

Подпишитесь на DeepL Pro для редактирования данного документа.  
Дополнительную информацию можно найти на странице [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document).

# 📘 День 3

## Boolean

Булевский тип данных представляет одно из двух значений: \_True\_ или \_False\_. Использование этих типов данных станет понятным, когда мы начнем использовать оператор сравнения. Первая буква \*\*T\*\* для True и \*\*F\*\* для False должна быть заглавной в отличие от JavaScript.

\*\*Пример: Булевы значения\*\*

``py

print(True)

print(False)

```

## Операторы

Язык Python поддерживает несколько типов операторов. В этом разделе мы остановимся на некоторых из них.

### Операторы присваивания

Операторы присваивания используются для присвоения значений переменным. В качестве примера возьмем =. Знак равенства в математике показывает, что два значения равны, однако в Python он означает, что мы храним значение в определенной переменной, и мы называем это присваиванием или присвоением значения переменной. В таблице ниже показаны различные типы операторов присваивания в Python, взятые из [w3school](https://www.w3schools.com/python/python\_operators.asp).

![Операторы присваивания](../images/assignment\_operators.png)

### Арифметические операторы:

- Сложение (+): a + b

- Вычитание (-): a - b

- Умножение (\*): a \* b

- Деление (/): a / b

- Модуль упругости (%): a % b

- Деление на пол(//): a // b

- Экспоненция (\*\*): a \*\* b

![Арифметические операторы](../images/arithmetic\_operators.png)

\*\*Пример:целые числа\*\*

``py

# Арифметические операции в Python

# Целые числа

print('Сложение: ', 1 + 2) # 3

print('Вычитание: ', 2 - 1) # 1

print('Умножение: ', 2 \* 3) # 6

print ('Деление: ', 4 / 2) # 2.0 Деление в Python дает плавающее число

print('Деление: ', 6 / 2) # 3.0

print('Деление: ', 7 / 2) # 3,5

print('Деление без остатка: ', 7 // 2) # 3, дает без плавающего числа или без остатка

print ('Деление без остатка: ',7 // 3) # 2

print('Модуль: ', 3 % 2) # 1, Получает остаток

print('Exponentiation: ', 2 \*\* 3) # 9 это значит 2 \* 2 \* 2

```

\*\*Пример: поплавки\*\*

``py

# Плавающие числа

print('Число с плавающей точкой, PI', 3.14)

print('Число с плавающей точкой, гравитация', 9.81)

```

\*\*Пример:комплексные числа\*\*

``py

# Комплексные числа

print('Комплексное число: ', 1 + 1j)

print('Умножение комплексных чисел: ',(1 + 1j) \* (1 - 1j))

```

Давайте объявим переменную и присвоим ей числовой тип данных. Я собираюсь использовать односимвольную переменную, но помните, что не стоит вырабатывать привычку объявлять такие типы переменных. Имена переменных должны быть все время мнемоническими.

\*\*Пример:\*\*

``python

# Сначала объявляем переменную вверху

a = 3 # a - имя переменной, а 3 - целочисленный тип данных

b = 2 # b - имя переменной, а 3 - целочисленный тип данных

# Арифметические операции и присвоение результата переменной

итого = a + b

дифф = a - b

продукт = a \* b

деление = a / b

остаток = a % b

деление\_пола = a // b

экспоненциальный = a \*\* b

# Я должен был использовать sum вместо total, но sum является встроенной функцией - старайтесь избегать переопределения встроенных функций

print(total) # если вы не пометите печать какой-либо строкой, вы никогда не узнаете, откуда получен результат

print('a + b = ', total)

print('a - b = ', diff)

print('a \* b = ', product)

print('a / b = ', division)

print('a % b = ', остаток)

print('a // b = ', floor\_division)

print('a \*\* b = ', экспоненция)

```

\*\*Пример:\*\*

``py

print('== Сложение, вычитание, умножение, деление, модуль ==')

# Объявление значений и их совместная организация

num\_one = 3

num\_two = 4

# Арифметические операции

итого = num\_one + num\_two

diff = num\_two - num\_one

продукт = число\_один \* число\_два

div = num\_two / num\_one

остаток = число\_два % число\_один

# Печать значений с меткой

print('total: ', total)

print('разница: ', diff)

print('product: ', product)

print('division: ', div)

print('остаток: ', остаток)

```

Давайте начнем соединять точки и использовать то, что мы уже знаем для вычислений (площадь, объем, плотность, вес, периметр, расстояние, сила).

