**Funksiyalar**

**Funksiya nima?**

Funksiya — bu ma’lum bir vazifani bajaradigan va kerak bo‘lganda qayta chaqiriladigan kod bloki.

Funksiyaning asosiy afzalliklari:

* Kodni tartibli qiladi.
* Bir xil kodni qayta-qayta yozish o‘rniga, uni chaqirib ishlatish mumkin.
* O‘qilishi va tushunilishi osonlashadi.

Funksiyaning umumiy ko’rinishi:

def **funksiya\_nomi**([parametrlar]):

    ko‘rsatmalar

Funksiya ta’rifi **def** ifodasi bilan boshlanadi, u funksiyaning nomidan, parametrlar yoziladigan qavslar majmuasidan va ikki nuqtadan iborat bo‘ladi. Qavs ichidagi parametrlar majburiy emas. Keyingi qatordan boshlab funksiyaning bajaradigan ko‘rsatmalar bloki yoziladi. Funksiyadagi barcha ko‘rsatmalar satr boshidan bo‘sh joy (otstup) bilan yoziladi.

Masalan:

def **say\_hello**():

**print**("Hello")

Funksiya **say\_hello** deb nomlanadi. U parametrga ega emas va faqat bitta ko‘rsatmadan iborat, ya’ni konsolga "Hello" satrini chiqaradi.

E’tibor bering, funksiyadagi ko‘rsatmalar satr boshidan bo‘sh joy (otstup) bilan yozilishi kerak. Masalan:

def **say\_hello**():

**print**("Hello")

**print**("Bye")

Bu yerda print("Bye") ko‘rsatmasi **say\_hello** funksiyasi tarkibida emas, chunki u satr boshidan bo‘sh joy (otstup) bilan yozilmagan. Odatda funksiya ta’rifidan keyin va funksiyaga kirmaydigan boshqa ko‘rsatmalardan oldin ikki bo‘sh qatordan foydalaniladi.

Funksiyani chaqirish uchun uning nomi yoziladi va qavs ichida barcha parametrlariga qiymatlar beriladi:

funksiya\_nomi([parametrlar])

Masalan, funksiyani aniqlab va chaqirib ko‘ramiz:

def **say\_hello**():    *# say\_hello funksiyasining ta’rifi:*

**print**("Hello")

**say\_hello**()         *# say\_hello funksiyasini chaqirish:*

**say\_hello**()

**say\_hello**()

Bu yerda say\_hello funksiyasi ketma-ket uch marta chaqiriladi. Natijada konsolda quyidagi chiqish hosil bo‘ladi:

Hello  
Hello  
Hello

E’tibor bering, funksiya avval aniqlanadi, so‘ngra chaqiriladi.

Agar funksiya faqat bitta ko‘rsatmaga ega bo‘lsa, uni funksiyaning ta’rifi bilan bir qatorda, bitta qatorda yozish mumkin:

def **say\_hello**(): **print**("Hello")

**say\_hello**()

Xuddi shu tarzda boshqa funksiyalarni ham aniqlash va chaqirish mumkin. Masalan, bir nechta funksiyalarni aniqlab, bajarib ko‘ramiz:

def **say\_hello**():

**print**("Hello")

def **say\_goodbye**():

**print**("Good Bye")

**say\_hello**()

**say\_goodbye**()

Konsoldagi chiqish:

Hello  
Good Bye

**Lokal funksiyalar**

Ba’zi funksiyalar boshqa funksiyalar ichida aniqlanishi mumkin — bunday ichki funksiyalarni lokal funksiyalar deb ham atashadi. Lokal funksiyalar faqat ular aniqlangan funksiya ichida ishlatiladi. Masalan:

def **print\_messages**():

*# lokal funksiyalarni aniqlash*

    def **say\_hello**(): **print**("Hello")

    def **say\_goodbye**(): **print**("Good Bye")

*# lokal funksiyalarni chaqirish*

**say\_hello**()

**say\_goodbye**()

*# print\_messages funksiyasini chaqirish*

**print\_messages**()

*#say\_hello() # print\_messages funksiyasidan tashqarida say\_hello funksiyasiga murojaat qilish mumkin emas*

Bu yerda say\_hello() va say\_goodbye() funksiyalari print\_messages() funksiyasi ichida aniqlangan, shuning uchun ular unga nisbatan lokal hisoblanadi. Demak, ular faqatgina print\_messages() funksiyasi ichida ishlatilishi mumkin.

