

ЦЕЛЬ И СТРУКТУРА ЛАБОРАТОРНОГО ПРАКТИКУМА	3
ЛАБОРАТОРНАЯ РАБОТА №1. ВВЕДЕНИЕ В PYTHON. ВЫБОР И НАСТРОЙКА IDE.....	5
Цель:	5
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ:.....	5
ПРАКТИЧЕСКИЕ ЗАДАНИЯ:.....	9
КОНТРОЛЬНЫЕ ВОПРОСЫ:	10
ЛАБОРАТОРНАЯ РАБОТА №2. ТИПЫ И СТРУКТУРЫ ДАННЫХ	11
Цель:	11
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ:.....	11
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	18
КОНТРОЛЬНЫЕ ВОПРОСЫ	19
ЛАБОРАТОРНАЯ РАБОТА №3. УСЛОВНЫЕ ОПЕРАТОРЫ И ЦИКЛЫ	20
Цель:	20
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	20
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	25
КОНТРОЛЬНЫЕ ВОПРОСЫ	26
ЛАБОРАТОРНАЯ РАБОТА №4. МЕТОДЫ И ФУНКЦИИ	28
Цель:	28
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	28
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	32
КОНТРОЛЬНЫЕ ВОПРОСЫ	33
ЛАБОРАТОРНАЯ РАБОТА №5. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ.....	34
Цель:	34
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	34
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	39
КОНТРОЛЬНЫЕ ВОПРОСЫ	41
ЛАБОРАТОРНАЯ РАБОТА №6. ОБРАБОТКА ОШИБОК И ИСКЛЮЧЕНИЙ. МОДУЛИ И ПАКЕТЫ. NUMPY ...	42
Цель:	42
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	42
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	54
КОНТРОЛЬНЫЕ ВОПРОСЫ	55
ЛАБОРАТОРНАЯ РАБОТА №7. PANDAS.....	56
Цель:	56
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	56
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	75

КОНТРОЛЬНЫЕ ВОПРОСЫ	78
ЛАБОРАТОРНАЯ РАБОТА №8. РАБОТА С API.....	79
ЦЕЛЬ:	79
ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ	79
ЗАДАЧА.....	81
ПРАКТИЧЕСКИЕ ЗАДАНИЯ.....	84
КОНТРОЛЬНЫЕ ВОПРОСЫ	85
СПИСОК ЛИТЕРАТУРЫ	86

Цель и структура лабораторного практикума

Python – высокоуровневый, интерпретируемый, объектно-ориентированный, интерактивный язык программирования, сочетающий в себе минималистичный синтаксис с высокой производительностью. За последние десять лет Python превратился в один из самых важных языков, применяемых в науке о данных, в машинном обучении и разработке ПО общего назначения в академических учреждениях и промышленности.

Согласно данным GitHub, Python входит в тройку самых популярных языков программирования. Python требуется в 84% вакансиях при устройстве на должность аналитика и почти в 100% вакансий, связанных с Data Science.

Python позволяет решить задачу автоматизации сбора данных, обработки данных, ускоряет анализ данных и позволяет реализовать на работе новые подходы к анализу, например, решать задачи с помощью обучения нейросетей [8].

Перечислим свойства этого языка, за которые его можно считать универсальным языком для анализа данных:

1. Высокая продуктивность разработки. Язык интерпретируемый, поэтому на нём можно писать быстрее, чем, например, на C. Неявная, но строгая типизация обеспечивает меньший объём кода для решения задач, чем в Java. А лаконичный и ясный синтаксис позволяет быстро писать читабельный код.
2. Низкий порог входа для изучения. Одним из главных преимуществ является простота его изучения. Любой человек с минимальными знаниями программирования может без проблем освоить принципы этого языка.
3. Открытый исходный код. Один из главных плюсов Python – это наличие множества библиотек с открытым исходным кодом, которые значительно упрощают работу с данными.
4. Динамичное развитие языка. Ещё одним аргументом в пользу Python является то, что этот язык быстро и интенсивно развивается. С каждой версией производительность языка повышается, а синтаксис совершенствуется. В Python процесс стандартизации более открыт для комьюнити, каждый может предложить свои идеи, и их количество быстро растёт. Пользователи Python, нуждающиеся в помощи, всегда могут обратиться к Stack Overflow, спискам рассылки, а также к коду и документации, предоставленным пользователями.

По окончании практикума, студенты:

- Узнают, как использовать Jupyter Notebook.
- Будут разбираться в синтаксе.
- Научатся проводить операции с разными типами данных, в особенности – познакомятся с методами работы со списками, словарями и кортежами.
- Узнают о динамической типизации – отличительной особенности Python.
- Освоят методы работы со строковыми данными.
- Научатся работать с условными операторами и циклами.
- Будут знать, как создавать и вызывать функции, особенности их использования.
- Научатся использовать объектно-ориентированное программирование, как создавать классы в Python.
- Узнают, что такое модули и пакеты, как создавать и использовать собственные модули.
- Поймут, как обрабатывать ошибки и исключения.
- Научатся работать с файлами разных форматов.
- Познакомятся с массивами в модуле NumPy, изучат операции с ними, проведут математические и статистические операции.
- Будут использовать готовые функции библиотеки Pandas, объединять данные из разных источников, фильтровать и сортировать данные.
- Освоят встроенные в Pandas и в других модулях графические функции для создания простейших графиков.
- Овладеют навыками работы с открытыми API различных сервисов.

Первая часть курса посвящена основам языка Python. В дальнейшем эти знания могут пригодиться, если студент захочет углубиться в другие области, не обязательно в анализ данных.

Вторая часть направлена на освоение библиотек и инструментов, необходимых аналитику данных. В этой части будут разобраны примеры решения реальных задач, чтобы студенты понимали, с чем может столкнуться аналитик данных на работе.

Лабораторный практикум состоит из 8 модулей, в каждом модуле содержится 5-10 небольших заданий.

Лабораторная работа №1. Введение в Python. Выбор и настройка IDE

Цель:

Ознакомиться со способами установки Python и настройки IDE.

Теоретический материал:

Язык программирования Python является, пожалуй, одним из самых популярных. Он используется в веб-разработке, Data Science и Data Analysis, системах автоматизации, приложениях. Но главное преимущество Python – это его низкий порог входа. Иными словами, обучиться программированию на этом языке может практически каждый.

В данном пособии рассматриваются основы языка Python, а также применение этого языка в анализе данных.

В качестве IDE выбран Jupyter Notebook. Jupyter Notebook – невероятно мощный инструмент для интерактивной разработки и представления проектов в области наук о данных. Это среда разработки, где сразу можно видеть результат выполнения кода и его отдельных фрагментов. Код в ноутбуках хранится в независимых ячейках и его можно запускать в любом порядке или поодиночке.

Для новичка проще всего начать работу с Jupyter Notebook, установив дистрибутив Anaconda. Anaconda является наиболее широко используемым дистрибутивом Python для работы с данными и поставляется с предустановленными наиболее популярными библиотеками и инструментами. Это позволит вам приступить к работе, без хлопот управления бесчисленными установками или беспокойства о зависимостях и проблемах установки, связанных с ОС [10].

Установка интерпретатора Python

Для установки интерпретатора Python на ваш компьютер, первое, что нужно сделать – это скачать дистрибутив. Загрузить его можно с официального сайта [19].

Порядок установки:

1. Запустите скачанный установочный файл.
2. Выберите способ установки.



Рисунок 1 – Выбор способа установки

В окне предлагается два варианта Install Now и Customize installation. При выборе Install Now, Python установится в папку по указанному пути. Помимо самого интерпретатора будет установлен IDLE (интегрированная среда разработки), pip (пакетный менеджер) и документация, а также будут созданы соответствующие ярлыки и установлены связи файлов, имеющие расширение .py с интерпретатором Python. Customize installation – это вариант настраиваемой установки. Опция Add python 3.10 to PATH нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.

3. Отметьте необходимые опции установки (доступно при выборе Customize installation).

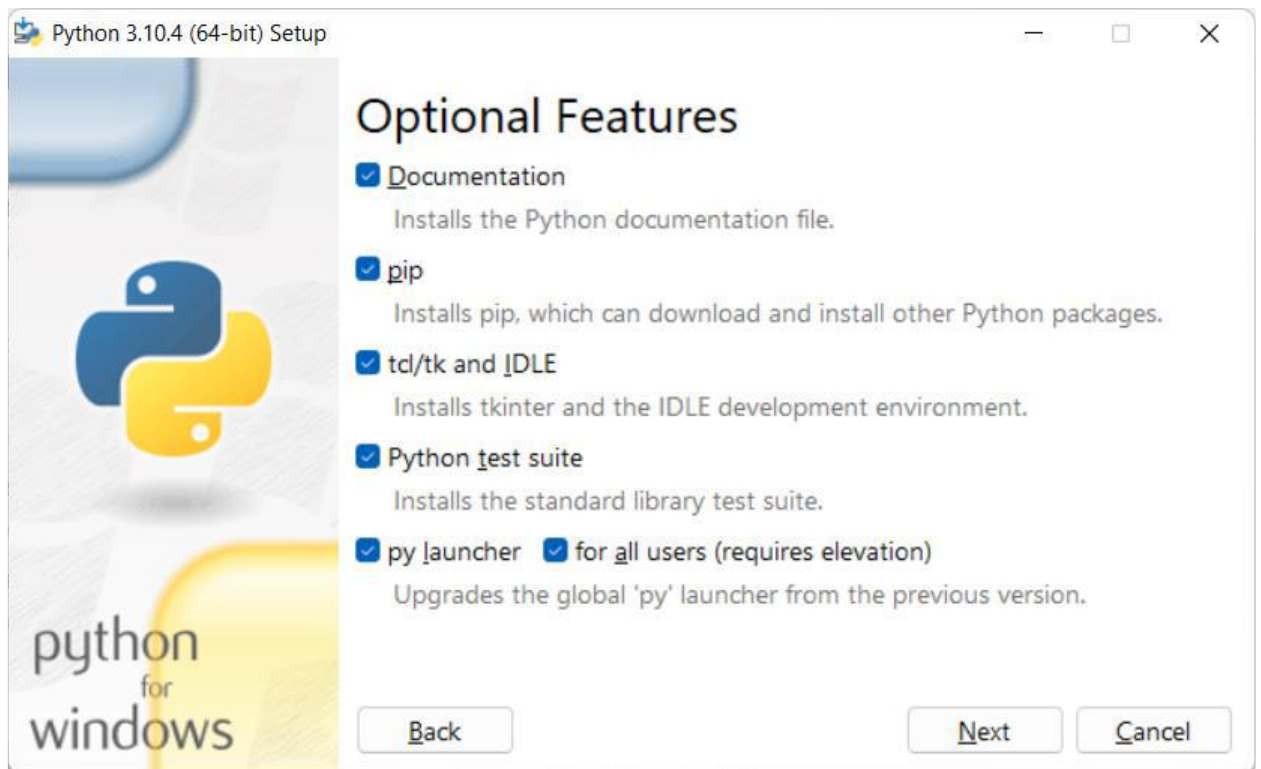


Рисунок 2 – Дополнительные опции

На этом шаге предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Рекомендуется выбрать все опции.

4. Выберите место установки (доступно при выборе Customize installation).

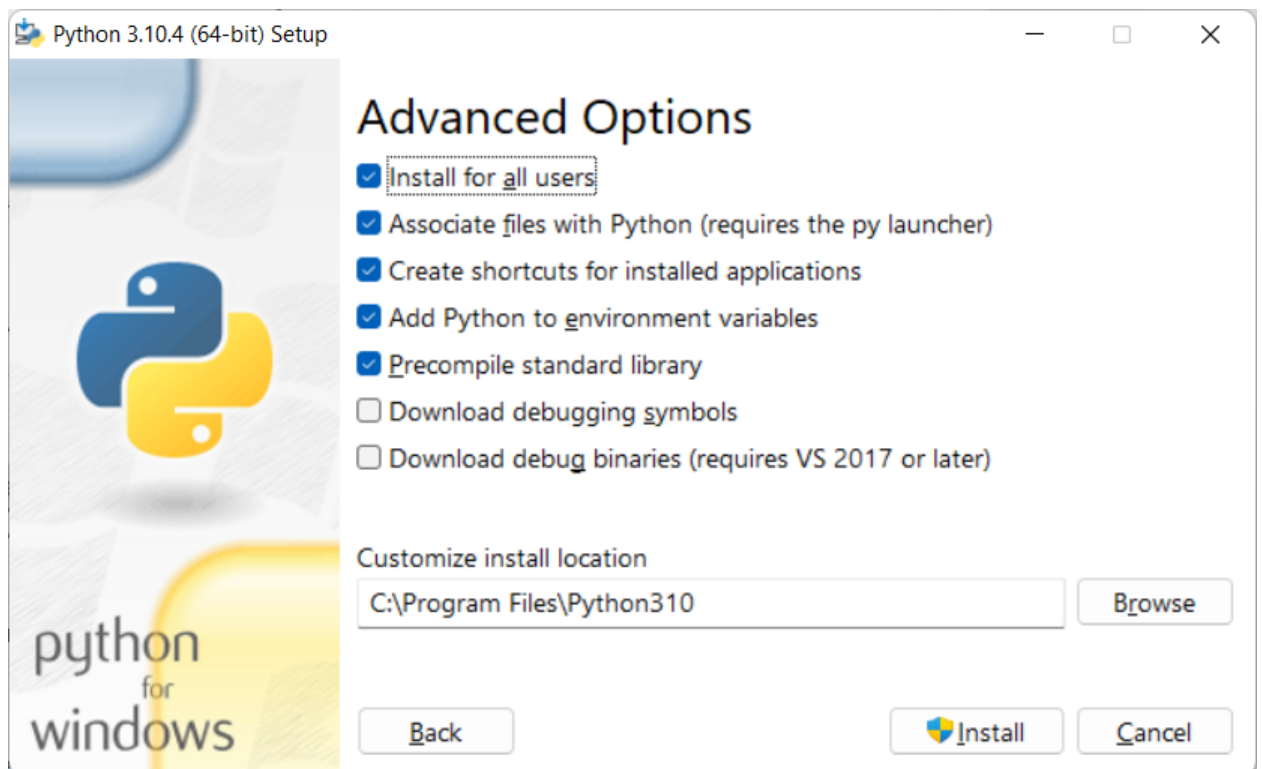


Рисунок 3 – Расширенные возможности

Помимо указания пути, данное окно позволяет внести дополнительные изменения в процесс установки с помощью опций:

Install for all users – Установить для всех пользователей. Если не выбрать данную опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор.

Associate files with Python – Связать файлы, имеющие расширение .py, с Python. При выборе данной опции будут внесены изменения в Windows, позволяющие запускать Python скрипты по двойному щелчку мыши.

Create shortcuts for installed applications – Создать ярлыки для запуска приложений.

Add Python to environment variables – Добавить пути до интерпретатора Python в переменную PATH.

Precompile standard library – Провести прекомпиляцию стандартной библиотеки.

Последние два пункта связаны с загрузкой компонентов для отладки, их мы устанавливать не будем.

5. После установки вас ждет сообщение о успешной установке.

Установка Anaconda

Необходимо скачать дистрибутив с официального сайта [11].

Есть варианты под Windows, Linux и MacOS.

1. Запустите скачанный инсталлятор. В первом появившемся окне необходимо нажать «Next».

2. Далее следует принять лицензионное соглашение.

3. Выберите одну из опций установки:

- Just Me – только для пользователя, запустившего установку;
- All Users – для всех пользователей.

4. Укажите путь, по которому будет установлена Anaconda.

5. Укажите дополнительные опции:

- Add Anaconda to the system PATH environment variable – добавить Anaconda в системную переменную PATH;
- Register Anaconda as the system Python 3.5 – использовать Anaconda, как интерпретатор Python 3.5 по умолчанию. Для начала установки нажмите на кнопку “Install”.

6. После этого будет произведена установка Anaconda на ваш компьютер.

7. Для проверки установки откройте Anaconda Navigator, найдите Jupyter Notebook и нажмите Launch. В результате запустится веб-сервер и среда разработки в браузере.

Создайте ноутбук для разработки, для этого нажмите на кнопку New (в правом углу окна) и в появившемся списке выберете Python.

В результате будет создана новая страница в браузере с ноутбуком. Введите в первой ячейке команду:

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

Нажмите Shift +Enter на клавиатуре. Ниже ячейки должна появиться соответствующая надпись.

Горячие клавиши в Jupyter Notebook:

- Выполнение ячейки: Ctrl + Enter или Shift + Enter.
- Комментирование строки: Ctrl + /.
- Создать ячейку выше: a.

Обратите внимание, эта команда и команды ниже будут работать, если у вас выделена ячейка (она подсвечена голубым, при этом курсор не мелькает внутри ячейки)

- Создать ячейку ниже: b.
- Объединить выделенные ячейки: Shift + m.
- Вырезать ячейки: x.
- Копировать ячейки: c.
- Вставить скопированные/вырезанные ячейки: v.
- Изменить тип ячейки на код: y.
- Изменить тип ячейки на Markdown: m.

Больше информации можно прочитать по ссылке
<https://towardsdatascience.com/jupyter-notebook-shortcuts-bf0101a98330>. 36

Практические задания:

1. Выполните установку Python на вашем ПК, выбрав Customize installation. Изучите параметры установки в режиме Customize installation.
2. Выполните установку Anaconda. Изучите содержимое дистрибутива Anaconda.
3. Создайте новый ipynb-файл в Jupyter Notebook.
4. Протестируйте работоспособность выполненных установок, написав первую программу "Hello, World".

Контрольные вопросы:

1. Какие версии Python доступны на текущий момент? В чем принципиальное отличие версий 2.xx и 3.xx? Как получить текущую версию Python?
2. Какие опции доступны при установке в режиме Customize installation? Пояснить за что отвечает каждая опция.
3. В каком файле можно посмотреть информацию о environment variables?
4. Для чего используется опция Add python 3.10 to PATH.
5. Перечислите базовые библиотеки дистрибутива. Изучите содержимое дистрибутива Anaconda (не менее 10).
6. Какое расширение имеют файлы Python?
7. Чувствителен ли Python к регистру символов (заглавная или прописная буквы)?
8. Каким образом можно закомментировать строку в Python?
9. Как прокомментировать несколько строк в Python?
10. Что такое строки документации в Python?

Лабораторная работа №2. Типы и структуры данных

Цель:

Познакомиться с различными типами и структурами данных Python.

Теоретический материал:

В таблице приведены встроенные в Python типы и структуры данных.

