# Нове унапређене верзије програма „Ка минималним паровима”

Научни рад DOI: 10.18485/judig.2025.1.ch22

*Данило Алексић*[1], *0000-0002-1478-7487*

## Апстракт

Рад је посвећен новим верзијама једног кратког програма на Python-у за аутоматску ексцерпцију сегменталних минималних парова и парова сродних са сегменталним минималним паровима. (Сегменталним минималним паровима сматрају се они парови речи чији се чланови фонолошки разликују само по једном пару [сегменталних] фонема [односно по једном пару {сегменталних} фонемских низова или једном пару сачињеном од једне {сегменталне} фонеме и једног {сегменталног} фонемског низа] само на једној позицији [нпр. *е* и *и* у *лéк* ~ *лúк* {и *овн* и *ск* у *свѐтōвнā* ~ *свѐтскā*}] или на више позиција [нпр. *л* и *р* у *мо̀лā̄р* ~ *мо̀рā̄л* {и *аз* и *ин* у *кā̀зāз* ~ *кѝнѭн*}]. Паровима сродним са сегменталним минималним паровима сматрају се они парови речи чији се чланови фонолошки разликују само по прозодији и по једном пару [сегменталних] фонема [односно само по прозодији и по једном пару {сегменталних} фонемских низова или једном пару сачињеном од једне {сегменталне} фонеме и једног {сегменталног} фонемског низа] само на једној позицији [нпр. *ћ* и *ч* у *прѐгледаће* ~ *прегледа́че* {и *ац* и *ер* у *дестила́ција* ~ *дестилѐрија*}] или на више позиција [нпр. *е* и *и* у *бе̏лē* ~ *би́ли* {и *ин* и *ос* у *лѝнѭн* ~ *лӧсос*}; уп. Алексић & Шандрих, 2021, pp. 569–570].) У најбржој досадашњој верзији овог програма (в. Алексић & Мркела, 2022) сегментални минимални парови и парови сродни са сегменталним минималним паровима образовани су помоћу речника, стандардног мапирајућег типа („standard mapping type”; Python

[1] Катедра за српски језик са јужнословенским језицима, Универзитет у Београду – Филолошки факултет, danilo.aleksic@fil.bg.ac.rs

Software Foundation, 2025b), помоћу ког је и токенизован улазни корпус. Нове, тренутно најбрже верзије образују парове који су посреди и токенизују улазни корпус помоћу класа, тако што се као речнички кључеви користе називи класовних атрибута, а као речничке вредности – вредности класовних атрибута.

Кључне речи: језик, класе, минимални парови, оптимизација, речници, Python.

## 1. Увод

У свим својим верзијама, програм „Ка минималним паровима” или, скраћено, КаМП (Алексић & Мркела, 2022; Алексић & Шандрих, 2021) упарује токене латиничког српског[2] улазног корпуса који се, сведени на мала слова, разликују само по задатим поднискама. Добијају се парови типа `"znače ~ znaće"` (ако су задате подниске `"č"` и `"ć"`), `"džak ~ đak"` (ако су задате подниске `"dž"` и `"đ"`) и `"Harlemovskom ~ harlemskom"` (ако су задате подниске `"ovsk"` и `"sk"`), какви се могу користити у настави српског као страног језика (Алексић & Мркела, 2022, p. 8; Алексић & Шандрих, 2021), у логопедији (уп. нпр. Storkel, 2022) и у лингвистици (Алексић & Мркела, 2022, p. 8; Алексић & Шандрих, 2021).

Прва верзија КаМП-а токене упарује тако што у улазном корпусу, помоћу регуларног израза, тражи токене са првом задатом подниском, замењује прву задату подниску другом задатом подниском и добијене ниске тражи у истом улазном корпусу (Алексић & Шандрих, 2021), што, показало се, није најбржи начин. Уз то, регуларни израз постаје доста сложен када му одговара дужа подниска, нпр.:

```
"(?<![A-Za-zĆ-ž])([A-Za-zĆ-ž-]*(IZACIJ|IZACIj|"
"IZACiJ|IZACij|IZAcIJ|IZAcIj|IZAciJ|IZAcij|"
"IZaCIJ|IZaCIj|IZaCiJ|IZaCij|IZacIJ|IZacIj|"
"IZaciJ|IZacij|IzACIJ|IzACIj|IzACiJ|IzACij|"
"IzAcIJ|IzAcIj|IzAciJ|IzAcij|IzaCIJ|IzaCIj|"
"IzaCiJ|IzaCij|IzacIJ|IzacIj|IzaciJ|Izacij|"
"iZACIJ|iZACIj|iZACiJ|iZACij|iZAcIJ|iZAcIj|"
"iZAciJ|iZAcij|iZaCIJ|iZaCIj|iZaCiJ|iZaCij|"
"iZacIJ|iZacIj|iZaciJ|iZacij|izACIJ|izACIj|"
```

