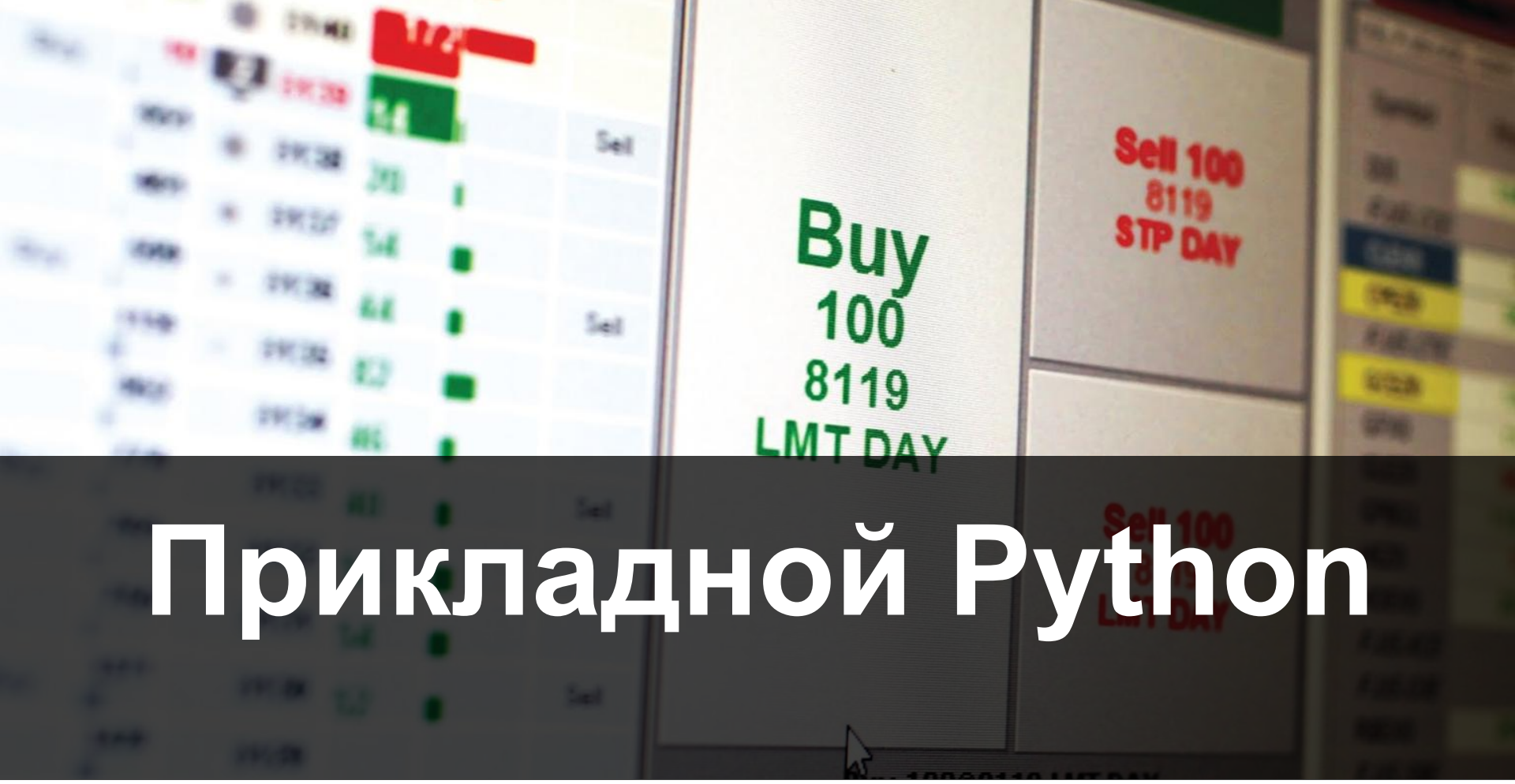


Прикладной Python

Подгорнов Алексей

2025

CQG

Структура лекции

- История Python
- Как развивается Python
- Кто использует Python и почему он популярен
- Как работает Python
- Синтаксис языка (переменные, циклы, условия, классы и т.д.)
- Объекты и сборка мусора в Python
- Python Global Interpreter Lock (GIL)
- Выбор GUI библиотеки для Geo Client (десктопный клиент для Geo Service)
- Более детально рассмотрим Geo Client
- Как запустить Geo Client

История

- В конце 70-х начале 80-х началась разработка ABC

Гвидо ван Россум

CQG

История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году

Гвидо ван Россум

CQG

История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC

Гвидо ван Россум

CQG

История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1989 году Гвидо начал приступил к разработке Python

Гвидо ван Россум

CQG

История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1989 году Гвидо начал приступил к разработке Python
- В феврале1991 года был опубликован код Python 0.9.0

Гвидо ван Россум

CQG

История

- В конце 70-х начале 80-х началась разработка ABC
- Присоединился к команде в 1983 году
- 1986 - 1987 году прекращена разработка ABC
- 1989 году Гвидо начал приступил к разработке Python
- В феврале1991 года был опубликован код Python 0.9.0
- В январе 1994 года вышел Python 1.0
- На данный момент актуальная версия Python 3.14

Гвидо ван Россум

Python Enhancement Proposals

«PEP расшифровывается как предложение по улучшению Python. PEP - это проектный документ, предоставляющий информацию сообществу Python или описывающий новую функцию для Python или его процессов или среды. PEP должен содержать краткую техническую спецификацию функции и обоснование для этой функции.» - PEP1

https://peps.python.org

CQG

Python Enhancement Proposals

«PEP расшифровывается как предложение по улучшению Python. PEP - это проектный документ, предоставляющий информацию сообществу Python или описывающий новую функцию для Python или его процессов или среды. PEP должен содержать краткую техническую спецификацию функции и обоснование для этой функции.» - PEP1

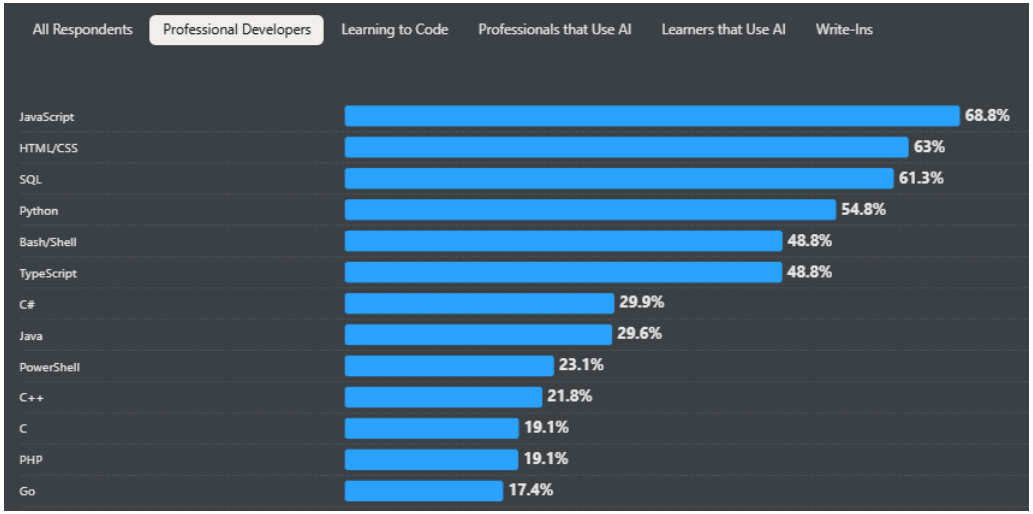
- PEP8 – Руководство по стилю для кода Python
- PEP484 – Подсказки типов
- PEP498 – f-строки
- PEP790 – График выпуска Python 3.15