Название	Тип	Описание
Integers	int	Целые числа: -1 3 300 200
Floating point	float	Числа с плавающей запятой: 2.3 4.6 100.0
Strings	str	Строки - упорядоченные последовательности символов: "hello" 'Sammy' "2000"
Lists	list	Списки - упорядоченные последовательности объектов: [10,"hello",200.3]
Dictionaries	dict	Словари - неупорядоченные пары Ключ:Значение: {"key" : "value" , "name" : "Frankie"}
Tuples	tup	Кортежи - упорядоченные неизменяемые последовательности объектов: (10,"hello",200.3)
Sets	set	Множества - неупорядоченные наборы уникальных объектов: {"a","b"}
Booleans	bool	Логические значения, указывающие True или False

Numbers Числа

Python имеет различные «типы» чисел. В основном мы сосредоточимся на целых числах и числах с плавающей запятой.

Интерпретатор действует как простой калькулятор. Синтаксис выражений прост: операторы +, -, *, / работают так же, как и в большинстве других языков; круглые скобки () можно использовать для группировки

Базовая арифметика:

```
In [1]: 2+2 # Сумма
Out[1]: 4
In [2]: 10-3 # Вычитание
Out[2]: 7
In [3]: 5*4 # Умножение
Out[3]: 20
In [4]: 35/2 # Деление
Out[4]: 17.5
In [5]: 7//4 #Целая часть от деления
Out[5]: 1
In [6]: 10%3 #Остаток от деления
Out[6]: 1
In [7]: 2**3 #Возведение в степень
Out[7]: 8
```

```
In [8]: 4**0.5 #Корень из числа
Out[8]: 2.0
In [9]: (50 - 5*6) / 4
Out[9]: 5.0
```

Переменные

Чтобы присвоить переменной значение используется знак равенства:

name = object.

Правила для имен переменных:

- 1) имена не могут начинаться с цифры;
- 2) имена не могут содержать пробелы, вместо этого используйте `_`;
- 3) имена не могут содержать ни одного из этих символов: `:"'<>/?|!@#%^&*~+;`
- 4) считается лучшей практикой (PEP8), чтобы имена были в нижнем регистре с символами подчеркивания;
- 5) избегайте использования встроенных ключевых слов Python, таких как *list* и *str*;
- 6) избегайте использования отдельных символов `l`, `O` и `I`, так как их можно спутать с 1 и 0.

Python использует динамическую типизацию, что означает, что вы можете переназначать переменные для разных типов данных. Это делает Python очень гибким в назначении типов данных и отличает его от других языков со статической типизацией [12].

У динамической типизации в Python есть свои плюсы и минусы.

Плюсы в том, что отсутствие необходимости указывать фактический тип данных экономит вам много времени и позволяет вам действительно легко и быстро создавать код Python. И это также делает ваш код очень читабельным.

Минусы в том, что это может привести к ошибкам неожиданного типа данных, потому что у вас нет ограничений, особенно когда вы имеете дело с пользовательским вводом.

```
In [10]: #Создадим переменную a и присвоим ей значение 10
a = 10
In [11]: a
Out[11]: 10
```

Здесь мы присвоили целочисленный объект 10 имени переменной *a*. Давайте назначим *a* что-то еще:

```
In [12]: a = 20
In [13]: a
Out[13]: 20
```

Теперь вы можете использовать *a* вместо числа 20:

```
In [14]: a + a
Out[14]: 40
```

Python позволяет переназначать переменные со ссылкой на один и тот же объект.

```
In [15]: a = a + 10
In [16]: a
Out[16]: 30
```

Вы можете проверить, какой тип объекта назначен переменной, используя встроенную *type()* функцию Python.

```
In [17]: type(a)
Out[17]: int
In [18]: a = a/4
In [19]: a
Out[19]: 7.5
In [20]: type(a)
Out[20]: float
```

Strings Строки

Строки в Python представляют собой последовательность символов, заключенные в одинарные или двойные кавычки. Строки – это упорядоченные последовательности. Например, Python понимает строку «hello» как последовательность букв в определенном порядке.

```
In [21]: 'hello'
Out[22]: 'hello'
In [23]: "Hello, world!"
Out[23]: 'Hello, world!'
In [24]: print('Hello, world')
Hello, world
```

Встроенная в Python функция *len()* подсчитывает все символы в строке, включая пробелы и знаки препинания.

```
In [25]: #Длина строки
len("Hello, world")
Out[25]: 12
```

Мы знаем, что строки представляют собой последовательность, а это означает, что Python может использовать индексы для вызова частей последовательности. В Python мы используем скобки `[]` после объекта для вызова его индекса. Индексация начинается с 0. На рисунке 4 наглядно представлена прямая и обратная индексация строк.

Character :	h	e	l	l	o
Index :	0	1	2	3	4
Reverse Index:	0	-4	-3	-2	-1

Рисунок 4 – Индексация строк

```
In [26]: mystring = 'Hello, world'
In [27]: print(mystring)
Hello, world
```

```
In [28]: mystring[0]
Out[28]: 'H'
In [29]: mystring[-1]
Out[29]: 'd'
```

Мы можем получить не один символ, а срез строки.

Синтаксис среза – *[start : stop : step]*

start – от какого элемента берём значения, по умолчанию равно 0.

stop – до какого элемента берём значения, не включая его, по умолчанию равно длине списка.

step – шаг, с которым берём элементы, по умолчанию равен 1.

Дефолтные аргументы можно пропускать.

```
In [30]: #Взять с определенного символа и до конца
mystring[7:]
Out[30]: 'world'
In [31]: #Взять с начала и до определенного символа (невключительно)
mystring[:5]
Out[31]: 'Hello'
In [32]: #Берем все с шагом 1
mystring[::1]
Out[32]: 'Hello, world'
In [33]: mystring[::2]
Out[33]: 'Hlo ol'
In [34]: #Можно напечатать строку в обратном порядке
mystring[::-1]
Out[34]: 'dlrow ,olleH'
```

Строки неизменяемы.

```
In [35]: name = 'Sam'
```

Нельзя переопределить определенный символ в строке:

```
In [36]: name[0] = 'P'
-----
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11760\4091125550.py in <module>
----> 1 name[0] = 'P'
TypeError: 'str' object does not support item assignment
```

Конкатенация – объединение строк:

```
In [37]: s = 'Hello '
In [38]: s = s + 'my name is Jack'
In [39]: print(s)
Hello my name is Jack
```

Объекты в Python обычно имеют встроенные методы. Эти методы являются функциями внутри объекта, которые могут выполнять действия или команды над самим объектом.

Мы вызываем методы точкой. Методы имеют вид:

объект.метод(параметры)

Вот несколько примеров встроенных методов в строках:

```
In [40]: #Верхний регистр
s.upper()
Out[40]: 'HELLO MY NAME IS JACK'
In [41]: #Нижний регистр
s.lower()
Out[41]: 'hello my name is jack'
In [42]: #Разделить строку пробелом (по умолчанию)
s.split()
Out[42]: ['Hello', 'my', 'name', 'is', 'Jack']
```

Довольно часто возникают ситуации, когда нужно сделать строку, подставив в неё некоторые данные, полученные в процессе выполнения программы (пользовательский ввод, данные из файлов и т. д.). Форматирование строк позволяет выполнять множество подстановок в строке за единственный шаг. Оно не является строго обязательным, но может быть удобным.

Проще всего показать это на примере:

```
In [43]: name = 'Jose'
In [44]: print('Hello, my name is {}'.format(name))
Hello, my name is Jose
In [45]: print(f'Hello, my name is {name}')
Hello, my name is Jose
```

Lists Списки

Списки – это упорядоченные последовательности, которые могут содержать объекты различных типов. Списки строятся с помощью квадратных скобок [] и запятых, разделяющих каждый элемент в списке.

```
In [46]: my_list = [12, 'String', 10.05, 'w']
In [47]: len(my_list)
Out[47]: 4
```

Значения (элементы) из списка можно достать с помощью индексирования, также можно брать срезы списка. Работают так же, как и в строках.

```
In [48]: my_list[1]
Out[48]: 'String'
In [49]: my_list[::-1]
Out[49]: ['w', 10.05, 'String', 12]
```

Мы также можем использовать + для объединения списков, как мы это делали для строк.

```
In [50]: my_list + ['new item']
Out[50]: [12, 'String', 10.05, 'w', 'new item']
```

Примечание. На самом деле это не меняет исходный список!

```
In [51]: my_list
Out[51]: [12, 'String', 10.05, 'w']
```

Если вы знакомы с другим языком программирования, вы можете начать проводить параллели между массивами в другом языке и списками в Python. Однако списки в Python, как правило, более гибкие, чем массивы в других языках, по двум веским причинам: у них нет фиксированного размера (это означает, что нам не нужно указывать, насколько большим будет список), и у них нет ограничений фиксированного типа. (как мы видели выше) [20].

Давайте продолжим и рассмотрим еще несколько специальных методов для списков:

```
In [52]: list1 = [1,2,3,4]
```

Используйте метод *append*, чтобы добавить элемент в конец списка:

```
In [53]: list1.append('five')
In [54]: list1
Out[54]: [1, 2, 3, 4, 'five']
```

Используйте *pop*, чтобы «вытолкнуть» элемент из списка. По умолчанию *pop* удаляет последний элемент, но вы также можете указать, какой индекс элемента, который следует удалить. Давайте посмотрим пример:

```
In [55]: list1.pop()
Out[55]: 1
In [56]: item = list1.pop()
In [57]: item
Out[57]: 'five'
In [58]: list1
Out[58]: [2, 3, 4]
```

Мы можем использовать метод сортировки и метод, разворачивающий список:

```
In [59]: new_list = ['b', 'x', 'a', 'w', 'i']
In [60]: new_list.sort()
```

Этот метод ничего не возвращает.

```
In [61]: new_list
Out[61]: ['a', 'b', 'i', 'w', 'x']
In [62]: new_list.reverse()
In [63]: new_list
Out[63]: ['x', 'w', 'i', 'b', 'a']
```

Отличительной особенностью структур данных Python является то, что они поддерживают вложенность. Это означает, что мы можем иметь структуры данных внутри структур данных. Например, список внутри списка.

```
In [64]: lst1 = [1,2,3]
lst2 = [4,5,6]
lst3 = [7,8,9]
matrix = [lst1, lst2, lst3]
In [65]: matrix
Out[65]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
In [66]: matrix[0]
Out[66]: [1, 2, 3]
In [67]: matrix[0][1]
Out[67]: 2
```


Dictionaries Словари

Словари – это набор объектов, которые хранятся по ключу, в отличие от последовательности, в которой объекты хранятся по их относительному положению. Это важное различие, поскольку словари не сохраняют порядок.

Так называемый ассоциативный тип данных – в нём каждый элемент является парой ключ-значение. Для создания нужно указать элементы внутри фигурных скобок. Синтаксис элементов в словаре – *{key: value}*

```
In [68]: my_dict = {'key1':'value1','key2':'value2'}
In [69]: my_dict['key1']
Out[69]: 'value1'
```

Важно отметить, что словари очень гибки в отношении типов данных.

```
In [70]: my_dict =
{'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
In [71]: my_dict['key2']
Out[71]: [12, 23, 33]
In [72]: my_dict['key2'][1]
Out[72]: 23
```

Есть несколько методов, которые мы можем вызывать для словаря. Давайте кратко познакомимся с некоторыми из них:

```
In [73]: d = {'key1':1,'key2':2,'key3':3}
In [74]: #Метод для возврата списка всех ключей
d.keys()
Out[74]: dict_keys(['key1', 'key2', 'key3'])
In [75]: #Метод для возврата списка всех значений
d.values()
Out[75]: dict_values([1, 2, 3])
In [76]: #Метод для возврата списка кортежей всех элементов
d.items()
Out[76]: dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```

Tuples Кортежи

В Python кортежи очень похожи на списки, однако, в отличие от списков, они неизменяемы. Вы могли бы использовать кортежи для представления вещей, которые не должны изменяться, таких как дни недели или даты в календаре.

При построении кортежа используйте `()` с элементами, разделенными запятыми.

```
In [77]: t = (1,2,3)
In [78]: len(t)
Out[78]: 3
In [79]: t = ('one',2)
In [80]: t[-1]
Out[80]: 2
```

У кортежей есть встроенные методы, но их не так много, как у списков. Рассмотрим два из них:

```
In [81]: t.index('one')
Out[81]: 0
In [82]: t.count('one')
Out[82]: 1
```

Sets Множества

Множества – это неупорядоченные наборы уникальных элементов. Мы можем создать их с помощью функции `set()`.

```
In [83]: myset = set()
In [84]: myset.add(1)
In [85]: myset
Out[85]: {1}
In [86]: myset.add(2)
In [87]: myset
Out[87]: {1, 2}
In [88]: myset.add(1)
In [89]: myset
Out[89]: {1, 2}
```

Обратите внимание, что он не поместит туда еще одну единицу. Это потому, что множество содержит только уникальные элементы. Мы можем преобразовать список с несколькими повторяющимися элементами в множество, чтобы получить уникальные элементы. Например:

```
In [90]: list1 = [1,1,2,2,3,4,5,6,1,1]
In [91]: set(list1)
Out[91]: {1, 2, 3, 4, 5, 6}
```

Booleans Логические значения

Логические значения – это операторы, в которые передаются истинные или ложные утверждения (*True* и *False*, которые в основном представляют собой целые числа 1 и 0).

```
In [92]: a = True
In [93]: a
Out[93]: True
In [94]: 1 > 2
Out[94]: False
```

Практические задания

1. Напишите уравнение, в котором используются умножение, деление, показатель степени, сложение и вычитание, равное 160,25.

2. Дана строка `s = 'hello'`.

1) Выведите букву 'e', используя индексацию.

2) Переверните строку, используя нарезку.

3) Для строки укажите два способа получения буквы 'o' с помощью индексации.

3. Напишите программу, которая выводит: "Mother's favorite book "Gone with the Wind""

4. Создайте переменную *name* с любым именем. Выведите две строки:

```
How are you, <name>?
I'm fine, thanks.
```

Используйте один `print()`.

5. Переназначьте 'hello' во вложенном списке `list1 = [1,2,[3,4,'hello']]`, чтобы вместо этого было 'goodbye'.

6. Отсортируйте список `list2 = [5,3,4,6,1]`

7. Используя ключи и индексацию, возьмите 'hello' из следующих словарей:

1. `d = {'simple_key': 'hello'}`
2. `d = {'k1': {'k2': 'hello'}}`
3. `d = {'k1': [{'nest_key': ['this is deep', ['hello']]]}`
4. `d = {'k1': [1,2,{'k2': ['this is tricky',{'tough':[1,2,['hello']]}]}`

8. Используйте множество, чтобы найти уникальные значения в списке `list3 = [1,2,2,33,4,4,11,22,3,3,2]`.

9. Напишите программу, которая берет исходное количество евро, записанное в переменную `euros_count`, переводит евро в доллары и выводит на экран. Затем полученное значение переводит в рубли и выводит на новой строке.

10. Создайте переменную `student` и присвойте ей строку, содержащую фамилию, имя и отчество, разделенные «;». Преобразуйте ее в список с помощью метода `split()` и выведите его на экран. Преобразуйте полученный список сначала в кортеж, а потом в множество. Выведите их на экран.

Контрольные вопросы

1. Python – интерпретируемый язык или компилируемый?
2. Какие есть меняющиеся и постоянные типы данных?
3. Какие есть типы данных и какая разница между `list` и `tuple`, зачем они?
4. В чем заключается сложность доступа к элементам `dict`?
5. Каковы преимущества использования Python?
6. Что такое PEP 8?
7. Как добавить новое значение в объект списка?
8. Что такое отрицательный индекс?
9. Что такое срез?
10. Как убрать из списка дубликат элемента?

Лабораторная работа №3. Условные операторы и циклы

Цель:

Изучить базовые конструкции Python.

Теоретический материал

Взгляните на эти два примера:

Version 1 (Другие языки)	Version 2 (Python)
<code>if (a>b) {a = 2; b = 4;}</code>	<code>if a>b: a = 2 b = 4</code>

Python избавляется от `()` и `{}`, добавляя два основных фактора: двоеточие и отступ. Оператор заканчивается двоеточием, а отступ используется для описания того, что происходит в случае выполнения условия.

Еще одним важным отличием является отсутствие точки с запятой. Точка с запятой используется для обозначения окончания операторов во многих других языках, но в Python конец строки совпадает с концом оператора.

Обратите внимание, как сильно Python зависит от отступов и пробелов в коде. Это означает, что читабельность кода является основной частью дизайна языка Python.

If, elif, else

Очень часто у вас есть большой фрагмент кода, и вы хотите, чтобы определенный код выполнялся только тогда, когда условие выполнено. Чтобы управлять потоком логики используются ключевые слова *if*, *elif*, *else*.

Синтаксис для *if* оператора:

```
if some_condition:  
    # execute some code  
elif some_other_condition:  
    # do smth different  
else:  
    # do smth else
```

Примеры:

```
In [1]:  
x = False  
if x:  
    print('x was True!')  
else:  
    print('I will be printed in any case where x is not true')  
Out [1]:  
I will be printed in any case where x is not true  
  
In [2]:  
person = 'George'
```

```
if person == 'Sammy':  
    print('Welcome Sammy!')  
elif person == 'George':  
    print('Welcome George!')  
else:  
    print("Welcome, what's your name?")  
Out [2]:  
Welcome George!
```

Цикл for

Цикл *for* действует в Python как итератор; он перебирает элементы в итерируемых объектах. Объекты, над которыми можно выполнять итерацию включают строки, списки, кортежи и словари.

Синтаксис:

```
for item in object:  
    statements to do stuff
```

Примеры:

```
In [3]:  
list1 = [1,2,3,4,5,6,7,8,9,10]  
In [4]:  
for num in list1:  
    print(num)  
Out[4]:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Теперь давайте добавим *if* оператор для проверки четных чисел.

```
In [5]:  
for num in list1:  
    if num % 2 == 0:  
        print(num)  
Out[5]:  
2  
4  
6  
8  
10  
In [6]:  
list_sum = 0  
for num in list1:  
    list_sum = list_sum + num  
print(list_sum)  
Out[6]:  
55
```

Кортежи обладают особым свойством, когда речь идет о циклах *for*. Во время цикла *for* мы распакуем кортеж внутри последовательности и сможем получить доступ к отдельным элементам внутри этого кортежа.

```
In [7]:  
list2 = [(2,4), (6,8), (10,12)]
```

```
In [8]:  
for tup in list2:  
    print(tup)
```

```
Out[8]:  
(2, 4)  
(6, 8)  
(10, 12)
```

```
In [9]: #Теперь с распаковкой!  
for (t1,t2) in list2:  
    print(t1)
```

```
Out[9]:  
2  
6  
10
```

Списковое включение (генератор списков) – одна из особенностей Python. Этот механизм позволяет кратко записать создание нового списка, образованного фильтрацией элементов коллекции с одновременным преобразованием элементов, прошедших через фильтр [13]. Основная синтаксическая форма такова:

```
[expr for val in collection if condition]
```

Это эквивалентно следующему циклу *for*:

```
result = []  
for val in collection:  
    if condition:  
        result.append(expr)
```

Условие фильтрации можно опустить, оставив только выражение.

```
In [10]: #Захватить каждую букву в цикле  
lst = [x for x in 'word']  
In [11]: lst  
Out[11]: ['w', 'o', 'r', 'd']  
In [12]: #Возведение в степень и преобразование в список  
lst = [x**2 for x in range(0,11)]  
In [13]: lst  
Out[13]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Можно добавить *if* оператор:

```
In [14]: #Проверяем наличие четных чисел в диапазоне  
lst = [x for x in range(11) if x % 2 == 0]  
In [15]: lst  
Out[15]: [0, 2, 4, 6, 8, 10]
```

Цикл While

Оператор *while* в Python является одним из наиболее распространенных способов выполнения итерации. Оператор *while* будет многократно выполнять один оператор или группу операторов, пока условие истинно.

Синтаксис:

```
while test:
    code statements
else:
    final code statements
```

Пример:

```
In [16]:
x = 0
while x < 5:
    print('x is currently: ',x)
    print(' x is still less than 5, adding 1 to x')
    x+=1
else:
    print('All Done!')
```

```
Out[16]:
x is currently: 0
x is still less than 5, adding 1 to x
x is currently: 1
x is still less than 5, adding 1 to x
x is currently: 2
x is still less than 5, adding 1 to x
x is currently: 3
x is still less than 5, adding 1 to x
x is currently: 4
x is still less than 5, adding 1 to x
All Done!
```

Мы можем использовать операторы *break*, *continue* и *pass* в наших циклах, чтобы добавить дополнительную функциональность для различных случаев. Три утверждения определяются:

break – выходит из текущего цикла;

continue – перепрыгивает на следующую итерацию цикла и не прекращает его исполнение;

pass – вообще ничего не делает.

С учетом утверждений *break* и *continue* общий формат *while* цикла выглядит следующим образом:

```
while test:
    code statement
    if test:
        break
    if test:
        continue
    else:
```

Пример:

```
In [17]:
x = 0
while x < 5:
    print('x is currently: ',x)
    print('x is still less than 5, adding 1 to x')
    x+=1
    if x==3:
        print('Breaking because x==3')
        break
    else:
        print('continuing...')
        continue
```

Полезные операторы

range

Функция позволяет быстро и удобно создать список целых чисел. Есть 3 параметра, которые вы можете передать: начало, конец и размер шага.

