GeekBrains

Лекция 2. Простые типы данных

Погружение в Python

Оглавление

На этой лекции мы

1. Познакомимся со строгой динамической типизацией языка Python.
2. Изучим понятие объекта в Python. Разберёмся с атрибутами и методами объектов.
3. Рассмотрим способы аннотации типов.
4. Изучим "простые" типы данных, такие как числа и строки.
5. Узнаем про математические модули Python.

Краткая выжимка, о чём говорилось в предыдущей лекции

1. На прошлой лекции мы разобрали установку и настройку Python. Были изучены правила создания виртуального окружения и работу с pip.
2. Мы повторили основы синтаксиса языка Python. Познакомились с рекомендациями по оформлению кода.
3. Изучили способы создания ветвящихся алгоритмов на Python. Разобрались с разными вариантами реализации циклов.

Термины лекции

- **Коллекция в программировании** — программный объект, содержащий в себе, тем или иным образом, набор значений одного или различных типов, и позволяющий обращаться к этим значениям. Коллекция позволяет записывать в себя значения и извлекать их. Назначение коллекции - служить хранилищем объектов и обеспечивать доступ к ним.
- **Простой тип** — в информатике тип данных, о объектах которого, переменных или постоянных, можно сказать следующее: работа с объектами осуществляется с помощью конструкций языка; внутреннее представление значений объектов может зависеть от реализации транслятора (компилятора или интерпретатора) и от платформы; объекты не включают в себя другие объекты и служат основой для построения других объектов.
- **Хеш** — это криптографическая функция хеширования, которую обычно называют просто хэшем.
- **Хеш-функция** представляет собой математический алгоритм, который может преобразовать произвольный массив данных в строку фиксированной длины, состоящую из цифр и букв.
- **Атрибуты** — это переменные, конкретные характеристики объекта, такие как цвет поля или имя пользователя.
- **Методы** — это функции, которые описаны внутри объекта или класса. Они относятся к определенному объекту и позволяют взаимодействовать с ними или другими частями кода.
- **Дандер (англ. Dunder) или магические методы в Python** — это методы, имеющие два префиксных и суффиксных подчеркивания в имени метода. Дандер здесь означает «Двойное Подчеркивание».
- **ASCII (англ. American standard code for information interchange)** — название таблицы (кодировки, набора), в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.
- **Комплексные числа (от лат. complexus — связь, сочетание)** — числа вида a+bi, где a,b — вещественные числа, i — мнимая единица, то есть число, для которого выполняется равенство: i2 = -1.

Подробный текст лекции

1. Простые типы данных и коллекции

В Python существует достаточно большое количество типов данных. И речь идёт не о множестве вариантов целого числа на 1, 2, 4 и 8 байт со знаком и без знака. Целый тип как раз представлен в единственном экземпляре. Условно все типы данных можно разделить на простые типы и коллекции. А вот примитивных типов, вроде того же целого беззнакового на 2 байта и т.п. в Python нет.

Коллекции объединяют в себе другие типы данных. Самый популярные пример коллекции в информатике — массив.
В противоположность коллекциям простые типы не объединяют объекты в группы, а используются сами по себе. О них пойдет речь не этой лекции. Там, где это необходимо, будем вспоминать коллекции. Но подробнее разберём их на следующей лекции.

Строгая динамическая типизация Python

Повторим один момент из первой лекции. Python является языком со строгой динамической типизацией. Это означает что тип объекта изменить невозможно, он строго задаётся при создании объекта. При этом переменные могут ссылаться на объекты разных типов. И это не вызывает ошибки.

```
a = 5
a = "hello world"
a = 42.0 * 3.141592 / 2.71828
```

Функция type()

Чтобы разобраться в динамической типизации, воспользуемся функцией type(). Она возвращает класс объекта, его тип.

```
a = 5
print(type(a))
a = "hello world"
print(type(a))
a = 42.0 * 3.141592 / 2.71828
print(type(a))
```

Как видите у нас меняется класс объекта, на который ссылается переменная. Она динамически изменяется.
Слово class при выводе типа сообщает, что мы обратились к объекту, экземпляру класса. Это не просто пара байт информации, а сложная структура, обеспечивающая работу кода в целом и динамическую типизацию в частности.

Строгая типизация

Вспомним функцию id() из прошлой лекции.

```
a = 5
```

```
print(type(a), id(a))
a = "hello world"
print(type(a), id(a))
a = 42.0 * 3.141592 / 2.71828
print(type(a), id(a)
```

Каждый раз возвращается новый адрес объекта в памяти. Python не меняет класс объекта. Он меняет переменную. Каждый раз она ссылается на новый объект. Такая механика прописана в самом интерпретаторе.

