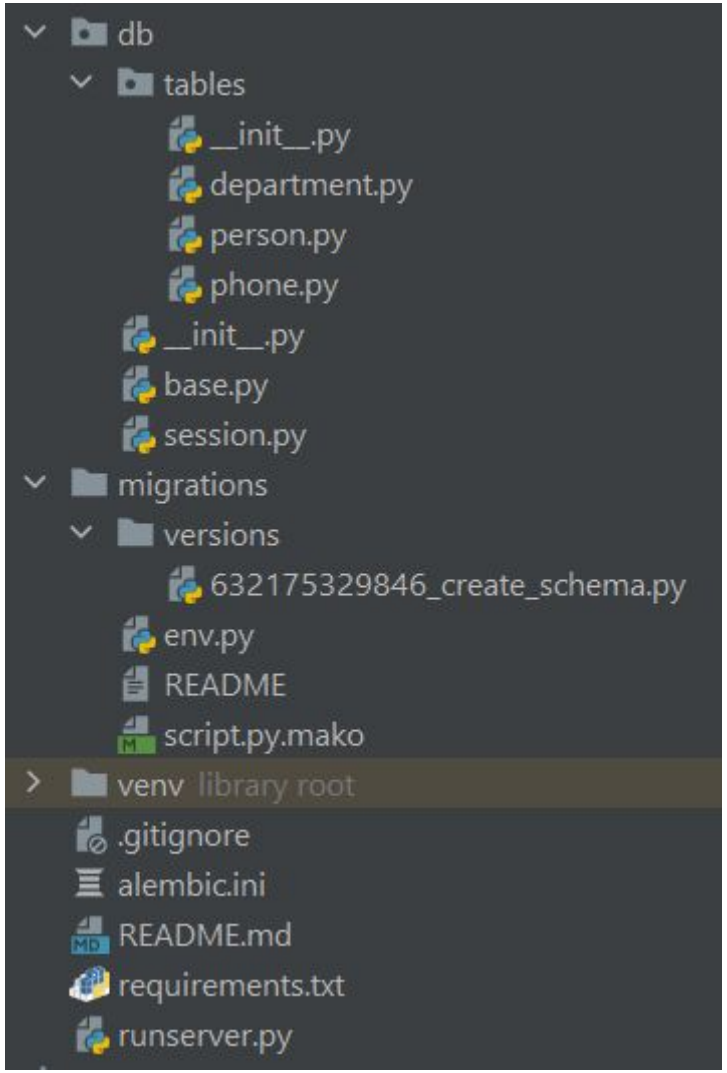
# Импорты

как подключать свой и чужой код в проект

# Модули и пакеты

В Python свой код мы организуем следующим образом:

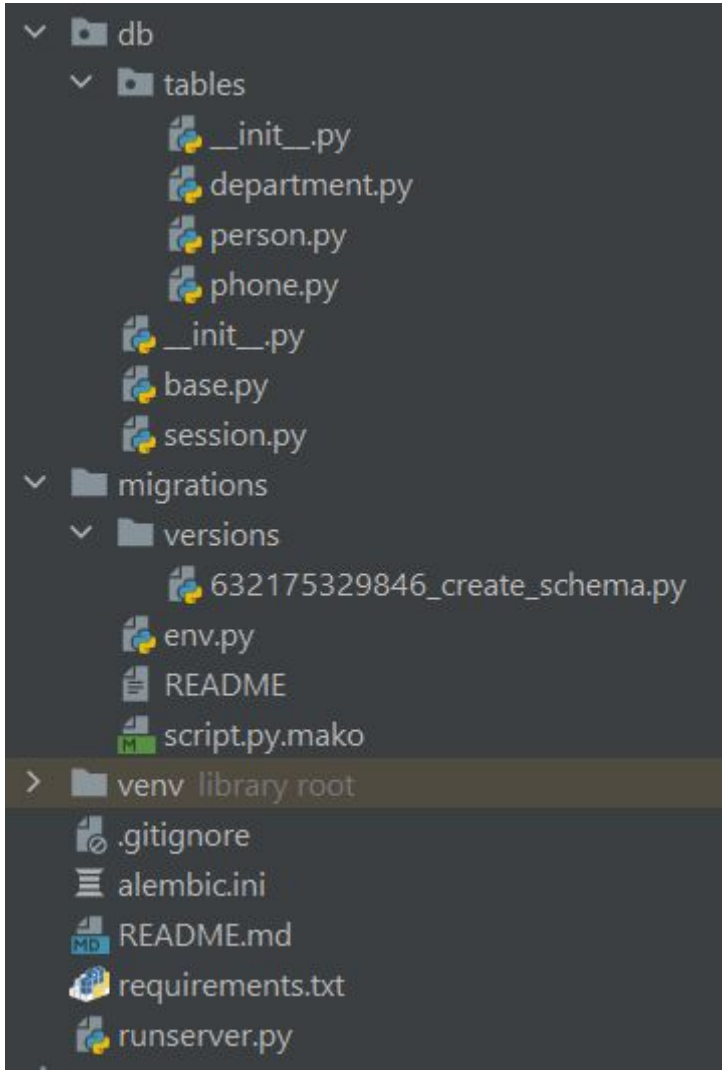
- Инструкции, функции, классы упаковываем в модули
- Модули упаковываем в пакеты



# Модули

**Модуль** – это всего лишь файл, содержащий код на Python.

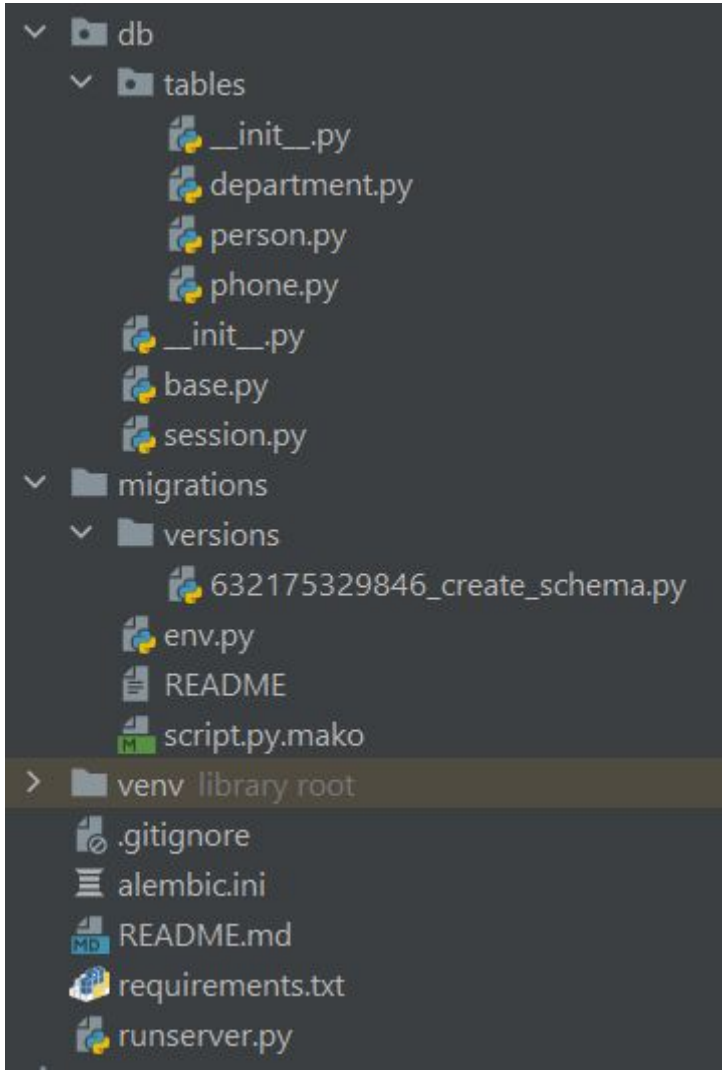
При импорте модуля Python выполняет весь код в нём



# Пакеты

Пакеты – директория, которая содержит некоторое количество файлов с кодом на Python и опциональный файл «__init__.py»

- При импорте пакета Python выполняет код в файле пакета __init__.py (если есть). Все объекты, определённые в модуле или __init__.py, становятся доступны импортирующему
- Имя пакета — имя директории



# Файл __init__.py

У файла __init__.py есть две функции:

- Превратить папку со скриптами в импортируемый пакет модулей (до Python 3.3). С версии Python 3.3 любая директория (даже без __init__.py) считается пакетом.
- Выполнить код инициализации пакета

```
"""Include imports from the sqlalchemy.dialects package for backwards
compatibility with pre 0.6 versions.

"""
from ..dialects.firebird import base as firebird
from ..dialects.mssql import base as mssql
from ..dialects.mysql import base as mysql
from ..dialects.oracle import base as oracle
from ..dialects.postgresql import base as postgresql
from ..dialects.sqlite import base as sqlite
from ..dialects.sybase import base as sybase


postgres = postgresql


__all__ = (
    "firebird",
    "mssql",
    "mysql",
    "postgresql",
    "sqlite",
    "oracle",
    "sybase",
)
```

# Оператор import

Для того, чтобы подключить сторонний код в модуль или пакет, используется ключевое слово:

- import [модуль], где модуль – это имя другого файла Python без расширения «.py».

```
1  import collections
2  import types
3  import weakref
```

# Импортируем модуль с другим именем

Если есть другой модуль с таким же именем или хочется использовать более короткое или простое имя, то модулю можно назначить псевдоним (alias).

