ISSN 3060-4567

FUNKSIYANI BOSHQA FUNKSIYANING TURI, PARAMETRI VA NATIJASI SIFATIDA ISHLATILISHI

Tojimamatov Israil Nurmamatovich
Farg'ona davlat universiteti amaliy matematika
va kafedrasi katta o'qituvchisi
E-mail: israiltojimamatov@gmail.com
Muhammadvaliyeva Mohichehra Zuhriddin qizi
Farg'ona davlat universiteti talabasi
E-mail: mohichehramuhammadvaliyeva@gmail.com

Annotatsiya: *Ushbu maqolada Python dasturlash tilida funksiyaning boshqa funksiyaga parametr, tur yoki natija sifatida uzatilishiga oid imkoniyatlar ko'rib chiqildi. Maqolada funksiyalarning birinchi darajali obyekt (first-class object) sifatida ishlatilishi nazariy jihatdan izohlangan hamda amaliy misollar orqali yoritib berilgan. Yuqori darajali funksiyalar (higher-order functions) yordamida kodni modullashtirish, qayta ishlatish va soddalashtirish usullari tahlil qilingan. Har bir funksional yondashuv uchun alohida kod namunasi va uning dasturiy foydasi ko'rsatib o'tilgan. Tadqiqot natijasida Python tilining funksional dasturlash paradigmasiga mosligi va real loyihalarda samarali qo'llanishi aniqlangan.*

Kalit so'zlar: *Python, funksiyalar, yuqori darajali funksiyalar, parametr sifatida funksiya, funksional dasturlash, birinchi darajali obyekt, dasturlash paradigmalar.*

Annotation: *This article explores the ability of Python programming language to use functions as parameters, types, and return values of other functions. The concept of functions as first-class objects is theoretically explained and illustrated with practical examples. The use of higher-order functions for modularization, code reuse, and simplification is analyzed in detail. Each functional approach is supported with relevant code snippets and explained in terms of its programming benefits. As a result, it has been shown that Python aligns well with the*

functional programming paradigm and can be effectively applied in real-world projects.

Keywords: *Python, Functions, Higher-order functions, Function as parameter, Functional programming, First-class object, Programming paradigms.*

Аннотация: *В данной статье рассматриваются возможности языка программирования Python по использованию функций в качестве параметров, типов и возвращаемых значений других функций. Теоретически объяснена концепция функций как объектов первого класса, а также представлены практические примеры. Проанализировано использование функций высшего порядка для модульности, повторного использования кода и его упрощения. Каждому подходу сопутствуют фрагменты кода с пояснениями о программной эффективности. В результате установлено, что Python хорошо соответствует парадигме функционального программирования и может эффективно применяться в реальных проектах.*

Ключевые слова: *Python, Функции, Функции высшего порядка, Функция как параметр, Функциональное программирование, Объекты первого класса, Парадигмы программирования.*

Kirish

Zamonaviy dasturlash tillarining rivojlanishi natijasida dasturiy yechimlarda modul va abstraksiya darajalarini oshirishga bo‘lgan ehtiyoj keskin ortdi. Ayniqsa, katta hajmdagi loyihalarda kodni qayta foydalanishga mos, o‘qilishi va test qilinishi oson tarzda tashkil qilish dasturchilar oldida turgan muhim vazifalardan biridir. Shu nuqtai nazardan qaralganda, funksiyalarning boshqa funksiyalar bilan o‘zaro ishlashi (ya’ni, funksiyani boshqa funksiyaning turi, parametri yoki natijasi sifatida ishlatish) zamonaviy dasturlash metodologiyasida markaziy o‘rin egallaydi.

Python dasturlash tili ushbu yondashuvni to‘liq qo‘llab-quvvatlovchi tillardan biri bo‘lib, unda funksiyalar birinchi darajali obyekt (first-class object) sifatida qaraladi. Bu shuni anglatadiki, funksiyalarni o‘zgaruvchiga biriktirish, boshqa funksiyaga argument sifatida uzatish, funksiyadan natija sifatida qaytarish yoki

massiv, lug‘at kabi tuzilmalarda saqlash imkoniyati mavjud. Bunday imkoniyatlar, ayniqsa, funksional dasturlash paradigmasida muhim rol o‘ynaydi.

Mazkur maqolada Python tilidagi funksiyalarni boshqa funksiyalar bilan birgalikda ishlatishning nazariy asoslari, amaliy ko‘rinishlari va dasturiy yechimlarga ta’siri yoritiladi. Funksiya ichida funksiya aniqlash, yuqori darajali funksiyalar (higher-order functions), lambda, map, filter, reduce, dekoratorlar (decorators) kabi tushunchalar orqali kodning modullashtirilgan, moslashuvchan va qayta foydalanishga yaroqli shakli shakllanishi tahlil qilinadi. Bunday yondashuv nafaqat kodni soddalashtiradi, balki murakkab muammolarga yondashishda ham samaradorlikni oshiradi.