\*\*Пример:\*\*

``py

# Вычисление площади круга

радиус = 10 # радиус круга

площадь\_окружности = 3,14 \* радиус \*\* 2 # два \* знак означает экспоненту или мощность

print('Площадь круга:', area\_of\_circle)

# Вычисление площади прямоугольника

длина = 10

ширина = 20

площадь\_прямоугольника = длина \* ширина

print('Площадь прямоугольника:', area\_of\_rectangle)

# Вычисление веса объекта

масса = 75

гравитация = 9,81

вес = масса \* гравитация

print(weight, 'N') # Добавление единицы к весу

# Вычислите плотность жидкости

масса = 75 # в кг

объем = 0,075 # в кубическом метре

плотность = масса / объем # 1000 кг/м^3

```

### Операторы сравнения

В программировании мы сравниваем значения, мы используем операторы сравнения для сравнения двух значений. Мы проверяем, больше, меньше или равно ли значение другому значению. В следующей таблице приведены операторы сравнения Python, взятые из [w3shool](https://www.w3schools.com/python/python\_operators.asp).

![Операторы сравнения](../images/comparison\_operators.png)

\*\*Пример: Операторы сравнения\*\*

``py

print(3 > 2) # Истина, потому что 3 больше 2

print(3 >= 2) # Истина, потому что 3 больше 2

print(3 < 2) # Ложь, потому что 3 больше 2

print(2 < 3) # Истина, потому что 2 меньше 3

print(2 <= 3) # Истина, потому что 2 меньше 3

print(3 == 2) # Ложь, потому что 3 не равно 2

print(3 != 2) # Истина, потому что 3 не равно 2

print(len('mango') == len('avocado')) # Ложь

print(len('mango') != len('avocado')) # True

print(len('mango') < len('avocado')) # True

print(len('milk') != len('meat')) # Ложь

print(len('milk') == len('meat')) # True

print(len('tomato') == len('potato')) # True

print(len('python') > len('dragon')) # Ложь

# Сравнение чего-либо дает либо истинное, либо ложное значение

print('True == True: ', True == True)

print('True == False: ', True == False)

print('False == False:', False == False)

```

В дополнение к вышеуказанному оператору сравнения Python использует:

- \_is\_: Возвращает true, если обе переменные являются одним и тем же объектом(x is y)

- \_is not\_: Возвращает true, если обе переменные не являются одним и тем же объектом(x не y)

- \_in\_: Возвращает True, если запрашиваемый список содержит определенный элемент(x в y)

- \_нет в\_: Возвращает True, если в запрашиваемом списке нет определенного элемента(x in y).

``py

print('1 есть 1', 1 есть 1) # Истина - потому что значения данных одинаковы

print('1 не 2', 1 не 2) # True - потому что 1 не 2

print('A в Asabeneh', 'A' в 'Asabeneh') # True - A найдена в строке

print('B в Asabeneh', 'B' в 'Asabeneh') # Ложь - нет заглавной буквы B

print('coding' in 'coding for all') # True - потому что в coding for all есть слово coding

print('a in an:', 'a' in 'an') # True

print('4 равно 2 \*\* 2:', 4 равно 2 \*\* 2) # True

```

### Логические операторы

В отличие от других языков программирования, python использует ключевые слова \_and\_, \_or\_ и \_not\_ для логических операторов. Логические операторы используются для объединения условных операторов:

![Логические операторы](../images/logical\_operators.png)

``py

print(3 > 2 и 4 > 3) # True - потому что оба утверждения истинны

print(3 > 2 и 4 < 3) # Ложь - потому что второе утверждение ложно

print(3 < 2 и 4 < 3) # Ложь - потому что оба утверждения ложны

print('True и True: ', True и True)

print(3 > 2 или 4 > 3) # True - потому что оба утверждения истинны

print(3 > 2 или 4 < 3) # True - потому что одно из утверждений истинно

print(3 < 2 или 4 < 3) # Ложь - потому что оба утверждения ложны

print('True or False:', True or False)

print(not 3 > 2) # False - поскольку 3 > 2 истинно, то not True дает False

print(not True) # Ложь - отрицание, оператор not превращает истину в ложь

print(not False) # True

print(not not True) # True

print(not not False) # False

```

🌕 У вас безграничная энергия. Вы только что завершили третий день испытаний и продвинулись на три шага вперед на пути к величию. Теперь сделайте несколько упражнений для мозга и мышц.