**Dastur tuzilishi va *main* funksiyasi**

Dasturda ko‘plab funksiyalar aniqlanishi mumkin. Ularning barchasini tartibga solish usullaridan biri — maxsus funksiya qo‘shishdir (odatda **main** deb nomlanadi), va keyin shu funksiya ichida boshqa funksiyalar chaqiriladi:

def **main**():

**say\_hello**()

**say\_goodbye**()

def **say\_hello**():

**print**("Hello")

def **say\_goodbye**():

**print**("Good Bye")

*# main funksiyasini chaqirish*

**main**()

**📘 Python’da Parametrli va Parametrsiz Funksiyalar**

**🔹 1. Parametrsiz funksiya**

Parametrsiz funksiya tashqaridan qiymat olmaydi. Faqat ichidagi ko‘rsatmalarni bajaradi.

**Misol 1:**

def **say\_hello**():

**print**("Salom, xush kelibsiz!")

*# Funksiyani chaqirish*

**say\_hello**()

**say\_hello**()

📌 **Natija:**

Salom, xush kelibsiz!  
Salom, xush kelibsiz!

➡ Bu funksiya faqat bitta xabar chiqaradi, tashqaridan qiymat olmaydi.

**🔹 2. Parametrli funksiya**

Parametrli funksiya tashqaridan qiymat qabul qiladi va uni ishlatadi.

**Misol 2:**

def **greet**(name):

**print**("Salom,", name, "!")

*# Funksiyani parametr bilan chaqirish*

**greet**("Ali")

**greet**("Laylo")

📌 **Natija:**

Salom, Ali !  
Salom, Laylo !

➡ Bu yerda name — parametr, funksiya chaqirilganda unga qiymat beriladi.

**🔹 3. Bir nechta parametrli funksiya**

Funksiyaga birdan ortiq parametr ham berish mumkin.

**Misol 3:**

def **add\_numbers**(a, b):

**print**("Yig‘indi:", a + b)

**add\_numbers**(5, 7)

**add\_numbers**(10, 25)

📌 **Natija:**

Yig‘indi: 12  
Yig‘indi: 35

➡ Bu yerda a va b parametrlari qabul qilinib, yig‘indisi chiqariladi.

**🔹 4. Standart qiymatga ega parametr**

Parametrlarga oldindan **standart qiymat** berish mumkin. Agar parametr qiymati ko‘rsatilmasa, shu standart qiymat ishlatiladi.

**Misol 4:**

def **greet**(name="Mehmon"):

**print**("Salom,", name, "!")

**greet**()          *# parametrsiz*

**greet**("Dilnoza") *# parametr bilan*

📌 **Natija:**

Salom, Mehmon !  
Salom, Dilnoza !

➡ Agar greet() chaqirilganda ism berilmasa, u holda Mehmon ishlatiladi.

**🔹 5. Xulosa**

* **Parametrsiz funksiya** → tashqaridan qiymat olmaydi.
* **Parametrli funksiya** → tashqaridan qiymat oladi va undan foydalanadi.
* **Bir nechta parametr** → bir nechta qiymat berib ishlatish mumkin.
* **Standart parametr qiymati** → parametr berilmasa, avtomatik qiymat ishlatiladi.

### 📘 Python’da ****Majburiy va Majburiy bo‘lmagan parametrlar****

Python’da funksiyalar parametrlar orqali tashqaridan qiymat qabul qiladi. Parametrlar **majburiy** va **majburiy bo‘lmagan (default)** turlarga bo‘linadi.

## 🔹 1. Majburiy parametrlar

Majburiy parametr — bu funksiyani chaqirganda **albatta qiymat berilishi kerak bo‘lgan parametr**. Agar qiymat berilmasa, xatolik yuz beradi.