[2] КаМП-ови су предвиђени за корпус кодиран схемом UTF-8 (уп. Алексић & Мркела, 2022, p.7). Тест прве верзије КаМП-а на тексту кодираном схемом ISO-8859-1 није био успешан (Алексић & Шандрих, 2021, p. 572). КаМП-ови корпус процесирају површно, само на нивоу писма (без прозодијских и изговорних репрезентација). Прилагођени КаМП 2.3 успешно је тестиран на турском тексту.

```
"izACiJ|izACij|izAcIJ|izAcIj|izAciJ|izAcij|"
"izaCIJ|izaCIj|izaCiJ|izaCij|izacIJ|izacIj|"
"izaciJ|izacij)[A-Za-zĆ-ž-]*)(?![A-Za-zĆ-ž])"
```

према поднисци `"izacij"`, и драстично успорава ионако трому прву верзију.

Брже су верзије 2 и 2.1 (Алексић & Мркела, 2022). (Верзију 2 је написао Данило Алексић, а упаривање токена у верзији 2.1 решење је Лазара Мркеле.) Оне кроз улазни корпус пролазе само једном, вадећи токене који садрже прву задату подниску, другу задату подниску или обе задате подниске. Задате подниске се у токенима записаним малим словима замењују специјалним карактером, `"◩"`. На пример, ако су задате подниске `"ovsk"` и `"sk"`, нисци `"Harlemovskom"`, пошто се задате подниске преклапају, одговарају две поредбене ниске – `"harlemov◩om"` и `"harlem◩om"` – а нисци `"harlemskom"` одговара једна поредбена ниска: `"harlem◩om"`. Главни део верзије 2 јесте Декартов производ торки које садрже токен са првом задатом подниском и торки које садрже токен са другом задатом подниском, нпр. Декартов производ торки `("ČAK", "čak", "◩ak")` и `("vičan", "vičan", "vi◩an")` на једној и торки `("Đak", "đak", "◩ak")` и `("viđan", "viđan", "vi◩an")` на другој страни, дакле нпр.:

```
{
    (
        ("ČAK", "čak", "◩ak"),
        ("Đak", "đak", "◩ak")
    ),
    (
        ("ČAK", "čak", "◩ak"),
        ("viđan", "viđan", "vi◩an")
    ),
    (
        ("vičan", "vičan", "vi◩an"),
        ("Đak", "đak", "◩ak")
    ),
    (
        ("vičan", "vičan", "vi◩an"),
        ("viđan", "viđan", "vi◩an")
    )
}
```

ако су задате подниске "č" и "ć" и ако нема других токена са задатим поднискама. У верзији 2 се из израчунатог Декартовог производа узимају међусобно различити токени које прате исте поредбене ниске. Главни део верзије 2.1 јесте речник (в. одељак „Новине у КаМП-у 2.2 и у КаМП-у 2.3") у чијим су вредностима токени са другом задатом подниском, а чији су кључеви поредбене ниске. У верзији 2.1 међу тим кључевима траже се поредбене ниске токена са првом задатом подниском. И верзија 2 и верзија 2.1 улазни корпус токенизују помоћу речника. У верзије 2 и 2.1 уведен је и мод у којем се не занемарују разлике између великих и малих слова код ексцерпираних токена (прва верзија је дотичне разлике игнорисала).

У одељцима који следе представља се један тип могућности, оличен у КаМП-у 2.2 и КаМП-у 2.3, да се до сада најбржи КаМП додатно убрза.

Сви КаМП-ови су писани на програмском језику Python и за потребе овог истраживања покретани на његовим верзијама 3.8.2 и 3.13.5.