## 💻 Упражнения - день 3

1. Объявите свой возраст целочисленной переменной

2. Объявите свой рост как переменную float

3. Объявите переменную, хранящую комплексное число

4. Напишите сценарий, который предлагает пользователю ввести основание и высоту треугольника и вычислить площадь этого треугольника (площадь = 0,5 x b x h).

``py

Введите базу: 20

Введите высоту: 10

Площадь треугольника равна 100

```

5. Напишите сценарий, который предлагает пользователю ввести сторону a, сторону b и сторону c треугольника. Вычислите периметр треугольника (периметр = a + b + c).

``py

Введите сторону a: 5

Введите сторону b: 4

Введите сторону c: 3

Периметр треугольника равен 12

```

6. Получите длину и ширину прямоугольника с помощью подсказки. Вычислите его площадь (площадь = длина x ширина) и периметр (периметр = 2 x (длина + ширина)).

7. Получите радиус окружности с помощью подсказки. Вычислите площадь (площадь = pi x r x r) и окружность (c = 2 x pi x r), где pi = 3,14.

8. Вычислите наклон, х-пересечение и у-пересечение y = 2x -2

9. Наклон равен (m = y2-y1/x2-x1). Найдите наклон и [евклидово расстояние](https://en.wikipedia.org/wiki/Euclidean\_distance#:~:text=В%20математике%2C%20the%20Euclidean%20distance, being%20called%20the%20Pythagorean%20distance.) между точкой (2, 2) и точкой (6,10).

10. Сравните наклоны в задачах 8 и 9.

11. Вычислите значение y (y = x^2 + 6x + 9). Попробуйте использовать различные значения x и выясните, при каком значении x значение y будет равно 0.

12. Найдите длину слов 'python' и 'dragon' и сделайте ложный оператор сравнения.

13. Используйте оператор \_and\_, чтобы проверить, встречается ли 'on' и в 'python', и в 'dragon'.

14. \_Я надеюсь, что этот курс не переполнен жаргоном\_. Используйте оператор \_in\_, чтобы проверить, есть ли \_жаргон\_ в предложении.

15. Как в dragon, так и в python нет "on".

16. Найдите длину текста \_python\_ и преобразуйте значение в float и преобразуйте его в строку

17. Четные числа делятся на 2, а остаток равен нулю. Как проверить, является ли число четным или нет, используя python?

18. Проверьте, равно ли полу деление 7 на 3 преобразованному в int значению 2,7.

19. Проверьте, равен ли тип '10' типу 10

20. Проверьте, равно ли int('9.8') 10

21. Напишите сценарий, который предлагает пользователю ввести часы и ставку за час. Рассчитать зарплату человека?

``py

Введите часы: 40

Введите ставку за час: 28

Ваш недельный заработок составляет 1120

```

22. Напишите сценарий, который предлагает пользователю ввести количество лет. Рассчитайте, сколько секунд может прожить человек. Предположим, что человек может прожить сто лет

``py

Укажите количество прожитых лет: 100

Вы прожили 3153600000 секунд.

