KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 2 урок > PYTHON

> Оглавление 2 урока

14. Создание функций
15. Время и погрешности в арифметике

> Библиотека (модуль/либа/пакет)

Библиотека — набор кода, посвящённый определённой цели (математические функции/работа в интернете/обработка текстов). Используется, чтобы облегчить работу, связанную с этой целью. В нашем курсе самый простой пример — библиотека `pandas`, которая используется для анализа данных.

Если есть библиотека, решающая ваша задачу, то почти всегда лучше воспользоваться ей, нежели решать всё самостоятельно с нуля.

По умолчанию содержимое библиотек недоступно при работе в python. Чтобы их использовать, необходимо их импортировать:

```
import pandas as pd
```

- `import` — ключевое слово, дающее питону понять, что мы собираемся импортировать библиотеку
- `pandas` — название библиотеки
- `as` — ключевое слово, означающее, что дальше будет использоваться другое название (элиас, alias)
- `pd` — то, какое название мы решили использовать в коде вместо названия этой библиотеки; они произвольны и обычно конвенциональны (в дальнейшем будем использовать название `pd`)

Более простой вариант импорта без изменения названия:

```
import module
```

`module` — название нужного вам модуля, далее в коде необходимо использовать его.

Существуют и другие способы импорта, но пока что остановимся на этих.

pandas

Библиотека для работы с данными. Без `pandas` анализ табличных данных в питоне совсем не то.

В следующих стэпах поговорим о его возможностях.

Документация

> Считывание csv

```
pd.read_csv('path_to_your.csv')  # read_excel for reading excel files
```

Считывает csv файл, который лежит по указанному в скобках пути. На Windows пути к файлам содержат символ `\`, который является специальным символом в строках во многих языках программирования, включая питон. Поэтому необходимо сделать следующее — либо удвоить все символы `\` в строке, содержащей путь, либо поставить `r` перед строкой:

- путь на windows - `C:\user\docs\Letter.txt`
- строка с путём - `'C:\user\docs\Letter.txt'`
- валидная строка с путём - `'C:\\user\\docs\\Letter.txt'` или `r'C:\user\docs\Letter.txt'`

На сервере мы работаем с Unix путями, например `/home/user/letter.txt`. С ними меньше таких проблем — достаточно поместить путь в кавычки, чтобы всё было хорошо.

Дополнительные аргументы функции read_csv

Аргументы (или параметры) — это настройки, которые мы можем задать для функции.

- **encoding** — параметр в `read_csv`, отвечает за кодировку текста, которая может быть различной. Самая распространённая — utf-8. Пример указания кодировки:

```
pd.read_csv('path_to_your.csv', encoding='Windows-1251')  # now you are reading file e
ncoded with Windows-1251
```

- **sep** — разделитель между ячейками в строке (по умолчанию `,`)

```
pd.read_csv('path_to_your.csv', encoding='Windows-1251', sep=';')  # now you additiona
lly specified that fields are separated with ;
```

- **parse_dates** — указывает, стоит ли воспринимать даты как даты? (по умолчанию они воспринимаются пандасом как строки)Параметр с датами может принимать несколько значений:
  - `True` — пытается перевести в дату первую колонку
  - список колонок — пытается перевести в дату указанные в списке колонки

```
# And create_data, payment_data columns will be treated as data
pd.read_csv('path_to_your.csv', encoding='Windows-1251', sep=';', parse_dates=['create
_data', 'payment_data'])
```

Документация

> Атрибуты vs Методы

У объектов есть и методы, и атрибуты. В чём же их различие? Прежде всего они отличаются внешним видом: после названия метода идут скобки, в которых могут быть аргументы, а после названия атрибутов скобок нет.

Методы являются функциями, связанными с объектом. То есть они что-то вычисляют/сохраняют/загружают.

Атрибуты — сохранённые признаки, связанные с объектом. Их значение сохранено и не вычисляется при вызове.

