторы тестирования и разработчики пишут большие программы с сотнями строк кода. Чтобы не запутаться, можно оставлять комментарии: заметки, какая команда для чего нужна.

Python не считывает комментарии как часть программы. Компьютер их не видит: они нужны только для другого человека.

Чтобы оставить комментарий, нужно поставить символ `#`. Например, так:

```
# Приветствие миру — традиционная первая строка в освоении нового языка
print('Привет, Мир!')
```

Когда комментариев много, нужно ставить `#` для каждой строки:

```
# Иногда бывает, что комментарии длинные
# В одну строчку они не уместятся
# Поэтому можно разбить на несколько строчек
# Получается ну просто Маяковский-стиль
print('Как удобно использовать комментарии!')
```

Функция `print()`

Команда выведет на экран любые данные, которые ты укажешь в скобках.

Например, `print('Яндекс')` или `print(12)`.

Кавычки в `print()`. Если хочешь вывести фразу, нужно заключить её в двойные или одинарные кавычки: `"Яндекс"` или `'Яндекс'`. Для чисел кавычки не нужны. В Python можно использовать и двойные кавычки, и одинарные. Разницы нет.

Запятая между аргументами

В скобках может стоять несколько аргументов: функция напечатает их все сразу. Например:

- две строки — `print('Быть или не быть', 'вот в чём вопрос');`
- два числа — `print(1, 2);`
- строка и число — `print('Ответ на главный вопрос жизни, вселенной и всего такого', 42);`

Когда пишешь несколько аргументов, запятая между ними заменится на пробел. Такой код:

```
print('Все', 'автотесты', 'выполнены')
```

Выведет:

```
Все автотесты выполнены
```

Иногда пробелы ставят, чтобы код было удобнее читать. Но на работу программы это не повлияет. Такой код выведет всё то же самое, что и выше:

```
print('Все', 'автотесты', 'выполнены')
```

Когда составляешь сложную фразу, можно не добавлять пробелы. Достаточно разделить аргументы функции `print()` запятой:

```
print('У тебя', 12, 'новых сообщений.')
```

Что такое переменная

Переменная — такая ячейка в программе, которая хранит какое-то значение.

Она похожа на коробку, в которую можно положить данные: число или строку текста. И достать или поменять его позже, когда оно понадобится.

Например, ты пишешь автотест, который проверяет пароль на сайте.

Минимальная длина пароля — восемь символов. Это значение можно положить в переменную. А потом достать его оттуда и сравнить с тем, что ввёл пользователь.

Как объявить переменную

Для этого нужно:

- Придумать имя. Например, нужно сохранить количество символов в пароле на сайте. Подойдёт имя `password`. Для названия переменной есть несколько правил, о которых ты узнаешь в следующем уроке.
- Задать значение через знак `=`. Например, `password = 8`.

Вот так объявили переменную `password`:

```
password = 8
```

Знак «равно» `=` называется **оператором присваивания**. В программировании выражение `x = 1` означает «теперь в переменной `x` хранится значение `1`».

Как заменить значение переменной

Если ты присвоишь переменной значение, это не значит, что оно будет храниться там вечно. Его можно заменить.

Например, сначала нужно запомнить число 8:

```
password = 8
```

А потом правила поменялись, и теперь уже нужно сохранить 6:

```
password = 8  
password = 6
```

Всё, значение поменялось. Теперь в переменной хранится не 8, а 6.

Как использовать переменную в программе. Когда переменная объявлена и ей присвоено значение, её имя можно указывать в коде. Например, как аргумент. Вместо имени будет подставляться её значение.

Код печатает фразу «Привет, Мир!». Она хранится в переменной `message` и передаётся как аргумент в `print()`.

```
message = 'Привет, Мир!'  
# В функцию передан не текст, а переменная, в которой хранится текст:  
print(message)
```

Как называть переменные

Только цифры и латинские буквы. Например, `result` или `salary123`. Если поставить символ вроде `?` или `#`, будет ошибка.

Имя начинается не с цифры. К примеру, `100500salary` — не подойдёт. Такое имя Python не прочтает.

Нет пробелов. Если поставить пробел, будет ошибка: `new message` — не подойдёт.