Для создания псевдонима используется оператор «**as**»

```
1  import datetime
2  import sys
3
4  from sqlalchemy import types as sqltypes
5  from sqlalchemy import util
6  from sqlalchemy.dialects.mssql.base import MSDateTime
7  from sqlalchemy.dialects.mssql.base import MSDialect
```

# Импортируем только то, что нужно

В Python можно импортировать не весь код, а только некоторую часть. Каждая часть может иметь свое оригинальное имя или ей можно назначить alias. Для импорта только необходимого, используется конструкция:

- from [модуль|пакет] import [что-то] [as псевдоним]

```
1  import datetime
2  import sys
3
4  from sqlalchemy import types as sqltypes
5  from sqlalchemy import util
6  from sqlalchemy.dialects.mssql.base import MSDateTime
7  from sqlalchemy.dialects.mssql.base import MSDialect
```

# Вопрос-ответ

? Как я могу исследовать содержимое импортируемого модуля?

✔ Для того, чтобы увидеть, что мы импортируем – можно воспользоваться функцией dir()

```
number = [1, 2, 3]
print(dir(number))

print('\nReturn Value from empty dir()')
print(dir())

"""
Output
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__',
'__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

Return Value from empty dir()
['__annotations__', '__builtins__', '__doc__', '__loader__',
'__name__', '__package__', '__spec__', 'number']
"""
```

# Какие бывают импорты в Python

Импорты в Python (да и не только в Python) бывают следующими:

- Абсолютные: используется полный путь (корневой папки проекта) к желаемому модулю
- Относительные: используется относительный путь (начиная с пути текущего модуля) к желаемому модулю

```
import operator
import weakref

from sqlalchemy.util.compat import inspect_getfullargspec
from . import base
from .. import exc as sa_exc
from .. import util
from ..sql import expression
```

# Относительные импорты

При относительном импорте используется **только** конструкция:

- from .[модуль|пакет] import [что-то], где символы точки показывают, на сколько директорий «вверх» нужно подняться. «.» — текущая директория, «..» — на одну директорию выше

```
from . import strategies
from . import util  # noqa
from .base import Connection  # noqa
from .base import Engine  # noqa
from .base import NestedTransaction  # noqa
from .base import RootTransaction  # noqa
from .base import Transaction  # noqa
from .base import TwoPhaseTransaction  # noqa
from .interfaces import Compiled  # noqa
from .interfaces import Connectable  # noqa
from .interfaces import CreateEnginePlugin  # noqa
from .interfaces import Dialect  # noqa
from .interfaces import ExceptionContext  # noqa
from .interfaces import ExecutionContext  # noqa
from .interfaces import TypeCompiler  # noqa
from .result import BaseRowProxy  # noqa
from .result import BufferedColumnResultProxy  # noqa
from .result import BufferedColumnRow  # noqa
from .result import BufferedRowResultProxy  # noqa
from .result import FullyBufferedResultProxy  # noqa
from .result import ResultProxy  # noqa
from .result import RowProxy  # noqa
from .util import connection_memoize  # noqa
from ..sql import ddl  # noqa
```

# if __name__ == "__main__"

Скрипт может выполняться и самостоятельно, и может быть импортирован другим скриптом. Так как при импорте скрипта выполняются все инструкции, то надо указать, какие-то строки не должны выполняться при импорте.

Все строки, которые находятся в блоке if __name__ == "__main__" не выполняются при импорте.

Условие if __name__ == "__main__" проверяет, был ли файл запущен напрямую (python some_script.py)

```
1  def some_function(name):
2      print(f"Hi {name}")
3
4
5  if __name__ == "__main__":
6      name = input("Put your name: ")
7      some_function(name)
```

# if __name__ == "__main__"

__name__ == "__main__" только если файл запускается как основная программа (точка входа в проект), и выставляется равной имени модуля при импорте модуля.

Так как относительные импорты завязаны на __name__, а если мы запускаем текущий файл, то __name__ будет равна "__main__", то в данном случае можно использовать только абсолютные импорты

```python
# Можно
import pathlib
import my_module
from db import connection
from some_package.some_module import SomeClass
# Нельзя
from . import some_module
from .some_module import SomeClass
from .. import some_package
from ..some_package import some_module
from ..some_package.some_module import AnotherClass


if __name__ == "__main__":
    name = input("Put your name: ")
    print(f"Hi {name}")
```

# Где Python ищет модули

Что происходит «под капотом», когда Python ищет модули:

1. При импорте интерпретатор сначала ищет встроенный модуль с таким именем.
2. Если такого модуля нет, то идёт поиск в списке директорий, определённых в переменной sys.path. sys.path инициализируется из следующих мест:
   - директории, содержащей исходный скрипт (или текущей директории, если файл не указан)
   - директории по умолчанию, которая зависит от дистрибутива Python
   - PYTHONPATH (список имён директорий; имеет синтаксис, аналогичный переменной окружения PATH)

# sys.path

sys.path – list с путями, где могут находиться модули и пакеты.

- Программы могут изменять переменную sys.path после её инициализации
- Директория, содержащая запускаемый скрипт, помещается в начало поиска перед путём к стандартной библиотеке. Это значит, что скрипты в этой директории будут импортированы вместо модулей с такими же именами в стандартной библиотеке

```
>>> import sys
>>> sys.path
[
    '',
    'C:\\Users\\Slava\\AppData\\Local\\Programs\\Python\\Python39\\python39.zip',
    'C:\\Users\\Slava\\AppData\\Local\\Programs\\Python\\Python39\\DLLs',
    'C:\\Users\\Slava\\AppData\\Local\\Programs\\Python\\Python39\\lib',
    'C:\\Users\\Slava\\AppData\\Local\\Programs\\Python\\Python39',
    'C:\\Users\\Slava\\AppData\\Roaming\\Python\\Python39\\site-packages',
    'C:\\Users\\Slava\\AppData\\Local\\Programs\\Python\\Python39\\lib\\site-packages'
]
>>>
```

# Указываем, что можно импортировать

При выполнении инструкции

- from [модуль|пакет] import *

выполняется и становится доступным весь код в модуле|пакете.

Для указания, что можно импортировать, следует на глобальном уровне определить переменную «**__all__**» и приравнять ей кортеж с именами

```
__all__ = (
    "AttributeExtension",
    "EXT_CONTINUE",
    "EXT_STOP",
    "EXT_SKIP",
    "ONETOMANY",
    "MANYTOMANY",
    "MANYTOONE",
    "NOT_EXTENSION",
    "LoaderStrategy",
    "MapperExtension",
    "MapperOption",
    "MapperProperty",
    "PropComparator",
    "SessionExtension",
    "StrategizedProperty",
)
```

# Резюмируя

- Все импорты, которые начинаются не с точки, считаются абсолютными
- Относительные импорты основаны на имени текущего модуля. Так как имя главного модуля всегда «__main__», модули, которые должны использоваться как главный модуль приложения, должны всегда использовать абсолютные импорты. Следовательно, как правило, абсолютные импорты предпочтительнее относительных. Они позволяют избежать путаницы между явными и неявными импортами. Кроме того, любой скрипт с явными относительными импортами нельзя запустить напрямую