Функция isinstance()

Для проверки типа объекта используется функция isinstance().

```
isinstance(object, classinfo)
```

Функция принимает на вход объект и класс и возвращает истину, если объект является экземпляром прямого или косвенного подкласса.

```
data = 42
print(isinstance(data, int))
```

Получили True, т.к. 42 — целое число.

```
data = True
print(isinstance(data, int))
```

Снова истина, т.к. логический объект True в Python подклассом, основанном на классе int. Проверка может проходить сразу по нескольким классам. В этом случае истину вернётся, если объект является экземпляром любого из переданных в кортеже классов.

```
data = 3.14
print(isinstance(data, (int, float, complex)))
```

Внимание! Обратите внимание, что функция принимает 2 аргумента. Второй аргумент — кортеж, внутри которого через запятую перечислены 3 класса.

🔥 **Важно!** Для проверки типа можно вызывать функцию type() и сравнить значение с результатом. Но документация к языку не рекомендует так делать. Для получения информации о типе объекта правильно использовать функцию isinstance().

Оператор is

Ещё один способ взаимодействия с типами данных — оператор is. Он похож на сравнение (двойное равно), но сравнивает не значения, а сами объекты.

```
num = 2 + 2 * 2
digit = 36 / 6
print(num == digit)
print(num is digit)
```

По сути оператор is занимается сравнением "айдишников" объектов, т.е. проверяет лежит ли объекты в одном месте в памяти компьютера.

Заменим в примере обычное деление на целочисленное. Python вычислил оба значения на этапе предкомпиляции, понял что это "целое шесть" и создал один объект вместо двух. Теперь оператор is возвращает истину, т.к. num и digit указывают на один и тот же объект в памяти.

Изменяемые и неизменяемые типы и их особенности

Класс объекта зафиксирован. Целое всегда будет целым, строка строкой, а кортеж кортежем. Однако объекты в Python делятся на изменяемые (mutable) и неизменяемые (immutable). При попытке изменить неизменяемый объект возможны два варианта:

- Замена на новый объект того же типа

```
a = 5
print(a, id(a))
a += 1
print(a, id(a))
```

Переменная a ссылается на число 5, которое лежит в определённом месте в ОЗУ. При увеличении значения на единицу был создан новый объект в памяти. Переменная a теперь указывает на него.

- Второй вариант развития событий при изменении неизменяемого — вызов ошибки. Строка неизменяема. Попытаемся заменить пробел подчеркиванием.

```
txt = 'Hello world!'
txt[5] = '_'
```

Получим `TypeError: 'str' object does not support item assignment` т.к. изменить символ в неизменяемой строке нельзя. Но ведь в Python есть возможность такой замены.

```
txt = 'Hello world!'
print(txt, id(txt))
txt = txt.replace(' ', '_')
print(txt, id(txt))
```

Присмотритесь к адресам. Мы создали новый объект строкового типа, который занимает новое место в памяти. Изменить старый не удалось.
К строкам мы вернёмся позже на этой и следующей лекциях.

Запоминаем неизменяемые типы данных

Как понять изменяемый перед нами объект или нет. Вариант 1 — запомнить таблицу.

Неизменяемы	**Изменяемые**
None	
Числа: int, bool, float, complex	
Последовательности: str, tuple, bytes	Последовательности: list, bytearray
Множества: set	Множества: frozenset
	Отображения: dict

Как вы можете заметить все числа в Python относятся к неизменяемым типам данных. А значит любые математические операции создают новый объект в памяти. Пример ниже служит подтверждением.

```
a = c = 2
b = 3
print(a, id(a), b, id(b), c, id(c))
a = b + c
print(a, id(a), b, id(b), c, id(c))
```

Обратите внимание на первую строку кода. Подобное присваивание допустимо в Python. Переменные a и c будут указывать на один и тот же объект.
После сложения переменная a указывает на новый объект. Переменная c по прежнему указывает на число 2, т.к. int — неизменяемый тип данных.

Хэш hash() как проверка на неизменяемость

Второй вариант убедиться в изменяемости или неизменяемости — проверка на хеширование.

Хеш — это криптографическая функция хеширования, которую обычно называют просто хэшем. **Хеш-функция** представляет собой алгоритм, который может преобразовать произвольный массив данных в набор бит фиксированной длины.

В Python для получения хеша используется функция hash(). Если объект является неизменяемым, функция возвращает число — хеш-сумму. Изменяемые объекты хешировать нельзя исходя из самого определения хеш-функции. Если массив данных может меняться, значит будет меняться и хеш-сумма. Подобное поведение нарушало бы логику кода. Рассмотрим на примере.

```
x = 42
y = 'text'
z = 3.1415
print(hash(x), hash(y), hash(z))
my_list = [x, y, z]
print(hash(my_list))  # получим ошибку, т.к. list изменяемый
```

Как видите нижняя строка кода вызывает ошибку `TypeError: unhashable type: 'list'` Если вдруг забыли изменяемый объект или нет, просто попробуйте получить его хеш.

