

# MUNDARIJA

KIRISH.....	5
1-BOB. Python dasturlash tiliga kirish .....	8
1. Pythonda dasturlash asoslari bo'yicha boshlang'ich ma'lumotlar.....	8
2. Ma'lumotlar turlari va o'zgaruvchilar.....	19
3. Operatorlar.....	20
4. Shartli ifodalar (Conditional Statements) .....	23
5. Funksiyalar .....	25
6. Sikllar .....	28
2-BOB. Ma'lumotlar bazalari. Web dasturlashga kirish .....	31
1. Ma'lumotlar bazalari (Databases) .....	31
2. Veb dasturlashga kirish .....	34
3. Pythonning Veb Frameworklari .....	37
4. Django Web Framework .....	39
5. Django yordamida web loyiha qurish.....	40
3- BOB. Suv xo'jalik tashkilotlari uchun nazorati tizimini ishlab chiqish .....	44
1. Loyihaning ishlash tizimi .....	44
2. Loyihani modellashtirish .....	47
3. Dasturiy ta'minotning API qismini qurish .....	51
4. Loyiha uchun Telegram Bot .....	57
XULOSA.....	60
FOYDALANILGAN MANBALAR .....	61

Adabiyotlar .....	61
Internet manbalar.....	61
Foydalanilgan dasturiy vositalar manzillari .....	61
ILOVALAR.....	62

## KIRISH

Raqamli texnologiyalar nafaqat mahsulotlar va xizmatlar sifatini oshiradi, xarajatlarni kamaytiradi. Dasturiy vositalar yordamida doimiy qilinadigan ishlarni avtomatlashtirish imkoniyatlari IT sohasining kelajagida muhim o'rin egallaydi va suv xo'jaligi sohasidagi monitoring va nazorat jarayonlarini yanada isloh etadi. Ushbu loyiha xalqimiz uchun suv resurslarini samarali va hamyonroq ishlatish imkonini yaratadi. Bu loyiha suv ta'minoti va xo'jaligi sohasida yaxshi hududiy va milliy imkoniyatlarga ega bo'lishini va bizning mamlakatimizning suv resurslarini muvozanatli ravishda boshqarishimizni ommaga foydali bo'ladi.

Yer osti va yer usti suvlarini nazoratini yuritish uchun ko'plab islohotlar olib borilmoqda. Ushbu qilanadigan loyiha ushbu nazorat qilish tiziming dasturiy tomoni hisoblanadi. Loyihada asosiy to'planadigan ma'lumotlar tizimning hardware tomoni(qurilmalar)dan keladigan ma'lumotlar hisoblanadi. Ushbu dasturiy ta'minotni qurish davomida hardware muhandislar bilan birga ishlar olib borildi. Qurilmalar va dasturiy ta'minot o'rtasida ma'lumotlar almashinish uchun RESTful API dan foydalanilgan.

Shu bilan birga, dasturiy ta'minot va avtomatlashtirish, suv xo'jaligi sohasida innovatsion yondashuv va o'zaro almashishning katta roli va ahamiyatiga ega. Bu loyiha suv resurslarini samarali boshqarish va foydalanish, hamda hududiy va milliy imkoniyatlarni rivojlantirishga yordam beradi.

Loyiha nomi:

- Yer osti va yer usti suvlarini nazorat qilish

Loyiha qismlari:

- Admin dashboard – super admin uchun boshqaruv paneli
- Foydalanuvchi tashkilotlar uchun platforma

- Qurilmalarga ma'sul shaxslar uchun telegram bot dasturi

Loyiha manzillari:

- gidrosmart.uz/admin – super admin uchun web ilova manzili
- gidrosmart.uz – foydalanuvchilar uchun platforma manzili
- t.me/managewaterbot – telegram bot dasturi manzili

Loyiha imkoniyatlari:

- Kanal va quduqlarda suv resursini nazorat qilish
- Suv holatini doimiy nazorat qilish

Loyiha vazifasi:

- Telegram bot va Admin dashboard orqali yangi qurilmalar bo'glash, boshqarish
- Kanallar soylardan har sekundda (minutda, soatda) o'tayotgan suv miqdorini ma'lumotlar bazasiga saqlash
- Quduqlardagi suv miqdorini va suv sho'rligini kunlik tekshirib malumotlar bazasiga saqlash
- Serverga kelib tushgan ma'lumotlarni Suv xo'jaligi vazirligi serveriga yuborish
- Admin panel va Telegram bot orqali qurilmalar holatini va kelib tushayotgan ma'lumotlarni tekshirib turish
- Qurilmalar joyidan qo'zg'alganda telegram bot orqali qurilma egasiga ogohlantirish yuborish
- Qurilmani ulashish (qurilma egasidan boshqalar ham qurilmani kuzatib tura oladi)

Loyihaning dasturiy qismi:

- Server qism – Python (Django, Django Rest Framework), PostgreSQL
- Telegram bot – Python (Aiogram), PostgreSQL

- Web site (Admin dashboard) – HTML, CSS, JavaScript, Tailwindcss, JQuery

## **1-BOB. Python dasturlash tiliga kirish**

### **1. Pythonda dasturlash asoslari bo'yicha boshlang'ich ma'lumotlar**

Python bu *yuqori darajali, umimiy maqsadli, interpreter* dasturlash tili hisoblandi.

#### *1. Yuqori darajali (High-level)*

Python yuqori darajali dasturlash tili hisoblanib bu uni o'rganish va ishlatishga qulay ekanligini bildiradi. Python dasturlarni samarali ishlab chiqish uchun kompyuterning tafsilotlarini tushunishingizni talab qilmaydi.

#### *2. Umumiy maqsadli*

Python umumiy maqsadli tildir. Bu shuni anglatadiki, siz Python'dan turli maqsadlarda foydalanishingiz mumkin, jumladan:

- Web dasturlar
- Big data dasturlari
- Testlash
- Avtomatlashtirish
- Ma'lumotlar ilmi - Data science
- Mashinani o'qitish - Machine learning
- Sun'iy intellekt - AI (Artificial Intelligence)
- Desktop dasturlar
- Mobil dasturlar

Relyatsion ma'lumotlar bazalaridan ma'lumotlarni so'rash uchun ishlatilishi mumkin bo'lgan SQL kabi maqsadli til.

#### *3. Interpreted*

Python interpreted dasturlash tili hisoblanadi. Bunda Python dasturini ishlab chiqish uchun Python kodini faylga yozasiz. Manba kodini bajarish uchun uni kompyuter tushunadigan mashina tiliga aylantirish kerak. Va Python tarjimoni Python dasturi bajarilganda manba kodini satrma-satr bir marta mashina kodiga aylantiradi. Java va C#

kabi kompilyatsiya qilingan tillar dasturni ishga tushirishdan oldin butun manba kodini kompilyatsiya qiluvchi kompilyatordan foydalanadi.

### **Nima uchun python dasturlash tilidan foydalanamiz?**

Bu savolga juda ko‘plab sabablarni ko‘rsatishimiz mumkin:

- Python sizning mahsuldorligingizni oshiradi. Python sizga murakkab muammolarni kamroq vaqt va kamroq kod qatorlarida hal qilish imkonini beradi. Pythonda prototip yaratish juda tez.
- Python web ilovalardan tortib Data Science va Machine Learning‘gacha bo‘lgan ko‘plab sohalarda yechimga aylanadi
- Pythonni boshqa dasturlash tillari bilan solishtirganda o‘rganish juda oson. Python sintaksisi aniq va chiroyli.
- Python ko‘plab kutubxonalar va framework’larni o‘z ichiga olgan katta ekotizimga ega.
- Python’ning cross-platform ekanligi. Python dasturlash tilida yozilgan dasturlar Windows, Linux, macOS hattoki android tizimlarida bir xilda ishlaydi.
- Python juda katta jamoaga ega. Qachon muammoga duch kelib qolsangiz, faol hamjamiyatdan yordam olishingiz mumkin.
- Python dasturchilariga bo‘lgan kuchli talab.

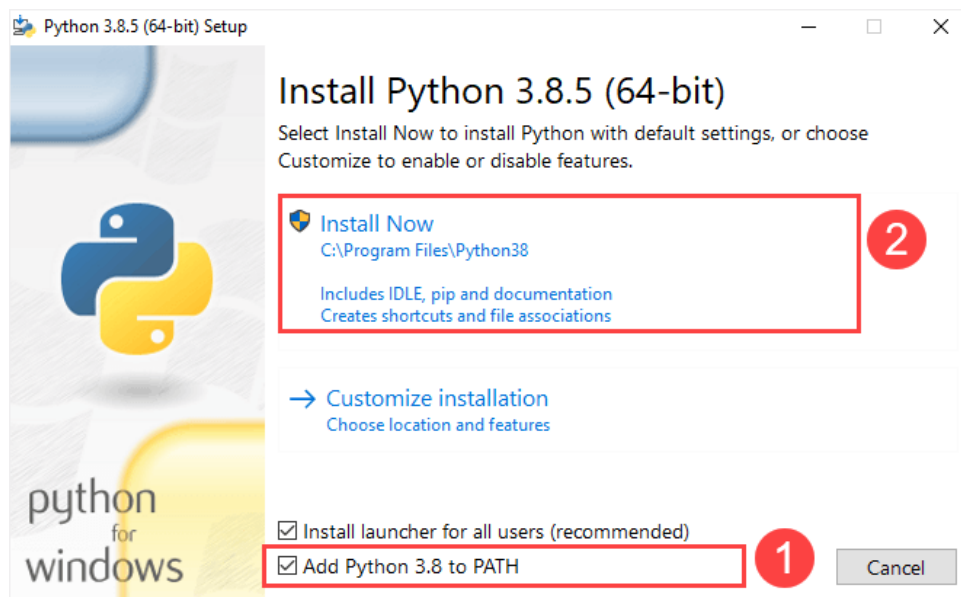
### **Python versiyalari**

Pythonning ikkita asosiy versiyasi mavjud:

- Python 2.x 2000 yilda chiqarilgan. Oxirgi versiyasi 2010 yilda chiqarilgan 2.7. Yangi loyihalarda foydalanish tavsiya etilmaydi.
- Python 3.x 2008 yilda chiqarilgan. Asosan, Python 3 Python 2 bilan mos kelmaydi. Yangi loyihalarangiz uchun Python 3 ning so‘nggi versiyalaridan foydalanish maqsadga muvofiq bo‘ladi.

## Python'ni kompyuterga o'rnatish

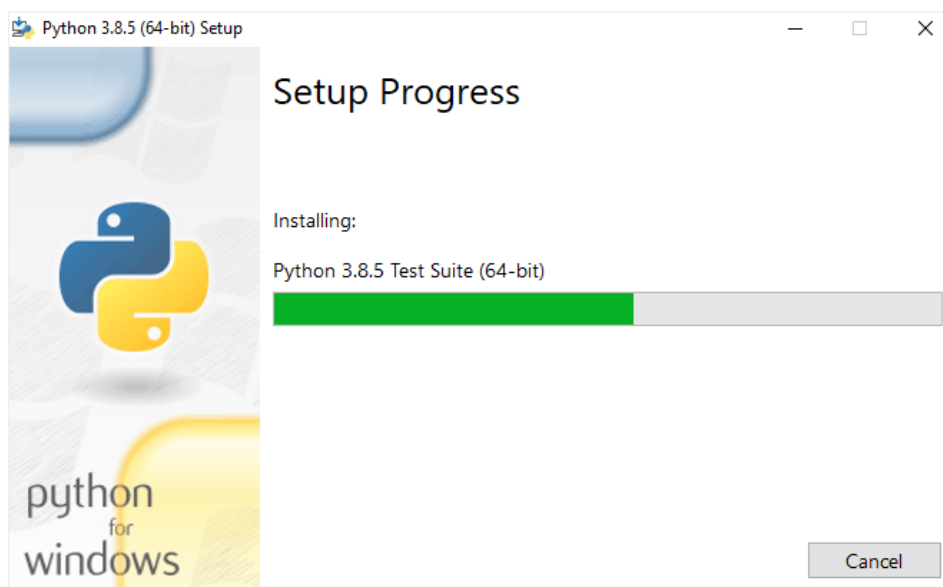
Python'ni windows operatsion tizimida o'rnatishni ko'rib chiqamiz. Buning uchun python'ning [rasmiy sahifasidan](#) uni yuklab olinadi. Yuklangan faylni ustiga sichqonchanning chap tugmasini bosish orqali uni o'rnatishni boshlashimiz mumkin. Bu jarayonda biz Add Python 3.x to PATH sozlamasini bosib ketishimiz zarur. Bundan keyin esa Install Now tugmasini bosamiz:



1.1.1-rasm.

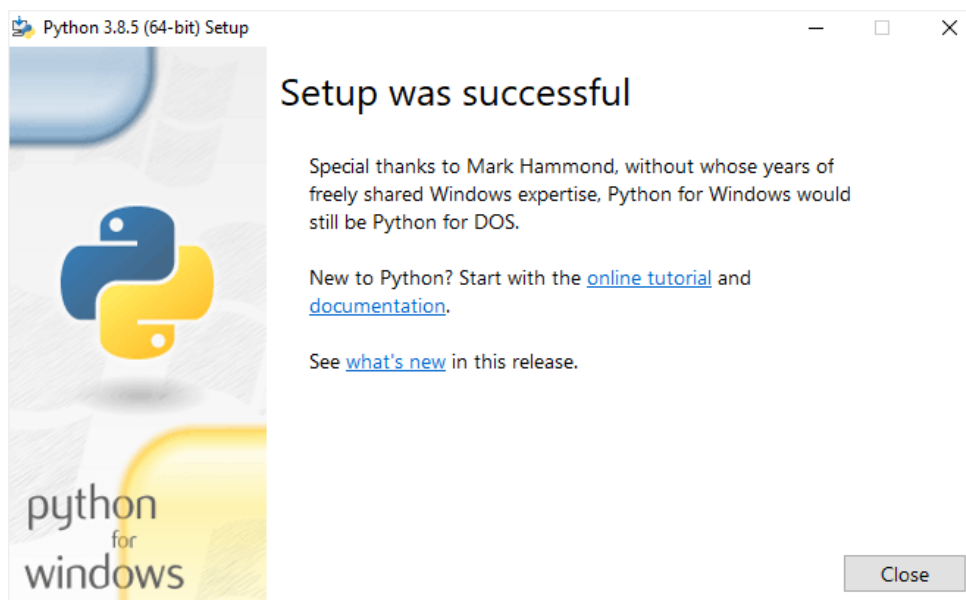
O'rnatish jarayoni bir necha daqiqa vaqt oladi:





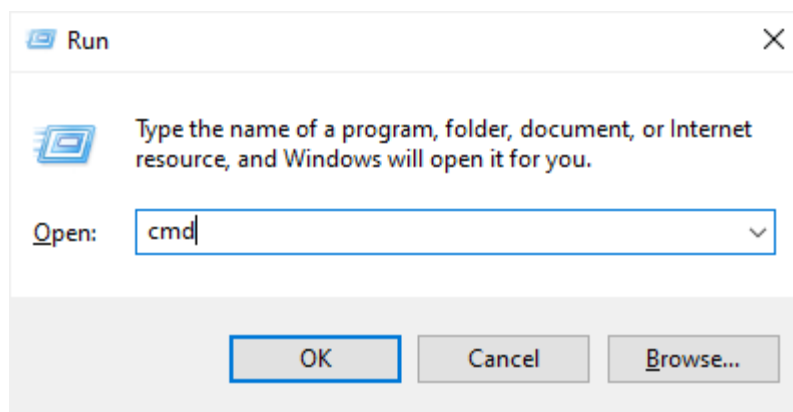
1.1.2-rasm.

O‘rnatish muvaffaqiyatli yakunlanganda quyidagicha oyna hosil bo‘ladi:



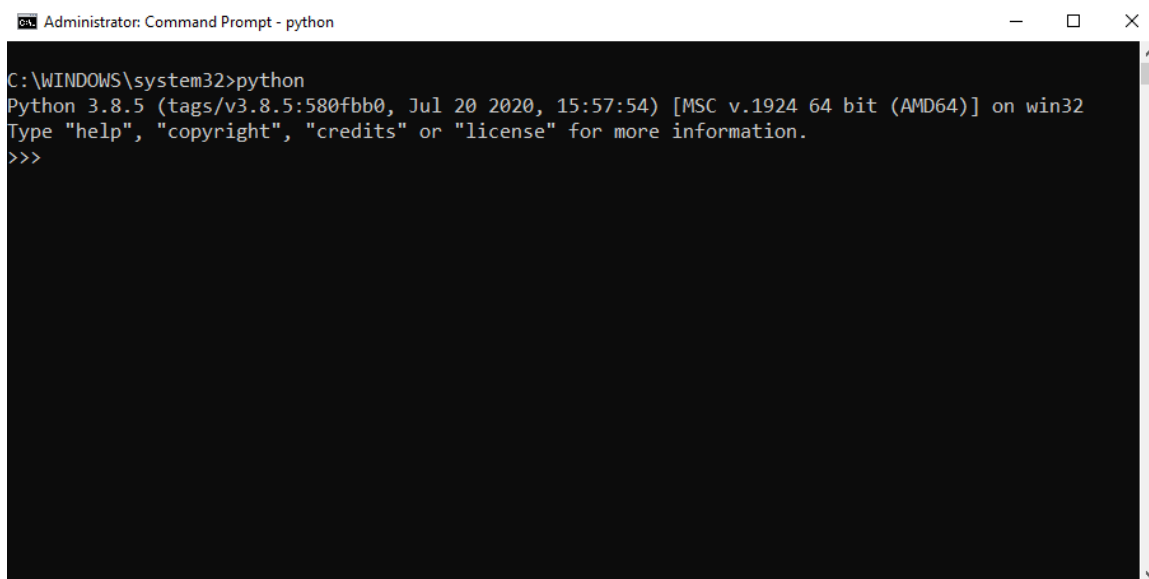
1.1.3-rasm.