Если в имени несколько слов, разделяй их символом нижнего подчёркивания:

`new_message`. Это считается хорошим тоном. Если не разделить и написать слитно `newmessage`, ошибки не будет. Просто смотрится неаккуратно.



Такой стиль написания называется **snake case** — «змеиный стиль». Слова могут получаться очень длинные, вот и напоминают змею. Например, `new_password_for_new_user`.

Есть ещё пара пунктов. Если их нарушить, ошибки не будет — программа сработает. Но они считаются правилами хорошего тона: так легче читать код.

Название отражает суть переменной. Старайся подбирать такие имена, чтобы было понятно, что именно в них хранится. Например, `expected_result` — для результата, который ожидаешь. Так и тебе, и коллегам будет проще.

Вот пример. Переменную с названием урока можно назвать буквой `l`, а можно — словом `lesson`. Второй вариант лучше: сокращение до одной буквы со временем забудут. Код станет сложнее читать.

Слова написаны на английском. Лучше не использовать русские слова в английской раскладке. Рано или поздно твой код прочтёт человек, который не знает русский язык. Он может не понять, что к чему. Сразу называй переменные по-английски: `child`, а не `rebyonok`.

Слова с маленькой буквы. Так нужно, чтобы переменные не путались с другими сущностями в коде. Например, с классами, о которых ты узнаешь позднее. Так что будет переменная `joanne_rowling`, а не `Joanne_Rowling`.

Типы данных

Строки

Это символ или набор символов внутри кавычек.

Какие кавычки ставить. Кавычки могут быть `'одинарными'` либо `"двойными"`. Можно и так, и так. Главное, чтобы открывающая и закрывающая были одинаковыми.

```
title = 'Французский вестник' # Строка в одинарных кавычках
director = "Уэс Андерсон" # Строка в двойных кавычках
year = '2021' # Число в кавычках тоже становится строкой
space = ' ' # Даже пробел — это тоже строка
```

Если написать разные кавычки, будет ошибка.

```
# Такой код ломает программу:
# открывающие и закрывающие кавычки должны быть одинаковыми
director = "Уэс Андерсон'
```


Целые числа

Называются типом данных `int` — от английского integer, «целое число».

С числами кавычки ставить не нужно:

```
twenty_five = 25
about_pi = 3
```

Обрати внимание: в редакторе кода числа и строки выделяются разными цветами, чтобы читать программу было легче.

Тут есть тонкость. Допустим, нужно задать переменной значение «3». Это можно сделать двумя способами:

```
a = '3'
a = 3
```

Разница в том, что типы данных будут отличаться. В первом случае — строка. А во втором — число.

С числом можно совершать математические операции, а вот со строкой нет. Например, числа можно складывать при помощи оператора `+`:

```
twenty_five = 25
about_pi = 3
# Переменные одного типа можно складывать,
# а получившееся значение присваивать другой переменной
total = twenty_five + about_pi

# Напечатаем эту переменную:
print(total)
```

Но если те же значения переменных записать как строки, результат изменится:

```
twenty_five = '25'
about_pi = '3'

total = twenty_five + about_pi
```





```
print(total)
```



Обращай внимание на то, какой тип данных тебе нужно получить. Если понадобятся математические операции, не нужно ставить кавычки.

Операции с числами

С числами можно выполнять все основные арифметические операции: сложение

, вычитание , умножение  и деление . Например:

```
print(2 + 3)
# 5
```

То же самое для переменных:

```
first_number = 4
second_number = 2
print(first_number - second_number)
# 2
```

В выражениях можно использовать и переменные, и отдельные числа:

```
number = 5
print(number * 6)
# 30
```

Или записывать в переменную новую значение:

```
age = 25
two_years = 2
age = age + two_years
# Теперь в переменной age хранится число 27
```

Приоритет операций

У умножения и деления более высокий приоритет, чем у сложения и вычитания. Это значит, они выполнятся в первую очередь:

```
first_number = 3
second_number = 4
result = 5 + first_number * second_number # result = 17
```

Сначала выполнилось умножение `first_number * second_number = 12` и только после этого — сложение `5 + 12 = 17`.

Задать приоритет скобками

Приоритет можно задавать скобками: выражение в скобках выполнится в первую очередь. Например, если в предыдущем примере расставить скобки, результат изменится:

```
first_number = 3
second_number = 4
result = (5 + first_number) * second_number # result = 32
```

Сначала выполнится то, что в скобках, — `5 + first_number = 8`, а после — умножение `8 * 4 = 32`.