Задание

Напишите небольшую программу, которая запрашивает у пользователя любой текст и выводит о нём следующую информацию:

- тип объекта,
- адрес объекта в оперативной памяти,
- хеш объекта.

Результат работы пришлите в чат. У вас 5 минут.

2. Аннотация типов

Учитывая динамическую типизацию, Python не требует аннотацию типов переменных перед их использованием. Однако подобная возможность в языке существует. Появилась она относительно недавно. Указание типов полезно программистам, которые перешли из других языков программирования со статической типизации и привыкли объявлять тип переменной. Также аннотация типов упрощает отладку кода. Ведь IDE может подсказать возможные ошибки, если программист присваивает переменной новый, другой тип данных.
Разберём на примере.

```
a: int = 42
b: float = float(input('Введи число: '))
a = a / b
```

После имени переменной ставим двоеточие и через пробел указываем тип данных. Он совпадает с именем функции-класса, но без круглых скобок на конце.
IDE и линтеры выдадут предупреждение для строки 3 Expected `type 'int', got 'float' instead` Мы пытаемся в переменную a, указанную как хранилище целых чисел сохранить результат деления, т.е. вещественное число. Исправить можно например изменив строку 1 на следующую: `a: float = 42.0`

Далее на курсе мы будем говорим о функциях. Забежим немного вперёд, чтобы рассмотреть пользу указания типов при создании своих функций. Внимание на пример кода:

```python
def my_func(data: list[int, float]) -> float:
    res = sum(data) / len(data)
    return res


print(my_func([2, 5.5, 15, 8.0, 13.74]))
```

Аннотация подсказывает, что в качестве аргумента функция получает список, заполненный целыми или вещественными числами. Результат работы функции - вещественное число.

🔥 **PEP-8!** Код до и после функции отделяется двумя пустыми строками.

Начиная с Python 3.10 возможности указания типов расширились. Например можно указать несколько типов через вертикальную черту. Первый пример из этой главы можно записать так:

```python
a: int | float = 42
b: float = float(input('Введи число: '))
a = a / b
```

Мы сообщили, что в переменной a будем хранить числа, целые или вещественные.

Модуль typing

Для указания типа переменной можно использовать модуль typing. Они содержать как готовые типы данных, так и обобщенные. В таблице представлен доступные на текущий момент типы, разделённые по группам. Назначение большинства из них понятно по названиям. А с некоторыми вы никогда не столкнётесь.

Примитивы супер специаль-ного типа	Абсолютные типы из collections.abc	Структур-ные проверки, протоколы	Коллекция конкретных типов	Другие конкретные типы	Одноразо-вые вещи
Annotated, Any, Callable, ClassVar, Final, ForwardRef, Generic,	AbstractSet, ByteString, Container, ContextManage r, Hashable, ItemsView,	Reversible, SupportsAbs, SupportsByte s, SupportsCom plex, SupportsFloat	ChainMap, Counter, Deque, Dict, DefaultDict, List, OrderedDict,	BinaryIO, IO, Match, Pattern, TextIO	AnyStr, cast, final, get_args, get_origin, get_type_hints, NewType,

Literal, Optional, Protocol, Tuple, Type, TypeVar, Union	Iterable, Iterator, KeysView, Mapping, MappingView, MutableMapping, MutableSequence, MutableSet, Sequence, Sized, ValuesView, Awaitable, AsyncIterator, AsyncIterable, Coroutine, Collection, AsyncGenerator, AsyncContextManager	, SupportsIndex, SupportsInt, SupportsRound	Set, FrozenSet, NamedTuple, TypedDict, Generator		no_type_check, no_type_check_decorator, NoReturn, overload, runtime_checkable, Text, TYPE_CHECKING

Ещё раз напомню, что программа будет работать без указания типа. Более того, в процессе исполнения Python игнорирует аннотации. Если переменная получит значение неподходящего типа, ошибку это не вызовет. Указание типов служит для повышения читаемости кода и более быстрой отладки.

Если вы будете работать в команде, которая придерживается аннотации, вы знаете где искать. Далее на курсе мы будем использовать указание типа там, где это уместно. Но не будет делать аннотацию обязательной.

3. В Python всё объект. Что такое атрибут и метод объекта

С объектами в Python мы разобрались. Всё есть объект. У объектов могут быть атрибуты и методы. Обычно о них говорят при изучении ООП. Но даже используя процедурный подход к написанию кода мы пользуемся объектами "из коробки", а следовательно можем обращаться к их атрибутам и методам.