Ushbu mavzuni chuqur o‘rganish, amaliy misollar orqali uning kuchli jihatlarini ochib berish va dasturiy loyihalarda to‘g‘ri qo‘llay olish — zamonaviy dasturchilar uchun zarur bo‘lgan bilim va ko‘nikmalardan biridir. Shu sababli, ushbu maqolada mavzu atroflicha yoritilib, real kod misollarida tahlil qilinadi.

Funksiyani parametrlarni o'rnatish

Python dasturlash tilida funksiyalar birinchi darajali obyektlar hisoblanadi. Bu degani, ular o‘zgaruvchi kabi saqlanishi, boshqa funksiyalarga parametr sifatida uzatilishi yoki natija sifatida qaytarilishi mumkin.

Dasturlashda funksiyaning asosiy kuchi — kiruvchi parametrlarga nisbatan natija qaytarish imkoniyatida mujassam. Python tilida funksiyalar parametrlar bilan juda moslashuvchan tarzda ishlay oladi, bu esa ularni yuqori darajali (abstrakt) va qayta foydalanishga yaroqli qiladi. Funksiya parametrlari bilan ishlashda quyidagi asosiy yondashuvlar ajralib turadi:

1. Oddiy parametrlar (Positional arguments)

Python funksiyalarida parametrlarni tartib asosida uzatish mumkin. Misol:

```
def greet(name):
    return f"Hello, {name}!"
print(greet("Salom"))
```

2. Nomi bilan uzatiladigan parametrlar (Keyword arguments)

Parametrlar nomi bilan birga uzatilganda aniqlik va tushunarlilik oshadi:

ISSN 3060-4567

```
def introduce(name, age):
    return f"My name is {name} and I'm {age} years old."
print(introduce(age=19, name="Mohichehra"))
```

Bu yondashuv oddiy va tezkor bo‘lsa-da, ba’zan parametrlar soni ko‘payganda chalkashlik keltirib chiqarishi mumkin.

3. Sukut bo‘yicha qiymatlar (Default parameter values)

Funksiya parametrlariga sukut (default) qiymatlar berish kodni moslashuvchan qiladi:

```
def power(base, exponent=2):
    return base ** exponent
print(power(5))      # 25
print(power(5, 3))   # 125
```

Bu yondashuv foydalanuvchiga ba’zi parametrlarni kiritmaslik imkoniyatini beradi.

4. Cheksiz parametrlar (Arbitrary arguments)

a) *args: Pozitsion parametrlarning to‘plami

Funksiya ixtiyoriy miqdordagi parametrlarni qabul qilishi mumkin:

```
def add_all(*numbers):
    return sum(numbers)
print(add_all(2, 4, 6, 8))  # 20
```

b) **kwargs: Kalit-qiymat juftligi shaklidagi parametrlar

```
def show_info(**data):
    for key, value in data.items():
        print(f"{key}: {value}")
show_info(name="Ali", field="Applied Math", age=19)
```

Funksiyalarni parametr sifatida uzatish esa yuqori darajadagi funksiyalar (higher-order functions) konsepsiyasiga asoslanadi. Bu yondashuv yordamida umumiy strukturaga ega funksiyalarga turli operatsiyalarni berish mumkin bo‘ladi.

5. Parametr sifatida funksiyani uzatish

ISSN 3060-4567

Python funksiyalarni birinchi darajali obyekt deb qabul qilgani uchun ularni parametr sifatida uzatish mumkin:

```
def operate(x, y, func):
    return func(x, y)
def multiply(a, b):
    return a * b
print(operate(4, 5, multiply))  # 20
```

