PROGRAMSKI JEZIK PAJTON

Pajton (engl. *Python*) je interpretirani programski jezik koji se odlikuje **jednostavnom i čitljivom sintaksom**. Za razliku od jezika kao što su C i Java, Pajton ne zahteva eksplicitnu kompilaciju – kod se izvršava direktno pomoću interpretera, što ga čini pogodnim za brzo razvijanje i testiranje programa.

Jedna od ključnih karakteristika Pajtona je **naglasak na čitljivosti koda**. Blokovi koda se ne definišu vitičastim zagradama kao u C/Java, već uvlačenjem (indentacijom), što podstiče pisanje urednog i preglednog koda.

Pajton je **dinamički tipiziran jezik**, što znači da nije potrebno unapred deklarisati tip promenljive. Ovo ubrzava razvoj, ali zahteva pažnju tokom rada zbog mogućih grešaka u tipovima u toku izvršavanja.

U poređenju sa Bešom, Pajton je znatno moćniji i pogodniji za razvoj **složenijih aplikacija**, dok zadržava jednostavnost skript jezika. U poređenju sa programskim jezikom C i Javom, Pajton omogućava brže pisanje koda sa manje formalnosti, ali uz nešto manju kontrolu nad resursima i performansama.

Pajton se široko koristi u različitim oblastima: automatizacija, obrada podataka, veštačka inteligencija, web razvoj i sistemsko programiranje, što ga čini jednim od najpraktičnijih jezika za savremene programere.

Instalacija programskog jezika Python dostupna je na zvaničnoj internet stranici, gde se mogu pronaći odgovarajuće verzije za različite operativne sisteme, kao i detaljna uputstva za instalaciju.

Nakon uspešne instalacije, u terminalu je moguće proveriti koja je verzija programskog jezika instalirana izvršavanjem sledeće komande:

```
python --version
```

Pokretanjem komande `python` ili `python3` u terminalu otvara se interaktivno Python okruženje, u okviru kojeg je moguće direktno izvršavati Python komande i testirati kraće delove koda.

Za razvoj složenijih programa i organizovanije pisanje koda preporučuje se korišćenje integrisanih razvojnih okruženja (engl. *Integrated Development Environment* - IDE). Pajton skripte se standardno čuvaju sa ekstenzijom `.py`.

Pored klasičnih razvojnih okruženja, preporučuje se i korišćenje Jupyter Notebook okruženja, koje omogućava interaktivno izvršavanje koda, kombinovanje programskog koda, rezultata i

tekstualnih objašnjenja u okviru jednog dokumenta, što ga čini naročito pogodnim za učenje, analizu podataka i akademski rad.

Uvod u Python kao skript jezik

Pajton je višepardigmski, skriptni i opštenamenski programski jezik visokog nivoa. Kao takav, omogućava primenu različitih programskih paradigmi, uključujući proceduralno, objektno-orijentisano i funkcionalno programiranje, u okviru istog jezika. Zbog svoje fleksibilnosti i jednostavne sintakse, Pajton se široko koristi u različitim domenima softverskog razvoja.

Kao skriptni jezik, Pajton se najčešće primenjuje u kontekstu **automatizacije zadataka, obrade podataka i brzog prototipisanja rešenja**. Njegova uloga u ovim scenarijima zasniva se na mogućnosti brzog pisanja, testiranja i izvršavanja koda bez potrebe za kompleksnim procesom kompilacije.

Karakteristike Pajtona uključuju visoku čitljivost i jednostavnu sintaksu, što doprinosi lakšem razumevanju i održavanju koda. Jezik koristi dinamičko tipiziranje, pri čemu se tipovi podataka određuju tokom izvršavanja programa, a ne u fazi deklaracije. Pajton poseduje i **bogatu standardnu biblioteku** koja obuhvata veliki broj modula za rad sa sistemom, podacima, mrežom i različitim formatima. Ove osobine ga čine posebno pogodnim za brzi razvoj softverskih rešenja (engl. *rapid prototyping*).

Primer primene Pajtona u automatizaciji zadataka:

```
import os



for ime in os.listdir("."):

    if ime.endswith(".txt"):

        print(ime)
```

U navedenom primeru, funkcija `os.listdir(".")` vraća listu svih fajlova i direktorijuma u tekućem radnom direktorijumu, slično komandi `ls` u Unix/Linux sistemima. Zatim se kroz dobijenu listu iterira pomoću petlje, pri čemu se svaki element proverava i filtrira na osnovu ekstenzije. Na taj način se logika filtriranja implementira direktno u jeziku, bez potrebe za eksternim alatima ili dodatnim sistemskim komandama.

Python kao interpretirani jezik

Pajton je interpretirani programski jezik, što znači da se izvorni kod izvršava putem interpretera, bez potrebe za eksplicitnim korakom kompilacije koji je vidljiv i odvojen od strane korisnika. Drugim rečima, programer ne pokreće zaseban proces prevođenja u mašinski kod, već se izvršavanje odvija direktno kroz interpretacioni sistem jezika.

Proces izvršavanja Pajton programa odvija se u nekoliko internih faza. Izvorni kod napisan u datoteci sa ekstenzijom `.py` najpre se automatski prevodi u međukod, poznat kao bajtkod (`.pyc`). Ovaj bajtkod nije direktno mašinski kod procesora, već predstavlja optimizovanu, platformski nezavisnu reprezentaciju programa. Nakon toga, bajtkod izvršava Pajton virtuelna mašina (engl. *Python Virtual Machine* – PVM), koja interpretira instrukcije i prosleđuje ih sistemu za izvršavanje.

U poređenju sa drugim programskim jezicima, ovaj model izvršavanja može se predstaviti na sledeći način:

U jeziku C, izvorni kod (`.c`) prolazi kroz eksplicitni proces kompilacije pomoću kompajlera, nakon čega se dobija direktan mašinski kod specifičan za određenu arhitekturu procesora.

U jeziku Java, izvorni kod (`.java`) se prevodi u bajtkod, koji se zatim izvršava unutar Java virtuelne mašine (JVM), pri čemu je kompilacija jasno odvojen korak.

U Pajtonu, izvorni kod (`.py`) se automatski prevodi u bajtkod (`.pyc`), nakon čega ga izvršava Python virtuelna mašina (PVM), pri čemu je ovaj proces implicitno integrisan u sistem izvršavanja i nije eksplicitno vidljiv korisniku.

Pajton interpreter i okruženje za rad

Pokretanje programskog jezika Pajton moguće je realizovati na više načina, u zavisnosti od potreba korisnika i tipa zadatka koji se izvršava.

Interaktivni režim rada pokreće se izvršavanjem sledeće komande u terminalu (ako je verzija python3):

```
python3
```

Na ovaj način otvara se interaktivno okruženje u kojem korisnik može neposredno unositi i izvršavati Pajton komande.

Izvršavanje Pajton skripte realizuje se navođenjem naziva datoteke nakon komande `python3`:

```
python3 program.py
```

U navedenom primeru izvršava se skripta pod nazivom `program.py`, koja sadrži Pajton programski kod.

Interaktivno Pajton okruženje funkcioniše po principu **REPL** sistema (*Read–Eval–Print Loop*), što podrazumeva sledeći proces:

- unos komande (*Read*),
- izvršavanje izraza (*Eval*),
- prikaz rezultata (*Print*),
- ponavljanje procesa (*Loop*).

Primer rada u REPL okruženju:

```
>>> 2 + 3
5
```

Ovakav način rada naročito je pogodan za brzo testiranje ideja, proveru sintakse i eksperimentisanje sa kraćim segmentima programskog koda. Konceptualno, REPL okruženje je slično komandnoj liniji u okviru Baš sistema, ali uz mogućnost korišćenja kompletne funkcionalnosti i izražajne snage programskog jezika Pajton.

Osnovna sintaksa Pajtona

Osnovna sintaksa Pajtona zasniva se na jednostavnosti, čitljivosti i minimalizmu u pisanju koda. Jedna od ključnih karakteristika ovog jezika jeste odsustvo nepotrebnih sintaksnih elemenata koji su uobičajeni u drugim programskim jezicima, kao što su znakovi za završetak naredbe ili eksplicitno označavanje blokova.

Jedna od osnovnih sintaksičkih osobenosti jeste činjenica da se na kraju naredbe ne koristi znak tačka-zarez (`;`), osim u specifičnim situacijama kada se više naredbi piše u jednoj liniji. Na ovaj način Pajton podstiče pisanje preglednog i linearnog koda, gde svaka linija predstavlja jednu logičku celinu.

Druga važna karakteristika jeste način definisanja blokova koda. Za razliku od jezika kao što su C ili Java, gde se blokovi ograničavaju vitičastim zagradama `{ }`, Pajton koristi uvlačenje/indentaciju (engl. *indent*) kao sintaksno sredstvo za definisanje strukture programa. Ovakav pristup čini kod vizuelno konzistentnim i znatno čitljivijim, jer struktura programa postaje direktno vidljiva kroz raspored linija.

Pajton je **dinamički tipiziran jezik**, što znači da nije neophodno eksplicitno deklarisati tip promenljive. Tip podatka se određuje tokom izvršavanja programa na osnovu dodeljene vrednosti. Ovakav model povećava fleksibilnost i ubrzava razvoj, ali zahteva pažljivije rukovanje podacima tokom izvršavanja.

U nastavku je prikazan jednostavan primer osnovne sintakse:

```
x = 10

if x > 5:

    print("Veće od 5")
```

U navedenom primeru, promenljivoj `x` dodeljuje se celobrojna vrednost. Nakon toga se izvršava uslovna struktura `if`, pri čemu se proverava da li je vrednost promenljive veća od 5. Ukoliko je uslov ispunjen, izvršava se naredba unutar bloka.