# Ваши вопросы

что необходимо прояснить в рамках данного раздела

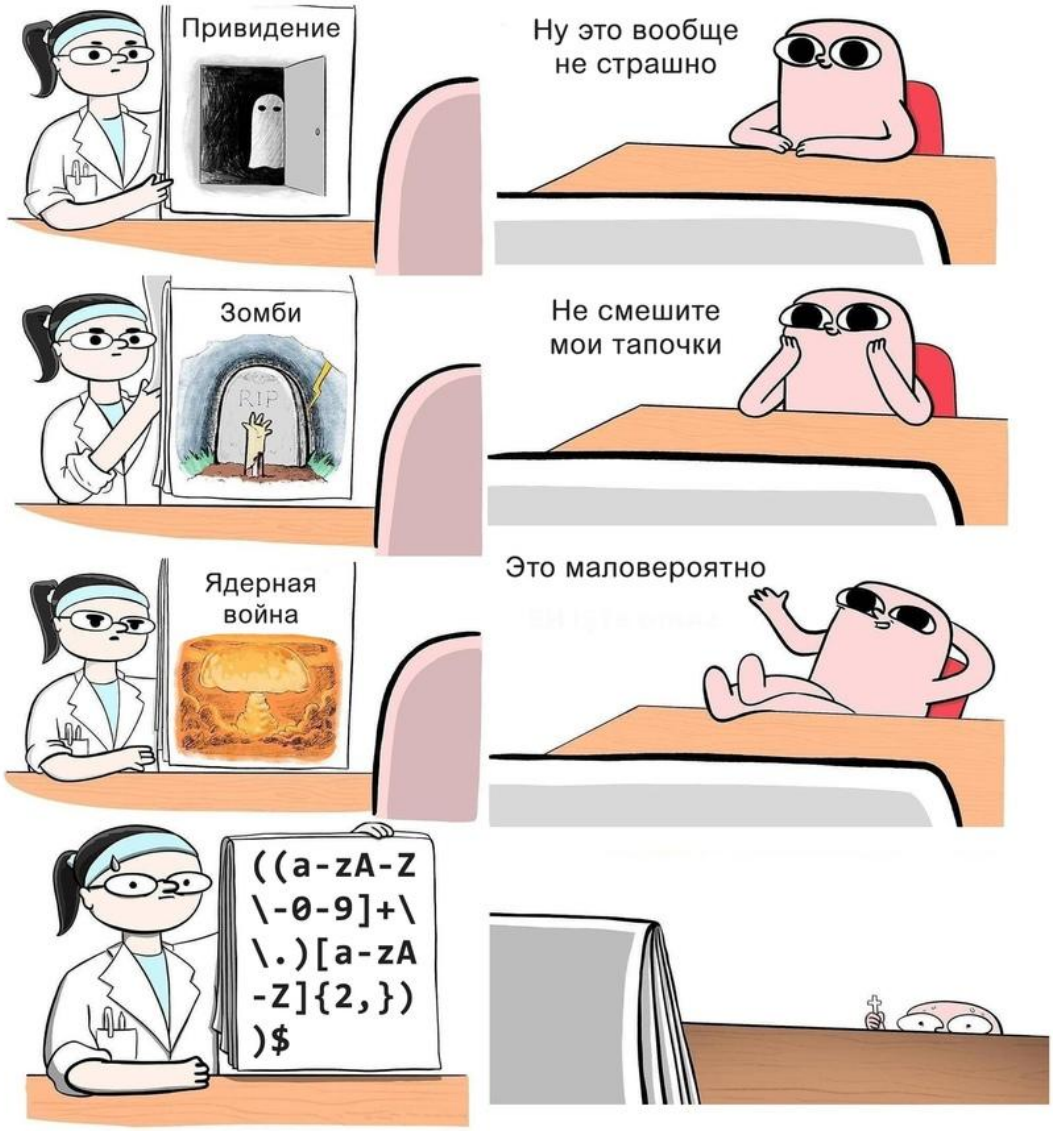
# Регулярные выражения

безграничные возможности, но ниже скорость

# Регулярные выражения

**Регулярные выражения** (англ. regular expressions или RegExp) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

По сути это строка-образец, состоящая из обычных символов (их называют литералами) и метасимволов, задающая правило поиска.



# Когда используются регулярные выражения

В народе их кличут «**регулярками**».

Примеры использования:

- Валидация данных (например, правильно ли заполнена строка email)
- Сбор данных (особенно веб-скрапинг)
- Обработка данных (преобразование сырых данных в нужный формат)
- Парсинг (например, достать все GET параметры из URL или текст внутри скобок)
- Замена данных в строке
- Подсветка синтаксиса и т.д.

# Плюсы и минусы регулярок

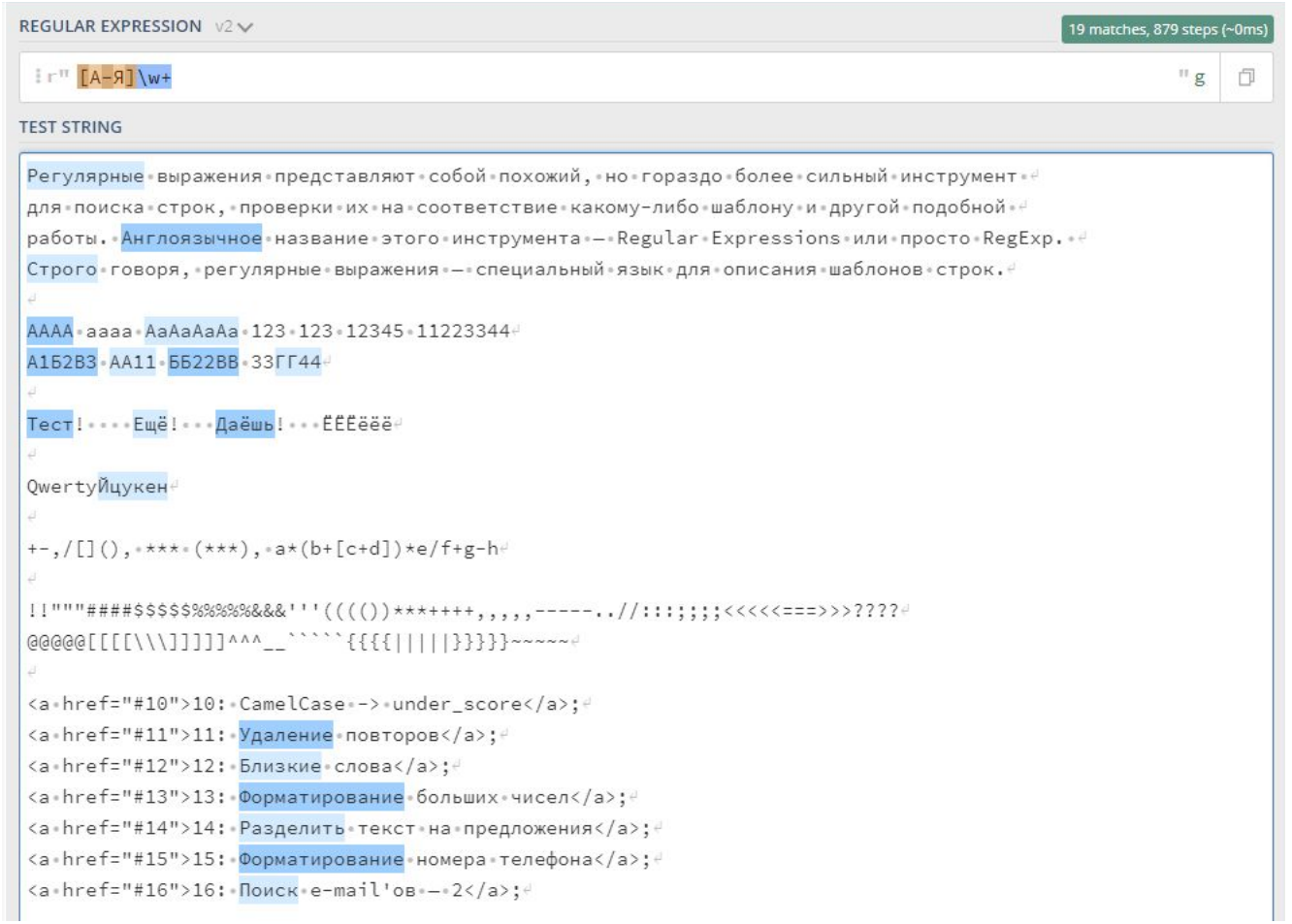
- ✔ Можно сделать практически все, что угодно со строками
- ✔ Сильные возможности поиска и замены
- ✔ Универсальны для разных языков программирования

- × Работают намного медленней, чем строковые методы (если есть возможность обойтись строковыми методами – стоит так и сделать)
- × Длинное выражение сложно понять и бывает практически невозможно в адекватное время проверить или найти ошибку

# Откроем песочницу

Для тестирования регулярных выражений будем использовать следующую песочницу:

- https://regex101.com/r/aGn8QC/2



# Примеры использования

| Регулярное выражение | Что делает |
|---|---|
| \d{1,2}\/\d{1,2}\/\d{4} | Дата (напр. 23.03.2009) |
| ^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$ | Email |
| ^[a-z0-9_-]{3,16}$ | Логин. Строка от 3х до 16 символов, которые могут быть: строчная буква (a-z), заглавные буквы(A-Z), число (0-9), подчеркивание или дефис |
| ^[A-Za-z0-9_-]{6,18}$ | Пароль. количество символов от 6 до 18 и присутствие заглавных букв A-Z |
| \#([a-fA-F]\|[0-9]){3, 6} | Шестнадцатеричный код цвета |
| (\<(/?[^\>])\>) | HTML-теги |
| ([^\s]+(?=\.(jpg\|gif\|png))\.\2) | Картинка с расширением jpg, gif или png |

# Спецсимволы, которые надо экранировать

Как и любой язык, регулярные выражения имеют спецсимволы, которые **нужно экранировать**:

Экранирование осуществляется обычным способом — **добавлением \ перед спецсимволом**.

| | | | |
|---|---|---|---|
| . | ^ | $ | * |
| + | ? | { | } |
| [ | ] | \ | \| |
| ( | ) | < | > |