Python'ning kompyuterizga o‘rnatilganini quyidagi amallar orqali tekshirib ko‘rishimiz mumkin. 1) *win + R* tugmalarini birgalikda bosish orqali *Run Window* oynasini ochib *cmd* deb yozgan holda *Enter* tugmasini bosamiz:



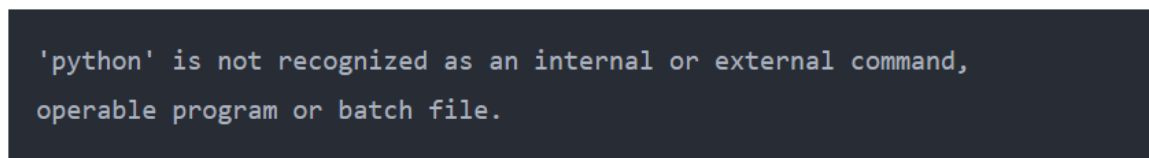
1.1.4-rasm.

Bunda *Command Prompt* dasturi ochiladi. Bu dasturda python buyrug‘ini beramiz:



1.1.5-rasm

Agar shunday natijani ko‘radigan bo‘lsak bu python muvaffaqiyatli o‘rnatilganini bildiradi. Aks holda quyidagicha natijani ko‘rishimiz mumkin bo‘ladi:



1.1.6-rasm.

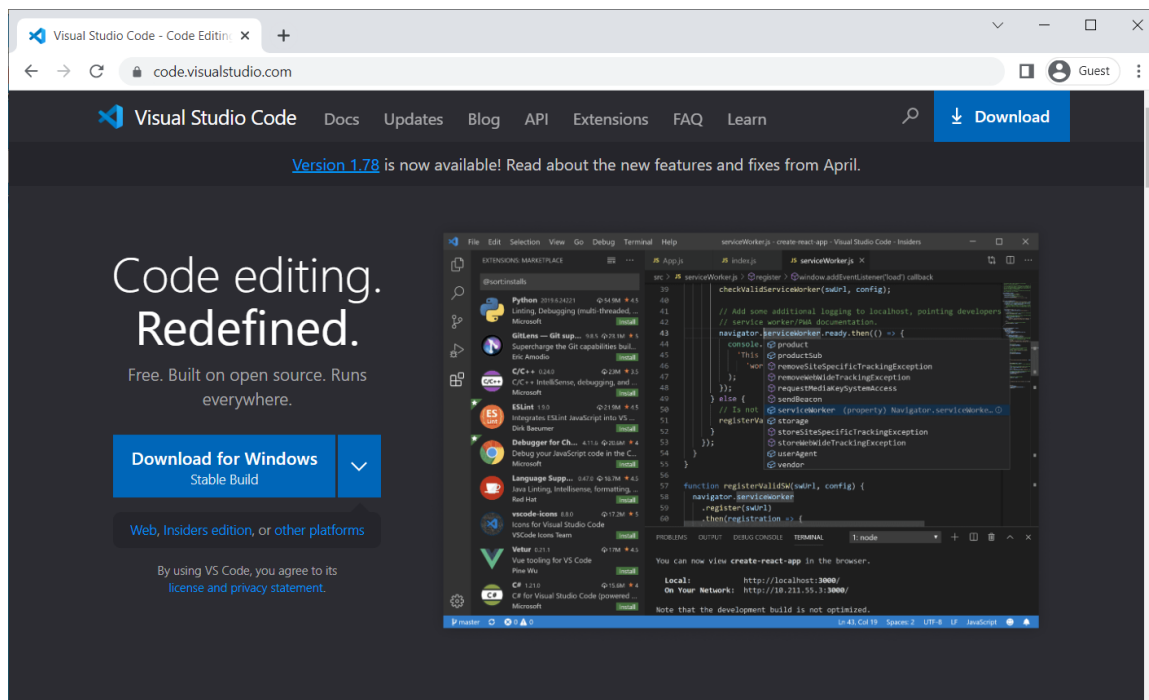
Python dasturi ko'plab linux distrolarida (Ubuntu, Linux Mint va h.k.) va Mac operation tizimida o'rnatilgan bo'ladi. Buni terminal dasturiga *python3* komandasini berish orqali tekshirib ko'rish mumkin.

## Dasturlar yozish uchun dasturlash muhitini sozlash

Python dasturlash tilida dasturlar tuzishda bizda matn muharrirlari kerak bu bo'ladi. Bu bizga dastur kodlarini yozishimizda bir qancha qulayliklar beradi. Hozirgi kunda bir necha zamonaviy kod tahrirlovchi dasturlar mavjud. Shular orasidan Visual Studio Code dasturini o'rnatish va uni python dasturlash tili uchun sozlashni ko'rib chiqamiz.

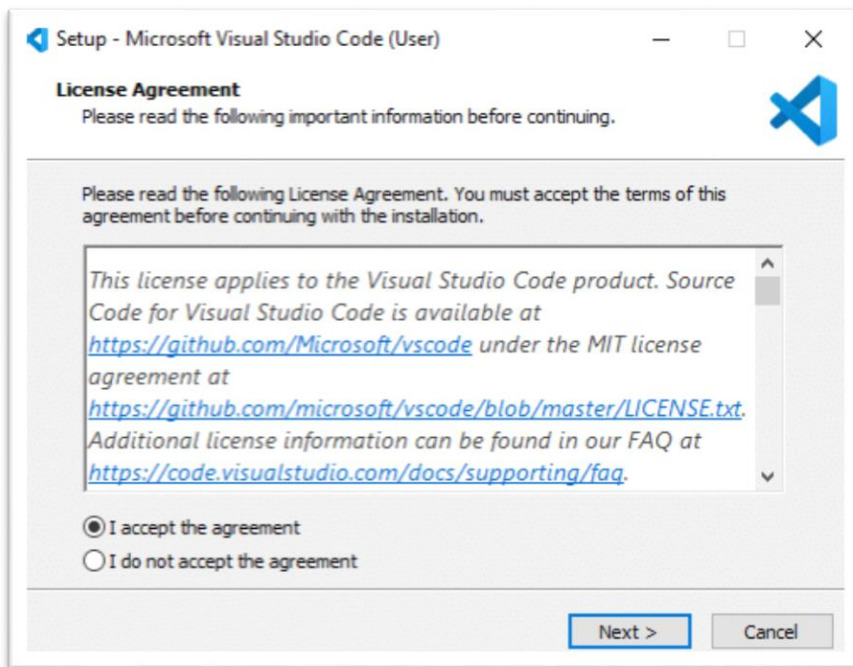
Visual Studio Code (VS Code) — bu kuchli va keng tarqalgan kod tahrirlovchisi, shuningdek, Python dasturlash tilini o'rganish va ishlash uchun katta imkoniyatlar yaratadi. Quyidagi matnda, Windows tizimiga VS Code ni o'rnatish va uni Python dasturlash tiliga sozlash haqida ma'lumotlar berilgan:

Visual Studio Code'ning rasmiy veb-saytidan ([code.visualstudio.com](https://code.visualstudio.com)) orqali Windows uchun "Download"ni bosing.

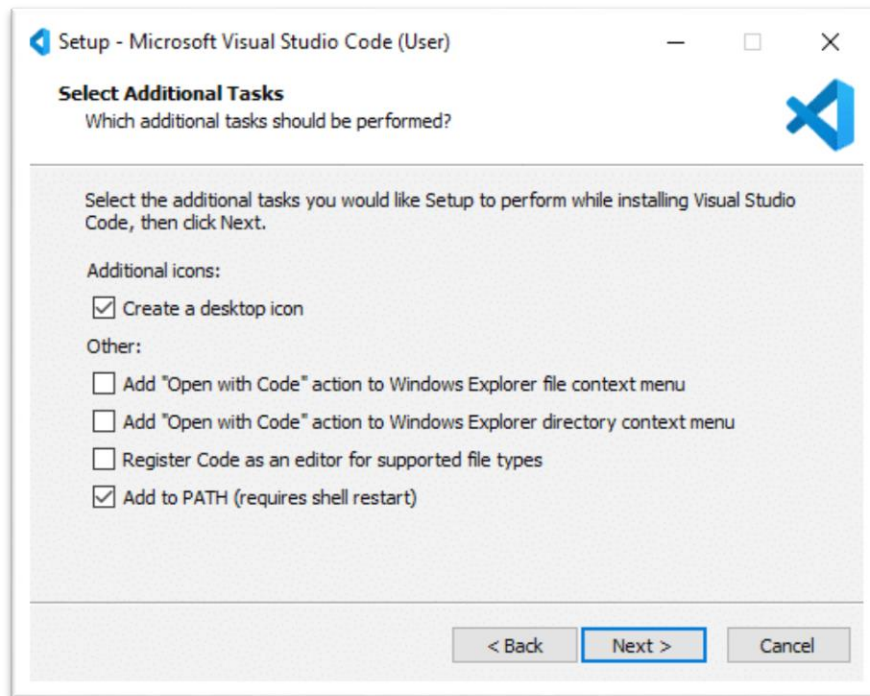


### 1.1.7-rasm.

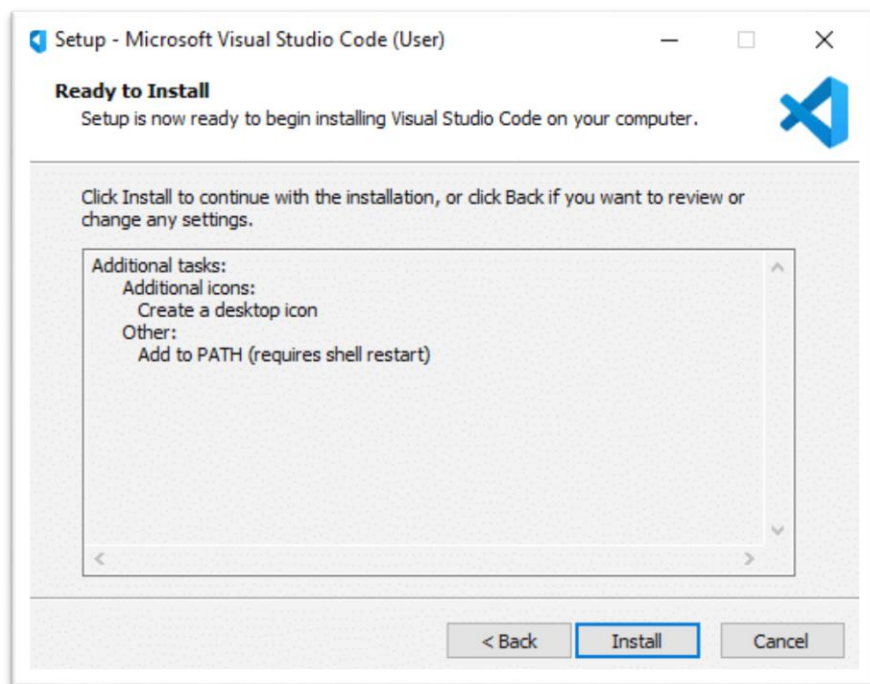
Dastur fayli yuklab olingan uning ustiga bosib, ko'rsatmalar bo'yicha boshqichma bosqich dastur o'rnatiladi.



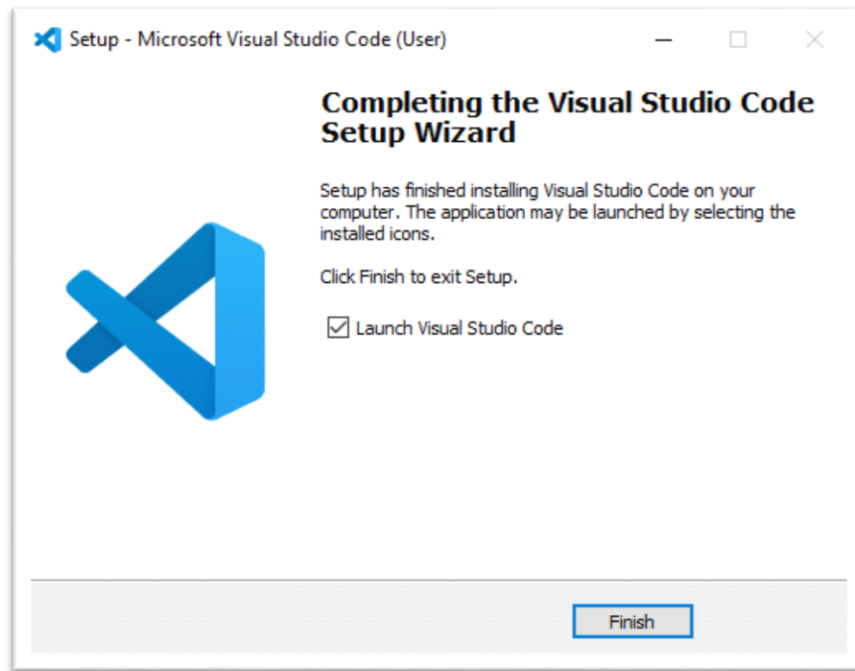
### 1.1.8-rasm.



1.1.9-rasm.

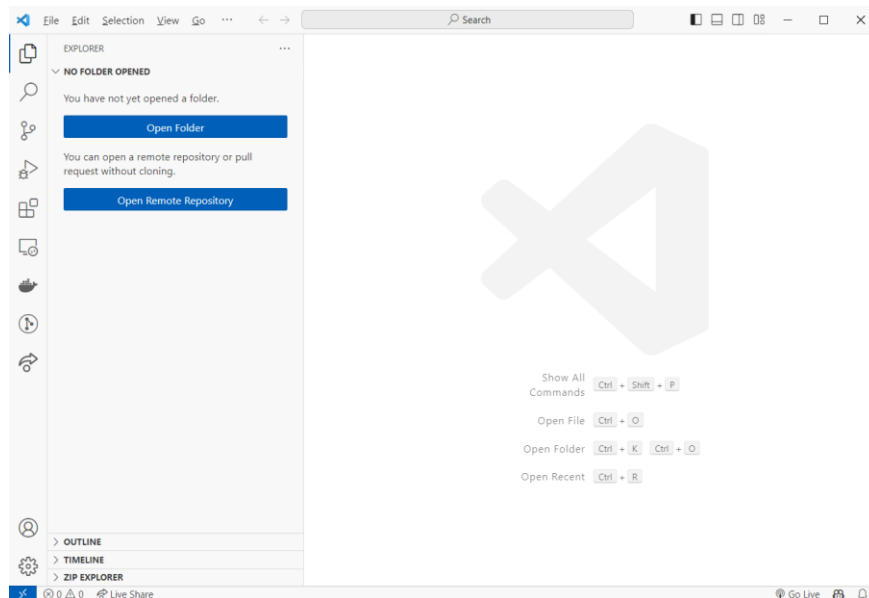


1.1.10-rasm.



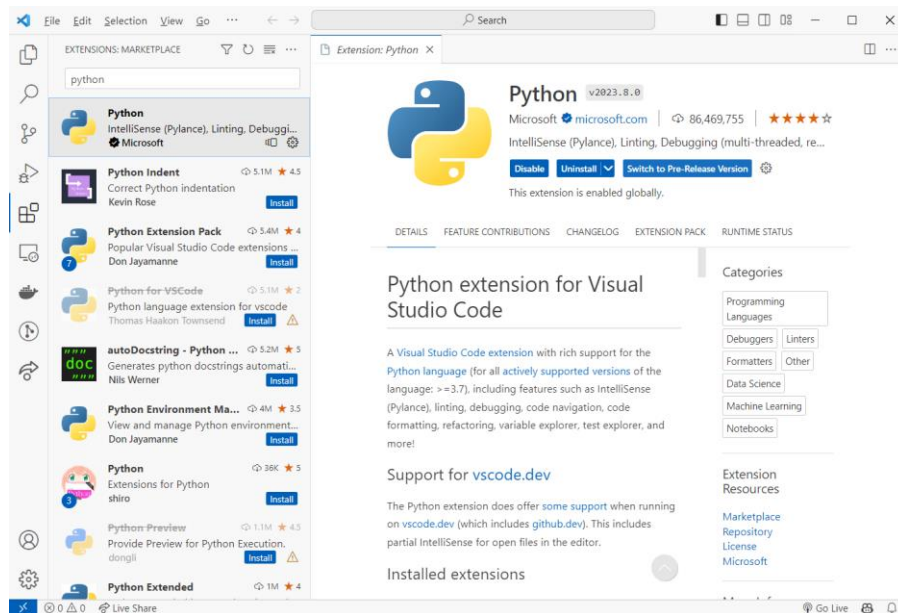
1.1.11-rasm.