Blok koda je definisan isključivo putem uvlačenja, pri čemu svi redovi koji pripadaju istom bloku moraju imati identičan nivo indentacije. Na ovaj način Pajton eliminiše potrebu za specijalnim znakovima za označavanje početka i kraja bloka, što doprinosi većoj preglednosti i doslednosti sintakse. O ovome će biti više reči u nastavku.

Struktura Pajton programa

Tipična struktura:

```
def main():
    print("Hello, world")

if __name__ == "__main__":
    main()
```

Ova poslednja konstrukcija u Pajtonu služi za kontrolu izvršavanja koda.
Atribut `__name__` ima vrednost `"__main__"` samo kada se fajl izvršava direktno kao glavna skripta. Ako se isti fajl uvozi kao modul u drugi program, `__name__` dobija ime modula umesto `"__main__"`.

Zbog toga se poziv funkcije `main()` unutar ovog uslova izvršava isključivo pri direktnom pokretanju fajla, dok se pri uvozu modula ne aktivira automatski. Ovo omogućava da isti fajl bude istovremeno i izvršiv program i ponovo upotrebljiv modul u drugim programima.

Komentari u Pajtonu

Komentari u Pajtonu predstavljaju deo izvornog koda koji se ne izvršava tokom interpretacije programa, već služi isključivo za objašnjenje, dokumentovanje i poboljšanje čitljivosti koda. Njihova osnovna uloga ogleda se u pojašnjavanju programske logike, olakšavanju razumevanja složenijih delova koda, dokumentovanju funkcionalnosti, kao i privremenom isključivanju delova koda tokom razvoja i testiranja.

Jednolinijski komentari u Pajtonu započinju znakom #, pri čemu interpreter ignoriše sav tekst koji se nalazi nakon ovog simbola u istoj liniji. Ovakav oblik komentara najčešće se koristi za kratka objašnjenja pojedinačnih naredbi ili izraza.

```
# Ovo je komentar
x = 5  # dodela vrednosti promenljivoj
```

Python ne poseduje posebnu sintaksu za višelinijske komentare u stilu drugih programskih jezika. Umesto toga, u praksi se koristi niz uzastopnih jednolinijskih komentara, čime se postiže isti efekat višelinijskog objašnjenja.

```
# Program računa zbir
# svih parnih brojeva
# u listi
```

Za dokumentovanje funkcija, klasa i modula koristi se docstring, odnosno dokumentacioni string, koji se piše unutar trostrukih navodnika. Docstring se smatra standardnim načinom formalne dokumentacije u Pajtonu i omogućava automatsko prikazivanje pomoću ugrađene funkcije `help()`.

```
def zbir(a, b):
    """
    Funkcija vraća zbir dva broja.
    """
    return a + b
```

Komentari se često koriste i tokom razvoja programa za privremeno isključivanje određenih delova koda, posebno u fazi testiranja i otklanjanja grešaka.

```
# print("Debug informacija")
```

Prema preporukama stila definisanog kroz PEP 8, komentar treba da bude kratak, jasan i informativan, pri čemu se posle znaka `#` ostavlja razmak. Takođe se preporučuje upotreba gramatički ispravnih i razumljivih rečenica kada je to potrebno, naročito u obimnijim objašnjenjima.

U profesionalnoj praksi, komentari se koriste pre svega za objašnjenje složenije logike, opis algoritama, objašnjenje neobičnih ili nestandardnih rešenja, kao i za dokumentovanje javnih funkcija i modula. Nasuprot tome, ne preporučuje se komentarisanje očiglednih operacija, kao ni zadržavanje zastarelih ili netačnih komentara, jer oni mogu dovesti do pogrešnog razumevanja koda.

U celini, kvalitetni komentari predstavljaju važan element dobrog softverskog dizajna, jer doprinose čitljivosti, održivosti i lakšoj saradnji između programera, pri čemu dobar komentar dopunjuje kod, a ne ponavlja njegovu očiglednu funkcionalnost.

Aritmetičke operacije u Pajtonu

Aritmetičke operacije omogućavaju izvođenje osnovnih matematičkih proračuna nad numeričkim vrednostima. Python podržava standardne operatore za sabiranje, oduzimanje, množenje, deljenje, računanje ostatka pri deljenju i stepenovanje.

Sabiranje (`+`)

Operator `+` koristi se za sabiranje numeričkih vrednosti.

```
a = 10
b = 5

print(a + b)
```

Rezultat:

15

Oduzimanje (`-`)

Operator `-` koristi se za izračunavanje razlike između dve vrednosti.

```
a = 10
b = 5

print(a - b)
```

Rezultat:

5

Množenje (`*`)

Operator `*` koristi se za množenje numeričkih vrednosti.

```
a = 10
b = 5

print(a * b)
```

Rezultat:

50

Deljenje (`/`)

Operator `/` vrši standardno deljenje i uvek vraća rezultat tipa `float`.

```
print(10 / 5)
```

Rezultat:

2.0

Ovo ponašanje predstavlja jednu od razlika u odnosu na neke druge programske jezike (npr. C ili Java pri radu sa celobrojnim tipovima). Ova razlika predstavlja jednu od karakterističnih osobina Python jezika u odnosu na mnoge druge programske jezike. Na primer:

```
print(5 / 2)
```

Rezultat:

2.5

Celobrojno deljenje (`//`)

Ako je potreban samo ceo deo količnika, koristi se operator `//`.

```
print(5 // 2)
```

Rezultat:

2

Ovaj operator ima slično ponašanje kao celobrojno deljenje u mnogim drugim programskim jezicima.

Ostatak pri deljenju (`%`)

Operator `%` vraća ostatak nakon deljenja.

```
print(17 % 5)
```

Rezultat:

2

Često se koristi za proveru da li je broj paran:

```
print(12 % 2)
```

Rezultat:

```
0
```

Napomena: Voditi računa za ponašanje ovog operatora pri radu sa negativnim vrednostima i po potrebi koristi funkciju math.fmod() iz modula math koji je potrebno uključiti u tom slučaju.

Stepenovanje (`**`)

Operator `**` koristi se za računanje stepena broja.

```
print(2 ** 3)
```

Rezultat:

```
8
```

Za razliku od jezika kao što su C, C++ i Java, gde se stepenovanje obično vrši pozivom posebnih funkcija (`pow()`), Python poseduje poseban operator za ovu namenu.

Prioritet operatora

Python poštuje standardna matematička pravila:

1. Zagrade `()`
2. Stepenovanje `**`
3. Množenje `*`, deljenje `/`, celobrojno deljenje `//`, ostatak `%`
4. Sabiranje `+` i oduzimanje `-`

Primer:

```
print(2 + 3 * 4)
```

Rezultat:

```
14
```

Dok:

```
print((2 + 3) * 4)
```

daje rezultat:

20

Modul `math`

Modul `math` predstavlja standardni Python modul koji sadrži veliki broj matematičkih funkcija i konstanti za izvođenje složenijih numeričkih proračuna. Njegova upotreba omogućava efikasno rešavanje matematičkih problema bez potrebe za implementacijom sopstvenih algoritama.

Pre korišćenja modula potrebno ga je uključiti u program:

```
import math
```

Najčešće korišćene funkcije

Funkcija	Namena	Primer	Rezultat
math.sqrt(x)	Kvadratni koren broja	math.sqrt(25)	5.0
math.pow(x, y)	Stepenovanje	math.pow(2, 3)	8.0
math.ceil(x)	Zaokruživanje naviše	math.ceil(4.2)	5
math.floor(x)	Zaokruživanje naniže	math.floor(4.8)	4
math.fabs(x)	Apsolutna vrednost	math.fabs(-7)	7.0
math.factorial(x)	Faktorijel broja	math.factorial(5)	120
math.sin(x)	Sinus ugla u radijanima	math.sin(math.pi/2)	1.0
math.cos(x)	Kosinus ugla u radijanima	math.cos(0)	1.0
math.log(x)	Prirodni logaritam	math.log(math.e)	1.0

Matematičke konstante

Modul sadrži i često korišćene matematičke konstante:

Konstanta	Značenje	Vrednost

math.pi	Broj π	3.141592653589793
math.e	Ojlerov broj	2.718281828459045
math.inf	Beskonačnost	∞

Primer:

```
import math

print(math.pi)
print(math.e)
```

Rezultat:

```
3.141592653589793
2.718281828459045
```

Modul `math` obezbeđuje skup gotovih matematičkih funkcija i konstanti koje omogućavaju jednostavno izvođenje numeričkih proračuna. Najčešće se koristi za rad sa korenima, stepenima, logaritmima, trigonometrijskim funkcijama i zaokruživanjem brojeva, čime značajno olakšava razvoj matematičkih i naučnih aplikacija.

Kontrola toka izvršavanja u Pajtonu

Kontrola toka (engl. *control flow*) u Pajtonu predstavlja mehanizam koji određuje redosled izvršavanja naredbi unutar programa. Ona definiše način na koji se pojedinačne instrukcije i blokovi koda organizuju i izvršavaju, u zavisnosti od uslova, iteracija i strukture programa. Kontrola toka je ključna za implementaciju logike svakog netrivijalnog softverskog sistema.

U najosnovnijem obliku, izvršavanje programa u Pajtonu odvija se **sekvencijalno**, što znači da se naredbe izvršavaju redom, od vrha ka dnu, onako kako su napisane u izvornoj datoteci. Ovakav model predstavlja podrazumevani tok izvršavanja ukoliko nisu uvedene dodatne strukture kontrole toka.