Атрибуты объекта

Атрибуты — это переменные, конкретные характеристики объекта, такие как цвет поля или имя пользователя. По сути атрибут хранит информацию о состоянии объекта. Для обращения к атрибуту объекта в Python нужно после его имени поставить точку и далее указать название атрибута. Пока мы не научились создавать свои классы, рассмотрим атрибут на следующем примере.

```
print("Hello world!".__doc__)
print(str.__doc__)
```

При запуске программы получим два совершенно одинаковых описания работы строк. Что произошло? Мы обратились к магическому атрибуту строки под названием __doc__. Данный атрибут хранит строку документации, описывающую объект.

И снова о подчёркивании "_"

Как вы только что заметили, мы обратились к атрибуту, в названии которого использован символ подчёркивания. А если точнее, то два подчёркивания до имени и два после. Такие переменные называют магическими или дандер. Магическим может быть как атрибут, хранящий какие-то данные, так и метод.

🔥 **Важно!** Магические атрибуты и методы более подробно рассмотрим во второй части курса, когда речь пойдёт о ООП. Сейчас о них стоит думать, как о внутренних механизмах, которые обеспечивают работу Python с объектами и позволяют программисту писать короткий код не задумываясь о внутренней реализации.

Методы объекта

Методы — это функции, которые описаны внутри объекта или класса. Они относятся к определенному объекту и позволяют взаимодействовать с ними или другими частями кода. По сути метод — это функция класса.

В Python для обращения к методу используется точечная нотация, как и с атрибутами. Отличии в том, что после имени метода ставятся круглые скобки. А если метод ожидает получить данные на вход, они указываются как аргументы в скобках. Рассмотрим на примере:

```
print("Hello world!".upper())
print("Hello world!".count('l'))
```

В первой строке мы вызвали метод строки, который пытается привести её к верхнему регистру. Метод count() принимает на вход аргумент — строку для поиска. В качестве ответа возвращается целое число, количество вхождений переданной строки внутри исходной.

Функция dir()

Один из способов проверки наличия атрибутов и методов у объекта - передача его функции dir() в качестве аргумента. Посмотрим на примере строки, раз уж мы с ней активно работаем в этой главе.

```
print(dir("Hello world!"))
```

Получили огромный список значений. Первая половина списка - дандер методы. Их легко отличить по подчёркиваниям. Обычно их не используют на прямую, если не решают узкоспециализированные задачи или не работают с переопределением методов в ООП. Впрочем, переопределение методов — узкоспециализированная задача.
Вторая часть списка — методы, которые доступны программисту для работы со строками. Тут есть upper и count, которые мы уже использовали и ещё десяток других. Поговорим о строковых методах позже. А пока уточню, что функция dir() полезна, если вы пишите код и забыли название нужного вам метода.

🔥 **Важно!** Некоторые IDE выводят информацию об атрибутах и методах объекта после нажатия точки за именем. IDE неявно вызывает функцию dir().

Функция help()

Встроенная функция help() выводит подсказку об объекте, который передается в качестве аргумента. Таким объектом может быть что угодно, будь то переменная, функция, класс. Можно сказать, что help() - более продвинутая версия dir(). Ведь программист получает не только имена методов и атрибутов объекта, но и описание того как они работают.

Попробуем вызвать "помощь" для "привет мира".

```
help("Hello world!")
```

На первый взгляд не получили ничего хороше, документации нет.

```
No Python documentation found for 'Hello world!'.
Use help() to get the interactive help utility.
Use help(str) for help on the str class.
```

Но если внимательно прочитать описание, станет понятно что:

1. Функция help() без аргументов запускает интерактивный режим. В нём можно указывать имена зарезервированных слов, встроенных функций, модулей и получать справочную информацию.

   ```
   help()
   ```

   Для выхода из режима справки используйте команду quit.

2. Если передать в функцию имя класса, получим подробное описание его работы.

   ```
   help(str)
   ```

Задание

Запустите интерактивный режим справки и проведите два небольших исследования.

1. Введите команду keywords, далее любое интересное вам ключевое слово из списка. Прочитайте описание и напишите в чат пару слов о том, что узнали. У вас 3 минуты.
2. Введите команду symbols, далее любой заинтересовавший вас символ из списка. Прочитайте описание и напишите в чат пару слов о том, что узнали. У вас 3 минуты.

4. Простые объекты

Рассмотрим подробнее особенности работы с простыми объектами, с числами. В финале лекции поговорим о строках. Их можно считать массивами, т.е. коллекциями. Но так как строки в Python неизменяемы и хранят только символы, назовём их условно простыми.

