**Foydalanuvchi funksiyalari**

**Python’da foydalanuvchi funksiyalari bilan ishlash asoslari**

Python’da standart funksiyalar va metodlardan foydalanishdan tashqari, o‘z funksiyalarimizni ham yaratib, ulardan foydalanishimiz mumkin.

Keling, funksiya yaratish sintaksisini ko‘rib chiqamiz. Funksiya e’lon qilish uchun def kalit so‘zi yoziladi, so‘ngra uning nomi va qavslar beriladi (ular ichida parametrlar yozilishi mumkin). Keyin ikki nuqta : qo‘yiladi va undan keyin pastki blokda kerakli kod (funksiya tanasi) yoziladi. Funksiya tanasi boshqa kod bloklari kabi bo‘sh joy (otstup) bilan yozilishi kerak.

Masalan, func nomli funksiya yaratamiz:

def **func**():

    '''

        Funksiya tanasi,

        unda bajariladigan

        kod yoziladi

    '''

Endi func funksiyasi ! belgisini chiqarsin:

def **func**():

**print**('!')

Funksiyani chaqirish uchun uning nomini va qavslarini yozish kifoya:

def **func**():

**print**('!')

*# Funksiyamizni chaqiramiz:*

func()  *# ekranga '!' chiqaradi*

Funksiyani bir necha marta chaqirish mumkin – bunda har safar u qaytadan ishlaydi:

def **func**():

**print**('!')

**func**()  *# ekranga '!' chiqaradi*

**func**()  *# ekranga '!' chiqaradi*

**func**()  *# ekranga '!' chiqaradi*

Muhim: Python’da funksiyani **faqat uning e’lon qilingan joyidan keyin** chaqirish mumkin:

func()  *# xato beradi*

def **func**():

**print**('!')

### Python’da funksiyalar parametrlari

Funksiyaning qavs ichiga **parametrlar** berish mumkin. Keling, misolda ko‘rib chiqamiz.

Ikki sonni parametr sifatida qabul qilib, ularning yig‘indisini ekranga chiqaradigan funksiya yozamiz. Buning uchun qavs ichiga vergul orqali o‘zgaruvchilar nomlarini yozamiz. Bu o‘zgaruvchilar nomi ixtiyoriy bo‘lishi mumkin. Masalan, num1 va num2 deb nomlaymiz. So‘ngra ularning yig‘indisini hisoblab chiqaramiz:

def **func**(num1, num2):

**print**(num1 + num2)

Endi funksiyamizni chaqirib, unga parametr sifatida sonlar beramiz:

func(2, 3)  *# ekranga 5 chiqaradi*

### Python’da return operatori

Keling, oddiy misoldan boshlaymiz. Quyidagi funksiya berilgan sonning kvadratini **ekranga chiqaradi**:

def **func**(num):

**print**(num \*\* 2)

Ammo ba’zida natijani darhol ekranga chiqarish kerak emas, balki uni oldin **o‘zgaruvchiga saqlash** kerak bo‘ladi. Masalan:

res = func(3)  *# res o‘zgaruvchisida 9 bo‘lishi kerak*

Buning uchun Python’da maxsus return operatori mavjud. U funksiya qanday qiymatni **qaytarishini** ko‘rsatadi. "Qaytaradi" deganda, funksiyani chaqirganda natija o‘zgaruvchiga yozilishi nazarda tutiladi.

Funksiyamizni return bilan yozamiz:

def **func**(num):

    return num \*\* 2

Endi natijani o‘zgaruvchiga yozib olishimiz mumkin:

res = func(3)

Keyin uni ekranga chiqaramiz:

res = func(3)

**print**(res)  *# ekranga 9 chiqaradi*

Natijani oldin o‘zgartirib, keyin chiqarish ham mumkin:

res = func(3)

res = res + 1

**print**(res)  *# ekranga 10 chiqaradi*

Yoki to‘g‘ridan-to‘g‘ri amal bajarib yozish mumkin:

res = func(3) + 1

**print**(res)  *# ekranga 10 chiqaradi*

Funksiyani bir necha marta chaqirib, ularning natijasini qo‘shish ham mumkin:

res = func(2) + func(3)

**print**(res)  *# ekranga 13 chiqaradi*

Qisqa yo‘li: funksiyani bevosita print ichida chaqirish:

**print**(func(3))  *# ekranga 9 chiqaradi*

### Python’da ixtiyoriy (majburiy bo‘lmagan) parametrlar

Funksiya parametrlari **majburiy bo‘lmasligi** ham mumkin. Buning uchun ularga **standart qiymat** (default value) beriladi.

Keling, misol ko‘ramiz. Oddiy funksiya yozamiz:

def **func**(num1, num2):

    return num1 + num2

**print**(**func**(1, 2))  *# ekranga 3 chiqaradi*

Endi num2 parametriga **standart qiymat** sifatida 2 beramiz:

def **func**(num1, num2=2):

    return num1 + num2

Endi faqat bitta parametr berib ham chaqirish mumkin:

**print**(func(1))  *# ekranga 3 chiqaradi*

Shunga qaramay, xohlasak, **ikkinchi parametrni ham o‘zimiz berishimiz** mumkin:

**print**(func(2, 4))  *# ekranga 6 chiqaradi*

### Python’da nomlangan parametrlar

Agar funksiyaga ko‘p parametr berilsa, ularni chalkashtirib yuborish ehtimoli ortadi. Shu muammoni hal qilish uchun Python’da **nomlangan parametrlar** mavjud.