Pored sekvencijalnog izvršavanja, Pajton omogućava **grananje**, odnosno **uslovno izvršavanje** koda. Grananje se implementira pomoću uslovnih struktura kao što je `if` naredba, pri čemu se određeni blok koda izvršava samo ukoliko je definisani logički uslov zadovoljen. Ovaj mehanizam omogućava donošenje odluka unutar programa i prilagođavanje ponašanja u zavisnosti od ulaznih podataka ili stanja sistema.

Ponavljanje, odnosno iteracija, predstavlja sledeći osnovni mehanizam kontrole toka. U Pajtonu se realizuje pomoću **petlji** `for` i `while`. Ove strukture omogućavaju višestruko izvršavanje istog bloka koda, bilo unapred definisan broj puta, bilo dok je određeni uslov ispunjen. Petlje predstavljaju osnovni alat za obradu kolekcija podataka, automatsku obradu nizova i implementaciju algoritamskih procedura.

Pored osnovnih struktura za kontrolu toka, **funkcije** imaju ključnu ulogu u organizaciji programa i ostvarivanju modularnosti. Funkcije omogućavaju razlaganje kompleksnog problema na manje, logički zaokružene celine koje se mogu samostalno definisati, testirati i ponovo koristiti. Na taj način se postiže viši nivo apstrakcije, bolja čitljivost koda i lakše održavanje programa.

U savremenom Pajton programiranju, kontrola toka se posmatra kao kombinacija sekvencijalnog izvršavanja, uslovnog grananja, iterativnog ponavljanja i funkcionalne dekompozicije, pri čemu svaka od ovih komponenti doprinosi jasnoj strukturi i efikasnoj implementaciji algoritamske logike.

Uslovne strukture i petlje

Uslovni izrazi u Pajtonu predstavljaju osnovni mehanizam grananja toka izvršavanja programa, pri čemu se određeni blok koda izvršava u zavisnosti od ispunjenosti logičkog uslova. Pajton koristi jednostavnu sintaksu bez zagrada oko uslova, pri čemu se struktura bloka definiše isključivo putem indentacije, što doprinosi čitljivosti i jasnoći koda.

```
x = 7

if x % 2 == 0:
```

```
    print("Paran")
else:
    print("Neparan")
```

U navedenom primeru, uslov `x % 2 == 0` određuje da li je broj paran. Ukoliko je uslov tačan, izvršava se prvi blok, u suprotnom se izvršava alternativni blok definisan kroz `else` granu. Za razliku od jezika kao što su C ili Java, Pajton ne zahteva zagrade oko uslova, dok se logička struktura u potpunosti oslanja na uvlačenje koda.

Petlja while

Petlja `while` predstavlja kontrolnu strukturu koja omogućava ponavljanje izvršavanja bloka koda sve dok je određeni logički uslov ispunjen. Ovaj oblik petlje se koristi kada broj iteracija nije unapred poznat, već zavisi od dinamike izvršavanja programa.

```
i = 0
while i < 5:
    print(i)
    i += 1
```

U ovom primeru, petlja se izvršava sve dok je vrednost promenljive `i` manja od 5. Nakon svake iteracije, vrednost se inkrementira, čime se obezbeđuje uslov za eventualni izlazak iz petlje. Ukoliko se ovaj korak ne izvrši, dolazi do beskonačne petlje.

Petlja for

Petlja `for` u Pajtonu predstavlja iterativnu strukturu koja se razlikuje od klasičnih C/Java for petlji, jer ne funkcioniše kao brojačka petlja u tradicionalnom smislu, već kao mehanizam za iteraciju kroz elemente bilo koje iterabilne strukture.

Osnovni oblik petlje for

```
for i in range(5):
    print(i)
```

Funkcija `range(5)` generiše sekvencu vrednosti od 0 do 4. Petlja iterira kroz ovu sekvencu i za svaku vrednost izvršava definisani blok koda.

range sa početkom i krajem

```
for i in range(2, 6):
    print(i)
```

Funkcija `range(start, stop)` generiše sekvencu počevši od početne vrednosti, uključujući je, pa sve do krajnje vrednosti koja nije uključena u opseg. U ovom slučaju generišu se vrednosti 2, 3, 4 i 5.

range sa korakom

```
for i in range(0, 10, 2):
    print(i)
```

Treći parametar funkcije `range` definiše korak iteracije. U ovom primeru generišu se vrednosti 0, 2, 4, 6 i 8, pri čemu se vrednost povećava za 2 u svakoj iteraciji.

Iteracija kroz listu

```
brojevi = [10, 20, 30]


for x in brojevi:
    print(x)
```

Ovaj oblik petlje omogućava direktnu iteraciju kroz elemente liste, bez potrebe za eksplicitnim korišćenjem indeksa. Ovakav pristup predstavlja jednu od ključnih razlika u odnosu na C/Java stil programiranja.

Iteracija kroz string

```
for slovo in "Python":
    print(slovo)
```

String u Pajtonu predstavlja sekvencu karaktera, što omogućava njegovu direktnu iteraciju kao i kod drugih kolekcija podataka.

Iteracija sa indeksom (enumerate)

```
imena = ["Ana", "Marko", "Iva"]

for i, ime in enumerate(imena):
    print(i, ime)
```

Funkcija `enumerate()` omogućava istovremeno dobijanje indeksa i vrednosti elementa, što predstavlja elegantnu zamenu za ručno upravljanje brojačem u klasičnim jezicima.

Iteracija kroz rečnik (dict)

```
osoba = {"ime": "Ana", "godine": 25}

for kljuc, vrednost in osoba.items():
    print(kljuc, vrednost)
```

Metoda `items()` omogućava iteraciju kroz parove ključ–vrednost u rečniku, što predstavlja prirodan i čest način rada sa mapiranim strukturama podataka u Pajtonu.

break i continue

```
for i in range(5):
    if i == 3:
        break
```

```
    print(i)
```

Naredba `break` prekida izvršavanje petlje nezavisno od preostalih iteracija.

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Naredba `continue` preskače trenutnu iteraciju i prelazi na sledeću, bez izvršavanja preostalog dela bloka u toj iteraciji.

for-else konstrukcija

```
for i in range(3):
    print(i)
else:
    print("Petlja završena")
```

Konstrukcija `for-else` predstavlja specifičnost Pajtona. Blok `else` se izvršava isključivo ako petlja nije prekinuta naredbom `break`, što nema direktan ekvivalent u jezicima kao što su C ili Java.

Funkcije u Pajtonu

U Pajtonu, funkcije predstavljaju imenovane blokove koda koji izvršavaju određenu, jasno definisanu operaciju i mogu se pozivati više puta tokom izvršavanja programa.

Funkcije se definišu pomoću ključne reči `def`, nakon čega sledi ime funkcije, lista parametara u zagradama i telo funkcije.

One omogućavaju modularnost, ponovnu upotrebu koda i bolju organizaciju programa, kao i lakše održavanje i testiranje. Funkcije mogu vraćati vrednosti pomoću naredbe `return`, ali to nije obavezno.

Primer:

```
def zbir(a, b):
    return a + b

print(zbir(3, 4))
```

Primetimo da nema tipova u potpisu funkcije.

Pozicioni i imenovani argumenti

U Pajtonu, argumenti funkcija mogu se prosleđivati kao pozicioni (engl. *positional*) i imenovani (engl. *keyword*) argumenti.

Pozicioni argumenti se dodeljuju parametrima funkcije na osnovu redosleda u kojem su navedeni pri pozivu funkcije. Redosled argumenta je tada značajan i mora odgovarati definiciji funkcije.

Imenovani argumenti se prosleđuju eksplicitnim navođenjem imena parametra pri pozivu funkcije, čime se uklanja zavisnost od redosleda i povećava čitljivost koda.

Pajton omogućava kombinovanje ova dva pristupa, pri čemu pozicioni argumenti moraju prethoditi imenovanim.

Pozicioni argumenti

```
def pozdrav(ime, prezime):
    print(ime, prezime)

pozdrav("Ana", "Anić")
```

Imenovani (keyword) argumenti

```
pozdrav(prezime="Anić", ime="Ana")
```

Primetimo:

- redosled nije bitan
- povećava čitljivost

Kombinovanje

```
def funkcija(a, b, c=0):
    print(a, b, c)

funkcija(1, 2)

funkcija(1, 2, 3)
```

Strukture podataka u Pajtonu

Strukture podataka u Pajtonu predstavljaju osnovni koncept organizacije, skladištenja i obrade podataka u programskim sistemima. Pajton obezbeđuje skup ugrađenih, fleksibilnih i efikasnih struktura podataka koje omogućavaju jednostavnu, ali i moćnu manipulaciju nad kolekcijama informacija. Najčešće korišćene strukture podataka su liste (list), torke (tuple), skupovi (set) i rečnici (dict), pri čemu svaka od njih ima jasno definisanu namenu, ponašanje i tipične oblasti primene.

