УНИВЕРЗИТЕТ У БЕОГРАДУ
ГЕОГРАФСКИ ФАКУЛЕТ

ПАЈТОН (PYTHON) и ...три тачке

Александар Пеулић

Београд, 2023

Издавач
Универзитет у Београду, Географски факултет
Студентски Трг 3/3, 11000, Београд

За издавача
проф. др Велимир Шећеров

Рецензенти
проф. др Сања Стојковић
проф. др Владимир Миловановић

Уредник
проф. др Сања Манојловић

ISBN
978-86-6283-136-1

Посвећено свим својим свесним професионалним промашајима!

Аутор

САДРЖАЈ

ПАЈТОН (PYTHON) и ...три тачке увод

Развој рачунарске технике, пре свега рачунарске опреме, како је усвојено из стручне терминологије хардвера, пре више од тридесет година отворио је улаз дигиталним уређајима, у то време персоналних рачуанара или популарних десктоп, у наше домове и канцеларије, затим у наше торбе и ранчеве, лап топове, па у наше џепове, паметни телефони и ...(три тачке), шта нас даље чека, развојем 5g мреже не можено ни да наслутимо!
Дакле, уређај имамо, треба га користити и користимо их немилице ево већ деценијама, свако на свој начин, зависи од примене и ситуације, пишемо текст, проверавамо електронску пошту, гледамо различите видео садржаје, ...програмирамо!
Када говоримо о програмирању, важно је истаћи да се од појаве првог дигиталног уређаја са могућношћу чувања информација, меморијом и Фон Нојмановом архитектуром, основним концептом архитектуре савременог рачунара, где се инструкције процесору смештају у меморију одакле их процесор узима и извршава као програм, развијају и алати за креирање програма, програмски језици.
У тој широкој лепези програмских језика, без претензија за рангирањем или поређењем, када говоримо о Пајтону, непобитне чињенице су да је бесплатан, свима доступан, да не постоји проблем, област, примена која не може да се опише у Пајтону написаној апликацији, почевши од основне манипулације подацима, аритметичко-логичким операцијама, сложеним математичким обрадама, преоцесирањем слика и видео садржаја, применом у области машинског учења и вештачке интелигенције и ...три тачке!
Ни најискуснији Пајтон програмер неће у својој програмерској пракси имати прилику да проба или користи све могућности, библиотеке, функције, које Пајтон несебично нуди.
Када кажемо Пајтон, мислимо на Пајтон отворену заједницу, огромну покретачку снагу која стоји иза развоја овог програмског језика из дана у дан, месеца у месец, додајући у огромно складиште Пајтонових библиотека нове станаре.
Одатле и инспирација за наслов ове књиге, Пајтон и три тачке, колико год се трудили, колико год дигиталних страница

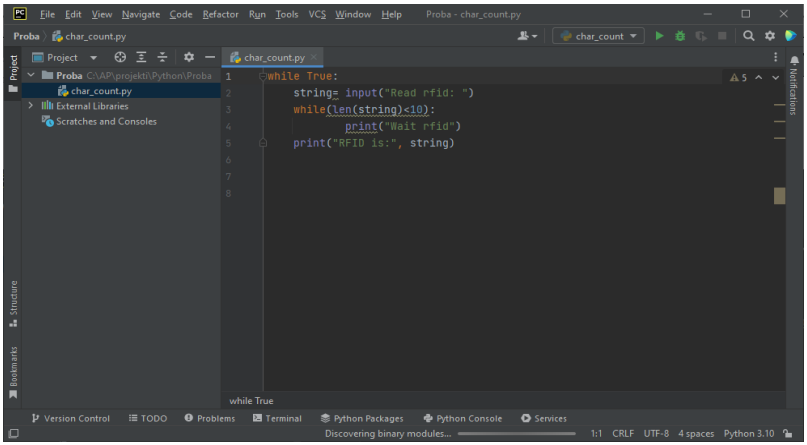
потрошили трудећи се да опишемо Пајтон, остаћемо недоречени, ускратићемо неког корисника за његову област интересовања, баш оно што њему треба, неће се наћи ни у нашој ни у било којој другој књизи, дакле, да би смо били коректни пре свега према програмском језику који описујемо и користимо, према Пајтону, алату који се свакодневно развија и несебично, бесплатно нам нуди све своје ресурсе, описаћемо основе Пајтона, Пајтон и математику, Пајтон и обраду слике, Пајтон и елементе машинског учења, Пајтон и географски информациони системи, Пајтон и … три тачке, једна књига није довољна!

Све старо, текуће и ново везано за Пајтон, налази се на званичном сајту, линк: https://www.python.org, одакле је могуће преузети најновију инсталацију Пајтона за оперативни систем који користите.

Сви примери у овој књизи тестирани су на Python 3.10 верзији програмског језика.

Пајтон је могуће користити и инсталацијом Анаконде, са званичног сајта, https://www.anaconda.com.

За развој програма, постоје развијена интегрисана развојна окружења, усвојено IDE(Integrated Development Environment), од којих је најпознатији и коришћен за тестирање примера у овој књизи PyCharm, који је могуће преузети са званичног линка: https://www.jetbrains.com/pycharm/



Циљ ове књиге је да читаоцима приближи и према ауторовој искреној жељи разјасни филозофију решавања само једне групе проблема коришћењем програмског језика Пајтон. У поглављима који следе, акценат је стављен на објашњење начина функционисања и коришћења одређених елемената програмског језика, без навођења великог броја примера, који би по мишљењу аутора оптеретили текст. Књига је намењена свима, студентима основних или мастер студија који у својим предметима имају неку овде описану област, почетницима који имају жељу да се упознају са Пајтоном, па можда и искусним програмерима који у овој књизи можда могу да нађу неки почетни пример за свој пројекат. Сви тестирани примери могу се наћи као додатак књизи на страници:

https://github.com/apeulic/Python-i-tri-tacke/find/main
https://drive.google.com/drive/folders/1oW3GjovmEuTO1ffa2GCTZvY-OcIyqqlm?usp=share_link

Срећно читање!

ПАЈТОН (PYTHON) и ...основе

Као било који програмски језик, уопштено говорећи као било који говорни језик, тако и Пајтон има неке своје основне постулате и правила која је потребно савладати на самом почетку да би смо били кадри да користимо у пуном капацитету потенцијал програмског језика којим се у овој књизи бавимо. Елементи овог поглавља названог основе представљају срж програмској језика Пајтон и без обзира на назив, након овог поглавља читаоци ће бити упознати са филозофијом и синтакском Пајтона и биће способни да прате следећа поглавља или да се ухвате у коштац са проблемима који у овој књизи нису описани и који су негде иза оне три тачке наслову књиге. У овом поглављу биће описане аритметичко логичке операције, уношење података у програм, такозвани стандардни улаз-излаз, инструкције гранања и понављања, функције, уграђене функције, рекурзивне функције, стрингови, листе, скупови, торке и речници. Концепт овог као и осталих поглавља књиге биће фокус на примерима. Али, да би се дошло до примера, па и правог програма који решава неки конкретан проблем, пре свега је важно разумети проблем и наравити стратегију решавања задатка. Под стратегијом подразумевамо концепт или план који следимо да бисмо решили неки проблем, ово је општи приступ решавању без претензија за било који програмски језик. Архитекта пројектује зграду, њену визуелну и функционалну основу, па се тек касније опредељује за материјал и грађевинске методологије које ће применити током реализације замишљеног објекта. У конкретном примеру то се зове архитектронска основа, у неким другим инжењерским дисциплинама пројекат или макета, у програмирању се почиње од алгоритма, графичког приказа тока програма којим се решава задати проблем.

Алгоритам представља прецизно дефинисан скуп правила помоћу којих се решава један или више задатака. Тај скуп правила најједноставиније је приказати текстуално, речима на

језику који разумемо. Недостатак оваквог приступа је да сви не говоримо истим језиком, па се језички или текстуални опис алгоритма на овај начин ограничава на локалну примену или на примену на језичко подручје које учесници пројекта разумеју. Генерализација приказа алгоритма на свима разумљив начин је графички приказ. Данас се класични графички приказ алгоритама врло ретко користи, али је за почетак писања програма врло важно направити план или неку врсту водича за решавање проблема који претварамо у код!

Узмимо пример замене батерија на даљинском управљачу. Текстуално, на српском језику који користимо у овој књизи, назовимо задатак **Замена батерија на даљинском**.

Задатак: Замена батерија на даљинском.
Предмет: Даљински
Ако су се батерије испразниле онда
    Услов: Купили смо нове батерије
    Акција: Отварамо поклопац на даљинском
        Вадимо старе батерије
        Водимо рачуна о поларитету
        Стављамо нове батерије
        Враћамо поклопац даљинског
        Пробамо функционалност
        Одложимо старе батерије у контејнер за рециклажу
    Користимо даљински

Опис овог алгоритма је једноставан и јасан, наравно за читаоце који размеју српски језик и ћирилично писмо. Некада се користио графички приказ алгоритама који има своје симболе старе већ дуги низ година. Комплексност пројеката, савремене методе управљања софтверским пројектима изнедриле су и нове начине планирања и дефинисања алгоритма пројекта. Када дефинишемо алгоритам, разумљив свим учесницима на пројекту, што практично значи пропишемо процедуру решавања проблема, језички, графички или на било који други начин, опредељујемо се за алат, програмски језик којим решавамо проблем, у нашем случају, избор је Пајтон којим се бавимо у овој књизи. Пајтон је бесплатан програмски језик, доступан за све познате оперативне системе. Након инсталације Пајтона, могуће

је користити га из Пајтон конзоле, у којој **>>>** означава текућу линију кода, притиком на ***Enter*** код се извршава. Наравно, овакав приступ је погодан само за неке једноставније инструкције или рецимо коришћење Пајтон конзоле као калкулатора, што ћемо искористити за примере у овом поглављу јер описујемо једноставне инструкције, али за било какав други задатак за који је потребно запамтити код, користи се текстуална датотека у коју се по правилима синтаксе уносе инструкције. Датотека се памти под именом које асоцира на циљ програма, а оно што је најважније, мора да има екстензију ***ру,*** да би била препозната од стране Пајтон развојног окружења. Па кренимо са основама Пајтона!

Променљиве

Сам назив нам сугерише да се ради о нечему што се током извршавања програма мења, може да узима различите вредности у складу са алгоритмом. Променљива мора да има своје име, које програмера треба да асоцира на улогу променљиве у програму, на пример ако рачунамо промене цена јабука услед инфлације, онда је логично и упутно да дефинишемо променљиву ***јабука***, јер се бавимо јабукама.
Пајтон има дефинисана правила за давање назива променљивама, а то су:
Име променљиве сме да садржи само:

- бројеве.
- мала и велика слова
- доњу цртицу _

Име променљиве **не сме**:

- да почиње бројем
- да буде кључна реч, а то су речи које су већ дефинисане у Пајтону.

Да би смо проверили које су то кључне речи, искористићемо Пајтон наредбе на следећи начин:

```
import keyword
keyword.kwlist
```

print(keyword.kwlist)

Добијамо листу кључних речи:

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

У имену променљиве Пајтон разликује велика и мала слова, што значи да су Jabuka, jabuka, JaBuKa, JABUKA, различите променљиве, па је препорука да се код давања имена променљивама, води рачуна да је име јасно и једнозначно, јер Пајтон ће да препозна разлику између Jabuka и jabuka, ми као програмери, посебно ако се ради о сложенијем програму са више линија кода, можемо да имамо потешкоће да уочимо грешку!

Пример недозвољеног имена променљиве може да буде: 123јабука, јабука друга, јабука#.
Поред писања програма, много више времена користи се на тестирање и отклањање грешака, међу програмерима, популарно ***Debug.*** Код отклањања грешака, важну улогу имају поруке о грешци коју добијамо од развојног окружења које нам помажу да нађемо и исправимо грешку. Најчешће грешке са којима се срећемо су синтаксне грешке или семантичке грешке на које нам рецимо развојно окружење ***PyCharm*** одмах сугерише бојењем текста црвеном бојом, па можемо да рагујемо и пре покретања програма. Ако ипак покренемо програм добијамо поруку о грешци и те грешке зовемо: **Runtime Error**.
Пример:

```
>>> a = 2
>>> b = 3
>>> c = 1.5
>>> a
2
>>> c
1.5
```

```
>>> x
Traceback (most recent call last):
    File "<pyshell#5>", line 1, in <module>
        x
NameError: name 'x' is not defined

>>> tekst = "Ovo je poglavlje Python osnove"
>>> pi = 3.14
>>> n = 15
>>> tekst
'Ovo je poglavlje Python osnove'
>>> pi
3.14

>>> n
15
```

Када дефинишемо променљиву и доделимо јој вредност, аутоматски се дефинише и тип променљиве. Уколико нисмо сигурни који је тип променљиве од интереса је, интерпретер нам даје одговор на следећи начин, коришћењем ***type***:

```
>>> type(tekst)
<class 'str'>
>>> type(pi)
<class 'float'>
>>> type(n)
<class 'int'>
```

Типови променљивих које се користе у Пајтону су:

- целобројни **int**
- реални **float**
- комплексни **complex**
- логички **bool**
- текстуални **str**
- торка **tuple**
- листа **list**
- скуп **set**
- речник **dict**

- недефинисан **NoneType**

Аритметичке операције

Аритметичке операције које можемо да користимо у Пајтону су:

- сабирање **+**
- одузимање -
- множење *
- дељење /
- остатак при дељењу %
- целобројно дељење //
- степеновање **

Приоритет код извршавања аритметичких операција је уобичајен, уколико нисмо сигурни, омеђимо операцију заградом!
Пример:

```
>>> 2 + 3
5
>>> 2 -3
-1
>>> 2 * 3
6
>>> 9 / 4
2.25
>>> 2 + 3 * 5
17
>>> 2 + 4 * 9 -3
35
>>> 37 // 10
3
>>> 37 / 10
3.7
>>> 37 % 10
7
```

Стандардни улаз-излаз

Функција ***print***

Функција ***print()*** нам омогућава да испишемо на екрану одговарајућу вредност која се налази унутар заграда:

```
print (‘Zdravo svete')
                                    одговор:  Zdravo svete
print (34)
                                    одговор: 34
print (12*12)
                                    одговор:144
print (12+14/(6+1))
                                    одговор:14.0
recenica= 'Pajton je lak '
print (recenica)
                                    одговор: Pajton je lak
rec1 = 'Lep je'
rec2 = 'pajton'
print (rec1, rec2)
                                    одговор: Lep je Pajton
ime= 'Aleksandar'
print (ime, 'ima', 95, 'kg')
                                    одговор: Aleksandar ima 95 kg
print ("Kvadrat broja %d je %d" %(5, 25))
                                    odgovorодговор :  Kvadrat broja 5 je 25

print ("%s je danas kupio %d kilograma %s" %("Aleksandar", 3,
"jabuka"))
                                                одговор: Aleksandar je danas kupio
3 kilograma jabuka
```

Функција ***input***

За унос неких података од стране корисника који су потребни за извршавање Пајтон програма користи се функција ***input***, на следећи начин:

• Позивом функције ***input***, програм “стаје“ и чека на корисника да унесе жељене податке.
• Пајтон чита те податке са стандардног улаза као стринг.
• Овај стринг се може конвертовати у ***float*** или ***int*** ако је уписан као број.

Функција ***input(),*** интерпретира кориснички улаз, што значи да ако корисник унесе **integer**, онда ће се **integer** и вратити, ако корисник унесе стринг, стринг ће се вратити, ако унесе **float**, float ће се вратити .

Пример: Рачунање квадрата броја.

```
print ("Unesite neki broj: ")
x = float(input())
print ("Kvadrat broja %.1f je %.1f" %(x,x*x))
```

Уграђене функције

Пајтон нам омогућава да израчунамо минимум или максимум два или више бројева на једноставан начин:

```
>>> min (4, 8)
4
```

min је функција која враћа мањи од два броја која се налазе у заградама. На сличан начин ради и функција **max**:

```
>>> max (4, 8)
8
```

Унутар функције min (или max), могу се налазити и друге функције, на пример:

```
>>> min (min (2, 3), 4)
2
```

Прво се рачуна унутрашња, па затим спољашња функција

```
>>> min (4, 8)
4
```

Апсолутна вредност се рачуна помоћу функције **abs**:

```
>>> abs (3)
3
>>> abs (-8)
8
```
Унутар функције abs, можемо унети било који израз:
```
>>> abs (5 - 3 * 2)
1
```
Пример комбинације више функција и променљиве:
```
>>> broj = -3
>>> max (abs (broj), min (2, 4))
3
```

Остале математичке функције

За коришћење осталих математичких функција, потребно је учитати(импортовати) математичку библиотеку **math**, тако што унисимо следећу наредбу:

import math
Након тога можемо да користимо математичке функције, на пример:
math.factorial(x) - враћа факторијел броја x
```
>>> math.factorial(4)
24
```
math.pow(x, y) - степеновање броја x бројем y
```
>>> math.pow(3, 2)
9.0
```
math.sqrt(x) - враћа квадратни корен броја x
```
>>> math.sqrt(16)
4.0
```
math.pi - враћа константу Pi (слично и за math.e)
```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
```
math.floor(x) - враћа највећи цео број мањи или једнак броју x
```
>>> math.floor(4.4)
```

4
>>> math.floor(4.9)
4
>>> math.floor(-4.4)
-5
math.trunc(x) - брише све иза децималне тачке
>>> math.trunc(4.4)
4
>>> math.trunc(-4.4)
-4
math.exp(x) - враћа вредност израза експонент
>>> math.exp(2)
7.38905609893065
math.log(x) - враћа логаритам од x за основу e
>>> math.log(10)
2.302585092994046
math.log(x, y) - враћа логаритам од x за основу y
>>> math.log(100, 10)
2.0

Тригонометријске функције (исписују вредности у радијанима): math.sin(x), math.cos(x), math.tan(x), math.asin(x), math.acos(x), math.atan(x)

Коментари

Коментари су врло корисни пре свега програмеру да протумачи филозофију функционисања кода, што се посебно односи на ситуацију када смо наследили или анализирамо туђи код. Уколико су делови програма добро коментарисани, лакше ћемо да се снађемо код анализе, што нам сугерише да и сами треба да се потрудимо да свој код што боље коментаришемо.
Коментар почиње симболом #, што за последицу има да је линија иза симбола коментар. За коментраисање више линија или већег дела текса користи се троструки наводник ''' '''. Текст између наводника је коментар.

Пакети и модули

Модул је датотека која садржи Пајтон код, док под пакетом подразумевамо нешто као складиште које садржи модуле и друге пакете. Библиотека садржи више модула. Повећавањем величине кода програма на коме радимо, повећава се и број коришћених модула. Модули који су по нечему слични стављају се у један пакет, а различити модули у различите пакете, што програм чини јаснијим и лакшим за управљање. За интеграцију пакета и модула у Пајтон код, користимо:
from ... import ....

Некада је потребно да знамо локацију, односно тренутни директоријум у коме се налазимо, тако да за те потребе учитавамо модуле **os:**

```
import os                 # import os paket
os.getcwd()               # uzmi tekuci direktorijum
print(os.getcwd())        # prikazi tekuci direktorijum
```

У претходном примеру учитали смо комплетан модул os, а потребан нам је само текући директоријум, тако да исти резултат постижемо учитавањем само методе getcwd(), на следећи начин:

```
from os import getcwd          # iz modula os ucitaj getcwd metodu
getcwd()                       # uzmi tekuci direktorijum
print(getcwd())                # prikazi tekuci direktorijum
```

Ако је име пакета сложено за позивање, на пример уколико је име дугачко, може му се доделити скраћени назив, псеудоним, као на пример:

```
import platform
platform.platform()
print(platform.platform())
```

```
import platform as pf          #platform menjamo pseudonimom pf
pf.platform()
print(pf.platform())
```

У претходном примеру, приказује се платформа на којој се

извршава код, у конкретном случају одговор је:
Windows-10-10.0.19045-SP0

Функције

Функција је скуп наредби које се извршавају по позиву. Пајтон има широку лепезу уграђених функција и допушта програмеру дефинисање сопствених функција.