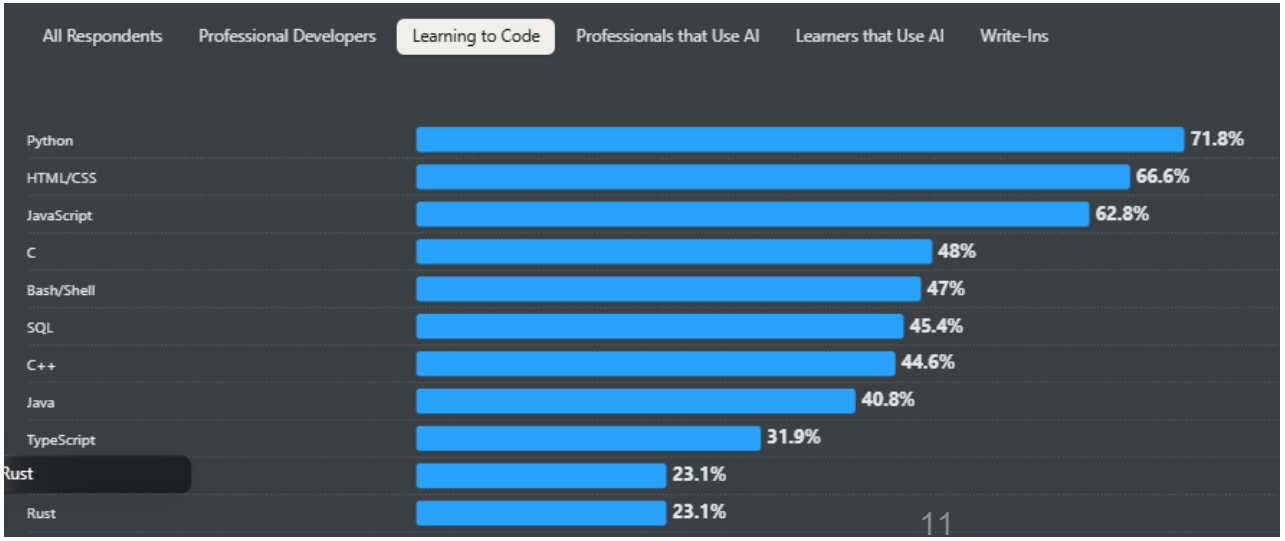
https://peps.python.org

CQG

Популярность

- 55% профессиональных разработчиков используют Python.





- Python на 1 месте (72 %) среди изучающих программирование

https://survey.stackoverflow.co/2025/technology/

Кто использует Python?

- Google
- Dropbox
- Яндекс
- CQG
- и т.д.

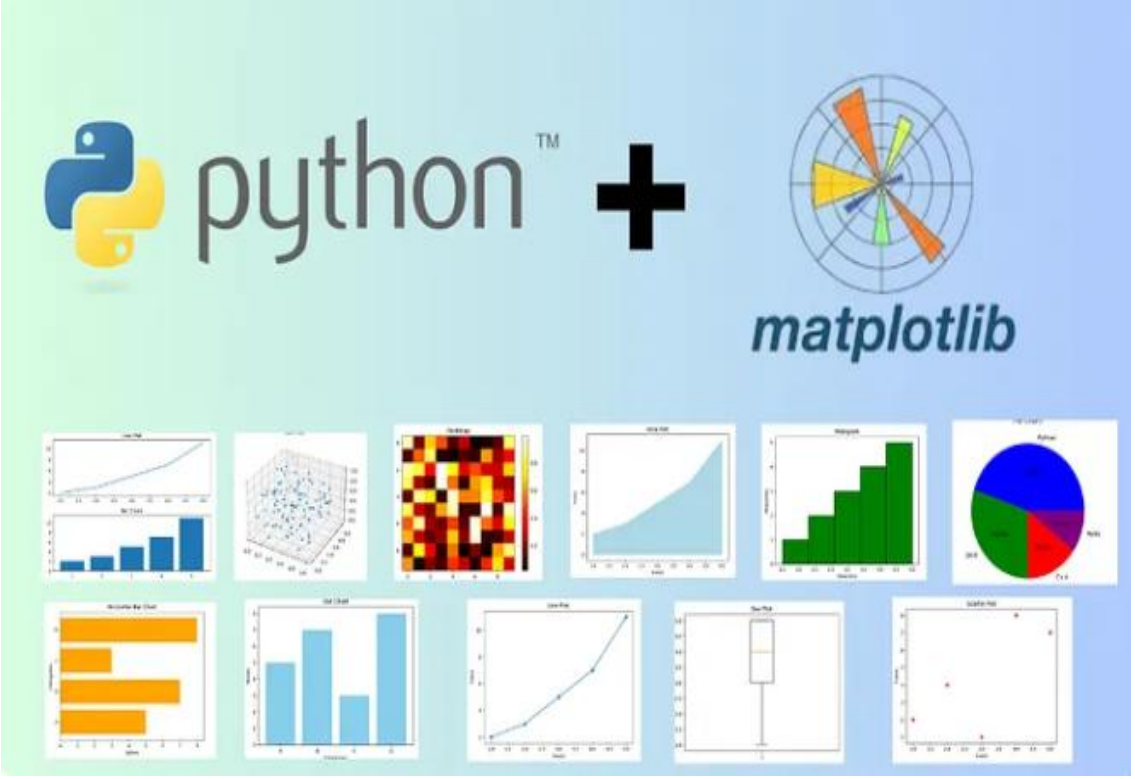
CQG

Кто ещё использует Python?

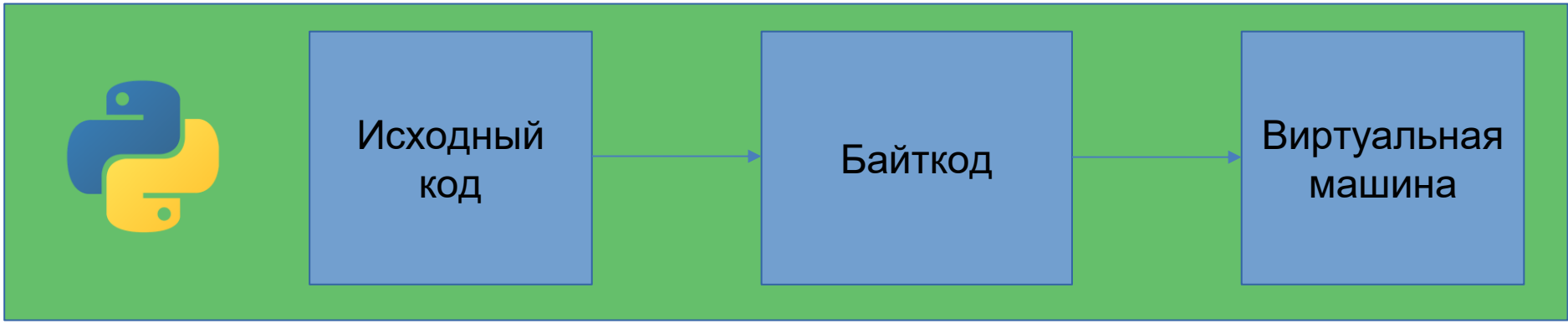
В школах и ВУЗах Python часто используется как язык для обучения программированию, за счет своего простого синтаксиса.

Физики и математики используют Python для сложных вычислений.

Так-же Python является главным языком в машинном обучении и анализе данных.