Dasturni oʻrnatishning oxirgi bosqichida yuqoridagidek oyna hosil boʻladi (1.1.11-rasm). Finish tugmasini bosish orqali dasturni ishga tushirishimiz mumkin:



1.1.12-rasm.

Visual Studio Code (VS Code) dan foydalanib python dasturlash tilida dasturlar tuzishimizda qulayliklar bo'lishi uchun VS Code dasturiga Python dasturlash tilining qo'shimcha paketini o'rnatib olamiz.

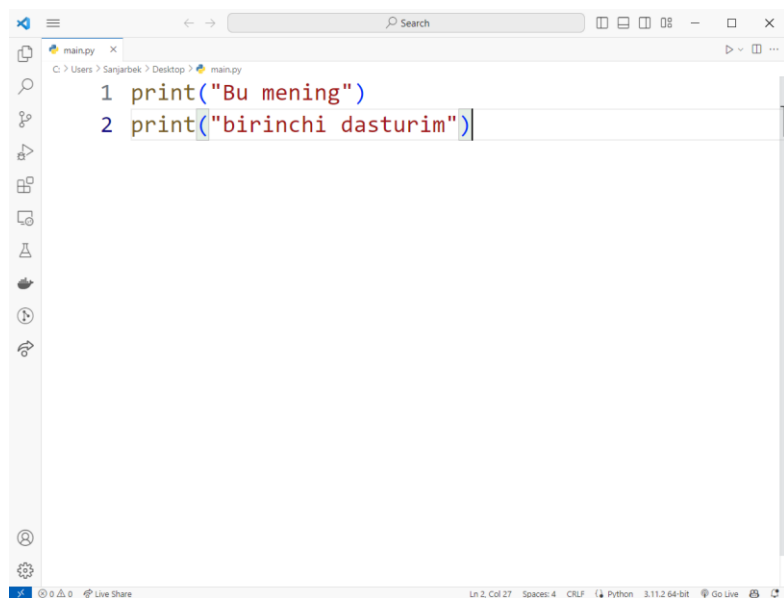


1.1.13-rasm.

Endi biz uchun barchasi tayyor. Birinchi dasturimizni yozib ko'rishimiz mumkin.

### **Python dasturlash tilida birinchi dasturni tuzish**

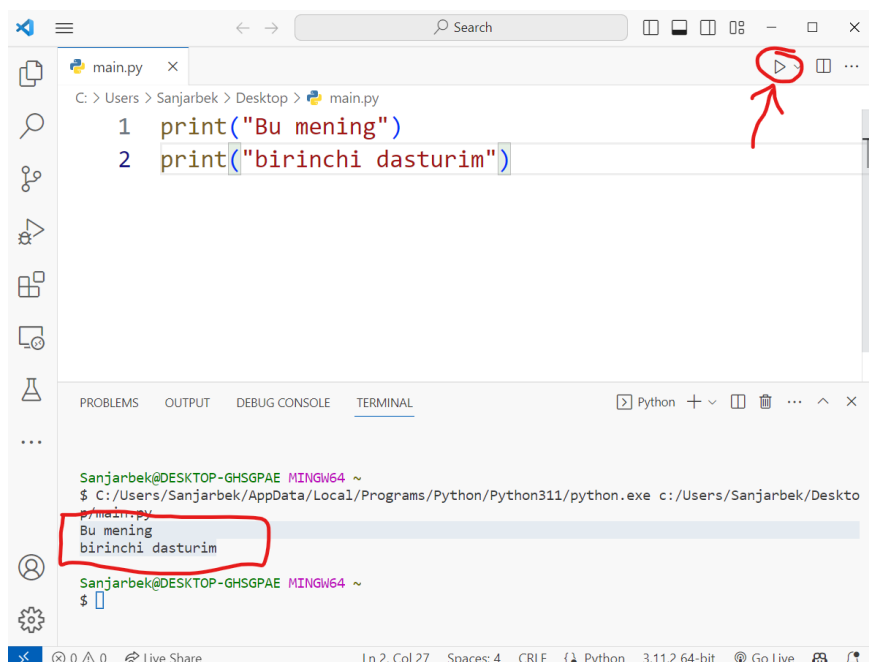
Buning uchun VS Code dasturiga kirib yangi fayl ochib olamiz. Buning uchun CTRL + N qisqa buyruqdan foydalanishimiz mumkin. CTRL + S buyrug'i orqali esa fayl manzilini ko'rsatgan holda *main.py* nomi bilan saqlaymiz. Bu nomni ixtiyoriy berishimiz mumkin.



1.1.14-rasm.

Birinchi dasturimizda `print` nomli funksiyadan foydalandik. `print()` funksiyasi, Python dasturlash tilida matnni konsolga chiqarish uchun ishlatiladi. U konsolda matn, qiymatlar, o'zgaruvchilar yoki boshqa ma'lumotlarni chiqarishni ta'minlaydi.

Dasturni yurgazish uchun quyidagi rasmda ko'rsatilgan Run Python File tugmasini bosamiz, dastur natijasi pastki qismda ko'rinadi.





### 1.1.15-rasm.

Shu bilan bizning birinchi dasturimiz muvaffaqiyatli bajarildi. Bu dastur konsolda ishlovchi dastur hisoblanadi. Ya'ni uning natijasi konsolda ko'rinadi.

## 2. Ma'lumotlar turlari va o'zgaruvchilar.

Ma'lumotlar turi (data type) va o'zgaruvchilar, dasturlashning asosiy qamrovlari bo'lib, Python va boshqa dasturlash tillarida keng qo'llaniladigan konseptlardir.

**Ma'lumotlar turi (Data Types):** Ma'lumotlar turi, ma'lumotlarni turiga va ularga amal qilish uchun berilgan xususiyatlarga asoslangan bo'lgan turi ifodalaydi. Python tilida quyidagi ma'lumotlar turini ko'ramiz:

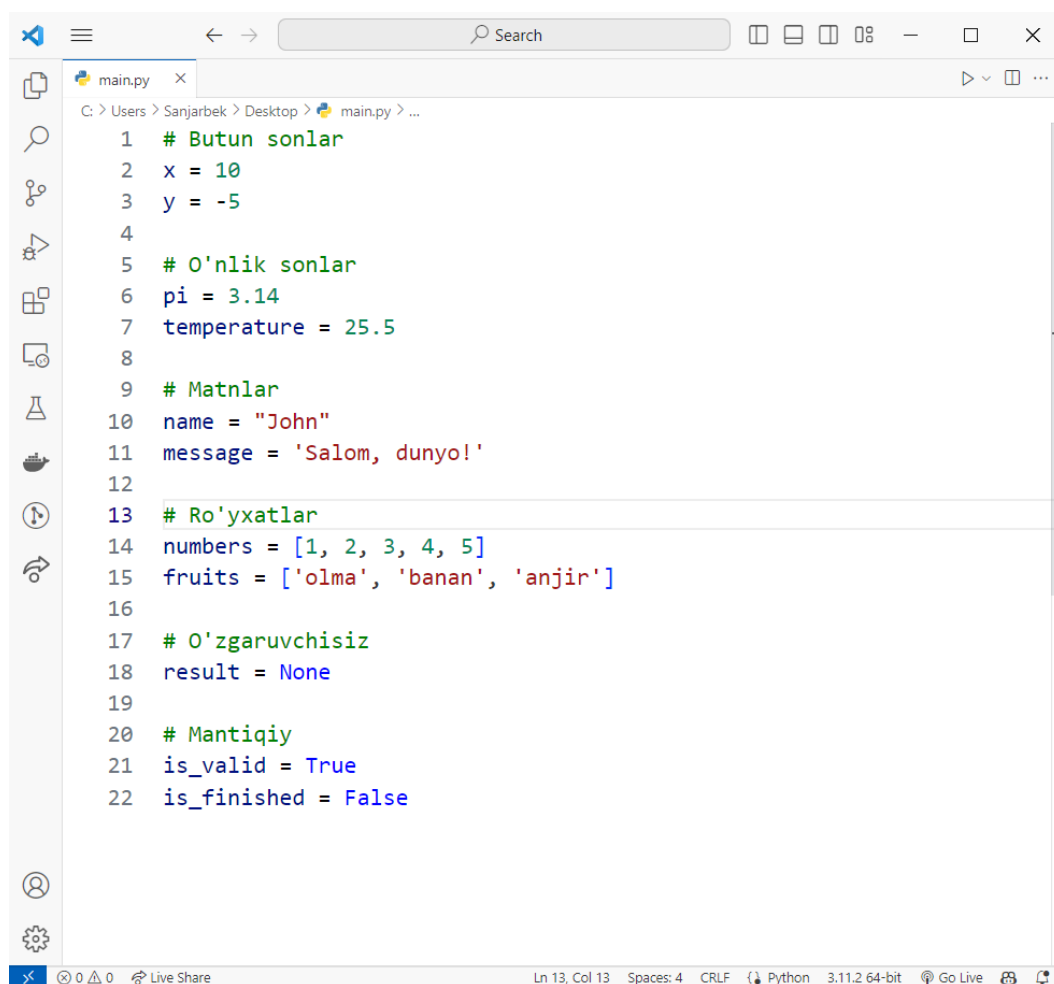
- Butun sonlar (Integer): Butun sonlarni ifodalaydi, masalan, 1, 2, -5, 100.
- O'nlik sonlar (Float): O'nlik sanoq sistemadagi sonlarni ifodalaydi, masalan, 3.14, -0.5, 2.0.
- Matnlar (String): Matnlar yoki yozuvlarni ifodalaydi, masalan, "Salom", 'Dunyo', "123".
- Ro'yxatlar (List): Elementlardan iborat bo'lgan tartiblangan ma'lumotlar to'plamini ifodalaydi.
- O'zgaruvchisiz (None): Hech qanday qiymat olmaydigan holatni ifodalaydi.
- Mantiqiy (Boolean): Faqat ikkita qiymatni o'z ichiga oladigan True yoki False qiymatlarini ifodalaydi.

**O'zgaruvchilar (Variables):** O'zgaruvchilar, ma'lumotlarni saqlab turib, ular bilan amallar bajarish uchun ishlatiladigan nomlangan identifikatorlar hisoblanadi. O'zgaruvchilar yordamida ma'lumotlarni saqlash, ularga murojaat qilish va ulardan foydalanish mumkin bo'ladi. Python tilida o'zgaruvchilar quyidagi shartlarga javob beradi:

- O'zgaruvchilar harf yoki \_ (pastki chiziqli chiziq) bilan boshlanishi kerak.

- O'zgaruvchilar harf, raqam va \_ dan iborat bo'lishi mumkin, lekin raqam bilan boshlanmasligi kerak.
- O'zgaruvchi nomi katta-kichik harf kiritsa ham farq qilmaydi (mavjud o'zgaruvchi bilan bir xil deb hisoblanadi).
- Pythonning ichki o'zgaruvchilari bilan bir xil nomni ishlatmaslik tavsiya etiladi.

Quyidagi misol kod o'zgaruvchilar haqida ko'proq tushuntiradi:



```

1  # Butun sonlar
2  x = 10
3  y = -5
4
5  # O'nlik sonlar
6  pi = 3.14
7  temperature = 25.5
8
9  # Matnlar
10 name = "John"
11 message = 'Salom, dunyo!'
12
13 # Ro'yxatlar
14 numbers = [1, 2, 3, 4, 5]
15 fruits = ['olma', 'banan', 'anjir']
16
17 # O'zgaruvchisiz
18 result = None
19
20 # Mantiqiy
21 is_valid = True
22 is_finished = False
  
```

1.2.1-rasm.

### 3. Operatorlar

Python tilida turli turlarda operatorlar mavjud, ular bilan ma'lumotlar ustida amallar bajarish mumkin. Quyidagi operatorlar Python dasturlash tili uchun mavjud:

1. **Arifmetik operatorlar:** Arifmetik operatorlar sonlar ustida amallar bajarish uchun ishlatiladi.

`+`: Yig'ish ( $a + b$ ).

`-`: Ayirish ( $a - b$ ).

`\*`: Ko'paytirish ( $a * b$ ).

`/`: Bo'lish ( $a / b$ ).

`//`: Butun bo'lish ( $a // b$ ).

`%`: Qoldiqni topish ( $a \% b$ ).

`\*\*`: Darajaga oshirish ( $a ** b$ ).

2. **Tenglik operatorlari:** Tenglik operatorlari ikki qiymatni solishtirish uchun ishlatiladi.

`==`: Tengmi? ( $a == b$ ).

`!=`: Teng emasmi? ( $a != b$ ).

3. **Taqqoslash operatorlari:** Taqqoslash operatorlari qiymatlarni solishtirish uchun ishlatiladi.

`>`: Katta ( $a > b$ ).

`<`: Kichik ( $a < b$ ).

`>=`: Katta yoki teng ( $a >= b$ ).

`<=`: Kichik yoki teng ( $a <= b$ ).

4. **Mantiqiy operatorlar:** Mantikiy operatorlar `True` va `False` qiymatlari bilan ishlaydigan mantiqiy ifodalarni boshqarish uchun ishlatiladi.

`and`: Va sharti ( $a \text{ and } b$ ).

`or`: Yoki sharti (a or b).

`not`: Aks sharti (not a).

**5. Bitwise operatorlar:** Bitwise operatorlar ikki sonning bitlariga amal bajarish uchun ishlatiladi.

`&`: Bitwise va (a & b).

`|`: Bitwise yoki (a | b).

`^`: Bitwise xor (a ^ b).

`~`: Bitwise negatsiya (~a).

`<<`: Bitlarni chapga surish (a << b).

`>>`: Bitlarni o'ngga surish (a >> b).

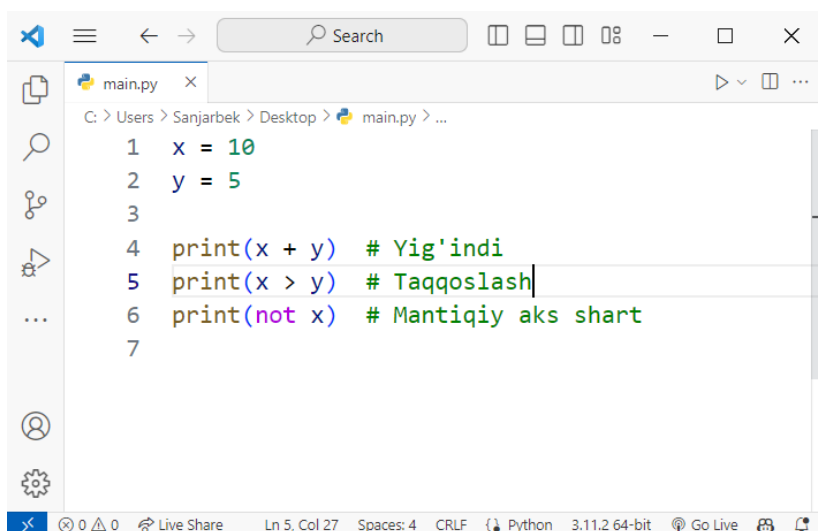
**6. Tartiblash operatorlari:** Tartiblash operatorlari ro'yxatlarda ishlatiladi.

`in`: Element ro'yxatda mavjudmi? (a in b).

`not in`: Element ro'yxatda mavjud emasmi? (a not in b).

Bu operatorlar Python tilidagi asosiy operatorlardan faqat bir qismidir. Boshqalar ham mavjud bo'lishi mumkin, masalan, o'zgaruvchi operatorlari, atribut operatorlari, funktsiya chaqirish operatorlari va hokazo.

Misol uchun:



1.2.2-rasm.

#### 4. Shartli ifodalar (Conditional Statements)

Shartli ifodalar (Conditional Statements) dasturlashda bir amalni bajarishga qaror qabul qilish uchun ishlatiladi. Shartli ifodalar orqali dastur biron bir shartni tekshirib, unga mos bo'lganda bir qator amalni bajaradi.

1. **if ifodasi:** **if** ifodasi, shartni tekshiradi va agar shart to'g'ri bo'lsa, ularni quyidagi bo'limda ko'rsatilgan amallarni bajaradi.

Sintaksis:

```
if shart:
    amal_1
    amal_2
    ...
```

2. **if-else ifodasi:** **if-else** ifodasi, shartni tekshiradi va shart to'g'ri bo'lsa bir bo'limdagi amallarni bajaradi, aks holda esa boshqa bir bo'limdagi amallarni bajaradi.