```
In [18]: range(0,11)
Out[18]: range(0, 11)
```

Обратите внимание, что это функция-генератор – особый тип функции, которая будет генерировать информацию и не сохранит ее в памяти [13]. Поэтому, чтобы фактически получить из нее список, нам нужно привести его к списку с помощью *list()*.

```
In [19]: list(range(0,11))
Out[19]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Обратите внимание, что 11 не включено.

```
In [20]: # Третий параметр - размер шага
list(range(0,11,2))
Out[20]: [0, 2, 4, 6, 8, 10]
In [21]: # list(range(0,101,10))
Out[21]: [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

enumerate

enumerate – очень полезная функция для циклов *for*. Представим следующую ситуацию:

```
In [22]:
index_count = 0
for letter in 'abcde':
    print("At index {} the letter is {}".format(index_count,letter))
    index_count += 1
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```


Отслеживание того, сколько циклов вы прошли, настолько распространено, что было создано перечисление, поэтому вам не нужно беспокоиться о создании и обновлении этой переменной *index_count* или *loop_count*.

Если *range()* позволяет получить только индексы элементов списка, то *enumerate()* – сразу индекс элемента и его значение.

```
In [23]:
for i, letter in enumerate('abcde'):
    print("At index {} the letter is {}".format(i, letter))
```

```
At index 0 the letter is a
At index 1 the letter is b
At index 2 the letter is c
At index 3 the letter is d
At index 4 the letter is e
```

[min, max](#)

С помощью этих функций можно быстро проверить минимум или максимум списка.

```
In [24]: mylist = [10, 20, 30, 40, 100]
In [25]: min(mylist)
Out[25]: 10
In [26]: max(mylist)
Out[26]: 100
```

[input](#)

```
In [27]:
input('Enter Something into this box: ')
Enter Something into this box: OK
Out[27]: 'OK'
```

Практические задания

1. Пройдитесь по приведенной ниже строке и, если длина слова четная, выведите «четный!»

```
st = 'Выведите каждое слово в этом предложении, имеющее четное количество букв'
```

2. Используйте списковое включение, чтобы создать список всех чисел от 1 до 50, которые делятся на 3.

3. Напишите программу, которая печатает целые числа от 1 до 100. Но для чисел, кратных трем, вместо числа печатайте «Fizz», а для кратных пяти – «Buzz». Для чисел, кратных как трем, так и пяти, выведите «FizzBuzz».

4. Напишите программу, которая выбирает случайное целое число от 1 до 100 и предлагает игрокам угадать число. Правила таковы:

1) Если предположение игрока меньше 1 или больше 100, вывести «Вне границы».

2) На первом ходу игрока, если его предположение:

- окажется в пределах 10 от загаданного числа, вернуть "Тепло!";
- отличается от загаданного числа более чем на 10, вернуть "Холодно!".

3) На всех последующих ходах, если догадка:

- ближе к числу, чем предыдущее предположение, вернуть "Теплее!";
- дальше от числа, чем предыдущее предположение, вернуть «Холоднее!».

4) Когда догадка игрока сравнивается с числом, скажите ему, что он угадал правильно, и за сколько попыток он угадал.

Примечание: `random.randint(a,b)` возвращает случайное целое число в диапазоне $[a, b]$, включая обе конечные точки.

Для этого нужно импортировать модуль `random`:

```
import random
```

5. Разработайте программу, запрашивающую у пользователя букву латинского алфавита. Если введенная буква входит в следующий список (а, е, и, о или u), необходимо вывести сообщение о том, что эта буква гласная. Если была введена буква y, программа должна написать, что эта буква может быть как гласной, так и согласной. Во всех других случаях должно выводиться сообщение о том, что введена согласная буква.

6. Количество дней в месяце варьируется от 28 до 31. Очередная ваша программа должна запрашивать у пользователя название месяца и отображать количество дней в нем. Поскольку годы мы не учитываем, для февраля можно вывести сообщение о том, что этот месяц может состоять как из 28, так и из 29 дней, чтобы учесть фактор високосного года.

7. Напишите программу, запрашивающую у пользователя дату его рождения и выводящую на экран соответствующий знак зодиака.

8. Строка называется палиндромом, если она пишется одинаково в обоих направлениях. Например, палиндромами в английском языке являются слова «anna», «civic», «level», «hannah». Напишите программу, запрашивающую у пользователя строку и при помощи цикла определяющую, является ли она палиндромом.

Контрольные вопросы

1. Обязательны ли отступы в Python?

2. Какой тип цикла является циклом с постусловием?

3. Зачем нужны `break` и `continue`?
4. Каков смысл оператора `pass` в Python?
5. Что такое функция `enumerate` в Python?
6. В чем разница между тем, когда функция `range()` принимает один аргумент, два и три?
7. Что такое распаковка кортежа?
8. Расскажите про генераторы списков (`list comprehension`).
9. Как обрабатывать входные данные в Python?
10. Почему мы используем `enumerate()` при итерации последовательности?

Лабораторная работа №4. Методы и функции

Цель:

Изучить встроенные методы и способы создания функций в Python.

Теоретический материал

Методы – это, по сути, функции, встроенные в объекты. Позже в ходе курса мы узнаем, как создавать собственные объекты и методы с помощью объектно-ориентированного программирования (ООП) и классов.

Методы выполняют определенные действия над объектом, а также могут принимать аргументы, как и функция. Методы имеют вид:

```
object.method(arg1, arg2, etc...)
```

Примеры:

```
In [1]:  
lst = [1,2,3,4,5]  
In [2]: # с помощью Tab можем быстро просмотреть все возможные методы  
lst.append(5)  
In [3]: # shift+tab посмотреть справку по методу  
lst.count(1)  
Out[3]: 1
```

Функция – это полезное устройство, которое группирует набор операторов, чтобы их можно было запускать более одного раза. Они также могут позволить нам указать параметры, которые могут служить входными данными для функций.

Функция позволит вам вызывать один и тот же блок кода без необходимости писать его несколько раз. Это, в свою очередь, позволит вам создавать более сложные сценарии Python [12].

Давайте посмотрим, как построить синтаксис функции в Python. Он имеет следующий вид:

```
def name_of_function(arg1, arg2):  
    """  
    This is where the function's Document String (docstring) goes.  
    When you call help() on your function it will be printed out.  
    """  
    # Do stuff here  
    # Return desired result
```

Мы начинаем с *def*, за которым следует имя функции. Старайтесь, чтобы имена были релевантными, например, *len()* – хорошее имя для функции *length()*. Также будьте осторожны с именами, не называйте функцию тем же именем, что и встроенную функцию в Python (например, *len*).

Далее идет пара круглых скобок с количеством аргументов, разделенных запятой. Эти аргументы являются входными данными для вашей функции. Вы сможете использовать эти входные данные в своей функции и ссылаться на них. После этого ставится двоеточие.

Теперь важный шаг, вы должны сделать отступ, чтобы правильно начать код внутри вашей функции.

Далее идет строка документации, где вы пишете основное описание функции. Используя Jupyter и Jupyter Notebook, вы сможете прочитать эти строки документации, нажав Shift+Tab после имени функции. Строки документации не нужны для простых функций, но их рекомендуется использовать, чтобы вы или другие люди могли легко понять код, который вы пишете [20].

После всего этого вы начинаете писать код, который хотите выполнить.

```
In [4]: # Простой пример функции
def say_hello():
    print('hello')
In [5]: # Вызов функции с помощью ()
say_hello()
Out[5]:
hello
In [6]: # Принятие параметров (аргументов)
def greeting(name):
    print(f'Hello {name}')
In [7]:
greeting("Nina")
Out[7]:
Hello Nina
```

До сих пор мы видели только использование *print()*, но если мы действительно хотим сохранить результирующую переменную, нам нужно использовать ключевое слово *return*.

return позволяет функции возвращать результат, который затем может быть сохранен как переменная или использован любым удобным для пользователя способом.

```
In [8]:
def add_num(num1,num2):
    return num1+num2
In [9]: add_num(4,5)
Out[9]: 9
In [10]: # Также можно сохранить в переменную
result = add_num(4,5)
In [11]:
print(result)
Out[11]: 9
```

Напишем функцию, которая будет возвращать все четные числа в списке, иначе вернем пустой список.

```

In [18]:
def check_even_list(num_list):
    even_numbers = []
    # Пройдемся по каждому числу
    for number in num_list:
        if number % 2 == 0:
            even_numbers.append(number)
        else:
            pass
    return even_numbers
In [19]:
check_even_list([1,2,3,4,5,6])
Out[19]:
[2, 4, 6]
In [20]:
check_even_list([1,3,5])
Out[20]:
[]

```

Функции часто используют результаты других функций, давайте рассмотрим простой пример игры в угадайку. В списке будет 3 позиции, одна из которых – буква «О», функция будет перемешивать список, другая будет принимать предположения игрока и, наконец, третья проверит, верны ли они.

Как перетасовать список в Python

```

In [21]: example = [1,2,3,4,5]
In [22]: from random import shuffle
In [23]: shuffle(example)
In [24]: example
Out[24]: [5, 3, 4, 2, 1]

```

Хорошо, давайте создадим нашу простую игру

```

In [25]:
mylist = [' ', 'O', ' ']
In [26]:
def shuffle_list(mylist):
    shuffle(mylist)
    return mylist
In [27]: mylist
Out[27]: [' ', 'O', ' ']
In [28]: shuffle_list(mylist)
Out[28]: ['O', ' ', ' ']
In [29]:
def player_guess():
    guess = ''
    while guess not in ['0','1','2']:
        guess = input("Pick a number: 0, 1, or 2: ")
    return int(guess)
In [30]:
player_guess()
Pick a number: 0, 1, or 2: 1
Out[30]: 1

```

Теперь проверим предположение пользователя.

```
In [31]:
def check_guess(mylist, guess):
    if mylist[guess] == 'O':
        print('Correct Guess!')
    else:
        print('Wrong! Better luck next time')
    print(mylist)
```

Создадим небольшую логику для запуска всех функций. Обратите внимание, как они взаимодействуют друг с другом!

```
In [32]: # Инициализируем список
mylist = [' ', 'O', ' ']
# Перемешиваем его
mixedup_list = shuffle_list(mylist)
# Получаем предположение игрока
guess = player_guess()
# Проверяем его предположение
check_guess(mixedup_list, guess)
Pick a number: 0, 1, or 2: 2
Correct Guess!
```

Лямбда-выражение

Одним из наиболее полезных (и сбивающих с толку) инструментов Python является лямбда-выражение. Лямбда-выражения позволяют нам создавать «анонимные» функции. В основном это означает, что мы можем быстро создавать специальные функции без необходимости правильно определять функцию с помощью *def* [13].

Объекты-функции, возвращаемые выполнением лямбда-выражений, работают точно так же, как объекты, созданные и назначенные *def*. Есть ключевое отличие, которое делает лямбду полезной в специализированных ролях: тело лямбды – это одно выражение, а не блок операторов.

lambda предназначена для кодирования простых функций, а *def* справляется с более крупными задачами.

Давайте медленно разберем лямбда-выражение, разобрав функцию:

```
In [33]:
def square(num):
    result = num**2
    return result
In [34]: square(2)
Out[34]: 4
```

Мы могли бы упростить это:

```
In [35]:
def square(num):
    return num**2
In [36]: square(2)
Out[36]: 4
```

Мы могли бы даже написать все это в одной строке.

```
In [37]: def square(num): return num**2
In [38]: square(2)
Out[38]: 4
```

Это форма функции, которую лямбда-выражение намеревается воспроизвести. Тогда лямбда-выражение может быть записано как:

```
In [39]: square = lambda num: num ** 2
In [40]: square(2)
Out[40]: 4
```

Практические задания

1. Напишите функцию, которая возвращает меньшее из двух заданных чисел, если оба числа четные, и большее, если одно или оба числа нечетные:

```
lesser_of_two_evens(2,4) --> 2
lesser_of_two_evens(2,5) --> 5
```

2. Напишите функцию, которая принимает строку из двух слов и возвращает True, если оба слова начинаются с одной и той же буквы:

```
animal_crackers('Levelheaded Llama') --> True
animal_crackers('Crazy Kangaroo') --> False
```

3. Верните предложение со словами в обратном порядке:

```
master_yoda('I am home') --> 'home am I'
master_yoda('We are ready') --> 'ready are We'
```

4. Учитывая список целых чисел, верните True, если в массиве есть тройки, стоящие рядом.

```
has_33([1, 3, 3]) → True
has_33([1, 3, 1, 3]) → False
has_33([3, 1, 3]) → False
```

5. Даны три целых числа от 1 до 11, если их сумма меньше или равна 21, вернуть их сумму. Если их сумма превышает 21, и есть 11, уменьшите общую сумму на 10. Наконец, если сумма превышает 21, верните 'BUST'.

```
blackjack(5,6,7) --> 18
blackjack(9,9,9) --> 'BUST'
blackjack(9,9,11) --> 19
```

6. Напишите функцию, которая возвращает количество простых чисел, которые существуют до заданного числа включительно.

```
count_primes(100) --> 25
```

7. Напишите функцию, которая принимает строку и вычисляет количество прописных и строчных букв.

Sample String : 'Hello Mr. Rogers, how are you this fine Tuesday?'
Expected Output :
No. of Upper case characters : 4
No. of Lower case Characters : 33

8. Напишите функцию, чтобы проверить, является ли строка панграммой или нет.
(Предположим, что переданная строка не имеет пунктуации).

Примечание. Панграммы – это слова или предложения, содержащие каждую букву алфавита хотя бы один раз.

Например : "The quick brown fox jumps over the lazy dog"

9. Реализуйте функцию `get_hidden_card()`, которая принимает на вход номер кредитки (состоящий из 16 цифр) в виде строки и возвращает его скрытую версию, которая может использоваться на сайте для отображения. Если исходная карта имела номер 2034399002125581, то скрытая версия выглядит так ****5581. Другими словами, функция заменяет первые 12 символов, на звездочки. Количество звездочек регулируется вторым необязательным параметром. Значение по умолчанию – 4.

Кредитка передается внутрь как строка

```
get_hidden_card('2034399002121100', 1) # "*1100"  
get_hidden_card('1234567812345678', 2) # "***5678"  
get_hidden_card('1234567812345678', 3) # "****5678"  
get_hidden_card('1234567812345678') # "****5678"
```

Контрольные вопросы

1. В чем смысл написания функций?
2. В какой момент Python создает функцию?
3. Когда выполняется код, вложенный внутри оператора определения функции?
4. Что такое лямбда-функция?
5. Почему лямбда-формы в Python не имеют операторов?
6. Является ли функция допустимой, если она не имеет оператора return?
7. Что такое строка документации (docstring)?
8. Что нужно сделать, чтобы функция возвратила значение?
9. В чем отличие между параметрами и аргументами функции?
10. Что делает метод `split()` ?

Лабораторная работа №5. Объектно-ориентированное программирование

Цель:

Изучить принципы объектно-ориентированного подхода.

Теоретический материал

В Python все является объектом. Помните из предыдущих занятий, что мы можем использовать `type()` для проверки типа объекта:

```
In [1]:
print(type(1))
print(type([]))
print(type(()))
print(type({}))

<class 'int'>
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

Итак, мы знаем, что все эти сущности являются объектами, так как же мы можем создавать свои собственные типы объектов? Определяемые пользователем объекты создаются с помощью ключевого слова `class`. Класс – это шаблон, определяющий характер будущего объекта. Из классов мы можем создавать экземпляры. Экземпляр – это конкретный объект, созданный из определенного класса.

Давайте посмотрим, как мы можем использовать `class`:

```
In [2]: # Создадим новый объект типа Sample
class Sample:
    pass
# Экземпляр класса Sample
x = Sample()
print(type(x))

<class '__main__.Sample'>
```

По соглашению мы даем классам имена, начинающиеся с заглавной буквы. Внутри класса у нас сейчас просто пропуск. Но мы можем определить атрибуты и методы класса.

Атрибут – это характеристика объекта. Метод – это операция, которую мы можем выполнять с объектом.

Например, мы можем создать класс с именем `Dog`. Атрибутом собаки может быть ее порода или имя, а метод собаки может быть определен методом `bark()`, который возвращает звук.

Давайте лучше разберемся с атрибутами на примере.

Синтаксис создания атрибута:

```
self.attribute = something
```

Существует специальный метод, который называется:

```
__init__()
```

Этот метод используется для инициализации атрибутов объекта.