```

23. Напишите сценарий на языке Python, который выводит на экран следующую таблицу

``py

1 1 1 1 1

2 1 2 4 8

3 1 3 9 27

4 1 4 16 64

5 1 5 25 125

```

# День 4

## Строки

Текст - это строковый тип данных. Любой тип данных, записанный как текст, является строкой. Любые данные, заключенные в одинарные, двойные или тройные кавычки, являются строками. Для работы со строковыми типами данных существуют различные строковые методы и встроенные функции. Для проверки длины строки используйте метод len().

### Создание строки

``py

letter = 'P' # Строка может быть одним символом или целым текстом.

print(letter) # P

print(len(letter)) # 1

greeting = 'Hello, World!' # Строка может быть сделана с использованием одинарной или двойной кавычки, "Hello, World!".

print(greeting) # Hello, World!

print(len(greeting)) # 13

предложение = "Надеюсь, вам нравятся 30 дней Python Challenge".

print(sentence)

```

Многострочная строка создается с помощью тройных одинарных (''') или тройных двойных кавычек ("""). См. пример ниже.

``py

multiline\_string = '''Я учитель и мне нравится преподавать.

Я не нашел ничего более полезного, чем расширение прав и возможностей людей.

Вот почему я создал 30 дней python.'''

print(multiline\_string)

# Другой способ сделать то же самое

multiline\_string = """Я учитель и мне нравится преподавать.

Я не нашел ничего более полезного, чем расширение прав и возможностей людей.

Именно поэтому я создал программу "30 дней python"."""

print(multiline\_string)

```

### Конкатенация строк

Мы можем соединять строки вместе. Слияние или соединение строк называется конкатенацией. Смотрите пример ниже:

``py

first\_name = 'Asabeneh'

last\_name = 'Yetayeh'

пробел = ' '

полное\_имя = имя\_имя + пробел + последнее\_имя

print(full\_name) # Asabeneh Yetayeh

# Проверка длины строки с помощью встроенной функции len()

print(len(first\_name)) # 8

print(len(last\_name)) # 7

print(len(first\_name) > len(last\_name)) # True

print(len(full\_name)) # 16

```

### Последовательности экранирования в строках

В Python и других языках программирования \, за которым следует символ, является управляющей последовательностью. Рассмотрим наиболее распространенные символы экранирования:

- \n: новая линия

- \t: Средства табуляции (8 пробелов)

- \\\\: Обратная косая черта

- \\': Одинарная кавычка (')

- \\": Двойная кавычка ("")

Теперь давайте посмотрим на примерах, как используются вышеуказанные управляющие последовательности.

``py

print('Надеюсь, всем нравится Python Challenge.\nА вам?') # перевод строки

print('Days\tTopics\tExercises') # добавление пробела табуляции или 4 пробелов

print('День 1\t3\t5')

print('День 2\t3\t5')

print('День 3\t3\t5')

print('День 4\t3\t5')

print('Это символ обратной косой черты (\\)') # Чтобы написать обратную косую черту

print('В каждом языке программирования он начинается с \"Hello, World!\"') # чтобы написать двойную кавычку внутри одинарной кавычки

# выход

Я надеюсь, что всем нравится Python Challenge.

Вы?

ДниТемыУпражнения

День 155

День 2620

День 3523

День 4135

Это символ обратной косой черты (\)

В каждом языке программирования он начинается со слов "Hello, World!".

```

### Форматирование строк

#### Форматирование строк в старом стиле (оператор %)

В Python существует множество способов форматирования строк. В этом разделе мы рассмотрим некоторые из них.

Оператор "%" используется для форматирования набора переменных, заключенных в "кортеж" (список фиксированного размера), вместе со строкой формата, которая содержит обычный текст вместе со "спецификаторами аргументов", специальными символами типа "%s", "%d", "%f", "%.<small>число цифр</small>f".

- %s - Строка (или любой объект со строковым представлением, например, числа)

- %d - Целые числа

- %f - числа с плавающей запятой

- "%.<small>число цифр</small>f" - числа с плавающей запятой с фиксированной точностью

``py

# Только струны

first\_name = 'Asabeneh'

last\_name = 'Yetayeh'

язык = 'Python'

formated\_string = 'Я %s %s. Я преподаю %s' %(имя\_фамилия, фамилия\_фамилия, язык)

print(formated\_string)

# Строки и числа

радиус = 10

pi = 3.14

площадь = пи \* радиус \*\* 2

formated\_string = 'Площадь круга с радиусом %d равна %.2f.' %(радиус, площадь) # 2 относится к двум значащим цифрам после точки

python\_libraries = ['Django', 'Flask', 'NumPy', 'Matplotlib', 'Pandas'].

formated\_string = 'Следующие библиотеки python:%s' % (python\_libraries)

print(formated\_string) # "Следующие библиотеки python:['Django', 'Flask', 'NumPy', 'Matplotlib', 'Pandas']".