### Misol:

def **greet**(name):

**print**("Salom,", name, "!")

**greet**("Ali")      *#* ✅ *to‘g‘ri ishlaydi*

**greet**()           *#* ❌ *TypeError: greet() missing 1 required positional argument*

📌 name — **majburiy parametr**, shuning uchun greet() chaqirilganda albatta qiymat berish kerak.

## 🔹 2. Majburiy bo‘lmagan parametrlar (default values)

Majburiy bo‘lmagan parametr — bu parametrga **standart qiymat** beriladi. Agar chaqirishda qiymat berilmasa, standart qiymat ishlatiladi.

### Misol:

def **greet**(name="Mehmon"):

**print**("Salom,", name, "!")

**greet**("Dilnoza")   *# → Salom, Dilnoza !*

**greet**()            *# → Salom, Mehmon !*

📌 Bu yerda name parametriga **default qiymat** sifatida "Mehmon" berilgan. Agar foydalanuvchi qiymat bermasa, avtomatik shu ishlatiladi.

## 🔹 3. Majburiy va majburiy bo‘lmagan parametrlarni birga ishlatish

Funksiya bir vaqtning o‘zida majburiy va majburiy bo‘lmagan parametrlarni qabul qilishi mumkin.

⚠️ Qoidaga ko‘ra:

* Avval **majburiy parametrlar**,
* Keyin **majburiy bo‘lmagan parametrlar** yoziladi.

### Misol:

def **introduce**(name, age=18):

**print**("Ism:", name)

**print**("Yosh:", age)

**introduce**("Ali", 25)   *# → Ism: Ali, Yosh: 25*

**introduce**("Laylo")     *# → Ism: Laylo, Yosh: 18*

📌 Bu yerda name — majburiy parametr, age esa majburiy bo‘lmagan parametr.

## 🔹 4. Bir nechta majburiy bo‘lmagan parametr

Bir nechta majburiy bo‘lmagan parametr yozish ham mumkin.

### Misol:

def **order**(drink="Choy", snack="Somsa"):

**print**("Ichimlik:", drink)

**print**("Tamaddi:", snack)

**order**("Sharbat", "Pitsa")

*# → Ichimlik: Sharbat, Tamaddi: Pitsa*

**order**("Kofe")

*# → Ichimlik: Kofe, Tamaddi: Somsa*

**order**()

*# → Ichimlik: Choy, Tamaddi: Somsa*

📌 Parametr berilmasa, avtomatik standart qiymatlar ishlatiladi.

Quyida bir nechta **majburiy** va **majburiy bo‘lmagan (default qiymatli)** parametrlarga ega funksiyaga misol keltirilgan:

def **student\_info**(name, age, group="No group", scholarship=False):

**print**(f"Ism: {name}")

**print**(f"Yosh: {age}")

**print**(f"Guruh: {group}")

    if scholarship:

**print**("Talaba stipendiya oladi.")

    else:

**print**("Talaba stipendiya olmaydi.")

*# Funksiyani chaqirish*

**student\_info**("Ali", 20, "CS-101", True)   *# Barcha parametrlar bilan*

**print**("------------")

**student\_info**("Dilnoza", 19)               *# Faqat majburiy parametrlar bilan*

Konsol chiqishi:

Ism: Ali  
Yosh: 20  
Guruh: CS-101  
Talaba stipendiya oladi.  
------------  
Ism: Dilnoza  
Yosh: 19  
Guruh: No group  
Talaba stipendiya olmaydi.

## 🔹 5. Xulosa

* **Majburiy parametr** → qiymat berilmasa, xatolik chiqadi.
* **Majburiy bo‘lmagan parametr** → qiymat berilmasa, standart qiymat ishlatiladi.
* Parametrlar tartibi muhim: avval majburiy, keyin majburiy bo‘lmagan yoziladi.