Liste (list)

Liste predstavljaju **uređene** i **promenljive** kolekcije elemenata. Elementi liste mogu biti različitih tipova podataka, a pristup elementima ostvaruje se putem indeksa, pri čemu indeksiranje počinje od nule. Dodatno, negativni indeksi omogućavaju pristup elementima sa kraja liste, gde vrednost `-1` označava poslednji element. Liste su dinamičke strukture, što znači da se njihov sadržaj može menjati tokom izvršavanja programa.

```
brojevi = [1, 2, 3, 4]

print(brojevi[0])
print(brojevi[-1])

brojevi[1] = 10
brojevi.append(5)
```

Dodavanje elemenata u listu najčešće se realizuje metodom `append`, koja dodaje element na kraj liste. Umetanje elemenata na određenu poziciju vrši se metodom `insert`, koja pomera postojeće elemente udesno.

```
brojevi = [10, 20, 30]

brojevi.append(40)
brojevi.insert(1, 15)
```

Uklanjanje elemenata može se vršiti metodom `remove`, koja uklanja prvo pojavljivanje zadate vrednosti, kao i metodom `pop`, koja uklanja element na osnovu indeksa.

```
brojevi = [5, 10, 15, 10]

brojevi.remove(10)
print(brojevi)
```

```
brojevi.pop(1)
print(brojevi)
```

Liste omogućavaju iteraciju kroz sve elemente, što se najčešće realizuje pomoću `for` petlje.

```
brojevi = [1, 2, 3, 4]

for broj in brojevi:
    print(broj)
```

Pored toga, Python podržava *list comprehension*, koji predstavlja kompaktan i idiomatski način kreiranja novih lista kroz transformaciju ili filtriranje postojećih podataka.

```
kvadrati = [x * x for x in range(5)]
parni = [x for x in range(10) if x % 2 == 0]
```

Torke (tuple)

Torke predstavljaju **uređene**, ali **nepromenljive** (engl. *immutable*) kolekcije elemenata. Nakon kreiranja, sadržaj torke se ne može menjati, što obezbeđuje stabilnost i zaštitu podataka od slučajnih izmena. Elementima torke se pristupa putem indeksa, na isti način kao kod lista.

```
koordinate = (10, 20)

print(koordinate[0])
print(koordinate[1])
```

Zbog svoje nepromenljivosti, torke se koriste za predstavljanje podataka koji imaju **fiksnu strukturu**, kao što su koordinate, datumi ili konfiguracioni parametri.

```
datum = (13, 5, 2026)
dan, mesec, godina = datum
print(godina)
```

Torke se često koriste za **vraćanje više vrednosti iz funkcija**, pri čemu se rezultat može direktno raspakovati u više promenljivih.

```
def minimum_maksimum(lista):
    return (min(lista), max(lista))
```

```
rezultat = minimum_maksimum([4, 7, 1, 9])
print(rezultat)

minimum, maksimum = minimum_maksimum([4, 7, 1, 9])
print(minimum)
print(maksimum)
```

Skupovi (set)

Skupovi predstavljaju **neuređene** kolekcije jedinstvenih elemenata, pri čemu se automatski **uklanjaju duplikati.** Elementi nisu indeksirani, ali skupovi omogućavaju veoma efikasnu proveru pripadnosti i podržavaju osnovne skupovne operacije.

```
brojevi = {1, 2, 3, 3, 2}
print(brojevi)
```

Skupovi se često koriste za uklanjanje duplikata iz postojećih kolekcija, posebno kada je važna jedinstvenost podataka.

```
imena = ["Ana", "Marko", "Ana", "Iva"]
jedinstvena_imena = set(imena)
print(jedinstvena_imena)
```

Jedna od ključnih prednosti skupova jeste vrlo brza provera postojanja elementa u kolekciji.

```
dozvoljeni = {"admin", "editor", "user"}

if "admin" in dozvoljeni:
    print("Postoji")
```

Skupovi podržavaju matematičke operacije kao što su unija, presek i razlika, što ih čini pogodnim za analitičke i algoritamske primene.

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)
print(a & b)
print(a - b)
```

Rečnici (dict)

Rečnici predstavljaju strukturu podataka zasnovanu na **mapiranju ključ–vrednost**, pri čemu svaki ključ jednoznačno određuje odgovarajuću vrednost. Ključevi moraju biti jedinstveni, dok vrednosti mogu biti proizvoljnog tipa.

```
student = {
    "ime": "Ana",
    "godine": 20
}

print(student["ime"])
```

Rečnici su dinamičke strukture, što omogućava dodavanje novih parova kao i izmenu postojećih vrednosti tokom izvršavanja programa.

```
student["prosek"] = 9.5
student["godine"] = 21
```

Iteracija kroz rečnik omogućava pristup ključevima i vrednostima, pri čemu se često koristi metoda `items()`.

```
for kljuc, vrednost in student.items():
    print(kljuc, vrednost)
```

Rečnici se često koriste u kombinaciji sa listama, posebno kada je potrebno predstaviti skup strukturisanih zapisa.

```
studenti = [
    {"ime": "Ana", "prosek": 9.2},
    {"ime": "Marko", "prosek": 8.1}
]

for student in studenti:
    print(student["ime"])
```

Odabir odgovarajuće strukture podataka u Pajtonu zavisi od prirode problema i zahteva konkretne primene. Liste se koriste kada je potrebna uređena i promenljiva kolekcija podataka, torke kada je neophodna nepromenljivost i stabilna struktura, skupovi kada je važna jedinstvenost i efikasna provera pripadnosti, dok se rečnici koriste za mapiranje i organizaciju

podataka u obliku parova ključ–vrednost. Pravilna upotreba ovih struktura doprinosi efikasnosti, jasnoći i dugoročnoj održivosti programskog koda.

Primeri u Pajtonu - osnovni algoritmi nad strukturama podataka

Uobičajene operacije nad listama u Pajtonu često se implementiraju kroz jednostavne iterativne algoritme koji ilustruju osnovne principe obrade sekvencijalnih struktura podataka. Navedeni jednostavni algoritmi su navedeni kao dodatni primeri u Pajtonu.

Linearna pretraga predstavlja osnovni algoritam pretraživanja u kojem se elementi liste prolaze sekvencijalno, sve dok se ne pronađe traženi element ili se ne iscrpi cela lista. U najgorem slučaju, vremenska složenost ovog algoritma je linearna, odnosno O(n), gde je n broj elemenata u listi.

```
def linearna_pretraga(lista, cilj):
    for x in lista:
        if x == cilj:
            return True
    return False
```

Pronalaženje maksimuma u listi zasniva se na iterativnom poređenju elemenata, pri čemu se održava trenutno najveća vrednost koja se ažurira tokom prolaska kroz listu.

```
def maksimum(lista):
    max_v = lista[0]
    for x in lista:
        if x > max_v:
            max_v = x
    return max_v
```

Zbir elemenata liste predstavlja akumulativnu operaciju u kojoj se svi elementi sukcesivno sabiraju, čime se dobija ukupna vrednost skupa podataka.

```
def zbir(lista):
    s = 0
    for x in lista:
        s += x
    return s
```

Brojanje pojavljivanja određenog elementa podrazumeva iteraciju kroz listu i inkrementiranje brojača svaki put kada se pronađe element koji odgovara zadatom kriterijumu.

```
def broj_ponavljanja(lista, element):
    count = 0
    for x in lista:
        if x == element:
            count += 1
    return count
```

Filtriranje elemenata liste predstavlja formiranje nove liste koja sadrži samo elemente koji zadovoljavaju određeni uslov, pri čemu se originalna struktura ne menja.

```
def parni(lista):
    rezultat = []
    for x in lista:
        if x % 2 == 0:
            rezultat.append(x)
    return rezultat
```

U Pajtonu se ovi algoritmi često mogu izraziti na sažet i idiomatski način korišćenjem list comprehension konstrukcije, što povećava čitljivost i kompaktnost koda.

```
def parni(lista):
    return [x for x in lista if x % 2 == 0]
```

Regularni izrazi u programskom jeziku Pajton

Kao što je već bilo reči u toku kursa, regularni izrazi predstavljaju standardni mehanizam za prepoznavanje, pretraživanje i obradu tekstualnih obrazaca. U programskom jeziku Python podrška za rad sa regularnim izrazima obezbeđena je kroz modul `re`, koji omogućava Pajton poznatih regularnih izraza nad tekstualnim podacima.

Za korišćenje ovih funkcionalnosti neophodno je uključiti odgovarajući modul:

```
import re
```

Modul `re` sadrži skup funkcija koje omogućavaju pretragu, izdvajanje, zamenu i validaciju tekstualnih sadržaja korišćenjem regularnih izraza.

Pretraga teksta

Jedna od najčešće korišćenih funkcija jeste `search()`, koja pretražuje tekst i pronalazi prvo poklapanje sa zadatim obrascem.

```
import re


tekst = "Programski jezik Python razvijen je 1991. godine."

rezultat = re.search(r"\d{4}", tekst)

print(rezultat.group())
```

Rezultat:

```
1991
```

U slučaju da obrazac nije pronađen, funkcija vraća vrednost `None`.

Pronalaženje svih poklapanja

Funkcija `findall()` vraća listu svih delova teksta koji odgovaraju definisanom regularnom izrazu.

```
import re
```

```
tekst = "Brojevi su 15, 27 i 103."

brojevi = re.findall(r"\d+", tekst)

print(brojevi)
```

Rezultat:

['15', '27', '103']

Ova funkcija se često koristi kada je potrebno izdvojiti više vrednosti iz tekstualnog dokumenta ili korisničkog unosa.

Provera poklapanja na početku niske

Funkcija `match()` proverava da li se obrazac nalazi na početku tekstualne niske.

```
import re

tekst = "Python je veoma popularan."

if re.match(r"Python", tekst):
    print("Poklapanje postoji.")
```

Za razliku od funkcije `search()`, koja pretražuje čitav tekst, funkcija `match()` ispituje samo početak niske.

Zamena sadržaja

Modul `re` omogućava zamenu delova teksta korišćenjem funkcije `sub()`.