```

#### Новый стиль форматирования строк (str.format)

Это форматирование появилось в Python версии 3.

``py

first\_name = 'Asabeneh'

last\_name = 'Yetayeh'

язык = 'Python'

formated\_string = 'Я {} {}. Я преподаю {}'.format(имя\_фамилия, фамилия\_фамилия, язык)

print(formated\_string)

a = 4

b = 3

print('{} + {} = {}'.format(a, b, a + b))

print('{} - {} = {}'.format(a, b, a - b))

print('{} \* {} = {}'.format(a, b, a \* b))

print('{} / {} = {:.2f}'.format(a, b, a / b)) # ограничивается двумя цифрами после десятичной дроби

print('{} % {} = {}'.format(a, b, a % b))

print('{} // {} = {}'.format(a, b, a // b))

print('{} \*\* {} = {}'.format(a, b, a \*\* b))

# выход

4 + 3 = 7

4 - 3 = 1

4 \* 3 = 12

4 / 3 = 1.33

4 % 3 = 1

4 // 3 = 1

4 \*\* 3 = 64

# Строки и числа

радиус = 10

pi = 3.14

площадь = пи \* радиус \*\* 2

formated\_string = 'Площадь круга с радиусом {} равна {:.2f}.'.format(radius, area) # 2 цифры после десятичной дроби

print(formated\_string)

```

#### Интерполяция строк / f-строки (Python 3.6+)

Еще одно новое форматирование строк - интерполяция строк, f-строки. Строки начинаются с f, и мы можем вводить данные в соответствующие позиции.

``py

a = 4

b = 3

print(f'{a} + {b} = {a +b}')

print(f'{a} - {b} = {a - b}')

print(f'{a} \* {b} = {a \* b}')

print(f'{a} / {b} = {a / b:.2f}')

print(f'{a} % {b} = {a % b}')

print(f'{a} // {b} = {a // b}')

print(f'{a} \*\* {b} = {a \*\* b}')

```

### Строки Python как последовательности символов

Строки Python представляют собой последовательности символов и имеют общие базовые методы доступа с другими упорядоченными последовательностями объектов Python - списками и кортежами. Самый простой способ извлечения отдельных символов из строк (и отдельных членов любой последовательности) - распаковать их в соответствующие переменные.

#### Распаковка символов

```

язык = 'Python'

a,b,c,d,e,f = language # распаковка символов последовательности в переменные

print(a) # P

print(b) # y

print(c) # t

print(d) # h

print(e) # o

print(f) # n

```

#### Доступ к символам в строках по индексу

В программировании счет начинается с нуля. Поэтому первая буква строки имеет нулевой индекс, а последняя буква строки равна длине строки минус один.

![String index](../images/string\_index.png)

``py

язык = 'Python'

первая\_буква = язык[0]

print(first\_letter) # P

вторая\_буква = язык[1]

print(second\_letter) # y

last\_index = len(language) - 1

last\_letter = language[last\_index]

print(last\_letter) # n

```

Если мы хотим начать с правого конца, мы можем использовать отрицательную индексацию. -1 - это последний индекс.

``py

язык = 'Python'

последняя\_буква = язык[-1]

print(last\_letter) # n

second\_last = language[-2]

print(second\_last) # o

```

#### Нарезка строк Python

В python мы можем разрезать строки на подстроки.

``py

язык = 'Python'

first\_three = language[0:3] # начинается с нулевого индекса и до 3, но не включает 3

print(first\_three) #Pyt

last\_three = language[3:6]

print(last\_three) # hon

# Другой способ

last\_three = language[-3:]

print(last\_three) # hon

last\_three = language[3:]

print(last\_three) # hon

```

#### Обратный ход строки

В python мы можем легко развернуть строки.

``py

greeting = 'Hello, World!'

print(greeting[::-1]) # !dlroW ,olleH

```

#### Пропуск символов при нарезке

Можно пропускать символы при нарезке, передавая аргумент step методу slice.