# Шаблоны для одного символа

| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| . | Любой одиночный символ кроме \n | .ень | тень, лень, сирень |
| a\|b | Символ a **или** символ b | д\|т | дом, том, лом |
| \s | Любой пробельный символ (пробел, табуляция, конец строки и т.п.) ([\f\n\r\t\v]) | кот\sпес | кот пес, котопес |
| \S | Любой непробельный символ | кот\Sпес | кот пес, котопес |
| \d | Любая цифра ([0-9]) | \d\d\d | Агент 007 |
| \D | Любой символ, но не цифра ([^0-9]) | \D\D\D | Агент 007 |
| \w | Любая буква, цифра или «_» ([0-9a-zA-Za-яA-ЯёЁ]) | \w+ | Агент «007» |
| \W | Не буква, не цифра и не «_» | \W+ | Агент «007» |

# Кванторы (указание количества повторений)

| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| a? | Одно или ни одного вхождения «a» | дома? | дом, дома, домов |
| a* | Вхождение «a» от 0 и больше | id\d* | id, id12 , iduser, id12user |
| a+ | Вхождение «a» от 1 и больше | id\d+ | id, id12, iduser, id12user |
| a{3} | Вхождение «a» ровно 3 раза | \d{3} | 1, 12, 123, 1234, 12345 |
| a{2,} | Вхождение «a» от 2 и более раз | \d{3,} | 1, 12, 123, 1234, 12345 |
| a{2,4} | Вхождение «a» от 2 до 4 включительно | \d{3,4} | 1, 12, 123, 1234, 12345 |
| *?<br>+?<br>??<br>{m,n}?<br>{,n}?<br>{m,}? | По умолчанию кванторы жадные — захватывают максимально возможное число символов. Добавление ? делает их ленивыми, они захватывают минимально возможное число символов | \(.*\)<br>\(.*?\) | (a + b) * (c + d) * (e + f)<br>(a + b) * (c + d) * (e + f) |

# Классы символов

| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| [abc] | Единичный символ: a, b или c | [лт]ень | лень, пень, тень |
| [^abc] | Любой единичный символ, кроме: a, b или c | [^лт]ень | лень, пень, тень |
| [a-z] | Единичный символ из диапазона a-z | [0-9]+ | Агент 007, дом15дом |
| [^a-z] | Единичный символ не входящий в диапазон a-z | [^0-9]+ | Агент 007, дом15дом |
| [a-zA-Z] | Единичный символ из диапазона a-z или A-Z | [а-яА-Я]+ | Агент 007, агЕНт 007 |
| [abc-]<br>[-1] | Если нужен минус, то его необходимо указать первым или последним | [-\d]+ | Агент-007, a-b |
| [*[(+\\\]\t] | Внутри скобок нужно экранировать только ] и \ | | |

# Скобочные группы

| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| (?:...) | Группировка без обратной связи. Используется если группа используется только для группировки и её результат в дальнейшем не потребуется. Под результат такой группировки не выделяется отдельная область памяти и, соответственно, ей не назначается номер. Это положительно влияет на скорость выполнения выражения, но понижает удобочитаемость | match (?:this) | match this |
| (...) | Изолирует часть совпадения и присваивает ID, по которому можно обратиться позже. ID начинаются с 1. | match (this )+ | match this this |

# Именованные группы и условия

| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| (?P<name>Sally) | На такую группу можно ссылаться по имени вместо ID | (?P<name>Sally) | Call me Sally. |
| (?(?=если)то\|иначе) | Если операция просмотра успешна, то далее выполняется часть то, иначе выполняется часть иначе. В выражении может использоваться любая из четырёх операций просмотра. Следует учитывать, что операция просмотра нулевой ширины, поэтому части то в случае позитивного или иначе в случае негативного просмотра должны включать в себя описание шаблона из операции просмотра | (?(?<=а)м\|п) | мам,пап |
| (?(n)то\|иначе) | Если n-я группа вернула значение, то поиск по условию выполняется по шаблону то, иначе по шаблону иначе. | (а)?(?(1)м\|п) | мам, пап |

# Просмотр вперед и назад

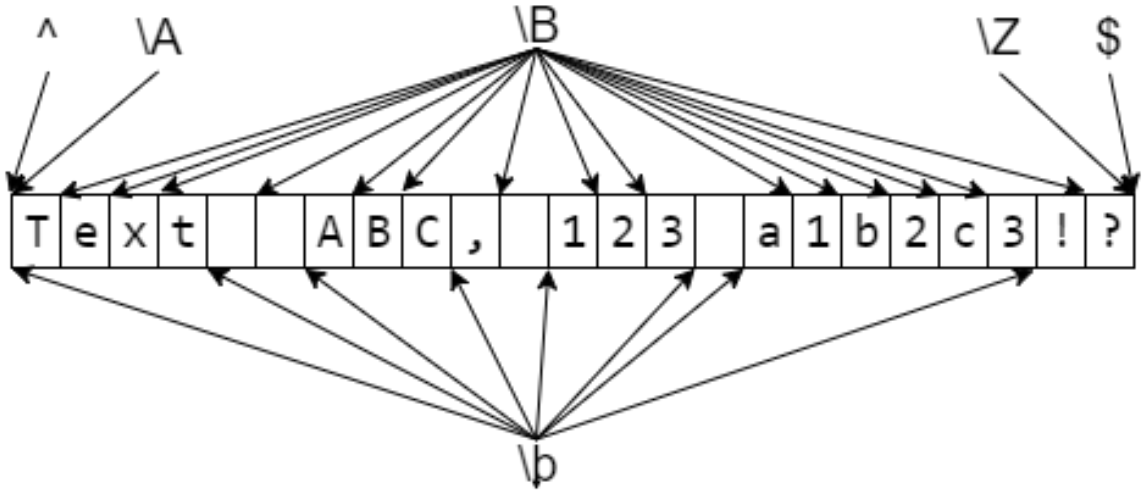
| Шаблон | Описание | Пример | Применение к тексту |
|---|---|---|---|
| (?=...) | Позитивный просмотр вперед. Группа не включается | foo(?=bar) | foobar foobaz |
| (?!...) | Негативный просмотр вперед. Группа не включается | foo(?!bar) | foobar foobaz |
| (?<=foo)bar | Позитивный просмотр назад. Группа не включается | (?<=foo)bar | foobar fuubar |
| (?<!...) | Негативный просмотр назад. Группа не включается | (?<!not )foo | not foo but foo |

# Дополнительные флаги в Python

| Шаблон | Описание |
|---|---|
| re.ASCII | По умолчанию \w, \W, \b, \B, \d, \D, \s, \S соответствуют все unicode символы с соответствующим качеством. Например, \d соответствуют не только арабские цифры, но и вот такие: ٠١٢٣٤٥٦٧٨٩. re.ASCII ускоряет работу, если все соответствия лежат внутри ASCII. |
| re.IGNORECASE | Не различать заглавные и маленькие буквы. Работает медленнее, но иногда удобно |
| re.LOCALE | Делает \w, \W, \b, \B зависимыми от текущей локализации |
| re.MULTILINE | Специальные символы ^ и $ соответствуют началу и концу каждой строки |
| re.DOTALL | По умолчанию символ \n конца строки не подходит под точку. С этим флагом точка — вообще любой символ |
| re.VERBOSE | Игнорирование пробельных символов и комментариев для улучшения вида регулярного выражения |
| re.UNICODE | Для совместимости. Игнорируется в строковых шаблонах (т.к. является значением по умолчанию) и запрещен для байтовых шаблонов |

# Якоря

| Шаблон | Описание |
|---|---|
| ^ | Начало всего текста или начало строки текста, если flag=re.MULTILINE |
| $ | Конец всего текста или конец строчки текста, если flag=re.MULTILINE |
| \A | Строго начало всего текста |
| \Z | Строго конец всего текста |
| \b | Начало или конец слова (слева пусто или не-буква, справа буква и наоборот) |
| \B | Не граница слова: либо и слева, и справа буквы, либо и слева, и справа НЕ буквы |



# Методы модуля re

| Метод | Описание |
|---|---|
| re.compile(pattern, flag=0) | Компилирует шаблон регулярного выражения в regular expression object, который может использоваться для поиска |
| re.search(pattern, string, flag=0) | Ищет первое совпадение по шаблону. Возвращает Match Object или None |
| re.match(pattern, string, flag=0) | Если начало строки совпадает с шаблоном, то возвращает Match Object, в противном случае None |
| re.fullmatch(pattern, string, flag=0) | Если вся строка совпадает с шаблоном, то возвращает Match Object, в противном случае None |
| re.split(pattern, string, maxsplit=0, flags=0) | Разбивает строку по шаблону |
| re.findall(pattern, string, flags=0) | Возвращает все непересекающиеся совпадения в строке |
| re.finditer(pattern, string, flags=0) | Возвращает итератор со всеми непересекающимися совпадениями в строке |
| re.sub(pattern, repl, string, count=0, flags=0) | Ищет шаблон в строке и заменяет его на указанную подстроку |
| re.subn(pattern, repl, string, count=0, flags=0) | Тоже, что и sub, только возвращает не строку, а кортеж (new_string, number_of_subs_made) |
| re.escape(pattern) | Экранирует спецсимволы в шаблоне |