Keling, oddiy funksiya bilan boshlaymiz:

def **func**(num1, num2):

    return num1 + num2

Uni odatdagidek chaqirsak:

res = func(5, 10)

**print**(res)  *# ekranga 15 chiqaradi*

Nomlarini ko‘rsatib chaqirish ham mumkin:

res = func(num1=5, num2=10)

**print**(res)  *# ekranga 15 chiqaradi*

Nomlangan parametrlarning qulay tomoni – **ularni tartibsiz ham yozish mumkin**:

res = func(num2=2, num1=5)

**print**(res)  *# ekranga 3 chiqaradi*

Agar funksiyada bir nechta **ixtiyoriy parametr** (standart qiymatli parametr) bo‘lsa, ulardan ham nomlangan parametr sifatida foydalanish mumkin. Bunda ularni e’lon qilingan tartibda yozish shart emas:

def **func**(num1, num2=1, num3=2):

    return num1 + num2 + num3

res = **func**(num1=2, num3=6)

**print**(res)  *# ekranga 8 chiqaradi*

### Python’da parametr-o‘zgaruvchilar

Funksiya parametri sifatida faqat son emas, **o‘zgaruvchi** ham berish mumkin.

Masalan, oddiy kvadrat hisoblovchi funksiya yozamiz:

def **func**(num):

    return num \*\* 2

Uni son bilan chaqirsak:

**print**(func(2))  *# ekranga 4 chiqaradi*

Ammo parametr sifatida **o‘zgaruvchi** ham berishimiz mumkin:

tst = 3

print(func(tst)) # ekranga 9 chiqaradi

### Python’da ichma-ich funksiyalar (nested functions)

Kod hajmini qisqartirish uchun funksiyalarni **bir-biriga ichma-ich chaqirish** mumkin.

Masalan, bizda sonni kvadratga oshiruvchi, kubga oshiruvchi va ikkita sonni qo‘shuvchi funksiyalar bo‘lsin:

def **square**(num):

    return num \*\* 2

def **cube**(num):

    return num \*\* 3

def **add**(num1, num2):

    return num1 + num2

Endi sonning kvadrati va kubining yig‘indisini olish uchun funksiyalarni **ichma-ich chaqiramiz**:

res = add(square(2), cube(4))

**print**(res)  *# ekranga 68 chiqaradi*

Bundan tashqari, funksiyalarni boshqa funksiya ichida ham chaqirish mumkin. Masalan, yuqoridagi misolni add funksiyasi ichida yozamiz:

def **add**(num1, num2):

    return square(num1) + cube(num2)

res = **add**(2, 4)

**print**(res)  *# ekranga 68 chiqaradi*

### Python’da pass kalit so‘zi funksiyada

Ba’zi hollarda biz faqat **funksiya shablonini** yozib qo‘yishni xohlaymiz, lekin uning tanasiga kod hali tayyor emas. Agar funksiya tanasini bo‘sh qoldirsak, Python darhol **xato beradi**:

def **func**(test):  *# xato beradi*

Bunday vaziyatda pass kalit so‘zidan foydalaniladi. U **bo‘sh joy to‘ldiruvchi** sifatida ishlaydi va kod yozilmagani uchun xato chiqmaydi:

def **func**(test):

    pass

Shunday qilib, pass funksiyani keyinchalik yozish uchun **shablon** sifatida qoldirishga yordam beradi.

### Python’da funksiyalarga izoh (kommentariya) yozish

Agar funksiya nima qilishini qisqa tushuntirish kerak bo‘lsa, izohni uning tanasining yuqori qatorida yozish mumkin. Buning uchun uchta ' yoki " qo‘shtirnoq ishlatiladi.

Masalan:

def **func**(num1, num2):

    '''Funksiya ikkita sonni qo‘shadi'''

    return num1 + num2

Bunday izohlar **docstring** deb ataladi va ular keyinchalik hujjatlashtirish yoki yordam (help(func)) chaqirilganda ham chiqadi.

**Python’da funksiyalar yaratishda maslahatlar**

* **Funksiya nomlari fe’l bo‘lishi kerak.**
* **Funksiya nomi** uning nima qilishini aniq ko‘rsatishi kerak.
* Funksiya faqat **nomida aytilgan ishni** bajarishi kerak, boshqa narsani emas.
* Har bir funksiya **bitta amalni** bajarishi maqsadga muvofiq.
* Katta funksiyalar ichida **yordamchi funksiyalardan** foydalaning.
* Funksiya kodi odatda **10–15 qator**dan uzun bo‘lmasligi kerak.
* Agar funksiya juda uzun bo‘lsa, uni bir nechta yordamchi funksiyalarga bo‘lib chiqing.
* Funksiya nomlarida keng tarqalgan prefikslardan foydalaning: show, get, set, calc, create, change, check.
* Agar funksiya nomi ikki yoki undan ortiq so‘zdan iborat bo‘lsa, ularni pastki chiziq (\_) bilan ajrating, masalan: get\_sum.
* **Takrorlanuvchi kodni** alohida funksiyaga ajratib chiqing.