Функција се дефинише помоћу кључне речи ***def,*** на следећи начин:

def име_функције(параметри):
    Наредбе

Свака функција има јединствен назив преко кога се може позвати произвољан број пута из било ког дела програма.

Пример функције за израчунавање разлике бројева 100 и 50:

```
def oduzmi():                    # funkcija
    print(100-50)                # telo funkcije
>>>oduzmi()                     # poziv funkcije
50
```

Позив функције је израз са следећом синтаксом:
ime_funkcije (argumenti)
Ако функција прихвата аргументе, онда параметри садрже један или више идентификатора, одвојених зарезима.
У следећем примеру је написана функција која штампа квадрат неког броја, а затим и позив функције којој се као аргумент прослеђује број чији квадрат фунција рачуна.

```
def izracunaj_kvadrat(x):
    print (x*x)
>>>izracunaj_kvadrat(3)     #pozivamo funkciju, koja ispisuje izlaz 9
9
>>>izracunaj_kvadrat(8)    #pozivamo funkciju, koja ispisuje izlaz 64
64
>>>izracunaj_kvadrat(12.5)  #pozivamo funkciju, koja ispisuje izlaz
```

```
                                        #156.25
156.25
```

Ако функсија има више аргумената, онда је позивамо тако што прослеђујемо дефинисан број аргумената, на пример функција која исписује збир два броја. Функција се позива за бројеве 10 и 15.

```
def saberi(x,y):
        print (x+y)
#pozivamo funkcije sa dva argumenta
>>>saberi(10,15)                #izlaz 25
25
>>>saberi(3,15)                 #izlaz 18
18
```

Локалне променљиве функције

Променљиве дефинисане и коришћене унутар функције, не могу да се користе ван функције, на следећем примеру променљива **c** је дефинисана унутар функције и не може да се користи изван функције.

```
def saberi(a,b):
        c = a + b
        print (c)

>>>saberi(3,5)
8
>>>  print(c)                    NameError: name 'c' is not
                                        defined
```

Враћање резултата функције

Функција као резултат може да врати вредност, на пример:

```
def saberi (a,b):
        c = a + b
        return c
>>> y = saberi(12,26)
>>> print(y)
```

Функција може да врати и више вредности, на пример ако желимо да претворимо центиметре у метре и преостале ценриметре:

```
def cm_u_mcm(cm):
        return (cm // 100, cm % 100)
>>>(m, cm) = cm_u_mcm(184)
>>>print(184, "cm", "=", m, "m", "i", cm, "cm")
```

Гранање

Гранање је програмска структура која усмерава ток програма зависно од резултата постављеног услова. Неке наредбе се извршавају само ако је испуњен одређени услов, а да би се извршило условно гранање користи се кључне речи if, else, elif, на следећи начин:

```
if uslov:
        naredba_1
        ...
        naredba_k

if uslov:
        naredba_1
        ...
        naredba_m
else:
        naredba_1
        ...
        naredba_n

if uslov1:
        naredba_1
        ...
        naredba_m
elif uslov2:
```

```
        naredba_1
        ...
        naredba_n
else:
        naredba_1
        ...
        naredba_k
```

Наредба услова вратиће тачно или нетачно, ако је услов тачан извршиће се наредба_1, у супротном се испитује услов_2.
За дефинисање услова користе се логички оператори:

- > веће
- < мање
- >= веће или једнако
- <= мање или једнако
- == једнако
- != различито

Понављање

Понављање представља механизам да се једна или више наредби понавља онолико пута колико је потребно да се задовољи задати услов. Претпоставимо да желимо пет пута да испишемо своје име, на пример Александар, на екрану. То лако можемо да урадимо уколико пет пута заредом унесемо инструкцију:

```
print("Aleksandar")
print("Aleksandar")
print("Aleksandar")
print("Aleksandar")
print("Aleksandar")
```

И ово је било досадно куцати, а замислимо потребу да се нека инструкција понавља неколико десетина или стотина пута! За такве примене, у Пајтону, као и у свим другим програмским језицима постоје наредбе понаљања, које се називају петље.
Петље су најједноставнији начин да рачунар изврши исту ствар више пута и у те сврхе најчешће коришћене су две врсте петљи:

for петље и ***while*** петље.
У већини случајева се могу користити и *for* и *while* петља, али корисно је знати обе петље.
Петља ***for*** се обично користи када знамо колико пута треба да се понови неки корак, на пример, ***for*** петљом се лако крећемо кроз неки опсег бројева.

Структура ***for*** петље

```
for «imePromenljive» in range(«pocetnaVrednost»,«krajnjaVrednost»):
        «uvucen blok komandi, telo petlje»
```

Редови кода који се извршава у оквиру петље морају бити поравнато увучени. Прво се тело петље извршава узимајући да је вредност променљиве ***imePromenljive*** једнака променљивој ***pocetnaVrednost***.

Затим се тело поново изврши, али сада променљива ***imePromenljive*** узима вредност ***pocetnaVrednost+1***.
Поступак се понавља све док ***imePromenljive*** не постане једнако ***krajnjaVrednost-1***.

Пример програма који исписује бројеве од 1 до 10, коришћењем for петље.

```
for i in range(1, 11):
    print(i)
```

Обратимо пажњу да смо за крајњу вредност поставили 11, јер се по дефиницији петља завршава када променљива узме вредност ***krajnjaVrednost-1.***
Петља дозвољава да се вредност променљиве повећава не увек за 1 већ у задатим корацима, тако што се корак задаје у структури петље иза крајње вредности:

```
for «imePromenljive» in range(«pocetnaVrednost», «krajnjaVrednost»,«korak»):
        «uvucen blok komandi, telo petlje»
```

На пример, следећи програм исписује бројеве од 5 до 95.

```
for i in range(5,100,5):
   print(i)                    # ispisuje broj
```

Петља ***while*** функционише на начин, да све док је задовољен задати услов, понављају се инструкције, када услов више није задовољен излази се из петље.
Структура ***while*** петље:

```
while «uslov»:
      «uvucen blok komandi, telo petlje»
```

Петља функционише тако да се проверава истинитост услова. Ако је услов True(тачан), извршавају се инструкције у телу петље, затим се поступак понавља све док услов не буде False(нетачан), када се петља завршава.
Следећим примером приказано је коришћење ***while*** петље, тако што се уносе бројеви све док се не унесе број 5.

```
x = int(input("Unesi broj: "))

while x != 5:                          # uslov
      print(x)                         # telo petlje
      x = int(input("Unesi broj: "))   # ucitava broj
```

Помоћу петљи, можемо да напишемо програм који се никада не завршава, користећи бесконачну петљу, на пример:

```
while True:
      print("Nema kraja...")
```

За прекид рада петље користи инструкција ***Break***, у случајевима када је неопходно да се прекине петља иако њен услов још увек није био ***False***.

Следећим примером, петља се завршава када променљива добије вредност 3, без обзира што је у структури петље дефинисана крајња вредност 10.

```
for i in range(1,10):
```

```
    if i== 3:
        break
print (i)
```

Инстуркција ***continue*** прекида се извршавање тренутне итерације, прескаче се део кода и прелази на следећу итерацију. Пример приказује исписивање свих бројева од 1 до 9 прескакањем броја 3.

```
for broj in range(1,10):
    if broj == 3:
        continue
    print (broj)
```

Стрингови(ниске)

Стринговима посвећујемо посебну пажњу. До сада смо се већ срели са стринговима, као на пример:

```
print ("Zdravo svete")
```

Стринг је секвенца или низ од 0 или више карактера које уписујемо преко једноструких или двоструких наводника, на пример:

```
'Python programiranje'
"Beograd, Kragujevac, Srbija"
'4 8 15 16 23 42'
''
```

Да бисмо приказали стрингове користимо ***print*** функцију на следећи начин:

```
>>>print ("Zdravo, svete!")
                                    одговор: Zdravo, svete!
>>>print ("Ovde pisemo neki string!")
                                    одговор: Ovde pisemo neki string!
```

Стринг можемо да доделимо и променљивој, као на пример:

```
 >>>s = 'Zdravo Aleksandar'
 >>>print (s)
                                    одговор: Zdravo Aleksandar
```

Дефинисали смо стринг као секвенцу или низ од 0 или више карактера које уписујемо преко једноструких или двоструких наводника, али морамо да водимо рачуна о комбинацији једноструких и двоструких наводника. Стринг који почиње двоструким наводницима мора да се заврши двоструким наводницима, тако да можемо да имамо једноструке наводнике у таквом стрингу. Стринг који почиње једноструким наводницима мора да се заврши једноструким наводницима, тако да можемо да имамо двоструке наводнике у таквом стрингу. Хајде да илуструјемо ово примером:

```
>>>s = 'Aleksandar je rekao, "Zdravo Python!"'
```

Стринг је са спољашње стране затворен једноструким, а унутар стринга имамо двоструке наводнике.

Врло често постоји могућност за приказивање више линија стринга. Ако покушамо да поставимо стрингове са више линија то нам неће успети, али ако почнемо и завршимо стринг са три једнострука или двострука наводника, онда ће Пајтон дозволити да пишемо више линија све док не завршимо стринг истим једноструким или двоструким наводницима.

```
>>>s = """Ovo
je string koji ima
vise linija."""
>>>print (s)
Ovo
je string koji ima
vise linija.
```

Ако у терминалу укуцамо само **s**, добијамо следеће:

```
>>>s
'Ovo\njestring koji ima\nlinija.'
```

Приметимо \n, што представља посебан карактер за нови ред, на пример:

```
s = 'Ovo je prvi red\novo je drugi red'
>>>print (s)
Ovo je prvi red
ovo je drugi red.
```

Додавањем \ (косе црте), унутар стринга сугеришемо Пајтону да је карактер непосредно иза косе црте у стрингу специјални карактер који има посебно значење. Ако иза косе црте унесемо следеће карактере, они имају следећа специјална значења на приказ стринга:

- \n – завршава тренутну линију и наставља у следећој
- \t – убацује таб у стринг (прави размак једног ‘’таб-а’’)
- \’ – убацује ‘ у стринг
- \“ – убацује “ у стринг
- \\ – убацује \ у стринг

Уколико желимо да два или више стрингова спојимо у један, то се ради коришћењем оператора +, а поступак се назива конкатенација.

```
>>>s1 = "Prvi string"
>>>s2 = ", drugi string"
>>>s1+s2
Prvi string, drugi string
```

Ако желимо да исечемо стринг или део стринга онда се то ради преко угластих заграда. У зависности од тога шта проследимо у угластим заградама, стринг ће бити различито исецкан.

У следећим примерима сецкања показаћемо како ради сецкање стринга str[]:

Напомена, # представља коментар!

```
>>>str = 'programiranje'
#ceo string
>>>print ('str = ', str)
str = programiranje
#prvo slovo
>>>print ('str[0] = ', str[0])
str[0] = p
#poslednje slovo
>>>print ('str[-1] = ', str[-1])
str[-1] = e
#od 1. do 5. indeksa tj od 2. do 6. slova
>>>print ('str[1:5] = ', str[1:5])
str[1:5] = rogr
#od 5. indeksa do 2. indeksa odpozadi
>>>print ('str[5:-2] = ', str[5:-2])
```

str[5:-2] = amiran
Када смо говорили о функцијама, рекли смо да Пајтон нуди низ уграђених функција, међу којима се налази и len() функција коју можемо да упортребимо за рачунање дужине стринга:

```
>>>s = "Hello!"
>>>print (len(s))
6
>>>c = "Zdravo, sta vi radite?"
>>>print (len(c))
22
>>> b = ""
>>>print (len(c))
0
```

Поред функције len, постоји уграђена функција која може да нађе стринг у оквиру неког другог стринга текста или томе сличног. Та функција се зове **find,** а синтакса је:

string1.find(string2, beg=0, end=len(string1))
"**string2**" у овој функцији је стринг који тражимо у "**string1**", **beg** представља од које позиције у стрингу "**string1**" тражимо "**string2**", а **end** представља до које позиције тражимо "**string2**" у стрингу "**string1**".
Argument **beg** и **end** нису обавезни, ако их не упишемо, онда ће **find** функција тражити **string2** у целом **string1**. Ако функција **find** пронађе реч или стринг у тексту, враћа позицију прве пронађене речи у тексту, а ако не нађе вратиће -1.

```
>>>str1 = "Ovo je reprezentacija nekog stringa!"
>>>str2 = "taci"
>>>print (str1.find(str2))
        одговор: 15
>>>print (str1.find(str2, 10))
        одговор: 15
>>>print (str1.find(str2, 16))
        одговор: -1
>>>print (str1.find(str2, 10, 20))
        одговор: 15
>>>print (str1.find(str2, 10, 16))
```

одговор: -1

Слична функција функцији find је **index**, а једина разлика је што ако не нађе ниједан стринг карактер који смо му задали, функција **index** пријављује грешку, док **find** исписује -1.

```
>>>print (str1.index(str2))
                                    одговор:15
>>>print (str1.index(str2, 16))
                                    одговор:        Traceback (most
recent call last):
                                                    File "python", line 3,
in <module>
                                                    ValueError: substring
not found
```

Листе

Листа представља променљив, уређен низ објеката. Код листе је важан редослед елемената. Сваки члан листе има свој индекс на основу кога га идентификујемо, то је заправо његова позиција у листи. Индексирање у Пајтон програмском језику креће од 0. Елементи листе могу да мењају вредност, па због тога кажемо да је листа променљив низ објеката.
Листу можемо генерисати и на следећи начин:

```
>>> lista=list(range(1,10))
>>> print (lista)
                    одговор:[1,2,3,4,5,6,7,8,9]
```

Креирали смо листу бројева од 1 до 10-1. Уколико додамо трећи аргумент рецимо lista=list(range(1,10,2)), креираће елементе листе са кораком 2, односно сваки други елемент. Уколико ранг иде од 1 до 10 то не укључује последњи број, већ иде од 1 до 10-1.

```
 >>> lista=list(range(1,10,2))
>>> print (lista)
                    одговор:[1,3,5,7,9]
```

Елементу листе можемо да приступимо преко његовог индекса то јест позиције у листи, а сетимо се да индекс почиње од 0.
На пример ако имамо листу елемената lista =[10,9,8,7,-1,-2,-3] и индексирање креће од 0, први елемент је lista[0] и то је број 10. Ако желимо да приступимо броју 8 и да га штампамо то бисмо урадили на следећи начин:

```
>>> lista=[10, 9, 8, 7,-1,-2,-3]
>>> print (lista[2])
```

одговор:8

Ако је дата листа природних парних бројева lista = [2,4,6,8] и желимо да променимо вредност броја 6 на 0 довољно је да знамо његов индекс, у конкретном случају индекс је 2.

```
>>> lista = [2, 4, 6, 8]
>>> lista[2]=0
>>> print ( lista )
```

одговор:8 [2, 4, 0, 8]

Две или више листи се могу спојити у једну једноставно знаком плус (+) .

```
>>> A = [1,2,3,4,5]
>>> B = [6,7,8,9]
>>> C = A + B
>>> print (C)
```

одговор:[1, 2, 3, 4, 5, 6, 7, 8, 9]

Као и код стрингова, и за листе ради уграђена функција **len** која враћа дужину листе.

```
>>> lista=[‘jedan’ , 2,3,4,5,[-2,-4,-5]]
>>> print (len (lista))
```

одговор:6

Обратимо пажњу да је стринг ‘jedan’, један елемент листе, да су бројеви, 2,3,4,5 посебни елементи листе и да је листа [-2,-4,-5], посебан елемент листе.
Функције **append** и **extand** се користе за додавање елемената на крај листе.
Прва функција додаје један елемент на крај листе, односно додаје елемент који представља листу, а друга функција додаје више елемената, тако да ако бисмо исти аргумент послали

обема функцијама имали би смо различит резултат.

```
>>> lista=[1,2,3]
>>> lista.append([4,5])
>>>print (lista)
                    одговор: [1,2,3,[4,5]]

>>> lista=[1,2,3]
>>> lista.extand([4,5])
>>>print (lista)
                    одговор: [1,2,3,4,5]
```

Функција **insert** додаје жељени елемент на задату позицију, има два аргумента, први аргумент је позиција (индекс) испред које желимо да додамо елемент, а други аргумент је вредност коју додајемо.

```
>>> A = [1,2,3,4,5]
>>> A.insert(2,0)
>>> print ( A )
                    одговор:[1, 2, 0,3, 4, 5]
```

Функције **max** и **min** проналазе максимум и минимум у листи, функција **count** враћа број понављања неког елемента у листи.
Функције **del** и **pop** се користе за брисање елемената из листе, с тим да прва функција као аргумент добија назив то јест вредност елемента листе, а друга добија индекс елемента листе.

```
>>> A=[1,2,3,4,5,6,7,8,9]
>>> del(A[3])
>>> print ( A )
                    одговор:[1,2,3,5,6,7,8,9]
>>> A.pop(1)
>>> print ( A )
                    одговор:1,3,5,6,7,8,9]
```

Функција **sort** сортира елементе листе у неопадајућем поретку, док функција **reverse** сортира елементе у опадајућем поретку.

Речници

Речници представљају генерализовану верзију листе чија је предност да не морамо да размишљамо о индексу, лакше их користимо и код је читљивији.
Да би смо дефинисали да је нешто речник, користимо велике заграде,**{ }**. Речници се састоје од елемената који представљају пар раздвојен са две тачке, тако да је први чинилац пара кључ, а други вредност, енглески, **key** и **value**.
Кључ представља главни елемент помоћу кога приступамо речнику, слично индексу, али кључ може да буде и стринг, као и реални и цели бројеви.
Карактеристика речника је да се у једном речнику могу наћи кључеви различитог типа, што олакшава програмирање.
Посматрајмо један речник:

```
>>> r={'A':50, 'B':100}
```

У овом примеру речника, А представља кључ, а 50 је вредност.
Уколико желимо да приступимо вредности:

```
>>> r['A']
                    одговор:50
```

Промена вредности елемента речника:

```
>>> r['A']=200
>>> r['A']
                    одговор:200
```

Убацивање новог елемента у речник:

```
>>>r['C']=400
>>>r
                    одговор:{'A': 200, 'B': 100, 'C': 400}
```

Уклањање елемента из речника:

```
>>>del r['A']
>>>r
                    одговор:{'B': 100, 'C': 400}
```

Копирање речника:

```
>>>r
                    одговор:{'B': 100, 'C': 400}
>>>c=r.copy()
>>>c
```

одговор:{'B': 100, 'C': 400}

Ако желимо да утврдимо да ли је нешто кључ у речнику или није, користимо оператор **in**, па на пример можемо да претражимо све кључеве и речнику:

```
>>>r
                одговор:{'B': 100, 'C': 400}
>>>for kljuc in r:
        print(kljuc)

                одговор:B
                одговор:C
```

или да штампамо све вредности из речника:

```
>>>r
                одговор:{'B': 100, 'C': 400}
>>>for kljuc in r:
   print(r[kljuc])


                одговор:100
                одговор:400
```

Обратимо пажњу на врсту и место заграда које користимо!

Из речника можемо да добијемо листе кључеве и њихових вредности, коришћењем функције ***list*** на пример:

```
>>>list(r)
                одговор:['B', 'C']
>>>list(r.values())
                одговор:[100, 400]
>>>list(r.items())
                одговор:[('B', 100), ('C', 400)]
```

У последњем примеру, парове које је смо добили из **list(r.items())**, називамо торке (tuples).