```
In [3]:
class Dog:
    def __init__(self, breed):
        self.breed = breed
sam = Dog(breed='Lab')
frank = Dog(breed='Huskie')
```

Давайте разберем. Специальный метод `__init__()` вызывается автоматически сразу после создания объекта:

```
def __init__(self, breed):
```

Каждый атрибут в определении класса начинается со ссылки на экземпляр объекта. По соглашению он называется *self*. Порода – это аргумент. Значение передается во время создания экземпляра класса.

```
self.breed = breed
```

Теперь мы создали два экземпляра класса `Dog`. с двумя типами породы. Мы можем получить доступ к этим атрибутам следующим образом:

```
In [4]: sam.breed
Out[4]: 'Lab'
In [5]: frank.breed
Out[5]: 'Huskie'
```

Обратите внимание, что у нас нет круглых скобок после породы; это потому, что это атрибут и не принимает никаких аргументов.

В Python также есть атрибуты объекта класса. Эти атрибуты объекта класса одинаковы для любого экземпляра класса. Например, мы могли бы создать атрибут вид для класса `Dog`. Собаки, независимо от их породы, имени или других признаков, всегда будут млекопитающими. Мы применяем эту логику следующим образом:

```
In [6]:
class Dog:
    # Class Object Attribute
    species = 'mammal'
    def __init__(self, breed, name):
        self.breed = breed
        self.name = name
In [7]: sam = Dog('Lab', 'Sam')
In [8]: sam.name
```

```
Out[8]: 'Sam'
```

Обратите внимание, что атрибут объекта класса определен вне каких-либо методов в классе. Также по соглашению мы помещаем их первыми перед `__init__()`.

```
In [9]: sam.species  
Out[9]: 'mammal'
```

Методы – это функции, определенные внутри тела класса. Они используются для выполнения операций с атрибутами наших объектов. Методы являются ключевой концепцией парадигмы ООП. Они необходимы для разделения обязанностей в программировании, особенно в больших приложениях [14].

В основном вы можете думать о методах как о функциях, воздействующих на объект, которые принимают во внимание сам объект через его собственный аргумент.

Давайте рассмотрим пример создания класса *Circle*:

```
In [10]:  
class Circle:  
    pi = 3.14  
    # Окружность задается с радиусом (по умолчанию = 1)  
    def __init__(self, radius=1):  
        self.radius = radius  
        self.area = radius * radius * Circle.pi  
    # Метод для установки радиуса  
    def setRadius(self, new_radius):  
        self.radius = new_radius  
        self.area = new_radius * new_radius * self.pi  
    # Метод для вычисления длины окружности  
    def getCircumference(self):  
        return self.radius * self.pi * 2  
  
c = Circle()  
print('Radius is: ',c.radius)  
print('Area is: ',c.area)  
print('Circumference is: ',c.getCircumference())  
  
Out [10]:  
Radius is: 1  
Area is: 3.14  
Circumference is: 6.28
```

В приведенном выше методе `__init__()` для вычисления атрибута площади нам пришлось вызвать *Circle.pi*. Это связано с тем, что у объекта еще нет собственного атрибута *pi*, поэтому вместо этого мы вызываем атрибут объекта класса *pi*. Однако в методе *setRadius* мы будем работать с существующим объектом *Circle*, у которого есть собственный атрибут *pi*. Здесь мы можем использовать либо *Circle.pi*, либо *self.pi*.

Теперь давайте изменим радиус и посмотрим, как это повлияет на наш объект *Circle*:

```
In [11]:
c.setRadius(2)
print('Radius is: ',c.radius)
print('Area is: ',c.area)
print('Circumference is: ',c.getCircumference())
```

```
Out [11]:
Radius is: 2
Area is: 12.56
Circumference is: 12.56
```

Наследование

Наследование – это способ формирования новых классов с использованием уже определенных классов. Важными преимуществами наследования являются повторное использование кода и снижение сложности программы. Производные классы (потомки) переопределяют или расширяют функциональные возможности базовых классов (предков) [14].

Давайте посмотрим на пример:

```
In [12]:
class Animal:
    def __init__(self):
        print("Animal created")
    def whoAmI(self):
        print("Animal")
    def eat(self):
        print("Eating")
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print("Dog created")
    def whoAmI(self):
        print("Dog")
    def bark(self):
        print("Woof!")
```

```
In [13]: d = Dog()
Animal created
Dog created
```

```
In [14]: d.whoAmI()
Dog
In [15]: d.eat()
Eating
In [16]: d.bark()
Woof!
```

В этом примере у нас есть два класса: *Animal* и *Dog*. *Animal* – это базовый класс, *Dog* – производный класс. Производный класс наследует функциональность базового класса – метод *eat()*, изменяет существующее поведение базового класса – метод *whoAmI()*, и расширяет функциональность базового класса, определяя новый метод *bark()*.

Полиморфизм

Мы узнали, что хотя функции могут принимать разные аргументы, методы принадлежат объектам, над которыми они действуют. В Python полиморфизм показывает, как разные классы объектов могут использовать одно и то же имя метода, и эти методы могут вызываться из одного и того же места, даже если могут быть переданы различные объекты [14].

```
In [17]:
class Dog:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return self.name+' says Woof!'
class Cat:
    def __init__(self, name):
        self.name = name
    def speak(self):
        return self.name+' says Meow!'

niko = Dog('Niko')
felix = Cat('Felix')
print(niko.speak())
print(felix.speak())

Niko says Woof!
Felix says Meow!
```

Здесь у нас есть класс Dog и класс Cat, и у каждого есть *speak()* метод. При вызове *speak()* каждый объект возвращает результат, уникальный для объекта.

Существует несколько различных способов демонстрации полиморфизма.

```
In [18]:
for pet in [niko, felix]:
    print(pet.speak())
```

```
Niko says Woof!
Felix says Meow!
```

```
In [19]:
def pet_speak(pet):
    print(pet.speak())
pet_speak(niko)
pet_speak(felix)
```

```
Niko says Woof!
Felix says Meow!
```

В обоих случаях мы могли передавать разные типы объектов и получать специфичные для объекта результаты с помощью одного и того же механизма.

Реальные примеры полиморфизма включают:

- открытие разных типов файлов – для отображения файлов Word, pdf и Excel необходимы разные инструменты;
- добавление разных объектов – оператор + выполняет арифметику и конкатенацию.

Специальные методы

Классы в Python могут реализовывать определенные операции со специальными именами методов [20]. Эти методы на самом деле вызываются не напрямую, а с помощью синтаксиса языка Python. Например, давайте создадим класс Book:

```
In [21]:
class Book:
    def __init__(self, title, author, pages):
        print("A book is created")
        self.title = title
        self.author = author
        self.pages = pages
    def __str__(self):
        return "Title: %s, author: %s, pages: %s" %(self.title,
self.author, self.pages)
    def __len__(self):
        return self.pages
    def __del__(self):
        print("A book is destroyed")
```

```
In [22]:
book = Book("Python Rocks!", "Jose Portilla", 159)
#Special Methods
print(book)
print(len(book))
del book

A book is created
Title: Python Rocks!, author: Jose Portilla, pages: 159
159
A book is destroyed
```

`__init__()`, `__str__()`, `__len__()`, `__del__()` эти специальные методы определяются использованием символов подчеркивания. Они позволяют нам использовать специальные функции Python для объектов, созданных с помощью нашего класса.

Практические задания

1. Заполните методы класса Line, чтобы они принимали координаты в виде пары кортежей и возвращали наклон и расстояние линии.

```
class Line:
    def __init__(self, coor1, coor2):
        pass
    def distance(self):
        pass
    def slope(self):
        Pass
```

2. Заполните класс:

```
class Cylinder:
    def __init__(self,height=1,radius=1):
        pass
    def volume(self):
        pass
    def surface_area(self):
        pass
```

3. Для этой задачи создайте класс банковского счета с двумя атрибутами: владелец (owner) и остаток средств (balance). И также два метода: депозит (deposit) и снять средства (withdraw)

В качестве дополнительного требования вывод средств не может превышать доступный баланс.

Создайте экземпляр своего класса, сделайте несколько депозитов и снятий средств и проверьте, не может ли счет быть перерасходованным.

4. Product Inventory Project – создайте приложение, которое управляет инвентаризацией продуктов. Создайте класс продукта, у которого есть цена, идентификатор и количество. Затем создайте класс запасов, который отслеживает различные продукты и может суммировать стоимость запасов.

5. Система бронирования авиакомпаний/отелей. Создайте систему бронирования, которая бронирует места в самолетах или номера в отелях. Он взимает различные тарифы за определенные участки самолета или отеля. Например, первый класс будет стоить больше, чем эконом. В гостиничных номерах есть пентхаусы, которые стоят дороже. Следите за тем, когда комнаты будут доступны и могут быть запланированы.

6. Менеджер компании – создайте иерархию классов – абстрактный класс Employee и подклассы HourlyEmployee, SalariedEmployee, Manager и Executive. Зарплата у всех рассчитывается по-разному, немного об этом узнайте. После того, как вы установили иерархию сотрудников, создайте класс Company, который позволит вам управлять сотрудниками. Вы должны иметь возможность нанимать, увольнять и повышать сотрудников.

7. Планировщик пациентов/врачей. Создайте класс пациента и класс врача. Наймите врача, который может обслуживать нескольких пациентов, и настройте программу планирования, согласно которой врач может обслуживать только 16 пациентов в течение 8-часового рабочего дня.

8. Создатель и менеджер рецептов. Создайте класс рецептов с ингредиентами и поместите их в программу менеджера рецептов, которая организует их по категориям, таким как десерты, основные блюда, или по ингредиентам, таким как курица, говядина, супы, пироги и т. д.

9. Создатель семейного древа. Создайте класс под названием «Человек», у которого будет имя, когда он родился и когда (и если) умер. Разрешить пользователю создавать эти классы Person и помещать их в структуру генеалогического древа. Распечатайте древо на экран.

10. Создайте класс игральных карт, а далее класс колоды с этими картами. В колоде 52 карты, каждая из которых принадлежит одной из четырех мастей и имеет одно из 13 значений. Масти такие: пики (англ.: Spades), червы (англ.: Hearts), бубны (англ.: Diamonds) и трефы (англ.: Clubs). Значения такие: туз (англ.: Ace), 2, 3, 4, 5, 6, 7, 8, 9, 10, валет (англ.: Jack), дама (англ.: Queen) и король (англ.: King). Должна быть возможность тасовать колоду, извлекать и сдавать карты.

Контрольные вопросы

1. Что делает Python объектно-ориентированным?
2. Что относится к основным принципам ООП?
3. Какой принцип ООП описывает следующее предложение? Этот принцип является способностью использовать общий интерфейс для нескольких форм (типов данных).
4. Приведите примеры использования полиморфизма в Python.
5. Какой параметр обязательно принимает в себя метод экземпляра?
6. Что означает self в классе?
7. Как реализуется наследование классов в Python?
8. Что произойдет, если в классе определены два метода с одинаковыми именами и разными списками параметров?
9. Определите класс car с двумя атрибутами: color и speed. Затем создайте экземпляр и верните speed.
10. Перечислите специальные методы и что они делают.

Лабораторная работа №6. Обработка ошибок и исключений. Модули и пакеты. NumPy

Цель:

Научиться обрабатывать ошибки и исключения. Освоить создание собственных модулей и пакетов. Познакомиться с библиотекой NumPy.

Теоретический материал

Ошибки неизбежны. Например:

```
In [1]: print('Hello)
File "C:\Users\babas\AppData\Local\Temp\ipykernel_5604\1679058590.py",
line 1
print('Hello)
^
SyntaxError: EOL while scanning string literal
```

Обратите внимание, как мы получаем `SyntaxError` с дальнейшим описанием того, что это была EOL (ошибка конца строки) при сканировании строкового литерала. Этого достаточно, чтобы мы увидели, что забыли одну кавычку в конце строки. Понимание этих различных типов ошибок поможет вам намного быстрее отлаживать свой код [20].

Даже если инструкция или выражение синтаксически правильное, оно может вызвать ошибку при попытке его выполнения. Ошибки, обнаруженные во время выполнения, называются исключениями и не являются безусловно фатальными.

try and except

Основная терминология и синтаксис, используемые для обработки ошибок в Python, – это операторы *try* и *except*. Код, который может вызвать исключение, помещается в *try* блок, а обработка исключения затем реализуется в *except* блоке кода. Синтаксис следующий:

```
try:
    Блок кода, который будет выполняться (может привести к ошибке)
    ...
except ExceptionI:
    Если ExceptionI, тогда выполнять этот блок.
except ExceptionII:
    Если ExceptionII, тогда выполнять этот блок.
    ...
else:
    Если не было исключений, тогда выполнять этот блок.
```

Пример:

```
In [2]:
try:
    f = open('testfile','w')
    f.write('Попробуй написать это')
except IOError:
    # Это только проверит исключение IOError и выполнит вывод на экран
    сообщения
    print("Ошибка: Не удалось найти файл или прочитать данные")
else:
    print("Содержимое записано успешно")
    f.close()
```

Содержимое записано успешно

Теперь давайте посмотрим, что было бы, если бы у нас не было разрешения на запись:

```
In [3]:
try:
    f = open('testfile','r')
    f.write('Попробуй написать это')
except IOError:
    # Это только проверит исключение IOError и выполнит вывод на экран
    сообщения
    print("Ошибка: Не удалось найти файл или прочитать данные")
else:
    print("Содержимое записано успешно")
    f.close()
```

Ошибка: Не удалось найти файл или прочитать данные

Обратите внимание, что мы сделали только *print*. Код все еще работал, и мы могли продолжать выполнять действия и запускать блоки кода. Это чрезвычайно полезно, когда вам нужно учитывать возможные ошибки ввода в вашем коде. Вы можете быть готовы к ошибке и продолжать выполнять код, вместо того, чтобы ваш код просто сломался, как мы видели выше.

Мы могли бы также написать просто *except:*, если бы не были уверены, какое исключение произойдет. Тогда *except* проверит любое исключение и не нужно запоминать этот список типов исключений!

finally

Блок *finally* кода будет выполняться всегда, независимо от наличия исключения в *try* блоке кода. Синтаксис:

```
try:
    Блок кода
    ...
    Из-за каких-либо исключений этот код может быть пропущен
finally:
    Этот блок кода всегда будет выполняться.
```

Например:

```
In [4]:
try:
    f = open("testfile", "w")
    f.write("Тестовая запись")
    f.close()
finally:
    print("Всегда выполняется код блока finally")
```

Всегда выполняется код блока finally

Мы можем использовать это вместе с *except*. Давайте посмотрим на новый пример, в котором будет учитываться неверный ввод данных пользователем:

```
In [5]:
def askint():
    while True:
        try:
            val = int(input("Введите целое число: "))
        except:
            print("Кажется, вы ввели не целое число!")
            continue
        else:
            print("Да, это целое число!")
            print(val)
            break
    finally:
        print("Наконец-то, я выполнен!")
```

```
In [6]:
askint()
```

Введите целое число: пять

Кажется, вы ввели не целое число!

Наконец-то, я выполнен!

Введите целое число: 6

Да, это целое число!

6

Наконец-то, я выполнен!

Модули и пакеты

Модули в Python – это просто файлы Python с расширением .py, которые реализуют набор функций. Модули предназначены для того, чтобы в них хранить часто используемые функции, классы, константы и т.п. Можно условно разделить модули и программы: программы предназначены для непосредственного запуска, а модули для импортирования их в другие программы [12].

Самый простой способ импортировать модуль в Python это воспользоваться конструкцией:

```
import имя_модуля
```

Импорт и использование модуля *math*, который содержит математические функции:

```
In [1]: # import the library
import math
In [2]: math.factorial(5)
Out[2]: 120
```

Если вы хотите задать псевдоним для модуля в вашей программе, можно воспользоваться вот таким синтаксисом:

```
import имя_модуля as новое_имя
```

Пример:

```
In [3]:
import math as m
m.sin(m.pi/3)
Out[3]: 0.8660254037844386
```

Используя любой из вышеперечисленных подходов, при вызове функции из импортированного модуля, вам всегда придется указывать имя модуля (или псевдоним).

Для того, чтобы этого избежать делайте импорт через конструкцию

```
from имя_модуля import имя_объекта
```

Пример:

```
In [4]: from math import cos
cos(3.14)
Out[4]: -0.9999987317275395
```

При этом импортируется только конкретный объект (в нашем примере: функция *cos*), остальные функции недоступны.

Для импортирования нескольких функций из модуля, можно перечислить их имена через запятую.

```
from имя_модуля import имя_объекта1, имя_объекта2
```

Пакет в Python – это каталог, включающий в себя другие каталоги и модули, но при этом дополнительно содержащий файл `__init__.py`. Этот файл может быть пустым, и это указывает на то, что содержащийся в нем каталог является пакетом Python. Пакеты используются для формирования пространства имен, что позволяет работать с модулями через указание уровня вложенности (через точку).

Создадим файл *mymodule.py* на Рабочем столе. Для этого перейдем в папку Desktop в Jupyter Notebook, нажмем *New* → *Text File*. Внутри модуля напишем очень простую функцию:

```
def my_func():
    print('I am in mymodule.py')
```

Сохраним этот файл с расширением *.py*.

Создадим Notebook *myprogram.ipynb.*, импортируем туда функцию из модуля и проверим ее работу.

```
In [1]: from mymodule import my_func
In [2]: my_func()
I am in mymodule.py
```

Теперь покажем, как создавать пакеты.

С помощью Jupyter Notebook создаем папку (*New → Folder*) *MyMainPackage*. И внутри нее создаем папку *SubPackage*. Внутри этих папок создадим файл *__init__.py*.

Далее в папке *MyMainPackage* создадим файл *some_main_script.py*:

```
def report_main():
    print("Hey I am in some_main_script in main package.")
```

В папке *SubPackage* создадим файл *mysubscript.py* :

```
def sub_report():
    print("Hey I am a function inside mysubscript")
```

Далее в ноутбуке *myprogram* импортируем модули из пакета и вызовем соответствующие функции.

```
In [3]:
from MyMainPackage import some_main_script
from MyMainPackage.SubPackage import mysubscript
some_main_script.report_main()
mysubscript.sub_report()
```

```
Hey I am in some_main_script in main package.
Hey I am a function inside mysubscript
```

Мы также можем импортировать модули из пакетов таким образом:

```
import MyMainPackage.some_main_script
```

NumPy

NumPy – это библиотека Python, которая предоставляет функциональные возможности, сравнимые с математическими инструментами, такими как MATLAB и R. Несмотря на то, что в NumPy значительно упрощено взаимодействие с пользователем, она предлагает большой набор математических функций.

Одна из ключевых особенностей NumPy – объект *ndarray* для представления N-мерного массива. Это быстрый и гибкий контейнер для хранения больших наборов данных в Python. Массивы позволяют выполнять математические операции над целыми блоками данных, применяя такой же синтаксис, как для соответствующих операций над скалярами [13].