Чтобы напечатать результат арифметической операции, необязательно сохранять его в переменную. Можно просто написать выражение внутри команды `print()`:

```
first_number = 3
second_number = 4
print(first_number + second_number) # напечатается число 7
```

Операторы `/` и `//`

Для деления можно использовать два оператора — `/` и `//`.

Целочисленное деление. Если использовать `//`, получится дробное число. Даже если делится нацело:

```
print(4 / 2) # напечатается число 2.0
```

Если использовать `//`, дробная часть отсекается.

```
print(4 // 2) # напечатается целое число 2
```



Обычно если делят целые числа, используют `//`, чтобы получилось тоже целое число. А если дробные, используют `/`.

Деление на ноль. При делении на ноль всегда возникает ошибка:

```
print(7 / 0) # ZeroDivisionError: division by zero
```

Оператор `%`

Иногда нужно получить остаток от деления. Это часть числа, которая остаётся после целочисленного деления. Например, число `5` делится целиком на число `3` один раз. В результате получается остаток `2`. Он меньше делителя.

Чтобы найти остаток, используют оператор `%`.

```
print(10 % 10) # будет напечатано 0
```

```
print(10 % 7) # будет напечатано 3
```

Остаток помогает определить кратность чисел и их делимость. Например, твоя программа должна найти нечётные числа или числа кратные 10-ти. Чтобы задать такое условие в коде, тебе нужно будет найти остаток от деления:

```
# если при целочисленном делении на 10 остаток равен 0 — число кратно 10
print(40 % 10) # 10 + 10 + 10 + 10 = 40
print(510 % 10) # 0
```

```
# если при целочисленном делении на 2 остаток равен 0 — число чётное
print(8 % 2) # 0
```

```
print(124 % 2) # 0

# если при целочисленном делении на 2 остаток равен 1 — число нечётное
print(11 % 2) # 1
print(123 % 2) # 1
```

Есть ещё один нюанс. Если число слева от оператора `%` меньше числа справа, остаток будет равен числу слева.

```
print(3 % 7) # будет напечатано 3

print(0 % 10) # 0
```

Сокращённые операции

Очень часто нужно выполнить операцию с переменной и записать в неё же полученный результат. Например, как во втором задании: `numbers = numbers - 12`.

Как ты знаешь, одно из правил Дзен Python гласит: «Простое лучше, чем сложное». Программисты находят особую красоту в простоте и лаконичности. Поэтому для каждой такой операции есть сокращённая запись.

При этом `+=`, `-=`, `*=`, `/=`, `//=` и `%=` считаются операторами. Просто они совершают сразу два действия: арифметическую операцию и приравнивание.

Вот так это выглядит в коде:

```
a = 248
a -= 48

print(a)
# Будет напечатано: 200

f = 45
f //= 5

print(f)
# Будет напечатано: 9
```

В переменную `days_of_the_year` сохранили количество дней в году. Попробуй использовать разные сокращённые операторы и перезаписать её значение.

Например, есть строки «я» и «автоматизатор тестирования». Python может склеить их в одну — «я автоматизатор тестирования».

Конкатенация строк

Это объединение нескольких строк в одну. От латинского concatenatio, «присоединение».

Для конкатенации нужен оператор сложения `+`. Например, `print('я ' + 'автоматизатор тестирования')` выдаст строку «я автоматизатор тестирования».

Пробелы в конкатенации. Обрати внимание на пробелы. Оператор сложения не заменяется на пробел. Поэтому нужно ставить его перед кавычками, иначе слова слипнутся.

Точно так же работает сложение переменных, в которых хранятся строки. Их тоже можно склеить:

```
# В четырёх переменных содержатся строки:
word1 = 'не '
word2 = 'выходи '
word3 = 'из '
word4 = 'комнаты'

# Можно сложить эти переменные, а результат сохранить в переменную text
text = word1 + word2 + word3 + word4

# ...а переменную text можно напечатать
print(text)
```

Складывать можно только значения одинакового типа, иначе Python выдаст ошибку. Например, строку и строку. Или число с числом. Строку с числом — нельзя:

```
year = '365' # Значение этой переменной — строка
day = 1 # Значение этой переменной — число

# Попробуем сложить и напечатать — ошибка:
print(year + day)
```

Умножить строку на целое число

В Python можно строку умножить на целое число. Понадобится оператор умножения — звёздочка `*`.