Торке (tuples)

Торке су сличне листама, али за разлику од њих представљају непроменљиве листе, наводе се у малим заградама, приступа им се преко индекса као и код листе, а можемо да користимо **len** за дужину торке.
Пример торке:

```
>>>aleksandar = ('Aleksandar', 'Peulic', 31,10,1969, 'Nis','Srbija')
>>>aleksandar[4]
                    одговор:1969
```

Као што смо могли да видимо из претходног примера нема неке велике разлике између листе и торке, једини разлог коришћења торке је њихова непроменљивост, тако их можемо користити као кључеве у речницима.

```
>>>ocene= {('Veljko', 'Anja'): 95, ('Didi', 'Vukasin, Djina'): 87}
>>>list(ocene)
                    одговор:[('Veljko', 'Anja'), ('Didi', 'Vukasin, Djina')]
```

За конверзију неког објекта у торку користимо функцију **tuple**.

```
>>>r1 = tuple([1,2,3])
>>>r1
                    одговор:(1, 2, 3)
>>>r2 = tuple('abcde')
>>>r2
                    одговор:('a', 'b', 'c', 'd', 'e')
```

Торке можемо да користимо као елементе листе:

```
meseci= [("јануар", 31), ("фебруар", 28), ("март", 31), \
      ("април", 30), ("мај", 31), ("јун", 30), \
      ("јул", 31), ("август", 31), ("септембар", 30), \
      ("октобар", 31), ("новембар", 30), ("децембар", 31)]
broj= int(input('Унеси редни број месеца:'))
mesec= meseci[broj-1]
print('Назив:', mesec[0], 'Број дана:', mesec[1])

Унеси редни број месеца:4
Назив: април Број дана: 30
```

Класе

Термин класе је директна асоцијација на објектно-орјентисано програмирање. Објектно-орјентисано програмирање захтева посебну пажњу и обрађује се као посебан предмет или курс на рачунарским и техничким факултетима. Ми ћемо само приказати основне појмове и начин дефинисања класа, да бисмо разумели неки део кода или читав код који садржи класе. Читаоци који током каријере дођу у прилику да се баве објектним програмирањем у Пајтону, класама ће посветити посебну пажњу. Класа је нешто што описује објекат. Када желимо да дефинишемо класу, користимо кључну реч ***class***.
Празна класа је класа која нема никакве податке, дефинише се на следећи начин:

```
class Klasa:
  pass
```

Као што смо већ напоменули, класа дефинише објекат, замислимо класу као неки шаблон, отисак који је могуће копирати и користити на различите начине. Класа може да наследи другу класу, као што дете наслеђује родитеља, само што код класа, дете класа наслеђује све особине класе родитељ, што у природи није случај. У литератури, дете класа се назива и поткласа, а родитељ класа, надкласа. Када говоримо о наслеђивању, једна класа може да наследи једну класу, али је могуће да класа наследи и више од једне класе и то се назива вишеструко наслеђивање. Једну класу може да наследи неограничен број класа, дете може да има једне родитеље, али родитељи могу да имају више деце, на пример:

```
class Zivotinja:
 pass
class Mesozder(Zivotinja):
 pass
class Lav(Mesozder):
 pass
```

Узимимо на пример класу Пас. Пас Уран, који живи у двориштy

породичне куће Пеулић и има 5 година је објекат.
Uran= Pas()
Класа може да има своје податке који се називају атрибути класе, као на пример:

```
class Klasa:
        promenljiva = 123456789
```

Функција може бити део класе, па имамо функције класа, као на пример:

```
class Klasa:
        def funkcija():
                print("Funkcija!")
```

Када говоримо о наслеђевању, класа наследник, дете, преузима све функције и податке класе коју наслеђује, родитељске класе.

```
class Pas:
        def kretanje(self):
        print("Skakuce po dvoristu")
class Nemacki_ovcar(Pas):
        pass
Uran = Nemacki_ovcar()
Uran.kretanje()
```

Дефинишимо неке класе и примере са класама, на пример програм који приказује координате коришћењем класе:

```
class koordinata(object):
   def __init__(self, x, y):
      self.x = x
      self.y = y

a, b = 3, 7
k = koordinata(a, b)
p = koordinata(0, 0)
print(k.x, k.y, p.x, p.y)
print(k)
print(p)
```

Класа koordinata је дефинисана init функцијом, def __init__(self, x, y): са својим објектима. Затим су дефинисане две инстанце, класа k и класа p, које наслеђују класу koordinata, али имају своје објекте и приказују их.
Уколико желимо да израчунамо растојање између две тачке дефинисане координатама, коришћењем класа, то можемо да урадимо на следећи начин:

```
class koordinata(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return '<'+str(self.x)+', '+str(self.y)+'>'
    def rastojanje(self, druga):
        return ((self.x - druga.x)**2 + \
                (self.y - druga.y)**2)**0.5
a, b = 3, 7
k = koordinata(a, b)
p = koordinata(0, 0)
print(k)
print(isinstance(k, koordinata))
```

Описом класа, боље речено, основним описом дефинисања класа у Пајтону, завршавамо поглавље Пајтон и основе чији је циљ да нас уведе у Пајтон, упозна са основним постулатима овог програмског језика, синтаксом и припреми за наредна поглавља која се баве применом Пајтона у неким специфичним областима, од којих је на првом месту, математика.

ПАЈТОН (PYTHON) и математика

Не постоји област технике у којој математика у већини случајева не игра главну улогу. Када говоримо о машинском учењу, обради слике, дигиталном процесирању сигнала, математички апарат потребан за имплементацију се једноставно подразумева као нешто већ научено и добро познато. Овакав приступ применићемо и у овом поглављу, дакле циљ овог поглавља није, нити је могуће, учење математике, то је и сувише озбиљна наука да би смо је сместили у једно поглавље, без обзира о којој области математике говоримо, циљ овог поглавља је да се упознамо како у Пајтону користити благодети које нам математика пружа као помоћ и алат у решавању могих проблема. Дакле, полазимо од претпоставке да смо упознати са теоријом математике коју овде користимо, а бавимо се математиком у Пајтону.
Објаснићемо основне операторе, линеарну алгебру, вероватноћу и статистику, елементе оптимизације заснованих на Пајтону које ћемо моћи да применимо на стварне пројекте и за решавање различитих проблема.

Основни оператори

Основно математичко знање се широко користи у обради различитих типова података, пројектовању алгоритама и нумеричкој обради. Поред математичке библитеке ***math*** са којом смо се упознали у поглављу основе, обрадићемо ***NumPy*** и ***SciPy*** библиотеке. Као што већ знамо, математичка библиотека је стандардна библиотека Пајтон-а и омогућава нам коришћење неких заједничких математичких функција. ***NumPy*** библиотека је проширена библиотека Пајтон-а, која се користи за нумеричка израчунавања, на пример решавање задатака из линеарне алгебре, случајно генерисање бројева и Фуријеова трансформација. Библиотека ***SciPy*** се користи за решавање задатака повезаних са статистиком, оптимизацијом,

интерполацијом и интеграцијом. Библиотеке учитавамо у свој код наредбом ***import***, па кренимо са имплементацијом основних оператора које нисмо поменули у поглављу Основе.

```
import math
import numpy as np
math.ceil(4.01)
        одговор: 5
math.ceil(4.99)
        одговор: 5
```

Функција ceil(x) враћа најмањи цео број већи или једнак од аргумента функције, у нашем примеру x. Из примера видимо да је за улазни аргумент 4.01 и 4.99 одговор, излаз из функције 5.

Обрнуто, функија floor(x) враћа највећи цео број мањи или једнак аргументу функције x, на пример:

```
math.floor(4.1)
        одговор: 4

math.floor(4.999)
        одговор: 4
```

Функција degrees(x) конвертује аргумент функције x задат у радијанима у угао у степенима, пример:

```
math.degrees(math.pi/4)
        одговор:45.0

math.degrees(math.pi)
        одговор:180.0
```

Функција exp(x) враћа math.e, односно 2,71828 на степен x.

```
math.exp(1)

        одговор:2.718281828459045
```

Фукција fabs(x) враћа апсолутну вредност x.

math.fabs(-0.003)
    одговор: 0.003

Функција fsum(iterable) сумира сваки елемент у итератору.

math.fsum([1,2,3,4])
    одговор: 10

Функција fmod(x, y) враћа остатак од x/y. Вредност је реалан број y .

math.fmod(20,3)
    одговор: 2.0

Функција log([x, base]) враћа природни логаритам од x. Подразумевано, e је основни број. Ако је основни параметар наведен, враћа се логаритам од x на основу дате основе. Формула за израчунавање је log(x)/log(base).

math.log(10)
    одговор: 2.302585092994046
Функција sqrt(x) враћа квадратни корен од x, док функција pow(x, y) враћа x на степен y.

math.sqrt(100)
    одговор: 10

math.pow(3,4)
    одговор: 81

Линеарна алгебра

Линеарна алгебра је дисциплина која се широко користи у различитим областима, чијим коришћењем у великој мери можемо да поједноставимо обраду велике количине података. Линеарном алгебром можемо да упростимо сложене проблеме коришћењем ефикасних математичких операција. То је

математички алат, који не само да обезбеђује технологију за операције са низовима, већ такође омогућава рад са векторима и матрицама, као и операције над истим.
NumPy је модул за нумеричку обраду заснован на Пајтон-у који садржи моћне функције и предности у обради матричних података. Како линеарна алгебра углавном обрађује матрице, овај одељак је углавном заснован на **NumPy**. Библиотека **SciPy** се такође користи за решавање једначина линеарне алгебре.
На почетку, увезимо наведене библиотке, обратимо пажњу да смо их увезли са акронимом!

```
import numpy as np
import scipy as sp
```

Reshape

У математици не постоји операција **reshape**, али је то врло уобичајена операција у библиотеци **NumPy**. Операција **reshape** се користи за промену димензије тензора или матрица. На пример, слика величине 10x10 се директно чува као секвенца која садржи 100 елемената. Након уноса слике, може се трансформисати са 10x10 на 1x100 коришћењем операције **reshape**.

Креирајмо вектор који садржи целе бројеве од 0 до 11:

```
x = np.arange(12)
print(x)
```

одговор: [0 1 2 3 4 5 6 7 8 9 10 11]

Проверимо величину низа:

```
x.shape
```

одговор:12

Претворимо x у дводимензионалну матрицу, где је прва димензија матрице 1:

```
x = x.reshape(1,12)
print(x)
        одговор: [[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11]]
```

Проверимо величину низа:

```
x.shape
        одговор: (1, 12)
```

Претворимо x у матрицу 3x4:

```
x = x.reshape(3,4)
print(x)
        одговор:        [[ 0, 1, 2, 3]
                         [ 4, 5, 6,7]
                         [ 8, 9, 10,11]]
```

Транспоновање вектора и матрица

Транспоновање вектора и матрица је замена индекса врсте и колоне. За транспоновање тензора од три димензије и више, потребно је да наведемо димензију транспоновања.

Креирајмо матрицу A, димензија 3x4 и њену транспоновану:

```
A = np.arange(12).reshape(3,4)
print(A)
```

```
        одговор:        [[ 0,  1,  2,  3],
                         [ 4,  5,  6,  7],
                         [ 8,  9, 10, 11]]
```

Транспонована:

```
A.T
        одговор:        array([[ 0,  4,  8],
                               [ 1,  5,  9],
                               [ 2,  6, 10],
                               [ 3,  7, 11]])
```

Множење матрица

Да бисмо помножили матрицу A и матрицу B, димензија колоне матрице A мора бити једнака димензији врсте матрице B.

```
A = np.arange(6).reshape(3,2)
B = np.arange(6).reshape(2,3)
print(A)
        одговор:    [[0 1]
                     [2 3]
                     [4 5]]
print(B)

        одговор:    [[0, 1, 2],
                     [3, 4, 5]]
```

Помножимо матрице A и B:

```
np.matmul(A,B)

        одговор: array([[ 3,  4,  5],
                        [ 9, 14, 19],
                        [15, 24, 33]])
```

Операције над матрицама

Креирајмо матрицу A:

```
A = np.arange(6).reshape(3,2)
```

Множење матрице члан по члан:

```
print(A*A)
        одговор: array([[ 0,  1],
                        [ 4,  9],
                        [16, 25]])
```

Сабирање матрице:
print(A + A)
odговор:array([[0, 2],
[4, 6],
[8, 10]])

Инверзна матрица

Инверзију можемо да применимо само над квадратном матрицом:

A = np.arange(4).reshape(2,2)
print(A)
odговор:array([[0, 1],
[2, 3]])

Инверзна матрица:
np.linalg.inv(A)
odговор: array([[-1.5, 0.5],
[1.0 , 0.0]])

Сопствена вредност и сопстевени вектор матрице

#Увезимо библиотеке:
from scipy.linalg import eig
import numpy as np

Креирајмо матрицу и израчунајмо сопствене вредности:

A = [[1, 2], # матрица 2x2
[2, 1]]

evals, evecs = eig(A)
(**evals**) and eigenvector (**evecs**) of A.
evecs = evecs[:, 0], evecs[:, 1]

Детерминанте

```
E = [[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]]
print(np.linalg.det(E))
```

Вероватноћа и статистика

Вероватноћа и статистика су гране математике које се баве обрадом случајних појава. Вероватноћа и статистика су математички алати који се користе за описивање неизвесности. Велики број алгоритама за обраду велике количине података гради моделе на основу пробабилистичких информација о узорку или путем статистичког закључивања.
Као и увек до сада, увезимо потребне познате библиотеке:

```
import numpy as np
import scipy as sp
```

Средња вредност:

```
ll = [[1,2,3,4,5,6],[3,4,5,6,7,8]]
```

Израчунавамо средњу вредност свих:

```
np.mean(ll)                    # Izracunava sredjnju vrednost
```

одговор:4.5

Израчунавамо средњу вредност по колони. Вредност 0 означава вектор колоне.

```
np.mean(ll,0)
```

одговор:array([2., 3., 4., 5., 6., 7.])

Израчунавамо средњу вредност по врсти. Вредност 1 означава вектор врсте.
np.mean(ll,1)
odговор:array([3.5, 5.5])

Варијанса

Припремимо податаке:

b=[1,3,5,6]
ll=[[1,2,3,4,5,6],[3,4,5,6,7,8]]

Рачунамо варијансу:

np.var(b)
odговор:3.6875

np.var(ll,1)
Вредност другог аргумента је 1, што нам говори да се варијанса рачуна по врсти.

odговор:[2.91666667 2.91666667]

Стандардна девијација

Прпремимо податак:

ll=[[1,2,3,4,5,6],[3,4,5,6,7,8]]
np.std(ll)
odговор:1.9790570145063195

Коваријанса

```
x = np.array([[1, 2], [3, 7]])
print(np.cov(x))
        одговор:[[0.5 2. ]
                 [2.  8. ]]
```

Коефицијент корелације

```
vc=[1,2,39,0,8]
vb=[1,2,38,0,8]
```
Имплеменација базирана на функцији:
```
np.corrcoef(vc,vb)
        одговор: array([[1.        , 0.99998623],
                        [0.99998623, 1.        ]])
```

Расподела података, Биномна

```
from scipy.stats import binom, norm, beta, expon
import numpy as np
import matplotlib.pyplot as plt
```

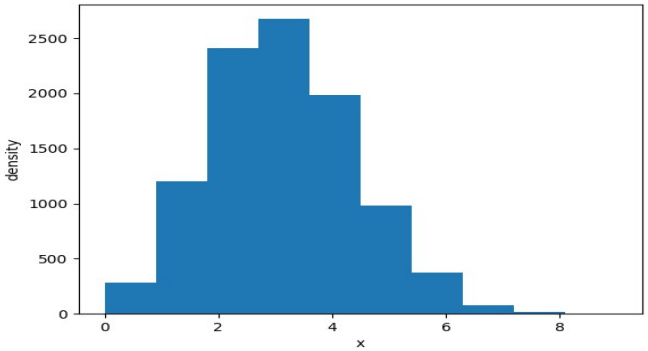
У овом примеру први пут увозимо библиотеку за графички приказ података, **matplotlib.pyplot**. Параметри **n** и **p** указују на број успеха и вероватноћу у биномној формули, респективно, а **size** указује на број времена узорковања.

```
binom_sim = binom.rvs(n=10, p=0.3, size=10000)
print('Data:',binom_sim)
print('Mean: %g' % np.mean(binom_sim))
print('SD: %g' % np.std(binom_sim, ddof=1))
```

Генеришемо хистограм. Параметар **bins** означава укупан број стубова на графику. Подразумевано, збир процената свих трака је 1.

```
plt.hist(binom_sim, bins=10)
plt.xlabel(('x'))
plt.ylabel('density')
plt.show()
```

одговор: Data: [2 4 3 ... 3 4 1]
Mean: 2.9821
SD: 1.43478



Поасонова расподела

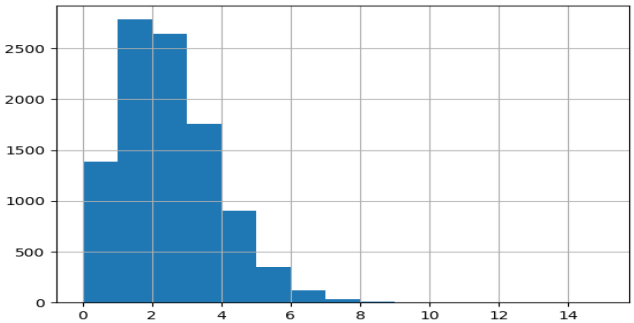
```
import numpy as np
import matplotlib.pyplot as plt
```

Генеришимо 10.000 бројева који су у складу са Поасоновом расподелом где је вредност ламбда 2, (lam=2, size=10000).

```
X= np.random.poisson(lam=2, size=10000)
a = plt.hist(X, bins=15, range=[0, 15])
```

Генеришимо мрежу на графику

```
plt.grid()
plt.show()
```



Нормална расподела

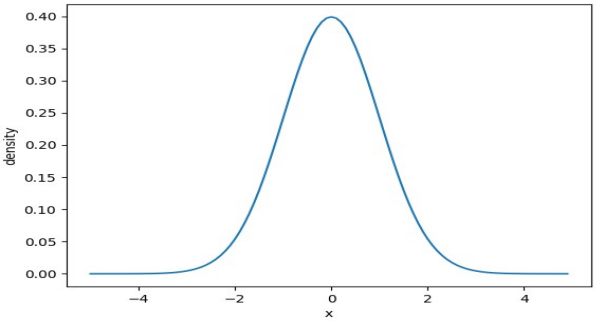
```
from scipy.stats import norm
import numpy as np
import matplotlib.pyplot as plt
mu = 0
sigma = 1
```

Врати равномерно распоређене вредности унутар датог интервала (start, stop, step).

```
x = np.arange(-5, 5, 0.1)
```

Генеришимо нормалну дистрибуцију која је у складу са **mu** и **sigma**:

```
y = norm.pdf(x, mu, sigma)
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('density')
plt.show()
```



Gradient Descent (Градијентно спуштање)

Градијентно спуштање је итеративни алгоритам за оптимизацију првог реда за проналажење минимума функције. У сваком кораку извршава се операција решавање вектора градијента циљне функције. Смер градијента негативан на тренутну позицију се користи као смер претраживања (пошто се циљна функција најбрже спушта у овом правцу, метода градијентног спуштања назива се и метода најстрмијег спуштања). Метода градијентног спуштања има следеће карактеристике: Ако је функција ближа циљној вредности, корак је мањи, а брзина спуштања спорија. Пронађимо локалне минимуме функције $y=(x-6)^2$ почевши од тачке x=1. Лако је пронаћи одговоре израчунавањем $y=(x-6)^2=0$, x=6. Дакле, x=6 је локални минимум функције.