## 2. Зашто нове верзије?

У једној књизи о Python-у срећу се следеће смернице:

> Now that you've made your code work, and possibly made it better than the initial version, it may be time to make it faster. Then, again, it may not. One very important rule (along with such principles as KISS = Keep It Small and Simple, or YAGNI = You Ain't Gonna Need It) that you should heed when tempted to fiddle with your code to speed it up:
>
> *Premature optimization is the root of all evil.*
>
> — Donald Knuth, paraphrasing C. A. R. Hoare
>
> Another way of stating this, in the words of Ken Thompson, co-inventor of UNIX, is "When in doubt, use brute force." In other words, don't worry about fancy algorithms or clever optimization tricks if you don't really, really need them. If the program is fast enough, chances are that the value of clean, simple, understandable code is much higher than that of a slightly faster program. After all, in a few months, faster hardware will probably be available anyway.

> But if you do need to optimize your program, because it simply isn't fast enough for your requirements, you absolutely should profile it before doing anything else. That is because it's really hard to guess where the bottlenecks are, unless your program is really simple. And if you don't know what's slowing down your program, chances are you'll be optimizing the wrong thing. (Hetland, 2008, pp. 362–363)

У другој се каже:

> This chapter covers the subjects in the natural order in which they occur in development: testing first and foremost, debugging next, and optimizing last. Most programmers' enthusiasm focuses on optimization: testing and debugging are often (wrongly, in our opinion) perceived as being chores, while optimization is seen as being fun. … the Pythonic approach to optimization—close to Jackson's classic "Rules of Optimization: Rule 1: Don't do it. Rule 2 (for experts only): Don't do it *yet*." (Martelli et al., 2017, p. 453)

У трећој: „Optimization is the altar where maintainability is sacrificed" (Ramalho, 2015, p. 91). Из перспективе датих цитата, КаМП 2.2 и КаМП 2.3 резултат су неодговорног располагања ресурсима. Наиме, (1) већ је постојала задовољавајуће ефикасна верзија, КаМП 2.1, (2) Лазар Мркела је као проблем у вези са брзином КаМП-а 2 који треба најпре решити препознао споро учитавање корпуса[3], (3) исти аутор је као први корак у убрзавању КаМП-а 2 предложио додавање опције за креирање речника од корпуса који би се сачувао на диску, а као следећи могући корак поменуо паралелизацију претраге (Алексић & Мркела, 2022, p. 19; како се види из одељка „Новине у КаМП-у 2.2 и у КаМП-у 2.3", КаМП 2.2 и КаМП 2.3 не остварују те циљеве) и (4) начин на који је у КаМП-у 2.2 и КаМП-у 2.3 постигнуто убрзање није осмишљен на основу профилисања.

У одговор би се могао цитирати Рејмонд Хетинџер: „There must be a better way!" (нпр. PyData, 2023, 12:03; SF Python, 2016, 9:21). Аутори који помажу програмерима да буду бољи у свом послу не сносе ризике преране и непотребне оптимизације који сносе програмери при развијању кода у некој софтверској компанији. Ако је контекст предавање о Python-у, сасвим је очекивано да се промовишу ефикаснији и лепши приступи. На пример, Хетинџер је током једног предавања о Python-у (Next Day Video, 2013, 12:50) питао публику како треба конкатенирати ниске и како ниске не треба конкатенирати иако и тај други, непрепоручени приступ постиже исти циљ. Из

[3]КаМП-ови корпус учитавају у деловима чију величину и сепараторе бира корисник (Алексић & Мркела, 2022, pp. 15, 20–21; Алексић & Шандрих, 2021, pp. 572, 581).

перспективе учења, или из академске перспективе, израда бржих верзија КаМП-а и писање чланка о њима ипак нису губљење времена.

## 3. Новине у КаМП-у 2.2 и у КаМП-у 2.3

Кôд КаМП-а 2.2 и кôд КаМП-а 2.3 биће објављени на сајту аутора (Алексић, n.d.).