# Объект скомпилированного паттерна

| Метод | Описание |
|---|---|
| pattern.search ,pattern.match, pattern.fullmatch, pattern.split, pattern.findall, pattern.finditer, pattern.sub, pattern.subn | Работают также, как и функции модуля |
| Pattern.flags | Комбинация всех флагов из шаблона |
| Pattern.groups | Количество групп в шаблоне |
| Pattern.groupindex | Словарь имя группы – id, содержащий (?P<id>) именованные группы |
| Pattern.pattern | Строка шаблон, которая была скомпилирована |

# Объект совпадения (Match)

| Метод | Описание |
|---|---|
| Match.expand(template) | Возвращает строку, полученную путем подстановки с помощью косой черты найденными значениями |
| Match.group([group1, ...]) | Возвращает tuple со значениями групп, номера (имена) которых мы передали |
| Match.__getitem__(g) | Обеспечивает удобный доступ к группам по индексу |
| Match.groups(default=None) | Возвращает tuple с результатами совпадений всех групп |
| Match.groupdict(default=None) | Возвращает словарь ключ-значение именованных групп |
| Match.start([group])<br>Match.end([group]) | Возвращают индексы начала вхождения шаблона и конца |
| Match.span([group]) | Возвращает начало и конец совпадения некоторой группы |

# Пример использования

1. Импортируем модуль re
2. Выполняем функции через модуль re (первый параметром передаем регулярное выражение)
3. Либо компилируем выражение с помощью re.compile и используем объект скомпилированного выражения

```
>>> import re
>>> some_text = "123 how are321 yo231u"
>>> template = re.compile(r'\d+')
>>> template.search(some_text)
<re.Match object; span=(0, 3), match='123'>
>>> template.match(some_text)
<re.Match object; span=(0, 3), match='123'>
>>> template.fullmatch(some_text)
>>> re.split(' ', some_text)
['123', 'how', 'are321', 'yo231u']
>>> template.findall(some_text)
['123', '321', '231']
>>> template.finditer(some_text)
<callable_iterator object at 0x000002054FAFF490>
>>> template.sub('NEW', some_text)
'NEW how areNEW yoNEWu'
```

# Ваши вопросы

что необходимо прояснить в рамках данного раздела

# Встроенные модули

полезные модули, которые встроены в Python и которые можно использовать в своей работе

# Модуль copy

Модуль copy используется для создания копий некоторого объекта:

- copy.copy(obj) – поверхностная копия объекта. Старается скопировать объект, но сохранить все вложенные ссылки
- copy.deepcopy(obj) – глубокая копия объекта. Копирует объект и все вложенные объекты. Ссылки не сохраняются

Документация:
https://docs.python.org/3/library/copy.html

```python
import copy


class SomeClass:
    val: dict

    def __init__(self):
        self.val = {}


some_obj = SomeClass()
shallow_copy = copy.copy(some_obj)
deep_copy = copy.deepcopy(some_obj)

assert shallow_copy is some_obj
assert shallow_copy.val is some_obj.val

assert deep_copy is some_obj
assert deep_copy.val is some_obj.val
```

# Модуль abc

Модуль abc позволяет работать с абстрактными классами (также метаклассами) и абстрактными методами:

- abc.ABC – наследники данного класса **могут** быть абстрактными классами
- abc.ABCMeta – классы, созданные из данного метакласса или его наследника, **могут** быть абстрактными классами
- abc.abstractmethod – **декоратор**, который метод делает абстрактным

Документация: https://docs.python.org/3/library/abc.html

```python
class AbstractClass1(metaclass=MetaAbstract):
    @abstractmethod
    def some_func1(self):
        pass


class AbstractClass2(ABC):
    @abstractmethod
    def some_func2(self):
        pass


class SomeClass(AbstractClass1, AbstractClass2):
    def some_func1(self):
        return 1

    def some_func2(self):
        return 2
```

# Модуль typing

Данный модуль в основном используется для аннотаций типов.

Обратите внимание, что это **не Runtime** проверка типа!

Документация: https://docs.python.org/3/library/typing.html

```python
from typing import Union, Optional, Any, List, Dict


# до Python 3.9
def some(val1: Optional[str], val2: Any) -> Union[int, float]:
    pass


def some2(val1: List[str], val2: Dict[str, int]) -> Any:
    pass


# с Python 3.9
def some3(val1: list[str], val2: dict[str, int]) -> Any:
    pass
```

# Модуль collections

Данный модуль содержит некоторые специальные типы данных, которые являются альтернативой встроенным типам-контейнерам, таким как dict, list, set, tuple. Документация:

| | |
|---|---|
| namedtuple() | Функция-фабрика для создания классов-кортежей с именованными полями |
| deque | Контейнер, который похож на list, но с возможность добавления (удаления) и в начало и в конец |
| ChainMap | Контейнер, который похож на dict, предоставляет единый интерфейс для mapping-объектов |
| Counter | Подкласс dict для подсчета hashable объектов |
| OrderedDict | Подкласс dict, который помнит порядок добавления элементов |
| defaultdict | Подкласс dict, который вызывает функцию-фабрику для поддержки отсутствующих значений |
| UserDict | Обертка над dict для упрощенного создания пользовательских словарей |
| UserList | Обертка над list для упрощенного создания пользовательских списков |
| UserS | |

# Модуль collections.abc

Данный модуль может быть использован для того, чтобы проверить, реализованы ли в классе некоторые магические методы.

Документация: https://docs.python.org/3/library/collections.abc.html

```
1  from collections import abc
2
3
4  class Some:
5      def __iter__(self):
6          return self
7
8      def __next__(self):
9          return 1
10
11
12 print(issubclass(Some, abc.Iterable))
```

# Модуль itertools – бесконечная итерация

С помощью этих методов можно генерировать объекты или совершать определенные действия неограниченное количество раз. Это значит, что программисту потребуется **самостоятельно прервать** созданный цикл.

Документация: https://docs.python.org/3/library/itertools.html

| | |
|---|---|
| count | Итерация с заданным шагом без ограничений |
| cycle | Итерация с повторением без ограничений |
| repeat | Итерация с повторением заданное количество раз |

```python
import itertools


# 0 2 4 6 8
for i in itertools.count(0, 2):
    if i >= 10:
        break
    print(i)

# D O G D O
count = 1
for i in itertools.cycle('DOG'):
    if count > 5:
        break
    print(i)
    count += 1

# ['DOG', 'DOG', 'DOG']
data = [i for i in itertools.repeat('DOG', 3)]
print(data)
```

# Модуль itertools – комбинация значений

Данные функции позволяют комбинировать различные значения, меняя местами их составляющие.

| | |
|---|---|
| combinations | Комбинация всех возможных значений без повторяющихся элементов |
| combinations_with_replacement | Комбинация всех возможных значений с повторяющимися элементами |
| permutations | Комбинация с перестановкой всех возможных значений |
| product | Комбинация, полученная из всех возможных значений вложенных списков |

```python
import itertools


# [('D', 'O'), ('D', 'G'), ('O', 'G')]
data = list(itertools.combinations('DOG', 2))
print(data)


# DD DO DG OO OG GG
for i in itertools.combinations_with_replacement('DOG', 2):
    print(''.join(i))


# DO DG OD OG GD GO
for i in itertools.permutations('DOG', 2):
    print(''.join(i))


# [(0, 2), (0, 3), (1, 2), (1, 3)]
data = list(itertools.product((0, 1), (2, 3)))
print(data)
```

# Модуль itertools – фильтрация

Для управления данными в последовательности значений используются инструменты фильтрации. Некоторые функции itertools умеют автоматически удалять отдельные элементы, не удовлетворяющие заданных программистом условий

| | |
|---|---|
| filterfalse | Все элементы, для которых функция возвращает ложь |
| dropwhile | Все элементы, начиная с того, для которого функция вернет ложь |
| takewhile | Все элементы, до тех пор, пока функция не вернет истину |
| compress | Удаление элементов, для которых было передано значение ложь |

```python
import itertools


# [1, 3, 5, 1]
data = list(itertools.filterfalse(
    lambda i: i % 2 == 0, [1, 2, 3, 0, 4, 5, 1])
)
print(data)


# [0, 4, 5, 1]
data = list(itertools.dropwhile(
    lambda i: i != 0, [1, 2, 3, 0, 4, 5, 1])
)
print(data)


# [1, 2, 3]
data = list(itertools.takewhile(
    lambda i: i != 0, [1, 2, 3, 0, 4, 5, 1])
)
print(data)


# ['D', 'G']
data = list(itertools.compress('DOG', [True, False, True]))
print(data)
```

# Модуль itertools – прочие итераторы

| | |
|---|---|
| chain | Поочередное объединение списков при помощи итераторов |
| chain.from_terable | Аналогично chain, но аргумент – список, в который вложены объединяемые списки. |
| islice | Получение среза, благодаря указанному количеству элементов |
| zip_longest | Объединение нескольких итераций с повышением размера до максимального |
| tee | Создание кортежа из нескольких готовых итераторов |
| groupby | Группировка элементов последовательности по некоторым ключевым значениям |
| accumulate | Каждый элемент результирующей последовательности равен сумме текущего и всех предыдущих исходной последовательности |
| starmap | В заданную функцию передает список подставляемых аргументов |

# Модуль math

Встроенный модуль **math** в Python предоставляет набор функций для выполнения математических, тригонометрических и логарифмических операций.