Python - интерпретируемый



Плюсы

- Кроссплатформенность
- Гибкость
- Рефлексия (модификация программы во время исполнения) и интроспекция (способность программы исследовать тип или свойства объекта во время работы программы)
- Динамическая типизация
- Меньшие затраты времени на разработку

Минусы

- Малая скорость выполнения
- Большая требовательность к ресурсам
- Выше требования к корректности написанных программ

CQG

Пример байткода

```
x = 2023
y = "Hello world"

print(y, x)

y = 1
z = x + y
```

```
  3           0 LOAD_CONST               1 (2023)
              2 STORE_FAST               0 (x)

  4           4 LOAD_CONST               2 ('Hello world')
              6 STORE_FAST               1 (y)

  6           8 LOAD_GLOBAL              0 (print)
             10 LOAD_FAST                1 (y)
             12 LOAD_FAST                0 (x)
             14 CALL_FUNCTION            2
             16 POP_TOP

  8          18 LOAD_CONST               3 (1)
             20 STORE_FAST               1 (y)

  9          22 LOAD_FAST                0 (x)
             24 LOAD_FAST                1 (y)
             26 BINARY_ADD
             28 STORE_FAST               2 (z)
```

Переменные

```
boolean = True                                          # bool
boolean = False                                         # bool
number = 123                                            # int
string = "hello, world!"                                # str
float_number = 1.23                                     # float
list = [1, 2.0, "three"]                                # list
tuple = (1, 2.0, "three")                               # tuple
set = {1, 2, 3, "apple", "banana"}                      # set
frozenset = frozenset([1, 2, 3, "apple", "banana"]) # frozenset
dictionary = {"key1": "value1", "key2": "value2"}   # dict

# Динамическая типизация
number = "string"                                       # str
```

mutable(изменяемые):
list, dict, set

immutable(неизменяемые):
bool, int, float, str, tuple, frozenset

```
list = [1, 2.0, "three"]
list.pop() # [1, 2.0] тот же объект т.к. list mutable
```

```
number = 123
number = 1234 # новый объект, так как int immutable
```


CQG

Работа со строками

```
var = 'World'

# С помощью format
s1 = 'Hello {}'.format(var) # Hello World

# С помощью %
s2 = 'Hello {}' % (var, ) # Hello World

# f-строки
s3 = f'Hello {var}' # Hello World
```

```
s1 = 'abcd'
s2 = 'ef'

# Сложение строк
print(s1 + s2) # abcdef

# Дублирование строк
print('spam' * 3) # spamspamspam

# Длина строки
len(s2) # 4
len('Any sentence') # 12

# Доступ по индексу
print(s1[0]) # a
print(s1[2]) # c
```

```
s = 'abc123'
print(s[1:]) # Удаляет первый символ.
# bc123
print(s[:-1]) # Удаляет последний символ.
# abc12
print(s[2:5]) # Возвращает символы с индекса 2
по 4 (включительно).
# c12
print(s[::2]) # Возвращает каждый второй символ
строки.
# ac2
```


CQG

List comprehension

```
# Создаём список квадратов чисел от 1 до 10
list = [i ** 2 for i in range(1, 11)]

print(list) # [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
# Заполняем список фруктами, содержащими "a"

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]

print(newlist) # ['apple', 'banana', 'mango’]
```

Ветвление и циклы

```python
if password == "admin" and username == "admin":
    print("You are the admin")
else:
    print("You are not the admin")
```

```python
array = [1, 2.0, "three"]
for value in array:
    print(value)

# 1
# 2.0
# three
```

```python
for i in range(7, 11):
    print(i)

# 7
# 8
# 9
# 10
```

https://peps.python.org/pep-0008/#indentation

CQG

Функции

```python
def add(l, r):
    return l + r

def multiply(l: int, r: int) -> int:
    return l * r


add(1, 3) # 4
multiply(3, 4) # 12
multiply("hi!", 5) # "hi!hi!hi!hi!hi!"
```

https://peps.python.org/pep-0484/

https://mypy-lang.org/

CQG

Классы

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height


    def area(self):
        return self.width * self.height

    def __del__(self):
        print("Удаление экземпляра: " + str(self))



rect = Rectangle(15, 15)


print(rect.area())
```

>> 225
>> Удаление экземпляра Rectangle<__main__.Rectangle object at 0x7f5d61e1f850>

Специальные методы

__init__(self, ...): Метод инициализации. Вызывается при создании нового объекта класса.
__del__(self, ...): Метод финализации. Вызывается при удаление объекта класса
__str__(self): Возвращает строковое представление объекта. Вызывается функцией print() или str().
__repr__(self): Возвращает "официальное" строковое представление объекта, которое должно быть максимально информативным.
__call__(self, ...): Позволяет вызывать объект как функцию.
__add__(self, other): Определяет, что происходит при использовании оператора + с объектами этого класса.

Методы сравнения объектов между собой
__lt__(self, other) — определяет поведение оператора сравнения «меньше», <.
__le__(self, other) — определяет поведение оператора сравнения «меньше или равно», <=.
__eq__(self, other) — определяет поведение оператора «равенства», ==.
__ne__(self, other) — определяет поведение оператора «неравенства», !=.
__gt__(self, other) — определяет поведение оператора сравнения «больше», >.
__ge__(self, other) — определяет поведение оператора сравнения «больше или равно», >=.

https://tproger.ru/articles/gajd-po-magicheskim-metodam-v-python

Классы (наследование)

```
class MapWidget(TkinterMapView):
    """Wrapper for the map widget functionality."""


    def __init__(self, parent):
        super().__init__(parent)


    def set_marker(self, latitude: float, longitude: float, name: str):
        """Set marker and position."""
        self.delete_all_marker()
        self.set_position(latitude, longitude)
        super().set_marker(latitude, longitude, text=name)
```


CQG

Менеджер контекста

```
# Пример менеджера контекста
class Context:
    def __enter__(self):
        print("__enter__")


    def method(self):
        return "In method"


    def __exit__(self, *args):
        print("__exit__")



with Context() as c:
    print(c.method())


# __enter__
# In method
# __exit__
```

```
# Без менеджера контекста
file = open("file.txt", "r")
try:
    # Действия с файлом
    content = file.read()
    print(content)
finally:
    file.close()
```

```
# С менеджером контекста
with open("file.txt", "r") as file:
    content = file.read()
    print(content)
```

Декораторы

```
def log_with_level(level):
    def decorator(func):
        def wrapper(*args, **kwargs):
            print(f"{level}: Calling function '{func.__name__}' with args: {args},
             kwargs: {kwargs}")
            result = func(*args, **kwargs)
            print(f"{level}: Function {func.__name__} is complete")
            return result
        return wrapper
    return decorator


@log_with_level("INFO")
def greet(name):
    print(f"Hello, {name}!")


greet("World")
```

Объекты и ссылки

```python
a = []
b = a


a.append(1)


print(b) # [1]


print(a == b) # True
print(a is b) # True
```

```cpp
std::vector<int> a;
auto b = a;


a.push_back(1);


print(b); # []


print(a == b); # false
print(&a == &b); # false
```

CQG

Объекты и ссылки

```python
a = []
b = a


a.append(1)


print(b) # [1]


print(a == b) # True
print(a is b) # True
```

```cpp
std::vector<int> a;
auto& b = a;


a.push_back(1);


print(b); # [1]