Главна разлика између КаМП-а 2.2 и КаМП-а 2.3 с једне и КаМП-а 2.1 с друге стране јесте употреба класа уместо неких речника. Уместо кључева и вредности, које имају речници, употребљавају се називи класовних атрибута, односно вредности класовних атрибута:

```
rečnik[reč] = reč.casefold()
"""Из токенизације КаМП-а 2.1."""

setattr(self, reč, reč.casefold())
"""Из токенизације КаМП-а 2.3."""
```

Према Бизлију (2006, p. 143), `setattr(`*`object`*`, `*`name`*`, `*`value`*`)` исто је што и *`object`*`.`*`name`*` = `*`value`*. Откуда онда избор уграђене функције `setattr()`, односно уграђених функција `getattr()` и `hasattr()` (уп.:

```
class Rečnik2:
    def __init__(self):
        for torka_ in torka_2[0]:
            if not hasattr(
                    self, torka_[0]):
                setattr(
                    self, torka_[0], {})
            getattr(self, torka_[0])[
                torka_[1]] = torka_[2]
```

), уместо нотације тачком (нпр. `self.reč = reč.casefold()`)? Разлог да се тако поступи можда добро формулише Лами (2021, p. 43): „the … functions are used to manipulate the attributes of an object when the name of the attribute is not known at the time of writing the program, but is available during execution in a variable (as a string)” (а КаМП-ови раде на корпусу који бира корисник). Заправо, идеја да се уместо нотације тачком одаберу уграђене функције `getattr()`, `hasattr()` и `setattr()` потиче са сајта „Stack Overflow”, али је он непогодан за цитирање јер није „ауторитативан извор”. Уп.:

```
"""Циљ је упарити све речи из листе
са њиховим верзијама добијеним
помоћу метода casefold().
"""

class Test1:
    def __init__(self):
        for reč in [
            "Један", "ДВА", "ТРИ"]:
            self.reč = reč.casefold()

test1 = Test1()
print(vars(test1))
"""Излаз: {'reč': 'три'}."""

class Test2:
    def __init__(self):
        for reč in [
            "Један", "ДВА", "ТРИ"]:
            setattr(
                self, reč,
                reč.casefold())

test2 = Test2()
print(vars(test2))
"""Излаз: {'Један': 'један',
'ДВА': 'два', 'ТРИ': 'три'}.
"""
```

КаМП 2.2 и КаМП 2.3 разликују се по декларацији `__slots__` (Goodrich et al., 2013, p. 99):

```
class Rečnik:  # КаМП 2.2
    tokeni_ = tokenizacija()
    __slots__ = tokeni_

    def __init__(self):
        for reč in self.tokeni_:
            setattr(
                self, reč,
                reč.casefold())

class Rečnik:  # КаМП 2.3
    def __init__(self):
        for reč in tokenizacija():
            setattr(
                self, reč,
                reč.casefold())
```

Као што се у речницима српског језика, нпр. у „Речнику српскохрватског књижевног и народног језика” (Институт за српск[охрватск]и језик, 1959–present), дефиниције упарују са лемама (уп. Heisler et al., 2021, p. 263), при чему је свака лема јединствена (ниједна се не дефинише два или више пута [нити иједна више пута упућује на другу лему или друге леме]), у Python-овим речницима похрањују се парови по једног кључа и по једне вредности, при чему је сваки кључ у датом речнику јединствен (Shovic & Simpson, 2021, p. 172). (Додуше, кључеви се у Python-овом речнику не могу сортирати, него се нижу по редоследу увођења у речник [Python Software Foundation, 2025b].) На пример:

```
književnici_mesto_rođenja = {
    "Данило Киш": "Суботица",
    "Иво Андрић": "Долац",
    "Милорад Павић": "Београд",
    "Александар Вучо": "Београд"
    }
```

У речнику `književnici_mesto_rođenja` кључеви су имена књижевника, а вредности су имена насеља.

Класа је у Python-у средство апстракције – врста објекта и нацрт, шаблон или фабрика за генерисање других објеката. Објекат је пак скуп података заједно са повезаним понашањима (Goodrich et al., 2013, p. 69; Hetland, 2008, p. 147; Lott & Phillips, 2021, p. 4; Lutz, 2007, p. 465; Parker, 2021, p. 220). У Python-у је сваки податак објекат (Brueck & Tanner, 2001, p. 14). И саме класе су објекти (Reitz & Schlusser, 2016, p. 66). Сви објекти припадају некој класи и зову се примерци (instances) те класе (Hetland, 2008, p. 147). Понашања објекта која су добро позната, или функције које припадају класи, зову се методи (Norton et al., 2005, p. 80; Parker, 2021, p. 220). На пример, речник има метод `clear()`, који брише све парове кључева и вредности у речнику (Python Software Foundation, 2025b). Променљиве које су повезане са објектима зову се атрибути (Sweigart, 2021, p. 278). Класа се може замислити као формулар или упитник (Heisler et al., 2021, p. 281), а атрибути као питања у њему. На пример (в. Shovic & Simpson, 2021, p. 222):

```
class Član:
    def __init__(self, k_ime, p_ime):
        self.korisničko_ime = k_ime
        self.puno_ime = p_ime
```