Целые числа, функция int()

Целое число имеет тип int. Для преобразования строки к числу используется одноименная функция.
int(x, base=10).
Первый аргумент — объект, который мы хотим преобразовать в число. Обычно это другое число и или числовая строка. Второй аргумент указывает на основание системы счисления. По умолчанию используется десятичная система, её можно не указывать. Для base допустимы значения от 2 до 36, т.е. от двоичной до тридцати шестиричной системы счисления. Примеры использования.

```
x = int("42")
y = int(3.1415)
z = int("hello", base=30)
print(x, y, z, sep='\n')
```

Мы преобразовали десятичное число из строки в число, отбросили у вещественного числа дробную часть и преобразовали строковую запись числа в тридцатиричной системе счисления в её десятичный числовой аналог.

🔥 **PEP-8!** При указании значений для ключевых аргументов функции пробелы вокруг знака равенства не ставятся.

У целый в Python есть одна полезная особенность. Объект изменяет свои размеры в зависимости от длины целого числа. Переполнения регистра не происходит. В Python "резиновый инт". При этом вы должны понимать, что любой объект хранит в себе "служебную информацию", которая также занимает место в памяти. Воспользуемся функцией getsizeof() из модуля sys, чтобы посмотреть на затраты памяти под целым числом.

```
import sys

STEP = 2 ** 16
num = 1
for _ in range(30):
    print(sys.getsizeof(num), num)
    num *= STEP
```

🔥 **PEP-8!** После импорта модулей ставится пустая строка.

Для хранения "единицы" в 64-х разрядной версии Python тратится 28(!) байт памяти. Это объект со своей служебной информацией и несколькими байтами под само число.

При этом мы можем хранить огромные числа, превышающие long integer на много порядков без проблем и лишних приёмов программирования. Число Гугол, т.е. 10 в степени 100 займет всего лишь 72 байта.

```
print(sys.getsizeof(10 ** 100))
```

Формат представления числа. Снова о "_".

Ещё одна особенность, которая упрощает чтение больших чисел появилась в Python 3.6. Это символ подчеркивания в качестве разделителя групп цифр. Да, снова он. И не в последний раз.
num = 7_901_123_456_789
Интерпретатор опускает символ подчеркивания. Они нужны лишь для программиста, читающего код.

🔥 **Внимание!** Кроме того, обратите внимание на цикл из примера кода о "резиновом инт".

for _ in range(30):

Конструкция цикла for in ожидает, что после for указывается переменная для приёма значений, которые берутся из итератора указанного после in. Но если внутри цикла значения не нужны, в качестве имени переменной используют подчеркивание "_". Важно понимать, что использование подчеркивания в теле цикла неверно. Скорее всего вам нужна переменная i, item или другая подобная. Если подчёркиваем, то только один раз.

Функции bin(), oct(), hex()

Помимо десятичной системы у Python есть функции для строкового представления чисел в двоичной, восьмеричной и шестнадцатеричной системах счисления.

```
num = 2 ** 16 - 1
b = bin(num)
o = oct(num)
```

```
h = hex(num)
print(b, o, h)
```

Обратите внимание на приставку в начале числа, состоящую из нуля и латинской строчной буквы: 0b — двоичное, 0o — восьмеричное, 0x — шестнадцатеричное представление. Подобная запись сразу говорит о системе счисления. А само представление в формате, отличном от десятичного, применяется для решения узкого спектра задач.

🔥 **PEP-8!** В качестве имени переменной мы использовали латинскую строчную "o". Это допустимое имя. Но никогда не используйте символы 'l' (строчная буква "л"), 'O' (заглавная буква "О") или 'I' (заглавная буква "Ай") в качестве имен переменных, состоящих из одного символа. В некоторых шрифтах эти символы неотличимы от цифр один и ноль.

Вещественные числа, функция float()

Числа с плавающей запятой представлены классом float. Для хранения таких чисел ПК, а не только Python используют особый формат представления числа: мантисса и порядок. Так число 23321.345 правильнее было бы представить как 2.3321345*10**-4.
Особенности подобного формата хранения чисел могут приводить к погрешностям вычисления.

```
print(0.1 + 0.2)
```

Вместо ожидаемого 0.3 получили 0.30000000000000004 При округлении это будет тот же 0.3. Но когда пишешь код, ожидаешь получить верный результат сразу.

Вторая особенность вещественных чисел - ограничение на хранения информации. Попробуем сохранить достаточно большое по количеству знаков число с плавающей точкой.

```
pi = 3.141_592_653_589_793_238_462_643_383_279_502_884_197_169_399_375
print(pi)
```

При выводе на печать, а если быть точным, то в момент сохранения объекта в память в качестве вещественного числа произошло отбрасывание части цифр. В

итоге в памяти осталось 3.141592653589793. И, да, вы верно заметили. Подчёркивание можно использовать для удобства чтения чисел любого типа.

В случае с математикой вещественных чисел в Python стоит ожидать подобных погрешностей вычисления и округления. О том как их избежать узнает в конце сегодняшней лекции. Пока же стоит понять, что float является компактным и быстрым типом данных для математических операций с плавающей запятой, если нам не требуется высокая точность результата.