Пример с методом `head()` и атрибутом `shape`

```
df.head()
```

	seq_id	source	type	start	end	score	strand	phase	attributes
0	chr17	RefSeq	exon	6811330	6811453	.	+	.	Parent=XM_001475631.1
1	chr17	RefSeq	CDS	6808204	6808245	.	+	0	Name=CDS:NC_000083.5:LOC100040603;Parent=XM_00...
2	chr17	RefSeq	CDS	6811792	6811869	.	+	2	Name=CDS:NC_000083.5:LOC100040603;Parent=XM_00...
3	chr17	RefSeq	exon	6806513	6806553	.	+	.	Parent=XM_001475631.1
4	chr17	RefSeq	five_prime_UTR	6806513	6806526	.	+	.	Parent=XM_001475631.1

```
df.shape
```
```
(512, 9)
```

Больше информации

> Размеры датафрэйма

`shape` — атрибут, хранящий данные о размерах таблицы. Возвращает кортеж (для простоты — воспринимайте его как на неизменяемый список). В случае датафрэйма кортеж содержит 2 значения — число строк и число колонок в нём.

```
data.shape
```
```
(48129, 22)
```

Например, в данном случае размер таблицы: 48129 строк и 22 колонки.

Документация

Типы колонок

Чтобы узнать типы колонок в вашем датафрэйме, 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 датафрэймом (например, чтобы не произвести сложение строк, думая, что это числа).

Документация

> Метод describe

`describe` — удобный метод для вывода описания числовых колонок в датафрэйме:

	age
count	4.000000
mean	28.250000
std	14.103782
min	12.000000
25%	19.500000
50%	28.500000
75%	37.250000
max	44.000000

`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).

Документация

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`. Пример переименования колонки **x** в **name,** а колонки **y** в **salary**:

```
# Rename columns
df = df.rename(columns={'x': 'name', 'y': 'salary'})
```

А это пример переименования лэйблов строк из **0** в **Ivanov** и из **1** в **Vasilev**:

```
# Rename index (row names)
df = df.rename(index={0: 'Ivanov', 1: 'Vasilev'})
```

Один из способов переименования — передать словарь, в котором ключами являются старые названия, а значениями — новые.

Документация

> Обращение к колонкам

В пандасе существует множество способов обратиться к колонке датафрэйма. Самый удобный:

```
df.column_name
```

- `df` – датафрэйм
- `column_name` – название колонки

Чтобы это работало, название колонки должно состоять из одного компонента (например, слова), и не должно совпадать с названием методов датафрэйма (имя колонки `count` не сработает). 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']]
```

Документация

Создание колонки

Делается так же просто, как и задавание нового значения в словаре :)

```
# Create new column in the df with name new_column which is equal to 5 in each cell
df['new_column'] = 5
```

Больше информации

> Применение вычислительных методов

Существует набор методов, доступных для колонок датафрэймов. Например, есть колонка `money` в датафрэйме, содержащая полученные объёмы денег. Применив метод `sum`, можно посчитать их сумму.

```
df.money.sum()
69120
```

Ещё несколько примеров:

- `product` – перемножение
- `std` – среднеквадратичное отклонение
- `var` – дисперсия

Больше информации

Цепочка методов (method chaining)

Приём для объединения нескольких действий в одно. Большинство методов датафрэймов возвращают вам результат, который довольно часто тоже является датафрэймом. Следовательно, от него тоже можно вызвать метод.

Ванильная запись:

```
df = df.query('income >=  1000')
df = df.groupby(['title', 'status'], as_index=False).agg({'income': 'sum', 'id': 'coun
t'})  # groupby is usually immediately followed by agg
df = df.sort_values(['title', 'status'])
```

Сокращённая запись:

```
df  = df.query('income >=  1000').groupby(['title', 'status'], as_index=False).agg({'i
ncome': 'sum', 'id': 'count'}).sort_values(['title', 'status'])
```

Как можно заметить, эта запись довольно длинная и не очень удобная для чтения. Обычно, такая цепочка оформляется в блок, где каждый метод идёт на своей строке. Есть 2 варианта оформления, какой выбрать — вопрос предпочтения и конвенций в вашей организации:

```
# \ after each nonfinal line to demarcate line continuation for python
df  = df.query('income >=  1000') \
  .groupby(['title', 'status'], as_index=False) \
  .agg({'income': 'sum', 'id': 'count'}) \
  .sort_values(['title', 'status'])
```

```
# Parentheses around the whole expression for the same purpose as backslash in previou
s example

df  = (df.query('income >=  1000')
        .groupby(['title', 'status'], as_index=False)
        .agg({'income': 'sum', 'id': 'count'})
        .sort_values(['title', 'status']))
```

Больше информации

> Группировка

Часто используемый приём для вычисления чего-либо по данным. Осуществляется с помощью метода `groupby` – группирует данные в датафрэйме по указанным колонкам:

```
df.groupby('company')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7ff101fa8390>
```

Применение одного метода `groupby` не даёт видимого эффекта, хотя на самом деле все строки были объединены в группы по значению в колонке `company`: с одним значением в одну группу, с другим – в другую. `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`

Документация

Агрегация

`agg` – функция для агрегирования данных, применяется после группировки методом `groupby`.

Как это работает:

- агрегируем методом `agg`, указывая на каких колонках какие действия произвести.

```
product_data.groupby('title').agg({'money': 'sum'})
```

title	money
Курс обучения «Консультант»	208163.49
Курс обучения «Специалист»	160862.64
Курс обучения «Эксперт»	148992.80
Курс от Школы Диетологов. Бизнес	18752.54
Курс от Школы Диетологов. Повышение квалификации.	88384.92
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием	366947.20

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.

Документация

> 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):

```
product_data.sort_values('city')
```

	id	status	money	city
14	1062947	Завершен	5044.05	Massagno
238	1064114	Завершен	2935.44	a.s
77	1063615	Завершен	2914.30	Алматы
154	1063708	Отменен	0.00	Амурская область
134	1063686	Отменен	0.00	Армавир

Дополнительные параметры

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

Документация

value_counts

Метод, который считает, сколько раз встречается каждое уникальное значение переменной. Например, имеется следующий набор данных:

	name	year
0	Persik	2015
1	Persik	2016
2	Barsik	2018
3	Tolya	2018
4	NaN	2020

Посчитать, сколько раз встречается каждое имя (`name`), можно с помощью следующей команды:

```
df['name'].value_counts()
```

Результат возвращается в формате pd.Series (серии) :

```
Persik    2
Tolya     1
Barsik    1
Name: name, 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

Несколько примеров:

1) Получаем частоту встречаемости (напр. Persik – в 40% наблюдений), также не удаляем из результата NaN:

```
df['name'].value_counts(normalize=True, dropna=False)
```

```
Persik    0.4
Tolya     0.2
Barsik    0.2
NaN       0.2
Name: name, dtype: float64
```

2) Разбиваем `year` на 2 промежутка:

```
df['year'].value_counts(bins=2)
```

```
(2017.5, 2020.0]      3
(2014.994, 2017.5]    2
Name: year, dtype: int64
```

Документация

> 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

```
product_data.query("status == 'Завершен'")
```

	id	create_data	payment_date	title	status
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен
12	1062938	05.12.2019 12:07	22.12.2019 12:29	Курс обучения «Консультант»	Завершен

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'")
```

Документация

> Запись в файл

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

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

Дополнительные параметры

- `index` – записать индекс датафрэйма в csv как первую колонку
- `sep` – используемый при записи разделитель колонок

Документация

> Использованные функции

- `replace` – применяется к строкам, принимает 2 строки: что заменить и на чтоБольше информации

```
my_string = 'Крайне важное название'
my_string.replace(' ', '_')
```

```
'Крайне_важное_название'
```

- `strip` – применяется к строкам, по умолчанию убирает пробелы слева и справаБольше информации

```
user_name = '     Vasya Vedrov '
user_name.strip()
```

```
'Vasya Vedrov'
```

- `round` – применяется к дробным числам, округляет их, можно передать дополнительный аргумент, который означает число знаков после запятойБольше информации

```
round(4.45555, 2)
```

```
4.46
```

> Векторизация

Векторизация - это специальная техника, позволяющая в пандасе быстро выполнять в одну строчку операции, которые в чистом питоне требуют как минимум одного цикла. Быстрота связана с тем, что код пандаса реализован на более быстром чем питон языке, а в питоне просто представлены функции. Благодаря векторизации, мы можем делать различные операции со всеми колонками целиком, не отвлекаясь на итерирование по элементам

```
df.driver_score
```

```
0       5.0
1       4.0
2       5.0
3       5.0
4       0.0
       ...
7426    0.0
7427    5.0
7428    5.0
7429    5.0
7430    0.0
Name: driver_score, Length: 7431, dtype: float64
```

Умножим каждый элемент на 3