Иницијализујмо параметре, корак учења, learning rate=0.01:

$$\frac{dy}{dx}=\frac{d\left(x-6\right)^2}{dx}=2\mathrm{x}\left(x-6\right) \qquad x_0=1$$

Итерација 1:

$$x_1=x_0-\left(learning\ rate\right)x\frac{dy}{dx}$$

$$x_1=1-0.01\times2\times\left(1-6\right)=1.1$$

Итерација 2:

$$x_2 = x_1 - (learning\,rate)\,x\,\frac{dy}{dx}$$
$$x_2 = 1.1 - 0.01 \times 2 \times (1.1 - 6) = 1.198$$

Пајтон код:

```
cur_x = 1                       # Algoritam pocinje za x=1
rate = 0.01                     # Korak ucenja
precision = 0.000001            #Govori nam kada se zaustavlja
previous_step_size = 1 #
max_iters = 10000               # maksimalan broj iteracija
iters = 0                       #brojac iteracija
df = lambda x: 2*(x-6)          #Gradijent nase funkcije
while previous_step_size > precision and iters < max_iters:
        prev_x = cur_x
        #Sacuvaj vrednos x u  prev_x
        cur_x = cur_x -rate * df(prev_x)
        #Grad descent
        previous_step_size = abs(cur_x -prev_x)
        #Promeni u x
        iters = iters+1 #iteration count
        print("Iteration",iters,"\nX value is",cur_x)
#Ispisi iteracije
print("The local minimum occurs at", cur_x)
```

Одговор, резултат:

```
Iteration 1
X value is 1.1
Iteration 2
X value is 1.1980000000000002
Iteration 3
X value is 1.29404
Iteration 4
X value is 1.3881592
Iteration 5
X value is 1.480396016
Iteration 6
X value is 1.57078809568
...
```

...
Iteration 570
X value is 5.99995013071414
Iteration 571
X value is 5.999951128099857
The local minimum occurs at 5.999951128099857

За математичаре или некога ко се темељно бави математиком, након овог поглавља остало је много недореченог када говоримо о математици, много више тачака од три....., али сетимо се, циљ нам је да након читања овог поглавља, разумемо како се у Пајтону решавају основни математички изрази које можемо да сретенемо у пракси, а за даље, па континуирано учење и усавршавање, а ми настављамо са обрадом података и елементима машинског учења, као наставак математике.

Пајтон и елементи машинског учења и обраде сигнала

Машинско учење, једноставно речено, представља низ мање или више сложених математичких процедура којима се машина, рачунар обучава да препозна неке појаве са већом или мањом тачношћу. Развој рачунарског хардвера и повећање перформанси рачунара које имамо на располагању, омогућавају да се сви ти алгоритми, који су, узгред буди речено, познати још од педесетих година двадесетог века, сада могу покренути и у задовољавајуће дугом временском периоду извршити. Ова могућност отворила је врата стручној и научној јавности широм света да може да се бави применом и развојем нових алгоритама. Када отоворимо било коју књигу која се бави озбиљном темом машинско учење, базни алгоритми који се неизоставно обрађују, су регресије и класификатори. Ово поглавље има за циљ да представи читаоцима начин решавања регресионих проблема коришћењем Пајтона, не улазећи у теорију машиског учења, детаљног описа алгоритама, јер је то тема којом се баве многе публикације које у свом наслову имају

нешто као Пајтон и машинско учење. Наш циљ је да уколико се нађемо у прилици да решавамо неки задатак који садржи неки регресиони проблем, читањем овог поглавља, уз примере који иду уз њега, будемо у стању да дођемо на почетну позицију, а касније, уосталом као и све у данашњој науци и техници, је ствар усавршавања.

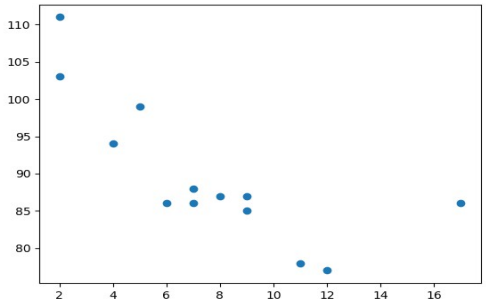
Линеарна регресија

Задатак линеарне регресије је да користећи позната растојања између тачака података, повуче праву линију кроз све њих, тако да колико је то могуће буде близу сваке тачке, што је врло ретко могуће, а у неким случајевима, у зависности од природе података и немогуће.
Добијена регресиона права се може користити за предвиђање будућих вредности, па смо на тај начин, израчунавањем једначине праве, научили машину да предвиди догађај, вредност податка који се појави у спектру података, рецимо мерењем излаза неког сензора. У машинском учењу, предвиђање будућности је веома важно и представља суштину машинског учења. Када бисмо се бавили математичком теоријом која описује алгоритме машинског учења, па након тога пробали да их имплементирамо у Пајтону, наишли бисмо на огроман програмерски подухват. На срећу, Пајтон је оно што је, баш из разлога што има методе за проналажење односа између тачака података и за цртање линије линеарне регресије, па можемо да користимо ове методе уместо да пролазимо кроз математичкке једначине.
У примеру којим ћемо да објаснимо Пајтон и линеарну регресију, претпоставимо да x-оса представља неке улазне, измерене податке, а y-оса представља резултат који се добија као последица улазник података.
Први корак је да погледамо како су распоређени подаци, а најлакши начин је да их прикажемо графички.

Користимо већ познате библиотеке, креирамо низове и приказјемо их:

```
import matplotlib.pyplot as plt
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

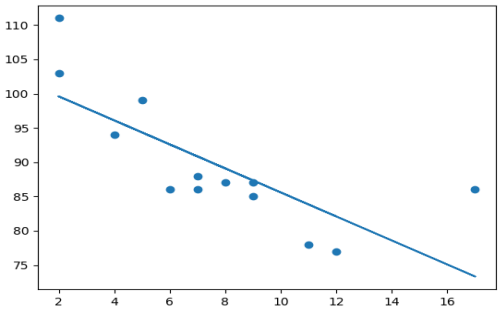
Одговор:



Нацртајмо линију линеарне регресије, користећи stats из scipy и функцију stats.linregress(x, y).

```
from scipy import stats
#Креирајмо низове који представљају вредности x и y осе:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
#Извршимо метод који враћа неке важне кључне
#вредности линеарне регресије:

slope, intercept, r, p, std_err = stats.linregress(x, y)

#Креирајмо функцију која користи вредности нагиба и пресека да
#врати нову вредност. Ова нова вредност представља где ће на
```

```
#y-оси бити постављена одговарајућа вредност x:

def myfunc(x):
  return slope * x + intercept

#Провуцимо сваку вредност x низа кроз функцију. Ово ће
#резултирати новим низом са новим вредностима за y-осу

mymodel = list(map(myfunc, x))

# Нацртајмо график, линију

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```
Одговор:



Већ смо напоменули да је задатак линеране регресије да предвидимо понашање, да добијемо што је могуће тачнији резултат за непознату улазну променљиву.
Због тога је важно да се утврди какав је однос између вредности улазних и излазних променљивих, x и y оса, јер ако не постоји веза, линеарна регресија се не може користити да било шта тачно предвиди. Овај однос се назива коефицијент корелације и означимо га са **r**.
Вредност коефицијента корелације креће се од -1 до 1, где 0 значи да нема корелације, а 1 (и -1) значи да је корелација 100%.

Пајтон, коришћењем Scipy модула, израчунава ову вредност за нас, све што треба да урадимо је да унесемо у функцију stats.linregress(x, y) вредности x и y и прикажемо резултат.

```
from scipy import stats
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

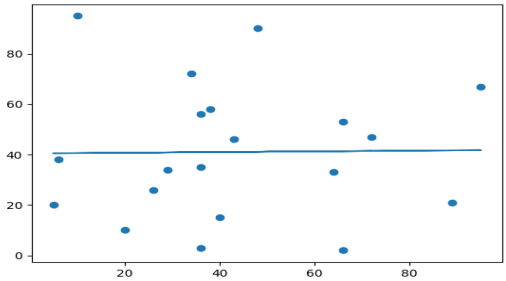
одговор:-0.758591524376155

Максимална корелација је 1 или -1, добили смо -0.759.

Креирајмо пример где линеарна регресија не може да се користи за предвиђање параметара.
Користимо потпуно исти пример, само је сет података другачији, што ћемо из графика и коефицијента корелација да видимо:

```
import matplotlib.pyplot as plt
from scipy import stats
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
print(r)
```

Одговор:



Koeficijent korelacije : 0.01331814154297491

Слика и вредност коефицијента корелације све кажу, на овај скуп података не можемо да користимо линерну регресију за предвиђање.

Када говоримо о линеарној регресији и регресији уопште, једна од основних функција и појмова је loss функција, функција губитака, која дефинише меру одступања тренутне вредности параметара од стварних.
Тренутна вредност параметара, у литератури се назива хипотеза код линеране регресије изражена је као:

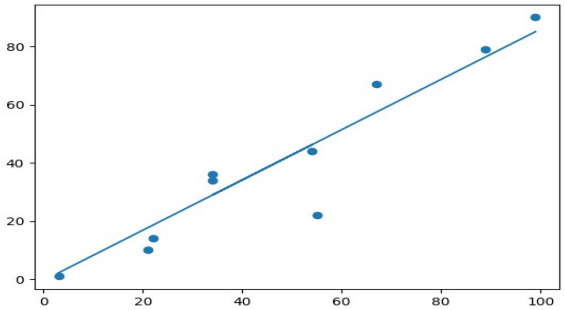
$$h(x) = w_0 + w_1 x$$

где су w параметри модела.
Функција грешке, **error/cost** функција, је просечна вредност функције губитака на свим подацима из посматраног скупа и код линеарне регресије најчешће се као функција грешке користи квадрат просека одступања, енлгески ***Mean Squared Error***.
У следећем примеру видећемо како се користе ове функције у Пајтону.
Креирајмо 10 података који се налазе у линерарној релацији. Подаци се конвертују у формат низа тако да могу директно да се користе за израчунавања.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# define data, and change list to array
x = [3,21,22,34,54,34,55,67,89,99]
x = np.array(x)
y = [1,10,14,34,44,36,22,67,79,90]
y = np.array(y)
#Основни модел линеарне регесије је дводимензионални wx+ b,
#па дефинишемо модел ax+ b
def model(a, b, x):
  return a*x + b
#Као што је речено  најчешће се као функција грешке користи
#квадрат просека одступања

def loss_function(a, b, x, y):
  num = len(x)
  prediction=model(a,b,x)
  return (0.5/num) * (np.square(prediction-y)).sum()
#Функција оптимизације углавном користи парцијалне изводе  за
# ажурирање параметара а и б

def optimize(a,b,x,y):
  num = len(x)
  prediction = model(a,b,x)
#Ажурирање вредности А и В проналажењем парцијалних
#извода
  da = (1.0/num) * ((prediction -y)*x).sum()
  db = (1.0/num) * ((prediction -y).sum())
  a = a - Lr*da
  b = b - Lr*db
  return a, b
#функција итерација враћа а и б
def iterate(a,b,x,y,times):
  for i in range(times):
    a,b = optimize(a,b,x,y)
  return a,b
#Покрећемо итерацију
#Инцијализујемо параметре
a = np.random.rand(1)
print(a)
b = np.random.rand(1)
```

```
print(b)
Lr = 1e-4
#рачунамо и приказујемо прву, другу, трећу,,,хиљадиту итерацију
#Прва итерација
a,b = iterate(a,b,x,y,1)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
#Друга итерација
a,b = iterate(a,b,x,y,2)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
#Трећа итерација
a,b = iterate(a,b,x,y,3)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
# 10000-а итерација
a,b = iterate(a,b,x,y,10000)
prediction=model(a,b,x)
loss = loss_function(a, b, x, y)
print(a,b,loss)
plt.scatter(x,y)
plt.plot(x,prediction)
plt.show()
```



Као резултат имамо приказан график и кораке током итерација

па можемо да пратимо конвергенцију вредности жељеној.
Комплетан пројекат се налази у додатку књиге.

Завршићемо линеарну регресију једним реалним примером из медицине, наиме применићемо линеарну регресију на процену дозе зрачења пацијената, на основу скупа података који смо добили табеларно, у овом случају у форамату xlsx.
Сада се први пут срећемо са учитавањем података из табеле, признаћемо, у дигиталној ери, све се своди на дигиталан облик и чување података у некој бази података, а база није ништа друго већ складиште разичитих табела података међусобно повезаних на одређени начин. Пајтон омогућава читање података из табела, како xlsx, тако и свих остали формата, csv, txt.
За ове намене увозимо библиотеку **pandas** која се користи за анализу података.
Ако је табела у csv формату, што је уобичајени формат за слободне алате, попут LibreOffice-а, у коме је између осталог и написана ова књига, учитавамо је на следећи начин:
df = pandas.read_csv("Air_kerma.csv")

Назив датотека са подацима у овом примеру је Air_kerma.csv и налази се у истом директоријуму са програмом који је учитава, а можемо и да наведемо апсолутну путању до датотеке.
За пример који обрађујемо, подаци су у Excel формату, па их учитавамо:

```
excel_file = "air_kerma_predi.xlsx"
df = pd.read_excel(r"air_kerma_predi.xlsx")
```

Назив датотеке са подацима је air_kerma_predi.xlsx.
Пајтон **pandas** библиотека омогућава рад са подацима, па ћемо искористити неке њене функционалности да прикажемо податке, приступимо одређеним колонама, обрадимо их и сачувамо у посебну датотеку, коју смо назвали predicted.csv. Обратимо пажњу на процедуру креирања датотеке и чувања података.

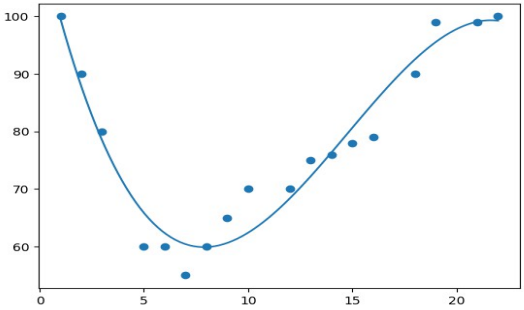
```
from scipy import stats
from sklearn import linear_model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use("seaborn-whitegrid")
from IPython.display import display
import numpy
#df = pandas.read_csv("Air_kerma.csv")
excel_file = "air_kerma_predi.xlsx"
#dtf = pd.read_excel(r"dodatni podaci.xlsx",sheet_name='ASTMA')
df = pd.read_excel(r"air_kerma_predi.xlsx")

pd.set_option('display.max_columns', None)
display(df)
x = df['DAP_Gycm2']
y = df['Air_Kerma']
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

kerma = myfunc(x)
kerma.to_csv('predicted.csv', index=False)
print(kerma)
```

Полиномска регресија

Ако наше тачке података очигледно не одговарају линеарној регресији (права линија кроз све тачке података) или врло низак коефицијент корелације, можда би могао да се користи за полиномску регресију. Полиномска регресија, као и линеарна регресија, користи однос између променљивих x и y да пронађе најбољи начин за цртање линије кроз тачке података. Наравно, Пајтон има методе за проналажење односа између тачака података и за цртање линије полиномске регресије. Показаћемо како да искористимо ове методе уместо да пролазимо кроз математичке алгоритме. Креирајмо два низа података као у случају линеарне регресије:

```
import numpy
import matplotlib.pyplot as plt
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
#NumPy има метод који нам омогућава да направимо
#полиномски модел
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
#Затим одредимо како ће се линија приказати, почињемо на
#позицији 1, а завршавамо на позицији 22:
myline = numpy.linspace(1, 22, 100)
# Цртамо оригиналну линију
plt.scatter(x, y)
#Цртамо линију полиномске регресије
plt.plot(myline, mymodel(myline))

#Приказујемо график
plt.show()
```

Одговор:

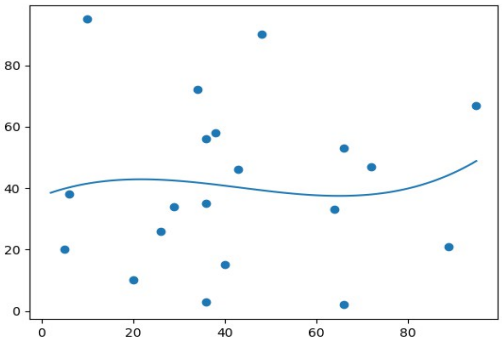


Коефицијент корелације: 0.9432150416451026

А пробајамо са другим сетом података:

```
import numpy
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
myline = numpy.linspace(2, 95, 100)
plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
print(r2_score(y, mymodel(x)))
```

Одговор:



Коефицијент корелације: 0.009952707566680652

Резултат очигледно није добар, што значи да не можемо да користимо полиномску регресију за предвиђање овог сета података. Уколико је полиномска регресија употребљива, резултат можемо да предвидимо на следећи начин. Креирајмо сет података и модел полиномске регресије:

```
import numpy
from sklearn.metrics import r2_score
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
# Желимо да предвидимо излазну променљиву ако је мерени
#улаз 17
predict = mymodel(17)
print(predict)
```

Одговор:
Предвиђена вредност је: 88.87331269698001

Multiple регресија(вишепараметарска регресија)

Сетимо се линеарне регресије и нагласимо чињеницу да смо имали само један параметар на основу кога смо процењивали, предвиђали будуће параметре.
Сложићемо се да у пракси имамо много примера са више параметера. Вишепараметарска регресија је попут линеарне регресије, али са више од једне независне вредности, што значи да покушавамо да предвидимо вредност на основу две или више променљивих.
Примену вишепараметарске регресије показаћемо на истом примеру процене дозе зрачења, само што ћемо у овом случају у процену да укључимо три параметра, **'Time', 'Age','DAP_Gycm2'** из датотеке подака **multi_reg.xlsx**, на основу којих процењујемо параметар **Air_Kerma** и процењене вредности чувамо у датотеку **predicted_multi_bez_age.csv**. Комплетан пример са датотеком података се налази као додатак ове књиге, као и сви примери из свих погалавља, а код је приказан у наставку:

```
from sklearn import linear_model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use("seaborn-whitegrid")
from IPython.display import display
import numpy
#df = pandas.read_csv("Air_kerma.csv")
excel_file = "multi_reg.xlsx"
#dtf = pd.read_excel(r"dodatni podaci.xlsx",sheet_name='ASTMA')
df = pd.read_excel(r"multi_reg.xlsx")
#print(df.head())
# postavi max columns na none
pd.set_option('display.max_columns', None)
display(df)
X = df[['Time', 'Age','DAP_Gycm2']]
#X = df[['Time', 'DAP_Gycm2']]
y = df['Air_Kerma']
```

```
################################################end
polinom
regr = linear_model.LinearRegression()
regr.fit(X, y)
print(regr.coef_)
import pandas
from sklearn import linear_model
excel_file = "multi_reg.xlsx"
#dtf = pd.read_excel(r"dodatni podaci.xlsx",sheet_name='ASTMA')
df = pd.read_excel(r"multi_reg.xlsx")
#print(df.head())
# postavi max columns na none
pd.set_option('display.max_columns', None)
display(df)
X = df[['Time', 'DAP_Gycm2']]
y = df['Air_Kerma']
regr = linear_model.LinearRegression()
regr.fit(X, y)
predicted_kerma = regr.predict(X)
#print('Za Time 2.5, Age 60, DAP_Gycm2 75.026')
print('predicted_Air_Kerma:', predicted_kerma)
# sacuvaj numpy array kao csv file
from numpy import asarray
from numpy import savetxt
#predicted_kerma.to_csv('predicted_kerma.csv', index=False)
savetxt('predicted_multi_bez_age.csv',predicted_kerma, delimiter=',')
```

SVM(support vector machine)модел

На крају овог поглавља, даћемо пример креирања SVM модела, без претензија да улазимо у детаље овог класификатора који је нашао своју широку примену, са идејом да уколико се пред читаоцима у било ком тренутку јави потреба да се баве класификаторима, конкретно SVM моделом класификатора, да могу да искористе овај пример, крену са радом и прилагоде га свом пројетку.

Наравно, подразумева се да ће у том случају морати да проуче теоријске основе класификатора, конкретног класификатора, што превазилази обим и намену ове књиге. Комплетан пројекат који описујемо, налази се као додатак уз књигу.

Прво, увезимо SVM модул и креирајмо објекат SVM класификатора прослеђивањем аргумента кернела као линеарног кернела у функцији SVM().

Затим подесимо, фитујемо свој модел на скупу података који се назива тренинг скуп података, **train set** користећи **fit()** и извршимо предвиђање на тест скупу података, помоћу **predict()**.

```
# Креирамо svm класификатор
clf = svm.SVC(kernel='linear')                # Linear Kernel
#Тренинг коришћењем training sets
clf.fit(X_train, y_train)
#Предикција  на дата сету података
y_pred = clf.predict(X_test)
#Хајде да проценимо колико тачно класификатор или модел
#може предвидети рак дојке код пацијената. Тачност се може
#израчунати упоређивањем стварних вредности тест скупа и
#предвиђених вредности.
from sklearn import metrics
# Процена тачности модела
print("Tacnost:",metrics.accuracy_score(y_test, y_pred))
одговор:  0.9649122807017544
#Покретањем програма добијамо резултат за тачност
#0.9649122807017544 што је одличан резултат
```

Дошли смо до краја овог поглавља у коме смо решавали неке медицинске проблеме, користили изразе за које и не знамо, или знамо врло површно шта значе и чему служе, али то нам је довољно, учимо филозофију Пајтона и његову примену на

различите проблеме, у овој књизи описана је само мрвица погаче различитих пројеката који су нам јавно доступни, важно је да разумемо то што преузмемо, а за даље, како рекосмо, три тачке...., а нека прва следећа буде Обрада слике.