Логические типы, функция bool()

С True и False мы уже знакомы. Это две неизменные константы, которые являются ответвлением числового типа. Функция bool() возвращает одно из двух значений, в зависимости от входного значения. У "логики" есть несколько полезных особенностей, которые облегчают разработку.
Все числа преобразуются к истине, за исключением нуля. Он считается ложью.

```
DEFAULT = 42
num = int(input('Введите уровень или ноль для значения по
умолчанию: '))
level = num or DEFAULT
print(level)
```

Любая строка текста также приводится к истине. Ложью является пустая строка (не путайте с пробелом), т.е. строка длиною в ноль символов.

```
name = input('Как вас зовут? ')
if name:
    print('Привет, ' + name)
else:
    print('Анонимус, приветствую')
```

Коллекции, о которых будем подробного говорить на следующей лекции приводятся к истине, если в них есть элементы. Пустая коллекция — ложь.

```
data = [0, 1, 1, 2, 3, 5, 8, 13, 21]
while data:
    print(data.pop())
```

Работу данного примера подробнее разберём на следующем уроке. Но если коротко, в цикле мы удаляем элементы списка до тех пор пока список не окажется пустым.

🔥 **Внимание!** Обратите внимание, что во всех примерах происходило неявное приведение к логическому типу, мы явно не использовали функцию bool().

Строки, функция str()

В Python нет типа данных символ. Есть только класс str. В нём можно хранить как один символ, аналог char в си-подобных языках, так и любое большее количество символов. Почему символы, а не буквы? Для хранения используется кодировка utf-8, т.е. в строке могут быть не только буквы, цифры, знаки препинания, но и иероглифы, смайлики и всё то, что хранится в кодировке Unicode.

При работе со строками стоит помнить, что это неизменяемый тип данных. При этом str можно представить как коллекцию символов — массив. Выходит, что строка является коллекцией? Верно.

На этой лекции рассмотрим несколько приёмов работы со строками как с неизменяемыми объектами. А на следующей поговорим о строке как о коллекции символов.

Способы записи строк

Python допускает одинарные или двойные кавычки для записи строки. Одни кавычки могут включать другие, но они не должны смешиваться.

```
txt = 'Книга называется "Война и мир".'
```

Тройный двойные кавычки позволяют писать текст в несколько строк. Если такой текст не присвоить переменной, он превращается в многострочный комментарий. Подобные комментарии используют для создания документации к коду. Например так:

```
class str(object):
    """
```

```
    str(object='') -> str
    str(bytes_or_buffer[, encoding[, errors]]) -> str
    ...
    """
```

Ещё один способ записать строки, перечислить их последовательно друг за другом.

```
text = 'Привет.' 'Как ты, друг?' 'Рад тебя видеть.'
print(text)
```

Такой приём работает, но считается плохим стилем. Плюс, а точнее минус, строки соединились без пробелов.

На этом способы записи строк не заканчиваются. Посмотрите на этот код:

```
very_long_text = 'Lorem ipsum dolor sit amet, consectetur
adipisicing elit. A ab alias animi assumenda at aut ' \
                 'commodi, consequatur cumque ea harum, hic id
illum ipsam itaque laboriosam magnam minus nam nulla ' \
                 'numquam obcaecati officia officiis porro
possimus praesentium quaerat temporibus ullam veniam? '
```

Как вы помните, PEP8 рекомендует не писать более 120 символов в одной строке. Обратный слеш “\” позволяет писать продолжение с новой строки. Требования в 120 символов выполняются. А Python воспринимает несколько строк как одну.

🔥 **Важно!** Подобный приём с обратным слешем работает не только для строк текста, но и для кода. Очень длинную строку можно разделить записать в несколько. Но помните, что зачастую такие строки трудно читать. Возможно не стоит лепить решение в одну мега строчку, а разбить его на более читаемый вариант.

Конкатенация строк

Отдельный способ создания строк — конкатенация. Или говоря проще, сложение. Разберём на примере и обсудим все его нюансы:

```
LIMIT = 120
ATTENTION = 'Внимание!'
```

```
name = input('Твоё имя? ')
age = int(input('Твой возраст? '))
text = ATTENTION + ' Хоть тебе и осталось ' + str(100 - age) +\
       " до ста лет, но длинна строки не должна превышать " +
str(LIMIT) + ' символов.'
print(text)
```

Переменная text получилась в результате сложения нескольких строк. При этом:

- Все элементы должны быть строками. Иначе получим ошибку вида: TypeError: can only concatenate str (not "int") to str Именно для этого мы обернули переменные с числами, такие как LIMIT, функцией str()
- Если в переменной хранится строковое значение, оборачивать её в str() не нужно. Получится масло масляное.
- Складывать можно строки в одинарных, двойных и даже в трех двойных кавычках. Но лучше выбрать единый стиль записи строк и придерживаться его во всём проекте.
- Если код не влазит в 120 символов строки, не стесняемся использовать обратный слеш для его разделения на несколько строк.