Документация: https://docs.python.org/3/library/math.html

```python
import math


# возведение числа 2 в степень 3
n1 = math.pow(2, 3)
print(n1)  # 8
# ту же самую операцию можно выполнить так
n2 = 2 ** 3
print(n2)

# возведение в квадрат
print(math.sqrt(9))  # 3
# ближайшее наибольшее целое число
print(math.ceil(4.56))  # 5
# ближайшее наименьшее целое число
print(math.floor(4.56))  # 4
# перевод из радиан в градусы
print(math.degrees(3.14159))  # 180
# перевод из градусов в радианы
print(math.radians(180))  # 3.1415.....
# косинус
print(math.cos(math.radians(60)))  # 0.5
# синус
print(math.sin(math.radians(90)))  # 1.0
# тангенс
print(math.tan(math.radians(0)))  # 0.0
print(math.log(8, 2))  # 3.0
print(math.log10(100))  # 2.0

radius = 30
# площадь круга с радиусом 30
area = math.pi * math.pow(radius, 2)
print(area)

# натуральный логарифм числа 10
number = math.log(10, math.e)
print(number)
```

# Модуль decimal

Тип **Decimal** создан, чтобы операции над рациональными числами в компьютере выполнялись также, как они выполняются людьми, как их преподают в школе. Иными словами, чтобы все-таки 0.1 + 0.1 + 0.1 == 0.3

Документация: https://docs.python.org/3/library/decimal.html

```python
from decimal import Decimal


number = Decimal("0.1")
number = number + number + number
print(number)       # 0.3

number = Decimal("0.1")
number = number + 2
print(number)

number = Decimal("0.444")
number = number + 0.1   # здесь возникнет ошибка

number = number.quantize(Decimal("1.00"))
print(number)  # 0.44

number = Decimal("0.555678")
print(number.quantize(Decimal("1.00")))  # 0.56

number = Decimal("0.999")
print(number.quantize(Decimal("1.00")))  # 1.00
```

# Сравнение Decimal и float

| Характеристика/тип | float | Decimal |
|---|---|---|
| Реализация | Аппаратная | Программная |
| Размер | 64 бита | Почти любой, пока есть память |
| Основание | 2 | 10 |
| Скорость | ✓ | ✗ |
| Настраиваемость | ✗ | ✓ |
| Для финансов и бизнеса | ✗ | ✓ |
| Для симуляций, визуализаций и игр | ✓ | ✗ |
| Для высокоточных вычислений | ✗ | ✓ |

# Модуль fractions

Модуль **fractions** пригодится в тех случаях, когда вам необходимо выполнить вычисления с дробями, или когда результат должен быть выражен в формате дроби.

Документация: https://docs.python.org/3/library/fractions.html

```python
from fractions import Fraction
from decimal import Decimal


Fraction(16, -10)  # Fraction(-8, 5)
Fraction(123)  # Fraction(123, 1)
Fraction()  # Fraction(0, 1)
Fraction('3/7')  # Fraction(3, 7)
Fraction(' -3/7 ')  # Fraction(-3, 7)
Fraction('1.414213 \t\n')  # Fraction(1414213, 1000000)
Fraction('-.125')  # Fraction(-1, 8)
Fraction('7e-6')  # Fraction(7, 1000000)
Fraction(2.25)  # Fraction(9, 4)
Fraction(1.1)  # Fraction(2476979795053773, 2251799813685248)
Fraction(Decimal('1.1'))  # Fraction(11, 10)
Fraction(1, 2) + Fraction(3, 4)  # Fraction(5, 4)
Fraction(1, 8) ** Fraction(1, 2)  # 0.3535533905932738
```

# Модуль random

Среди инструментов, которые предназначены для работы с псевдослучайными числами, находится довольно обширная библиотека **random**.

Данный модуль использует в своих целях текущее системное время, которое установлено на компьютере. Это гарантирует получение разных последовательностей значений при каждом новом обращении к генератору.

Документация: https://docs.python.org/3/library/random.html

```
1  import random
2
3
4  random.random()
5  random.randint(1, 5)
6  random.randrange(1, 5, 2)
```

# Модуль random – основные методы

| | |
|---|---|
| random() | возвращает число в диапазоне от 0 до 1 |
| seed(a) | настаивает генератор на новую последовательность a |
| randint(a, b) | возвращает целое число в диапазоне от a и b |
| randrange(a, b, c) | возвращает целое число в диапазоне от a до b с шагом c |
| uniform(a, b) | возвращает вещественное число в диапазоне от a и b |
| shuffle(a) | перемешивает значения в списке a |
| choice(a) | возвращает случайный элемент из списка a |
| sample(a, b) | возвращает последовательность длиной b из набора a |
| getstate() | возвращает внутреннее состояние генератора |
| setstate(a) | восстанавливает внутреннее состояние генератора a |
| getrandbits(a) | возвращает a случайно сгенерированных бит |
| triangular(a, b, c) | возвращает вещественное число от a до b с распределением c |

# Модуль statistics

В Python есть встроенный модуль для работы со статистикой (он конечно не идет в конкуренцию с NumPy и SciPy). Но для простых задач сгодится.

Модуль может работать с int, float, Decimal и Fractions.

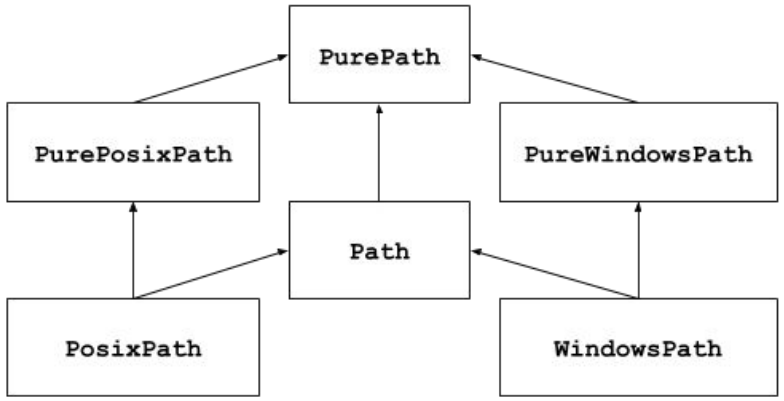
Документация: https://docs.python.org/3/library/statistics.html

```python
import statistics
import random
from fractions import Fraction as F
from decimal import Decimal as D


statistics.mean([11, 2, 13, 14, 44])
# returns 16.8

statistics.mean([F(8, 10), F(11, 20), F(2, 5), F(28, 5)])
# returns Fraction(147, 80)

statistics.mean([D("1.5"), D("5.75"), D("10.625"), D("2.375")])
# returns Decimal('5.0625')

data_points = [random.randint(1, 100) for x in range(1, 1001)]
statistics.mode(data_points)
# returns 94

data_points = [random.randint(1, 100) for x in range(1, 1001)]
statistics.mode(data_points)
# returns 49

data_points = [random.randint(1, 100) for x in range(1, 1001)]
statistics.mode(data_points)
# returns 32

data_points = [random.randint(1, 100) for x in range(1, 50)]
statistics.median(data_points)
# returns 53

data_points = [random.randint(1, 100) for x in range(1, 51)]
statistics.median(data_points)
# returns 51.0

data_points = [random.randint(1, 100) for x in range(1, 51)]
statistics.median(data_points)
# returns 49.0

data_points = [random.randint(1, 100) for x in range(1, 51)]
statistics.median_low(data_points)
# returns 50

statistics.median_high(data_points)
# returns 52

statistics.median(data_points)
# returns 51.0
```

# Модуль pathlib

Данный модуль используется для работы с путями операционной системы.

Документация: https://docs.python.org/3/library/pathlib.html




```python
from pathlib import Path


wave = Path("ocean", "wave.txt")
print(wave)

home = Path.home()
wave_absolute = Path(home, "ocean", "wave.txt")
print(home)
print(wave_absolute)


wave = Path("ocean", "wave.txt")
print(wave)
print(wave.name)
print(wave.suffix)

shark = Path("ocean", "animals", "fish", "shark.txt")
print(shark)
print(shark.parent)

for txt_path in Path().glob("*.py"):
    print(txt_path)
```

# Модуль os

Модуль os предназначен для широкого круга задач (это такой ящик со всем, связанным с операционной системой).