# **📌 Python’da Anonim va Lambda funksiyalar**

## 1. Oddiy (nomlangan) funksiya

Python’da odatda funksiya def kalit so‘zi bilan aniqlanadi va **nomga ega bo‘ladi**:

def **kvadrat**(x):

    return x \* x

**print**(**kvadrat**(5))  *# 25*

👉 Bu yerda kvadrat — **nomlangan funksiya**.

## 2. Anonim funksiya nima?

**Anonim funksiya** — nomga ega bo‘lmagan, faqat **bir marta ishlatish uchun** yoziladigan funksiya.  
Python’da **anonim funksiyalarni yaratish uchun faqat lambda ishlatiladi**.

## 3. Lambda funksiya

**lambda** kalit so‘zi yordamida yoziladi:

**Sintaksis:**

lambda argumentlar: ifoda

**Misol:**

f = lambda x: x \* x

**print**(f(5))  *# 25*

👉 Bu oddiy kvadrat(x) funksiyasining qisqa yozilishi.

## 4. Lambda funksiyaga misollar

### 🔹 Bir nechta argument bilan

add = lambda a, b: a + b

**print**(add(3, 7))  *# 10*

### 🔹 Shartli ifoda bilan

is\_even = lambda x: "Juft" if x % 2 == 0 else "Toq"

**print**(is\_even(4))  *# Juft*

**print**(is\_even(7))  *# Toq*

### 🔹 map bilan ishlatish

sonlar = [1, 2, 3, 4, 5]

kvadratlar = **list**(**map**(lambda x: x\*\*2, sonlar))

**print**(kvadratlar)  *# [1, 4, 9, 16, 25]*

### 🔹 filter bilan ishlatish

sonlar = [10, 15, 20, 25, 30]

juftlar = **list**(**filter**(lambda x: x % 2 == 0, sonlar))

**print**(juftlar)  *# [10, 20, 30]*

### 🔹 sorted bilan ishlatish

mevalar = ["olma", "banan", "anjir", "gilos"]

tartib = **sorted**(mevalar, key=lambda x: **len**(x))

**print**(tartib)  *# ['olma', 'anjir', 'gilos', 'banan']*

## 5. Lambda va oddiy funksiyaning taqqoslanishi

*# Oddiy funksiya*

def **kvadrat**(x):

    return x \* x

*# Lambda funksiya*

kvadrat\_lambda = lambda x: x \* x

**print**(**kvadrat**(6))        *# 36*

**print**(kvadrat\_lambda(6)) *# 36*

👉 Har ikkisi ham natija bir xil, faqat lambda — qisqa yozilishi.

## 6. Muhim farqlar

1. **Oddiy funksiya (def)**
   * Ko‘p qatorli bo‘lishi mumkin.
   * Murakkab logikalarni yozish mumkin.
   * Nom beriladi va qayta chaqirish mumkin.
2. **Lambda funksiya**
   * Har doim **bitta ifoda** (return yo‘q, o‘zi avtomatik qaytaradi).
   * Asosan **bir martalik, qisqa operatsiyalar** uchun ishlatiladi.
   * Ko‘p hollarda map, filter, sorted bilan ishlatiladi.

## 7. Amaliy mashqlar

### Mashq 1

Raqamlar ro‘yxatidan faqat toq sonlarni filter va lambda yordamida ajratib oling.

**Kutilgan natija:**

[1, 3, 5, 7, 9]

### Mashq 2

Berilgan ro‘yxatdagi so‘zlarni uzunligiga qarab sorted va lambda bilan tartiblang.

**Kutilgan natija:**

['olma', 'anjir', 'banan', 'mandarin']

### Mashq 3

map va lambda yordamida har bir elementni 3 ga ko‘paytiring.

**Kutilgan natija:**

[3, 6, 9, 12]

### Mashq 4

Berilgan son lambda orqali **musbat, manfiy yoki nol** ekanini aniqlasin.

**Kutilgan natija:**

0 → "nol"  
5 → "musbat"  
-3 → "manfiy"

✅ Xulosa:

* Python’da **har bir lambda funksiya anonim hisoblanadi**.
* Ammo boshqa dasturlash tillarida anonim funksiyani faqat lambda emas, balki boshqa usullar bilan ham yozish mumkin.