Умножение строки на число копирует строку несколько раз:

```
la_la_land = 'la'
print(la_la_land * 10)
#получится lalalalalalalalalala
```

При этом нужно учитывать, что со сложением это не работает. Такой код не сработает, Python выдаст ошибку:

```
# Объявили две переменные разных типов
number = 100
rubles = ' рублей'

print(number + rubles)
# В тексте ошибки будет сказано, что оператор '+' не складывает целые числа со строками
```

Сокращённая запись

У конкатенации и умножения строки на целое число есть краткая запись. Здесь используют те же операторы, что и для чисел: `+=` и `*=`.

```
load_test = 'нагрузочное'
load_test += ' тестирование'

print(load_test) # 'нагрузочное тестирование'

lion_roar = 'мяу'
lion_roar *= 3

print(lion_roar) # 'мяумяумяу'
```

Форматирование строк через `f-string`

Ещё один способ объединять несколько строк в одну. Например, есть строка `name` со значением «Билли». Нужно написать «Ты какой-то странный, Билли».

Если использовать `+`, нужно создать ещё одну переменную строку со значением `Ты какой-то странный,`, а потом соединить её с `name`.

Вот как можно сделать иначе — через `f-string`:

```
name = 'Билли'

print(f'Ты какой-то странный, {name}')
```

Как сделать. Нужно:

- написать букву `f`, а после — одинарные кавычки: `print(f'...')`;
- внутри кавычек написать то, что нужно вывести. Имя переменной — в фигурных скобках: `ты какой-то странный, {name}`.

Программа выведет: «Ты какой-то странный, Билли».

Ещё пример. Есть две переменных: `day` и `weather`. Нужно написать «Сегодня будет солнечно».

```
day = 'Сегодня'
weather = 'солнечно'

print(f'{day} будет {weather}')
```

Получится «Сегодня будет солнечно»

Фигурные скобки и экранирование

Python подставляет в строку значение выражения, которое написано внутри фигурных скобок. Например, значение переменной.

Внутри фигурных скобок можно написать и арифметическое выражение:

```
num = 37
three = 3

print(f'{num} умножим на {1 * three}, получим {num * 1 * three}')
```

будет выведено «37 умножим на 3, получим 111»

Если в `f-string` нужно вывести фигурные скобки текстом, окружи их ещё одной парой скобок. Это называется экранировать:


```
print(f'Фигурные скобки: {{{}}}')  
# Фигурные скобки: {}
```

Если значение переменной нужно вывести в кавычках, окружи ими фигурные скобки: `"{name}"`.

```
name = 'Гранд Будапешт'  
  
print(f'Отель "{name}"')  
# Получится: Отель "Гранд Будапешт"
```

Конвертировать типы данных

Python умеет переводить один тип в другой: например, превратить число 3 в строку «3» или наоборот. Это называется **конвертировать тип данных**.

Для этого нужны специальные функции.

Превратить число в строку — `str()`

Например, нужно число 5 превратить в строку «5». Понадобится функция `str()`. Если написать `str(5)`, программа превратит число в строчку 5.

Так же это работает с переменными:

```
number = 5  
friends = ' друзей'  
# преобразуем число в строку и проведём конкатенацию строк  
print(str(number) + friends)
```

Программа напечатает «5 друзей»: это будет одна строка.

Превратить строку в целое число — `int()`

Понадобится функция `int()`. Например, с помощью `int('5')` получится число 5.

Посмотри на пример с переменными:

```
twenty = '20'  
twenty_two = '22'  
# сложили две строки  
print(twenty + twenty_two)
```

Программа напечатает «2022». Чтобы получилось 42, нужно преобразовать строки в числа и сложить их:

```
twenty = '20'  
twenty_two = '22'  
print(int(twenty) + int(twenty_two))
```

Теперь результат другой: 42.

Дробные числа

Для десятичных дробей в Python есть специальный тип данных — `float`, от английского «плавать». Дробные числа называют «числами с плавающей запятой».