print(a == b); # true
print(&a == &b); # true
```

Объекты и ссылки

```python
a = 12
b = a


a += 7


print(b) # 12


print(a == b) # False
print(a is b) # False
```

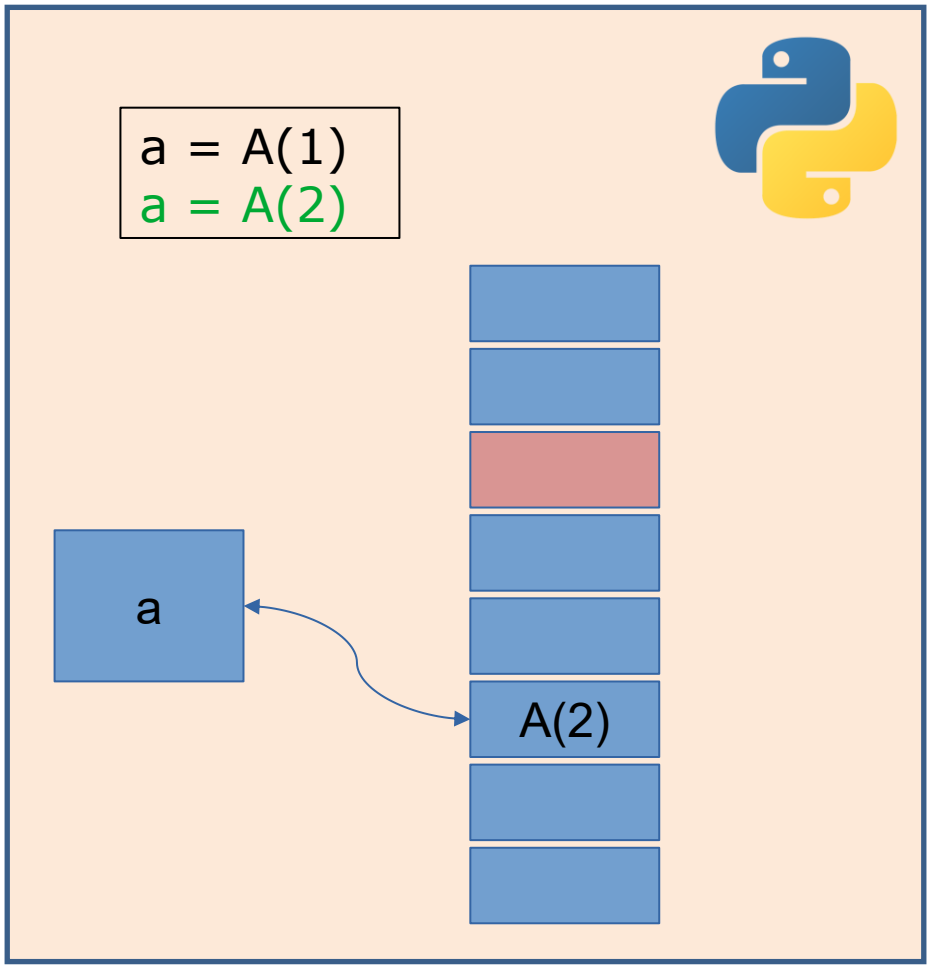
```cpp
int a = 12;
int b = a;


a += 7;


print(b); # 12


print(a == b); # false
print(&a == &b); # false
```

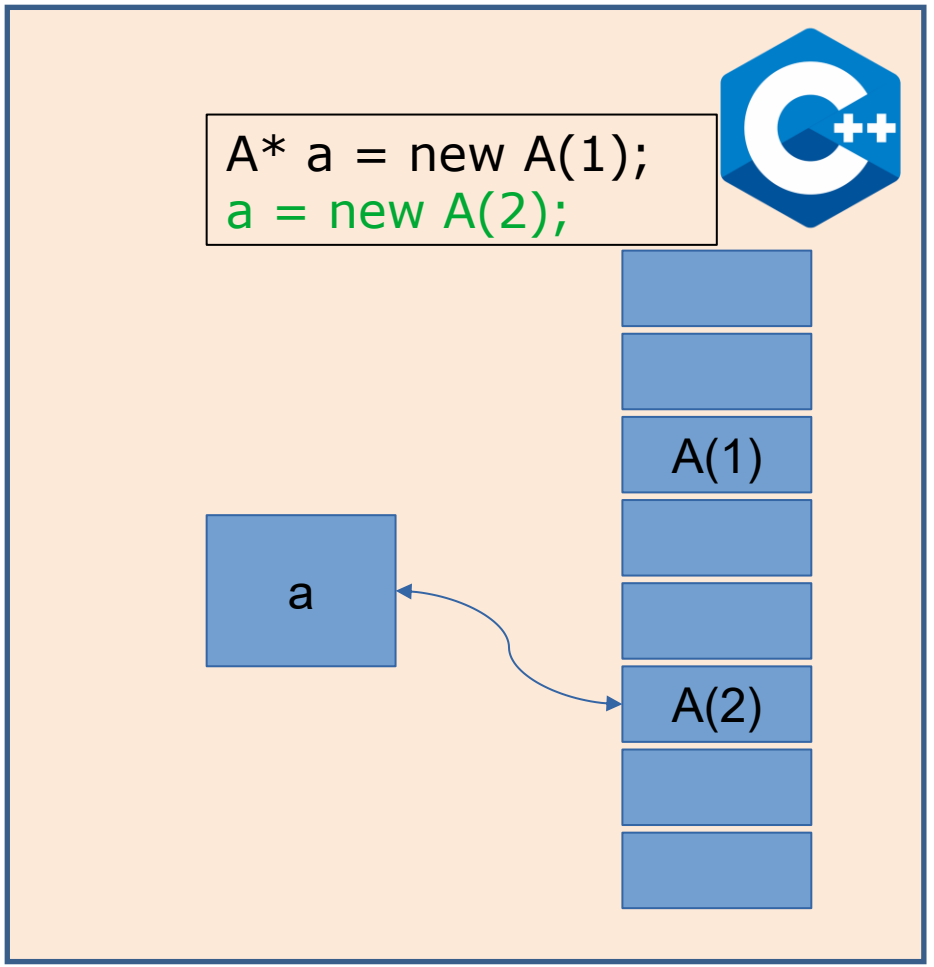
Объекты и ссылки

```python
a = 12
b = a


a += 7


print(b) # 12


print(a == b) # False
print(a is b) # False
```

```cpp
int a = 12;
int& b = a;


a += 7;