ПАЈТОН (PYTHON) и обрада слике

Развојем вештачке интелигенције, обрада слике постаје неизоставан сегмент примене Пајтона у различитим апликацијама, од препознавања лица, детекције објеката, даљинског управљања, геогафских информационих система, различите врсте праћења објеката, обраде медициских слика и …

Када говоримо о обради слике, најпопуларнија библиотека за процесирање слика као и видеа који није ништа друго, него низ слика, такозваних фрејмова, је OpenCV (Open Source Computer Vision Library).

Креирана је 2000-те године од стране Интела и данас је практично стандардни алат у истраживању и развоју у области рачунарског вида (Computer Vision).

Основна верзија је напиасана у програмском језику C++, али постоје интерфејси и за програмкси језик Пајтон. Коришћењем OpenCV-а довољно је основно знање Пајтона да можемо да напишемо различите програме за обраду слике.

Да бисмо користили OpenCV библиотеке, први корак је инсталација OpenCV-а, што је најлакше урадити коришћењем пакета ***pip***, који је пре инсталације потребно ажурирати, што радимо командом у конзоли:

pip install --upgrade pip

Основне, главне модуле OpenCV-а инсталирамо инструкцијом:
pip install opencv-python

Инсталација ће потрајати неко време, након завршетка

инсталирали смо основне модуле. Постоји проширена верзија OpenCV-a, која садржи додатне модуле, који се користе за детекцију објеката, GPU(Graphical Processing Unit) оптимизацију, па препоручујемо да OpenCV инсталирамо са додатним модулима одмах на почетку и то радимо инструкцијом:

pip install opencv-contrib-python

На овај начин смо сигурни да имамо инсталирано све што је тренутно на располагању у области рачунарског вида, а покривено OpenCV библиотекама.
Проверу да ли смо успешно инсталирали OpenCV и која верзија је инсталирана, урадићемо једноставном инструкцијом за увоз OpenCV библиотеке и провером верзије:

```
import cv2
print(cv2.__version__)
```

одговор:4.6.0

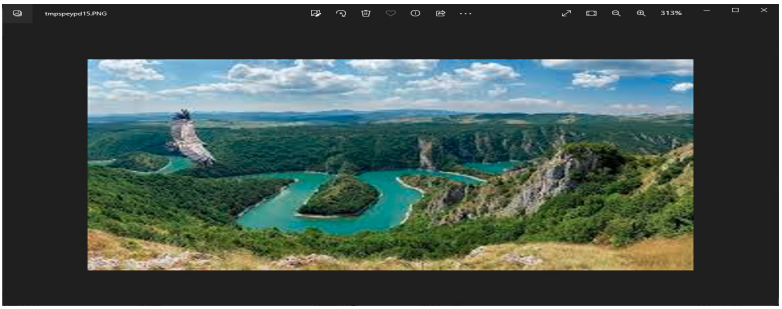
На рачунару на коме је покренута ова инструкција, инсталирана је верзија 4.6.0

Уколико нисмо инсталирали OpenCV, за брзо учитавање и обраду слике можемо да користимо и PIL библиотаке, као на пример:

```
from PIL import Image
from PIL import ImageOps
```

Учитавамо и приказујемо слику:

```
im = Image.open("Reka.jpg")
print(im.format, im.size, im.mode)
im.show()
```



Обзиром да смо инсталирали OpenCV и констатовали да се ради о важном алату у области рачунарског вида, вратимо се на OpenCV и његову примену на истој слици:

```
import cv2
img = cv2.imread("Reka.jpg")
cv2.imshow("Reka",img)
cv2.waitKey(0)
```

Обратимо пажњу на инструкцију, cv2.waitKey(0), која нам омогућава да се слика прикаже и да се чека притисак са тастатуре, 0 значи да чекамо бесконачно, до притиска тастера ESC у овом случају. Ако уместо 0 унесемо неки број, тај број означава чекање у милисекундама.

Након учитавања слике, пре обраде можемо да проверимо основне карактеристике слике, на следећи начин:

```
import cv2
img = cv2.imread('Reka.jpg')                    # ucitavanje slike
(rows, cols, channels) = img.shape              # dimenzija slike
print("Dimenzije slike u boji (visina, sirina, dubina):", rows, cols, channels)
print("Tip promenljive piksela:", img.dtype)
cv2.imshow("Slika", img)                        # prikaz slike
cv2.waitKey(0)
```

Резултат програма горе може да буде грешка следећег типа:
AttributeError: 'NoneType' object has no attribute 'shape', што значи да смо позвали атрибут '**shape**' који није дефинисан, већ је тип **None**.
Проверу да ли је атрибут None, можемо да урадимо додавањем следећих наредби:

```
if img is not None:
   print('variable is not None')
   print(img.shape)
else:
   print('variable is None')
```

одговор: variable is None

што нам као одговор даје потврду да је променљива **None**.
Решење проблема је да при учитавању слике наведемо апсолутну путању до директоријума у коме се слика налази, што је за конкретан пример на Windows оперативном систему:

```
C:\AP\projekti\Python\Knjiga\obrada_slike\Primeri_slika\Reka.jpg'
```

а пример учитавања и провере особина слике:

```
import cv2
print(cv2.__version__)
img = cv2.imread(r'C:\AP\projekti\Python\Knjiga\obrada_slike\Primeri_slika\Reka.jpg')
if img is not None:
   print('variable is not None')
   print(img.shape)
else:
   print('variable is None')
(rows, cols, channels) = img.shape        # dimenzije slike kao matrice
print("Dimenzije slike u boji (visina, sirina, dubina):", rows, cols, channels)
```

```
print("Tip promenljive piksela:", img.dtype)
cv2.imshow("Slika", img)                    # prikaz slike na ekran
cv2.waitKey(0)
```

одговор:
```
4.6.0
variable is not None
(163, 310, 3)
Dimenzije slike u boji (visina, sirina, dubina):
163 310 3
Tip promenljive piksela: uint8
```

а учитана слика:

Дигитална слике ја матрица елемената који се називају пиксели и број пиксела зависи од резолуције камере којом је слика снимљена.
За приказ слике користимо инструкцију cv2.imshow("Slika", img), где је под наводницима наслов учитане слике img.
Интензитет пиксела су целобројне вредности што смо и добили као одговор Tip promenljive piksela: uint8.
Боја пиксела се добија као комбинација три основне боје, црвене, зелене и плаве, популано, RGB, свака од три наведене боје се приказује целим бројем у опсегу 0-255.
Комбинацијом вредности за сваку појединачну компоненту, добијамо различите боје пиксела, па на пример, црвена боја је комбинација (255, 0,0), зелена (0,255, 0), плава(0,0, 255), црна боја (0,0,0), а бела (255,255,255).

Ово је генерално правило за RGB приказ пиксела. Морамо да обратимо пажњу и имамо на уму да је у OpenCV-у редослед боја промењен и гласи, BGR, па у скаду са тим комбинујемо вредности и добијамо боју пиксела.
Дакле, дигитална слика је практично тродимензионална матрица која се састоји од три матрице, што су у ствари канали боје. Ако желимо да променимо неки пиксел и да му приступимо, потребно је да знамо врсту, колону и канал жељеног пиксела.
Инструкцијом *img[10, 60, 2]*, приступамо пикселу у 10-ом реду, 60-ој колони матрице слике, при чему узимамо само *R* канал. Сетимо се код OpenCV-а, редослед је BGR. Наравно, врло често су нам потребна сва три канала одређеног пискела, што добијамо **не навођењем** треће компоненте, *img[10, 60].* На пример:

```
# Pristup pikselu na poziciji [10, 60]
(b, g, r) =  img[10, 60]
print("Komponente boje piksela (blue, green, red):", b, g, r);
        odgovor: Komponente boje piksela (blue, green, red):
                225 185 143
```

или ако желимо сваки канал посебно:

```
b = img[10, 60, 0]          # B komponenta boje piksela
print("Komponenta blue:", b);
g = img[10, 60, 1]          # G komponenta boje piksela
print("Komponenta green:", g);
r = img[10, 60, 2]          # R komponenta boje piksela
print("Komponenta red:", r);
        odgovor:        Komponenta blue: 225
                        Komponenta green: 185
                        Komponenta red: 143
```

Видимо да је одговор исти само је приказан за појединачне канале.
Исти резултат, али брже извршење добијамо инструкцијом :

```
img.item(red, kolona, kanal)
```

па за исти пиксел из претходног примера добијамо:

```
b = img.item(10, 60, 0) # B komponenta boje piksela
g = img.item(10, 60, 1) # G komponenta boje piksela
r = img.item(10, 60, 2) # R komponenta boje piksela
print("Komponente boje piksela (blue, green, red) - varijanta 2:", b, g, r);
```
одговор: Komponente boje piksela (blue, green, red) - varijanta 2: 225 185 143

Резултат је исти, а приликом покретања програма, примећујемо осетну разлику у брзини извршавања.
Сада када смо научили како да приступимо одређеном пикселу, можемо лако да променимо његову боју, на пример променимо боју пиксела из претходих примера у црвену:
```
img[10, 60] = (0, 0, 255)                    # Боји пиксел u црвено
```

Исти ефекат, само се брже извршава као у прошлом примеру:
```
img.itemset((10, 60, 0),0)
img.itemset((10, 60, 1) ,0)
img.itemset((10, 60, 2) ,255)
```
У многим применама обраде слике, доћи ћемо у ситуацију да издвојимо део слике, на пример код обраде медицинских слика, издвајање региона где се налази тумор, или код саобраћајних камера издвајање региона где се налази таблица возила, код препознавања лица, издвајање лица и ...
Издвојени регион у литератури се назива ROI(Region Of Interest) и помоћу OpenCV-a и NumPy-a можемо да издвојимо жељени регион од интереса на слици, што радимо тако што наводимо матрицу ограничену врстама и колонама, на пример, врстама 100 и 190 и колонама 40 и 340:
```
roi = img[100:190, 40:340]
cv2.imshow("Izdvojen ROI", roi)
cv2.waitKey(0)
```
Одговор:

Нагласимо овде да се оператор две тачке може користити без навођења почетног и коначног индекса, тако да се у том случају подразумевају сви елементи одговарајућег индекса.
На пример img[: , 50:550] означава регион који укључује све редове слике у колонама од 50 до 550. Уколико желимо да издвојимо сваки канал боје као једну дводимензионалну матрицу, урадићемо то тако што ћемо изабрати све типове и све колоне, али само један канал боје по дубини:

```
B = img[:, :, 0] # B kanal boje kao podmatrica
G = img[:, :, 1] # G kanal boje kao podmatrica
R = img[:, :, 2] # R kanal boje kao podmatrica
```

Често је за процесирање и даљу обраду погоднија једноканална, сива слика, па горе наведено можемо да применимо да добијемо сиву слику.

```
# Y = 0.299*R + 0.587*G + 0.114*B
img_gray = np.uint8( 0.299*R + 0.587*G + 0.114*B )
cv2.imshow("Siva slika (grayscale, monohromatska, 8-bitna)", img_gray)
cv2.waitKey(0)
```

одговор:

Исто ово могуће је постићи позивом OpenCV функције *cv2.cvtColor().*

```
imgGry = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Siva slika poziv cv2.cvtColor )", imgGry)
cv2.waitKey(0)
```

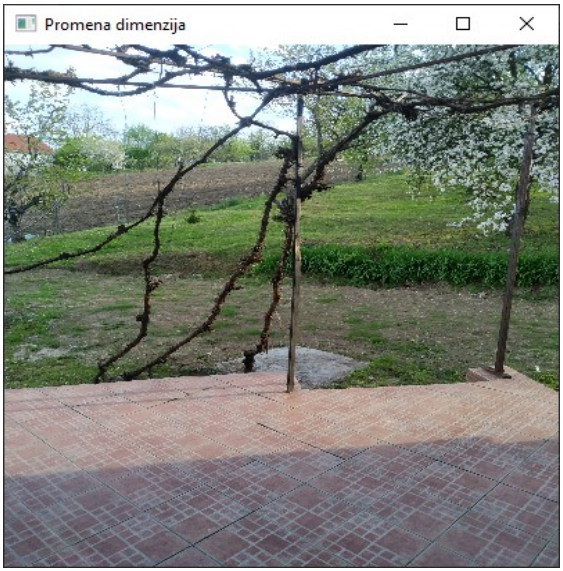
Одговор:

Промену величине слике можемо да користимо из више разлога, на пример у случајевима да комплетну слику не можемо да прикажемо на екрану на пример слика доле:

Да бисмо видели комплетну слику, променићемо димензије слике на следећи начин:

```
import cv2
img=cv2.imread(r'C:\AP\projekti\Python\Knjiga\obrada_slike\Primeri_slika\rekovac.jpg')
cv2.imshow("Rekovac",img)
cv2.waitKey(0)
img_resized = cv2.resize(img, (300, 300))
cv2.imshow("Promena dimenzija ", img_resized);
cv2.waitKey(0)
```

Одговор:



Променили смо димензију слике тако што смо дефинисали фиксне димензије слике на 300x300 пиксела, што значи да нисмо водили рачуна о правом односу ширине и висине слике, па слика може да буде деформисана.
Да бисмо смањили величину слике, а сачуавали однос висине и ширине, потребно је да израчунамо однос висине и ширине и применимо га на умањеној слици. Задржимо се на фиксној ширини слике 300 пиксела, али сада желимо да добијемо висину тако да однос ширине и висине и на смањеној слици одговара оригиналу.
Дефинисаћемо *colona_resized=300* и постављамо пропорцију:
colona/vrsta = colona_resized /vrsta_resized
па је: *vrsta_resized = colona_resized*vrsta/colona*

У Пајтону то можемо да урадимо на следећи начин:

```
(vrsta, colona, depth) = img.shape
colona_resized = 400; # fiksiramo širinu nove slike
vrsta_resized = int(colona_resized * vrsta / colona)
img_resized = cv2.resize(img, (colona_resized, vrsta_resized))
cv2.imshow("Smanjena zadrzan odnos", img_resized);
cv2.waitKey(0)
```

Одговор је слика смањених димензија, али истог односа ширине и висине:

Узмимо пример обраде медицинских слика, применом алгоритама вештачке интелигенције или машинског учења са циљем да детектујемо тумор на мамографском снимку.
Након дефинисања региона од интереса, кориснику програма је од интереса да види дефинисани регион, што повлачи потребу за означавањем региона од интереса, на пример цртањем правоугаоника на слици.
Слична потреба се јавља и на примеру детекције или праћења објекта, објекат од интереса се оивичава правоугаоником и исписује текст који описује објекат.
OpenCV има функције које нам омогућавају цртање и писање на слици.

Нацртајмо правоугаоник на слици из претходног примера:

```
out_img = img_resized.copy()
top_left_x = 10 # x koordinata - odgovara koloni matrice
top_left_y = 10 # y koordinata - odgovara redu matrice
bottom_right_x = img.shape[1] - 10                # sirina pravougaonika
bottom_right_y = top_left_y + 30
rect_color = (0, 255, 0) # BGR
cv2.rectangle(out_img, (top_left_x, top_left_y), (bottom_right_x, bottom_right_y), rect_color, 1)
cv2.imshow("Nacrtan zeleni pravougaonik", out_img);
cv2.waitKey(0)
```

Одговор:


Nacrtan zeleni pravougaonik

У правоугаонику можемо да напишемо и жељени текст на следећи начин:

```
font_face = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 0.6
font_color = (0, 0, 255) # BGR
font_thick = 1 # integer
bottom_left_x = top_left_x + 10; # x - odgovara koloni matrice
bottom_left_y = bottom_right_y - 10; # y - odgovara redu matrice
cv2.putText(out_img, "OVO JE TEKST", (bottom_left_x, bottom_left_y), font_face, font_scale, font_color, font_thick)
cv2.imshow("Rezultat", out_img)
cv2.waitKey(0)
```

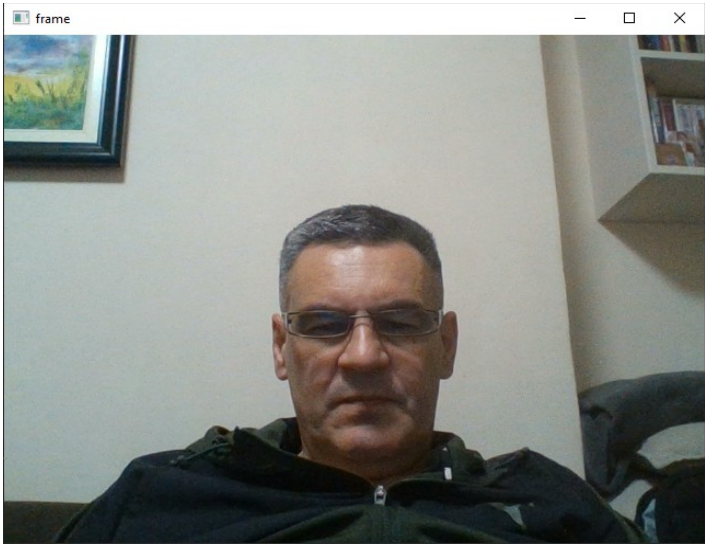
Одговор:


Rezultat
OVO JE TEKST

Видео процесирање

Поред слике, OpenCV омогућава и покретање камере и обраду видео записа.
Камеру покрећемо инструкцијом cam = cv2.VideoCapture(0), где 0 означава индекс расположивих камера, а ако их има више, онда би се следећа позивала навођењем броја 1 у функцији.
Пример покретања камере:

```
# uvezimo opencv biblioteku
import cv2
# definisemo  video capture objekat
vid = cv2.VideoCapture(0)
while (True):
    # Capture video frame
    # po  frame
    ret, frame = vid.read()
    # Prikazi rezultujuci frame
    cv2.imshow('frame', frame)
    # taster 'q' je setovan za napustanje programa
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Nakon petlje napustimo video
vid.release()
# Zatvorimo sve prozore
cv2.destroyAllWindows()
```



Поред обраде слике у реалном времену, директно са камере, имаћемо потребу за обрадом снимљеног видео записа, на пример у mp4 формату.
Поступак је готово исти, у претходом примеру променићемо само
vid = cv2.VideoCapture(0), коментарисањем ове инструкције и додавањем vid = cv2.VideoCapture('zvezda.mp4'), уместо индекса камере, наводимо назив видео датотеке или апсолутне путање до видео датотеке, у конкретном случају, коментарисањем:
#vid = cv2.VideoCapture(0)
и додававањем линије кода:
vid = cv2.VideoCapture('zvezda.mp4')
са нагласком да наведена видео датотека мора да се налази у истом директоријуму или можемо да додатамо апсолутну путању до видео датотеке.
Коришћен је видео zvezda.mp4, власништво РСТ-а, преузет са интернета. Као резултат ове промене у коду, добијамо:

За вежбу можемо да користимо неки други видео у неком другом формату, avi формат на пример.