```
In [1]:
import numpy as np
```

Создание массива

В NumPy существует много способов создать массив. Один из наиболее простых – создать массив из обычных списков или кортежей Python, используя функцию `numpy.array()`.

```
In [2]: a = np.array([1, 2, 3])
In [3]: a
Out[3]: array([1, 2, 3])
In [4]: type(a)
Out[4]: numpy.array
```

Функция `array()` трансформирует вложенные последовательности в многомерные массивы.

```
In [5]: b = np.array([[1.5, 2, 3], [4, 5, 6]])
In [6]: b
Out[6]:
array([[1.5, 2. , 3. ],
       [4. , 5. , 6. ]])
```

Свойства (атрибуты) массива

У любого массива есть атрибут `shape` – кортеж, описывающий размер по каждому измерению, и атрибут `dtype` – объект, описывающий тип данных в массиве.

Важным свойством (или как правильнее говорить атрибутом) массива является количество его измерений, `ndim`.

```
In [7]: a.ndim
Out[7]: 1
In [8]: b.ndim
Out[8]: 2
```

Теперь, с помощью атрибута `shape`, посмотрим на количество элементов в каждом измерении.

```
In [9]: a.shape
Out[9]: (3,)
In [10]: b.shape
Out[10]: (2, 3)
```

Массив `a` имеет одно измерение и 3 элемента в нем, а массив `b` – по 3 элемента в 2 измерениях.

Если явно не задано противное, то функция `np.array` пытается самостоятельно определить подходящий тип данных для создаваемого массива. Этот тип данных хранится в специальном объекте `dtype`.

```
In [11]: a.dtype
Out[11]: dtype('int32')
In [12]: b.dtype
Out[12]: dtype('float64')
```

Функция `array()` не единственная функция для создания массивов. Обычно элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива – `float64`). Функция `zeros()` создает массив из нулей, а функция `ones()` – массив из единиц. Обе функции принимают кортеж с размерами, и аргумент `dtype`:

Для создания последовательностей чисел, в NumPy имеется функция `arange()`, аналогичная встроенной в Python `range()`, только вместо списков она возвращает массивы, и принимает не только целые значения:

```
In [13]: np.arange(10, 30, 5)
Out[13]: array([10, 15, 20, 25])
In [14]: np.arange(0, 1, 0.1)
Out[14]: array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Арифметические операции с массивами NumPy

Любая арифметическая операция над массивами одинакового размера применяется к соответствующим элементам

```
In [15]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])
In [16]: arr * arr
Out[16]: array([[ 1.,  4.,  9.],
               [16., 25., 36.]])
In [17]: arr - arr
Out[17]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

Как легко догадаться, арифметические операции, в которых участвует скаляр, применяются к каждому элементу массива:

```
In [18]: 1 / arr
Out[18]: array([[1. , 0.5 , 0.33333333],
               [0.25 , 0.2 , 0.16666667]])
In [19]: arr ** 0.5
Out[19]: array([[1. , 1.41421356, 1.73205081],
               [2. , 2.23606798, 2.44948974]])
```

Индексирование и вырезание

Индексирование массивов NumPy – обширная тема, поскольку подмножество массива или его отдельные элементы можно выбрать различными способами. С одномерными массивами все просто. На поверхностный взгляд они ведут себя как списки Python:

```
In [20]: arr = np.arange(10)
In [21]: arr
Out[21]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
In [22]: arr[5]
Out[22]: 5
In [23]: arr[5:8]
Out[23]: array([5, 6, 7])
```



```
In [24]: arr[5:8] = 12
In [25]: arr
Out[25]: array([ 0, 1, 2, 3, 4, 12, 12, 12, 8, 9])
```

Как видите, если присвоить скалярное значение срезу, как в `arr[5:8] = 12`, то оно распространяется (или укладывается) на весь срез. Важнейшее отличие от списков состоит в том, что срез массива является представлением исходного массива. Это означает, что данные на самом деле не копируются, а любые изменения, внесенные в представление, попадают и в исходный массив.

В случае двумерного массива результатом индексирования с одним индексом является не скаляр, а одномерный массив:

```
In [26]: arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
In [27]: arr2d[2]
Out[27]: array([7, 8, 9])
```

К отдельным элементам можно обращаться рекурсивно. Но это слишком громоздко, поэтому для выбора одного элемента можно указать список индексов через запятую.

Таким образом, следующие две конструкции эквивалентны:

```
In [28]: arr2d[0][2]
Out[28]: 3
In [29]: arr2d[0, 2]
Out[29]: 3
```

Рассмотрим приведенный выше двумерный массив `arr2d`. Применение к нему вырезания дает несколько иной результат:

```
In [30]: arr2d
Out[30]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
In [31]: arr2d[:2]
Out[31]: array([[1, 2, 3],
                [4, 5, 6]])
```

Выражение `arr2d[:2]` полезно читать так: «выбрать первые две строки `arr2d`».

Можно указать несколько срезов – как несколько индексов:

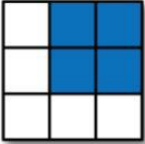

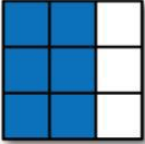
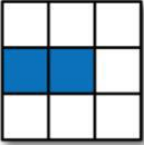
	Выражение	Форма
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code> <code>arr[2, :]</code> <code>arr[2:, :]</code>	<code>(3,)</code> <code>(3,)</code> <code>(1, 3)</code>
	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code> <code>arr[1:2, :2]</code>	<code>(2,)</code> <code>(1, 2)</code>

Рисунок 7 – Вырезание из двумерного массива

Универсальные функции

Универсальной функцией, или *u-функцией*, называется функция, которая выполняет поэлементные операции над данными, хранящимися в объектах *ndarray*.

```
In [34]: arr
Out[34]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
In [35]: np.sqrt(arr)
Out[35]: array([0. , 1. , 1.41421356, 1.73205081, 2. ,
2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.] )
```

С остальными унарными и бинарными *u-функциями* можно ознакомиться в документации.

Программирование с применением массивов

С помощью массивов NumPy многие виды обработки данных можно записать очень кратко, не прибегая к циклам.

В качестве простого примера предположим, что нужно вычислить функцию $\sqrt{x^2 + y^2}$ на регулярной сетке. Функция *np.meshgrid* принимает два одномерных массива и порождает две двумерные матрицы, соответствующие всем парам (x, y) элементов, взятых из обоих массивов:

```
In [36]: points = np.arange(-5, 5, 0.01) # 1000 равноотстоящих точек
In [37]: xs, ys = np.meshgrid(points, points)
In [38]: xs
```

```
Out[38]:
array([[ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99],
       [ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99],
       [ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99],
       ...,
       [ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99],
       [ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99],
       [ -5. , -4.99, -4.98, ..., 4.97, 4.98, 4.99]])
```

Теперь для вычисления функции достаточно написать такое выражение:

```
In [39]: z = np.sqrt(xs ** 2 + ys ** 2)
In [40]: z
Out[40]:
array([[7.07106781, 7.06400028, 7.05693985, ..., 7.04988652,
        7.05693985, 7.06400028],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774,
        7.04985815, 7.05692568],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603,
        7.04278354, 7.04985815],
       ...,
       [7.04988652, 7.04279774, 7.03571603, ..., 7.0286414,
        7.03571603, 7.04279774],
       [7.05693985, 7.04985815, 7.04278354, ..., 7.03571603,
        7.04278354, 7.04985815],
       [7.06400028, 7.05692568, 7.04985815, ..., 7.04279774,
        7.04985815, 7.05692568]])
```

Генерация псевдослучайных чисел

Модуль *numpy.random* дополняет встроенный модуль *random* функциями, которые генерируют целые массивы случайных чисел с различными распределениями вероятности. Например, с помощью функции можно получить случайный массив 4×4 с нормальным распределением:

```
In [41]: samples = np.random.normal(size=(4, 4))
In [42]: samples
Out[42]:
array([[ 1.38529845, -0.14386935, 1.33112098, 0.43375974],
       [-1.02659656, -0.30618953, 0.08827632, -1.72090844],
       [-1.60120252, 0.90287083, -0.92394833, -0.53273616],
       [ 0.25665238, -2.20787587, -2.25654285, -0.87220835]])
```

Математические и статистические операции

Среди методов массива есть математические функции, которые вычисляют статистики массива в целом или для данных вдоль одной оси. Выполнить агрегирование типа *sum*, *mean* или стандартного отклонения *std* можно с помощью как метода экземпляра массива, так и функции на верхнем уровне NumPy.

Ниже я сгенерирую случайные данные с нормальным распределением и вычислю некоторые агрегаты:

```
In [43]: arr = np.random.randn(5, 4)
In [44]: arr
Out[44]:
array([[ -0.77807628, -1.19947938, -0.11205892, -0.45264041],
       [-0.96820725,  0.51945868, -0.62273955,  1.04514895],
       [ 0.72222931, -0.67987006, -1.19035282, -1.49013953],
       [ 1.47774324,  0.59074125,  0.78031398, -0.41301117],
       [-2.08564941, -0.33848305,  0.7487961 , -0.95048145]])
In [45]: arr.mean()
Out[45]: -0.2698378871359619
In [46]: arr.sum()
Out[46]: -5.396757742719238
```

Функции типа *mean* и *sum* принимают необязательный аргумент *axis*, при наличии которого вычисляется статистика по заданной оси. В результате порождается массив на единицу меньшей размерности:

```
In [47]: arr.mean(axis=1)
Out[47]: array([-0.63556374, -0.00658479, -0.65953327,  0.60894683,
               -0.65645445])
In [48]: arr.sum(axis=0)
Out[48]: array([-1.63196038, -1.10763256, -0.3960412 , -2.26112361])
```

Здесь `arr.mean(axis=1)` означает «вычислить среднее по столбцам», а `arr.sum(axis=0)` – «вычислить сумму по строкам».

Сортировка

Как и встроенные в Python списки, массивы NumPy можно сортировать на месте методом *sort*:

```
In [49]: arr = np.random.randn(6)
In [50]: arr
Out[50]: array([-1.6074477 , -0.12534409,  0.11638427,  0.66276114,
               0.23113246, -1.36066133])
In [51]: arr.sort()
In [52]: arr
Out[52]: array([-1.6074477 , -1.36066133, -0.12534409,  0.11638427,
               0.23113246,  0.66276114])
```

Любой одномерный участок многомерного массива можно отсортировать, передав методу *sort* номер оси.

Линейная алгебра

Линейно-алгебраические операции: умножение и разложение матриц, вычисление определителей и другие – важная часть любой библиотеки для работы с массивами. В отличие от некоторых языков, например, MATLAB, в NumPy применение оператора `*` к двум двумерным массивам вычисляет поэлементное, а не матричное произведение. А для

перемножения матриц имеется функция *dot* – в виде как метода массива, так и функции в пространстве имен *numpy*:

```
In [53]: x = np.array([[1., 2., 3.], [4., 5., 6.]])
In [54]: y = np.array([[6., 23.], [-1, 7], [8, 9]])
In [55]: x
Out[55]: array([[1., 2., 3.],
               [4., 5., 6.]])
In [56]: y
Out[56]: array([[ 6., 23.],
               [-1., 7.],
               [ 8., 9.]])
In [57]: x.dot(y) # эквивалентно np.dot(x, y)
Out[57]: array([[ 28., 64.],
               [ 67., 181.]])
```

В модуле *numpy.linalg* имеется стандартный набор алгоритмов, в частности разложение матриц, нахождение обратной матрицы и вычисление определителя.

Изменение формы массива

Во многих случаях изменить форму массива можно без копирования данных. Для этого следует передать кортеж с описанием новой формы методу

экземпляра массива *reshape*. Например, предположим, что имеется одномерный массив, который мы хотели бы преобразовать в матрицу

```
In [58]: arr = np.arange(8)
In [59]: arr
Out[59]: array([0, 1, 2, 3, 4, 5, 6, 7])
In [60]: arr.reshape((4, 2))
Out[60]:
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

Форму многомерного массива также можно изменить:

```
In [61]: arr.reshape((4, 2)).reshape((2, 4))
Out[61]:
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

Обратная операция – переход от многомерного к одномерному массиву – называется **линеаризацией**:

```
In [62]: arr = np.arange(15).reshape((5, 3))
In [63]: arr
Out[63]: array([[ 0, 1, 2],
               [ 3, 4, 5],
               [ 6, 7, 8],
               [ 9, 10, 11],
               [12, 13, 14]])
In [64]: arr.ravel()
Out[64]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
```

Практические задания

1. Обработайте исключение, вызванное приведенным ниже кодом, с помощью блоков try и except:

```
In [1]:
for i in ['a', 'b', 'c']:
    print(i**2)
-----
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3912\2353522992.py in <module>
1 for i in ['a', 'b', 'c']:
----> 2 print(i**2)
      TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

2. Обработайте исключение, вызванное приведенным ниже кодом, с помощью блоков try и except. Затем используйте finally блок для печати «Все готово».

```
In [2]: x = 5
y = 0
z = x/y
-----
ZeroDivisionError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3912\983096354.py in <module>
2 y = 0
3
----> 4 z = x/y
      ZeroDivisionError: division by zero
```

3. Напишите функцию, которая запрашивает целое число и печатает его квадрат. Используйте while цикл с блоком try, except, else для учета неправильных входных данных.

```
In [3]:
def ask():
    pass
```

Возможный output:

```
Введите целое число: null
Произошла ошибка! Пожалуйста, попробуйте еще раз!
Введите целое число: 2
Ваше число в квадрате: 4
```

4. Выполните следующие действия:

а) Создайте проект со следующей структурой

```
Project_creatures
├── human
├── man
├── john.py
├── woman
├── mary.py
├── barbara.py
├── cat.py
├── dog.py
└── cow.py
```

- b) Импортируйте модуль `mary.py` в модуль `dog.py`
- c) Импортируйте модуль `cat.py` в модуль `barbara.py`
- d) Импортируйте модуль `john.py` в модуль `barbara.py`
- e) Создайте метод `foo()` внутри модуля `mary.py`
- f) Импортируйте метод `foo()` в модуль `barbara.py`

5. Подсчитать произведение ненулевых элементов на диагонали прямоугольной матрицы `x = np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]])`.

6. Найти максимальный элемент в векторе `x` среди элементов, перед которыми стоит нулевой. Для `x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0])`.

7. Напишите программу NumPy, чтобы добавить границу (заполненную нулями) вокруг существующего массива (можно воспользоваться `numpy.pad`).

8. Создайте диагональную матрицу размера $n \times n$. На главной диагонали должны быть числа от 1 до n . Сохраните матрицу в переменную `Z` (можно воспользоваться `np.diag`).

9. Создать массив чисел от 2 до 75. Вывести только нечётные. Присвоить нечётным числам этого массива значение -1.

10. Создать случайную квадратную матрицу случайного размера от 10 до 100. Найти максимум и сумму элементов. Поделить каждый элемент на максимум. Отнять от каждой строки матрицы среднее по строке. Заменить максимальное значение на -1.

Контрольные вопросы

- 1. Когда выполняется `except`, в блоке `try-except`?
- 2. Что такое модули Python?
- 3. Являются ли массивы Python NumPy лучше списков?
- 4. В чем разница между модулем и пакетом?
- 5. Как импортировать NumPy под именем `np`?
- 6. Способы проверки пуст ли массив?
- 7. Каким образом можно получить версию NumPy?
- 8. Какая функция NumPy используется для того, чтобы получить помощь по функции библиотеки NumPy?
- 9. С помощью какой функции NumPy можно выполнить матричное произведение матриц?
- 10. Где применяется библиотека NumPy?

Лабораторная работа №7. Pandas

Цель:

Научиться решать аналитические задачи с помощью библиотеки Pandas.

Теоретический материал

Pandas – это очень популярная библиотека Python для анализа и обработки данных. Pandas похожа на Excel для Python, поскольку включает простые в использовании функции для таблиц данных.

```
In [1]: import pandas as pd
```

Считывание csv

Функция `read_csv()` считывает csv файл, который лежит по указанному в скобках пути.

Дополнительные аргументы функции `read_csv`:

1) `encoding` – параметр в `read_csv`, отвечает за кодировку текста, которая может быть различной. Самая распространённая – utf-8.

2) `sep` – разделитель между ячейками в строке (по умолчанию ,)

3) `parse_dates` – указывает, стоит ли воспринимать даты как даты? (по умолчанию они воспринимаются пандасом как строки).

```
In [2]: # Файл с информацией о продажах различных курсов
df = pd.read_csv('pandas1_data.csv', encoding='windows-1251', sep=';')
```

```
In [3]: df
```

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.30	Краснодар	Яндекс.Касса
2	1062856	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.00	NaN	NaN
3	1062880	03.12.2019 0:18	NaN	Курс обучения «Консультант»	Отменен	0.00	г.Москва и Московская область	NaN
4	1062899	03.12.2019 21:43	NaN	Курс обучения «Эксперт»	Отменен	0.00	г.Москва и Московская область	NaN
...
287	1064720	30.12.2019 9:42	30.12.2019 12:49	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	Самарская область	Яндекс.Касса
288	1064724	30.12.2019 11:32	NaN	Курс обучения «Консультант»	Отменен	0.00	NaN	NaN
289	1064775	31.12.2019 2:17	31.12.2019 2:22	Курс обучения «Консультант»	Завершен	7423.92	NaN	Сбербанк эквайринг, Бонусный счет
290	1064793	31.12.2019 16:40	01.01.2020 14:29	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	Республика Карелия	Яндекс.Касса
291	1064796	31.12.2019 17:29	31.12.2019 17:32	Курс от Школы Диетологов. Повышение квалификац...	Завершен	9898.56	Уфа	Сбербанк эквайринг, Бонусный счет

292 rows × 8 columns

Рисунок 6 – Вывод датафрейма df

Объект `DataFrame` представляет табличную структуру данных, состоящую из упорядоченной коллекции столбцов, причем типы значений (числовой, строковый, булев и

т. д.) в разных столбцах могут различаться. В объекте *DataFrame* хранятся два индекса: по строкам и по столбцам.

Атрибуты и методы

Метод *head()* отбирает только первые пять строк, а метод *tail()* отбирает только последние пять строк

shape – атрибут, хранящий данные о размерах таблицы. Возвращает кортеж, содержащий 2 значения – число строк и число колонок в нём.

```
In [4]: df.shape  
Out[4]: (292, 8)
```

Чтобы узнать типы колонок в вашем датафрейме, воспользуйтесь атрибутом *dtypes* – он возвращает серию с описанием типа каждой колонки

Типы более-менее совпадают с типами в Python, однако есть и различия:

- здесь у типов присутствует описание размера (числа битов);
- все сложные типы (не числа или логические значения) отображаются как *object*.