```
import re
```

```
tekst = "Verzija Python 3.10"

novi_tekst = re.sub(r"3\.10", "3.12", tekst)

print(novi_tekst)
```

Rezultat:

Verzija Python 3.12

Ovakav pristup je naročito koristan prilikom automatske obrade i transformacije većih količina tekstualnih podataka.

Korišćenje grupa

Grupe definisane zagradama omogućavaju izdvajanje pojedinačnih delova pronađenog obrasca.

```
import re


tekst = "Datum: 12/05/2025"

rezultat = re.search(r"(\d{2})/(\d{2})/(\d{4})", tekst)

print(rezultat.group(1))

print(rezultat.group(2))

print(rezultat.group(3))
```

Rezultat:

12

05

2025

Na ovaj način moguće je pristupiti pojedinačnim komponentama složenijih obrazaca bez dodatne obrade stringova.

Iteriranje kroz pronađena poklapanja

Funkcija `finditer()` vraća iterator svih pronađenih poklapanja zajedno sa informacijama o njihovoj poziciji u tekstu.

```
import re


tekst = "Python 3.10 i Python 3.12"

for poklapanje in re.finditer(r"Python", tekst):

    print(poklapanje.start())
```

Rezultat:

```
0

14
```

Ova funkcija je korisna kada je potrebno analizirati pozicije pronađenih obrazaca unutar dokumenta.

Sirovi stringovi

Prilikom definisanja regularnih izraza u Pythonu uobičajena je upotreba sirovih stringova (engl. *raw strings*), označenih prefiksom `r`.

```
obrazac = r"\d+"
```

Na ovaj način izbegavaju se problemi vezani za interpretaciju specijalnog znaka `\` unutar Python niske, čime zapis regularnih izraza postaje pregledniji i lakši za održavanje.

Primena regularnih izraza u Pythonu

U praktičnim primenama regularni izrazi se često koriste za:

- validaciju korisničkih unosa,
- izdvajanje podataka iz tekstualnih dokumenata,
- obradu log datoteka,
- parsiranje strukturiranih i polustrukturiranih podataka,
- automatsku transformaciju tekstualnog sadržaja,
- obradu podataka automatski prikupljenih sa internet stranica (engl. *Web scraping*).

Python modul `re` omogućava efikasnu primenu regularnih izraza kroz skup funkcija namenjenih pretrazi, validaciji, izdvajanju i modifikaciji tekstualnih podataka. Zahvaljujući jednostavnoj sintaksi i bogatom skupu funkcionalnosti, regularni izrazi predstavljaju značajan alat za automatizaciju poslova obrade teksta i analize podataka u savremenim softverskim sistemima.

Rad sa ulazom, izlazom i datotekama u Pajtonu

Rad sa ulazom, izlazom i datotekama u Pajtonu predstavlja osnovni deo programiranja koji omogućava interakciju sa korisnikom, kao i trajno čuvanje i obradu podataka. Pajton obezbeđuje jednostavne i intuitivne mehanizme za unos podataka sa standardnog ulaza, ispis rezultata na standardni izlaz, kao i rad sa različitim tipovima datoteka i formatima podataka.

Standardni ulaz i izlaz

Standardni ulaz u Pajtonu realizuje se putem funkcije `input()`, koja omogućava unos podataka od strane korisnika. Važno je napomenuti da funkcija `input()` uvek vraća podatke u obliku stringa, nezavisno od stvarnog sadržaja unosa.

```
ime = input("Unesite ime: ")

print(ime)
```

U situacijama kada je potrebno raditi sa numeričkim podacima, neophodna je eksplicitna konverzija tipa, najčešće u celobrojni ili decimalni tip.

```
godine = int(input("Unesite godine: "))

print(godine)
```

Ova karakteristika je posebno važna jer omogućava pravilnu obradu podataka i sprečava logičke greške u programu usled neodgovarajućeg tipa podataka.

Za prikaz podataka koristi se funkcija `print()`, koja omogućava standardni izlaz i podržava **formatirani ispis** putem f-string sintakse, što značajno povećava čitljivost koda.

```
ime = "Ana"

godine = 20


print(f"{ime} ima {godine} godina")
```

Rad sa datotekama

Rad sa datotekama omogućava trajno čuvanje podataka van memorije programa. Osnovni mehanizam podrazumeva otvaranje datoteke pomoću funkcije `open()`, pri čemu se definiše režim rada.

```
f = open("podaci.txt", "r")

tekst = f.read()

print(tekst)

f.close()
```

Režimi rada sa datotekama određuju način pristupa sadržaju. Režim `"r"` označava čitanje, `"w"` upis (uz prepisivanje sadržaja), `"a"` dodavanje na postojeći sadržaj, dok se `"rb"` koristi za binarno čitanje.

U savremenoj Python praksi preporučuje se korišćenje konteksta `with`, koji obezbeđuje automatsko zatvaranje datoteke nakon završetka operacija.

```
with open("podaci.txt", "r") as f:
    tekst = f.read()


print(tekst)
```

Upis u datoteku se realizuje na sličan način, pri čemu se podaci zapisuju pomoću metode `write()`.

```
with open("izlaz.txt", "w") as f:
    f.write("Python\n")
    f.write("Rad sa datotekama\n")
```

Čitanje datoteka može se vršiti i **liniju po liniju**, što je posebno korisno kod obrade velikih tekstualnih fajlova.

```
with open("podaci.txt", "r") as f:
    for linija in f:
        print(linija.strip())
```

Rad sa sistemom datoteka

Python omogućava rad sa sistemom datoteka putem modula `os`, koji pruža funkcionalnosti za pregled direktorijuma, upravljanje putanjama i proveru postojanja datoteka.

```
import os

print(os.listdir("."))
```

Provera postojanja datoteke realizuje se pomoću funkcije `os.path.exists()`, što je korisno u situacijama kada je potrebno izbeći greške pri radu sa nepostojećim fajlovima.

```
import os

print(os.path.exists("podaci.txt"))
```

Rad sa JSON formatom

JSON (JavaScript Object Notation) predstavlja standardni format za razmenu strukturisanih podataka između sistema. Python obezbeđuje modul `json` za rad sa ovim formatom.

Učitavanje JSON podataka iz datoteke vrši se pomoću funkcije `json.load()`.

```
import json


with open("student.json", "r") as f:
    podaci = json.load(f)


print(podaci)
```

Upis podataka u JSON format vrši se pomoću funkcije `json.dump()`.

```
import json


student = {
    "ime": "Ana",
    "godine": 20
}


with open("student.json", "w") as f:
    json.dump(student, f)
```

Rad sa CSV i TSV formatom

CSV (Comma-Separated Values) predstavlja format za čuvanje tabelarnih podataka u tekstualnom obliku. Python koristi modul `csv` za obradu ovih datoteka.

Čitanje CSV datoteka omogućava se pomoću `csv.reader`, pri čemu se podaci iteriraju red po red.

```
import csv


with open("podaci.csv", "r") as f:
    reader = csv.reader(f)


    for red in reader:
        print(red)
```

Upis u CSV datoteku vrši se pomoću `csv.writer`, gde se podaci zapisuju kao redovi.

```
import csv


with open("izlaz.csv", "w") as f:
    writer = csv.writer(f)


    writer.writerow(["ime", "godine"])
    writer.writerow(["Ana", 20])
```

TSV format predstavlja varijantu CSV formata u kojoj se kao separator koristi tabulator umesto zareza.

```
import csv


with open("podaci.tsv", "r") as f:
    reader = csv.reader(f, delimiter="\t")


    for red in reader:
        print(red)
```

Rad sa XML formatom

XML (eXtensible Markup Language) predstavlja hijerarhijski format za strukturisanje podataka. U Pajtonu se za rad sa XML datotekama koristi modul `xml.etree.ElementTree`.

Parsiranje XML datoteke omogućava pristup strukturi dokumenta kroz stablo elemenata.

```
import xml.etree.ElementTree as ET


tree = ET.parse("student.xml")
root = tree.getroot()


print(root.tag)
```

Iteracija kroz XML elemente omogućava pristup svakom čvoru u hijerarhiji dokumenta.

```
for element in root:
    print(element.tag, element.text)
```

Kao što se moglo primetiti u prethodnim primerima, Pajton obezbeđuje jednostavne i efikasne mehanizme za rad sa ulazom, izlazom i različitim formatima datoteka. Ove mogućnosti uključuju obradu tekstualnih podataka, rad sa strukturiranim formatima kao što su JSON, CSV i XML, kao i interakciju sa sistemom datoteka. Zbog svoje jednostavnosti i fleksibilnosti, Pajton se široko primenjuje u oblastima automatizacije, obrade podataka, veb razvoja, administracije sistema, kao i u oblastima data science i veštačke inteligencije.

Dobra programska praksa u Pajtonu

Dobra programska praksa u Pajtonu podrazumeva pisanje koda koji je čitljiv, dosledan i lako održiv, pri čemu se naglasak stavlja na razumljivost pre optimizacije „kompaktnosti“ ili složenosti rešenja. Osnovni cilj je da kod može biti jednostavno interpretiran od strane drugih programera, kao i da se olakša njegovo održavanje i dalje razvijanje.

Mnoga pravila važe i za mnoge druge programske jezike.

Čitljivost koda predstavlja jedan od ključnih principa. Kod treba da bude napisan na način koji omogućava jasno razumevanje bez potrebe za dodatnim objašnjenjima. Na primer, umesto sažetog pisanja više naredbi u jednoj liniji, preporučuje se njihovo razdvajanje radi preglednosti.