print(b); # 19


print(a == b); # true
print(&a == &b); # true
```

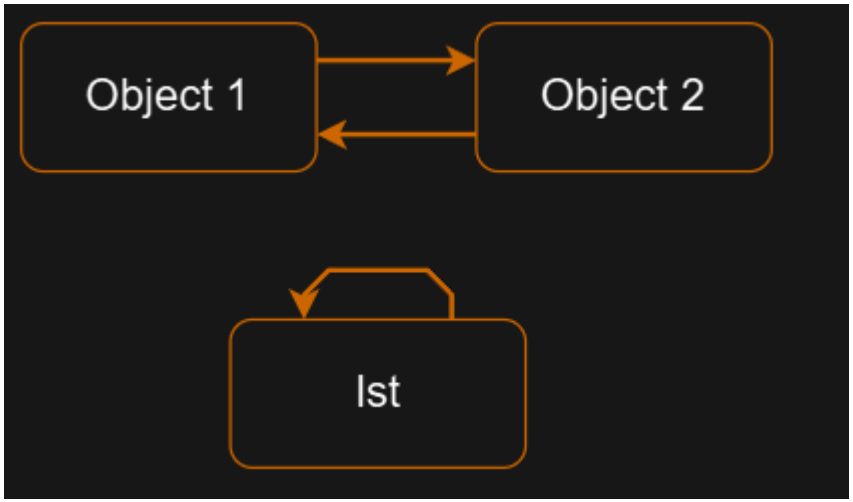
Сборка мусора





Сборка мусора (дополнительный сборщик мусора)

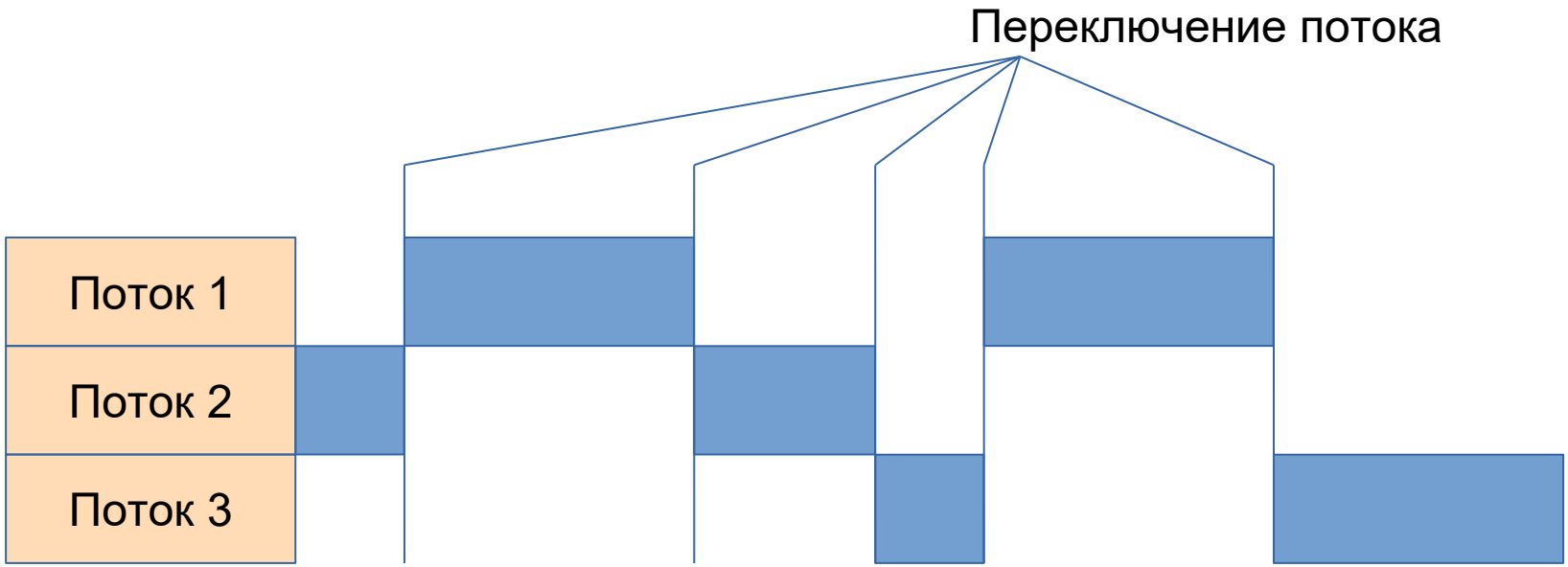
Нужен он, что бы определять циклические ссылки



В отличие от алгоритма подсчета ссылок, циклический сборщик мусора не работает в режиме реального времени и запускается периодически. Каждый запуск сборщика создаёт микропаузы в работе кода, поэтому CPython (стандартный интерпретатор) использует различные эвристики, для определения частоты запуска сборщика мусора.

CQG

GIL Global Interpreter Lock



Плюсы

- Защищает интерпретатор от ошибок многопоточности

Минусы

- Не защищает ваш код от ошибок многопоточности
- Потоки выполняются последовательно

https://wiki.python.org/moin/GlobalInterpreterLock

CQG

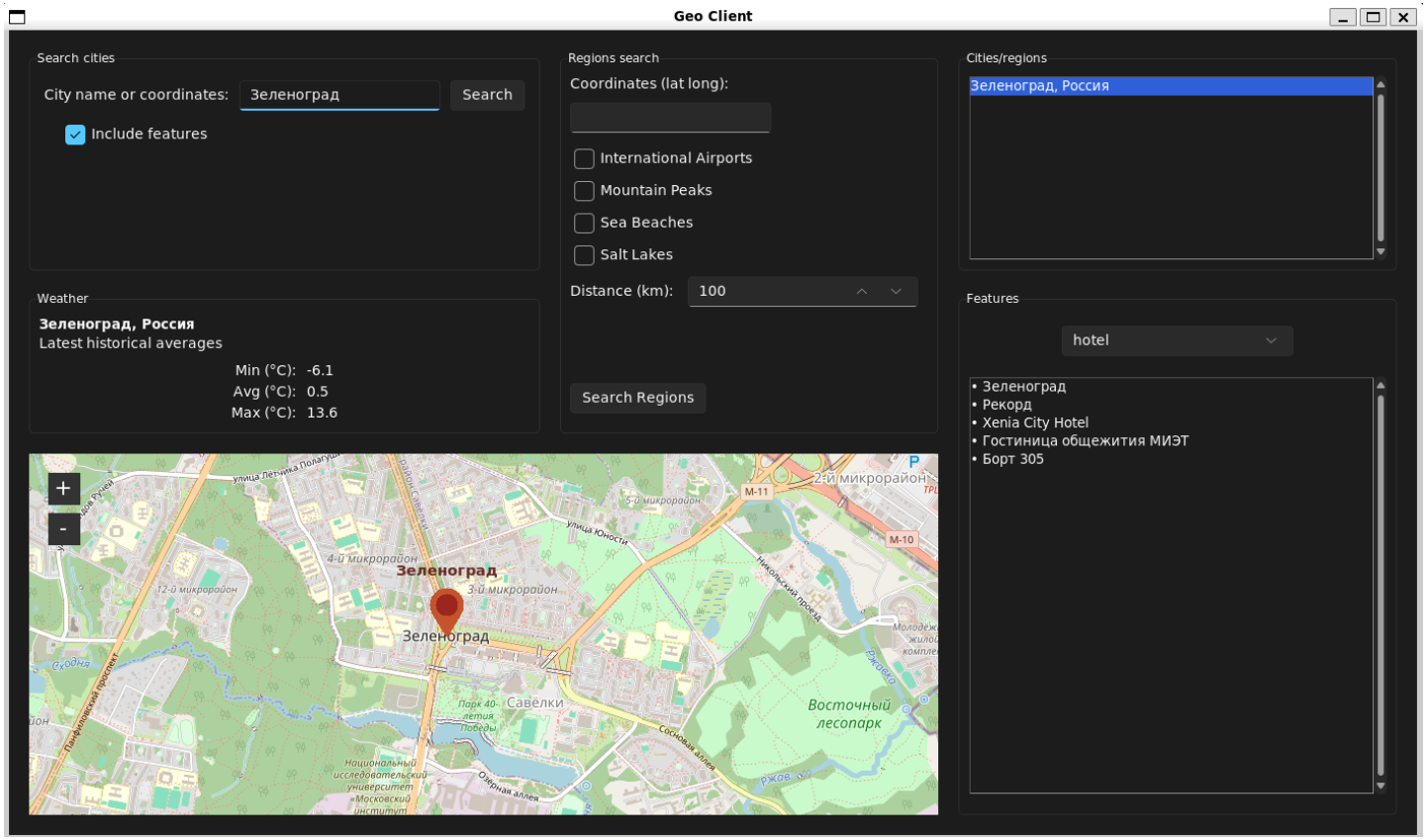
Где уместен Python?

- Бэкенд (Django для сложных приложений и Flask/FastAPI для микросервисов**)**
- Data Science (pandas и numpy стали стандартом де-факто)