Sintaksis:

```
if shart:
```

```
    amal_1
    amal_2
    ...
else:
    amal_3
    amal_4
    ...
```

3. **if-elif-else ifodasi:** **if-elif-else** ifodasi, bir nechta shartlarni tekshiradi. Agar bir shart bajarilsa, unga mos bo‘lgan bo‘limdagi amallarni bajaradi. Aks holda, keyingi shartlarni tekshiradi. Agar hech qanday shart bajarilmasa, else bo‘limdagi amallarni bajaradi.

Sintaksis:

```
if shart_1:
    amal_1
    amal_2
    ...
elif shart_2:
    amal_3
    amal_4
    ...
elif shart_3:
    amal_5
    amal_6
    ...
else:
    amal_7
    amal_8
    ...
```

Quyidagi misollar yordamida shartli ifodalar haqida ko‘proq tushunish mumkin:

```
# if ifodasi
x = 5
if x > 0:
    print("x musbat son")
# if-else ifodasi
x = -3
if x > 0:
    print("x musbat son")
else:
    print("x manfiy son")
# if-elif-else ifodasi
x = 0
if x > 0:
    print("x musbat son")
elif x < 0:
    print("x manfiy son")
else:
    print("x nolga teng")
```

Dastur natijasi:

```
x musbat son
x manfiy son
x nolga teng
```

## 5. Funksiyalar

Funksiyalar, qayta-qayta ishlatiladigan va bajarilishi kerak bo‘lgan biror amalni birlashtiruvchi va bajaruvchi bo‘lmagan dastur qatorlari to‘plamidir. Funksiyalar

dasturlashda kodni yorliqlashtirish va o‘zaro qayta-qayta foydalanishni ta’minlash maqsadida ishlatiladi.

Python tilida funksiyalar quyidagi xususiyatlarga ega:

1. **Funksiya yaratish:** Funksiyani yaratish uchun **def** kalit so‘zi va funksiya nomi ishlatiladi. Funksiyaga kerak bo‘lgan ma’lumotlar (argumentlar) berilishi mumkin.

Sintaksis:

```
def funksiya_nomi(argumentlar):  
    # Amallar  
    return natija
```

2. **Funksiya skopi:** Funksiya ichidagi o‘zgaruvchilar funksiya skopida aniqlanadi. Ular lokal o‘zgaruvchilar deb ataladi va faqat funksiya ichida yaradilgan bo‘lib, boshqa joylarda aniqlanmagan.
3. **Tuldiruvchi funksiyalar:** Funksiyalar bajaradigan amallarni to‘plab turadi va return kalit so‘zi yordamida funksiya natijasini qaytaradi.
4. **Lambda funksiyalar:** Ular bir marta ishlatiladigan oddiy funksiyalar bo‘lib, asosan boshqa funksiyalar yoki qayta-qayta ishlatilmaydigan amallar uchun ishlatiladi.
5. **Funksiyalar argument sifatida:** Funksiyalar argument sifatida o‘zgaruvchilarni qabul qilishi mumkin. Bu, funksiyaning bir funksiya argumenti sifatida uzatish va uni boshqa funksiya ichida ishlatish imkonini beradi.
6. **Rekursiya:** Rekursiya, funksiya o‘zini chaqirishi orqali o‘zini bajarish imkonini beradi. Bu, masalan, faktorialni hisoblash uchun foydalaniladi.

Quyidagi misollar yordamida funksiyalar va ularning xususiyatlari haqida ko‘proq tushunish mumkin:

```
# Funksiya yaratish  
def salom_ber():
```



```

    print("Salom, dunyo!")
# Funksiya chaqirish
salom_ber()
# Funksiya argumentlari
def salom(ism):
    print("Salom, " + ism + "!")
salom("Ali")
# Funksiya qaytarish
def yigindi(a, b):
    return a + b
natija = yigindi(5, 3)
print(natija)
# Lambdalar
kvadrat = lambda x: x**2
print(kvadrat(5))
# Funksiyalar argument sifatida
def hisobla(a, b, funksiya):
    return funksiya(a, b)
natija = hisobla(4, 2, yigindi)
print(natija)
# Rekursiya
def faktorial(n):
    if n == 0:
        return 1
    else:
        return n * faktorial(n - 1)
print(faktorial(5))
# Builtin string funksiyalari
matn = "Python dasturlash tili"
print(len(matn))
print(matn.upper())

```

```
print(matn.lower())  
print(matn.split())
```

Natijalar:

```
Salom, dunyo!  
Salom, Ali!  
8  
25  
6  
120  
20  
['Python', 'dasturlash', 'tili']
```

## 6. Sikllar

Sikllar, biror amalni bir necha marta takrorlash uchun ishlatiladigan dastur qatorlari to‘plamidir. Sikllar dasturlashda kodni qisqartirish, qayta-qayta ishlatish va ko‘p elementlarni boshqarish uchun qulay imkoniyatlar yaratadi. Python tilida ikki turdagi sikllar mavjud:

1. **for sikli:** **for** sikli, ro‘yxatdagi har bir element uchun biror amalni takrorlaydi. Uchragan elementlar soni bilan chegaralanadi.

Sintaksis:

```
for element in ro‘yxat:  
    # Amallar
```

2. **while sikli:** **while** sikli, berilgan shart to'g'ri bo'lgan vaqtgacha biror amalni takrorlaydi. Shart to'g'ri bo'lishi uchun tekshiriladi.

Sintaksis:

```
while shart:  
    # Amallar
```

Quyidagi misollar yordamida sikllar haqida ko'proq tushunish mumkin:

```
# for sikli  
mehmonlar = ["Ali", "Vali", "Hasan", "Husan"]  
for mehmon in mehmonlar:  
    print("Salom, " + mehmon + "!")  
  
# Nested for sikllari  
for son in range(1, 4):  
    for kop in range(1, 4):  
        print(son * kop)  
  
# while sikli  
son = 1  
while son <= 5:  
    print(son)  
    son += 1
```

Natijalar:

```
Salom, Ali!  
Salom, Vali!  
Salom, Hasan!  
Salom, Husan!
```

1

2

3

2

4

6

3

6

9

1

2

3

4

5

## 2-BOB. Ma'lumotlar ba'zalari. Web dasturlashga kirish

### 1. Ma'lumotlar bazalari (Databases)

Ma'lumotlar bazalari (databases) dasturlashda ma'lumotlarni saqlash va ularga kirish/chiqishni tashkil etish uchun ishlatiladigan tuzilmalardir. Ular ma'lumotlarni tashkil etish, saqlash, o'zgartirish va o'qish imkoniyatlarini beradi.

**SQL sintaksi.** SQL (Structured Query Language) - bu ma'lumotlar bazalari bilan ishlash uchun mo'ljallangan so'rovlarni yozish uchun ishlatiladigan standart boshqa dasturlash tili. SQL sintaksisi yordamida ma'lumotlarni yaratish, o'qish, yozish, yangilash, o'chirish va so'rovlar bilan ma'lumotlarga amal qilish mumkin.

**SQLite ma'lumotlar bazasi.** SQLite, yoki SQLite3, yozuvlar bazasi sifatida ishlatiluvchi xavfsiz, tarmoqli va server tomonidan boshqarilmaydigan ma'lumotlar bazasidir. SQLite ma'lumotlar bazasi fayl shaklida saqlanadi va Python tilida keng ishlatiladi.

**Python SQLite ma'lumotlar bazasi kutubxonasi.** Python tilida SQLite ma'lumotlar bazasiga kirish va ma'lumotlar bilan ishlash uchun `sqlite3` kutubxonasidan foydalaniladi. Ushbu kutubxona Python orqali SQLite ma'lumotlar bazasiga so'rovlar yuborish, ma'lumotlar bazasini yaratish, tablitsalarni yaratish va o'zgartirish, ma'lumotlarga murojaat qilish va boshqalar kabi vazifalarni bajarish imkonini beradi.

Quyidagi misollar yordamida SQL sintaksisi, SQLite ma'lumotlar bazasi va *sqlite3* kutubxonasini qanday ishlatishni ko'rish mumkin:

```
# SQL sintaksisi
# CREATE TABLE
CREATE TABLE users (
    id INTEGER PRIMARY KEY,
    name TEXT,
    email TEXT
);
```

```

# INSERT INTO
INSERT INTO users (name, email) VALUES ('John', 'john@example.com');

# SELECT
SELECT * FROM users;

# UPDATE
UPDATE users SET email='john.doe@example.com' WHERE id=1;

# DELETE
DELETE FROM users WHERE id=1;

# SQLite ma'lumotlar bazasi va sqlite3 kutubxonasidan foydalanish
import sqlite3

# Ma'lumotlar bazasiga bog'lanish
conn = sqlite3.connect('database.db')

# Cursor yaratish
cursor = conn.cursor()

# Ma'lumotlar bazasini yaratish
cursor.execute("CREATE TABLE users (id INTEGER PRIMARY KEY, name TEXT, email TEXT)")

# Ma'lumotlarni qo'shish
cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ('John', 'john@example.com'))

# Ma'lumotlarni olish
cursor.execute("SELECT * FROM users")

```

```
result = cursor.fetchall()
for row in result:
    print(row)

# Ma'lumotlarni yangilash
cursor.execute("UPDATE users SET email=? WHERE id=?", ('john.doe@example.com',
1))

# Ma'lumotlarni o'chirish
cursor.execute("DELETE FROM users WHERE id=?", (1,))

# Ma'lumotlar bazasini saqlash va bog'lanishni yopish
conn.commit()
conn.close()
```

## PostgreSQL ma'lumotlar bazasi

PostgreSQL, avtonom server tomonidan boshqariladigan biror tashqi ma'lumotlar bazasi tizimi hisoblanadi. Bu ma'lumotlar bazasi tizimi yuqori darajadagi qulaylik va xavfsizlik darajasiga ega bo'lib, moslashtirilgan transaksiya integriteti, boshqaruv, o'zaroqli suhbat va keng tarqalgan funksiyalarga ega.

**PostgreSQL serverni o'rnatish.** PostgreSQL serverni o'rnatish uchun quyidagi qadamlarni bajarishingiz kerak:

- PostgreSQL web saytiga o'ting: <https://www.postgresql.org/>
- Qo'llab-quvvatlashdan foydalanib, o'rnatish uchun mos versiyani tanlang va o'rnatish jarayonini bajarishni boshlang.
- O'rnatish jarayonida talab qilinadigan ma'lumotlarni kiritish va sozlamalarni tanlash imkoniyati mavjud bo'ladi.

- O'rnatilgandan so'ng PostgreSQL server ishga tushadi va siz uni lokal tarmoqda yoki serverda ishlatishingiz mumkin.

**PgAdmin interfeysi.** PostgreSQL ma'lumotlar bazasini boshqarish uchun grafik interfeysga ega bo'lgan bir vosita. U sizga ma'lumotlar bazasini vizual ravishda yaratish, o'zgartirish va o'chirish, so'rovlar yuborish, ma'lumotlar bazasi ob'ektlarini ko'rish va boshqalar kabi imkoniyatlarni beradi. PgAdmin ni o'rnatish jarayoni quyidagicha bo'ladi.

- PostgreSQL web saytiga o'ting: <https://www.pgadmin.org/>
- Qo'llab-quvvatlashdan foydalanib, mos versiyani tanlang va o'rnatish jarayonini bajarishni boshlang.
- O'rnatilgandan so'ng, PgAdmin interfeysi PostgreSQL ma'lumotlar bazasini boshqarish uchun oson va qulay interfeysni taklif etadi.

**Komandalar ustida interfeys.** PostgreSQL server bilan to'lovchilar orqali komandalar ustida ishlash imkonini beradi. Bunda siz ma'lumotlar bazasiga terminal yoki komandalar satriga so'rovlar yuborish, so'rov natijalarini ko'rish, ma'lumotlarni yaratish, o'zgartirish va boshqarish imkoniyatiga ega bo'lasiz.

PostgreSQL komandalar ustida ishlash uchun terminal yoki komandalar satrini ochib, so'ng psql ni yoki PostgreSQL binariga joylashgan boshqa komandalar ustida ishlashingiz mumkin.

## **2. Veb dasturlashga kirish**

Veb dasturlash, internetda veb saytlar va ilovalar yaratish uchun dasturlash tajribasini qo'llaydigan soha hisoblanadi. Bu sohada faoliyat ko'rsatayotgan dasturchilar veb ilovalarini ishlab chiqish, foydalanuvchilarga interaktivlik taqdim etish, ma'lumotlarni saqlash va taqdim etish, veb-saytlarni to'plam va turli tizimlar bilan bog'lash, ma'lumotlarni to'g'ridan-to'g'ri ko'rish, sanash va boshqalar kabi vazifalarni bajarishadi.



**Veb nima?** Veb, internet orqali dunyo bo'ylab tashqi ma'lumotlarga kirish, ularga bog'lanish va ulardan foydalanish uchun bir vosita hisoblanadi. Bu tarzda, veb bir tashqi ma'lumotlar tarmog'i bo'lib, foydalanuvchilar bilan interaktivlik, ma'lumot almashish, tashqi resurslarga bog'lanish, ma'lumotlar almashish va boshqalarni amalga oshirish imkoniyatiga ega.

**Server tomoni va mijoz tomoni (Front-End va Back-End).** Veb dasturlashda, server tomoni va mijoz tomoni mavjud. Server tomonida server dasturlari ishlaydi va tashqi ma'lumotlar bilan ishlashdan javob beradi. Mijoz tomonida esa foydalanuvchi interfeysi, buning o'rniga brauzerda web-saytlar va ilovalar yoki mobil ilovalarda mobil interfeyslar yaratiladi.

**HTML.** HyperText Markup Language (HTML), veb-saytlarni yaratish uchun ishlatiladigan markaziy til hisoblanadi. HTML orqali ma'lumotlar strukturasi aniqlab berish, ma'lumotlarni tashqi ko'rinishda namoyish etish, tugmalar, havolalar, rasmlar, jadvallar va boshqa elementlar bilan foydalanuvchiga interaktivlik taqdim etish mumkin.

**CSS.** Cascading Style Sheets (CSS), HTML elementlarga o'zgaruvchanligini va ko'rinishini belgilash uchun ishlatiladigan til hisoblanadi. CSS orqali veb-saytlarning tashqi ko'rinishi, ranglari, matn xossalari, chizishlar, tartib va boshqalarini belgilash mumkin.

**JavaScript.** JavaScript (JS), veb-saytlarda interaktivlikni amalga oshirish uchun ishlatiladigan dasturlash tili hisoblanadi. Bu skript dasturlash tili orqali foydalanuvchilar bilan interaktivlik o'rnatish, sahifalarda ma'lumotlarni dinamik ravishda o'zgartirish, so'rovlar yuborish, animatsiyalar va effektlar yaratish, formalar va boshqa interfeys elementlarini tekshirish, ma'lumotlar bazasidan ma'lumotlarni yuklash va saqlash kabi vazifalarni bajarish mumkin.

**Backend nima?** Backend, veb ilovalarda server tomonini o'z ichiga olgan qism hisoblanadi. U veb-saytlarda va ilovalarda ma'lumotlar bilan ishlashdan javob beradigan

dasturlar va qo'llanmalar to'plamini o'z ichiga oladi. Backend dasturlari foydalanuvchining so'rovlari va murojaatlari bilan ishlayadi, ma'lumotlar bazasidan ma'lumotlarni o'qish, yozish va o'chirish, autentifikatsiya va avtorizatsiya bilan shug'ullanish, loyihada biznes logikani bajarish, xavfsizlikni ta'minlash, RESTful API'lar orqali xizmat ko'rsatish, tashqi xizmatlar va tizimlar bilan integratsiya qilish, monitoring va qo'shimcha qo'llanmalar bilan boshqarish kabi vazifalarni bajaradi.

Backend dasturlarining yaratilishi uchun ko'p til va texnologiyalar mavjud. Ba'zi ommaviy backend dasturlash tillari Python, JavaScript (Node.js), Ruby, PHP, Java, C# kabi. Ulardan foydalanish kerak bo'lgan til va texnologiya ilovangizning kerakliliklari, sizning tajribangiz, loyihangizning o'lchami va tarqalishi, integratsiya talablari, tizimning o'rnatilishi va boshqalar kabi ko'rsatuvlar asosida tanlanishi mumkin.

**Monolitik arxitektura.** Monolitik arxitektura, ilovani yanaqtirish uchun butun komponentlarni o'z ichiga olgan traditsioniy dasturlash uslubidir. Uning asosiy xususiyatlari:

- Barcha komponentlar birlikda yoki bir bitta bino ostida joylashadi. Bunda backend, frontend, ma'lumotlar bazasi, autentifikatsiya tizimi va boshqa komponentlar bir paketdagi ilova ichida joylashadi.
- Ilova to'liq o'rnatilgan va ishga tushirilgan holatda bo'ladi. Barcha funksiyalar, tizimlar va tizim interfeyslari bir biriga bog'liq bo'lganligi uchun o'zaro bog'lanish va ko'p yoki bitta komponentda o'zgartirishlar kuchayadi.
- Monolitik ilova qurish va yanaqtirish oson bo'lib, tizim boshqaruvini osonlashtiradi. Bundan tashqari, dasturchilar o'z fikr-mulohazalarini bir-biriga tez o'zgartirishlarni osonlashtiradi.