У класи Član атрибути су korisničko_ime и puno_ime. Помоћу класе Član може се начинити нови објекат: novi_član = Član("petar1502", "Петар Јовановић").

Класи је обично могуће накнадно додати арбитраран атрибут (в. Martelli et al., 2017, p. 122). На пример, објекту `novi_član` може се додати година рођења: `novi_član.godina_rođenja = 1990`. Међутим, објекту `novi_član` накнадно дòдати арбитраран атрибут није могуће ако је класа `Član` дефинисана овако:

```
class Član:
    __slots__ = (
        "korisničko_ime",
        "puno_ime")

    def __init__(self, k_ime, p_ime):
        self.korisničko_ime = k_ime
        self.puno_ime = p_ime
```

Класовни атрибут `__slots__` серија је ниски које ће класа прихватити као називе атрибута (в. Beazley, 2006, p. 94; Martelli et al., 2017, p. 122). У одређеним случајевима употребом атрибута `__slots__` могу се уштедети меморија и време (Python Software Foundation, 2025a; Ramalho, 2015, pp. 264–267).

## 4. Време извршавања КаМП-а 2.2 и КаМП-а 2.3

Скоро исто као у раду Данила Алексића и Лазара Мркеле, „[б]рзина је мерена у Python-у 3.8.2” и 3.13.5, „на Manjaro Linux-у, рачунаром са процесором i5-11600K и два DDR4-3200 CL16 SDRAM-а од по 16 GB и на корпусу `POL`, који” (Алексић & Мркела, 2022, p. 17) „броји око 117.900.900 речи из 223.308 текстова са сајта *Политика*” (Алексић & Шандрих, 2021, p. 575; в. Табелу 1 и Табелу 2).

*Табела 1*

*Време извршавања КаМП-а 2.2 и КаМП-а 2.3 (у секундама)*

| Задате подниске | Верзија Python-а | Време у секундама (просек 10 сукцесивних мерења) | | | | |
|---|---|---|---|---|---|---|
| | | КаМП | КаМП 2 (мод А) | КаМП 2.1 (мод А) | КаМП 2.2 (мод А) | КаМП 2.3 (мод А) |
| "ac", "aca" | 3.8.2 | 182 | 150 | 67 | 59 | 58 |
| | 3.13.5 | 154 | 116 | 63 | 54 | 51 |
| "č", "ž" | 3.8.2 | 676 | 398 | 67 | 61 | 57 |
| | 3.13.5 | 589 | 280 | 64 | 55 | 52 |
| "lac", "telj" | 3.8.2 | 187 | 68 | 67 | 61 | 57 |
| | 3.13.5 | 163 | 65 | 63 | 54 | 51 |
| "š", "ž" | 3.8.2 | 646 | 382 | 67 | 61 | 58 |
| | 3.13.5 | 608 | 266 | 64 | 54 | 52 |
| "z", "ž" | 3.8.2 | 1669 | 912 | 67 | 62 | 57 |
| | 3.13.5 | 1536 | 579 | 63 | 55 | 51 |
| "ž", "ža" | 3.8.2 | 165 | 147 | 67 | 59 | 57 |
| | 3.13.5 | 136 | 113 | 63 | 54 | 50 |