``py

язык = 'Python'

pto = language[0:6:2] #

print(pto) # Pto

```

### Строковые методы

Существует множество строковых методов, которые позволяют нам форматировать строки. Некоторые из строковых методов показаны в следующем примере:

- capitalize(): Преобразует первый символ строки в заглавную букву

``py

challenge = 'тридцать дней python'

print(challenge.capitalize()) # 'Тридцать дней питона'

```

- count(): возвращает вхождения подстроки в строку, count(substring, start=..., end=...). Начало - это начальный индекс для подсчета, а конец - последний индекс для подсчета.

``py

challenge = 'тридцать дней python'

print(challenge.count('y')) # 3

print(challenge.count('y', 7, 14)) # 1,

print(challenge.count('th')) # 2`

```

- endswith(): Проверяет, заканчивается ли строка указанным окончанием

``py

challenge = 'тридцать дней python'

print(challenge.endswith('on')) # True

print(challenge.endswith('tion')) # Ложь

```

- expandtabs(): Заменяет символы табуляции пробелами, размер табуляции по умолчанию равен 8. Принимает аргумент размера табуляции

``py

challenge = 'thirty\tdays\tof\tpython'

print(challenge.expandtabs()) # "тридцать дней питона

print(challenge.expandtabs(10)) # "тридцать дней питона

```

- find(): Возвращает индекс первого вхождения подстроки, если не найден, возвращает -1

``py

challenge = 'тридцать дней python'

print(challenge.find('y')) # 16

print(challenge.find('th')) # 17

```

- rfind(): Возвращает индекс последнего вхождения подстроки, если не найден, возвращает -1

``py

challenge = 'тридцать дней python'

print(challenge.rfind('y')) # 5

print(challenge.rfind('th')) # 1

```

- format(): форматирует строку в более красивый вывод

Подробнее о форматировании строк смотрите здесь [ссылка](https://www.programiz.com/python-programming/methods/string/format)

``py

first\_name = 'Asabeneh'

last\_name = 'Yetayeh'

возраст = 250

должность = 'учитель'

страна = 'Финляндия'

предложение = 'Я {} {}. Я - {}. Мне {} лет. Я живу в {}.'.format(имя\_фамилия, фамилия\_фамилия, возраст, работа, страна)

print(sentence) # Меня зовут Асабенех Йетайех. Мне 250 лет. Я учительница. Я живу в Финляндии.

радиус = 10

pi = 3.14

площадь = пи \* радиус \*\* 2

result = 'Площадь круга радиусом {} равна {}'.format(str(radius), str(area))

print(result) # Площадь круга радиусом 10 равна 314

```

- index(): Возвращает наименьший индекс подстроки, дополнительные аргументы указывают начальный и конечный индекс (по умолчанию 0 и длина строки - 1). Если подстрока не найдена, возникает ошибка valueError.

``py

challenge = 'тридцать дней python'

sub\_string = 'da'

print(challenge.index(sub\_string)) # 7

print(challenge.index(sub\_string, 9)) # ошибка

```

- rindex(): Возвращает наибольший индекс подстроки, дополнительные аргументы указывают начальный и конечный индекс (по умолчанию 0 и длина строки - 1)

``py

challenge = 'тридцать дней python'

sub\_string = 'da'

print(challenge.rindex(sub\_string)) # 8

print(challenge.rindex(sub\_string, 9)) # ошибка

```

- isalnum(): Проверяет буквенно-цифровой символ

``py

challenge = 'ThirtyDaysPython'

print(challenge.isalnum()) # True

challenge = '30DaysPython'

print(challenge.isalnum()) # True

challenge = 'тридцать дней python'

print(challenge.isalnum()) # Ложь, пробел не является буквенно-цифровым символом

challenge = 'тридцать дней python 2019'

print(challenge.isalnum()) # Ложь

```

- isalpha(): Проверяет, все ли элементы строки являются символами алфавита (a-z и A-Z).

``py

challenge = 'тридцать дней python'

print(challenge.isalpha()) # Ложь, пространство снова исключено

challenge = 'ThirtyDaysPython'

print(challenge.isalpha()) # True

num = '123'

print(num.isalpha()) # Ложь

```

- isdecimal(): Проверяет, все ли символы в строке являются десятичными (0-9).