Monolitik arxitekturaning kamchiliklari esa:

- O'zgarishlarni qo'llab-quvvatlash va yangilash qiyin bo'lishi mumkin. Barcha komponentlarning birlikda yanaqtirilishi kerak bo'lgani uchun, bir qismi o'zgartirishlar boshqa qismga ta'sir etishi mumkin.
- Katta o'lchovli ilovalarda tizimga kiritiladigan komponentlar soni katta bo'lishi mumkin va tizimni mustahkamlash va yangilashni qiyinlashtirishi mumkin.
- Scalability (o'sishga kushish) qiyinchiliklarni tug'dirishi mumkin. Monolitik arxitektura uning yuqori miqdordagi tizim so'rovlarini to'g'ri o'lchamoqda qabul qilib, qo'shimcha resurslar orqali o'sishga qodir bo'lmaydi.

Bu sababli, katta o'lchovli ilovalarda, bozor talablari o'zgarishlari va o'sishiga mos keladigan mikroservislar arxitekturasini qo'llash yoki boshqa arxitektura usullarini ko'rib chiqish tavsiya qilinadi. Mikroservislar arxitekturasini komponentlarni kichik xizmatlar sifatida ajratib olishni taqdim etadi, bu esa dasturchilarga qulaylik, avtomatlashtirish, o'zgarishlarni yengil qabul qilish, skalabilnost (o'sishga kushish) va boshqa imkoniyatlar beradi.

**Mikroservislar arxitekturasini.** Mikroservislar arxitekturasini, ilovadagi komponentlarni o'zaro aloqada ishlatishning bir qismini ifodalaydi. Bu, kichik, mustaqil va keng tarqalgan xizmatlar ko'p to'plamlarini yaratish, boshqarish va ishlatish imkonini beradi. Mikroservislar orqali ilovadagi bir qismini o'zgartirish yoki yangilash, ko'p foydalanuvchilarning murojaatlari bilan muomalaga kelishish va tizimni kengaytirish hamda ta'mirlash imkoniyatiga ega bo'lasiz.

### 3. Pythonning Veb Frameworklari

Python dasturlash tilida bir nechta mashhur web frameworklar mavjud, ulardan ba'zilari quyidagilardir:

1. **Django.** Django, Pythonning eng mashhur va kuchli web frameworklaridan biridir. Uning afzalliklari:

- To'liq funktsional bo'lgan va yuqori darajadagi kuchli ORM (Object-Relational Mapping) tizimi orqali ma'lumotlar bazasi bilan ishlash imkonini beradi.
- Tizimni yaratish uchun ko'p modullarni o'z ichiga olgan. Autentifikatsiya, ma'lumotlar tashqarida ma'lumotlarni kiritish, tahrirlash va ko'rish, admin paneli va boshqalar kabi tayyor modullarni yaratishni qulaylashtiradi.
- Tizimni o'rganish uchun katta va keng asosiy dokumentatsiya va qo'llanma mavjud.

Kamchiliklari:

- Django tizimi hamda tayyor modullar tizimni yuklab olish va o'rnatish jarayonida o'zaro bog'liqli bo'lishi mumkin.
- Tizimning o'lchami va muhim funksiyalari uchun qo'shimcha vaqt talab qilishi mumkin.

2. **Flask.** Flask, Pythonning kichik va oson ishlovchi web frameworklaridan biri hisoblanadi. Uning afzalliklari:

- Minimalistik tuzilishga ega, ammo kuchli va samarali.
- Yagona fayl yoki modullar orqali tizimni yaratish imkonini beradi.
- Keng imkoniyatlarga ega bo'lgan ekosistema bilan integratsiya qilish imkonini beradi.
- Yuqori darajadagi HTTP protokolini qo'llab-quvvatlaydi.

Kamchiliklari:

- To'liq funktsional tizimlarda o'rniga, qo'shimcha modullar va kutubxonalardan foydalanish talab qiladi.
- Keng imkoniyatlar oldindan kelgan muhandislik vazifalarni o'z ichiga olmaganligi sababli, katta va murakkab tizimlarni yaratishda qiyinliklarga uchrashadi.

### 3. **Pyramid.** Pyramid, Pythonning boshqaruvchi web frameworklaridan biri hisoblanadi.

Uning afzalliklari:

- Arzon, oddiy va samarali.
- Boshqaruvchi tizimlar va RESTful API'lar yaratishga yo'naltirilgan.
- Pluggable arxitekturasini orqali modullar yig'ish imkonini beradi.
- Keng tarqalgan dokumentatsiyaga ega.

Kamchiliklari:

- Kichik ekinchi ekosistema haqida qaror qabul qilganligi sababli, katta va murakkab tizimlarni yaratishda ba'zi imkoniyatlar qiyinliklarga uchray oladi.

Ushbu frameworklar har birining o'ziga xos afzalliklari va kamchiliklari mavjud. Boshqa Python web frameworklaridan bir necha misollar esa Bottle, Tornado, CherryPy, Falcon va boshqalarini o'z ichiga oladi. Frameworkni tanlashda loyihaning turi, tizimning muhim funksiyalari, imkoniyatlari, tajribangiz va tanlaganliklaringiz muhim o'rin tutadi.

## 4. **Django Web Framework**

Django, Python tilida yaratilgan yetakchi veb frameworkidir. U veb ilovalarni tezkor, ishonchli va qulay shaklda yaratish imkonini beradi. Django turli turdagi veb ilovalar yaratish uchun tayyor bo'lgan kuchli va hamkorlikga asoslangan bir kutubxonadir.

**Django ni kimlar foydalanadi?** Django keng qo'llaniladigan va seviladigan bir frameworkdir. U dunyo bo'ylab bir qancha katta kompaniyalar tomonidan ishlatiladi. Misol uchun, Instagram, Pinterest, The Washington Post, NASA, Mozilla, Spotify, Eventbrite kabi etkazib beruvchilar Django-dan foydalanadigan kompaniyalardan faqat bir nechasi hisoblanadi. Django ning kuchli va ergonomik to'lovi va yaxshi dokumentatsiyasi tufayli, kompaniyalar uchun yuqori darajada ishlab chiqarishni osonlashtiradi.

**Django Design Pattern (MVT).** Django MVT (Model-View-Template) modelini o'rganish uchun katta ahamiyatga ega. U Django-dagi asosiy dizayn patternidir va tashqi ko'rinish va biznes-logicni ajratishga yordam beradi.

- **Model:** Model, ilova uchun ma'lumotlar saqlash va ularni boshqarishdan mas'ul bo'lgan qismidir. Model, ma'lumotlar bazasida ma'lumotlar obyektlari saqlashni ta'minlaydi.
- **View:** View, foydalanuvchi so'rovlarini qabul qilib, ma'lumotlarni olish, ularni ko'rsatish va biznes-logicni amalga oshirishni bajaradi. View, foydalanuvchiga qaysi ma'lumotlarni ko'rsatish kerakligini aniqlab, shu ma'lumotlarni template-ga uzatadi.
- **Template:** Template, foydalanuvchiga natijalarni ko'rsatish uchun HTML va Django-ninigina ko'rinishni beruvchi fayl hisoblanadi. Template, View dan kelgan ma'lumotlarni ishlayadi va foydalanuvchiga natijani qaytaradi.

**Django Admin Site.** Django Admin Site, Django-ni o'rnatgandan so'ng avtomatik ravishda yaratilgan boshlang'ich admin panelidir. U tizim boshqaruvchilari uchun ma'lumotlarni boshqarish va tahrirlashni osonlashtiradi. Django Admin Site orqali ma'lumotlarni kiritish, o'qish, yangilash, o'chirish, izlash va filtrlash, ma'lumotlarni import qilish, export qilish va boshqalar kabi amallarni bajarish mumkin.

Django, shunchaki katta va kuchli bir veb frameworkdir, uni Python dasturlash tilida yozilgan ilovalarni tezkor va qulay shaklda yaratish uchun ishlatish mumkin. Django-dan foydalanish orqali, turli turdagi veb ilovalarni sifatli va ishonchli shaklda yaratish mumkin.

## **5. Django yordamida web loyiha qurish**

**Django Loyihasi.** Django ilovasini boshlash uchun loyihani yaratish kerak. Loyiha, bir nechta ilova dasturlarini o'z ichiga olgan to'plam hisoblanadi. Loyiha yaratish uchun quyidagi komandalarni ishga tushiramiz:

- Django frameworkini komputerga o'rnatishimiz kerak. Bunda pythonning paketlar manajeri bo'lgan **pip** dan foydalanamiz:

```
pip install django
```

- Django paketi o'rnatilgan so'ng djangoda loyihani boshlaymiz:

```
django-admin startproject mysite
```

"Hello World" ilovasi. Loyiha bitta asos va bir nechta qism dasturlardan tashkil topadi. Bu qism dasturlar *app* lar deb ataladi. Hozir yangi *helloworld* nomli app yaratamiz:

```
python manage.py startapp helloworld
```

Bu komanda helloworld nomli ilova dasturini yaratadi. Keyin helloworld ilova dasturining views.py faylida quyidagi kodni qo'shamiz.

```
from django.http import HttpResponseRedirect

def hello(request):
    return HttpResponseRedirect("Hello, World!")
```

So'ng, loyihaning urls.py faylida helloworld ilovasining URL-routingini qo'shamiz.

```
from django.urls import path
from helloworld.views import hello

urlpatterns = [
    path('hello/', hello),
]
```

Bu kod loyihaga /hello/ yo'lovchisiga keldikda hello funksiyasini ishga tushiradi va "Hello, World!" xabarini qaytaradi.

**Statik va dinamik routing.** Django-da statik va dinamik routing mavjud. Statik routing, yo'lovchilarni belgilangan URL-ga to'g'ri yo'naltirish uchun ishlatiladi. Dinamik routing esa URL-dagi ma'lumotlarni olish uchun ishlatiladi.

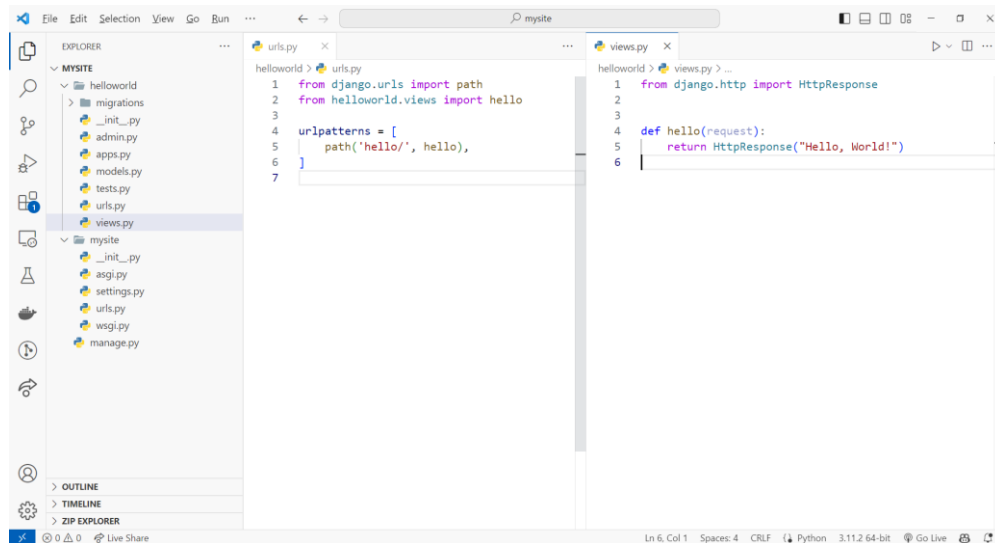
**Path Converter bilan dinamik routing.** Django-da Path Converterlar orqali dinamik routing yaratish mumkin. Path Converterlar, URL-dagi ma'lumotlarni parametrlar sifatida qabul qilishga imkon beradi. Misol uchun, `<int:id>` orqali `/users/5/` URL-da id nomli butun sonni qabul qilish mumkin.

**URL Mapping.** URL Mapping, URL-larni dastur funksiyalari bilan bog'lashda ishlatiladi. Django-da URL Mapping ilova loyihadagi `urls.py` faylda amalga oshiriladi. Bu faylda yo'lovchilar va ularga bog'liq funksiyalar yoki sinflar belgilanadi. Misol uchun, `urls.py` faylda quyidagi kodni ko'rishingiz mumkin:

```
from django.urls import path
from helloworld.views import hello

urlpatterns = [
    path('hello/', hello),
]
```

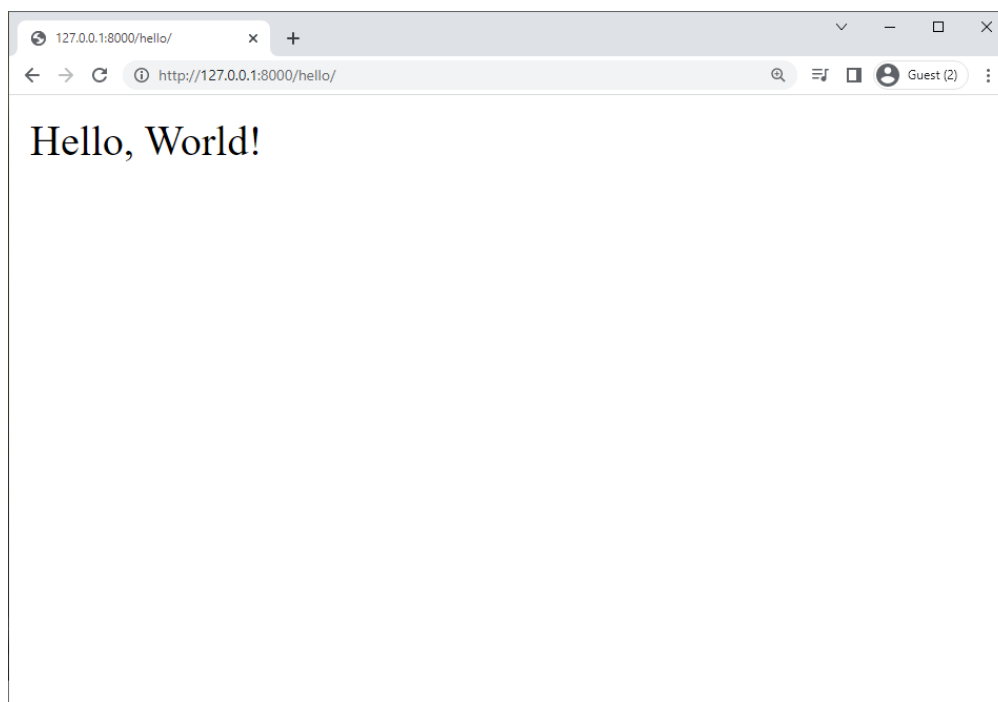
Bu kod loyiha tizimiga `/hello/` yo'lovchisiga keldikda `hello` funksiyasini ishga tushiradi.



2.5.1-rasm.

Loyihadagi fayllarni VS Code dasturining chap qismida ko'rish mumkin (2.5.1-rasm).





2.5.2-rasm.

Dastur natijasi Google Chrome browserida <http://127.0.0.1:8000/hello/> manzili orqali ko‘ra olamiz (2.5.2-rasm).

### **3- BOB. Suv xo‘jalik tashkilotlari uchun nazorati tizimini ishlab chiqish**

#### **1. Loyihaning ishlash tizimi**

Ushbu loyiha 2 xil turdagi qurilmalardan ma’lumotlarni qabul qilish uchun xizmat qiladi. Ulardan birinchisi kanallardan o‘tayotgan suv miqdorini nazorat qilish, ikkinchisi esa quduqlardagi suv miqdori, sho‘rlanish darajalarini doimiy nazoratda bilib turish uchun. Quyida qurilmalarning o‘rnatilgan holatini ko‘rishimiz mumkin.



3.1.1 va 3.1.2-rasmlar. Kanal qurilmalaring o‘rnatilgan holati.



3.1.3 va 3.1.4-rasmlar. Quduq qurilmalaring oʻrnatilgan holati.

Qurilmalar quyosh paneli orqali quvvatlanadi, hech qanday elektr tarmogʻiga ulanmagan. Qurilma ichida joylashgan akumliyator bilan bitta toʻliq quvvati bilan 1-2 hafta ishlashi mumkin. Buning yaxshi tomoni bulutli kunlar ketma-ket boʻlib qolganda ham qurilma ishdan toʻxtab qolmaydi.

Bu yerda biz quradigan loyiha ushbu qurilmalar uchun “Server side” yaʼni dasturiy taʼminoti hisoblanadi. Bunda qurilma va dasturiy qism REST API yordamida JSON koʻrinishida maʼlumot almashadi.