# **📘 Pythonda Dekoratorlar**

## 1️. Dekorator nima?

Dekorator — bu **funksiyani qabul qilib, unga qo‘shimcha imkoniyat qo‘shuvchi** funksiya. U mavjud kodni o‘zgartirmasdan funksiyaga **yangi xatti-harakatlar qo‘shadi**.

👉 Ko‘pincha quyidagi joylarda ishlatiladi:

* **logging** (dastur ishini kuzatish),
* **authentication** (foydalanuvchi huquqlarini tekshirish),
* **memoization** (natijalarni keshlash),
* **retry** (xatolik bo‘lganda qayta urinish).

## 2️. Oddiy dekorator misoli

def **decorator**(func):

    def **wrapper**():

**print**("Before calling the function.")

        func()

**print**("After calling the function.")

    return **wrapper**

**@decorator**   *# qisqa yozuv: greet = decorator(greet)*

def **greet**():

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

**greet**()

📌 Natija:

Before calling the function.  
Hello, World!  
After calling the function.

✅ @decorator yozuvi greet = decorator(greet) degani.

## 3️. Parametrli dekorator

Dekoratorlar ko‘pincha argumentli funksiyalarga ishlatiladi. Buning uchun \*args va \*\*kwargs yoziladi.

def **decorator\_name**(func):

    def **wrapper**(\*args, \*\*kwargs):

**print**("Before execution")

        result = func(\*args, \*\*kwargs)

**print**("After execution")

        return result

    return **wrapper**

**@decorator\_name**

def **add**(a, b):

    return a + b

**print**(**add**(5, 3))

📌 Natija:

Before execution  
After execution  
8

## 4️. Funksiyalar — birinchi darajali obyektlar

Pythonda funksiyalar **obyekt sifatida** ishlaydi. Ularni:

* o‘zgaruvchiga berish,
* argument sifatida uzatish,
* qaytarish,
* list/dict ichida saqlash mumkin.

def **greet**(n):

    return f"Hello, {n}!"

say\_hi = **greet**

**print**(say\_hi("Alice"))

def **apply**(f, v):

    return f(v)

**print**(**apply**(say\_hi, "Bob"))

def **make\_mult**(f):

    def **mult**(x):

        return x \* f

    return **mult**

dbl = **make\_mult**(2)

**print**(dbl(5))

📌 Natija:

Hello, Alice!  
Hello, Bob!  
10

👉 Shu xususiyat tufayli **dekoratorlar** ishlaydi.

## 5️. Yuqori darajali funksiyalar (Higher-Order Functions)

**Higher-order function** — bu boshqa funksiyani qabul qiladigan yoki funksiyani qaytaradigan funksiya.  
Dekoratorlar — yuqori darajali funksiyalar turidir.

def **fun**(f, x):

    return f(x)

def **square**(x):

    return x \* x

**print**(**fun**(**square**, 5))

📌 Natija:

25

## 6️. Dekorator turlari

### 🔹 Funksiya dekoratorlari

def **simple\_decorator**(func):

    def **wrapper**():

**print**("Before function")

        func()

**print**("After function")

    return **wrapper**

**@simple\_decorator**

def **greet**():

**print**("Hello!")

**greet**()

### 🔹 Method dekoratorlari (klass ichidagi metodlar uchun)

def **method\_decorator**(func):

    def **wrapper**(self, \*args, \*\*kwargs):

**print**("Before method")

        res = func(self, \*args, \*\*kwargs)

**print**("After method")

        return res

    return **wrapper**

class **MyClass**:

**@method\_decorator**

    def **say\_hello**(self):

**print**("Hello!")

obj = **MyClass**()

obj.**say\_hello**()

### 🔹 Klass dekoratorlari

Klassning o‘zini o‘zgartiradi:

def **add\_class\_name**(cls):

    cls.class\_name = cls.\_\_name\_\_

    return cls

**@add\_class\_name**

class **Person**:

    pass

**print**(**Person**.class\_name)  *# Person*

## 7️. Pythondagi ****built-in**** dekoratorlar

### 🔹 @staticmethod

Klass obyektiga bog‘liq bo‘lmagan metod.