``py

challenge = 'тридцать дней python'

print(challenge.isdecimal()) # Ложь

вызов = '123'

print(challenge.isdecimal()) # True

challenge = '\u00B2'

print(challenge.isdigit()) # Ложь

вызов = '12 3'

print(challenge.isdecimal()) # Ложь, пробел не разрешен

```

- isdigit(): Проверяет, все ли символы в строке являются цифрами (0-9 и некоторые другие символы юникода для цифр).

``py

вызов = 'Тридцать'

print(challenge.isdigit()) # Ложь

вызов = '30'

print(challenge.isdigit()) # True

challenge = '\u00B2'

print(challenge.isdigit()) # True

```

- isnumeric(): Проверяет, все ли символы в строке являются числами или связаны с числами (как и isdigit(), только принимает больше символов, например ½).

``py

num = '10'

print(num.isnumeric()) # True

num = '\u00BD' # ½

print(num.isnumeric()) # True

num = '10.5'

print(num.isnumeric()) # False

```

- isidentifier(): Проверяет наличие действительного идентификатора - проверяет, является ли строка действительным именем переменной.

``py

challenge = '30DaysOfPython'

print(challenge.isidentifier()) # Ложь, потому что он начинается с числа

challenge = 'thirty\_days\_of\_python'

print(challenge.isidentifier()) # True

```

- islower(): Проверяет, все ли символы алфавита в строке являются строчными.

``py

challenge = 'тридцать дней python'

print(challenge.islower()) # True

challenge = 'Тридцать дней python'

print(challenge.islower()) # Ложь

```

- isupper(): Проверяет, все ли символы алфавита в строке являются прописными.

``py

challenge = 'тридцать дней python'

print(challenge.isupper()) # False

challenge = 'THIRTY DAYS OF PYTHON'

print(challenge.isupper()) # True

```

- join(): Возвращает конкатенированную строку

``py

web\_tech = ['HTML', 'CSS', 'JavaScript', 'React'].

result = ' '.join(web\_tech)

print(result) # 'HTML CSS JavaScript React'

```

``py

web\_tech = ['HTML', 'CSS', 'JavaScript', 'React'].

result = '# '.join(web\_tech)

print(result) # 'HTML# CSS# JavaScript# React'

```

- strip(): Удаляет все заданные символы, начиная с начала и конца строки

``py

challenge = 'тридцать дней питоннн'

print(challenge.strip('noth')) # 'irty days of py'

```

- replace(): Заменяет подстроку заданной строкой

``py

challenge = 'тридцать дней python'

print(challenge.replace('python', 'coding')) # 'тридцать дней кодирования'

```

- split(): Разделяет строку, используя заданную строку или пробел в качестве разделителя

``py

challenge = 'тридцать дней python'

print(challenge.split()) # ['thirty', 'days', 'of', 'python'].

challenge = 'тридцать, дней, питона'

print(challenge.split(', ')) # ['thirty', 'days', 'of', 'python']

```

- title(): Возвращает строку с заголовком

``py

challenge = 'тридцать дней python'

print(challenge.title()) # Thirty Days Of Python

```

- swapcase(): Преобразует все прописные символы в строчные и все строчные символы в прописные

``py

challenge = 'тридцать дней python'

print(challenge.swapcase()) # ТРИДЦАТЬ ДНЕЙ ПИТОНА

challenge = 'Thirty Days Of Python'

print(challenge.swapcase()) # ДЕВЯТЬ ДНЕЙ ПИТОНА

```

- startswith(): Проверяет, начинается ли строка с указанной строки

``py

challenge = 'тридцать дней python'

print(challenge.startswith('thirty')) # Правда

challenge = '30 дней python'

print(challenge.startswith('thirty')) # Ложь

```

🌕 Вы необыкновенный человек и у вас замечательный потенциал. Вы только что завершили 4-й день испытаний, и вы на четыре шага впереди на пути к величию. Теперь сделайте несколько упражнений для вашего мозга и мышц.