Документация: https://docs.python.org/3/library/os.html

```python
import os


# Переменные среды
print(os.environ)

# Текущая рабочая директория
print(os.getcwd())

# Смена рабочей директории
os.chdir(r"D:\folder")
os.chdir(r"..")

# распечатать все файлы и каталоги в текущем каталоге
print(os.listdir(r"D:\folder"))

# Проверка существования пути
print(os.path.exists(r"D:/test.txt"))

# Проверка: это файл?
print(os.path.isfile(r"D:/test.txt"))

# Проверка: это директория?
print(os.path.isdir(r"D:/test.txt"))

# объединение и разделение путей
print(os.path.split(r"D:\folder\test.txt"))
print(os.path.join(r"D:\folder", r"test.txt"))
```

```python
import os


# Создать директорию
os.mkdir(r"D:\folder")

# Удалить директорию (если в ней нет файлов)
os.rmdir(r"D:\folder")

# Создать файл
file = open(r'D:\test.txt', 'w')
file.close()

# Удалить файл
os.remove(r"D:\test.txt")

# Удалит folder, first, second, third
os.removedirs(r"D:\folder\first\second\third")

# запуск отдельных файлы и папки прямиком из программы
os.startfile(r"D:\test.txt")

# Получение информации о файле
print(os.stat(r"D:\test.txt"))

# размер файла
print(os.path.getsize("D:\\test.txt"))

# test.txt
print(os.path.basename("D:/test.txt"))

# D:/folder
print(os.path.dirname("D:/folder/test.txt"))

# Переименовать файл/директорию
os.rename(r"D:\folder", r"D:\catalog")

# переименовать folder\first\second в catalog\one\two
os.renames(r"D:\folder\first\second", r"D:\catalog\one\two")

# Перемещение файлов
os.replace("renamed-text.txt", "folder/renamed-text.txt")

for root, directories, files in os.walk(r"D:\projects"):
    print(root)
    for directory in directories:
        print(directory)
    for file in files:
        print(file)
```

# pathlib vs os.path

Следующие методы будут аналогичны в обоих модулях.

Хорошая статья про сравнение pathlib и os.path: https://habr.com/ru/post/453862/

| os and os.path | pathlib |
|---|---|
| os.path.abspath() | Path.resolve() |
| os.chmod() | Path.chmod() |
| os.mkdir() | Path.mkdir() |
| os.makedirs() | Path.mkdir() |
| os.rename() | Path.rename() |
| os.replace() | Path.replace() |
| os.rmdir() | Path.rmdir() |
| os.remove(), os.unlink() | Path.unlink() |
| os.getcwd() | Path.cwd() |
| os.path.exists() | Path.exists() |
| os.path.expanduser() | Path.expanduser() and Path.home() |
| os.listdir() | Path.iterdir() |
| os.path.isdir() | Path.is_dir() |
| os.path.isfile() | Path.is_file() |
| os.path.islink() | Path.is_symlink() |
| os.link() | Path.link_to() |
| os.symlink() | Path.symlink_to() |
| os.readlink() | Path.readlink() |
| os.stat() | Path.stat(), Path.owner(), Path.group() |
| os.path.isabs() | PurePath.is_absolute() |
| os.path.join() | PurePath.joinpath() |
| os.path.basename() | PurePath.name |
| os.path.dirname() | PurePath.parent |
| os.path.samefile() | Path.samefile() |
| os.path.splitext() | PurePath.suffix |

# Модуль sys

Модуль **sys** предоставляет набор функций, которые дают информацию о том, как интерпретатор Python взаимодействует с операционной системой:

- Какая версия Python запущена
- Путь к интерпретатору Python, исполняющему текущий скрипт
- Параметры командной строки, используемые при запуске на выполнение скрипта
- Флаги, установленные интерпретатором
- Представление значений с плавающей точкой
- И т.д.

Документация:
https://docs.python.org/3/library/sys.html

```python
import sys


# текущий Python или виртуальное окружение
print(sys.exec_prefix)
# путь к исполняемому файлу Python
print(sys.executable)
# глобальный Python
print(sys.base_exec_prefix)
# Кодировка ОС (как правило UTF-8)
print(sys.getfilesystemencoding())
# имя кодировки, используемой по умолчанию
print(sys.getdefaultencoding())
# Версия Windows
print(sys.getwindowsversion())
# Версия ОС
print(sys.platform)
# Список аргументов командной строки
print(sys.argv)
# Возвращает максимально возможное значение рекурсии
# и максимальную глубину стека интерпретатора
print(sys.getrecursionlimit())
# изменить значение можно с помощью sys.setrecursionlimit()
# print(sys.setrecursionlimit(ПРЕДЕЛ))
# Возвращает число, определяющее, частоту переключения потоков.
# Это вещественное число. Значение в секундах
print(sys.getswitchinterval())
# sys.setswitchinterval(интервал)
# Кортеж, содержащий информацию о параметрах хеша
print(sys.hash_info)
# информация о запущенном в данный момент интерпретаторе Python
print(sys.implementation)
print(sys.version_info)
# загруженные модули и пути к ним
print(sys.modules)
# Список строк, который показывает, в каких директориях ищутся модули
# Инициализируется из переменной среды PYTHONPATH и установок по умолчанию
print(sys.path)
# количество ссылок на переданный в качестве аргумента объект
# sys.getrefcount(ОБЪЕКТ)
# Показывает, сколько байтов выделяется для хранения объекта любого типа
print(sys.getsizeof(123))
# Выход из Python. Вызывает исключение SystemExit, которое можно перехватить
# sys.exit([arg])
# Функция, возвращающая кортеж, состоящий из трёх элементов,
# которые показывают информацию
# об обрабатываемом в настоящее время исключении.
print(sys.exc_info())

```

# Добавляем путь в sys.path

Чтобы Python знал, где нужно искать модули проект – нужно добавить путь к проекту в sys.path.

```
1  import pathlib
2  import sys
3
4  sys.path.extend([
5      str(pathlib.Path(__file__).parent.resolve())
6  ])
```

# Модуль datetime

С помощью данного модуля мы работаем с датой и временем.

Документация: https://docs.python.org/3/library/datetime.html.

У него есть следующие классы:

| Класс | Описание |
|---|---|
| date | представляет собой дату, полностью исключая данные о времени, на основе Григорианского календаря |
| time | включает данные о времени, полностью игнорируя сведения о дате |
| datetime | содержит информацию о времени и дате, основываясь на данных из Григорианского календаря |
| timedelta | описывает определенный период во времени, который находится между двумя различными моментами |
| tzinfo | представляет различные сведения о часовом поясе |
| timezone | описывает время, руководствуясь стандартом UTC |

# Модуль datetime – date

Класс **date** используется для представления данных о дате, которые включают **год, месяц и число**. Необходимо предварительно импортировать модуль datetime. Для создания объектов типа date следует произвести вызов одноименного конструктора, указав ему в качестве параметров сведения о дате. При этом нельзя забывать о порядке, в котором находятся аргументы: год, затем месяц и число.

```python
import datetime


a = datetime.date(2001, 10, 28)
# 2001-10-28
print(a)
# <class 'datetime.date'>
print(type(a))

# 2021-02-11
print(datetime.date.today())

a = datetime.date(2012, 7, 21)
# 2012
print(a.year)
# 7
print(a.month)
# 21
print(a.day)
```

# Модуль datetime – time

Класс **time** служит для демонстрации данных о времени, полностью игнорируя дату. Следует импортировать модуль datetime. Создать объект, принадлежащий к типу time можно с помощью конструктора, который принимает такие аргументы как количество часов, минут, секунд и микросекунд. Указывая данные параметры, не стоит забывать об их необходимом порядке, расставляя числа на правильных позициях.

```python
import datetime


a = datetime.time(12, 18, 35, 5867)
# 12:18:35.005867
print(a)
# <class 'datetime.time'>
print(type(a))

a = datetime.time(23, 5, 30)
b = datetime.time(7, 26)
c = datetime.time(21)
# 23:05:30
print(a)
# 07:26:00
print(b)
# 21:00:00
print(c)

a = datetime.time(16, 3, 49, 23578)
# 16
print(a.hour)
# 3
print(a.minute)
# 49
print(a.second)
# 23578
print(a.microsecond)
```

# Модуль datetime – datetime

Класс **datetime** объединят дату и время. Используется конструктор с аргументами под каждое значение

```python
import datetime


a = datetime.datetime(2020, 2, 10, 8, 12, 24, 34574)
print(a.year)  # 2020
print(a.month)  # 2
print(a.day)  # 10
print(a.hour)  # 8
print(a.minute)  # 12
print(a.second)  # 24
print(a.microsecond)  # 345740
```

```python
import datetime


c = datetime.datetime(2017, 7, 18, 4, 52, 33, 51204)
# 2017-07-18 04:52:33.051204
print(c)
# <class 'datetime.datetime'>
print(type(c))

a = datetime.datetime(2007, 2, 13)
b = datetime.datetime(2013, 10, 25, 12, 8, 47)
# 2007-02-13 00:00:00
print(a)
# 2013-10-25 12:08:47
print(b)

a = datetime.datetime.today()
b = datetime.datetime.now()
# 2018-11-08 11:12:53.256261
print(a)
# 2018-11-08 11:12:53.256270
print(b)

a = datetime.datetime.today().strftime("%d.%m.%Y")
b = datetime.datetime.today().strftime("%H:%M:%S")
# 08.11.2018
print(a)
# 11:41:04
print(b)
```

# Модуль datetime – timedelta

Класс **timedelta** предназначен для удобного выполнения различных манипуляций над датами и временем. Можно создать объект данного класса, воспользовавшись конструктором.