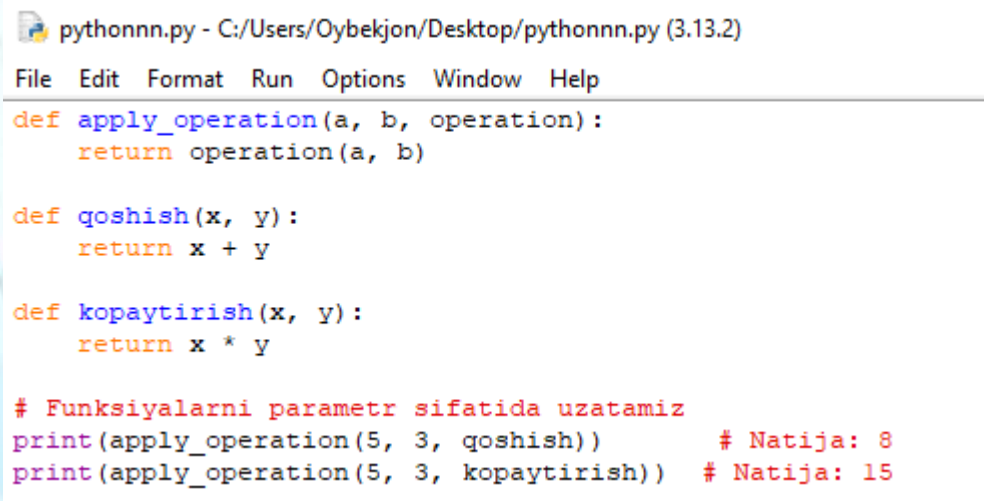
Bu metod yuqori darajali funksiyalarni yaratishda juda muhim.

Masalan, matematik amallar, satrlar bilan ishlash, filtrlash yoki o‘zgarishlarni qo‘llash kabi hollarda bir xil shakldagi asosiy funksiya yozilib, unga har xil xatti-harakatlarni amalga oshiruvchi funksiyalar parametr sifatida uzatiladi.

Bunday yondashuv:

- kodni qisqartiradi,
- modullilikni ta’minlaydi,
- qayta foydalanish imkonini oshiradi,
- kod o‘qilishini soddalashtiradi.

Python dasturida bu uslub ayniqsa map(), filter(), sorted() funksiyalari, yoki lambda iboralari bilan birgalikda keng qo‘llanadi.



ISSN 3060-4567

```
Python 3.13.2 (tags/v3.
AMD64)] on win32
Type "help", "copyright
>>>
================ RESTAR
8
15
>>>
```

Tahlil:

- apply_operation(a, b, operation)funksiyasi umumiy ko'rinishda nazorat va yana ikki_
- Uchinchisi — operationparametrik funksiya uzatilmoqda .
- addqiziqarlimultiplyapply_operationichida funksiya sifatida chaqirilmoqda .
- Bu korxona modullilik , qayta tiklash , xatolikni tiklash va erkiniluvchanlik kabi afzalliklarni beradi.

Funksiyani natija sifatida belgilash:

Python tilida funksiyalar nafaqat parametr sifatida uzatilishi, balki natija sifatida qaytarilishi ham mumkin. Bunday funksiyalarni yaratish orqali dasturchi dinamik tarzda boshqa funksiyalarni shakllantirishi va ularni istalgancha chaqira olishi mumkin.

Bu yondashuv ham yuqori darajadagi funksiyalar tushunchasiga kiradi. Funksiyani natija sifatida qaytarish quyidagi hollarda juda foydali:

- Shablon (template) funksiyalar yaratishda
- Fabrikalar (function factories) yasashda
- Yopiluvchilar (closures) yaratishda
- Kengaytiriluvchi tizimlar ishlab chiqishda

Bu usul yordamida kodlar dinamik, shablon asosida ishlovchi va modullashtirilgan bo‘ladi. Ayniqsa, funktsional dasturlash tamoyillarida keng qo‘llaniladi.

Endi amalda sinab ko‘ramiz:

Dastur kodi:

ISSN 3060-4567

```
File  Edit  Format  Run  Options  Window  Help
def power_function(n):
    def power(x):
        return x ** n
    return power

# power_function funksiyasi boshqa funksiyani qaytaryapti
square = power_function(2)  # Endi square(x) = x^2
cube = power_function(3)    # Endi cube(x) = x^3

print(square(4))  # Natija: 16
print(cube(2))    # Natija: 8
```

Natija:

```
========== RESTART: C:/Users/(
16
8
>>
```

Yuqoridagi dasturimizni tahlil qilamiz. Bu yerda:

- `power_function(n)` — bu asosiy funksiya bo'lib, u ichida `power(x)` nomli ichki funksiya mavjud.
- `power_function`chaqirilganda, u `power`funksiyasini **natija sifatida** qaytarmoqda.
- `square = power_function(2)` orqali biz `square(x)` degan yangi funksiya hosil qildik, u har doim x ni kvadratga oshirad_
- Bu misolda **closure (yopiluvchi)** hodisasi yuz beradi: ya'ni ichki funksiya funktsiyasining `n`parametrini "eslab" qoladi.

Funksiyani tur (turi) sifatida hisobga olinadi:

Python dasturlash tilida funksiyalar obyekt sifatida qaraladi. Shuning uchun har bir funksiya o‘zining tipi (type) ga ega. Bu holat orqali:

- funksiyani turini tekshirish,
- dinamik tahlil qilish,
- funksiyalarni ma’lum turdagi obyektlar sifatida qayta ishlash mumkin bo‘ladi.