ПАЈТОН и географски информациони системи

Географски информациони систем (ГИС) представља моћан алат који омогућава просторни приказ и анализу података из најразличитијих области и широког спектра примене.
Не постоји област у савременом свету данас, где је потребно просторно приказати неке податке, а да се не користи ГИС. Поред визуелизације података, што је свакако важно, једна од најважнијих особина ГИС-а је да је то моћан алат за анализу просторних података.
У свом називу има епитет информациони систем, али то је једини инфорамциони систем који ради са просторним базама података и то га чини јединственим. За рад са ГИС-ом користе се различити алати, од професионланих, до алата отвореног кода и сваки понаособ има своје предности и недостатке.
О ГИС-су су написане многе књиге, за оне који желе да се детаљније упознају са овом темом, циљ овог поглавља је да прикаже могућност примене Пајтона и програмирања у Пајтону за просторно приказивање података.
Пре свега, Пајтон је изузетно користан програмски језик у шта смо до сада могли да се уверимо, а када говоримо о ГИС-у, многи или већина различитих ГИС софтверских пакета, као на пример ArcGis, Qgis, омогућавају интерфејс за рад са Пајтон скриптама, што је Пајтон код имлементиран у ГИС софтвер.
Међутим, Пајтон омогућава ГИС примене без икаквог додатног софтвера, ГИС алата, таква врста примена је циљ овог поглавља и постоји неколико разлога за овакавим приступом.
Као прво, све је бесплатно, не постоји потреба за куповином било каквих скупих лиценци да би смо користили Пајтон скрипте у неком програмском језику, ArcGis на пример.
Коришћење Пајтона подразумева писање кода, програма, што за последицу има дубље разумевање како различите операције геопрецесирања функционишу.
Пајтон је врло ефикасан, тако да може да се користи за анализу велике количине података, на другој страни врло је флексибилан, па подржава све постојеће типове података.

Коришћење Пајтона као програмског језика отвореног кода, омогућава свима да користе ресурсе и допринесу својим радом развоју програмског језика. Пајтон даје могућност повезивања са различитим софтвером неких других, независних произвођача.
За рад са геопросторним подацима, потребно је као и до сада увести одређене библиотеке, припремити радно окружење. У зависности од верзије Пајтона и оперативног система који се користи, бирамо тип библиотека. За Windows 10 и Пајтон 3.10 на рачунару на коме се пише ово поглавље, процедура инсталације подразумева неколико корака:

1. Преузимање **gdal** и **fiona** за верзију оперативног система и Пајтона.
2. Инсталација Microsoft Visual C++ 14.0 или новијег је неопходна уколико је немамо инсталирану
3. Покрећемо инсталацију: **pip install GDAL-3.4.3-cp310-cp310-win_amd64.whl**, где је whl преузета датотека из тачке 1.
4. Затим, **pip install Fiona-1.8.21-cp310-cp310-win_amd64.whl,** где је whl преузета датотека из тачке 1
5. pip install geopandas

Након завршене исталације можемо да кренемо у ГИС програмирање коришћењем Пајтона. Већ смо се срели са **pandas** пакетом, овде се користи **geopandas**, пакет који нам омогућава увоз и рад са просторним подацима. Свакако је на почетку пројекта важно да дефинишемо директоријум у који смо сместили просторне податке. Просторни подаци који се чувају у векторском **shape(shp)** формату лако се учитавају у Пајтон код, коришћењем **geopandas**.
За почетак, увезимо потребне модуле:

```
# импортовање непходних модула
import geopandas as gpd

# Постављање путање релативно у односу на радни
 # директоријум

fp = r"\Data\ime_fajla.shp"
```

```
# или задавање апсолутне путање, препоручено
fp=r"C:\Users\Aleksndar\geopython\Data\ime_fajla.shp"
#Читање коришћењем  gpd.read_file() функције
data = gpd.read_file(fp)
#Провера који тип података је променљива data
type(data)

#Odgovor:  geopandas.geodataframe.GeoDataFrame
```

Из одговора се види да је променљива GeoDataFrame, што проширује функционалност ***pandas.DataFrame,*** на начин да је могуће користити и управљати просторним подацима унутар ***pandas,*** па отуда и назив **geopandas**.
GeoDataFrame има неке посебне функције и карактеристике које се користе у ГИС-у.
Преглед података и штампање 5 редова извршава се коришћењем функције, **head()**, која подразумевано штампа 5 редова.
За приказ података на мапи, може да се користи функција .plot() која креира једноставну мапу од података:

```
data.plot()
```

Креирање нове shape датотеке је нешто што се врло често користи. Узмимо на пример првих 50 редова улазних података и упишимо их нови **shapefile** тако што ћемо прво да изаберемо податке селекцијом одговарајућих индекса, а затим селектоване податке уписујемо у **shapefile** помоћу функције **gpd.to_file().**

```
# Kreiranje novog shape fajla
out_file_path = r"Data\naziv_novog_shape.shp"

# Selekcija prvih 50 redova,od pozicije 0 do (isključujući) 50

selection = data[0:50]

# Upis selekcije u novi Shapefile (podrazumevani format izlazne
#datoteke je Shapefile)

selection.to_file(out_file_path)
```

Напомена: У QGIS-у или неком другом ГИС алату са којим радите отворите добијени shape и проверите истравност.

Geopandas користи предности **Shapely** геометријских објеката тако што се геометрије обично чувају у колони која се зове **geometry** (ili **geom**). Ово је подразумевано име колоне за складиштење геометријксих информација у **geopandas**.
Ако желимо да прикажемо првих 5 редова колоне **geometry,** користимо наредбу:
data['geometry'].head()
Наравно могуће је користити само одређене колоне навођењем назива колоне у угластим заградама [ ].
Сви просторни подаци чувају се као **Shapely** објекти, могуће је користити све функционалности **Shapely** модула, на пример приказивање површине привих 5 полигона:
selection = data[0:5]
Могућа је итерација изабраних редова коришћењем посебне **.iterrows()** функције и приказивање области за сваки полигон:
```
for index, row in selection.iterrows():
    poly_area = row['geometry'].area
        print("Polygon area at index {0} is: {1:.3f}".format(index, poly_area))
```

Ако урадимо неку обраду над просторним подацима и желимо да сачувамо резултат обраде, можемо да креирамо нову колону у којој израчунавамо и чувамо области појединачних полигона.
Па на пример, за израчунавање површине полигона користимо **.area** атрибут.
data['area'] = data.area
и можемо да видимо прва два реда колоне **area:**
data['area'].head(2)

Рад са површина полигона захтева проналажење минималне, максималне вредности или просечне површине тих области:
```
max_area = data['area'].max()
mean_area = data['area'].mean()
print("Max area: {:.2f}\nMean area: {:.2f}".format(round(max_area, 2), round(mean_area, 2))
```

Једна од предности Shapely геометријских објеката је могућност креирања потпуно нове Shapely датотеке прослеђивањем геометријских објеката у GeoDataFrame.

Креирајмо празан GeoDataFrame

```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point, Polygon
import fiona

# Креирајмо празан geopandas GeoDataFrame

newdata = gpd.GeoDataFrame()

#провера шта је креирано
newdata ili print(newdata)
```

Одговор:

```
Empty GeoDataFrame
Columns: []
Index: []
```

Видимо да је GeoDataFrame празан јер смо га само креирали без података.

```
# Креирајмо нову колону под називом geometry
newdata['geometry'] = None

#провера шта је креирано
newdata ili print(newdata)
```

Одговор:

```
Empty GeoDataFrame
Columns: [geometry]
Index: []
```

Сада имамо колону, али још увек немамо податке.
Креирајмо полигон, квадрат у простору који можемо да убацимо у GeoDataFrame.

```
# Координате квадрата у простору у степенима децималним
coordinates = [(24.950899, 60.169158), (24.953492, 60.169158),
                        (24.953510, 60.170104), (24.950958,
                                60.169990)]

# Креирање Shapely из полигона из креираних координата, торки
poly = Polygon(coordinates)

# приказ шта је добијено
 poly ili print(poly)
 <shapely.geometry.polygon.Polygon at 0x1500f65c6a0>

# Убацивање полигона у  'geometry' -колона на индексу 0

newdata.loc[0, 'geometry'] = poly

# приказ шта је добијено
newdata ili print(newdata)

geometry
0  POLYGON ((24.950899 60.169158, 24.953492 60.16...
```

Сада имамо креиран полигон који можемо да извеземо у shape датотеку.
Додајмо још једну колону под називом **Location** са текстом који описује локацију:

```
# додавање нове колоне
newdata.loc[0, 'Location'] = 'Opis lokacije, npr. Trg republike'

# Провера
newdata
                                geometry              Location
0    POLYGON ((24.950899 60.169158, 24.953492 60.16... Trg republike
```

Пре извоза података потребно је подесити референтни координатни систем, пројекцију за GeoDataFrame, тако што користимо **.crs** за приказ координатог система:
print(newdata.crs)

None

Одговор је (None) јер податке креирамо од нуле.

Пајтон модул fiona има фукцију под називом from_epsg() за прослеђивање координатног система за GeoDataFrame.

Увезимо 'from_epsg' из fiona модула

from fiona.crs import from_epsg

# Постављање GeoDataFrame' координатног систама на #WGS84, на пример
newdata.crs = from_epsg(4326)

Провера
newdata.crs
Out[28]: {'init': 'epsg:4326', 'no_defs': True}

Коначно можемо да извеземо податке помоћу функције **GeoDataFrames.to_file()**, дефинисањем путање за shape датотеку.

out_file = r"Data\Naziv_fajla.shp"

# Упис података у Shapefile
newdata.to_file(out_file)

На овај начин креирали смо Shape датотеку само помоћу Пајтона. Сличан приступ могуће је применити за читање координата из неке текстуалне датотеке и креирње Shape датотеке.
Напомена: Проверити табелу атрибута креиране Shape

датотеке у неком ГИС софтеру.

Координатни референтни системи(CRS) су веома важни јер су геометријски облици у GeoDataFrame једноставно скуп координата.
CRS даје Пајтону информацију како се те координате односе на места на Земљи. Картографкса пројекција или пројектовани координатни систем је системска трансформација географских ширина и дужина у равну површину где су јединице врло често представљене као метри уместо децималних степена.
Важно је да слојеви у ГИС пројекту имају исту пројекцију јер то омогућава анализу просторних односа између слојева, као што је извршавање просторног упита на пример, положај тачке у полигону.
Координатни референтни системи(CRS) подразумевају два уобичајена типа:

1. Географкси координатни систем
2. Пројектовани координатни систем

Географкси координатни систем користи елипсоидну површину за дефинисање локација на Земљи и састоји се из три дела, датум, главни меридијан и угаоне јединице мере.
И географска ширина и дужина обично се представљају на два начина:

- Степени, минуте, секунде(Degrees, Minutes, Seconds, DMS),на пример 58° 23′ 12′ ′N, 26° 43′ 21′ ′E
- Децимални степени(Decimal Degrees , DD), које рачунари користе и чувају се као тип података са покретним зарезом, на пример: 58.38667 i 26.7225

Пројектовани координатни системи дефинишу равну, Декартову површину и за разлику од географског координатног система, има константне дужине, углове и површине у две димензије.
Састоји се од географског координатног система, метода пројекције, параметара пројекције (стандардне тачке и линије, географска ширина и дужина, лажни исток, лажни север итд.) и линеарне јединице(метри, километри, миље...)
Срећом, дефинисање и промена координатног система у

Пајтону, коришћењем **geopandas** је лако.
Пројектоваћемо датотеку из VGS84 (лат, лон координате) у Ламбертову пројекцију, што је пројекција за Европу коју препоручује Европска комисија. Користићемо Shape датотеку која представља Европу. Датотеку **Europe_borders.zip** која садржи Shape датотеку са следећим датотекама:
Europe_borders.cpg,
Europe_borders.prj,
Europe_borders.sbx,
Europe_borders.shx,
Europe_borders.dbf,
Europe_borders.sbn,
Europe_borders.shp,
можемо преузети са следећег линка:
https://kodu.ut.ee/~kmoch/geopython2018/_static/data/L2/Europe_borders.zip
GeoDataFrame прочитан из Shape филе-а требало би да садржи информације о координатном систему.

Читање података из датотеке **Europe_borders.shp**:

```
import geopandas as gpd

# Путања до Europe borders Shapefile
fp = r"Data\Europe_borders.shp"

# Читање података
data = gpd.read_file(fp)
```

Тренутни координатни систем можемо видети из **.crs** атрибута:

```
data.crs
```

Одговор: Out: {'init': 'epsg:4326'}

Подаци су нешто што се зове epsg:4326. EPSG број (“European Petroleum Survey Group”), је код који говори о координатном систему скупа података.
EPSG скуп података геодетских параметара је збирка дефиниција референтних координатних система и трансформација координата које могу бити глобалне,

регионалне, националне или локалне.

EPSG број 4326 припада координатном систему VGS84 (тј. координате су у децималним степенима (лат, лон)).

Провера вредности у колони геометрије:

```
data['geometry'].head()
```

Хајде да конвертујемо у Lambert Azimuthal Equal Area пројекцију EPSG: 3035. Промену CRS-а је заиста лако урадити у Geopandas помоћу функције **.to_crs()**. Као улаз у функцију, морамо да дефинишемо колону која садржи геометрије, или геометрију у овом случају, и epgs вредност CRS-а коју желимо.

```
# Копија слоја
data_proj = data.copy()
# Репројекција геометрије
data_proj = data_proj.to_crs(epsg=3035)
```

Провера како координате сада изгледају:

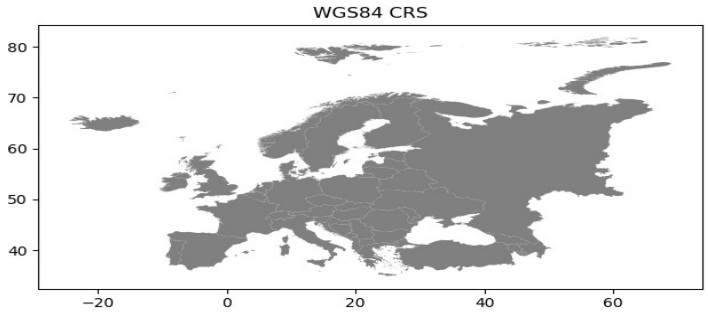
```
data_proj['geometry'].head()
```

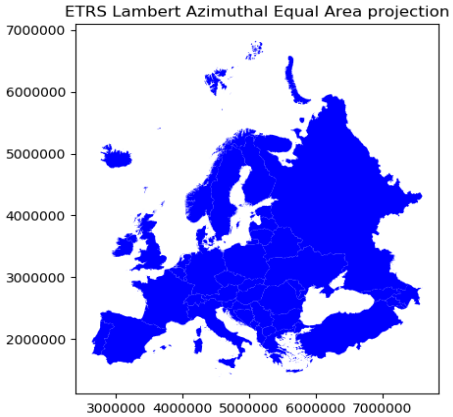
Визуелни приказ је најбољи начин да се разуме шта се заиста дешава. Упоредимо скупове података правећи карте од њих.

```
import matplotlib.pyplot as plt

# Плот WGS84
data.plot(facecolor='gray');
# Додајмо наслов
plt.title("WGS84 CRS");
# Уклонимо празан бели простор
plt.tight_layout()
# Нацртајмо ону са ETRS-LAEA пројекцијом
data_proj.plot(facecolor='blue');
# Додајмо наслов
plt.title("ETRS Lambert Azimuthal Equal Area projection");
# Уклонимо празан бели простор
```

plt.tight_layout()

Одговор:





Заиста, изгледају сасвим другачије, а редизајнирано изгледа много боље у Европи јер су области реалније и нису тако растегнуте као у VGS84.
Пројектовани слој се чува у Shape датотеци тако да се може

користити касније.

```
# Путања излазне датотеке
out_fp = r"Data\Europe_borders_epsg3035.shp"
# Сачувајмо на диск
data_proj.to_file(out_fp)
```

Пример креирање геометрије, тачка

```
# Увезимо потребне геометријске објекте из модула # shapely
#Тачка
from shapely.geometry import Point, LineString, Polygon
# Креирајмо објекат тачка, Point са координатама
point1 = Point(2.2, 4.2)
point2 = Point(7.2, -25.1)
point3 = Point(9.26, -2.456)
point3D = Point(9.26, -2.456, 0.57)
# Проверимо тип објекта point
point_type = type(point1)
#Хајде да видимо како изгледају променљиве
print(point1)
print(point3D)
print(type(point1))
```

одговор:

```
POINT (2.2 4.2)
POINT Z (9.26 -2.456 0.57)
<class 'shapely.geometry.point.Point'>
```

```
#Атрибути и функције тачке
#Издвајање координата Тачке може се обавити на #неколико различитих начина
# Прочитајмо координате
point_coords = point1.coords
print(point_coords)
# Проверимо тип
type(point_coords)
#Да видимо како можемо да добијемо стварне координате:
# Читање координата xy
```

```
xy = point_coords.xy
# Само  x координата објекта Point1
x = point1.x
# y координата
y = point1.y
print(xy)
print(x)
print(y)
```

одговор: 2.2
4.2

```
# Рачунање растојања између point1 и point2
point_dist = point1.distance(point2)
print("Distance between the points is {0:.2f} decimal degrees".format(point_dist))
```

одговор: Distance between the points is 29.72 decimal degrees

Пример креирање геометрије, линија

```
# Увозимо неопходне објекте и креирајмо објекте тачка као у
# претходном примеру

from shapely.geometry import Point, LineString, Polygon
# Креирамо Point геометријски објекат са координатама
point1 = Point(2.2, 4.2)
point2 = Point(7.2, -25.1)
point3 = Point(9.26, -2.456)
point3D = Point(9.26, -2.456, 0.57)
# Проверимо тип објекта
point_type = type(point1)
#Хајде да видимо како изгледају променљиве
print(point1)
print(point3D)
print(type(point1))
```

```
point_coords = point1.coords
print(point_coords)
type(point_coords)
print(type(point_coords))
xy = point_coords.xy
x = point1.x
y = point1.y
print(xy)
print(x)
print(y)
point_dist = point1.distance(point2)
print("Distance between the points is {0:.2f} decimal degrees".format(point_dist))
# Креирајмо линију од тачака
line = LineString([point1, point2, point3])
 # Такође је могуће користити координатне скупове са истим
# исходом
line2 = LineString([(2.2, 4.2), (7.2, -25.1), (9.26, -2.456)])
#Погледајмо како изгледа креирана линија
print(line)
print(line2)
type(line)

#Читамо координате линије
lxy = line.xy
print(lxy)
#Такође можемо да читамо појединачне координате
line_x = lxy[0]
line_y = line.xy[1]
print(line_x)
print(line_y)

#Дужина линије
l_length = line.length
# Тежиште линије
l_centroid = line.centroid
# Тип тежишта
centroid_type = type(l_centroid)
print("Length of our line: {0:.2f}".format(l_length))
```

```
print("Centroid of our line: ", l_centroid)
print("Type of the centroid:", centroid_type)
```
одговор:
```
Length of our line: 52.46
Centroid of our line: POINT (6.229961354035622 -11.892411157572392)
Type of the centroid: <class 'shapely.geometry.point.Point'>
```

Пример креирање геометрије, полигон

Креирање полигона из координата тачака

```
from shapely.geometry import Point, LineString, Polygon
# Креирање полигона из координата тачака
poly = Polygon([(2.2, 4.2), (7.2, -25.1), (9.26, -2.456)])
# Креирајмо објекте тачка са координатама као до сада
point1 = Point(2.2, 4.2)
point2 = Point(7.2, -25.1)
point3 = Point(9.26, -2.456)
point3D = Point(9.26, -2.456, 0.57)
point_type = type(point1)
# Креирајмо полигон коришћењем координата тачака
poly2 = Polygon([[p.x, p.y] for p in [point1, point2, point3]])
# Хајде да видимо како изгледа наш Полигон
print(poly)
print(poly2)
print("Geometry type as text:", poly_type)
print("Geometry how Python shows it:", poly_type2)
# Хајде да направимо полигон са рупом унутра
# Прво дефинишемо нашу спољашњост
world_exterior = [(-180, 90), (-180, -90), (180, -90), (180, 90)]
# Хајде да направимо једну велику рупу где остављамо десет
 # децималних степени на границама
hole = [[(-170, 80), (-170, -80), (170, -80), (170, 80)]]
# Без рупе
# Сада можемо да конструишемо наш Полигон са рупом унутра
```

```
world_has_a_hole = Polygon(shell=world_exterior, holes=hole)
print(world)
print(world_has_a_hole)
print(type(world_has_a_hole))
# Тежиште полигона
world_centroid = world.centroid
# Површина полигона
world_area = world.area
# Границе полигона
world_bbox = world.bounds
# Спољашњост полигона
world_ext = world.exterior
# Дужину екстеријера
world_ext_length = world_ext.length
print("Poly centroid: ", world_centroid)
print("Poly Area: ", world_area)
print("Poly Bounding Box: ", world_bbox)
print("Poly Exterior: ", world_ext)
print("Poly Exterior Length: ", world_ext_length)
```

Пример густина распрострањености популације дивљачи на територији Републике Србије

Циљ овог пројекта је да се прикаже мапа распрострањености одређене популација дивљачи у општинама Републике Србије. Алгоритам ради тако да се учитава Shape датотеку, бришу непотребне колоне, израчунавају површине општина на територији Републике Србије и смештају у посебну колону, затим у новој колони на основу података из колоне броја јединки и површина општина рачуна густину распрострањености у свакој општини и тај податак смешта у нову колону. Последња колона представља тип густине распрострањености на основу густине распрстрањености сваке општине.