Kanal qurilmalari quyidagi koʻrinishda serverga maʼlumot yuboradi.

```
{
  "device_id": "KNNAVOI0027",
  "h": "152.70",
```

```
"bat": "3.98",  
"charging": "1",  
"net": "20",  
"latitude": "40.072849",  
"longitude": "65.703587",  
"re_settings": false,  
"taked": "0"  
}
```

Bu yerdagi har bir ma'lumot haqida alohida to'xtalamiz:

“device\_id” – bu har bir qurilma uchun takrorlanmas identifikator bo‘lib, bu id orqali server qurilmaning tanib oladi va kerakli vazifalarni bajaradi.

“h” – qurilmaning suv yuzasidan qancha santimetr yuqorida ekanligini bildiradi. Bu kattalik suv ko‘paygan vaqtda kamroq, suv ozaygan vaqtda esa kattaroq qiymat ko‘rsatadi. Serverda suv tubidan qurilmagacha bo‘lgan masofa avvaldan kiritilgan bo‘lib, o‘tayotgan suv miqdorini hisoblashda bu ikki masofa ayriladi. Ayirish yordamidan suv sathi balandligini bilib olishimiz mumkin. Bundan tashqari qurilmagan bo‘g‘langan ravishda dasturiy ta‘minotda jadval kiritilgan bo‘ladi. Bu jadvaldan suv sathi balandligi bilgan holda undan o‘tayotgan suv miqdorini (tonnalarda) bilib olish mumkin.

“bat” – bu qurilmaning zaryadini ko‘rsatib turuvchi kattalik (volt o‘lchov birligida keladi).

“charging” – bunda keladigan ma'lumot qurilmaning quvvatlanmoqda yoki quvvat olmayotganini bildiradi. Keladigan ma'lumot 0 yoki 1 bo‘lib, agar 0 kelsa quvvat olmayapti va 1 kelsa qurilma quvvatlanmoqda.

“net” – bu kattalik qurilmada jo‘ylashgan sim kartaga bog‘liq ravishda aloqa sifatini bildiradi. Qurilmada joylashgan bu sim modul internet orqali serverga ma'lumot yuborish uchun xizmat qiladi.

“latitude” va “longitude” – bu ikki kattalik qurilma joylashivi koordinatalari hisoblanadi. Qurilma har safar ma'lumot yuborganda o‘zing joylashuvini shu ko‘rinishda serverga uzatadi.

“re\_settings” – bu qurilmani qayta sozlash funksiyasi uchun kerak bo‘lib, odatiy holatda “false” qiymat yuboradi. Agar “true” qiymat keladigan bo‘lsa serverda ba‘zi ma'lumotlar yangilanib oladi.

“taked” – bunda keladigan ma'lumot faqatgina 0 yoki 1 bo‘lib, bu qurilmanining o‘g‘irlanishiga qarshi himoya tizimi hisoblanadi. Agar kimdir qurilmaga kimdir teginsa u

serverga tez-tez o'z joylashuvini yuboradi va bu ma'lumotlar serverda alohida jadvalda saqlanib boriladi.

Quduq qurilmalari quyidagi ko'rinishda serverga ma'lumot yuboradi.

```
{  
  "device_id": "QDUEHFU0027",  
  "level": "64.09",  
  "meneral": "1.95",  
  "temp": "14.14",  
  "bat": "4.07",  
  "charging": "0",  
  "net": "22",  
  "latitude": "41.527268",  
  "longitude": "69.315487",  
  "taked": "0"  
}
```

Bu json ko'rinishidagi ma'lumot yuqorida ko'rganimiz kanal qurilmasinikidan ozgina farq qiladi. Shuning uchun barchasini emas balki ba'zi qiymatlarni ko'rib chiqishimiz yetarli bo'

“level” – bu qurilmadan suv sathigacha bo'lgan masofani bildiradi.

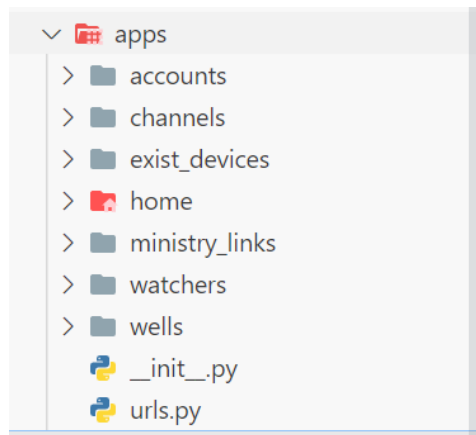
“meneral” – bu kattalik suvning sho'rlanish darajasini ko'rsadi

“temp” – bu kattalik suvning haroratini ko'rsatadi.

Xulosa qiladigan bo'lsak qurilmalar serverga doimiy shu ko'rinishida ma'lumot uzatib boradi. Serverda esa bu ma'lumotlar to'g'ri saqlanishi va kerakli ma'lumotlarni foydalanuvchiga chiroyli interfeys bilan chiqarib berish zarur bo'ladi. Bundan tashqari bu yerga kelib tushayotgan ma'lumotlar “Qishloq xo'jaligi vazirligi”ga ham yuborilib turadi.

## 2. Loyihani modellashtirish

Foydalanuvchi qurilmalarni dasturiy ta'minotda qo'shishi va boshqarishi uchun albatta o'z hisobini yaratishi kerak bo'ladi. Buning uchun foydalanuvchilar modelini tuzib olishimiz zarur. Loyihani qurishda Django'dan foydalanamiz va bir nechta qism dasturlar (app'lar) hosil qilamiz. Foydalanuvchilar modelini ham alohida app'da tuzib olamiz.



3.2.1-rasm. Loyihaning asosiy qismlari fayllar daraxti

*apps/accounts/models.py* fayli

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import gettext_lazy as _

class User(AbstractUser):
    username = models.CharField(
        max_length=20,
        unique=True,
        verbose_name=_('Phone number'),
        error_messages={
            'unique': _("A user with that phone number already exists."),
        },
    )

    region = models.CharField(
        max_length=255,
        verbose_name=_('Region'),
        blank=True,
        null=True,
    )

    city = models.CharField(
        max_length=255,
        verbose_name=_('City'),
        blank=True,
        null=True,
    )

    org_name = models.CharField(
        max_length=500,
        verbose_name=_('Organization name'),
```



```

        blank=True,
        null=True,
    )

    telegram_id = models.CharField(
        max_length=255,
        verbose_name=_('Telegram ID'),
        blank=True,
        null=True,
    )

    is_master = models.BooleanField(
        default=False,
        verbose_name=_('Is channels master')
    )

    def __str__(self):
        return self.username

    class Meta:
        verbose_name = _('User')
        verbose_name_plural = _('Users')

```

Bu yerda User modelini qurish hosil qilishda `AbstractUser` modelidan voris olinganini ko‘rishimiz mumkin. Bu bizga user modeli uchun kerakli bo‘ladigan ba’zi qismlarini tayyor qilib beradi.

Qurilmalar ushbu tizimda o‘rnatilmasdan avval kiritib qo‘yiladi. Bunda ularning qaysi turdagi qurilma ekanligi va uning id raqami kiritiladi. Buning uchun esa Device nomi model qurishimiz kerak bo‘ladi. Buning uchun ham alohida mavjud qurilmalar (`exist_devices`) nomli app hosil qilamiz.

*apps/exist\_devices/models.py* fayli

```

from django.db import models
from django.utils.translation import gettext_lazy as _

class Device(models.Model):
    DEVICE_TYPE_CHOICES = (
        ('channel', _('Water Channel')),
        ('well', _('Water Well')),
    )

```

```

id = models.CharField(
    max_length=11,
    primary_key=True,
    verbose_name=_('ID')
)
type = models.CharField(
    max_length=20,
    choices=DEVICE_TYPE_CHOICES,
    verbose_name=_('Type of device')
)
added_at = models.DateTimeField(
    auto_now_add=True,
    verbose_name=_('Added at')
)
is_active = models.BooleanField(
    default=False,
    verbose_name=_('Is active'),
    blank=True
)
def __str__(self):
    return self.id
class Meta:
    verbose_name = _('Device')
    verbose_name_plural = _('Devices')

```

Foydalanuvchi qurilmani o‘ziga qo‘shgan payt bu yerdagi `is_active` maydoni `true` qiymatga o‘zgaradi va bunda qurilmani boshqa foydalanuvchi qo‘sha olmay qoladi.

Kanal qurilmalari uchun alohida app (channels) hosil qilamiz. Bu qismda 4 ta model quramiz, bular:



ChannelDevice – kanal qurilmalarini kiritish uchun. Bunda qurilmaga tegishli ma'lumotlar saqlanadi.

ChannelDeviceVolumeTable – bu model qurilmaning suv sathiga mos ravishda undan o'tayotgan suv miqdorini aniqlash uchun.

ChannelMessage – bu model qurilma yuborayotgan ma'lumotlarni saqlab borish uchun bo'ladi.

ChannelMovement – bu model qurilmada siljish aniqlanganda keladigan joylashuv ma'lumotlarni saqlab borish uchun bo'ladi.

Qudug qurilmalari uchun ham alohida app (wells) qo'shamiz. Bu qismda 3 ta model quramiz, bular:

WellDevice – bunda quduq qurilmalari ma'lumotlari kiritiladi.

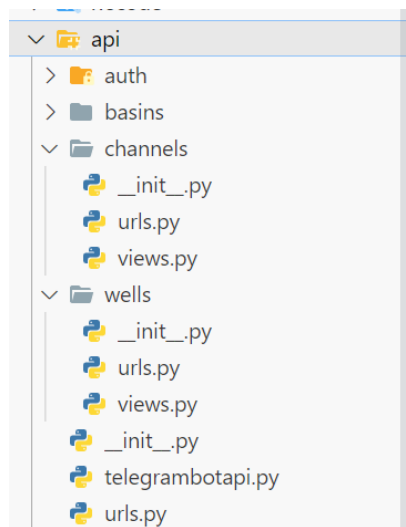
WellDeviceMessage – bunda qurilma yuborgan ma'lumotlar saqlanib boradi.

WellMovement – bu modelda quduq qurilmasida siljish aniqlangan paytda keladigan joylashuv ma'lumotlari saqlanadi.

### **3. Dasturiy ta'minotning API qismini qurish**

Biz API tayyorlashda "Django Rest Framework" dan foydalanamiz. Bu framework Rest API'lar qurish uchun juda qulay bo'lgan vosita hisoblanadi. Bizda ikki xil qurilmalar mavjud bo'lgani sabab asosiy 2 ta API qurishimiz kerak bo'ladi.

**Kanal qurilmasi uchun API tayyorlashda mantiqiy qismni qurish.** Bu qismni qurishda "api" nomi bilan yangi app hosil qilamiz. Unda views.py nomi faylda dastur kodlari yozishimiz mumkin.



### 3.3.1-rasm. Loyiha API qismi fayllar daraxti

*api/channels/views.py* fayli

```
import datetime
import decimal
import json
import requests

from django.utils import timezone

from rest_framework.response import Response
from rest_framework.decorators import api_view
from rest_framework.status import HTTP_201_CREATED, HTTP_400_BAD_REQUEST

from apps.ministry_links.models import MinistryChannelLink
from apps.channels.models import (
    ChannelDevice,
    ChannelMessage,
    ChannelDeviceVolumeTable,
    ChannelMovement,
)
from api.telegrambotapi import send_message_to_user_by_telegram,
send_message_to_admin

def get_formatted_time(now: datetime) -> str:
    return now.strftime("%H:%M:%S %d/%m/%Y")
```

```

def update_device_location(
    device: ChannelDevice, latitude: str, longitude: str
) -> None:
    device.latitude = latitude
    device.longitude = longitude
    device.save()

def send_channel_data_to_ministry_server(
    device: ChannelDevice, new_message: ChannelMessage
):
    if device.ministry_id and device.permission_to_send:
        try:
            url = MinistryChannelLink.objects.first().link
            res = requests.post(
                url=url,
                headers={"Content-Type": "application/json"},
                data=json.dumps(
                    {
                        "code": device.ministry_id,
                        "data": {
                            "level": str(
                                decimal.Decimal(device.full_height)
                                - decimal.Decimal(new_message.h)
                                if device.full_height
                                else device.height
                            ),
                            "volume": str(new_message.water_volume),
                            "vaqt":
                                get_formatted_time(datetime.datetime.now()),
                        },
                    }
                ),
            )
            if res.status_code == 200:
                if res.json().get("status") == "success":
                    new_message.is_sent = True
                    new_message.save()
            except Exception as e:
                print(e)

@api_view(["POST"])
def receive_channel_message(request):

```

```

data = request.data
h = data.get("h")
bat = data.get("bat")
is_charging = data.get("charging")
net = data.get("net")
latitude = data.get("latitude")
longitude = data.get("longitude")
device_id = data.get("device_id")
taken = data.get("taken", 0)

re_settings = data.get("re_settings", False)

# Get device
device = ChannelDevice.objects.filter(device_id=device_id).first()

if int(taken):
    last_location = (
        ChannelMovement.objects.filter(device=device).order_by("created
_at").last()
    )

    ChannelMovement.objects.create(
        device=device, latitude=latitude, longitude=longitude
    )

    update_device_location(device, latitude, longitude)

    # send message owner of device about device stolen
    condition = (
        timezone.now() - last_location.created_at >
datetime.timedelta(minutes=1)
        if last_location
        else True
    )
    if condition:
        send_message_to_user_by_telegram(device, "channel")

    return Response(
        {
            "request": "success",
            "datetime": get_formatted_time(datetime.datetime.now()),
        },
        status=HTTP_201_CREATED,
    )

```

```

# Check message is valid
if all((h, bat, is_charging, net, latitude, longitude, device)):
    # Create new message
    new_message = ChannelMessage.objects.create(
        device=device, h=h, bat=bat,
is_charging=bool(int(is_charging)), net=net
    )

    # Update device full height
    if re_settings and not bool(device.full_height):
        device.full_height = device.height +
decimal.Decimal(new_message.h)
        device.save()

    # Set volume to message by volume table
    if device.full_height is not None:
        water_height = round(
            device.full_height + device.height_conf -
decimal.Decimal(new_message.h)
        )
        water_height_ones = water_height % 10
        water_height_tens = water_height - water_height_ones

        volume_row = (
            ChannelDeviceVolumeTable.objects.filter(device_id=device_id
)
            .filter(tens=water_height_tens)
            .first()
        )

        if volume_row is not None:
            new_message.water_volume =
volume_row.get_value(water_height_ones)
            new_message.save()

    # Update device height
    update_device_location(device, latitude, longitude)

    # Send message to ministry server
    send_channel_data_to_ministry_server(device, new_message)

    # Success response
    return Response(

```

```

        {
            "request": "success",
            "datetime": get_formatted_time(datetime.datetime.now()),
        },
        status=HTTP_201_CREATED,
    )

    # Error response
    return Response(
        {"request": "error", "datetime":
get_formatted_time(datetime.datetime.now())},
        status=HTTP_400_BAD_REQUEST,
    )

```

Bu mantiqiy qism kanal qurilmalari uchun ishlovchi qism bo'lib, yuqorida gaplashib o'tilgan json ko'rinishidagi ma'lumotni qabul qiladi. Dastur ishlashi natijasida esa:

```

{
    "request": "success",
    "datetime": "12:00:00 01/01/2018"
}

```

ko'rinishida muvaffaqiyatli javob yoki

```

{
    "request": "error",
    "datetime": "12:00:00 01/01/2018"
}

```

Ko'rinishida xatolik xabari qaytadi.

#### **4. Loyiha uchun Telegram Bot**

Aiogram Telegram botning Python tillariga mos moduli hisoblanadi va oson va samarali Telegram botlarini yaratish imkonini beradi. Ushbu modul Telegram Bot API bilan ishlash uchun oson interfeyslar va ko‘plab yordamchi funktsiyalarni taklif etadi.

Aiogram moduli qisqa va intuitiv interfeyslar orqali Telegram botlarini boshqarish imkonini beradi. Bu modul yordamida quyidagi funktsiyalarni bajarish mumkin:

- Xabarlar: Foydalanuvchidan kelgan xabarlar, rasm, video, dokumentlar, audio va boshqa fayllarni qabul qilish va javob berish.
- Klaviaturalar: Oson va samarali klaviaturalarni yaratish va ularga xabar javoblash.
- Tasdiqlash tugmachalari: Foydalanuvchiga savollar berish va ularni tasdiqlash uchun tugmalarni qo‘shish.
- Ma'lumotlar bazasi: Foydalanuvchilarning ma'lumotlari, statistikalar va boshqa ma'lumotlar uchun ma'lumotlar bazasini o‘rganish va ishlash.
- Guruhlar: Guruhlar, superguruhlar va kanallar bilan ishlash imkonini beradi.
- Asinxron ishlash: Xabarlar, ma'lumotlarni olish va qo‘shish, tarmoqlar bilan asinxron ishlash imkonini beradi.

Aiogram modulini o‘rnatish uchun quyidagi komanda orqali PyPI (Python Package Index) dan o‘rnatishingiz mumkin:

```
pip install aiogram
```

### Kichik Telegram bot loyihasi

Bu kichik loyiha Telegram botiga kelgan xabarlarga javob beradi va foydalanuvchining ismidan foydalanin u bilan salomlashadi. Bu misolni o‘zingizga moslashtirishingiz mumkin:

```
import logging
```

```
from aiogram import Bot, Dispatcher, types
```

```
from aiogram.contrib.fsm_storage.memory import MemoryStorage
```

```
# Bot tokenini o‘zingizning bot tokeningiz bilan almashtiring
```

```
TOKEN = 'your_bot_token'
```

```
# Botni yaratish
```

```
bot = Bot(token=TOKEN)
```

```
storage = MemoryStorage()
```

```
dp = Dispatcher(bot, storage=storage)
```

```
# Loglarni sozlash
```

```
logging.basicConfig(level=logging.INFO)
```

```
# /start komandasi uchun buyruqni qabul qilish
```

```
@dp.message_handler(commands=['start'])
```

```
async def start(message: types.Message):
```

```
    # Botni ismi bilan xush kelibsiz xabarini yuborish
```

```
    await message.reply(f"Assalomu alaykum,  
{message.from_user.first_name}!")
```



```
# Xabarlar uchun buyruqni qabul qilish

@dp.message_handler()

async def echo(message: types.Message):

    # Xabarni qaytadan yuborish

    await message.reply(message.text)

# Botni ishga tushirish

if __name__ == '__main__':

    from aiogram import executor

    executor.start_polling(dp, skip_updates=True)
```

Ushbu kod xabarlar qabul qilib, ularga javob qaytaradi. Botning TOKEN o'zgaruvchisini Telegram bot token bilan almashtirish lozim bo'ladi. Bu token Telegramdagi Bot Father telegram botidan olinadi.

Uning to'g'ri ishlashini tekshirish uchun Telegramda botni ishga tushirish uchun komandasi terminal dasturida beramiz. Telegram bot dasturini yurgazish uchun quyidagi komanda beriladi:

```
python dastur_fayli_nomi.py
```

## XULOSA

Bu loyihani qilish davomida men ko‘plab texnologiyalarni o‘rgandim. Bunday loyihalar orqali bir necha inson qiladigan ishlarni avtomatlashtirish mumkin. Avtomatlashtirish bizdan doimiy qiladigan ishlarni olib qo‘ysada bizga yangi imkoniyatlar ochib beradi. Masalar doim bir xil ish qilishdan inson zerikadi. Endi biz shu vaqtda sohada yanada bilimlarimizni oshirishga yoki yangi sohalarni sinab ko‘rishga vaqt topishimiz mumkin bo‘ladi. Men yana shuni o‘rgandimki python dasturlash tili yordamida veb ilovalar qilish juda qular va oson. Ayniqsa Django buni yanada oshirishga xizmat qiladi. Kengayuvchan loyihalar qurish, o‘zining MVT arxitekturasi va kuchli ORM tizimi uning eng katta yutuqlaridan hisoblanadi. Shular orqali Django pythonda veb loyihalar qurish uchun yuqori o‘rinlarda sanaladi. Men bu loyihani qilish davomida yuqorida aytganimdek juda ko‘plab yangi bilimlarni o‘rganishga majbur bo‘ldim. Hattoki qishloq xo‘jaligi sohasini ham oz bo‘lsada o‘rganishga to‘g‘ri keldi. O‘zim dasturchi muhandis sifatida “Hardware engineering” sohasi bilan aloqador ish qilganimdan xursadaman. Bu sohada ishlovchilar bilan kelishgan holatda ma’lumot almashish bosqichlari eng qiziq va murakkab qismlaridan bo‘ldi.

Dasturiy taminotning asosiy vazifalari quyidagilardan iborat bo‘ldi:

- Foydalanuvchilarni ro‘yxatdan o‘tkazish (telegram bot orqali)
- Qurilmalarni osonlik bilan qo‘shish va sozlash
- Ma’lumotlarni avtomatik Qishloq xo‘jaligi ba’zasiga yuborish
- Ma’lumotlarni istalgan vaqt excel ko‘rinishida yuklab olish
- Qurilmaga qo‘shimcha kuzatuvchilar qo‘shish ularni boshqarish
- Admin sahifasi orqali barcha ma’lumotlarni ko‘rish

# **FOYDALANILGAN MANBALAR**

## **Adabiyotlar**

1. Django for APIs: Build web APIs with Python and Django - William S. Vincent. Feb, 2022
2. Django: Web Development with Python - Samuel Dauzon, Aidas Bendoraitis, Arun Ravindran. Avg, 2016

## **Internet manbalar**

1. <https://realpython.com/>
2. <https://www.tutorialspoint.com/>
3. <https://www.educative.io/>

## **Foydalanilgan dasturiy vositalar manzillari**

1. <https://www.python.org/downloads/>
2. <https://code.visualstudio.com/download>
3. <https://www.postgresql.org/download/>

# ILOVALAR

apps/accounts/models.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import gettext_lazy as _

class User(AbstractUser):
    username = models.CharField(
        max_length=20,
        unique=True,
        verbose_name=_('Phone number'),
        error_messages={
            'unique': _("A user with that phone number already exists."),
        },
    )

    region = models.CharField(
        max_length=255,
        verbose_name=_('Region'),
        blank=True,
        null=True,
    )

    city = models.CharField(
        max_length=255,
        verbose_name=_('City'),
        blank=True,
        null=True,
    )

    org_name = models.CharField(
        max_length=500,
        verbose_name=_('Organization name'),
        blank=True,
        null=True,
    )

    telegram_id = models.CharField(
        max_length=255,
        verbose_name=_('Telegram ID'),
        blank=True,
        null=True,
    )
```

```

is_master = models.BooleanField(
    default=False,
    verbose_name=_('Is channels master')
)

def __str__(self):
    return self.username

class Meta:
    verbose_name = _('User')
    verbose_name_plural = _('Users')

```

## apps/accounts/views.py

```

from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login, logout

def login_view(request):
    if request.user.is_authenticated:
        return redirect('home')
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
    return render(request, 'auth/login.html')

def logout_view(request):
    logout(request)
    return redirect('login')

```

## apps/accounts/urls.py

```

from django.urls import path
from . import views

```

```
urlpatterns = [
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
]
```

apps/exist\_devices/models.py

```
from django.db import models
from django.utils.translation import gettext_lazy as _
```

```
class Device(models.Model):
    DEVICE_TYPE_CHOICES = (
        ('channel', _('Water Channel')),
        ('well', _('Water Well')),
    )

    id = models.CharField(
        max_length=11,
        primary_key=True,
        verbose_name=_('ID')
    )

    type = models.CharField(
        max_length=20,
        choices=DEVICE_TYPE_CHOICES,
        verbose_name=_('Type of device')
    )

    added_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name=_('Added at')
    )

    is_active = models.BooleanField(
        default=False,
        verbose_name=_('Is active'),
        blank=True
    )

    def __str__(self):
        return self.id

    class Meta:
```

```
verbose_name = _('Device')
verbose_name_plural = _('Devices')
```

apps/exist\_devices/admin.py

```
from django.contrib import admin
from .models import Device

# Register your models here.

class DeviceAdmin(admin.ModelAdmin):
    list_display = ('id', 'type', 'added_at', 'is_active')
    list_filter = ('is_active', 'type')
    search_fields = ('id',)
    ordering = ('-added_at',)
    date_hierarchy = 'added_at'

admin.site.register(Device, DeviceAdmin)
```

apps/channels/models.py

```
from datetime import timedelta
from django.utils import timezone

from django.db import models
from django.utils.translation import gettext_lazy as _
from django.contrib.auth import get_user_model
from apps.exist_devices.models import Device

class ChannelDevice(models.Model):
    device = models.OneToOneField(
        to=Device,
        on_delete=models.CASCADE,
        primary_key=True,
        verbose_name=_('Device')
    )

    ministry_id = models.CharField(
        max_length=20,
```

```

        blank=True,
        verbose_name=_('Ministry ID')
    )

    permission_to_send = models.BooleanField(
        default=False,
        verbose_name=_('Permission to send data')
    )

    name = models.CharField(
        max_length=100,
        verbose_name=_('Name')
    )

    user = models.ForeignKey(
        to=get_user_model(),
        on_delete=models.SET_NULL,
        related_name='channeldevices',
        null=True,
        blank=True,
        verbose_name=_('Belong to')
    )

    master = models.ForeignKey(
        to=get_user_model(),
        on_delete=models.SET_NULL,
        related_name='channel_devices',
        null=True,
        blank=True,
        verbose_name=_('Master')
    )

    phone_number = models.CharField(
        max_length=13,
        verbose_name=_('Phone number')
    )

    full_height = models.DecimalField(
        default=0.0,
        max_digits=8,
        decimal_places=2,
        verbose_name=_('Full height (sm)')
    )

    height = models.DecimalField(
        default=0.0,

```



```

        max_digits=8,
        decimal_places=2,
        verbose_name=_('Height of water (sm)')
    )

    height_conf = models.IntegerField(
        default=0,
        blank=True,
        verbose_name=_('Height conf (sm)')
    )

    latitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Latitude of location'),
        blank=True,
        null=True
    )

    longitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Longitude of location'),
        blank=True,
        null=True
    )

    def __str__(self):
        return f'{self.name} - {self.device.id}'

    def save(self, *args, **kwargs):
        self.device.is_active = True
        self.device.save()
        super(ChannelDevice, self).save(*args, **kwargs)

    def delete(self, *args, **kwargs):
        self.device.is_active = False
        self.device.save()
        super(ChannelDevice, self).delete(*args, **kwargs)

    class Meta:
        verbose_name = _('Channel device')
        verbose_name_plural = _('Channel devices')

class ChannelDeviceVolumeTable(models.Model):

```

```

device = models.ForeignKey(
    to=ChannelDevice,
    on_delete=models.CASCADE,
    verbose_name=_('Device')
)

tens = models.IntegerField(
    default=0,
    verbose_name=_('Tens')
)

zero = models.FloatField(default=0, verbose_name='0')
one = models.FloatField(default=0, verbose_name='1')
two = models.FloatField(default=0, verbose_name='2')
three = models.FloatField(default=0, verbose_name='3')
four = models.FloatField(default=0, verbose_name='4')
five = models.FloatField(default=0, verbose_name='5')
six = models.FloatField(default=0, verbose_name='6')
seven = models.FloatField(default=0, verbose_name='7')
eight = models.FloatField(default=0, verbose_name='8')
nine = models.FloatField(default=0, verbose_name='9')

def __str__(self):
    return f'{self.id} / {self.device} / {self.tens}'

def get_value(self, ones):
    match_values = {
        0: self.zero,
        1: self.one,
        2: self.two,
        3: self.three,
        4: self.four,
        5: self.five,
        6: self.six,
        7: self.seven,
        8: self.eight,
        9: self.nine
    }
    return match_values.get(ones)

class Meta:
    verbose_name = _('Channel device volume table')
    verbose_name_plural = _('Channel device volume tables')

class ChannelMessage(models.Model):

```

```

device = models.ForeignKey(
    to=ChannelDevice,
    on_delete=models.CASCADE,
    verbose_name=_('Channel device')
)

is_sent = models.BooleanField(
    default=False,
    verbose_name=_('Is sent')
)

h = models.DecimalField(
    max_digits=7,
    decimal_places=2,
    verbose_name=_('From device to water (sm)')
)

water_volume = models.DecimalField(
    max_digits=12,
    decimal_places=2,
    blank=True,
    null=True,
    verbose_name=_('Volume of water (cubic meters/sec)')
)

bat = models.DecimalField(
    max_digits=5,
    decimal_places=2,
    verbose_name=_('Battery power (volt)')
)

is_charging = models.BooleanField(
    blank=True,
    null=True,
    verbose_name=_('Is charging')
)

net = models.SmallIntegerField(
    verbose_name=_('Network quality')
)

created_at = models.DateTimeField(
    auto_now_add=True,
    verbose_name=_('Added time')
)

```

```

def __str__(self):
    return self.device.name

def get_water_height(self):
    return self.device.full_height - self.h if self.device.full_height else
self.device.height

def get_device_active(self):
    return timezone.now() - self.created_at < timedelta(days=1)

class Meta:
    verbose_name = _('Channel message')
    verbose_name_plural = _('Channels messages')

class ChannelMovement(models.Model):
    device = models.ForeignKey(
        to=ChannelDevice,
        on_delete=models.CASCADE,
        verbose_name=_('Channel device')
    )

    latitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Latitude of location'),
        blank=True,
        null=True
    )

    longitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Longitude of location'),
        blank=True,
        null=True
    )

    created_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name=_('Added time')
    )

    def __str__(self):
        return self.device.name

```

```

class Meta:
    verbose_name = _('Channel movement message')
    verbose_name_plural = _('Channels movement messages')

```

apps/channels/views.py

```

from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect

```

```

from .forms import ChannelDeviceForm, ChannelDeviceEditForm, ChannelDeviceVolumeForm
from .models import ChannelDevice, ChannelMessage, ChannelDeviceVolumeTable

```

```

@login_required(login_url='login')
def home_page(request):
    devices = sorted(
        ChannelDevice.objects.filter(master=request.user),
        key=lambda channeldevice: channeldevice.device.id[-4:],
        reverse=True
    )
    context = {
        'channel_devices': [
            {
                'device': device,
                'device_last_message':
ChannelMessage.objects.filter(device=device).last()
            } for device in devices
        ]
    }
    return render(request, 'channels/channel_devices_list.html', context)

```

```

@login_required(login_url='login')
def device_detail(request, device_id):
    selected_device =
ChannelDevice.objects.filter(master=request.user).filter(device_id=device_id).first()
    if selected_device is None:
        return render(request, 'home/404.html')
    context = {
        'selected_device': selected_device,
        'latitude': str(selected_device.latitude),
        'longitude': str(selected_device.longitude),
        'selected_device_messages':

```

```

        ChannelMessage.objects.filter(device_id=device_id).order_by('-
created_at')[:20]
    }
    return render(request, 'channels/channel_device_detail.html', context)

@login_required(login_url='login')
def delete_device(request, device_id):
    selected_device =
ChannelDevice.objects.filter(master=request.user).filter(device_id=device_id).first()
    if selected_device is not None:
        selected_device.master = None
        selected_device.save()
    return redirect('channels_dashboard')

@login_required(login_url='login')
def edit_device(request, device_id):
    selected_device =
ChannelDevice.objects.filter(master=request.user).filter(device_id=device_id).first()
    if selected_device is not None:
        if request.method == 'POST':
            form = ChannelDeviceEditForm(data=request.POST, instance=selected_device)
            if form.is_valid():
                form.save()
                return redirect('channel_device_detail', selected_device.device_id)
            form = ChannelDeviceEditForm(instance=selected_device)
        else:
            form = ChannelDeviceEditForm()
    context = {
        'device_id': selected_device.device_id,
        'title': 'Edit channel device',
        'form': form
    }
    return render(request, 'channels/add_or_edit_device.html', context)

@login_required(login_url='login')
def add_new_device(request):
    form = ChannelDeviceForm()
    if request.method == 'POST':
        form = ChannelDeviceForm(data=request.POST)
        if form.is_valid():
            form.save(master=request.user)
            return redirect('channels_dashboard')
    context = {
        'title': 'Add channel device',

```

```

        'form': form
    }
    return render(request, 'channels/add_or_edit_device.html', context)

@login_required(login_url='login')
def add_new_row_for_volume_table(request, device_id):
    last_row =
ChannelDeviceVolumeTable.objects.filter(device_id=device_id).order_by('tens').last()
    if last_row is None:
        obj = ChannelDeviceVolumeTable.objects.create(device_id=device_id, tens=0)
    else:
        obj = ChannelDeviceVolumeTable.objects.create(device_id=device_id,
tens=last_row.tens + 10)
    return redirect('edit_new_row_for_volume_table', obj.id)

@login_required(login_url='login')
def edit_new_row_for_volume_table(request, row_id):
    obj = ChannelDeviceVolumeTable.objects.filter(pk=row_id).first()
    if obj is not None:
        device_id = obj.device_id
        if request.method == 'POST':
            form = ChannelDeviceVolumeForm(data=request.POST, instance=obj)
            if form.is_valid():
                form.save()
                return redirect('volume_table', device_id)
        context = {
            'title': 'Edit row',
            'device_id': device_id,
            'form': ChannelDeviceVolumeForm(instance=obj),
            'selected_device_volume_table':
                ChannelDeviceVolumeTable.objects.filter(device_id=device_id).order_by(
'tens')
        }

        return render(request, 'channels/edit_row_volume_table.html', context)

    return redirect('channels_dashboard')

@login_required(login_url='login')
def delete_new_row_for_volume_table(request, row_id):
    obj = ChannelDeviceVolumeTable.objects.filter(pk=row_id).first()
    if obj is not None:
        device_id = obj.device_id
        obj.delete()