*Табела 2*

*Време извршавања КаМП-а 2.2 и КаМП-а 2.3 (у процентима)*

| Задате подниске | Верзија Python-а | Проценат најдужег времена у датом реду (према подацима из Табеле 1) | | | | |
|---|---|---|---|---|---|---|
| | | КаМП | КаМП 2 (мод А) | КаМП 2.1 (мод А) | КаМП 2.2 (мод А) | КаМП 2.3 (мод А) |
| "ac", "aca" | 3.8.2 | 100 | 82,42 | 36,81 | 32,42 | 31,87 |
| | 3.13.5 | 100 | 75,32 | 40,91 | 35,06 | 33,12 |
| "č", "ž" | 3.8.2 | 100 | 58,88 | 9,91 | 9,02 | 8,43 |
| | 3.13.5 | 100 | 47,54 | 10,87 | 9,34 | 8,83 |
| "lac", "telj" | 3.8.2 | 100 | 36,36 | 35,83 | 32,62 | 30,48 |
| | 3.13.5 | 100 | 39,88 | 38,65 | 33,13 | 31,29 |
| "š", "ž" | 3.8.2 | 100 | 59,13 | 10,37 | 9,44 | 8,98 |
| | 3.13.5 | 100 | 43,75 | 10,53 | 8,88 | 8,55 |
| "z", "ž" | 3.8.2 | 100 | 54,64 | 4,01 | 3,71 | 3,42 |
| | 3.13.5 | 100 | 37,7 | 4,1 | 3,58 | 3,32 |
| "ž", "ža" | 3.8.2 | 100 | 89,09 | 40,61 | 35,76 | 34,55 |
| | 3.13.5 | 100 | 83,09 | 46,32 | 39,71 | 36,76 |

## 5. Коментар о утицају класовног атрибута `__slots__` на време извршавања

Табела 1 и Табела 2 показују да дефинисање класовног атрибута `__slots__` не само да не убрзава КаМП 2.2 него га, у односу на КаМП 2.3, и успорава. Узрок одсуству убрзања највероватније је то што је број примерака класа по један. Наиме, класовни атрибут `__slots__` вреди додати само оним класама које имају толико примерака да је важна уштеда неколико десетина бајтова по примерку, а то су типично класе које имају милионе примерака „живих" у исто време (Martelli et al., 2017, p. 122). Велики објекти, какви могу бити објекти у КаМП-у 2.2 и у КаМП-у 2.3, ову малу уштеду чине ирелевантном (в. Reitz & Schlusser, 2016, p. 122). Употреба атрибута `__slots__` убрзала је Рамаљов (2015, pp. 265–266) скрипт који креира 10.000.000 примерака („*mem_test.py creates 10 million Vector2d instances using the class defined in the named module (e.g., vector2d_v3.py)*").

## 6. Закључак

Најбрже верзије КаМП-а тренутно су КаМП 2.2 и КаМП 2.3, у којима се уместо речника за токенизацију и речника за упаривање по поредбеној нисци користе класе (и функције `getattr()`, `hasattr()` и `setattr()`); кључевима речника одговарају називи класовних атрибута, а вредностима речника одговарају вредности класовних атрибута.

Верзија КаМП-а са класама у којима се дефинише класовни атрибут `__slots__` (2.2) бржа је од верзије са речницима (2.1), а спорија од верзије са класама у којима се тај атрибут не дефинише.

## Референце

[1] Алексић, Д. (n.d.). *Препоручене верзије неких објављених радова*. Данúло Алексић. Retrieved July 31, 2025, from https://daleksic.rs/preporucene_verzije/

[2] Алексић, Д., & Мркела, Л. (2022). Унапређене верзије програма „Ка минималним паровима". *Инфотека*, *22*(1), 7–31. https://infoteka.bg.ac.rs/ojs/index.php/Infoteka/article/view/2022.22.1.1_sr

[3] Алексић, Д., & Шандрих, Б. (2021). Аутоматска ексцерпција парова речи за учење изговора у настави српског као страног језика. *Српски језик*, *26*(1), 567–584. https://doi.org/10.18485/sj.2021.26.1.32

[4] Институт за српск(охрватск)и језик. (1959–present). *Речник српскохрватског књижевног и народног језика* (Vols. I–XXII).

[5] Beazley, D. M. (2006). *Python: Essential reference* (3rd ed.). Sams Publishing.

[6] Brueck, D., & Tanner, S. (2001). *Python 2.1 bible*. Hungry Minds.

[7] Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). *Data structures and algorithms in Python*. John Wiley & Sons.

[8] Heisler, F., Amos, D., Bader, D., & Jablonski, J. (2021). *Python basics: A practical introduction to Python 3* (4th ed.). Real Python.

[9] Hetland, M. L. (2008). *Beginning Python: From novice to professional* (2nd ed.). Apress.