```
df.driver_score * 3
```

```
0       15.0
1       12.0
2       15.0
3       15.0
4        0.0
        ...
7426     0.0
7427    15.0
7428    15.0
7429    15.0
7430     0.0
Name: driver_score, Length: 7431, dtype: float64
```

Или выясним для каждого значения больше ли оно чем 3

```
df.driver_score > 3
```

```
0        True
1        True
2        True
3        True
4       False
        ...
7426    False
7427     True
7428     True
7429     True
7430    False
Name: driver_score, Length: 7431, dtype: bool
```

None

`None` это специальный тип данных в питоне, который имеет только одно одноимённое значение – `None`. Используется в тех случаях, когда нужно обозначить ничто. Обычно (но совсем необязательно) его возвращают функции, которые как-то изменяют данные

```
xs = [1, 2, 3]
a = xs.append(4)

print(xs)
[1, 2, 3, 4]

print(a)
None
```

Толку от присвоения выше (`a = xs.append(4)`) нет, просто постепенно запоминайте такие функции, и не перезаписывайте ваши данные вызовами типа

```
xs = xs.append(4)
```

Документация

> Свои функции

Служат для переиспользования кода – вы написали изумительный скрипт, который решает 20% вашей работы. Теперь вам нужно использовать его каждый день на новых данных. Чтобы это упростить, существуют функции. Они позволяют свести весь код к 1-ой строке и менять название поступающих файлов и других параметров только в 1-ом месте.

Задавание функций

```
def function(parameter):
    what to do
```

- `def` – ключевое слово, даёт питону понять, что дальше будет задана функция
- `function` – название, которое мы выбрали
- `parameter` – инпут функции, то, какие данные ей нужны для работы. Параметров может быть 0 (без инпута) или больше. Это может быть почти что угодно:
  - путь к файлу
  - название папки
  - число строк, с которыми нужно работать
  - путь к отчёту, который будет создан
- `what to do` – тело функции, функционал, который будет работать

По умолчанию функция при завершении своей работы ничего не вернёт обратно (в отличие от, например, метода `head` у датафрэйма). Чтобы она возвращала значение после вызова, добавьте ключевое слово `return` в теле функции перед переменной, которую хотите вернуть:

```
# 2 simple functions# Without returndef print_a(a, b):
    print(a)

# With returndef return_a(a, b):return a

x = print_a(3, 5)  # x is None3

y = return_a(3, 5)  # y is 3
```

Больше информации

> Время

Для работы с датой и временем можно использовать модуль `datetime`. Для получения данных о времени в момент вызова функции используйте функцию `today` в одноимённом подмодуле:

```
import datetime  # by convention imports are placed in the head of file and separa
ted with 2 blank lines from other code


date = datetime.datetime.today()
```

Само по себе это даст вам специальный тип даты. Чтобы перевести его в строку сделайте следующее:

```
datetime.datetime.today().strftime('%Y-%m-%d-%H:%M:%S')
'2020-01-30-00:07:12'
```

`strftime` форматирует дату по переданному ему формату:

- % – обозначает что дальше будет часть даты
- Y – год 4-мя знаками
- m – месяц 2-мя знаками

- d – день
- H – час
- M – минуты
- S – секунды

Можно использовать только часть фрагментов даты, разделители между ними – на ваше усмотрение (в примере это `-` и `:`). Немного примеров

```
from datetime import datetime


# current date and time
now = datetime.now()
print(f'Full time format of now is {now}')
Full time format of now is 2020-06-01 17:54:40.010540
```

```
# Year
year = now.strftime("%Y")
print("year:", year)
year: 2020
```

```
# Month
month = now.strftime("%m")
print("month:", month)
month: 06
```

```
# Day
day = now.strftime("%d")
print("day:", day)
day: 01
```

```
# Time
time = now.strftime("%H:%M:%S")
print("time:", time)
time: 17:54:40
```

```
# Date and time
date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
```

```
print("date and time:",date_time)
date and time: 06/04/2020, 17:54:40
```

Документация

Погрешность арифметики

В компьютере используется двоичная система счисления, в которой не выразить точно любое десятичное число. Из-за этого при выполнении действий может накапливаться ошибка. Обычно это не страшно (она в порядках меньше -5-ого), но бывает нужна точность. Для этого есть специальные библиотеки

Документация