- Научные вычисления (SciPy и SymPy)
- Машинное обучение (TensorFlow и PyTorch)
- Автоматизация
- GUI приложения(Tkinter, PyQt/PySide, wxPython, Kivy)

Выбор библиотеки для GUI

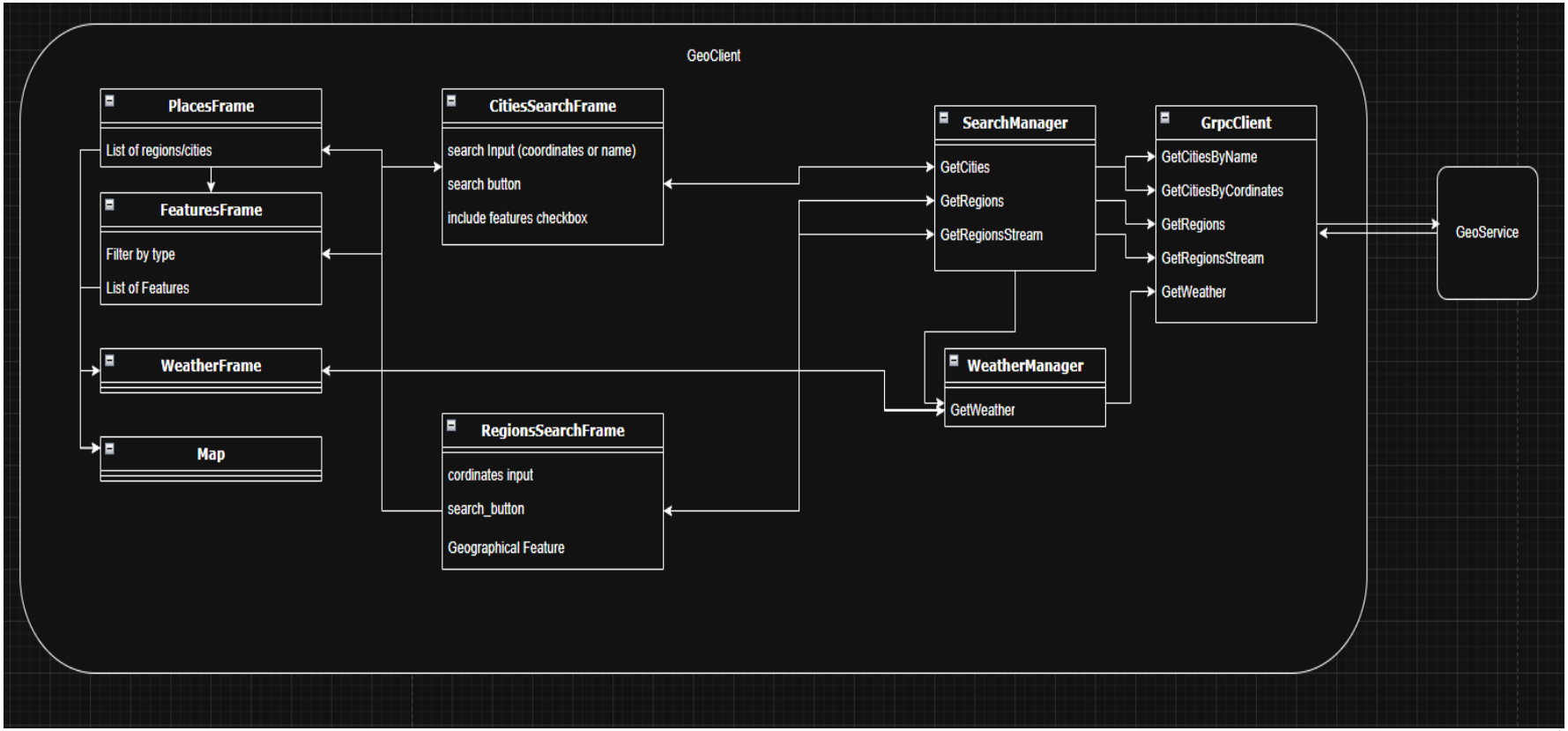
	Tkinter	PyQt/PySide	wxPython	Kivy
Сложность изучения	Низкая	Высокая	Средняя	Средняя
Внешний вид	Базовый	Профессиональный	Нативный	Кастомный
Производительность	Высокая	Средняя	Средняя	Низкая
Лицензия	Python	GPL/LGPL	wxWindows	MIT
Кроссплатформенность	✅	✅	✅	✅
Мобильные устройства	❌	❌	❌	✅
Мультитач	❌	❌	❌	✅

Geo Client



- https://docs.python.org/3/library/tk.html

Geo Client


GeoClient
PlacesFrame
List of regions/cities
FeaturesFrame
Filter by type
List of Features
WeatherFrame
Map
CitiesSearchFrame
search Input (coordinates or name)
search button
include features checkbox
RegionsSearchFrame
cordinates input
search_button
Geographical Feature
SearchManager
GetCities
GetRegions
GetRegionsStream
WeatherManager
GetWeather
GrpcClient
GetCitiesByName
GetCitiesByCordinates
GetRegions
GetRegionsStream
GetWeather
GeoService

Geo Client