```
# loše

x=1;y=2;z=x+y



# dobro

x = 1

y = 2

z = x + y
```

Posebnu ulogu ima upotreba smislenih imena identifikatora. Promenljive i funkcije treba da imaju nazive koji jasno opisuju njihovu svrhu, čime se povećava razumljivost i smanjuje potreba za dodatnim komentarima.

```
broj_studenata = 25
```

Upotreba funkcija predstavlja osnovu modularnog programiranja i doprinosi smanjenju dupliranja koda. Kroz izdvajanje ponovljivih operacija u funkcije postiže se bolja organizacija i veća ponovna upotrebljivost koda.

```
def zbir(a, b):

    return a + b
```

Standard PEP 8 predstavlja zvanični vodič za stil pisanja Python koda. Njegova svrha je uspostavljanje uniformnog stila koji poboljšava čitljivost i olakšava timski rad na projektima. PEP 8 naglašava doslednu primenu pravila kao što su korišćenje razmaka oko operatora, ograničenje dužine linije koda i standardizovano imenovanje identifikatora.

U skladu sa ovim smernicama, preporučuje se korišćenje četiri razmaka za uvlačenje (indentaciju), pri čemu se tabulatori izbegavaju zbog mogućih nekonzistentnosti između različitih okruženja. Takođe se preporučuje ograničenje dužine linije koda na približno 79 karaktera, uz razbijanje dužih izraza radi bolje preglednosti.

```
if x > 0:
    print("Pozitivan")
```

Upotreba razmaka doprinosi dodatnoj čitljivosti koda, posebno oko operatora i nakon zareza, dok se razmaci unutar zagrada izbegavaju.

```
x = 5 + 3
print(a, b)
```

Pravila imenovanja u Pajtonu slede ustaljene konvencije. Promenljive i funkcije se pišu u snake_case stilu imenovanja, odnosno zapis reči malim slovima razdvojenih donjom crtom, a klase u PascalCase (konvencija imenovanja u kojoj se svaka reč u nazivu piše sa početnim velikim slovom), dok se konstante pišu velikim slovima. Ove konvencije doprinose jasnoći i doslednosti u strukturi koda.

Komentari i dokumentacija koriste se kada je potrebno objasniti nameru koda, a ne očiglednu funkcionalnost. Posebno je važno da komentari objašnjavaju razloge implementacije, dok se docstring koristi za formalni opis funkcija i njihovih ulaznih i izlaznih vrednosti.

```
def zbir(a, b):
    """Vraća zbir dva broja."""
    return a + b
```

Dobra praksa takođe podrazumeva princip da svaka funkcija treba da ima jednu jasno definisanu odgovornost, čime se povećava modularnost i olakšava testiranje. Izbegavanje dupliranja koda, poznato kao DRY (od engleskog *Don't Repeat Yourself*) princip, dodatno doprinosi kvalitetu implementacije.

Pored formalnih pravila, preporučuje se primena tzv. „Pythonic“ pristupa, koji podrazumeva korišćenje ugrađenih funkcija i idiomatskih konstrukcija jezika umesto ručnih implementacija, čime se postiže kraći, jasniji i efikasniji kod.

Sve navedene smernice zajedno doprinose stvaranju koda koji nije samo funkcionalan, već i dugoročno održiv, skalabilan i pogodan za timski rad.

Kojoj paradigmi pripada Pajton?

Pajton je prvobitno razvijen kao interpreterski i skriptni jezik, ali se tokom vremena razvio u višepardigmski programski jezik koji se koristi u širokom spektru softverskih sistema, od jednostavnih skripti do kompleksnih aplikacija.

Pajton kao skriptni jezik

Kada se kaže da je Pajton skriptni jezik, to znači da se programi najčešće izvršavaju direktno putem interpretera, bez klasične kompilacije u mašinski kod pre izvršavanja. Program se pokreće direktno nad izvorno napisanom datotekom.

```
python program.py
```

Skriptni jezici se najčešće koriste za automatizaciju zadataka, brzo prototipisanje, obradu podataka i povezivanje različitih softverskih sistema. Njihova osnovna karakteristika je jednostavnost upotrebe i visoka produktivnost programera.

Važno je naglasiti da Pajton nije ograničen samo na skriptni način rada. On je danas višepardigmski jezik koji podržava proceduralno, objektno-orijentisano, funkcionalno i modularno programiranje, kao i asinhrone modele izvršavanja.

Skriptni jezici su dizajnirani tako da omogućavaju **brzo pisanje programa** koji automatizuju konkretne zadatke. Za razliku od klasično kompajliranih jezika, oni se izvršavaju preko interpretera.

Pajton kod se najpre prevodi u bajtkod, koji se zatim izvršava u okviru virtuelne mašine. Ovaj proces omogućava kombinaciju interpretacije i internog prevođenja, čime se postiže dobra ravnoteža između jednostavnosti i performansi.

```
python program.py
```

Ovakav model izvršavanja omogućava brzo testiranje i iterativni razvoj koda, što je posebno važno u istraživačkom i aplikativnom programiranju.

Karakteristike Pajtona kao skriptnog jezika

Jedna od ključnih karakteristika Pajtona jeste **brzina razvoja**. Veliki broj zadataka može se realizovati sa relativno malim brojem linija koda, što ga čini pogodnim za automatizaciju i prototipisanje.

```
for i in range(5):
    print(i)
```

Pajton automatski upravlja memorijom pomoću garbage collector-a, što eliminiše potrebu za ručnim oslobađanjem resursa. Time se smanjuje kompleksnost razvoja i mogućnost grešaka.

Takođe, Pajton podržava interaktivni rad kroz REPL okruženje, koje omogućava trenutno izvršavanje izraza.

```
>>> 2 + 2
4
```

Ova karakteristika je posebno značajna u obrazovanju, analizi podataka i eksperimentalnom programiranju.

Pajton se često koristi i za automatizaciju sistemskih zadataka.

```
import os

for fajl in os.listdir("."):
    print(fajl)
```

Pajton kao proceduralni jezik

Pajton podržava proceduralni stil programiranja, gde se program sastoji od **sekvencijalnog** izvršavanja instrukcija organizovanih u funkcije.

```
def saberi(a, b):
    return a + b

rezultat = saberi(2, 3)
print(rezultat)
```

U ovom pristupu logika programa se izvršava korak po korak, bez potrebe za kompleksnim objektno-orijentisanim strukturama.

Proceduralni stil se najčešće koristi u jednostavnijim programima, skriptama i algoritamskim rešenjima.

Pajton kao objektno-orijentisani jezik

Pajton je snažno objektno-orijentisan jezik u kojem je gotovo sve objekat, uključujući brojeve, funkcije i module.

```
x = 5

print(type(x))
```

Osnovni koncepti uključuju klase, objekte, metode i atribute.

```
class Pas:

    def __init__(self, ime):

        self.ime = ime


    def oglasi_se(self):

        print("Av av")
```

Nasleđivanje omogućava proširenje postojećih klasa.

```
class Zivotinja:

    def disi(self):

        print("Dišem")
```

```
class Pas(Zivotinja):
    pass
```

Polimorfizam omogućava različite implementacije iste metode u različitim klasama.

```
class Macka:
    def zvuk(self):
        print("Mjau")

class Pas:
    def zvuk(self):
        print("Av")

def pusti_zvuk(zivotinja):
    zivotinja.zvuk()
```

Pajton kao funkcionalni jezik

Pajton podržava elemente funkcionalnog programiranja, uključujući **funkcije višeg reda i lambda izraze.**

```
kvadrat = lambda x: x * x
print(kvadrat(4))
```

Funkcije se mogu dodeljivati promenljivama i prosleđivati kao argumenti.

```
def pozdrav():
```

```
    print("Zdravo")


x = pozdrav

x()
```

Map i filter funkcije omogućavaju funkcionalni pristup obradi podataka.

```
brojevi = [1, 2, 3]


rezultat = list(map(lambda x: x * 2, brojevi))
```

Pajton kao modularni jezik

Pajton podržava organizaciju koda u module i pakete, što omogućava bolju strukturu i ponovnu upotrebu.

```
def saberi(a, b):
    return a + b


import matematika


print(matematika.saberi(2, 3))
```

Ovakav pristup je posebno važan u većim softverskim sistemima.

Dinamička tipizacija i duck typing

Pajton je dinamički tipiziran jezik, što znači da se tip promenljive određuje tokom izvršavanja.

```
x = 5

x = "tekst"
```

Duck typing je koncept u Pajtonu koji se zasniva na ideji da je pri radu sa objektima važnije šta objekat može da uradi (njegovo ponašanje), nego kojoj konkretnoj klasi ili tipu pripada. Drugim rečima, tip objekta nije primaran kriterijum; relevantno je da objekat poseduje potrebne metode ili atribute koji se koriste u datom kontekstu.

Naziv potiče od intuitivne analogije: „ako nešto izgleda kao patka i ponaša se kao patka, onda se tretira kao patka“. U programskom smislu, to znači da se kompatibilnost objekta ne proverava eksplicitnim proveravanjem tipa, već postojanjem odgovarajućeg interfejsa (metoda).

```
class Pas:
    def zvuk(self):
        print("Av")


class Macka:
    def zvuk(self):
        print("Mjau")
```

U ovom primeru, i `Pas` i `Macka` imaju metodu `zvuk()`, iako pripadaju različitim klasama. Sa stanovišta Pajtona, oba objekta su kompatibilna za korišćenje u kontekstu koji zahteva poziv metode `zvuk()`.