class **MathOperations**:

**@staticmethod**

    def **add**(x, y):

        return x + y

**print**(**MathOperations**.**add**(5, 3))

### 🔹 @classmethod

Klassning o‘zini (cls) qabul qiladi.

class **Employee**:

    raise\_amount = 1.05

**@classmethod**

    def **set\_raise\_amount**(cls, amount):

        cls.raise\_amount = amount

**Employee**.**set\_raise\_amount**(1.10)

**print**(**Employee**.raise\_amount)

### 🔹 @property

Metodni **obyekt atributi** sifatida ishlatish imkonini beradi.

class **Circle**:

    def **\_\_init\_\_**(self, radius):

        self.\_radius = radius

**@property**

    def **radius**(self):

        return self.\_radius

**@radius.setter**

    def **radius**(self, value):

        if value >= 0:

            self.\_radius = value

        else:

            raise **ValueError**("Radius cannot be negative")

**@property**

    def **area**(self):

        return 3.14159 \* (self.\_radius \*\* 2)

c = **Circle**(5)

**print**(c.radius)   *# 5*

**print**(c.area)     *# 78.53975*

c.radius = 10

**print**(c.area)     *# 314.159*

## 8️. Bir nechta dekoratorni zanjir qilib ishlatish

Dekoratorlar ketma-ket o‘rab ishlashi mumkin:

def **decor1**(func):

    def **inner**():

        return func() \*\* 2

    return **inner**

def **decor**(func):

    def **inner**():

        return func() \* 2

    return **inner**

**@decor1**

**@decor**

def **num**():

    return 10

**@decor**

**@decor1**

def **num2**():

    return 10

**print**(**num**())   *# 400*

**print**(**num2**())  *# 200*

## 9️. Amaliy qo‘llanishlar

Dekoratorlar real loyihalarda:

* 📌 **Logging** → funksiyalarning chaqirilishini yozib borish.
* 📌 **Authentication** → foydalanuvchi huquqlarini tekshirish.
* 📌 **Caching (functools.lru\_cache)** → hisoblangan natijalarni saqlash.
* 📌 **Retry logic** → xatolikda qayta urinish.

✅ **Xulosa:**

* Dekorator — funksiyaga qo‘shimcha imkoniyat qo‘shuvchi vosita.
* Ular **funksiya, metod va klass** uchun yozilishi mumkin.
* Pythonning o‘zida ham @staticmethod, @classmethod, @property kabi dekoratorlar mavjud.
* Real hayotda logging, autentifikatsiya, caching kabi vazifalarda keng ishlatiladi.

# **📘 Pythonda Global va Lokal o‘zgaruvchilar**

Python dasturlash tilida **o‘zgaruvchining ko‘lami (scope)** muhim tushuncha hisoblanadi. O‘zgaruvchilar **qayerda e’lon qilinganiga qarab** global yoki lokal bo‘ladi.

## 🔹 Lokal o‘zgaruvchi (Local Variable)

* **Funksiya ichida** e’lon qilingan o‘zgaruvchi **faqat o‘sha funksiya davomida mavjud bo‘ladi**.
* Funksiya bajarilishi tugagach, o‘zgaruvchi yo‘qoladi.
* Funksiya tashqarisida foydalanib bo‘lmaydi.

### Misol 1: Lokal o‘zgaruvchidan foydalanish

def **greet**():

    msg = "Hello from inside the function!"

**print**(msg)

**greet**()

📌 Natija:

Hello *from* inside the function!

➡ msg faqat greet() funksiyasi ichida mavjud.

### Misol 2: Lokal o‘zgaruvchini tashqarida ishlatish xatosi

def **greet**():

    msg = "Hello!"

**print**("Inside function:", msg)

**greet**()

**print**("Outside function:", msg)  *#* ❌ *xato*

📌 Natija:

NameError: name 'msg' *is not* defined

➡ Chunki msg lokal o‘zgaruvchi bo‘lib, tashqarida mavjud emas.