## 💻 Упражнения - день 4

1. Конкатенируйте строки 'Thirty', 'Days', 'Of', 'Python' в одну строку 'Thirty Days Of Python'.

2. Объедините строки 'Coding', 'For' , 'All' в одну строку 'Coding For All'.

3. Объявите переменную с именем company и присвойте ей начальное значение "Coding For All".

4. Выведите переменную company с помощью \_print()\_.

5. Выведите длину строки компании, используя метод \_len()\_ и \_print()\_.

6. Измените все символы на заглавные буквы, используя метод \_upper()\_.

7. Измените все символы на строчные буквы с помощью метода \_lower()\_.

8. Используйте методы capitalize(), title(), swapcase() для форматирования значения строки \_Coding For All\_.

9. Вырежьте (нарежьте) первое слово строки \_Coding For All\_.

10. Проверьте, содержит ли строка \_Coding For All\_ слово Coding, используя метод index, find или другие методы.

11. Замените слово coding в строке 'Coding For All' на Python.

12. Замените Python для всех на Python для всех с помощью метода замены или других методов.

13. Разделите строку 'Coding For All', используя пробел в качестве разделителя (split()) .

14. "Facebook, Google, Microsoft, Apple, IBM, Oracle, Amazon" разделяет строку на запятые.

15. Какой символ находится под индексом 0 в строке \_Coding For All\_.

16. Каков последний индекс строки \_Coding For All\_.

17. Какой символ находится под индексом 10 в строке "Coding For All".

18. Придумайте акроним или аббревиатуру для названия "Python для всех".

19. Создайте акроним или аббревиатуру для названия 'Coding For All'.

20. Используйте индекс для определения позиции первого вхождения C в Coding For All.

21. Используйте индекс для определения позиции первого вхождения F в Coding For All.

22. Используйте rfind для определения позиции последнего вхождения l в Coding For All People.

23. С помощью index или find найдите позицию первого вхождения слова 'because' в следующем предложении: "Вы не можете закончить предложение словом because, потому что because - это связка".

24. Используйте rindex, чтобы найти позицию последнего вхождения слова because в следующем предложении: 'You cannot end a sentence with because because because because is a conjunction'.

25. Выделите фразу "потому что, потому что" в следующем предложении: "Вы не можете закончить предложение фразой "потому что, потому что, потому что".

26. Найдите позицию первого появления слова "потому что" в следующем предложении: "Вы не можете закончить предложение словами "потому что", потому что "потому что" - это связка".

27. Выделите фразу "потому что, потому что" в следующем предложении: "Вы не можете закончить предложение фразой "потому что, потому что, потому что".

28. Начинается ли '\'Coding For All' с подстроки \_Coding\_?

29. Заканчивается ли 'Coding For All' подстрокой \_coding\_?

30. '&nbsp;&nbsp; Кодирование для всех &nbsp;&nbsp;&nbsp; &nbsp; &nbsp;' &nbsp;, удаляет левый и правый пробелы в заданной строке.

31. Какая из следующих переменных возвращает True при использовании метода isidentifier():

- 30DaysOfPython

- тридцать\_дней\_из\_питона

32. В следующем списке приведены названия некоторых библиотек python: ['Django', 'Flask', 'Bottle', 'Pyramid', 'Falcon']. Соедините список хэшем с пробельной строкой.

33. Используйте новую последовательность перевода строки для разделения следующих предложений.

``py

Мне нравится этот вызов.

Мне просто интересно, что будет дальше.

```

34. С помощью последовательности символов табуляции напишите следующие строки.

``py

Имя Возраст Страна Город

Асабенех 250 Финляндия Хельсинки

```

35. Используйте метод форматирования строки, чтобы отобразить следующее:

``sh

радиус = 10

площадь = 3,14 \* радиус \*\* 2

Площадь круга радиусом 10 равна 314 метрам квадратным.

```

36. Сделайте следующее, используя методы форматирования строк:

``sh

8 + 6 = 14

8 - 6 = 2

8 \* 6 = 48

8 / 6 = 1.33

8 % 6 = 2

8 // 6 = 1

8 \*\* 6 = 262144

```