```
self.root = tk.Tk()

"""Configure the main window geometry and title."""
self.root.title('Geo Client')
self.root.geometry('1400x800')
self.root.minsize(800, 600)
sv_ttk.set_theme('dark')
```

```
        self.search_entry = ttk.Entry(self)
        self.search_entry.grid(column=1, row=0, padx=5, pady=5, sticky='W')
        self.search_entry.bind('<Return>', lambda e: self._handle_search_input())


        self.search_button = ttk.Button(
            self,
            text='Search',
            command=self._handle_search_input
        )
```

.

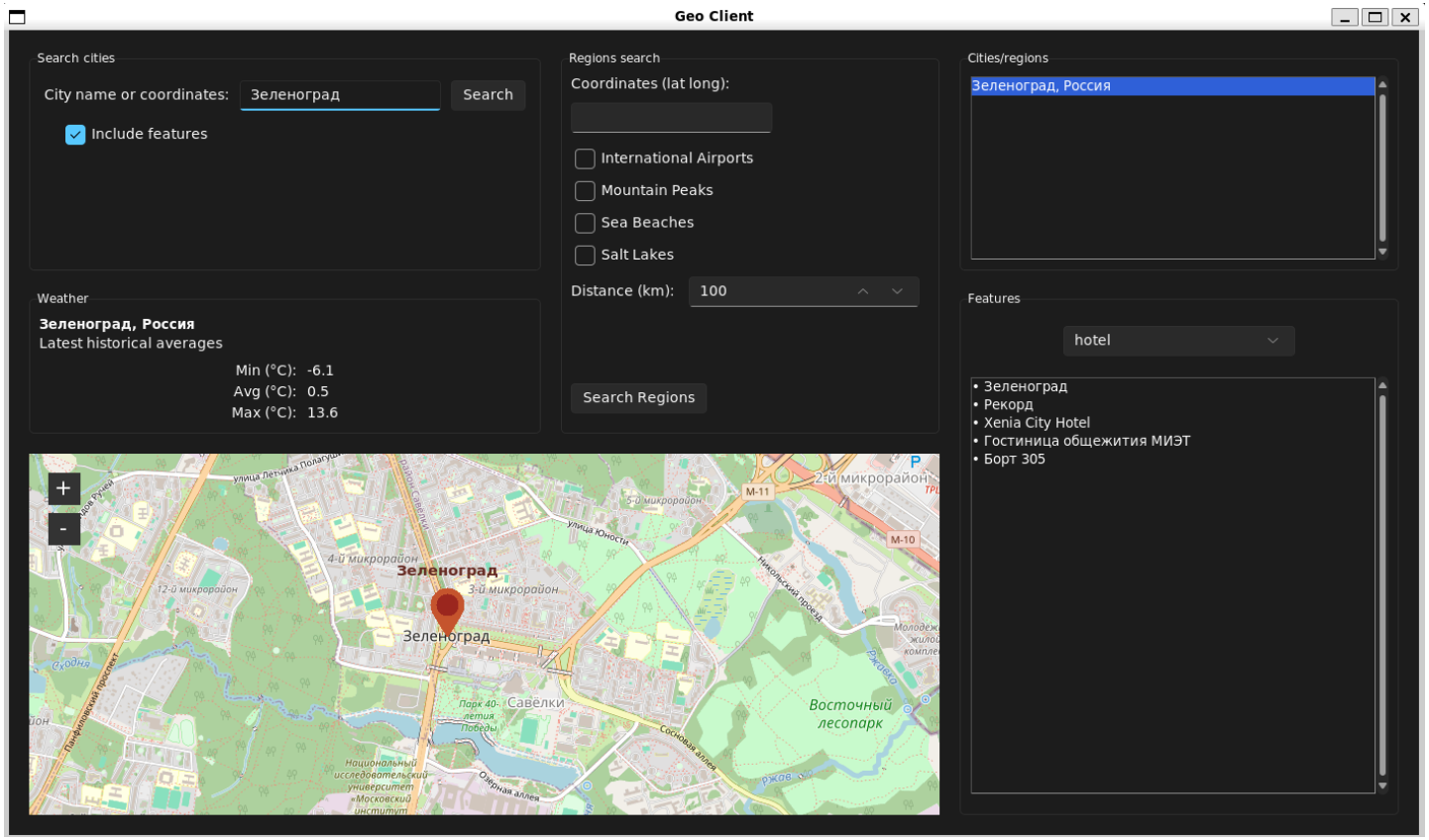
CQG

Менеджеры геометрии в Tkinter

Характеристика	pack()	grid()	place()
Простота	✅ Очень простой	✅ Средняя (логика сетки)	⚠️ Сложный (ручное управление)
Гибкость	❌ Ограничен (выравнивание в ряд)	✅ Высокая (сетка + объединение)	✅ Максимальная (координаты)
Контроль позиции	❌ Минимальный	✅ Точно через строки/столбцы	✅ Полный (x, y, ширина, высота)
Сложные макеты	❌ Не рекомендован	✅ Идеально подходит	⚠️ Возможно, но неудобно
Пример кода	label.pack(side='top')	label.grid(row=0, column=0)	label.place(x=10, y=20)

Важно!
Нельзя смешивать pack() и grid() в одном контейнере(блоке, который содержит несколько виджетов).

Geo Client



- https://docs.python.org/3/library/tk.html

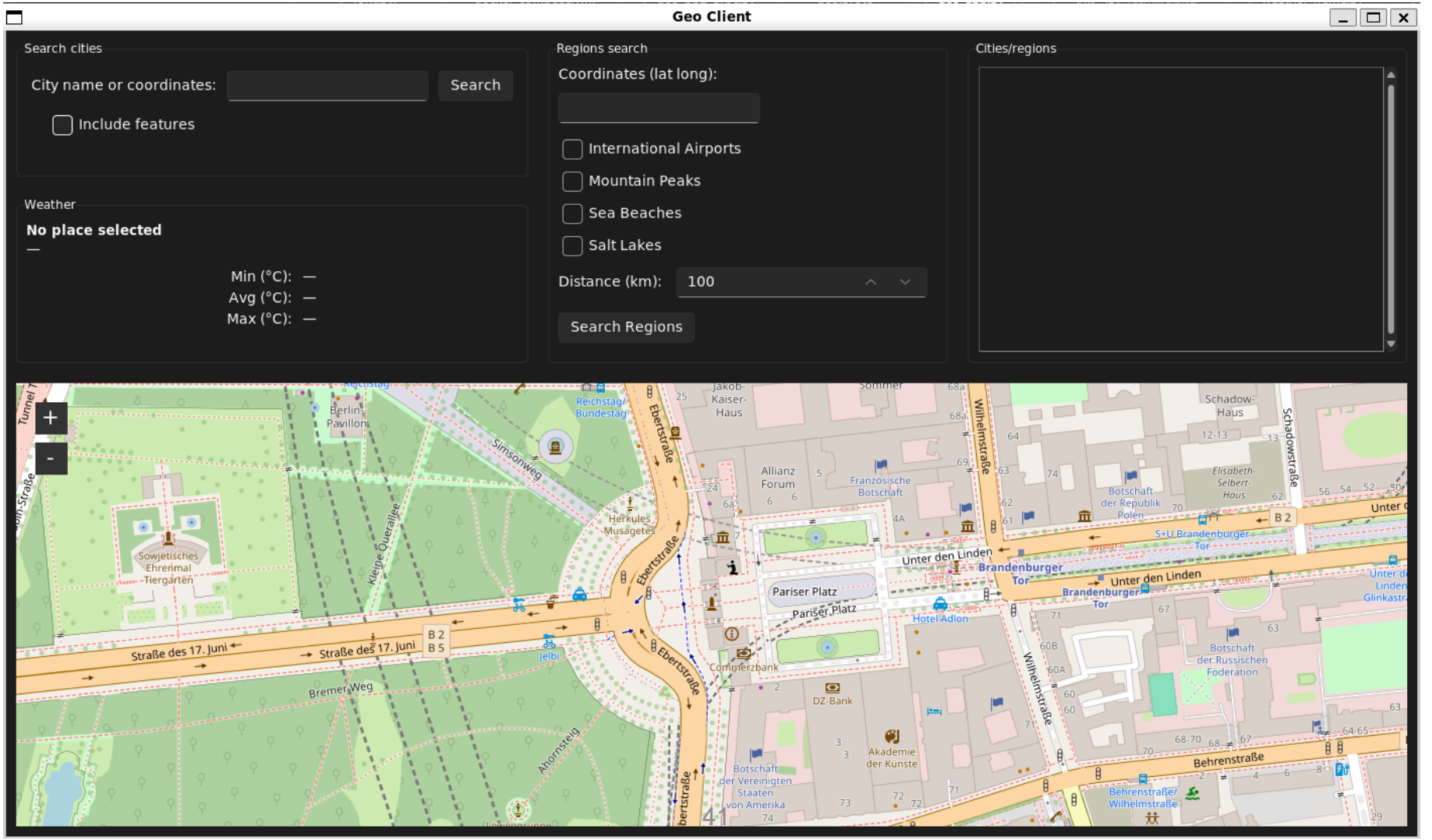
Использование Grid в Geo Client

```
 def _setup_widgets(self):
        self.root.grid_columnconfigure(3, weight=1)
        self.root.grid_rowconfigure(2, weight=1)

        # Create UI components
        self.city_search_frame = CitySearchFrame(self.root, self.search_manager,
self.display_results)
        self.city_search_frame.grid(column=0, row=0, padx=10, pady=10, sticky='NSEW')

        self.weather_frame = WeatherFrame(self.root)
        self.weather_frame.grid(column=0, row=1, padx=10, pady=10, sticky='NSEW')

        self.map_widget = MapWidget(self.root)
        self.map_widget.grid(column=0, row=2, columnspan=4, padx=10, pady=10,
sticky='NSEW')