```
def pusti_zvuk(zivotinja):
    zivotinja.zvuk()
```

Funkcija `pusti_zvuk` ne vrši proveru tipa objekta koji joj se prosleđuje. Ona jednostavno pretpostavlja da objekat poseduje metodu `zvuk()` i poziva je. Ako objekat zaista ima tu metodu, kod će se ispravno izvršiti, bez obzira na njegovu klasu.

```
pusti_zvuk(Pas())
```

```
pusti_zvuk(Macka())
```

Suština duck typing principa je u tome da se greške ne otkrivaju kroz formalnu proveru tipova unapred, već u toku izvršavanja, ukoliko objekat ne podržava očekivano ponašanje. Ovo omogućava veću fleksibilnost i jednostavniji dizajn koda, ali zahteva pažljivije projektovanje i testiranje, jer ne postoji stroga statička kontrola tipova kao u nekim drugim programskim jezicima.

Zaključak

Pajton kao skriptni jezik označava njegovu sposobnost brzog izvršavanja programa kroz interpreter i jednostavnog razvoja bez klasične kompilacije. Međutim, Pajton istovremeno predstavlja višepardigmski programski jezik koji integriše proceduralni, objektno-orijentisani, funkcionalni i modularni pristup.

Njegova glavna snaga je u kombinaciji jednostavne sintakse, fleksibilnog modela izvršavanja i bogatog ekosistema biblioteka, što ga čini jednim od najpraktičnijih jezika u savremenom programiranju.

Zastupljenost i primena programskog jezika Pajton u praksi

Položaj programskog jezika Python na tržištu

Python predstavlja jedan od najzastupljenijih i najuticajnijih programskih jezika današnjice. Njegova popularnost kontinuirano raste zahvaljujući jednostavnoj sintaksi, velikom broju dostupnih biblioteka i širokoj primeni u različitim oblastima informacionih tehnologija.

Prema najpoznatijim indeksima popularnosti programskih jezika, kao što su TIOBE i PYPL, Python se tokom 2025. i 2026. godine nalazi na prvom mestu među programskim jezicima. Takođe, istraživanja sprovedena među programerima pokazuju da je Python jedan od najčešće korišćenih jezika u profesionalnom razvoju softvera.

Popularnost Pajtona može se objasniti sledećim karakteristikama:

- jednostavna i pregledna sintaksa;
- brz razvoj aplikacija;
- veliki broj gotovih biblioteka i razvojnih okruženja;
- podrška za više programskih paradigmi;
- primena u nauci, industriji i obrazovanju;
- dominantna uloga u oblastima veštačke inteligencije i analize podataka.

Oblasti primene Pajtona

Python se danas koristi u velikom broju oblasti razvoja softvera.

Veb programiranje

Python omogućava razvoj veb aplikacija korišćenjem okvira kao što su Django i Flask.

Primeri primene:

- internet prodavnice;
- informacioni sistemi;
- poslovne aplikacije;
- REST servisi i API sistemi.

Analiza podataka i naučno računarstvo

Python je postao standardni alat za obradu i analizu podataka.

Primeri primene:

- statistička analiza;
- obrada velikih skupova podataka;
- naučna istraživanja;
- simulacije i matematičko modelovanje.

Veštačka inteligencija i mašinsko učenje

Najveći rast popularnosti Python-a poslednjih godina povezan je sa razvojem veštačke inteligencije.

Primeri primene:

- sistemi za preporuke;
- prepoznavanje govora i slike;
- generativna veštačka inteligencija;
- neuronske mreže i duboko učenje.

Automatizacija i skriptovanje

Python se često koristi za automatizaciju svakodnevnih zadataka.

Primeri:

- obrada datoteka;
- automatsko slanje elektronske pošte;
- administracija servera;
- automatizovano testiranje softvera.

Obrazovanje

Zbog jednostavne sintakse Python je jedan od najčešće korišćenih jezika za učenje programiranja na univerzitetima i školama.

Studenti se mogu fokusirati na algoritme i strukture podataka bez dodatnog opterećenja složenom sintaksom karakterističnom za neke druge programske jezike.

Primeri upotrebe u industriji

Python koriste brojne velike kompanije za razvoj svojih proizvoda i servisa, među kojima su Google, Dropbox i Instagram. Python se koristi za razvoj serverskih aplikacija, obradu podataka, automatizaciju procesa i sisteme zasnovane na veštačkoj inteligenciji.

Najčešće korišćene Pajton biblioteke

Biblioteke predstavljaju skup gotovih funkcija i modula koji programerima omogućavaju brži razvoj aplikacija.

Biblioteka	Oblast primene	Tipični zadaci
NumPy	Numeričko računanje	Rad sa matricama i vektorima
Pandas	Analiza podataka	Obrada tabela i statistička analiza
Matplotlib	Vizuelizacija podataka	Grafikoni i dijagrami
SciPy	Naučno računarstvo	Optimizacija, integracija i statistika
Scikit-learn	Mašinsko učenje	Klasifikacija, regresija i grupisanje
TensorFlow	Duboko učenje	Neuronske mreže
PyTorch	Veštačka inteligencija	Istraživanja i razvoj AI modela
Flask	Veb programiranje	REST servisi i manje veb aplikacije
Django	Veb programiranje	Kompleksne veb aplikacije
Requests	Mrežna komunikacija	Slanje HTTP zahteva
BeautifulSoup	Veb scraping	Prikupljanje podataka sa veb stranica
OpenCV	Obrada slike	Prepoznavanje objekata i računarski vid

Python je jedan od najznačajnijih programskih jezika savremenog doba i trenutno zauzima vodeću poziciju na većini svetskih rang-lista popularnosti programskih jezika. Njegova jednostavnost, velika zajednica korisnika i bogat ekosistem biblioteka omogućili su široku primenu u različitim oblastima.

Upotreba Pajtona kao skript jezika u praksi (u odnosu na Beš)

Programski jezik Python se u savremenoj praksi često koristi kao skript jezik za automatizaciju različitih zadataka u oblasti sistemske administracije, obrade podataka i razvoja softverskih sistema. Iako je **Bash** tradicionalno primarni alat u Unix/Linux okruženjima za pisanje shell skripti, Python je u velikoj meri proširio domen skript programiranja zahvaljujući svojoj višem nivou apstrakcije, boljoj čitljivosti i bogatom ekosistemu biblioteka.

Bash kao klasični skript jezik

Bash predstavlja standardni shell skript jezik u Unix i Linux sistemima. Njegova osnovna namena je direktna interakcija sa operativnim sistemom kroz komandnu liniju.

Tipične oblasti primene Bash skripti uključuju:

- automatizaciju sistemskih komandi;
- upravljanje datotekama i direktorijumima;
- pokretanje i kontrolu procesa;
- ulančavanje sistemskih alata kroz *pipe* mehanizam;
- automatizaciju backup procedura i batch obrada.

Primer Bash skripte za obradu fajlova:

```
#!/bin/bash

for file in *.txt
do
  echo "Obrada fajla: $file"
  wc -l "$file"
done
```

Bash je posebno efikasan za kratke i direktne operacije nad operativnim sistemom, ali njegova sintaksa postaje složena i manje pregledna pri implementaciji kompleksnijih algoritamskih struktura.

Python kao skript jezik

Python se u praksi koristi kao skript jezik u širem spektru domena, posebno kada zadaci prevazilaze jednostavnu sistemsku automatizaciju. Njegova primena je značajna u slučajevima kada je potrebna:

- implementacija složenije poslovne logike;
- obrada strukturiranih i nestrukturiranih podataka;
- integracija sa web servisima i API interfejsima;
- rad sa bazama podataka i različitim formatima datoteka;
- razvoj automatizovanih sistema i alata višeg nivoa.

Za razliku od Bash-a, Python omogućava korišćenje objektno-orijentisanog i proceduralnog pristupa, što doprinosi boljoj organizaciji koda i skalabilnosti rešenja.

Primer Python skripte za obradu fajlova:

```
import os

for file in os.listdir("."):
    if file.endswith(".txt"):
        print(f"Obrada fajla: {file}")
```

Komparativna analiza Python-a i Bash-a kao skript jezika

Kriterijum	Bash	Python
Primarna namena	Sistemcka automatizacija	Opšta automatizacija i obrada podataka
Čitljivost koda	Ograničena kod složenih skripti	Visoka
Složenost logike	Ograničena	Visoko skalabilna
Rad sa strukturiranim podacima	Ograničen (alatima kao awk/sed)	Nativno podržan (JSON, CSV, XML)
Integracija sa web servisima	Ograničena	Standardizovana (requests, API biblioteke)
Prenosivost	Linux/Unix fokus	Višeplatformska
Ekosistem biblioteka	Mali	Veoma razvijen

Zaključak

U savremenoj praksi, Bash zadržava dominantnu ulogu u kontekstu niskonivoovske sistemske automatizacije i administracije operativnih sistema. Međutim, Python sve više preuzima ulogu univerzalnog skript jezika u situacijama koje zahtevaju veću fleksibilnost, kompleksniju logiku i rad sa podacima.

Zbog svoje jednostavne sintakse, bogatog ekosistema biblioteka i visoke prenosivosti, Python se često koristi kao zamena ili dopuna Bash skriptama, posebno u profesionalnim okruženjima gde skripte evoluiraju u kompleksne automatizovane sisteme.

Uloga i pozicioniranje programskog jezika Python u savremenom softverskom inženjerstvu

Dualnost Pythona kao skript jezika i arhitekturalne komponente velikih sistema

U modernoj računarskoj nauci, klasifikacija programskih jezika često se vrši na osnovu njihove primarne namene, načina izvršavanja i nivoa apstrakcije. Programski jezik **Python** zauzima specifičnu i visokofunkcionalnu poziciju na ovom spektru: on je istovremeno i **visokonivojski skript jezik** i **robustan opštenamenski jezik** (engl. *general-purpose language*) sposoban za pokretanje enterprise sistema.

Da bi se razumela njegova široka tržišna primena, neophodno je dekonstruisati pojam „skript jezika". Tradicionalno, skript jezici su korišćeni za automatizaciju operativnih zadataka, manipulaciju tekstualnim datotekama i povezivanje nezavisnih softverskih komponenti. Python zadržava ove fundamentalne prednosti skriptovanja:

- **Interpretatorski režim i dinamička tipizacija:** Omogućavaju brz ciklus razvoja (engl. *REPL - Read-Eval-Print Loop*), gde se kôd izvršava direktno bez potrebe za eksplicitnim, dugotrajnim procesom kompilacije.
- **Sintaksna konciznost i ekspresivnost:** Smanjuju kognitivno opterećenje programera, omogućavajući fokus na semantiku problema, a ne na sintaksni „boilerplate" kôd (karakterističan za jezike poput C++ ili Jave).

Međutim, evolucija Pythona ga je transformisala u tzv. **„lepak jezik" (engl. *glue language*)**. Python ne pokušava da se takmiči sa jezicima niskog nivoa u pogledu sirove brzine izvršavanja na procesoru. Umesto toga, on koristi svoju skript arhitekturu kako bi obezbedio jednostavan interfejs (omotač ili engl. *wrapper*) za kompleksne, visokooptimizovane module napisane u jezicima C, C++ ili Fortran. Na taj način, programer piše kôd u visokoapstraktnom, skript okruženju, dok se računski zahtevne operacije izvršavaju nativnom brzinom kompajliranog koda.

Praktična demonstracija dualnosti kroz kôd

Razlika između mikro-skriptovanja (automatizacije) i makro-primene (napredne analitike i integracije) se može videti u nastavku gde su navedeni reprezentativni primeri koda.

Primer 1: Moć skriptovanja – Automatizacija mrežne integracije (Web Scraping i Notifikacija)

Ovaj primer demonstrira kako se u svega nekoliko linija koda može rešiti kompleksan problem slanja HTTP zahteva, parsiranja nestrukturiranih HTML podataka i integracije sa eksternim API sistemima.

```
import requests

from bs4 import BeautifulSoup

import smtplib

from email.mime.text import MIMEText


# 1. Ekstrakcija podataka sa ciljne web stranice (Web Scraping)

url = "https://api.vremenskaprognoza.rs/beograd"

response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

temperatura = soup.find("span", {"class": "temp"}).text


# 2. Logika za evaluaciju uslova

poruka_tela = f"Upozorenje: Trenutna temperatura u Beogradu iznosi {temperatura} stepeni celzijusa."


# 3. Automatizovano slanje e-mail obaveštenja putem SMTP protokola

msg = MIMEText(poruka_tela)

msg['Subject'] = 'Automatski vremenski izveštaj'

msg['From'] = 'skripta@matf.bg.ac.rs'

msg['To'] = 'student@matf.bg.ac.rs'


with smtplib.SMTP('smtp.matf.bg.ac.rs', 25) as server:

    server.login("korisnik", "lozinka")

    server.send_message(msg)
```

U jezicima poput C++ ili Jave, implementacija mrežnih soketa, parsiranje HTML stabla i konfiguracija SMTP klijenta zahtevala bi stotine linija koda, definisanje klasa i upravljanje memorijom. U Pythonu, apstrakcija je toliko visoka da se proces završava u okviru jednog linearnog skripta.

Primer 2: Široka primena – Napredna analiza podataka i vizuelizacija

Ovaj primer ilustruje kako se Python pozicionira u nauci o podacima (Data Science). Korišćenjem biblioteke pandas, koja u pozadini koristi visokooptimizovani C kôd, vrši se manipulacija nad velikim setovima podataka (Big Data).

```
import pandas as pd

import matplotlib.pyplot as plt



# 1. Učitavanje eksternog skupa podataka (npr. milioni zapisa o Spotify strimovima)

data = pd.read_csv("spotify_global_tracks_2026.csv")



# 2. Agregacija i filtriranje podataka u jednoj liniji koda

# Grupisanje po žanru, računanje prosečne popularnosti i selekcija top 5 žanrova

top_genres =
data.groupby('genre')['popularity'].mean().sort_values(ascending=False).head(5)



# 3. Deskriptivna vizuelizacija rezultata

top_genres.plot(kind='bar', color='skyblue')

plt.title('Top 5 muzičkih žanrova po indeksu popularnosti')

plt.xlabel('Žanr')

plt.ylabel('Prosečna popularnost')

plt.grid(axis='y', linestyle='--')



# Rendering grafikona

plt.show()
```

Ovaj primer pokazuje zašto je Python postao de facto standard u industriji podataka. Operacija koja zahteva kompleksne SQL upite ili map-reduce algoritme ovde se izvršava nad strukturiranim DataFrame objektom sa ekstremno visokom čitljivošću koda.

Analiza primene u ključnim industrijama

Pozicija Pythona na tržištu rada direktno je korelisana sa njegovom dominacijom u tri strateška tehnološka sektora.

Veštačka inteligencija i mašinsko učenje (AI & Machine Learning)

Sektor veštačke inteligencije predstavlja najznačajnije uporište Python jezika. Gotovo svi savremeni sistemi zasnovani na dubokom učenju (Deep Learning) i velikim jezičkim modelima (LLM - *Large Language Models*) razvijeni su u Python ekosistemu.

Organizacije kao što su **OpenAI** (tvorci ChatGPT-ja) i **Google DeepMind** koriste Python kao primarni interfejs za treniranje i evaluaciju modela. Iako se samo izračunavanje tenzora i operacije na grafičkim procesorima (GPU) vrše na najnižem nivou preko CUDA (C++) drajvera, celokupna logika arhitekture mreže, upravljanje podacima i API slojevi napisani su u Pythonu kroz biblioteke **PyTorch** i **TensorFlow**.

Svemirska istraživanja i astrofizika (Aerospace & Astrophysics)

U sektorima gde su preciznost, matematičko modeliranje i obrada masovnih naučnih podataka od kritičkog značaja, Python je uspešno zamenio tradicionalne jezike poput Fortran-a i MATLAB-a.

Projekat Event Horizon Telescope (Fotografisanje crne rupe): Godine 2019, kada je čovečanstvo dobilo prvu sliku crne rupe u galaksiji M87, algoritam koji je izvršio interferometrijsku rekonstrukciju podataka sa osam teleskopa širom sveta bio je napisan pretežno u Pythonu. Količina sirovih podataka merila se petabajtima, a biblioteke **NumPy** i **SciPy** su iskorišćene za implementaciju brze Furijeove transformacije (FFT) i filtriranje šuma iz kosmičkog zračenja.

NASA i ESA (Evropska svemirska agencija): Ove agencije koriste Python u okviru svojih kontrolnih sistema za automatizaciju provere telemetrije sa satelita, planiranje putanja rovera (poput *Perseverance* misije) i analizu spektralnih podataka koji pristižu sa svemirskog teleskopa *James Webb*. Python omogućava naučnicima koji nisu primarno softverski inženjeri da pišu validne naučne skripte za interpretaciju kosmoloških fenomena.

Gejming industrija i računarska animacija (Gaming & Computer Animation)

Iako se sami grafički endžini video-igara (poput Unreal Engine-a) pišu u C++ jeziku zbog hardverskih ograničenja i renderinga u realnom vremenu, prateći ekosistem i produkcija u velikoj meri zavise od Pythona.

Automatizacija Pipeline-a u Holivudu (Pixar, Industrial Light & Magic): U modernoj 3D animaciji i vizuelnim efektima (VFX), vreme je kritičan resurs. Softverski paketi kao što su *Autodesk Maya*, *Houdini* i *Blender* imaju integrisan Python interpretator. Inženjeri tehničke režije (TD - *Technical Directors*) koriste Python skripte za automatizaciju prenosa hiljada 3D modela kroz različite faze produkcije, upravljanje farmama za renderovanje (render farms) i proceduralno generisanje kompleksnih tekstura ili simulacija čestica (voda, vatra, kosa).

Logika igara i backend (Battlefield, EVE Online): U razvoju kompleksnih igara, Python se često koristi za definisanje logike ponašanja likova (AI agenata u igri), sistema ekonomije unutar igre (ingame economy) i za orkestraciju servera za više igrača (multiplayer backend), gde njegova skript priroda omogućava dizajnerima igara da menjaju pravila igre bez potrebe za ponovnim kompajliranjem celog grafičkog koda.

Zašto industrija bira Pajton?

Tržišni uspeh Pythona može se sumirati kroz koncept **„Time-to-Market" (Vreme plasiranja na tržište)**. U savremenom poslovanju, cena vremena programera je često viša od cene hardverskih resursa. Ukratko, Pajton omogućava kompanijama da:

1. Razviju funkcionalni prototip (MVP - *Minimum Viable Product*) brže od bilo kog konkurentskog jezika.
2. Lako skaliraju sistem dodavanjem C-ekstenzija ili migracijom na cloud arhitekture.
3. Zaposle inženjere koji mogu raditi na različitim pozicijama – od automatizacije infrastrukture (DevOps), preko backend razvoja (Django, FastAPI), do analize podataka i veštačke inteligencije.

Možemo posmatrati Pajton kao skript jezik koji poseduje industrijsku snagu.