Конкатенация строк затратна по памяти и по времени. Для простых задач допустим подобный подход. В реальных проектах конкатенация используется для формирования констант. В остальных случаях используют способы форматирования строк. О них поговорим на следующей лекции.

Кстати, рекомендации Google по стилю кода, которые являются продвинутой версией PEP-8 с дополнительными требования запрещают программистам использовать конкатенацию строк, особенно если она происходит внутри цикла. Почему? Поговорим о размере строки в памяти.

Размер строки в памяти

Строки как объекты тратят память на служебную информацию, а как массивы на хранение текста. В 64-х разрядной версии Python служебная информация занимает 48 байт. Разберём пример кода:

```
empty_str = ''
en_str = 'Text'
ru_str = 'Текст'
unicode_str = '😀😍😉🙃'
print(empty_str.__sizeof__())
print(en_str.__sizeof__())
print(ru_str.__sizeof__())
print(unicode_str.__sizeof__())
```

Во-первых обратите внимание на магический метод __sizeof__(). Он работает аналогично sys.getsizeof и возвращает количество байт занятых объектом. Почему же пустая строка заняла 49 байт, если служебная информация использует 48? Один байт - символ конца строки.

Теперь посмотрим на текст. Английские буквы тратят по одному байту на символ. Если же речь идёт о русском языке или любых других символах, кодирование занимает 2 или 4 байта. Это особенность хранения информации в кодировке UTF-8 и фишка языка Python для доступа к букве по индексу.
А теперь зная, что строка тратит много памяти, что строка неизменяемый тип данных и что при конкатенации строк создаются новые объекты, которые занимают дополнительную память вы сами можете сделать вывод почему сложение строк не приветствуется.

Методы проверки строк

Ряд методов анализируют строку текста и возвращают истину или ложь в зависимости от содержимого строки. Часто используемые методы перечислены в таблице:

Название метода	**Описание**
str.isalnum()	Возвращает True, если все символы в строке буквенно-цифровые. Символ является буквенно-цифровым, если одно из следующих значений возвращает True: c.isalpha(), c.isdecimal(), c.isdigit()или c.isnumeric().
str.isalpha()	Возвращает True, если все символы в строке являются буквенными. Алфавитные символы — это символы, определенные в базе данных символов Юникода как «буква»
str.isdecimal()	Возвращает True, если все символы в строке являются десятичными символами
str.isdigit()	Возвращает True, если все символы в строке являются цифрами. Цифры включают десятичные символы и цифры, требующие специальной обработки, например цифры надстрочного индекса совместимости.
str.isnumeric()	Возвращает True, если все символы в строке являются

Название метода	Описание
str.isalnum()	Возвращает True, если все символы в строке буквенно-цифровые. Символ является буквенно-цифровым, если одно из следующих значений возвращает True: c.isalpha(), c.isdecimal(), c.isdigit()или c.isnumeric().
	числовыми символами. Числовые символы включают цифровые символы и все символы, которые имеют свойство числового значения Unicode
str.isascii()	Возвращает True, если строка пуста или все символы в строке ASCII
str.islower()	Возвращает True, если все символы в строке в нижнем регистре
str.istitle()	Возвращает True, если строка является строкой с заглавным регистром и содержит хотя бы один символ
str.isupper()	Возвращает True, если все символы в строке в верхнем регистре
str.isprintable()	Возвращает True, если все символы в строке доступны для печати или строка пуста. Непечатаемые символы — это символы, определенные в базе данных символов Unicode как «Другие» или «Разделители», за исключением пробела ASCII (0x20), который считается печатаемым.
str.isspace()	Возвращает True, если в строке есть только пробельные символы

🔥 **Внимание!** Обратите внимание, что методы начинаются с приставки is. Подобный приём часто используется в программировании. Если переменная содержит логическое значение или функция/метод возвращает логическое значение, в начале добавляют приставку is — намёк на истину или ложь в качестве результата.

Задание

Напишите небольшую программу, которая запрашивает у пользователя текст. Если текст можно привести к целому числу, выведите его двоичное, восьмиричное и шестнадцатиричное представление. А если преобразование к целому невозможно, сообщите написан ли текст в ASCII или нет. Результат работы отправьте в чат. У вас 7 минут.