        self.regions_search_frame = RegionsSearchFrame(self.root, self.search_manager,
self.display_results)
        self.regions_search_frame.grid(column=3, row=0, rowspan=2, padx=10, pady=10,
sticky='NSEW')
```

.

CQG

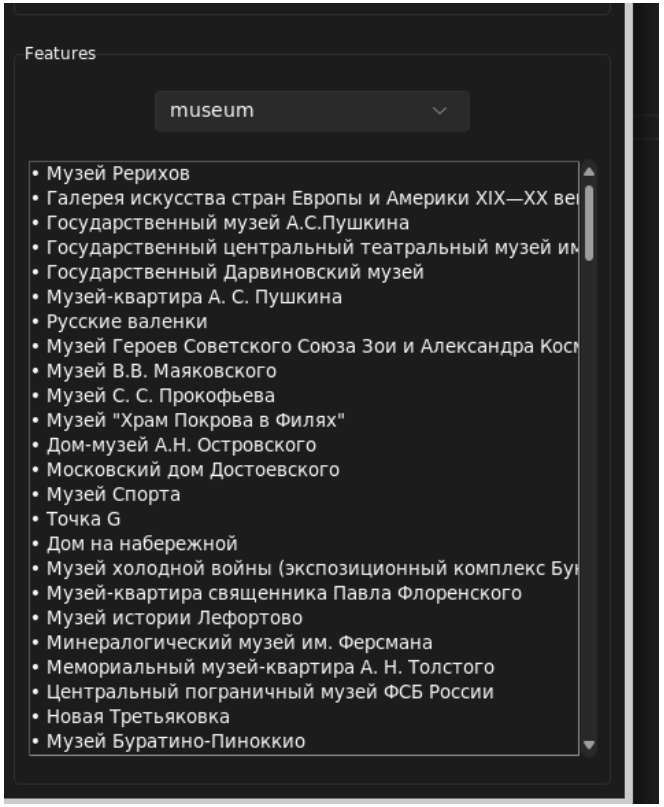
Geo Client


Geo Client
Search cities
City name or coordinates:
Search
Include features
Weather
No place selected
—
Min (°C): —
Avg (°C): —
Max (°C): —
Regions search
Coordinates (lat long):
International Airports
Mountain Peaks
Sea Beaches
Salt Lakes
Distance (km): 100
Search Regions
Cities/regions
Straße des 17. Juni
Unter den Linden
Brandenburger Tor
Pariser Platz
Hotel Adlon
Reichstag/ Bundestag
Sowjetisches Ehrenmal Tiergarten
Botschaft der Russischen Föderation
Botschaft der Vereinigten Staaten von Amerika
Akademie der Künste
Behrenstraße

Использование Pack в Geo Client

Пример из FeaturesFrame:

```
def _setup_ui(self):
    self._tags_combobox.pack(side='top', pady=10)
    self._features_list.pack(side='left', fill='y',
expand=True)
    self._features_scrollbar.pack(side='right',
fill='y')
```



.

Geo Client работа с GeoService

```python
class GrpcClient:
    def __init__(self, host: str, port: int):
        """Initialize the gRPC client."""
        self.channel = grpc.insecure_channel(f'{host}:{port}')
        self.stub = geo_pb2_grpc.GeoStub(self.channel)


    def __del__(self):
        """Clean up the channel when the object is destroyed."""
        self.channel.close()


    def get_cities_by_name(self, name: str, include_details: bool) ->
geo_pb2.CitiesResponse:
        """Search cities by name using gRPC API."""
        request = geo_pb2.CitiesRequest(name=name, include_details=include_details)
        return self.stub.GetCities(request)
```

```python
# Создание объекта GrpcClient

client = GrpcClient('localhost', 50051)
```

https://grpc.github.io/grpc/python/grpc.html#

CQG

Geo Client поиск городов и регионов

```
def _perform_search(self, task: Callable, on_response: Callable, is_stream: bool = False):
        """Run search tasks asynchronously with proper UI marshalling."""
        if self.is_searching():
            return
        self._searching_event.set()

        def wrapped_task():
            try:
                if is_stream:
                    for response in task():
                        self._process_response(response, on_response)
                else:
                    response = task()
                    self._process_response(response, on_response)
            except Exception as e:
                self._ui_dispatch(lambda e=e: self._handle_search_error(e))
            finally:
                self._searching_event.clear()
                self._ui_dispatch(lambda: self._toggle_search_ui(True))

        threading.Thread(target=wrapped_task, daemon=True).start()

self._perform_search(lambda: self.client.get_regions(lat, lon, distance, mask,
     properties), on_response)

self._perform_search(lambda: self.client.get_regions_stream(lat, lon, distance, mask,
     properties), on_response, is_stream=True)
```

Geo Client поиск городов и регионов

```
def _process_response(self, response, on_response: Callable) -> None:
        """Update internal state (thread-safe) and fetch weather."""
        if response is None:
            return

        # Determine which field is present and extract places
        if hasattr(response, 'cities'):
            new_places = list(response.cities)
        else:
            new_places = list(response.regions)

        with self._state_lock:
            self._places.extend(new_places)

        self._ui_dispatch(lambda r=response: on_response(new_places))
        # Uncomment to get weather
        # self._weather.fetch_weather(new_places)
```


CQG

Запрос погоды

```
self._executor = ThreadPoolExecutor(max_workers=2, thread_name_prefix='weather')
self._cache: Dict[utils.LocationKey, geo_pb2.Weather] = {}

    def fetch_weather(self, places: Iterable) -> None:
        """Deduplicate and schedule background batches for given places."""
        points: List[geo_pb2.Point] = []
        keys: List[utils.LocationKey] = []

        with self._lock:
            for place in places:
                key = utils.key_for_location(place)
                if key in self._cache:
                    continue
                keys.append(key)
                points.append(geo_pb2.Point(latitude=key[0], longitude=key[1]))


        if not points:
            return
        # Batch submit
        for i in range(0, len(points), self._batch_size):
            batch = points[i: i + self._batch_size]
            self._executor.submit(self._fetch_batch, batch)
```

Стрим и запрос погоды

search_manager.py

```
# Uncomment to get weather
# self._weather.fetch_weather(new_places)
```

regions_search_frame.py

```
self.search_manager.search_regions(lat, lon, distance, self.mask, properties,
                                   self._on_response)
# self.search_manager.search_regions_stream(
#     lat, lon, distance, self.mask, properties, self._on_response)
```

Как запустить Geo Client на Windows

- Установить Python
- git clone https://github.com/cgginternship/geo-client.git GeoClient
- cd .\GeoClient\
- Python -m pip install --upgrade pip
- python -m venv .venv
- .venv\Scripts\activate
- pip install -r requirements.txt
- python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. .\proto\geo.proto
- python .\src\main.py (или открыть из любой IDE, которую вы используете для Python)

Как запустить Geo Client на WLS (Windows 11)

- $ cd ~
- $ git clone https://github.com/cqginternship/geo-client.git geoClient на wsl/linux
- Запустить Docker/Rancher Desktop
- $ cd ~/geoClient
- $ code .
- Ctrl+Shift+P Dev Containers: Reopen in Container
- Запускать/отлаживать ctrl+F5/F5

Как запустить Geo Client на WLS (Windows 10)

- $ cd ~
- $ git clone https://github.com/cqginternship/geo-client.git geoClient

1. Установить VcXsrv Windows X Server https://sourceforge.net/projects/vcxsrv/files/latest/download
2. Запустить XLaunch
3. Нажать Next, Next
4. Изменить следующие настройки: "Primary selection" – unchecked, "Disable access control" - checked
5. Нажать Next, Finish
6. Зайти в WSL и ввести: echo "export DISPLAY=<your Windows host>:0" >> ~/.bashrc

Где <your Windows host> это имя вашего компьютера, можно посмотреть в System->Full Device name

- Запустить Docker/Rancher Desktop
- $ cd ~/geoClient
- $ code .
- Ctrl+Shift+P Dev Containers: Reopen in Container
- Запускать/отлаживать ctrl+F5/F5

Как запустить Geo Client на Linux

- $ cd ~
- $ git clone https://github.com/cqginternship/geo-client.git geoClient
- Запустить VS code, открыть папку с geoClient
- Ctrl+Shift+P Dev Containers: Reopen in Container
- Запускать/отлаживать ctrl+F5/F5


Buy
100
8119
LMT DAY
Sell 100
8119
STP DAY
Sell 100
8119
LMT DAY
Buy 100@8119 LMT DAY

Q&A