```

```

        return redirect('volume_table', device_id)

    return redirect('channels_dashboard')

@login_required(login_url='login')
def volume_table(request, device_id):
    context = {
        'title': 'Device rows',
        'device_id': device_id,
        'selected_device_volume_table':
            ChannelDeviceVolumeTable.objects.filter(device_id=device_id).order_by('ten
s')
    }
    return render(request, 'channels/edit_row_volume_table.html', context)

```

## apps/channels/forms.py

```

from django import forms
from django.forms import ModelForm
from .models import ChannelDevice, ChannelDeviceVolumeTable

from apps.exist_devices.models import Device

class ChannelDeviceForm(ModelForm):
    # change the widget of the device field to text input
    device = forms.ModelChoiceField(
        queryset=Device.objects.filter(is_active=False).filter(type='channel'),
        widget=forms.TextInput(attrs={'autocomplete': 'off'}),
        error_messages={
            'required': 'Device id bo\'lishi shart',
            'invalid_choice': 'Noto\'g\'ri device id kiritildi'
        }
    )

    class Meta:
        model = ChannelDevice
        fields = ('device', 'name', 'phone_number', 'height')

    def __init__(self, *args, **kwargs):
        super(ChannelDeviceForm, self).__init__(*args, **kwargs)

        for visible in self.visible_fields():

```



```

        visible.field.widget.attrs['class'] = 'w-full mt-2 mb-5 p-1 border-gray-
900 rounded-md ' \
                                                'focus:border-indigo-600 focus:ring
focus:ring-opacity-40' \
                                                'focus:ring-indigo-500 border-2
border-black border-slate-500'

```

```

def save(self, master=None, commit=True):
    obj = super(ChannelDeviceForm, self).save(commit=False)
    if master:
        obj.master = master
    if commit:
        obj.save()
    return obj

```

```

class ChannelDeviceEditForm(ModelForm):
    class Meta:
        model = ChannelDevice
        fields = (
            'name', 'phone_number', 'ministry_id', 'permission_to_send',
            'full_height', 'height', 'height_conf', 'latitude', 'longitude'
        )

```

```

def __init__(self, *args, **kwargs):
    super(ChannelDeviceEditForm, self).__init__(*args, **kwargs)

    for visible in self.visible_fields():
        visible.field.widget.attrs['class'] = 'w-full mt-2 mb-5 p-1 border-gray-
900 rounded-md ' \
                                                'focus:border-indigo-600 focus:ring
focus:ring-opacity-40' \
                                                'focus:ring-indigo-500 border-2
border-black border-slate-500'

```

```

class ChannelDeviceVolumeForm(ModelForm):
    class Meta:
        model = ChannelDeviceVolumeTable
        fields = (
            'tens', 'zero', 'one', 'two', 'three', 'four', 'five',
            'six', 'seven', 'eight', 'nine'
        )

```

```

def __init__(self, *args, **kwargs):
    super(ChannelDeviceVolumeForm, self).__init__(*args, **kwargs)
    for visible in self.visible_fields():

```

```

        visible.field.widget.attrs['class'] = 'w-32 -mt-1 p-1 border-gray-900
rounded-md ' \
                                                'focus:border-indigo-600 focus:ring
focus:ring-opacity-40' \
                                                'focus:ring-indigo-500 border-2
border-black border-slate-500'

```

apps/channels/admin.py

```

from django.utils.translation import gettext_lazy as _
from django.contrib import admin
from .models import ChannelDevice, ChannelMessage, ChannelDeviceVolumeTable,
ChannelMovement
from apps.exist_devices.models import Device

class ChannelDeviceAdmin(admin.ModelAdmin):
    list_display = (
        'name', 'device', 'ministry_id', 'permission_to_send',
        'user', 'phone_number', 'full_height', 'height', 'height_conf', 'get_region',
        'get_city'
    )
    list_filter = ('user__region', 'user')
    search_fields = ('name', 'phone_number', 'ministry_id')

    # Additional fields

    def get_region(self, obj):
        if obj.user:
            return obj.user.region
        return None

    def get_city(self, obj):
        if obj.user:
            return obj.user.city
        return None

    get_region.short_description = _('Region')
    get_city.short_description = _('City')

    # Custom form

    obj_id = None

    def get_form(self, request, obj=None, **kwargs):
        if obj:
            self.obj_id = obj.device.id

```

```

        return super(ChannelDeviceAdmin, self).get_form(request, obj, **kwargs)

    def formfield_for_foreignkey(self, db_field, request, **kwargs):
        if db_field.name == 'device':
            if self.obj_id:
                kwargs['queryset'] = Device.objects.filter(type='channel')
            else:
                kwargs['queryset'] = Device.objects.filter(
                    type='channel').filter(is_active=False)
        return super(ChannelDeviceAdmin, self).formfield_for_foreignkey(db_field,
request, **kwargs)

class ChannelMessageAdmin(admin.ModelAdmin):
    list_display = ('device', 'is_sent', 'h', 'water_volume',
                    'bat', 'is_charging', 'net', 'created_at')
    list_filter = ('device',)
    ordering = ('-created_at',)

class ChannelDeviceVolumeTableAdmin(admin.ModelAdmin):
    list_display = (
        'device', 'tens', 'zero', 'one', 'two', 'three',
        'four', 'five', 'six', 'seven', 'eight', 'nine'
    )
    list_filter = ('device',)
    ordering = ('device', 'tens')

class ChannelMovementAdmin(admin.ModelAdmin):
    list_display = ['device', 'latitude', 'longitude', 'created_at', 'id']
    list_filter = ['device']

admin.site.register(ChannelDevice, ChannelDeviceAdmin)
admin.site.register(ChannelMessage, ChannelMessageAdmin)
admin.site.register(ChannelDeviceVolumeTable, ChannelDeviceVolumeTableAdmin)
admin.site.register(ChannelMovement, ChannelMovementAdmin)

```

apps/wells/models.py

```

import decimal
from datetime import timedelta
from django.utils import timezone

from django.contrib.auth import get_user_model
from django.db import models

```

```
from django.utils.translation import gettext_lazy as _
from apps.exist_devices.models import Device
```

```
class WellDevice(models.Model):
    device = models.OneToOneField(
        to=Device,
        on_delete=models.CASCADE,
        primary_key=True,
        verbose_name=_('Device')
    )

    name = models.CharField(
        max_length=100,
        verbose_name=_('Name')
    )

    user = models.ForeignKey(
        to=get_user_model(),
        on_delete=models.SET_NULL,
        related_name='welldevices',
        null=True,
        blank=True,
        verbose_name=_('Belong to')
    )

    phone_number = models.CharField(
        max_length=13,
        verbose_name=_('Phone number')
    )

    ministry_id = models.CharField(
        max_length=20,
        blank=True,
        verbose_name=_('Ministry ID')
    )

    permission_to_send = models.BooleanField(
        default=False,
        verbose_name=_('Permission to send data')
    )

    master = models.ForeignKey(
        to=get_user_model(),
        on_delete=models.SET_NULL,
        related_name='well_devices',
```

```

        null=True,
        blank=True,
        verbose_name=_('Master')
    )

    full_height = models.DecimalField(
        max_digits=6,
        decimal_places=2,
        default=0.0,
        verbose_name=_('Full height of groove (sm)')
    )

    height = models.DecimalField(
        max_digits=6,
        decimal_places=2,
        default=0.0,
        verbose_name=_('Height of upper part (sm)')
    )

    latitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Latitude of location'),
        blank=True,
        null=True
    )

    longitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Longitude of location'),
        blank=True,
        null=True
    )

    def __str__(self):
        return f'{self.name} - {self.device}'

    def save(self, *args, **kwargs):
        self.device.is_active = True
        self.device.save()
        super(WellDevice, self).save(*args, **kwargs)

    def delete(self, *args, **kwargs):
        self.device.is_active = False
        self.device.save()

```

```

        super(WellDevice, self).delete(*args, **kwargs)

class Meta:
    verbose_name = _('Well device')
    verbose_name_plural = _('Wells devices')

class WellDeviceMessage(models.Model):
    device = models.ForeignKey(
        to=WellDevice,
        on_delete=models.CASCADE,
        related_name='messages',
        verbose_name=_('Well device')
    )

    is_sent = models.BooleanField(
        default=False,
        verbose_name=_('Is sent')
    )

    h = models.DecimalField(
        max_digits=10,
        decimal_places=2,
        verbose_name=_('Water height (sm)')
    )

    mineral = models.DecimalField(
        max_digits=10,
        decimal_places=4,
        verbose_name=_('Mineral (gramm/liter)')
    )

    temperature = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        verbose_name=_('Temperature (Celsius)')
    )

    bat = models.DecimalField(
        max_digits=5,
        decimal_places=2,
        blank=True,
        null=True,
        verbose_name=_('Battery power (volt)')
    )

```

```

is_charging = models.BooleanField(
    blank=True,
    null=True,
    verbose_name=_('Is charging')
)

net = models.SmallIntegerField(
    blank=True,
    null=True,
    verbose_name=_('Network quality')
)

created_at = models.DateTimeField(
    auto_now_add=True,
    verbose_name=_('Added time')
)

def __str__(self):
    return self.device.name

def get_device_active(self):
    return timezone.now() - self.created_at < timedelta(days=2)

def get_view_height(self):
    h = self.device.full_height - \
        self.device.height - decimal.Decimal(self.h)
    return h if h > 0 else 0.0

def get_mineral(self):
    mineral_int = int(self.mineral)
    if self.mineral - mineral_int == 0:
        return mineral_int
    return round(self.mineral, 2)

def save(self, *args, **kwargs):
    if decimal.Decimal(self.h) < 0:
        self.h = decimal.Decimal('0.01')
    super(WellDeviceMessage, self).save(*args, **kwargs)

class Meta:
    verbose_name = _('Well message')
    verbose_name_plural = _('Wells messages')

class WellMovement(models.Model):
    device = models.ForeignKey(

```

```

        to=WellDevice,
        on_delete=models.CASCADE,
        verbose_name=_('Device')
    )

    latitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Latitude of location'),
        blank=True,
        null=True
    )

    longitude = models.DecimalField(
        max_digits=9,
        decimal_places=6,
        verbose_name=_('Longitude of location'),
        blank=True,
        null=True
    )

    created_at = models.DateTimeField(
        auto_now_add=True,
        verbose_name=_('Added time')
    )

    def __str__(self):
        return self.device.name

    class Meta:
        verbose_name = _('Well movement message')
        verbose_name_plural = _('Wells movement messages')

```

## apps/wells/views.py

```

from django.contrib.auth.decorators import login_required
from django.shortcuts import render, redirect

from .models import WellDevice, WellDeviceMessage
from .forms import WellDeviceForm, WellDeviceEditForm

@login_required(login_url='login')
def home_page(request):

```



```

devices = sorted(
    WellDevice.objects.filter(master=request.user),
    key=lambda welldevice: welldevice.device.id[-4:],
    reverse=True
)
context = {
    'well_devices': [
        {
            'device': device,
            'device_last_message':
WellDeviceMessage.objects.filter(device=device).last()
        } for device in devices
    ]
}
return render(request, 'wells/well_devices_list.html', context)

@login_required(login_url='login')
def add_new_device(request):
    form = WellDeviceForm()
    if request.method == 'POST':

        device_id = request.POST.get('device')
        device = WellDevice.objects.filter(device_id=device_id).first()
        if device is not None:
            device.master = request.user
            device.name = request.POST.get('name')
            device.phone_number = request.POST.get('phone_number')
            device.full_height = request.POST.get('full_height')
            device.height = request.POST.get('height')
            device.save()
            return redirect('wells_dashboard')

        form = WellDeviceForm(data=request.POST)
        if form.is_valid():
            form.save(master=request.user)
            return redirect('wells_dashboard')
    context = {
        'title': 'Add well device',
        'form': form
    }
    return render(request, 'wells/add_or_edit_device.html', context)

@login_required(login_url='login')
def device_detail(request, device_id):
    selected_device = WellDevice.objects.filter(

```

```

        master=request.user).filter(device_id=device_id).first()
if selected_device is None:
    return render(request, 'home/404.html')
context = {
    'selected_device': selected_device,
    'latitude': str(selected_device.latitude),
    'longitude': str(selected_device.longitude),
    'selected_device_messages':
        WellDeviceMessage.objects.filter(
            device_id=device_id).order_by('-created_at')[:20]
}
return render(request, 'wells/well_device_detail.html', context)

@login_required(login_url='login')
def edit_device(request, device_id):
    selected_device = WellDevice.objects.filter(
        master=request.user).filter(device_id=device_id).first()
    if selected_device is not None:
        if request.method == 'POST':
            form = WellDeviceEditForm(
                data=request.POST, instance=selected_device)
            if form.is_valid():
                form.save()
                return redirect('well_device_detail', selected_device.device_id)
            form = WellDeviceEditForm(instance=selected_device)
        else:
            form = WellDeviceEditForm()
    context = {
        'device_id': selected_device.device_id,
        'title': 'Edit well device',
        'form': form
    }
    return render(request, 'wells/add_or_edit_device.html', context)

@login_required(login_url='login')
def delete_device(request, device_id):
    selected_device = WellDevice.objects.filter(
        master=request.user).filter(device_id=device_id).first()
    if selected_device is not None:
        selected_device.master = None
        selected_device.save()
    return redirect('wells_dashboard')

```

## apps/wells/forms.py

```
from django import forms
from django.forms import ModelForm
from .models import WellDevice
from apps.exist_devices.models import Device

class WellDeviceForm(ModelForm):
    # change the widget of the device field to text input
    device = forms.ModelChoiceField(
        queryset=Device.objects.filter(is_active=False).filter(type='well'),
        widget=forms.TextInput(attrs={'autocomplete': 'off'}),
        error_messages={
            'required': 'Device id bo\'lishi shart',
            'invalid_choice': 'Noto\'g\'ri device id kiritildi'
        }
    )

    class Meta:
        model = WellDevice
        fields = ('device', 'name', 'phone_number', 'full_height', 'height')

    def __init__(self, *args, **kwargs):
        super(WellDeviceForm, self).__init__(*args, **kwargs)

        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'w-full mt-2 mb-5 p-1 border-gray-900 rounded-md ' \
                                                    'focus:border-indigo-600 focus:ring focus:ring-opacity-40' \
                                                    'focus:ring-indigo-500 border-2 border-black border-slate-500'

    def save(self, master=None, commit=True):
        obj = super(WellDeviceForm, self).save(commit=False)
        if master:
            obj.master = master
        if commit:
            obj.save()
        return obj

class WellDeviceEditForm(ModelForm):
    class Meta:
        model = WellDevice
        fields = (
```

```

        'name', 'phone_number', 'full_height', 'height',
        'ministry_id', 'permission_to_send', 'latitude', 'longitude'
    )

    def __init__(self, *args, **kwargs):
        super(WellDeviceEditForm, self).__init__(*args, **kwargs)

        for visible in self.visible_fields():
            visible.field.widget.attrs['class'] = 'w-full mt-2 mb-5 p-1 border-gray-
900 rounded-md ' \
                                                    'focus:border-indigo-600 focus:ring
focus:ring-opacity-40' \
                                                    'focus:ring-indigo-500 border-2
border-black border-slate-500'

```

## apps/wells/admin.py

```

from django.contrib import admin
from .models import WellDevice, WellDeviceMessage, WellMovement
from ..exist_devices.models import Device
from django.utils.translation import gettext_lazy as _

class WellDeviceAdmin(admin.ModelAdmin):
    list_display = ('name', 'device', 'user', 'master', 'phone_number', 'ministry_id',
                    'permission_to_send', 'get_region', 'get_city')
    search_fields = ('name', 'phone_number')

    list_filter = ('user__region', 'user')

    # Additional fields

    def get_region(self, obj):
        if obj.user:
            return obj.user.region
        return None

    def get_city(self, obj):
        if obj.user:
            return obj.user.city
        return None

    get_region.short_description = _('Region')
    get_city.short_description = _('City')

```

```

# Custom form

obj_id = None

def get_form(self, request, obj=None, **kwargs):
    if obj:
        self.obj_id = obj.device.id
    return super(WellDeviceAdmin, self).get_form(request, obj, **kwargs)

def formfield_for_foreignkey(self, db_field, request, **kwargs):
    if db_field.name == 'device':
        if self.obj_id:
            kwargs['queryset'] = Device.objects.filter(type='well')
            self.obj_id = None
        else:
            kwargs['queryset'] = Device.objects.filter(
                type='well').filter(is_active=False)
    return super(WellDeviceAdmin, self).formfield_for_foreignkey(db_field,
request, **kwargs)

class WellDeviceMessageAdmin(admin.ModelAdmin):
    list_display = ('device', 'is_sent', 'h', 'mineral',
                    'temperature', 'bat', 'is_charging', 'net', 'created_at')

class WellMovementAdmin(admin.ModelAdmin):
    list_display = ['device', 'latitude', 'longitude', 'created_at', 'id']
    list_filter = ['device']

admin.site.register(WellDevice, WellDeviceAdmin)
admin.site.register(WellDeviceMessage, WellDeviceMessageAdmin)
admin.site.register(WellMovement, WellMovementAdmin)

```