Также можно прибавлять разные даты, формируя тем самым новый объект.

```python
from datetime import timedelta, datetime


a = timedelta(days=5, hours=21, minutes=2, seconds=37)
# 5 days, 21:02:37
print(a)
# <class 'datetime.timedelta'>
print(type(a))

b = datetime(2006, 12, 5)
c = a + b
# 2006-12-10 21:02:37
print(c)
```

# Модуль datetime – tzinfo и timezone

Классы **tzinfo** и **timezone** применяются для работы с информацией, которая содержит сведения о часовых поясах. Создать объект, принадлежащий типу tzinfo невозможно, поскольку этот класс является абстрактным. Однако можно воспользоваться наследованием, создав собственный класс на основе tzinfo. При этом следует помнить, что для работы с такими объектами придется реализовать несколько абстрактных методов, к числу которых относятся utcoffset (смещение по местному времени с UTC), dst (настройка перехода на летнее время), а также функция tzname (имя часового пояса в виде строки).

```python
from datetime import tzinfo, timedelta, datetime, timezone


class UTC0530(tzinfo):
    def __init__(self, offset=19800, name=None):
        self.offset = timedelta(seconds=offset)
        self.name = name or self.__class__.__name__

    def utcoffset(self, dt):
        return self.offset

    def tzname(self, dt):
        return self.name

    def dst(self, dt):
        return timedelta(0)


a = datetime.now(timezone.utc)
# 2021-02-11 10:09:09.284306+00:00
print(a)
b = datetime.now(UTC0530())
# 2021-02-11 15:39:09.284306+05:30
print(b)
# 5:30:00
print(b.utcoffset())
# UTC0530
print(b.tzname())
# 0:00:00
print(b.dst())
```

# Спецификация даты

| | | | |
|---|---|---|---|
| %a | название дня недели в сокращенном виде | %M | количество минут в виде целого числа |
| %A | название дня недели в полном виде | %S | количество секунд в виде целого числа |
| %w | номер дня недели в виде целого числа | %f | количество микросекунд в виде целого числа |
| %d | номер дня месяца в виде целого числа | %z | часовой пояс в формате UTC |
| %b | название месяца в сокращенном виде | %Z | название часового пояса |
| %B | название месяца в полном виде | %j | номер дня в году |
| %m | номер месяца в числовом представлении | %U | номер недели в году, если считать с воскресенья |
| %y | номер года без столетия | %W | номер недели в году, если считать с понедельника |
| %Y | номер года в полном представлении | %c | местное представление даты и времени |
| %H | количество часов в 24-часовом формате | %x | местное представление даты |
| %I | количество часов в 12-часовом формате | %X | местное представление времени |
| %p | часовом формате | | |

# Операции с датами

| | |
|---|---|
| a + b | суммирует значения дат a и b |
| a – b | суммирует значения дат a и b |
| a * i | умножает численное представление свойств даты a на некую константу i |
| a // i | делит численное представление свойств даты a на некую константу i, остаток отбрасывается |
| +a | возвращает объект timedelta с полностью идентичным значением a |
| –a | возвращает объект timedelta с полностью противоположным значением a |
| a > b | возвращает true, если a больше b |
| a < b | возвращает true, если a меньше b |
| abs(a) | возвращает объект timedelta с положительным значением всех свойств a |
| str(a) | возвращает строковое представление объекта a в формате, заданном по умолчанию |
| repr(a) | возвращает строковое представление объекта a в формате с отрицательными значениями |

# Модуль dataclasses

В Python 3.7 появился модуль, с помощью которого можно создавать свои классы данных

Документация:

https://docs.python.org/3/library/dataclasses.html

```python
from dataclasses import dataclass


class SimplePerson:
    name: str
    surname: str
    age: int

    def __init__(self, name, surname, age):
        self.name = name
        self.surname = surname
        self.age = age


@dataclass
class Person:
    name: str
    surname: str
    age: int
```

# Модуль enum

Представьте, что вам нужно описать набор всех возможных состояний чего-либо (набор констант).

Перечисление является неизменяемым как для набора элементов — нельзя определить новый член перечисления во время выполнения и нельзя удалить уже определенный член, так и для тех значений элементов, которые он хранит — нельзя [пере]назначать любые значения атрибута или удалять атрибут.

Документация:

https://docs.python.org/3/library/enum.html

```python
from enum import Enum


class Colors(Enum):
    RED = "#FF0000"
    GREEN = "#008000"
    BLUE = "#0000FF"


print(Colors.RED)
print(Colors.RED.value)


class ColorsWithNames(Enum):
    RED = ("#FF0000", "Red")
    GREEN = ("#008000", "Green")
    BLUE = ("#0000FF", "Blue")

    def __init__(self, val, fullname):
        self.val = val
        self.fullname = fullname


print(ColorsWithNames.RED)
print(ColorsWithNames.RED.val)
print(ColorsWithNames.RED.fullname)
```

# Модуль argparse

Модуль argparse позволяет разбирать аргументы, передаваемые скрипту при его запуске из командной строки, и даёт возможность пользоваться этими аргументами в скрипте.

Модуль argparse — это средство, с помощью которого можно наладить общение между автором программы и тем, кто ей пользуется, например — между вами, когда вы сегодня пишете скрипт, и вами же, когда вы завтра его запускаете, что-то ему передавая

```python
import argparse


def parse_cl_args():
    """
    Парсинг аргументов командной строки
    :return: Namespace(port=8000, debug=False, skip=False)
    """
    parser = argparse.ArgumentParser(
        description='Run backend',
        epilog='Author: https://github.com/EdiBoba'
    )
    parser.add_argument(
        "-H",
        "--host",
        help="TCP/IP hostname to serve on (default: %(default)r)",
        default="localhost",
    )
    parser.add_argument(
        '-p',
        '--port',
        type=int,
        help='TCP/IP port to serve on (default: %(default)r)',
        default=8000
    )
    parser.add_argument(
        '-d',
        '--debug',
        help='activate debug mode (default: %(default)r)',
        action='store_true',
        default=False
    )
    parser.add_argument(
        '-s',
        '--skip',
        help='skip old messages (default: %(default)r)',
        action='store_true',
        default=False
    )
    parser.add_argument(
        '-v',
        '--version',
        help='show app version and exit',
        action='version',
        version='v1'
    )
    arguments = parser.parse_args()
    return arguments
```

# Модуль logging

Модуль **logging**, входящий в состав стандартной библиотеки Python, предоставляет большую часть необходимых для журналирования функций. Если его правильно настроить, записи лога могут предоставить большое количество полезной информации о работе приложения: в каком месте кода была сформирована запись, какие потоки и процессы были запущены, каково состояние памяти.

Документация: https://docs.python.org/3/library/logging.html

```python
import logging


logging.basicConfig(
    level=logging.DEBUG, filename='app.log', filemode='a',
    format='%(asctime)s - [%(levelname)s] -  %(name)s - '
           '(%(filename)s).%(funcName)s(%(lineno)d) - %(message)s'
)
logging.debug('This is a debug message')
logging.info('This is an info message')
logging.warning('This is a warning message')
logging.critical('This is a critical message')

a = 5
b = 0
try:
    c = a / b
except ZeroDivisionError as e:
    logging.error("This is an error message", exc_info=True)


logger = logging.getLogger('another_logger')
logger.info('message from another logger')
```

# Использование handlers

Если нам нужны более широкие возможности для логгирования, например, запись в файл и вывод в консоль одновременно, смена лог файла каждый день, то необходимо использовать hadlers

```python
import logging


# Create a custom logger
logger = logging.getLogger(__name__)
# Create handlers
c_handler = logging.StreamHandler()
f_handler = logging.FileHandler('file.log')
c_handler.setLevel(logging.WARNING)
f_handler.setLevel(logging.ERROR)
# Create formatters and add it to handlers
c_format = logging.Formatter('%(name)s - %(levelname)s - %(message)s')
f_format = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
c_handler.setFormatter(c_format)
f_handler.setFormatter(f_format)
# Add handlers to the logger
logger.addHandler(c_handler)
logger.addHandler(f_handler)
logger.warning('This is a warning')
logger.error('This is an error')
```