ISSN 3060-4567

Python’da funksiyaning turi function bo‘lib, u types modulidagi FunctionType orqali yanada chuqurroq aniqlanishi mumkin. Bundan tashqari, callable() funksiyasi yordamida obyekt chaqirilishi mumkin bo‘lganligini tekshirish mumkin.

Bu yondashuv funksiyalar bilan ishlashda moslashuvchanlik, xavfsizlik, va avtomatik tahlil imkonini beradi. Ayniqsa, katta tizimlarda yoki dynamic dispatch, decorator va introspection bilan ishlaganda muhim ahamiyatga ega.

Amalda qo‘llanilishi:

Dastur kodi:

```
File  Edit  Format  Run  Options  Window  Help
def greet(name):
    return f"Hello, {name}!"

print(type(greet))            # <class 'function'>
print(callable(greet))        # True
print(callable(123))          # False
```

Yuqoridagi kodni tahlil qilamiz:

- `type(greet)` funksiyasi orqali `greet`nomli funksiyaning turi: bu `<class 'function'>`.
- `callable(greet)` — bu funksiya yord`greet`obyektning chaqirilishi tekshirilmoqda. Agar obyekt chaqiriladigan bo'lsa (`()` bilan ishlatilsa), u `True`qaytaradi.
- `callable(123)` — oddiy son bo'lgani uchun, chaqirib bo'lmaydi va `False`natija qaytariladi.

Xulosa:

Xulosa qilib aytganda Python dasturlash tilining muhim xususiyatlaridan biri — funksiyalarning birinchi darajali obyekt sifatida ishlatilishi atroflicha yoritildi. Ya’ni, funksiyalarni boshqa funksiyalarning parametri, natijasi yoki turi sifatida qo‘llash imkoniyati nafaqat sintaktik soddalik, balki dastur tuzilmasida yuqori darajadagi moslashuvchanlik va qayta ishlatish imkoniyatini ham ta’minlaydi. Bu yondashuv funksional dasturlash paradigmasi asoslaridan biri bo‘lib, murakkab vazifalarni modullashtirish, umumlashtirish va qisqa, aniq algoritmik yechimlar

yaratish imkonini beradi. Maqolada ushbu tamoyilning nazariy asoslari yoritildi va Python dasturlash tilida bu imkoniyat qanday amalga oshirilishi real kod namunalarida bosqichma-bosqich ko‘rsatib berildi. Jumladan, yuqori darajali funksiyalar (higher-order functions) — funksiyani argument yoki natija sifatida qabul qiluvchi funksiyalar — dasturdagi abstraksiyani oshirish, takrorlanuvchi kodlarni bitta umumiy unksiyada jamlash va kodni toza, o‘qilishi oson holatda saqlashda muhim vosita ekanligi ta’kidlandi.Shuningdek, map(), filter(), reduce(), lambda kabi funksional vositalar yordamida ma’lumotlar oqimiga funksiyalarni qo‘llash orqali kod tuzilmasini soddalashtirish va samaradorlikni oshirish mumkinligi ko‘rsatildi. Ayniqsa, funksiya ichida funksiya aniqlash (nested functions), closurelar, hamda decoratorlar kabi ilg‘or konsepsiyalar orqali Python tilining funksional imkoniyatlari chuqur ochib berildi.

Natijada, funksiyalarni boshqa funksiyalar bilan kombinatsiyalab ishlatish texnikasi — bu oddiy dasturlash usuli emas, balki modullararo bog‘liqlikni kamaytirish, kodni qayta foydalanishga moslashtirish va ilg‘or dasturiy arxitekturani yaratish imkonini beruvchi samarali yondashuv ekanligi isbotlandi. Bu esa Python tilining nafaqat o‘rganish uchun qulay, balki professional miqyosda kuchli imkoniyatlarga ega til ekanini yana bir bor tasdiqlaydi.

FOYDALANILGAN ADABIYOTLAR:

1. Van Rossum, G., & Drake, F. L. (2009). The Python Language Reference Manual. Network Theory Ltd.

2. Lutz, M. (2013). Learning Python (5th ed.). O’Reilly Media.

3. Beazley, D. M. (2009). Python Cookbook. O’Reilly Media.

4. Hetland, M. L. (2017). Beginning Python: From Novice to Professional (3rd ed.). Apress.

5. Python Software Foundation. (2024). Python Documentation. Retrieved from https://docs.python.org

6. Real Python. (2024). How to Use Higher-Order Functions in Python. Retrieved from https://realpython.com

ISSN 3060-4567

7. Geeks for Geeks. (2024). First-Class Functions in Python. Retrieved from https://www.geeksforgeeks.org