Информация о типе важна для дальнейшей работы с датафреймом (например, чтобы не произвести сложение строк, думая, что это числа).

```
In [5]: df.dtypes  
Out[5]:  
Номер int64  
Дата создания object  
Дата оплаты object  
Title object  
Статус object  
Заработано float64  
Город object  
Платежная система object  
dtype: object
```

describe() – удобный метод для вывода описания числовых колонок в датафрейме:

```
In [6]: df.describe()
```

	Номер	Заработано
count	2.920000e+02	292.000000
mean	1.063745e+06	3397.615034
std	4.438688e+02	5771.572829
min	1.062823e+06	0.000000
25%	1.063608e+06	0.000000
50%	1.063698e+06	2935.440000
75%	1.063807e+06	2935.440000
max	1.064796e+06	42750.000000

Рисунок 7 – Метод *describe()*

describe() выводит информацию о числе строк, среднем значении, стандартном отклонении, минимуме, максимуме и значениях по 25-му, 50-му и 75-му квартилям. Он действует только на числах, так как большинство этих параметров неочевидно определяются для других типов данных (например, строк).

Переименование колонок

В идеале названия колонок осмыслены, актуальны, не содержат пробелов и на английском. Конечно, для каких-то задач, они могут быть и с пробелами, и на другом языке. В любом случае, если вы хотите их переименовать, для этого есть метод *rename()*. Один из способов переименования – передать словарь, в котором ключами являются старые названия, а значениями – новые.

```
In [7]: df.columns
Out[7]:
Index(['Номер', 'Дата создания', 'Дата оплаты', 'Title', 'Статус',
'Заработано', 'Город', 'Платежная система'],
dtype='object')
```

```
In [8]: df = df.rename(columns={'Номер': 'number',
'Дата создания': 'create_date',
'Дата оплаты': 'payment_date',
'Title': 'title',
'Статус': 'status',
'Заработано': 'money',
'Город': 'city',
'Платежная система': 'payment_system'})
```

```
In [9]: df.columns
Out[9]:
Index(['number', 'create_date', 'payment_date', 'title', 'status',
'money', 'city', 'payment_system'], dtype='object')
```

```
In [10]: df.head()
```

	number	create_date	payment_date	title	status	money	city	payment_system
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.5	Чита	Сбербанк эквайринг
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.3	Краснодар	Яндекс.Касса
2	1062856	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.0	NaN	NaN
3	1062880	03.12.2019 0:18	NaN	Курс обучения «Консультант»	Отменен	0.0	г.Москва и Московская область	NaN
4	1062899	03.12.2019 21:43	NaN	Курс обучения «Эксперт»	Отменен	0.0	г.Москва и Московская область	NaN

Рисунок 8 – Вывод датафрейма *df*

Обращение к колонкам

В пандасе существует множество способов обратиться к колонке датафрейма. Самый удобный *df.column_name*, где *df* – датафрейм, *column_name* – название колонки.

Чтобы это работало, название колонки должно состоять из одного компонента (например, слова), и не должно совпадать с названием методов датафрейма (имя колонки

count не работает). Для языковой однородности – ещё и на английском, но это не является обязательным.

Что делать, если название колонки состоит из 2-х слов? Либо переименовать колонку, либо использовать другой способ доступа: `df['column name']`.

Работает для всех случаев кроме тех, когда в названии присутствуют одинарные кавычки. Тогда либо используйте вокруг названия двойные, либо поставьте \ перед кавычками внутри. Лучше называть колонки без кавычек.

Для получения нескольких колонок передайте внутрь квадратных скобок список с именами желаемых колонок:

```
df[['column1', 'column2', 'column3']]
In [11]: df.payment_system
Out[11]:
0 Сбербанк эквайринг
1 Яндекс.Касса
2 NaN
3 NaN
4 NaN
...
287 Яндекс.Касса
288 NaN
289 Сбербанк эквайринг,Бонусный счет
290 Яндекс.Касса
291 Сбербанк эквайринг,Бонусный счет
Name: payment_system, Length: 292, dtype: object
```

Результат возвращается в формате *pd.Series* (серии).

pd.Series – более примитивный тип данных в pandas, соответствует колонке датафрейма. В ней хранятся данные одного типа (числа/строки/...). Работая с колонкой, мы работаем именно с серией. Часть методов и атрибутов серии и датафрейма совпадают.

unique – метод, возвращающий уникальные значения в колонке.

nunique – метод, который считает число уникальных значений в колонке.

```
In [12]: df['title'].head()
Out[12]:
0 Курс обучения «Эксперт»
1 Курс обучения «Эксперт»
2 Курс обучения «Специалист»
3 Курс обучения «Консультант»
4 Курс обучения «Эксперт»
Name: title, dtype: object

In [13]: df[['title', 'status']].head()
```

	title	status
0	Курс обучения «Эксперт»	Завершен
1	Курс обучения «Эксперт»	Завершен
2	Курс обучения «Специалист»	Отменен
3	Курс обучения «Консультант»	Отменен
4	Курс обучения «Эксперт»	Отменен

Рисунок 9 – Вывод колонок *title* и *status*

Найдем суммарное количество денег по каждому из курсов, отсортируем курсы по заработку и также посмотрим разбивку по городам.

Перед тем как сделать все необходимые преобразования с данными, хорошей практикой будет создать переменную, в которую сохраним, в нашем случае, сумму всех денег. Чтобы после проделанной работы, мы проверили и убедились, что ничего не потеряли и это число не изменилось.

Существует набор методов, доступных для колонок датафреймов. Например, есть колонка *money* в датафрейме, содержащая полученные объёмы денег. Применяв метод *sum()*, можно посчитать их сумму.

```
In [14]: all_money = df.money.sum()
In [15]: all_money
Out[15]: 992103.5900000001
```

Группировка

Часто используемый приём для вычисления чего-либо по данным. Осуществляется с помощью метода *groupby()* – группирует данные в датафрейме по указанным колонкам.

```
In [16]: df.groupby('title')
Out[16]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000112CD495A00>
```

Применение одного метода *groupby()* не даёт видимого эффекта. *groupby()* обычно используется не сам по себе, а в связке с *agg()* или другим методом. Можно использовать несколько колонок для группировки, передав их в виде списка.

Дополнительные параметры:

as_index – принимает True или False для обозначения того, нужно ли использовать переданные для группировки колонки в качестве индекса, по умолчанию True

Агрегация

agg() – функция для агрегирования данных, применяется после группировки методом *groupby()*.

Как это работает – агрегируем методом *agg()*, указывая на каких колонках какие действия произвести.

Существуют разные способы передать в *agg()* что и как вы хотите агрегировать. Самый простой и полный – использовать словарь, в котором ключами являются названия колонок, а значениями – применяемые к ним функции. Чтобы применить несколько функций, используйте список функций. Можно передать как сами функции (*sum*), так и обозначающие их строки (*'sum'*).

В результате агрегации из массива значений (колонка) получается одно значение на каждую агрегирующую функцию.

```
In [17]: df.groupby('title', as_index=False).aggregate({'money': 'sum'})
```

	title	money
0	Курс обучения «Консультант»	208163.49
1	Курс обучения «Специалист»	160862.64
2	Курс обучения «Эксперт»	148992.80
3	Курс от Школы Диетологов. Бизнес	18752.54
4	Курс от Школы Диетологов. Повышение квалификац...	88384.92
5	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	366947.20

Рисунок 10 – Вывод результата

Сортировка значений

Для такой сортировки используется метод *sort_values()*, получающий колонку/список колонок, по которым будет идти сортировка (обратите внимание, заглавные буквы считаются меньше обычных):

Дополнительные параметры:

ascending – принимает логическое значение, показывающее сортировать ли колонку по возрастанию

```
In [18]: # Добавим группировку по городу
money_by_city = df.groupby(['title', 'city'], as_index=False) \
    .aggregate({'money': 'sum'}) \
    .sort_values('money', ascending=False)
In [19]: money_by_city
```

	title	city	money
51	Курс обучения «Эксперт»	г.Санкт-Петербург и Ленинградская область	59195.00
156	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	г.Москва и Московская область	46967.04
39	Курс обучения «Эксперт»	Балхаш	42750.00
95	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Краснодарский край	38169.78
30	Курс обучения «Специалист»	Краснодар	29695.70
...
63	Курс от Школы Диетологов. Повышение квалификац...	Кемерово	0.00
61	Курс от Школы Диетологов. Повышение квалификац...	Екатеринбург	0.00
55	Курс от Школы Диетологов. Бизнес	Крым Советский	0.00
54	Курс от Школы Диетологов. Бизнес	Киев	0.00
158	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	нижний Новгород	0.00

159 rows × 3 columns

Рисунок 11 – Вывод датафрейма *money_by_city*

value_counts

Метод, который считает, сколько раз встречается каждое уникальное значение переменной.

Посчитать, сколько раз встречается каждое название курса (*title*), можно с помощью следующей команды:

```
In [20]: df['title'].value_counts()
Out[20]:
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием 182
Курс обучения «Консультант» 52
Курс обучения «Эксперт» 22
Курс от Школы Диетологов. Повышение квалификации. 15
Курс обучения «Специалист» 12
Курс от Школы Диетологов. Бизнес 9
Name: title, dtype: int64
```

Также метод *value_counts()* принимает на вход несколько параметров:

normalize – показать относительные частоты уникальных значений (по умолчанию равен *False*).

dropna – не включать количество *NaN* (по умолчанию равен *True*)

bins – сгруппировать количественную переменную (например, разбить возраст на возрастные группы); для использования данного параметра нужно указать, на сколько групп разбить переменную

```
In [21]: # Получаем частоту встречаемости:
df['title'].value_counts(normalize=True)
Out[21]:
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием 0.623288
Курс обучения «Консультант» 0.178082
Курс обучения «Эксперт» 0.075342
Курс от Школы Диетологов. Повышение квалификации. 0.051370
Курс обучения «Специалист» 0.041096
```

Курс от Школы Диетологов. Бизнес 0.030822
Name: title, dtype: float64

Запись в файл

Датафрейм можно записать в различный формат, самый распространённый, пожалуй, csv. Для этого нужно применить к датафрейму метод `to_csv` и передать в него путь, по которому вы хотите создать файл.

```
df.to_csv('my.csv')
```

Дополнительные параметры:

`index` – записать индекс датафрейма в csv как первую колонку

`sep` – используемый при записи разделитель колонок

```
In [22]: money_by_city.to_csv('money_by_city.csv', index=False)
```

Посчитаем сколько было суммарно денег и сколько завершённых заказов.

```
In [23]: df.groupby('title', as_index=False) \
.aggregate({'money': 'sum', 'number' : 'count'}) \
.sort_values('money', ascending=False)
```

	title	money	number
5	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	366947.20	182
0	Курс обучения «Консультант»	208163.49	52
1	Курс обучения «Специалист»	160862.64	12
2	Курс обучения «Эксперт»	148992.80	22
4	Курс от Школы Диетологов. Повышение квалификац...	88384.92	15
3	Курс от Школы Диетологов. Бизнес	18752.54	9

Рисунок 12 – Вывод результата

Запросы

В пандасе есть возможность фильтровать данные используя SQL-like синтаксис. Для этого нужен метод `query()`, принимающий строку с запросом. Внутри него можно использовать названия колонок (если они без пробелов). При использовании строк внутри запроса экранируйте кавычки \ или используйте другую пару

В `query` также можно передать сразу несколько условий. Условия, которые должны выполняться одновременно, соединяются с помощью `and` или `&`:

```
product_data.query("title == 'Курс обучения «Эксперт»' and  
                    status == 'Завершен'")
```

Когда должно удовлетворяться одно из условий – or или |:

```
product_data.query("title == 'Курс обучения «Эксперт»'  
                  or status == 'Завершен'")
```

```
In [24]:  
money_title = df \  
.query("status == 'Завершен'") \  
.groupby('title', as_index=False) \  
.aggregate({'money': 'sum', 'number': 'count'}) \  
.sort_values('money', ascending=False) \  
.rename(columns={'number': 'success_orders'})  
In [25]: money_title
```

	title	money	success_orders
5	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	366947.20	125
0	Курс обучения «Консультант»	208163.49	31
1	Курс обучения «Специалист»	160862.64	7
2	Курс обучения «Эксперт»	148992.80	5
4	Курс от Школы Диетологов. Повышение квалификац...	88384.92	9
3	Курс от Школы Диетологов. Бизнес	18752.54	3

Рисунок 13 – Вывод датафрейма *money_title*

Самопроверка

```
In [26]: money_title.money.sum()  
Out[26]: 992103.59  
In [27]: all_money  
Out[27]: 992103.5900000001  
In [28]:  
if int(money_title.money.sum()) == int(all_money):  
    print('OK')  
else:  
    print('ERROR!!')
```

OK

Теперь приведем пример решения реальной аналитической задачи.

К нам обратилось несколько крупных брендов, которые размещают на нашей платформе товары, и они хотят тестировать систему лояльности. Они хотят самым лояльным пользователям, то есть тем, кто постоянно покупают определенный бренд, выдать карты с большой скидкой. Соответственно, скидка такая большая, что выдавать всем пользователям подряд они не хотят, хотят только тем пользователям, которые в долгосрочной перспективе в каком-то смысле эту скидку компенсируют своей лояльностью.

Выгрузили таблицу покупок каждого пользователя за последние несколько месяцев.
Необходимо найти лояльных пользователей.

Данные находятся в файле : pandas2_data.csv.

```
In [1]: import pandas as pd
In [2]: df = pd.read_csv('pandas2_data.csv', encoding='windows-1251')
In [3]: df
```

tc	cta	id_art	id_subsubfam	id_subfam	...	id_famn	id_seccion	id_subagr	id_agr	vta	uni	id_artn	art_sp	fam_sp	fam_en
110000761	11000076	21895	101070640100	1010706401	...	10107064	10107	101	1	0.68	1.0	21895	MARAVILLA 500 G Store_Brand	PASTA ALIMENTICIA SE	pasta
110000761	11000076	21816	101070640100	1010706401	...	10107064	10107	101	1	0.38	1.0	21816	FIDEO CABELLIN 500 G Store_Brand	PASTA ALIMENTICIA SE	pasta
28491841	2849184	562840	101070640100	1010706401	...	10107064	10107	101	1	1.55	2.0	562840	SPAGUETTI № 5 500 G Brand_1	PASTA ALIMENTICIA SE	pasta
95931501	9593150	28914	101070640100	1010706401	...	10107064	10107	101	1	1.03	2.0	28914	FIDEO FIDEUБ 500 Brand_7	PASTA ALIMENTICIA SE	pasta
93265591	9326559	159867	101070640100	1010706401	...	10107064	10107	101	1	1.09	1.0	159867	MACARRONES GRATINAR 5 Brand_2	PASTA ALIMENTICIA SE	pasta

Рисунок 14 – Вывод датафрейма *df*

```
In [4]: df.shape
Out[4]: (48129, 21)
```

Предподготовка данных

В наших данных есть две основные колонки, которые нам понадобятся – *tc* (id пользователя) и *art_sp* (название бренда – производители различной пасты)

```
In [5]: user_df = df[['tc', 'art_sp']]
In [6]: user_df
```

	tc	art_sp
0	110000761	MARAVILLA 500 G Store_Brand
1	110000761	FIDEO CABELLIN 500 G Store_Brand
2	28491841	SPAGUETTI № 5 500 G Brand_1
3	95931501	FIDEO FIDEUБ 500 Brand_7
4	93265591	MACARRONES GRATINAR 5 Brand_2
...
48124	45518841	FIDEOS 0 500 G Brand_4
48125	110824211	PLUMAS 3 500 G Brand_4
48126	1408670389	MACARRONES 500 G Store_Brand
48127	1408670389	SPAGHETTI 500 G Store_Brand
48128	48582221	SPAGHETTINI 500 G Store_Brand

Рисунок 15 – Вывод датафрейма *user_df*

Переименуем колонки по смыслу:

```
In [7]: user_df = user_df.rename(columns={'tc': 'user_id',
'art_sp': 'brand_info'})
In [8]: user_df.head()
```

	user_id	brand_info
0	110000761	MARAVILLA 500 G Store_Brand
1	110000761	FIDEO CABELLIN 500 G Store_Brand
2	28491841	SPAGUETTI № 5 500 G Brand_1
3	95931501	FIDEO FIDEUБ 500 Brand_7
4	93265591	MACARRONES GRATINAR 5 Brand_2

Рисунок 16 – Вывод результата

Нужно из строки достать только информацию о бренде, которая хранится в конце строки.

```
In [9]: brand_name = 'MARAVILLA 500 G Store_Brand'
```

Воспользуемся методом *split()*.

```
In [10]: brand_name.split(' ')[-1]
```

```
Out[10]: 'Store_Brand'
```

```
In [11]:
```

```
def split_brand(brand_name_data):
    return brand_name_data.split(' ')[-1]
```

Теперь нужно применить эту функцию ко всей колонке *brand_info*. Воспользуемся методом *apply()* – применяет переданную в него функцию ко всем колонкам вызванного датафрейма.

```
In [12]:
```

```
user_df['brand_name'] = user_df.brand_info.apply(split_brand)
```

```
In [13]: user_df.head()
```

	user_id	brand_info	brand_name
0	110000761	MARAVILLA 500 G Store_Brand	Store_Brand
1	110000761	FIDEO CABELLIN 500 G Store_Brand	Store_Brand
2	28491841	SPAGUETTI № 5 500 G Brand_1	Brand_1
3	95931501	FIDEO FIDEUБ 500 Brand_7	Brand_7
4	93265591	MACARRONES GRATINAR 5 Brand_2	Brand_2

Рисунок 17 – Вывод результата

Приступаем к аналитике

Проверим гипотезу, что в наших данных о покупках есть такие пользователи, которые являются лояльными пользователями к некоторому бренду. То есть, если человек совершил, например, 10 покупок за последнее время, то у него будет паттерн покупать

скорее один и тот же бренд, а не каждый раз разный. И тем пользователям, которые демонстрируют лояльность, мы можем выдать карту со скидкой.

Посмотрим на число покупок, который совершил пользователь.

```
In [14]: users_purchases = user_df.groupby('user_id', as_index=False) \
        .agg({'brand_name': 'count'}).rename(columns={'brand_name': 'purchases'})
In [15]: users_purchases.head()
```

	user_id	purchases
0	-1236394515	1
1	1031	6
2	4241	5
3	17311	2
4	17312	2

Рисунок 18 – Вывод датафрейма *users_purchase*

Очевидно, что для того, чтобы эту задачу решить, не имеет смысла брать информацию про каждого из пользователя. Если пользователь совершил всего лишь одну покупку, мы ничего не можем сказать о его лояльности, потому что возможно, следующую покупку он совершит другого бренда. Пользователи с малым числом покупок – это очень ненадежный источник выводов о лояльности к бренду.