## 🔹 Global o‘zgaruvchi (Global Variable)

* **Funksiyalardan tashqarida** e’lon qilingan o‘zgaruvchi global hisoblanadi.
* Uni dasturning istalgan joyida, jumladan funksiyalar ichida ham ishlatish mumkin.
* Butun dastur davomida mavjud bo‘ladi.

### Misol 1: Global o‘zgaruvchidan foydalanish

msg = "Python is awesome!"

def **display**():

**print**("Inside function:", msg)

**display**()

**print**("Outside function:", msg)

📌 Natija:

Inside function: Python *is* awesome!  
Outside function: Python *is* awesome!

### Misol 2: Global o‘zgaruvchi boshqa funksiya ichida

def **fun**():

**print**("Inside Function:", s)

s = "I love Python"

**fun**()

**print**("Outside Function:", s)

📌 Natija:

Inside Function: I love Python  
Outside Function: I love Python

## 🔹 Lokal o‘zgaruvchi globalni “soya qilishi”

Agar bir xil nomdagi o‘zgaruvchi **globalda ham, lokalda ham mavjud bo‘lsa** – funksiya ichida faqat lokal ishlatiladi.

def **fun**():

    s = "Me too."

**print**(s)

s = "I love Python"

**fun**()

**print**(s)

📌 Natija:

Me too.  
I love Python

➡ fun() ichida s lokal bo‘lgani uchun globalni “soya qiladi”.

## 🔹 Funksiya ichida globalni o‘zgartirish

Funksiya ichida **global o‘zgaruvchini bevosita o‘zgartirish mumkin emas**. Buning uchun global kalit so‘zidan foydalaniladi.

### Xatolik:

def **fun**():

    s += " Python"

**print**(s)

s = "I love"

**fun**()

📌 Natija:

UnboundLocalError: local variable 's' referenced before assignment

### To‘g‘ri yo‘l: global bilan

def **fun**():

    global s

    s += " Python"

**print**("Inside function:", s)

s = "I love"

**fun**()

**print**("Outside function:", s)

📌 Natija:

Inside function: I love Python  
Outside function: I love Python

## 🔹 Yana bir misol

a = 1  *# Global o‘zgaruvchi*

def **f**():

**print**("f():", a)  *# Globaldan foydalanadi*

def **g**():

    a = 2  *# Lokal o‘zgaruvchi globalni soya qiladi*

**print**("g():", a)

def **h**():

    global a

    a = 3  *# Globalni o‘zgartiradi*

**print**("h():", a)

**print**("global:", a)

**f**()

**print**("global:", a)

**g**()

**print**("global:", a)

**h**()

**print**("global:", a)

📌 Natija:

*global*: 1  
f(): 1  
*global*: 1  
g(): 2  
*global*: 1  
h(): 3  
*global*: 3

## 🔹 Global va Lokal o‘zgaruvchilar farqi

| **Asos** | **Global o‘zgaruvchi** | **Lokal o‘zgaruvchi** |
| --- | --- | --- |
| **E’lon qilish joyi** | Funksiyalardan tashqarida | Funksiya ichida |
| **Yashash muddati** | Dastur boshlanganidan tugaguncha | Funksiya ishlagan vaqt ichida |
| **Ko‘lami (scope)** | Dastur bo‘ylab hamma joyda | Faqat funksiya ichida |
| **Ma’lumot almashish** | Hamma funksiyalar bilan | Faqat funksiya ichida |
| **Parametr kerakmi?** | Yo‘q | Ko‘pincha kerak bo‘ladi |
| **Ta’sir doirasi** | O‘zgarishi butun dasturga ta’sir qiladi | Faqat o‘z funksiyasiga ta’sir qiladi |

✅ **Xulosa:**

* **Lokal o‘zgaruvchi** → faqat funksiya ichida mavjud.
* **Global o‘zgaruvchi** → butun dastur davomida mavjud.
* Agar bir xil nom ishlatilsa → lokal globalni soya qiladi.
* Funksiya ichida globalni o‘zgartirish uchun global kalit so‘zi ishlatiladi.