[10] Lamy, J.-B. (2021). *Ontologies with Python: Programming OWL 2.0 Ontologies with Python and Owlready2*. Apress. https://doi.org/10.1007/978-1-4842-6552-9

[11] Lott, S. F., & Phillips, D. (2021). *Python object-oriented programming: Build robust and maintainable object-oriented Python applications and libraries* (4th ed.). Packt Publishing.

[12] Lutz, M. (2007). *Learning Python* (3rd ed.). O'Reilly Media.

[13] Martelli, A., Ravenscroft, A., & Holden, S. (2017). *Python in a nutshell: The definitive reference* (3rd ed.). O'Reilly Media.

[14] Next Day Video. (2013, March 21). *Transforming code into beautiful, idiomatic Python* [Video]. YouTube. https://www.youtube.com/watch?v=OSGv2VnC0go

[15] Norton, P., Samuel, A., Aitel, D., Foster-Johnson, E., Richardson, L., Diamond, J., Parker, A., & Roberts, M. (2005). *Beginning Python*. Wiley Publishing.

[16] Parker, J. R. (2021). *Python: An introduction to programming* (2nd ed.). Mercury Learning and Information.

[17] PyData. (2023, January 11). *Raymond Hettinger: Numerical marvels inside Python - keynote | PyData Tel Aviv 2022* [Video]. YouTube. https://www.youtube.com/watch?v=wiGkV37Kbxk

[18] Python Software Foundation. (2025a, June 11, 15:46 UTC). *3.3.2.4. __slots__*. Python 3.13.5 documentation. Retrieved October 30, 2025, from https://docs.python.org/release/3.13.5/reference/datamodel.html#slots

[19] Python Software Foundation. (2025b, June 11, 15:46 UTC). *Mapping types — dict*. Python 3.13.5 documentation. Retrieved October 30, 2025, from https://docs.python.org/release/3.13.5/library/stdtypes.html#mapping-types-dict

[20] Ramalho, L. (2015). *Fluent Python: Clear, concise, and effective programming*. O'Reilly Media.

[21] Reitz, K., & Schlusser, T. (2016). *The hitchhiker's guide to Python: Best practices for development*. O'Reilly Media.

[22] SF Python. (2016, December 16). *Modern dictionaries by Raymond Hettinger* [Video]. YouTube. https://www.youtube.com/watch?v=p33CVV29OG8

[23] Shovic, J. C., & Simpson, A. (2021). *Python all-in-one for dummies* (2nd ed.). John Wiley & Sons.

[24] Storkel, H. L. (2022). Minimal, maximal, or multiple: Which contrastive intervention approach to use with children with speech sound disorders? *Language, Speech, and Hearing Services in Schools*, *53*(3), 632–645. https://doi.org/10.1044/2021_LSHSS-21-00105

[25] Sweigart, A. (2021). *Beyond the basic stuff with Python: Best practices for writing clean code*. No Starch Press.

## New Enhanced Versions of the Program "Ka minimalnim parovima" ("Towards Minimal Pairs")

*Danilo Aleksić*

### Summary

The paper presents a modest optimization technique applied to further increase the speed of a specific Python script. The purpose of the script is to find pairs of strings which differ in two specified substrings only (in one or more positions). Depending on the language of the input corpus, these pairs, or some of them, can be useful in linguistics, language teaching, and speech-language pathology. The core of the script pairs up strings with which identical comparison strings are associated. For example, in the versions 2 and 2.1 of the script, if the specified strings are `"a"` and `"e"` and if the list of the input corpus tokens contains the strings `"real"` and `"reel"`, the strings `"real"` and `"reel"` will both have `"r▽▽l"` as the comparison string and the output will contain the pair `"real ~ reel"`. Until now, the fastest version of the script (2.1; Алексић & Мркела, 2022) coupled strings by means of a dictionary, in which the keys were the comparison strings. In this version, a dictionary was also a component (and the output) of the tokenization function. The current fastest versions, 2.2 and 2.3, couple strings by means of classes in which attribute

names and attribute values correspond to the 2.1 version's keys of the comparison dictionary and values of the comparison dictionary, respectively. The tokenization dictionary is replaced by a class, too.

Key words: language, classes, minimal pairs, optimization, dictionaries, Python.