5. Математика в Python

Немного математики в финале лекции. Не пугайтесь, если что-то подзабыли. Разберём в общих чертах математические некоторые математические возможности Python. В решение сложных математических задач углубляться не будем.
В Python есть несколько модулей в стандартной библиотеке, которые облегчают математические расчёты. Для доступа к ним необходимо выполнить импорт в начале файла.

```
import math
import decimal
import fractions
```

Модуль math

В математическом модуле содержаться константы, например число "пи", "е", бесконечность и др.

```
print(math.pi, math.e, math.inf, math.nan, math.tau, sep='\n')
```

Кроме того внутри модуля есть множество математических функций: синусы, косинусы, тангенсы, логарифмы, факториал.
Вы можете подробно изучить многообразие модуля самостоятельно. В этом вам помогут функции dir() и help()

Модуль decimal

Помните пример про точность вещественных чисел: `print(0.1 + 0.2)` ?

Модуль decimal позволяет хранить числа с плавающей запятой и проводить с ними математические выселения без потери точности и ошибок преобразования. Для этого надо воспользоваться классом Decimal из модуля. Рассмотрим на примере.

```
pi =
decimal.Decimal('3.141_592_653_589_793_238_462_643_383_279_502_88
4_197_169_399_375')
print(pi)
num = decimal.Decimal(1) / decimal.Decimal(3)
print(num)
```

Передав в качестве аргумента число пи в виде строки, мы получаем его математическое представление без потери точности. Помимо строки аргументом может быть целое или вещественное число. Например разделив 1 на 3 мы получаем вещественно число с большой точность знаков после запятой.
Часто при работе с модулем указывают точность вычислений. По умолчанию она равна 28 знакам до и после запятой. Например нам надо что-то посчитать с точность 120 знаков:

```
decimal.getcontext().prec = 120
science = 2 * pi * decimal.Decimal(23.452346) ** 2
print(science)
```

В первой строке кода задали точность через обращение к атрибуту prec у метода getcontext(). Дальше считаем как обычно.

🔥 **Важно!** Если вы планируете писать код для банковской системы или любой другой, где важна точность расчётов, используйте модуль decimal вместо класса float.

Модуль fraction

Для работы с дробями есть модуль fractions. Откроем пример, чтобы сразу стало ясно как он работает.

```
f1 = fractions.Fraction(1, 3)
print(f1)
f2 = fractions.Fraction(3, 5)
print(f2)
print(f1 * f2)
```

Передаём в класс Fractions через запятую числитель и знаменатель дроби. Далее работаем как с обычными числами. Модуль полезен при работе с бесконечными дробями, когда decimal не обеспечивает необходимую точность хранения числа.

Класс complex()

Если вы проходили комплексные числа в школе или вузе, поздравляю. Python позволяет работать и с ними. Если нет — не переживайте. Скажу о них всего пару слов. Комплексный числа — числа с мнимой единицей, квадратным корнем из минус одного.

```
a = complex(2, 3)
b = complex('2+3j')
print(a, b, a == b, sep='\n')
```

В каждом случае получим `(2+3j)` — комплексное число в Python.

🔥 **Внимание!** Обратите внимание на то, что Python для обозначения мнимой единицы использует букву "j", а в математике принято использовать "i". Во всём остальном математические операции в Python совпадают с классической математикой.

Математические функции "из коробки"

В финале лекции несколько слов о математических функциях "из коробки".

- `abs(x)` — возвращает абсолютное значение числа x, число по модулю.
- `divmod(a, b)` — функция принимает два числа в качестве аргументов и возвращает пару чисел — частное и остаток от целочисленного деления. Аналогично вычислению a // b и a % b.
- `pow(base, exp[, mod])` — при передаче 2-х аргументов возводит base в степень exp. При передаче 3-х аргументов, результат возведения в степень делится по модулю на значение mod.
- `round(number[, ndigits])` — округляет число number до ndigits цифр после запятой. Если второй аргумент не передать, округляет до ближайшего целого.

6. Выводы

На этой лекции мы:

1. Познакомились со строгой динамической типизацией языка Python.
2. Изучили понятие объекта в Python. Разобрались с атрибутами и методами объектов.
3. Рассмотрели способы аннотации типов.
4. Изучили "простые" типы данных, такие как числа и строки.
5. Узнали про математические возможности Python.

Краткий анонс следующей лекции

1. Разбёмся что такое коллекция и какие коллекции есть в Python
2. Изучим работу со списками, как с самой популярной коллекцией.
3. Узнаем как работать со строкой в ключе коллекция
4. Разберём работу с кортежами
5. Узнаем что такое словари и как с ними работать
6. Изучим множества и особенности работы с ними
7. Разберём работу с байтами как с массивами

Домашнее задание

Поработайте со справочной информацией в Python, функцией `help()`. Попробуйте найти зарезервированные слова, функции и модули, которые прошли за две лекции и почитать про них. Если вы плохо читаете на английском, воспользуйтесь любым онлайн переводчиком.