Можем посчитать медиану:

```
In [16]: users_purchases.purchases.median()
Out[16]: 2.0
```

Медианное количество покупок равно двум, это означает, что 50% наших пользователей совершало меньше двух покупок, остальные 50% – больше двух покупок.

Посмотрим, сколько уникальных пользователей у нас получилось.

```
In [17]: users_purchases.shape
Out[17]: (11764, 2)
In [18]: users_purchases.describe()
```

	user_id	purchases
count	1.176400e+04	11764.000000
mean	7.690517e+07	4.091210
std	1.622210e+08	4.573143
min	-1.236395e+09	1.000000
25%	1.503761e+07	1.000000
50%	4.682179e+07	2.000000
75%	9.311601e+07	5.000000
max	1.408849e+09	60.000000

Рисунок 19 – Вывод метода *describe()*

Видим, что есть 75% перцентиль, который равен 5, то есть 25% наших пользователей совершило больше 5 покупок. Для первого прикидочного анализа имеет смысл отобрать тех пользователей, которые совершили больше 5 покупок.

```
In [19]: users_purchases = user_df.groupby('user_id', as_index=False) \
.agg({'brand_name': 'count'}) \
.rename(columns={'brand_name': 'purchases'}) \
.query('purchases >= 5')
In [20]: users_purchases.describe()
```

	user_id	purchases
count	3.383000e+03	3383.000000
mean	6.421500e+07	9.320130
std	1.504830e+08	5.623993
min	1.031000e+03	5.000000
25%	8.871271e+06	6.000000
50%	2.842547e+07	7.000000
75%	8.542964e+07	11.000000
max	1.408810e+09	60.000000

Рисунок 20 – Вывод метода *describe()*

```
In [21]: users_purchases.head()
```

	user_id	purchases
1	1031	6
2	4241	5
11	25971	7
14	40911	27
16	45181	5

Рисунок 21 – Вывод результата

Мы для каждого пользователя рассчитали, сколько покупок он совершил. Теперь нужно понять, например, пользователь 1031 совершил 6 покупок одного бренда или все покупки разных брендов, или 5 покупок одного бренда и еще одну покупку другого.

Для этого снова возьмем сырые данные, сгруппируем по *user_id* и *brand_name*, и посчитаем сколько покупок было совершено пользователем именно в разрезе бренда.

```
In [22]: user_df.groupby(['user_id', 'brand_name'], as_index=False) \
.agg({'brand_info': 'count'}) \
.query('user_id == 1031')
```

	user_id	brand_name	brand_info
1	1031	Brand_3	1
2	1031	Store_Brand	5

Рисунок 22 – Вывод результата группировки

Теперь для каждого пользователя найдем любимый бренд, и какой процент от всех покупок приходится на любимый бренд.

Для каждого пользователя найдем бренд, в котором у него было максимальное количество покупок.

```
In [23]: user_df.groupby(['user_id', 'brand_name'], as_index=False) \
        .agg({'brand_info': 'count'}) \
        .sort_values(['user_id', 'brand_info'], ascending=[False, False])
```

	user_id	brand_name	brand_info
18187	1408849249	Store_Brand	1
18186	1408840919	Store_Brand	1
18185	1408832719	Brand_4	3
18184	1408825059	Brand_1	1
18183	1408817589	Store_Brand	2
...
3	4241	Brand_4	3
4	4241	Store_Brand	2
2	1031	Store_Brand	5
1	1031	Brand_3	1
0	-1236394515	Brand_4	1

Рисунок 23 – Вывод результат

Взять для каждого пользователя его первое значение с максимальным количеством покупок. Для этого воспользуемся `groupby('user_id')`, а чтобы взять первую строку - `head(1)`, который после группировки вернет первое наблюдение для каждого пользователя, а так как внутри пользователя у нас покупки брендов отсортированы по их как бы привлекательности для пользователя, таким образом этой командой мы для каждого пользователя найдем бренд с максимальным числом покупок.

```
In [24]: lovely_brand_purchases_df = user_df.groupby(['user_id',
'brand_name'], as_index=False) \
        .agg({'brand_info': 'count'}) \
        .sort_values(['user_id', 'brand_info'], ascending=[False, False]) \
        .groupby('user_id') \
        .head(1) \
        .rename(columns={'brand_name': 'lovely_brand', 'brand_info':
'lovely_brand_purchases'})
In [25]: lovely_brand_purchases_df
```

	user_id	lovely_brand	lovely_brand_purchases
18187	1408849249	Store_Brand	1
18186	1408840919	Store_Brand	1
18185	1408832719	Brand_4	3
18184	1408825059	Brand_1	1
18183	1408817589	Store_Brand	2
...
6	17312	Brand_1	1
5	17311	Brand_4	2
3	4241	Brand_4	3
2	1031	Store_Brand	5
0	-1236394515	Brand_4	1

Рисунок 24 – Вывод датафрейма *lovely_brand_purchase_df*

У нас есть два разных датафрейма. В одном датафрейме хранятся данные по общему числу покупок для тех пользователей, у которых число покупок больше 5. Во втором датафрейме для каждого пользователя хранится число покупок любимого бренда. Теперь нам нужно эти данные как-то объединить, чтобы они оказались в одном датафрейме.

Перед этим посчитаем число уникальных брендов. Если человек совершил больше пяти покупок и у него всего лишь один уникальный бренд, на самом деле мы уже решили нашу задачу. Этих пользователей по нашему определению уже можно считать лояльными.

```
In [26]: users_unique_brands = user_df.groupby('user_id', as_index=False) \
        .agg({'brand_name' : pd.Series.nunique}) \
        .rename(columns={'brand_name': 'unique_brands'})
In [27]: users_unique_brands
```

	user_id	unique_brands
0	-1236394515	1
1	1031	2
2	4241	2
3	17311	1
4	17312	2
...
11759	1408817589	2
11760	1408825059	1
11761	1408832719	1
11762	1408840919	1
11763	1408849249	1

Рисунок 25 – Вывод датафрейма *users_unique_brands*

Объединение датафреймов

Зачастую называется джойном. Очень частая операция, которую можно сделать с помощью нескольких функций. Одна из них – *merge*. Обязательным аргументом является другой датафрейм, с которым планируется объединение. Объединение идёт по общей колонке, у которой имеется одинаковый смысл и общие значения в обоих датафреймах. Существуют различные типы джойнов, самый частый, пожалуй, *inner*.

Дополнительные аргументы функции *merge*:

how – как объединять датафреймы, одно из *inner, outer, left, right* (по умолчанию *inner*)

on – общая колонка, по которой будет идти объединение.

```
In [28]: loyalty_df = users_purchases \
.merge(users_unique_brands, on='user_id') \
.merge(loyalty_brand_purchases_df, on='user_id')
In [29]: loyalty_df.head()
```

	user_id	purchases	unique_brands	lovely_brand	lovely_brand_purchases
0	1031	6	2	Store_Brand	5
1	4241	5	2	Brand_4	3
2	25971	7	2	Store_Brand	5
3	40911	27	5	Brand_4	19
4	45181	5	4	Store_Brand	2

Рисунок 26 – Вывод датафрейма *loyalty_df*

Выберем тех пользователей, у которых только один любимый бренд. Их мы уже можем отнести к лояльным пользователям.

```
In [30]: loyal_users = loyalty_df[loyalty_df.unique_brands == 1]
In [31]: loyal_users
```

	user_id	purchases	unique_brands	lovely_brand	lovely_brand_purchases
13	86281	14	1	Brand_4	14
18	94961	6	1	Brand_4	6
29	132061	9	1	Brand_4	9
30	134281	6	1	Brand_4	6
35	157311	12	1	Brand_4	12
...
3372	1010244089	9	1	Store_Brand	9
3374	1010247239	5	1	Brand_4	5
3376	1010274559	5	1	Brand_4	5
3377	1110091379	5	1	Brand_1	5
3378	1408767189	5	1	Brand_4	5

Рисунок 27 – Вывод датафрейма *loyal_users*

Далее есть пользователи, которые покупают не один бренд. Давайте придумаем метрику, которую назовем лояльность пользователей. С помощью этой метрики мы можем измерить лояльность каким-то числом – например, 0 – совсем не лояльный, 1 – лояльный к определенному бренду.

Для каждого пользователя посчитаем, какая доля от всех покупок прилась на любимый бренд.

```
In [32]: loyalty_df['loyalty_score'] = loyalty_df.lovely_brand_purchases /  
loyalty_df.purchases  
In [33]: loyalty_df
```

	user_id	purchases	unique_brands	lovely_brand	lovely_brand_purchases	loyalty_score
0	1031	6	2	Store_Brand	5	0.833333
1	4241	5	2	Brand_4	3	0.600000
2	25971	7	2	Store_Brand	5	0.714286
3	40911	27	5	Brand_4	19	0.703704
4	45181	5	4	Store_Brand	2	0.400000
...
3378	1408767189	5	1	Brand_4	5	1.000000
3379	1408783189	10	2	Store_Brand	8	0.800000
3380	1408783379	6	2	Brand_1	4	0.666667
3381	1408798879	8	3	Store_Brand	4	0.500000
3382	1408810219	9	2	Brand_2	6	0.666667

Рисунок 28 – Вывод датафрейма *loyalty_df*

Визуализация данных

Графики – важная часть анализа данных, так как они наглядно – если тип графика хорошо подобран – представляют данные и позволяют быстро разобраться в сути.

Чтобы в юпитер ноутбуке отображались графики, выполните строчку:

```
In [34]: %matplotlib inline
```

seaborn

Продвинутая библиотека, позволяющая сделать очень красивые графики.

```
In [35]: import seaborn as sns
```

```
In [36]: ax = sns.displot(loyalty_df.loyalty_score, kde=False)
```

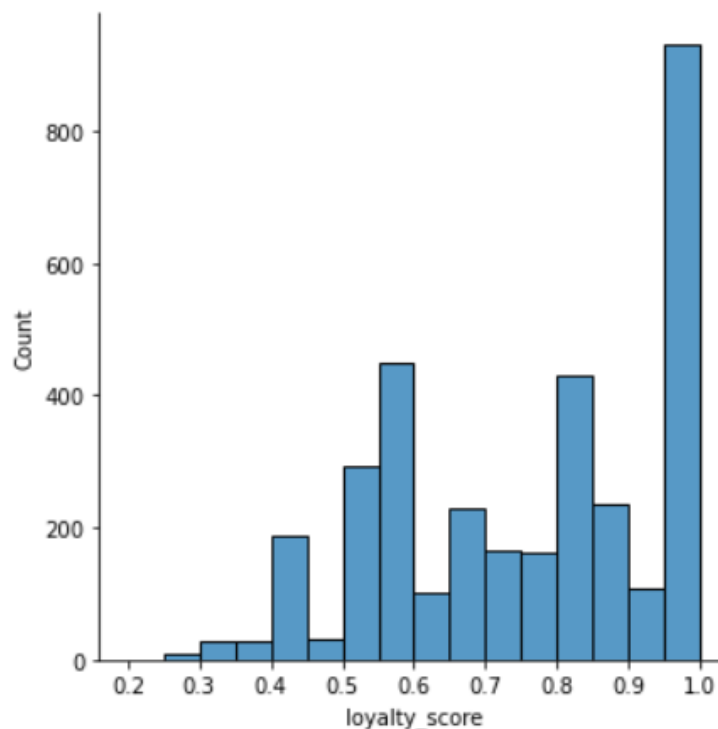



Рисунок 29 – Столбчатая диаграмма

По оси x *loyalty_score* от 0 до 1, а высота столбика показывает, сколько человек обладает такой лояльностью. Мы видим, что около 900 человек с *loyalty_score* = 1. Также есть достаточно большой набор пользователей, у которых *loyalty_score* от 0.7 до 0.9. И при этом есть пользователи, у которых лояльность довольно маленькая, то есть у них меньше половины всех покупок приходит на любимый бренд.

```
In [37]: loyalty_df.loyalty_score.median()
Out[37]: 0.8
```

Дальше нам необходимо принять чисто аналитическое решение. Каких пользователей, с каким *loyalty_score* считать лояльными? Правильного ответа на него нет. Все зависит от поставленной задачи. Посмотрим еще на бренды, ведь они тоже различаются по тому, как много у них лояльных пользователей.

```
In [38]: brands_loyalty = loyalty_df.groupby('lovely_brand', as_index=False) \
        \
        .agg({'loyalty_score': 'median', 'user_id': 'count'})
In [39]: brands_loyalty
```

	lovely_brand	loyalty_score	user_id
0	Brand_1	0.679487	410
1	Brand_2	0.600000	88
2	Brand_3	0.500000	115
3	Brand_4	0.818182	2041
4	Brand_5	0.600000	5
5	Brand_7	0.444444	9
6	Store_Brand	0.750000	715

Рисунок 30 – Вывод датафрейма *brands_loyalty*

```
In [40]: ax = sns.barplot(x="lovely_brand", y="loyalty_score",
data=brands_loyalty)
```

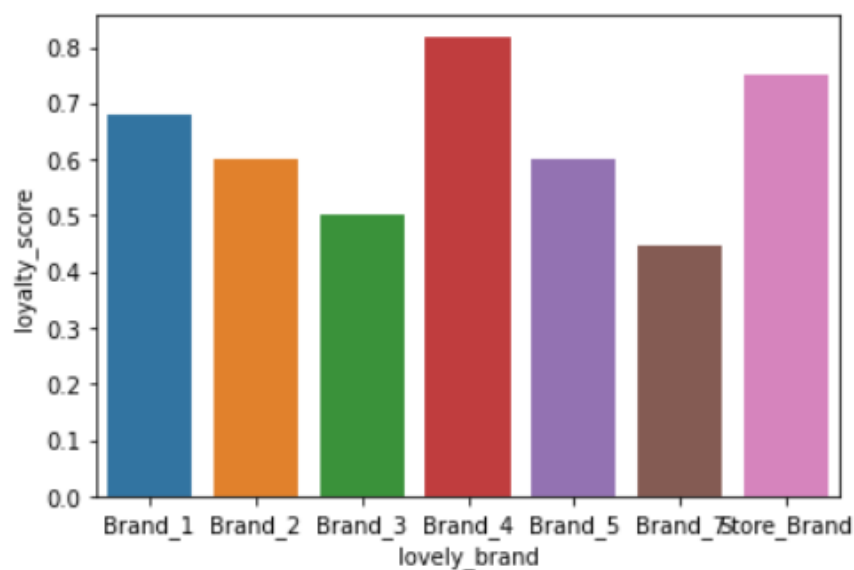


Рисунок 31 – График «*barplot*»

Видим, что у *Brand_4* самый высокий показатель лояльности. Также мы можем посчитать количество пользователей, для которых бренд оказался лояльным.

```
In [41]:
ax = sns.barplot(x="lovely_brand", y="user_id", data=brands_loyalty)
```

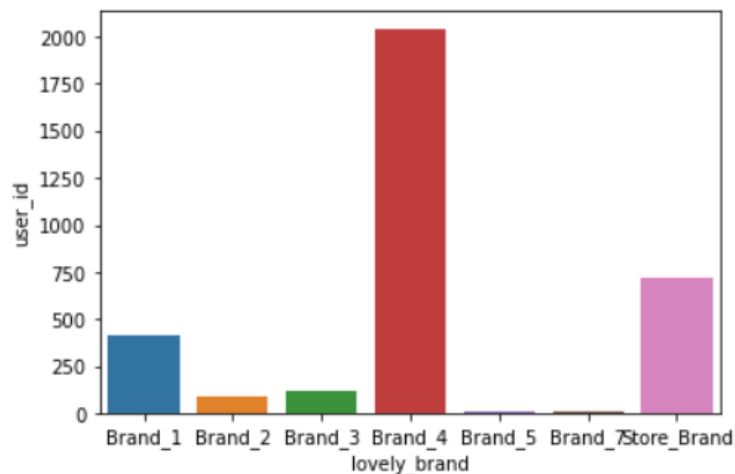


Рисунок 32 – График «*barplot*»

Теперь мы видим, что *Brand_4* в целом практически для подавляющего числа пользователей является лояльным из тех пользователей, кто часто покупает. При этом у других брендов, по сравнению с ним, довольно мало пользователей, которые выбрали его в качестве оптимального. Также создавать графики можно с помощью библиотек *pandas* и *matplotlib*. Самый простой способ визуализировать данные – вызвать метод *plot* у датафрейма (или его колонки). А через *matplotlib* можно нарисовать что угодно, но часто на это уходит слишком много строк кода, и её в основном используют для тонкой настройки графиков и их сохранения.

Практические задания

1. Проанализируйте данные о бронировании отелей.

a) Импортируйте библиотеку *pandas* как *pd*. Загрузите датасет *bookings.csv* с разделителем `;`. Проверьте размер таблицы, типы переменных, а затем выведите первые 7 строк, чтобы посмотреть на данные.

b) Приведите названия колонок к нижнему регистру и замените пробелы на знак нижнего подчеркивания.

c) Пользователи из каких стран совершили наибольшее число успешных бронирований? Укажите топ-5.

d) На сколько ночей в среднем бронируют отели разных типов?

e) Иногда тип номера, полученного клиентом (*assigned_room_type*), отличается от изначально забронированного (*reserved_room_type*). Такое может произойти, например, по причине овербукинга. Сколько подобных наблюдений встретилось в датасете?

f) На какой месяц чаще всего успешно оформляли бронь в 2016? Изменился ли самый популярный месяц в 2017?

g) Сгруппируйте данные по годам и проверьте, на какой месяц бронирования отеля типа City Hotel отменялись чаще всего в каждый из периодов.

h) Посмотрите на числовые характеристики трёх переменных: adults, children и babies. Какая из них имеет наибольшее среднее значение?

i) Создайте колонку total_kids, объединив children и babies. Отели какого типа в среднем пользуются большей популярностью у клиентов с детьми?

j) Создайте переменную has_kids, которая принимает значение True, если клиент при бронировании указал хотя бы одного ребенка (total_kids), и False – в противном случае. Посчитайте отношение количества ушедших пользователей к общему количеству клиентов, __ выраженное в процентах (churn rate). Укажите, среди какой группы показатель выше.

Описание данных:

Hotel – тип отеля (City Hotel или Resort Hotel).

Is canceled – бронирование было отменено (1) или нет (0); неотменённое считается успешным.

Lead time – количество дней, прошедших между датой бронирования и датой прибытия.

Arrival full date – полная дата прибытия.

Arrival date year – год прибытия.

Arrival date month – месяц прибытия.