Что за 🏊 плавающая 🏊 запятая

Запятая «плавает» по числу, когда его представляют в виде произведения значащей части и степени. Например, число 3,14159 можно записать так:

• $314,159 \cdot 10^{-2}$;
• $0,0314159 \cdot 10^{(2)}$;
• $314159,0 \cdot 10^{(-5)}$.

Нам привычно видеть в таких записях запятую. В англоязычной литературе запятая становится точкой — floating point. Там принято писать десятичные дроби через точку: в коде тоже так.

Как объявить. Этот тип данных объявляется так же, как целое число, — через знак `=`:

```
first_number = 87.2
```

Как выполнять операции. Так же, как с целыми. Например, дробные числа тоже можно складывать:

```
first = 87.2
second = 50.2
third = 50.242
print(first + second + third)

# Будет напечатано: 187.642
```

Преобразования для дробных чисел

Дробные числа преобразуют в строки так же, как и целые — функцией `str()`:

```
first = 87.2
second = 50.2
third = 50.242
print(str(first) + str(second) + str(third))

# Будет напечатано: 87.250.250.242
```

И наоборот: можно преобразовать строку в дробное число. Для этого есть функция `float()`:

```
first = '87.2'    # Строка
second = '50.2'   # Тоже строка
third = '50.242'  # И это строка
print(float(first) + float(second) + float(third)) # А в итоге получится число!

# Будет напечатано: 187.642
```

Как привести дробное число к целому

Дробные числа можно привести к целым. Например, нужно вывести, сколько тебе полных лет. Если 24 года с половиной — вывести `24`.

Вот как это работает:

```
# Функция int() просто убирает всё, что после запятой
a = int(24.5)
print(a)

# Будет напечатано: 24
```



Обрати внимание: `int()` не округляет числа по правилам арифметики, а просто отбрасывает дробную часть.

Вот пример:

```
a = int(2.72)
print(a)

# Будет напечатано: 2
```

Функция `int()` одинаково работает с положительными и отрицательными числами:

```
a = int(-3.14)
print(a)

# Будет напечатано: -3
```

Несколько преобразований сразу

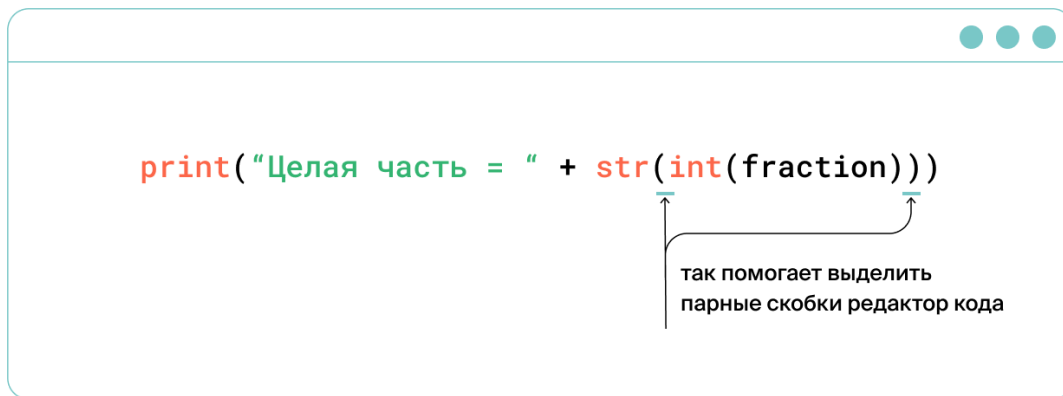
Можно сделать несколько преобразований в одной строке. Например, сначала превратить дробь в целое число, а затем преобразовать в строку:

```
fraction = 1.5 # Дробь
print("Целая часть = " + str(int(fraction)))
# Вернётся строка – целочисленная часть дроби
```

Получится так:

```
Целая часть = 1
```

Важно не сбиться в количестве скобок: закрывающих должно быть столько же, сколько и открывающих. В тренажёре пары скобок подсвечиваются, когда рядом оказывается курсор.



Для операций с целыми и дробными числами приводить их к одному типу не нужно, тут Python разберётся сам:

```
a = 16 * 2.2 + 7 - 0.2
print(a)

# Будет напечатано: 42.0
```