На почетку је потребно да се увезу библиотеке:

```
from shapely.geometry import Point
import pandas as pd
import geopandas as gpd
from fiona.crs import from_epsg
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Затим учитавамо Shape датотеку општине Републике Срије, наводећи путању.

```
file = "Opstine SRB projekat.shp"
opstine= gpd.read_file(file)
print(opstine)
type(opstine)
```

Радимо пројекат за територију Републике Србије, па проверавамо координатни систем.

```
opstine.crs
print(opstine.crs)
```
Из табеле атрибута бришемо непотребну колону Opstine_lat:
```
brisanje = gpd.GeoDataFrame(opstine)
del opstine['Optine_lat']
print(opstine)
```
У табелу додајемо колону *Povrsina* изражену у квадратим километрима и колону *Gustina rasprostranjenosti* као број јединки по квадратном метру:
```
opstine['Povrsina'] = opstine.area/1000000
print(opstine)
opstine['Gustina naseljenosti']=opstine['br_jedinki']/opstine['Povrsina']
print(opstine)
```
Додајемо празну колону Tip gustine
```
opstine['Tip gustine'] = None
print(opstine)
```

Дефинишемо 5 типова густине распрострањености јединки према броју јединки по квадратном километру:

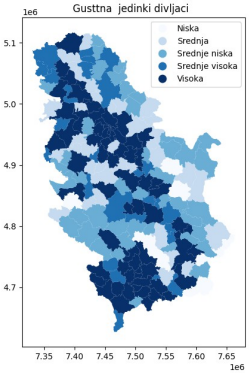
1.0-25 ст/км² - Ниска
2.25-50 ст/км² - Средње ниска
3.50-75 ст/км² - Средња
4.75-100 ст/км² - Средње висока
5.100-200 ст/км² – Висока

```
df = gpd.GeoDataFrame(opstine)
df.loc[df['Gustina rasprosranjenosti'] < 25, 'Tip gustine'] = 'Niska'
df.loc[df['Gustina rasprosranjenosti'] > 25, 'Tip gustine'] = 'Srednje niska'
df.loc[df['Gustina rasprosranjenosti'] > 50, 'Tip gustine'] = 'Srednja'
df.loc[df['Gustina rasprosranjenosti'] > 75, 'Tip gustine'] = 'Srednje visoka'
df.loc[df['Gustina rasprosranjenosti'] > 100, 'Tip gustine'] = 'Visoka'
```
На крају додајемо празну колону Tip_gustine_broj:

```
opstine['Tip gustine_broj'] = None
print(opstine)
```
Сада одређујемо тип густине распрострањености јединки дивљачи на основу дефинисаног типа:
```
df = gpd.GeoDataFrame(opstine)
df.loc[df['Gustina rasprosranjenosti'] < 25, 'Tip gustine_broj'] = 0
df.loc[df['Gustina rasprosranjenosti'] > 25, 'Tip gustine_broj'] = 1
df.loc[df['Gustina rasprosranjenosti'] > 50, 'Tip gustine_broj'] = 2
df.loc[df['Gustina rasprosranjenosti'] > 75, 'Tip gustine_broj'] = 3
df.loc[df['Gustina rasprosranjenosti'] > 100,'Tip gustine_broj'] = 4
############################################################
print(df)
print(opstine)
```

Графички приказ просторне расподеле густине распрострањености јединки дивљачи на територији Републике Србије:
```
opstine.plot(column='Tip gustine', categorical=True, legend=True, figsize=(20,20),cmap="Blues")
plt.title('Gustna  jedinki divljaci  ')
plt.show()
#snimanje dobijenog fajla na disk
izlaz= "Mapa gustine .shp"
opstine.to_file(izlaz)
```

Одговор:



Описани пример је модификација пројектног задатака студента Бојана Благојевића на предмету ГИС програмирање на коме се обрађује ова материја, на Мастер ГИС, на Географском факултету Универзитета у Београду.

Пример преклапање геометријског облика, тачке и слоја на карти

У овом примеру обједињећемо дефинисање геометријског облика, тачке, које рецимо приказују неке мерне тачке, читање shape датотеке, одређивање координатног система и приказа мапе са дефинисам тачкама.
Да би пример био што је могуће ближи реалном пројекту, претпоставимо да су тачке неке мерне станице на територији града Београда.
На почетку, наравно увозимо потребне библиотеке:

```
from shapely.geometry import Point
import geopandas as gpd
from fiona.crs import from_epsg
import matplotlib.pyplot as plt
```

Затим дефинишемо 11 тачака као објекте са припадајућим координатама:

```
t_1 = Point(7457264, 4956249)
t_2 = Point(7454151, 4954792)
t_3 = Point(7454149, 4959597)
t_4 = Point(7463725, 4963451)
t_5 = Point(7463540, 4960590)
t_6 = Point(7458462, 4964893)
t_7 = Point(7453836, 4961861)
t_8= Point(7459817, 4956731)
t_9= Point(7461916, 4960250)
t_10 = Point(7460551, 4958996)
t_11 = Point(7452338, 4966795)
#Провера типа објекта
tip_tacke = type(t_1)
print(t_1)
Креирајмо листу тачака и креирајмо GeoDataFrame()
lista = [(t_1, "Tacka_1"), (t_2, "Tacka_2"),( (t_3, "Tacka_3")), ((t_4, "Tacka_4")),((t_5,
```

"Tacka_5")),((t_6, "Tacka_6")),((t_7, "Tacka_7")), ((t_8, "Tacka_8")),((t_9, "Tacka_9")), ((t_10, "Tacka_10")),((t_11, "Tacka_11"))]

```
prostor= gpd.GeoDataFrame()
prostor['geometry'] = None
for ID, (tacka,naziv) in enumerate(lista):
    prostor.loc[ID, 'geometry'] = tacka
    prostor.loc[ID, 'Naziv tacke'] = naziv
```

Затим учитавамо и подешавамо координатни систем и прикажимо тачке:

```
prostor.crs = from_epsg(6316)
prostor.plot(facecolor='red');
plt.title("Merne tacke")
plt.show()
```

Учитајмо shape датотеку Bg_opstine.shp, проверимо координатни систем:

```
fp = "Shp\Bg_opstine.shp"
#Citanje fajla
prostor_poly = gpd.read_file(fp)
prostor_poly
prostor_poly.crs
print(prostor_poly.crs)
prostor_poly['geometry'].head()
print(prostor_poly['geometry'].head())
```

Следећим делом кода показаћемо промену назива атрибута:

```
#Preimenovanje atributa
prostor_poly = prostor_poly.rename(columns={'Naziv': 'Naziv opstine'})
prostor_poly.columns
print(prostor_poly)
```

Затим додајемо нови атрибут за сваки ред:

```
prostor_poly ['area'] = None
for povrsina, row in prostor_poly .iterrows():
    prostor_poly .loc[povrsina, 'area'] = row['geometry'].area
print(prostor_poly ['area'].head())
print("Povrsine su jednake:", prostor_poly )
prostor_poly .crs = from_epsg(6316)
prostor_poly .crs
```

```
print(prostor_poly .crs)
```

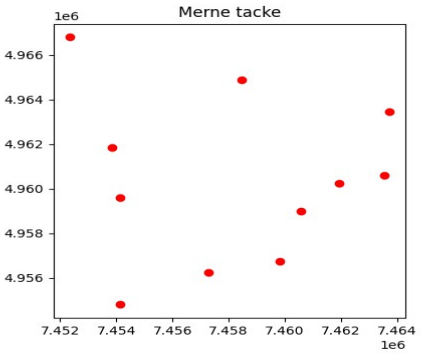
Приказујемо карту општина Београда:

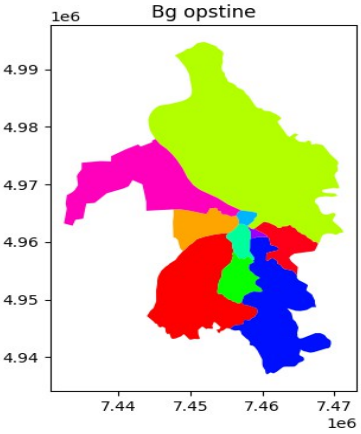
```
prostor_poly.plot(column='Naziv opstine', cmap='hsv');
#Dodavanje naslova
plt.title("Bg opstine");
plt.tight_layout()
plt.show()
```

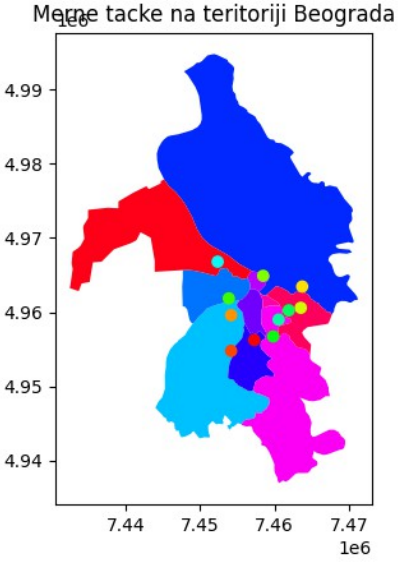
Спајамо тачке, мерне станица са општинама и приказјемо мапу:

```
print(prostor.crs)
print(prostor_poly.crs)
prostor.to_crs(prostor_poly.crs, inplace=True)
spojeno = prostor.geometry.append(prostor_poly.geometry)
print(spojeno.crs)
print(spojeno)
#Prikaz mernih stanica na teritoriji  Beograda
spojeno.plot(cmap="hsv")
plt.title("Merne tacke na teritoriji Beograda")
plt.show()
```

Као резултат добијамо мапу просторних података.




Bg opstine
1e6
4.99
4.98
4.97
4.96
4.95
4.94
7.44
7.45
7.46
7.47
1e6


Merne tacke na teritoriji Beograda
1e6
4.99
4.98
4.97
4.96
4.95
4.94
7.44
7.45
7.46
7.47
1e6

Описани пример је модификација пројектног задатака студенткиње Иване Петровић на предмету ГИС програмирање на коме се обрађује ова материја, на Мастер ГИС, на Географском факултету Универзитета у Београду.

Сви примери из овог поглавља, као и претходних налазе се у додатку књиге, тестрирани су PyChram развојном окружењу и могу се користити као шаблон за почетак неког новог пројекта.

Закључак

Уместо закључка,

• • •

Литература

1. https://www.python.org/

2. Petrovic Milos N|,Vukicevic Arso M|,Djapan Marko J|0000-0002-8016-8422,**Peulic Aleksandar S**|0000-0003-3043-6879,Jovicic Milos N|,Mijailovic Nikola V|,Milovanovic Petar D|0000-0003-1461-8437,Grajic Mirko M|,Savkovic Marija|,Caiazzo Carlo|,Isailovic Velibor M|,Macuzic Ivan D|,Jovanovic Kosta M|0000-0002-9029-4465 (2022) Experimental Analysis of Handcart Pushing and Pulling Safety in an Industrial Environment by Using IoT Force and EMG Sensors: Relationship with Operators' Psychological Status and Pain Syndromes, SENSORS, vol. 22, br. 19, str. - (Article)

3. Isailovic Velibor M|,**Peulic Aleksandar S**|0000-0003-3043-6879,Djapan Marko J|0000-0002-8016-8422,Savkovic Marija|,Vukicevic Arso M| (2022) The compliance of head-mounted industrial PPE by using deep learning object detectors, SCIENTIFIC REPORTS, vol. 12, br. 1, str. - (Article)

4. Sustersic Tijana|0000-0003-1417-0521,**Peulic Aleksandar S**| (2022) FPGA Implementation of Expert System for Medical Diagnosis of Disc Hernia Diagnosis Based on Bayes Theorem, JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS, vol. , br. , str. - (Article; Early Access)

5. **Peulic Aleksandar S**|0000-0003-3043-6879,Peulic Miodrag|0000-0002-5984-2902,Jovanovic Zeljko|,Jokovic Milos|,Stojkovic Sanja|0000-0003-2292-5082,Vagic Nemanja| (2021) Locating and categorizing causes of discomfort during transport of patients to medical facilities, TRANSACTIONS IN GIS, vol. 25, br. 6, str. 2963-2981 (Article)

6. Vukicevic Arso M|,Macuzic Ivan D|,Mijailovic Nikola V|,**Peulic Aleksandar S**|0000-0003-3043-6879,Radovic Milos D| (2021) Assessment of the handcart pushing and pulling safety by using deep learning 3D pose estimation and IoT force sensors, EXPERT SYSTEMS WITH APPLICATIONS, vol. 183, br. , str. - (Article)

7. Sustersic Tijana|0000-0003-1417-0521,Peulic Miodrag|0000-0002-5984-2902,**Peulic Aleksandar S**|0000-0003-3043-6879 (2021) FPGA Implementation of Fuzzy Medical Decision Support System for Disc Hernia Diagnosis, COMPUTER SCIENCE AND INFORMATION SYSTEMS, vol. 18, br. 3, str. 619-640 (Article)

8. Peulic Miodrag|,Jokovic Milos|,Sustersic Tijana|0000-0003-1417-0521,**Peulic Aleksandar S**|0000-0003-3043-6879 (2020) A Noninvasive Assistant System in Diagnosis of Lumbar Disc Herniation, COMPUTATIONAL AND MATHEMATICAL METHODS IN MEDICINE, vol. 2020, br. , str. - (Article)

9. Sustersic Tijana|0000-0003-1417-0521,Rankovic Vesna M|0000-0002-5445-9971,Peulic Miodrag|0000-0002-5984-2902,**Peulic Aleksandar S**|0000-0003-3043-6879 (2020) An Early Disc Herniation Identification System for Advancement in the Standard Medical Screening Procedure Based on Bayes Theorem, IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, vol. 24, br. 1, str. 151-159 (Article)

10. Jovanovic Zeljko|,Blagojevic Marija|0000-0003-4186-0448,Jankovic Dragan S|,**Peulic Aleksandar S**|0000-0003-3043-6879 (2019) Patient comfort level prediction during transport using artificial neural network, TURKISH JOURNAL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, vol. 27, br. 4, str. 2817-2832 (Article)

11. Jovanovic Zeljko|,Milosevic Marina|,Jankovic Dragan S|,**Peulic Aleksandar S|**0000-0003-3043-6879 (2019) Comfort level classification during patients transport, TECHNOLOGY AND HEALTH CARE, vol. 27, br. 1, str. 61-77 (Article)

12. **Peulic Aleksandar S**|0000-0003-3043-6879,Sustersic Tijana|0000-0003-1417-0521,Peulic Miodrag|0000-0002-5984-2902 (2019) Non-invasive improved technique for lumbar discus hernia classification based on fuzzy logic, BIOMEDICAL ENGINEERING-BIOMEDIZINISCHE TECHNIK, vol. 64, br. 4, str. 421-428 (Article)

13. Sustersic Tijana|0000-0003-1417-0521,**Peulic Aleksandar S**|0000-0003-3043-6879 (2019) Implementation of Face Recognition Algorithm on Field Programmable Gate Array (FPGA), JOURNAL OF CIRCUITS SYSTEMS AND COMPUTERS, vol. 28, br. 8, str. - (Article)

14. Djordjevic Natasa Z|0000-0002-1100-5689,Paunovic Milica G|0000-0003-1557-7497,**Peulic Aleksandar S**|0000-0003-3043-6879 (2017) Anxiety-like behavioural effects of extremely low-frequency electromagnetic field in rats, ENVIRONMENTAL SCIENCE AND POLLUTION RESEARCH, vol. 24, br. 27, str. 21693-21699 (Article)

15. Milankovic Ivan L|,Mijailovic Nikola V|,Filipovic Nenad D|,**Peulic Aleksandar S**|0000-0003-3043-6879 (2017) Acceleration of Image Segmentation Algorithm for (Breast) Mammogram Images Using High-Performance Reconfigurable Dataflow Computers, COMPUTATIONAL AND MATHEMATICAL METHODS IN MEDICINE, vol. , br. , str. - (Article)

16. Radovic Milos D|,Milosevic Marina|,Ninkovic Srdjan M|0000-0002-0396-0176,Filipovic Nenad D|,**Peulic Aleksandar S**|0000-0003-3043-6879 (2015) Parameter optimization of a computer-aided diagnosis system for detection of masses on digitized mammograms, TECHNOLOGY AND HEALTH CARE, vol. 23, br. 6, str. 757-774 (Article)

17. Mijailovic Nikola V|,Vulovic Radun|,Milankovic Ivan L|,Radakovic Radivoje|,Filipovic Nenad D|,**Peulic Aleksandar S**|0000-0003-3043-6879

(2015) Assessment of Knee Cartilage Stress Distribution and Deformation Using Motion Capture System and Wearable Sensors for Force Ratio Detection, COMPUTATIONAL AND MATHEMATICAL METHODS IN MEDICINE, vol. , br. , str. - (Article)

18. Milovanovic Alenka M|0000-0002-7651-2489,Koprivica Branko M|0000-0001-5014-6866,**Peulic Aleksandar S**|0000-0003-3043-6879,Milankovic Ivan L| (2015) Analysis of Square Coaxial Line Family, APPLIED COMPUTATIONAL ELECTROMAGNETICS SOCIETY JOURNAL, vol. 30, br. 1, str. 99-108 (Article)

19. Milosevic Marina|,Jankovic Dragan S|0000-0003-1198-0174,**Peulic Aleksandar S**|0000-0003-3043-6879 (2015) Comparative analysis of breast cancer detection in mammograms and thermograms, BIOMEDICAL ENGINEERING-BIOMEDIZINISCHE TECHNIK, vol. 60, br. 1, str. 49-56 (Article)

20. Milosevic Marina|,Jankovic Dragan S|0000-0003-1198-0174,**Peulic Aleksandar S**|0000-0003-3043-6879 (2014) Thermography Based Breast Cancer Detection Using Texture Features and Minimum Variance Quantization, EXCLI JOURNAL, vol. 13, br. , str. 1204-1215 (Article)

21. Milosevic Marina|,Jankovic Dragan S|0000-0003-1198-0174,**Peulic Aleksandar S**|0000-0003-3043-6879 (2014) Segmentation for the enhancement of microcalcifications in digital mammograms, TECHNOLOGY AND HEALTH CARE, vol. 22, br. 5, str. 701-715 (Article)

22. Jeremic Svetlana R|0000-0001-5571-6880,Filipovic Nenad D|,**Peulic Aleksandar S**|0000-0003-3043-6879,Markovic Zoran S|0000-0001-5964-049X (2014) Thermodynamical aspect of radical scavenging activity of alizarin and alizarin red S. Theoretical comparative study, COMPUTATIONAL AND THEORETICAL CHEMISTRY, vol. 1047, br. , str. 15-21 (Article)

23. Cvetkovic Aleksandar M|0000-0001-9659-2001,Milasinovic Danko Z|0000-0002-8640-7973,**Peulic Aleksandar S**|0000-0003-3043-6879,Mijailovic Nikola V|,Filipovic Nenad D|,Zdravkovic Nebojsa D|0000-0003-0877-3684 (2014) Numerical and experimental analysis of factors leading to suture dehiscence after Billroth II gastric resection, COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE, vol. 117, br. 2, str. 71-79 (Article)

24. Filipovic Nenad D|,Djukic Tijana R|0000-0002-9913-6527,Radovic Milos D|,Cvetkovic Danijela M|0000-0001-7767-1345,Curcic Milena G|,Markovic Snezana D|0000-0002-3892-8977,**Peulic Aleksandar S**|0000-0003-3043-6879,Jeremic Branislav| (2014) Electromagnetic field investigation on different cancer cell lines, CANCER CELL INTERNATIONAL, vol. 14, br. , str. - (Article)

25. Dragicevic Snezana M|0000-0002-6244-0111,**Peulic Aleksandar S**|0000-

0003-3043-6879,Bjekic Miroslav M|,Krneta Radojka|0000-0002-2974-5132 (2013) Measurement and Simulation of Energy Use in a School Building, ACTA POLYTECHNICA HUNGARICA, vol. 10, br. 2, str. 109-120 (Article)

26. Ebersold Zoran Z|,Mitrovic Nebojsa S|0000-0002-7971-6321,Djukic Slobodan R|0000-0002-9348-6374,**Peulic Aleksandar S**|0000-0003-3043-6879,Obradovic Dragisa| (2013) Low Frequencies for Cardboard Quality Assurance, MATERIALS TESTING, vol. 55, br. 2, str. 109-113 (Article)

27. **Peulic Aleksandar S**|0000-0003-3043-6879,Milojevic Natasa|,Jovanov Emil S|0000-0001-6754-3518,Radovic Milos D|,Saveljic Igor B|,Zdravkovic Nebojsa D|0000-0003-0877-3684,Filipovic Nenad D| (2013) Modeling of Arterial Stiffness using Variations of Pulse Transit Time, COMPUTER SCIENCE AND INFORMATION SYSTEMS, vol. 10, br. 1, str. 547-565 (Article)

28. Filipovic Nenad D|,Isailovic Velibor M|,Nikolic Dalibor D|0000-0001-5491-4586,**Peulic Aleksandar S**|0000-0003-3043-6879,Mijailovic Nikola V|,Petrovic Suzana|0000-0002-5629-6221,Cukovic Sasa|,Vulovic Radun|,Matic Aleksandar|0000-0001-9843-4357,Zdravkovic Nebojsa D|0000-0003-0877-3684,Devedzic Goran B|0000-0002-6589-5883,Ristic Branko M| (2013) Biomechanical Modeling of Knee for Specific Patients with Chronic Anterior Cruciate Ligament Injury, COMPUTER SCIENCE AND INFORMATION SYSTEMS, vol. 10, br. 1, str. 525-545 (Article)

29. **Peulic Aleksandar S**|0000-0003-3043-6879,Dragicevic Snezana M|0000-0002-6244-0111,Jovanovic Zeljko|,Krneta Radojka|0000-0002-2974-5132 (2013) Flexible GPS/GPRS based System for Parameters Monitoring in the District Heating System, INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL, vol. 8, br. 1, str. 105-110 (Article)

30. Ebersold Zoran Z|,Mitrovic Nebojsa S|0000-0002-7971-6321,Djukic Slobodan R|0000-0002-9348-6374,Jordovic Branka|,**Peulic Aleksandar** S|0000-0003-3043-6879 (2012) Defectoscopy of Direct Laser Sintered Metals by Low Transmission Ultrasonic Frequencies, SCIENCE OF SINTERING, vol. 44, br. 2, str. 177-185 (Article)

31. **Peulic Aleksandar** S|0000-0003-3043-6879,Dragicevic Snezana M|0000-0002-6244-0111,Jovanovic Zeljko|,Krneta Radojka|0000-0002-2974-5132 (2012) Laboratory experience in solving real-life engineering problems - Design and implementation of GPS/GPRS System for District Heating System Parameters Monitoring, TECHNICS TECHNOLOGIES EDUCATION MANAGEMENT-TTEM, vol. 7, br. 3, str. 1311-1318 (Article)

32. Dragicevic Snezana M|0000-0002-6244-0111,**Peulic Aleksandar** S|0000-0003-3043-6879,Bjekic Miroslav M|,Krneta Radojka|0000-0002-2974-5132 (2012) Temperature Monitoring of School Building. Validation of Simulation Model by Comparison with Wireless Flexible Measurement Data, COMPTES RENDUS DE L ACADEMIE BULGARE DES SCIENCES, vol.

65, br. 8, str. 1145-1150 (Article)

33. Filipovic Nenad D|,Rosic Mirko A|0000-0002-2433-6882,Tanaskovic Irena|0000-0002-1877-3250,Milosevic Zarko|,Nikolic Dalibor D|0000-0001-5491-4586,Zdravkovic Nebojsa D|0000-0003-0877-3684,**Peulic Aleksandar** S|0000-0003-3043-6879,Kojic Milos R|,Fotiadis Dimitrios I|,Parodi Oberdan| (2012) ARTreat Project: Three-Dimensional Numerical Simulation of Plaque Formation and Development in the Arteries, IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, vol. 16, br. 2, str. 272-278 (Article)

34. Djokovic Marina|,**Peulic Aleksandar** S|0000-0003-3043-6879,Filipovic Nenad D| (2011) Automatic identification breast cancer using multiresolution algorithm, HEALTHMED, vol. 5, br. 6, str. 2051-2064 (Article)

35. Filipovic Nenad D|,**Peulic Aleksandar** S|0000-0003-3043-6879,Zdravkovic Nebojsa D|0000-0003-0877-3684,Grbovic-Markovic Vesna M|,Jurisic-Skevin Aleksandra J| (2011) Transient finite element modeling of functional electrical stimulation, GENERAL PHYSIOLOGY AND BIOPHYSICS, vol. 30, br. 1, str. 59-65 (Article)
36. Filipovic Nenad D|,Vulovic Radun|,**Peulic Aleksandar** S|0000-0003-3043-6879,Radakovic Radivoje|,Kosanic Djordje|,Ristic Branko M| (2009) Noninvasive determination of knee cartilage deformation during jumping, JOURNAL OF SPORTS SCIENCE AND MEDICINE, vol. 8, br. 4, str. 584-590 (Article)
37. Nemanja Vagić, **Aleksandar Peulić**, Sanja Stojković, OBJECT DETECTION IN ORDER TO DETERMINE LOCATIONS FOR WILDLIFE CROSSINGS, Zbornik radova - Geografski fakultet Univerziteta u Beogradu ,ISSN:1450-7552, doi:10.5937/zrgfub2270023V
38. Milanković Ivan, Mijailović Nikola, Končar Igor, Nikolić Dalibor, Filipović Nenad, **Peulić Aleksandar**, Development of the system for abdominal aortic aneurysm mechanical properties research using 'Bubble inflated' method, Serbian Journal of Electrical Engineering, Vol.10, No.3, pp. 415-423, ISSN 1451-4869, 2013

39. Nikola Mijailović, **Aleksandar Peulić**, Nenad Filipović, Emil Jovanov, Implementation of Wireless
Sensor System in Rehabilitation After Back Spine Surgery, SERBIAN JOURNAL OF ELECTRICAL
ENGINEERING, Vol.9, No.1, pp 63-70, ISSN 1451-4869, Doi 10.2298/SJEE1201063M, 2012
40. Maja Milosevic, Nikola Mijailovic, Dalibor Nikolic, Nenad Filipovic, **Aleksandar Peulic**, Mirko Rosic1and Suzana Pantovic,MANUFACTURING OF BIODEGRADABLE SCAFFOLDS TO ENGINEER ARTIFICIAL BLOOD VESSEL**,** Serbian Journal Exp Clin Res 2018,The Journal of Faculty of Medical Sciences, University of Kragujevac ; 19 (3): 215-221,DOI: https://doi.org/10.1515/sjecr-2017-0032

41. Aleksandra Vulovic, Tijana Sustersic, **Aleksandar Peulic**, Nenad Filipovic,

Vesna Rankovic,Comparison of different neural network training algorithms with application to face recognition,EAI Endorsed Transactions on Industrial Networks and Intelligent Systems Vol.4, No.12, Doi:10.4108/eai.10-1-2018.153550, 2018

42. Filipović N., Nedeljković M., **Peulić A**., Finite element modeling of a transient functional electrical stimulation**,** Journal of Serbian Society for Computational Mechanics, 2007, vol. 1, br. 1, str. 154-163, Univerzitet u Kragujevcu - Fakultet inženjerskih nauka, Kragujevac
43. I Milanković, N Mijailović, **A Peulić**, N Filipović, Application of data flow engines in biomedical images processing, IPSI BgD Trans Adv Res (TAR), Vol.14, No.1, 2018
44. Tijana Šušteršič, Aleksandra Vulović, Nenad Filipović, **Aleksandar Peulić**, FPGA implementation of face recognition algorithm, Pervasive Computing Paradigms for Mental Health, pp 93-99, Doi 10.1007/978-3-319-74935-8_13, 2016
45. Radivoje Radaković, Radun Vulović, **Aleksandar Peulić**, Dalibor Nikolić, Nenad Filipović,METHOD FOR SOFTWARE TRACKING AND ANALYSIS OF PLAYER'S MOTION DURING A FOOTBALL MATCH AND GENERAL PARAMETERS FOR FC RED STAR PLAYERS DURING THE QUALIFICATION ROUNDS FOR UEFA EUROPE LEAGUE,1st International Conference on Chemo and BioInformatics, ICCBIKG 2021, (161-164),DOI: 10.46793/ICCBI21.161R
46. Šušteršič, T., Milovanović, V., Ranković, V., Filipović, N., **Peulić, A**. (2020). Medical Image Processing Using Xilinx System Generator. In: Filipovic, N. (eds) Computational Bioengineering and Bioinformatics. ICCB 2019. Learning and Analytics in Intelligent Systems, vol 11. Springer, Cham. https://doi.org/10.1007/978-3-030-43658-2_10
47. Mijailovic, N., Radakovic, R., **Peulic, A**., Vidanovic, N., Dimitrijevic, D., Filipovic, N. (2018). Assessment of Mechanical Stiffness of Jumping Using Force Plate. In: Oliver, N., Serino, S., Matic, A., Cipresso, P., Filipovic, N., Gavrilovska, L. (eds) Pervasive Computing Paradigms for Mental Health. FABULOUS MindCare IIOT 2016 2016 2015. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 207. Springer, Cham. https://doi.org/10.1007/978-3-319-74935-8_11
48. Tijana Šušteršič, Aleksandra Vulović, Nenad Filipović, **Aleksandar Peulić**, FPGA Implementation of Face Recognition Algorithm, ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2018 N. Oliver et al. (Eds.): MindCare 2016/Fabulous 2016/IIoT 2015, LNICST 207, pp. 81–86, 2018,vol 207. Springer, Cham. https://doi.org/10.1007/978-3-319-74935-8_13
49. Medical Image Processing using Xilinx System Generator – Tijana Šušteršič, Vladimir Milovanović, Vesna Ranković, Nenad Filipović, **Aleksandar Peulić**, VIII International Conference on Computational Bioingeenering, September 4-6 2019, Belgrade, pp. 90

50. **A.Peulic**, T.Sustersic, N. Milosevic, ESTIMATION OF ANXIETY-LIKE BEHAVIOR IN RATS EXPOSED TO EXTREMELY LOW-FREQUENCY ELECTROMAGNETIC FIELD**,** Пленарно предавање, Смедерево Еколошки град, Конференција са међународним учешћем, Смедерево, 23.11.2018
51. **Aleksandar Peulic**, Emil Jovanov, Milos Radovic, Igor Saveljic, Nebojsa Zdravkovic, NenadFilipovic, Arterial Stiffness modeling using variations of Pulse Transit Time, 10thBioEng, 5-7October2011,Kos,Greece, ISBN: 987-1-4577-0552-6, IEEE, pp 1 – 4
52. Jevremović Z., Peulić A., Jordović B., Knezević T, Automatic System for Monitoring of Vibro-Comfort of Vehicle, 15th International Research/Expert Conference Trends in the Development of Machinery and Associated Technology, Czech Republic, 2011, ISBN 1840-4944, pp 961-964
53. **A.Peulic**, I.Milankovic, N. Mijailovic, Z.Jovanovic,Remotely Analyze Spine Angle in Rehabilitation After Spine Surgery using Acceleration and Gyro Sensors,IEEE 13th International Conference on Remote Engineering and Virtual Instrumentation, 24-26 February 2016, UNED, Madrid, Spain, ISBN: 978-1-4673-8245-8/16, pp 275-276
54. **Aleksandar Peulić**, Đorđe Damnjanović, Radojka Krneta, A Laboratory Setup for Magnetic Filed Distribution Monitoring, The Experiment@International Conference 2015 (exp.at'15), Ponta Delgada, São Miguel Island, Azores, Portugal, 2015, 2-4 jun, ISBN: 978-1-4673-7716-4/15, pp 331-336
55. Karl Benkič, Aleksandar Peulić, Routing tree implementation based on semimatching algorithm(s), 21st telecommunications forum TELFOR 2013 Serbia, Belgrade, November 26-28, 2013, ISBN: 978-1-4799-1420-3/13, pp.809-812
56. Tijana Šušteršič, Nikola Mijailović, Ivan Milanković, Nenad Filipović , **Aleksandar Peulić**, Segmentation and Three-Dimensional Visualization of Brain Tumor and Possibility of Mapping Such Algorithms on High Performance Reconfigurable Computers, CD zbornik, ICIST 2015 5th International Conference on Information Society and Technology, Kopaonik, Serbia, 2015, ISBN: 978-86-85525-16-2, pp 455-460
57. Tijana Šušteršič; Miodrag Peulić; Nenad Filipović; **Aleksandar Peulić**,Application of active contours method in assessment of optimal approach trajectory to brain tumor , Bioinformatics and Bioengineering (BIBE), 2015 IEEE 15th International Conference on, Belgrade, 2015, pp. 1 – 5
58. Rankovic, Vesna; Milankovic, Ivan; Peulic, Miodrag; Filipovic, Nenad; **Peulic, Aleksandar**, A fuzzy model for supporting the diagnosis of lumbar disc herniation, Bioinformatics and Bioengineering (BIBE), 2015 IEEE 15th International Conference on, Belgrade, 2015, pp. 1 – 5
59. Nikola Mijailović , Jasna Radulović , Miroslav Trajanović , Nenad Filipović , **Aleksandar Peulić**, Multimodal Imaging for PET Attenuation Correction, CD zbornik, ICIST 2015 5th International Conference on Information Society and Technology, Kopaonik, Serbia, 2015, ISBN: 978-86-85525-16-2, pp 464-467
60. Milos Radovic , Marina Djokovic , **Aleksandar Peulic** , Nenad Filipovic, Application of Data Mining Algorithms for Detection of Masses on Digitalized

Mammograms,CD zbornik, ICIST 2015 5th International Conference on Information Society and Technology, Kopaonik, Serbia, 2015, ISBN: 978-86-85525-16-2, pp.13-18

61. Milankovic, Ivan; **Peulic, Aleksandar**; Ysasi, Alexandra B.; Wagner, Willi L.; Pabst, Andreas M.; Ackermann, Maximilian; Houdek, Jan; Fohst, Sonja; Mentzer, Steven J.; Konerding, Moritz A.; Filipovic, Nenad; Tsuda, Akira, Acceleration of image filtering algorithms for 3D visualization of murine lungs using dataflow engines, Flesh zbornik, Bioinformatics and Bioengineering (BIBE), 2015 IEEE 15th International Conference on, Belgrade, 2015, pp. 1 – 5
62. Mijailovic, Nikola; Radakovic, Radivoje; **Peulic, Aleksandar**; Milankovic, Ivan; Filipovic, Nenad, Using force plate, computer simulation and image alignment in jumping analysis, Bioinformatics and Bioengineering (BIBE), 2015 IEEE 15th International Conference on, Belgrade, 2015, pp.1-5
63. Ivan Milanković, Vesna Ranković , Miodrag Peulić, Nenad Filipović , **Aleksandar Peulić**, Diagnosis of Lumbar Disc Herniation using Multilayer Perceptron Neural Network, CD zbornik, ICIST 2015 5th International Conference on Information Society and Technology, Kopaonik, Serbia, 2015, ISBN: 978-86-85525-16-2, pp.210-213
64. Vanja Luković, Radojka Krneta, **Aleksandar Peulić**, Željko Jovanović, Đorđe Damnjanović,, LEARNING BITWISE OPERATIONS IN C USING REMOTE EXPERIMENT ON FLOATING LED'S BLINKING, International Conference on Electrical, Electronic and Computing Engineering, Zlatibor, Serbia, 2016, June 13–16, ISBN: 978-86-7466-618-0, pp. AUI1.4. 1-6
65. Nikola Mijailović, Suzana Petrović, Dalibor Nikolić, **Aleksandar Peulić**, Nebojša Zdravković, Branko Ristić, Nenad Filipović, NON-INVASIVELY ASSESSMENT OF KNEE CARTILAGE STRESS DISTRIBUTION USING MOTION CAPTURE SYSTEM AND FINITE ELEMENT METHOD, Fourth Serbian (29th Yu) Congress on Theoretical and Applied Mechanics, Vrnjacka Banja, 2013, ISBN 978-86-909973-5-0, pp.809-814
66. Miloš Radović, Marina Đoković, **Aleksandar Peulić**, Nenad Filipović, APPLICATION OF DATA MINING TECHNIQUES FOR MAMMOGRAM CLASSIFICATION, Fourth Serbian (29th Yu) Congress on Theoretical and Applied Mechanics, Vrnjacka Banja, 2013, ISBN 978-86-909973-5-0, ISBN 978-86-909973-5-0, pp.769-774
67. Radivoje Radaković, **Aleksandar Peulić**, Slobodan Kovač, Nenad Filipović, ELECTROMYOGRAPHY DETECTION OF MUSCLE RESPONSE IN MUSCULUS QUADRICEPS FEMORIS OF ELITE VOLLEYBALL PLAYERS ON DIFFERENT TRAINING STIMULI, Fourth Serbian (29th Yu) Congress on Theoretical and Applied Mechanics, Vrnjacka Banja, 2013, ISBN 978-86-909973-5-0, pp.793-796
68. **Aleksandar Peulic**,Zeljko Jovanovic, SMART SYSTEM FOR COMFORT PREDICTION AND ACTIVE SUSPENSIONS CONTROL, International Congress Motor Vehicles & Motors 2016, *Kragujevac, October 6th-8th, 2016,* pp.313-320
69. Nikola Mijailovic, Radivoje Radakovic, **Aleksandar Peulic**, Neda Vidanovic, Djordje Dimitrijevic and Nenad Filipovic, Assessment of mechanical stiffness of jumping using force plate, 2nd EAI International Conference on

Future Access Enablers of Ubiquitous and Intelligent Infrastructures OCTOBER 24–25, 2016 , BELGRADE, SERBIA,http://fabulous-conf.org/2016/show/accepted-papers

70. Tijana Šušteršič, Aleksandra Vulović, Nenad Filipović, and **Aleksandar Peulić**, FPGA Implementation of Face Recognition Algorithm, 2nd EAI International Conference on Future Access Enablers of Ubiquitous and Intelligent InfrastructuresOCTOBER 24–25, **2**016 **,** BELGRADE, SERBIA**,** http://fabulous-conf.org/2016/show/accepted-papers
71. Milos Radovic, Marina Djokovic, **Aleksandar Peulic**, Application of Data Mining Algorithms for Mammogram Classification, *Proceedings of the 13th International IEEE Conference on Bioinformatics and Bioengineering (BIBE)*, Chania, November 10-13, 2013, pp. 1-4, ISBN:978-1-4799-3162-0.
72. Ivan Milankovic, Nikola Mijailovic, Jasna Radulovic, **Aleksandar Peulic**, Nenad Filipovic, Development a system for analyzing the electromagentic radiation caused by the ct scanner, 8th International Quality Conference, May 23th, 2014, Kragujevac, Serbia, pp.1-6
73. Nikola Mijailovic, Jasna Radulovic, **Aleksandar Peulic**, Miroslav Trajanovic, Nikola Radulovic, Ct scanner quality according to exposure dose during scanning procedure, 8th International Quality Conference, May 23th, 2014, Kragujevac, Serbia, pp.201-206