Arrival date week number – номер недели прибытия.

Arrival date day of month – день прибытия.

Stays in weekend nights – количество выходных (суббота или воскресенье), которые гость забронировал для проживания в отеле.

Stays in week nights – количество дней (с понедельника по пятницу), которые гость забронировал для проживания в отеле.

Stays total nights – общее число забронированных ночей (сумма двух предыдущих колонок).

Adults – число взрослых.

Children – число детей.

Babies – число младенцев.

Meal – выбранный тип питания.

Country – страна происхождения клиента.

Reserved room type – тип зарезервированного номера.

Assigned room type – тип полученного номера (может отличаться от забронированного).

Customer type – тип бронирования.

Reservation status – значение последнего статуса брони:

Canceled – было отменено клиентом;

Check-Out – клиент зарегистрировался, но уже покинул отель;

No-Show – клиент не зарегистрировался и сообщил администрации отеля причину.

Reservation status date – дата обновления статуса.

2. Загрузите два датасета user_data .csv и logs.csv.

a) Импортируйте библиотеку pandas как pd. Проверьте размер таблицы, типы переменных, наличие пропущенных значений, описательную статистику.

b) Какой клиент совершил больше всего успешных операций? (success == True)

c) С какой платформы осуществляется наибольшее количество успешных операций?

d) Какую платформу предпочитают премиальные клиенты?

e) Визуализируйте распределение возраста клиентов в зависимости от типа клиента (премиум или нет)

f) Постройте график распределения числа успешных операций

g) Визуализируйте число успешных операций, сделанных на платформе computer, в зависимости от возраста, используя sns.countplot (x – возраст, y – число успешных операций). Клиенты какого возраста совершили наибольшее количество успешных действий?

Описание данных:

user_data:

client – идентификатор пользователя;

premium – является ли клиент премиальным;

age – возраст.

logs:

client – идентификатор пользователя;

success – результат (успех – 1, нет – 0);

platform – платформа;

time – время в формате Unix.

3. Управление социального обеспечения США выложило в Сеть данные о частоте встречаемости детских имен за период с 1880 года по 2010 год: babynames.zip.

а) наглядно представьте долю младенцев, получавших данное имя (совпадающее с вашим или какое-нибудь другое) за весь период времени;

б) определите относительный ранг имени;

с) найдите самые популярные в каждом году имена или имена, для которых фиксировалось наибольшее увеличение или уменьшение частоты;

д) проанализируйте тенденции выбора имен: количество гласных и согласных, длину, общее разнообразие, изменение в написании, первые и последние буквы;

е) проанализируйте внешние источники тенденций: библейские имена, имена знаменитостей, демографические изменения.

Контрольные вопросы

1. Чем отличаются серии и датафреймы?
2. Расскажите два способа как можно обратиться к столбцу датафрейма?
3. Какие дополнительные аргументы есть у метода считывания csv файла?
4. Как отсортировать данные по убыванию?
5. Колонки какого типа данных задействованы в методе describe()?
6. Что происходит при вызове функции groupby()?
7. Как можно объединить два датафрейма? Какие виды объединения бывают?
8. Что такое медиана?
9. Зачем нужны метрики?
10. Какие библиотеки можно использовать для визуализации данных? Какую строчку необходимо выполнить, чтобы в юпитер ноутбуке отображались графики?

Лабораторная работа №8. Работа с API

Цель:

Научиться работать с API, на примере API ВКонтакте.

Теоретический материал

API (Application Programming Interface – программный интерфейс приложения, или интерфейс программирования приложений) – специальный протокол для взаимодействия компьютерных программ, который позволяет использовать функции одного приложения внутри другого [17].

API значительно облегчает выполнение задач. По сути, это библиотека от создателей веб-сервиса (сайта, где можно что-то сделать), позволяющая быстро выполнить действия с этим сервисом.

Аналитики должны уметь постоянно обогащать свои данные той информацией, которая им необходима. Эту информацию вы можете найти на просторах интернета. Это могут быть данные с соц.сетей, или счетчика web- аналитики, например Яндекс.Метрика, или просто справочные данные – курс валюты, ВВП в определенный момент. В общем, данных огромное количество и нужно уметь к ним подключаться и использовать эти данные. Чтобы подключаться к данным как раз и используют API.

API для аналитика это инструмент связи с данными, которые находятся внутри сторонних сайтов. Через API мы передаем команды-запросы, а взамен получаем ответ. Все API разные, но есть общие подходы:

- у большинства API есть адрес, по которому нужно отправить запрос;
- в документации указаны определенные требования к структуре запроса.

API бывают открытыми и закрытыми. Открытые – общедоступные, а закрытые – требуют разнообразных процедур авторизации.

Что можно получить от API сайтов?

Чаще всего возвращают данные в формате json. Json - текстовый формат данных. Он используется для пересылки данных между веб-сервисами, и очень похож на словари в Python.

Requests – библиотека инструментов для обмена информацией по интернету. С ее помощью можно легко и быстро подключаться к сайтам, забирать оттуда информацию, отправлять туда какую-то информацию (например, логин и пароль для авторизации). В работе аналитика эта библиотека чаще всего используется для получения данных из сети.

Для этого используется функция `get()`, которая получает на вход URL API и параметры запроса в форме словаря.

API Вконтакте

API Вконтакте позволяет получать информацию из базы данных vk.com

Документация VK API : <https://dev.vk.com/api/getting-started>.

Первые шаги:

1. Получение токена.

Для этого переходим по ссылке <https://vkhost.github.io/>. Нажимаем на Настройки, выбираем Тип токена – Пользователь, и необходимые права доступа. Разрешаем доступ, и копируем и сохраняем `access_token` из адреса страницы.

Не передавайте токены другим людям и не публикуйте их в интернете. Токен имеет такую же силу, как логин и пароль. Владея токеном, можно получить доступ к вашему аккаунту и совершать действия от вашего имени.

2. Выбор нужного метода из документации.

3. Получение информации об ограничениях.

Самым главным ограничением является частота обращения к API – не более 3 запросов в секунду. Отдельные методы имеют свои ограничение на количество передаваемых данных в параметрах или на объем получаемых данных. В случае превышения лимита API будет выдавать специфические ошибки, которые можно найти в документации.

4. Создание запроса из нашего кода.

Любой запрос к API ВКонтакте состоит из 4 частей:

- метод API(обязательно);
- параметры (опционально);
- `access_token` (обязательно);
- версия API (обязательно).

Мы будем использовать библиотеку `requests` для GET-запросов, метод API будет частью URL-адреса, а токен, версия API и опциональные параметры будут передаваться в словаре `params`.

Отправим сообщение себе или другому пользователю:

```
In [1]: import requests
my_id = user_id
token = your_token
url = 'https://api.vk.com/method/messages.send'
params = {
    'user_id': my_id,
    'access_token': token,
```



```

        'random_id' : 0,
        'message' : 'Привет',
        'v' : '5.131'
    }
    res = requests.get(url, params=params)
    res.json()

```

Задача

Мы хотим разместить рекламу о курсе по анализу данных в тематических сообществах. Нужно получить список 100 самых посещаемых групп/страниц по поисковому запросу «python» и дополнительную информацию о них: название, количество пользователей, количество постов.

Для это нам сначала нужно понять, какие методы нам потребуются. В документации мы нашли методы *groups.search*, *groups.GetByld*, *wall.get*. Далее ищем группы. Задача поиска групп по разным запросам кажется довольно универсальной, поэтому напомним функцию, которая будет находить группы по поисковому запросу при помощи метода *groups.search*.

```

In [2]:
def search_query(q, sorting = 0):
    # Параметры sort
    #0 – сортировать по умолчанию (аналогично результатам поиска в
        полной версии сайта);
    #6 – сортировать по количеству пользователей.
    params = {
        'q' : q, #Поисковый запрос
        'access_token' : token,
        'v' : '5.131',
        'sort' : sorting,
        'count' : 100 # Просим отдать нам 100 записей, это
                        максимум для обычного запроса

    req = requests.get('https://api.vk.com/method/groups.search',
                        params).json()
    req = req['response']['items']
    return req
python_communities = search_query('python', 6)
group_df = pd.DataFrame(python_communities)
group_df.head()

```

	id	name	screen_name	is_closed	type	is_admin	is_member	is_advertiser
0	3183750	Веб программист - PHP, JS, Python, Java, HTML 5	php2all	0	page	0	0	0
1	42565717	Python	club42565717	0	group	0	0	0
2	152111071	Python	python_django_programirovanie	0	group	0	0	0
3	11899736	Видеоуроки программиста PHP, Javascript, Python, C#	program1	0	page	0	0	0
4	142410745	Python 3.10 ЯПрограммист	open_sourcecode	0	page	0	0	0

Рисунок 33 – Вывод датафрейма *group_df*

Найдем количество подписчиков для каждой группы. Запросим количество подписчиков всех выбранных групп одним запросом через метод *groups.GetById*. Для этого нам нужно передать в параметры запроса строку с ID групп. Создадим ее для начала.

```
In [3]:
groups_ids = '' # Сформируем строку с ID групп через запятую
for x in group_df['id']:
    groups_ids += str(x) # Преобразуем ID из чисел в строку
    groups_ids += ', '
Out[3]: '3183750, 42565717, 152111071, 11899736, 142410745, 178774705,
38080744, 55702386, 172288069, ... '
```

Мы подготовили список id групп. Теперь нужно дописать запрос к API ВК, который вернет данные с описанием *description*, и количеством подписчиков *members_count*. Создадим датафрейм с данными количества подписчиков в переменной *groups_members_df* и отсортируем по количеству подписчиков.

```
In [4]:
url_data = 'https://api.vk.com/method/groups.getById'
params = {
    'access_token' : token,
    'v' : '5.131',
    'group_ids' : groups_ids, # список ID групп
    'fields' : 'members_count,description' # здесь мы передали
        дополнительные параметры групп
}
req = requests.get(url_data, params)
res = req.json()['response']

groups_members_df = pd.DataFrame(res).sort_values('members_count',
        ascending=False)
groups_members_df.head()
```

	id	description	members_count	name	screen_name	is_closed	type	is_admin	is_member	is_advertiser
0	3183750	Самое крупное сообщество веб-программистов ВКО...	103577	Веб-программист - PHP, JS, Python, Java, HTML 5	php2all	0	page	0	0	0
1	42565717		61011	Python	club42565717	0	group	0	0	0
2	152111071	python django flask\n\nPython — высокоуровневы...	47184	Python python_django_programirovanie		0	group	0	0	0
3	11899736	Видеоуроки по PHP, JS, Mysql, Python, Java ...	40568	Видеоуроки программиста PHP, Javascript, Python, C#	programl	0	page	0	0	0
5	178774705		40037	Языки программирования: Python, Java, JS, PHP...	pryaz	0	page	0	0	0

Рисунок 34 – Вывод датафрейма *groups_members_df*

Далее получим количество постов в каждой группе. Информацию о постах можно получить из метода *wall.get*.

Сформируем запрос:

```
In [5]:
url_data = 'https://api.vk.com/method/wall.get'
params = {
    'access_token' : token,
    'v' : '5.131',
    'owner_id' : '-' + str(3183750), # Возьмем какой-нибудь ID
    'filter' : 'all' # Берем все посты
}
req = requests.get(url_data, params)
res = req.json()
res['response']['count']
Out[5]: 57129
```

Теперь нам нужно применить этот запрос к каждой группе. Сделаем это в цикле. Так как в цикле мы будем отправлять много запросов, нужно подготовиться и защитить код. Для этого мы импортируем библиотеку `time`, в которой есть функция `time.sleep()`, а в скобках передадим ей 0.5, то есть один запрос в полсекунды.

```
In [6]: import time
post_counts = [] # в этот массив будем последовательно записывать
информацию о количестве записей в каждой группе
for group_id in groups_members_df['id']:
    url_data = 'https://api.vk.com/method/wall.get'
    params = {
        'access_token' : token,
        'v' : '5.131',
        'owner_id' : '-' + str(group_id),
        'filter' : 'all'
    } # Берем все посты
    req = requests.get(url_data, params)
    res = req.json()
    try:
        count = res['response']['count']
        post_counts.append(count)
    except:
        post_counts.append(0)
        time.sleep(0.5)
```

Забираем нужные колонки и добавляем новую колонку с количеством постов.

```
In [7]:
groups_members_df = groups_members_df[['id', 'description',
                                         'members_count', 'name', 'screen_name']]
groups_members_df['posts'] = post_counts
groups_members_df.sort_values('posts', ascending=False).head(10)
```

	id	description	members_count	name	screen_name	posts
24	3183750	Самое крупное сообщество веб-программистов вко...	103574	Веб программист - PHP, JS, Python, Java, HTML 5	php2all	57128
26	11899736	Видеоуроки по PHP, JS, Mysql, Python, Java ...	40566	Видеоуроки программиста PHP, Javascript, Python, C#	programl	27841
19	2579743	Python (Питон, пайтон) — высокоуровневый язык ...	21060	Python	python.payton.python	26827
0	152111071	python django flask\n\nPython — высокоуровневы...	47182	Python	python_django_programirovanie	25001
16	38080744		34376	Python community	python_community	11907
28	194576836	Сообщество канала PyLounge. Лучший источник по...	5071	PyLounge - программирование на Python и всё о IT	pylounge	4930
14	55702386	Чудеса случаются, это Вам любой программист ск...	29303	Создание Сайтов PHP, JS, HTML5, CSS, Python	webprogramist	4890
25	69108280	Разработка и создание сайтов, интернет приложе...	19364	Python/Django Pirsipy	pirsipy	3544
10	149218373	Сообщество программистов python.	14379	Python/Django	we_use_django	2666
2	52104930	Русскоязычное Python сообщество.	11270	Python	we_use_python	2398

Рисунок 35 – Вывод датафрейма *groups_members_df*

Практические задания

1. Написать функцию прогнозирования возраста пользователя по возрасту его друзей, для этого необходимо получить список всех друзей, отфильтровать тех, у кого возраст не указан или указаны только день и месяц рождения, извлечь год рождения и усреднить.

2. Одной из задач при анализе социальных сетей является построение и анализ социального графа, то есть графа, «узлы которого представлены социальными объектами, такими как пользовательские профили с различными атрибутами, сообщества, медиаконтента и так далее, а рёбра – социальными связями между ними». Необходимо построить одну из разновидностей социального графа – эгоцентричный граф или граф друзей. Обычно под эгоцентричным графом понимают граф, в котором устанавливаются связи между друзьями некоторого пользователя. Для этого вам потребуется:

- сначала сделать запросы к методу `friends.getMutual`, который позволяет получить список общих друзей между парой пользователей;
- реализовать функции `ego_network()`, которая позволяет построить эгоцентричный граф друзей для указанного пользователя (по умолчанию текущего) и заданного множества его друзей (по умолчанию всех друзей).

Граф должен быть представлен в виде списка ребер:

```
def ego_network( user_id: tp.Optional[int] = None,
                friends: tp.Optional[tp.List[int]] = None
                ) -> tp.List[tp.Tuple[int, int]]:
    """
    Построить эгоцентричный граф друзей.
    :param user_id: Идентификатор пользователя, для которого строится граф
    :param friends: Идентификаторы друзей, между которыми
                    устанавливаются связи.
    """
    Pass
```

Контрольные вопросы

1. Что такое API?
2. Для чего создаются API?
3. Какие бывают API?
4. Какие преимущества API дают бизнесу?
5. Что такое токен и почему его нельзя никому передавать?
6. Какое основное ограничение использования API ВКонтакте? Что последует, если его не придерживаться?
7. Расскажите про библиотеку request.
8. В каком формате чаще всего приходит ответ от API?
9. Почему лучше использовать API, чем просто парсить сайт?
10. Из каких частей состоит любой запрос к API ВКонтакте?

СПИСОК ЛИТЕРАТУРЫ

1. What Does a Data Analyst Do? – URL: <https://brainstation.io/careerguides/what-does-a-data-analyst-do>.
2. What is Data Analysis: Methods, Process and Types Explained – URL: <https://www.simplilearn.com/data-analysis-methods-process-types-article>.
3. What Does a Data Analyst Do? 2022 Career Guide – URL: <https://www.coursera.org/articles/what-does-a-data-analyst-do-a-career-guide>
4. Зачем вам столько аналитиков: чем бизнес-аналитик отличается от системного и Data Analyst'a. – URL: <https://babok-school.ru/blogs/businessanalyst-vs-system-analyst/>
5. Data Analysis Skills. – URL: <https://brainstation.io/careerguides/what-skills-do-you-need-to-be-a-data-analyst>
6. The Role of Data Analysts in 2020 and Beyond. – URL: <https://quanthub.com/dataanalysts/#:~:text=WEF%20also%20found%20that%2096,in%2Ddemand%20jobs%20by%202022>.
7. The Future of Jobs Report 2020. – URL: <https://www.weforum.org/reports/the-future-of-jobs-report-2020#report-nav>.
8. Курс «Python для анализа данных» – URL: <https://skillfactory.ru/python-analytics>.
9. Ермаков, О. А. Python - как инструмент для анализа данных / О. А. Ермаков, Н. П. Брозгунова // Наука и Образование. – 2020. – Т. 3. – № 4. – С. 26. – EDN TYVCOP.
10. Jupyter Notebook для начинающих: учебник. – URL: <https://webdevblog.ru/jupyter-notebook-dlya-nachinajushhih-uchebnik/>.
11. Anaconda distribution. – URL: <https://www.anaconda.com/products/>
12. Python для сетевых инженеров. – URL: <https://pyneng.readthedocs.io/ru/latest/contents.html>
13. Маккини У. Python и анализ данных / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.
14. Object-Oriented Programming in Python. – URL: <https://pythontextbok.readthedocs.io/en/1.0/index.html>.
15. Python Modules and Packages – An Introduction. – URL: <https://realpython.com/python-modulespackages/#:~:text=Additionally%2C%20if%20you%20are%20developing,The%20import%20system>.
16. Усачева, А. Б. Библиотеки Python для анализа данных / А. Б. Усачева, Д. Р. Галиаскаров // Colloquium-journal. – 2020. – № 29-1(81). – С. 58-59. – EDN FIYAFU.

17. Как API-интерфейсы изменяют роль специалистов по data science? –URL: https://www.sas.com/ru_ru/insights/articles/analytics/apis-datascientists.html

18. Требования к компьютеру для Python. – URL: https://coddyschool.com/upload/files/Treb_Python.pdf

19. Python. – URL: <https://www.python.org/downloads>

20. 2022 Complete Python Bootcamp From Zero to Hero in Pythonudemy.– URL: <https://www.udemy.com/course/complete-python-bootcamp/>

21. Аналитик данных. – URL: https://hh.ru/search/vacancy?area=1&area=113&search_field=name&search_field=company_name&search_field=description&text=аналитик+данных