Zmogljiva integracija brez REST strežnika ali mikrostoritev na primeru Python-a in .NET

Matjaž Prtenjak

Endava d.o.o, Ljubljana, Slovenija
matjaz@matjazev.net

V današnjem času umetne inteligence in podatkovno vodenih odločitev ni več vprašanje ali, temveč, kako hitro lahko razvijalci vklopimo napredne funkcionalnosti v naše aplikacije. Python je nesporni kralj podatkovne znanosti in umetne inteligence – vendar mnogi razvijalci še vedno ustvarjamo poslovne aplikacije v C# in .Net. Prispevek demonstrira neposredno izvajanje Python kode znotraj .Net aplikacij, povezovanje z orodji, kot so: NumPy, Pandas, Matplotlib, TensorFlow, PyTorch, OpenCV, itd., uporabo virtualnih okolij, tesno integracijo z .Net tipi (tudi naprednimi, kot je Span). Prispevek je smiselno nadaljevanje mojih prispevkov iz OTS 2019 (Vue), 2022 (Blazor) in 2023 (Blazor Hybrid), vendar pa je vseeno zaključen prispevek, neodvisen od prejšnjih prispevkov in predznanja. Nadaljevanje pa je v smislu, da .Net okolje zopet nadgrajujemo in povezujemo z zunanjim svetom.

Ključne besede:
.Net,
Python,
integracije,
REST,
Blazor.

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.15

1 Uvod

Python in .NET sta orodji/okolji, ki blestita na različnih področjih. .NET je robusten, strukturiran in primeren za večja podjetja, medtem ko Python s svojo preprostostjo in obsežno zbirko knjižnic izstopa predvsem na področju analize podatkov, umetne inteligence in avtomatizacije.

V povezavi ponujata najboljše iz obeh svetov: zmogljivost in širino uporabe.

Ta članek opisuje, kako lahko razvijalci uporabijo Python-ove zmogljivosti neposredno znotraj svojih .NET aplikacij.

Programsko kodo v Python-u bomo klicali direktno iz C# programske kode, kot da bi bila del C# kode, kar pomeni, da ne bomo uporabljali kakšnih mikrostoritev, povezav preko REST, sporočilnih vrst ali česa podobnega.

Celotne programske kode ne bo v članku, temveč jo lahko snamete s spleta, z GitHub-a, na naslovu: https://github.com/MPrtenjak/OTS2025-NET-And-Python

1.1. Zakaj povezati .Net in Python?

Glavni razlog je razpoložljivost brezplačnih in zmogljivih Python knjižnic. Tipičen primer je obdelava PDF dokumentov, kjer Python nudi knjižnice, kot je PyPDF2, brezplačno, medtem ko so .NET alternative pogosto plačljive.

V povezavi s Python svetom lahko:

- hitro prototipiramo in testiramo zamisli,
- uporabljamo napredne algoritme strojnega učenja,
- avtomatiziramo rutinske procese,
- razširimo funkcionalnosti brez dodatnih stroškov.

1.2. Namen prikazane aplikacije

Ker je bistvo prispevka v prikazu integracije .Net in Python okolja, ne bomo razvili »uporabne« aplikacije, temveč nekaj primerov integracije med okoljema. V vseh primerih bo realno delo opravljala Python koda .Net del rešitve pa bo služil samo za pripravo podatkov in prejem rezultatov.

Razvili bomo pet različnih primerov integracije, od najlažjega do (malce) bolj kompleksnega:

1. `HelloWorld`: klasičen programski primer 'Pozdravljen svet! Pozdravlja te <ime>.', kjer bo spremenljivko '`ime`' podal .Net del, sporočilo generiral Python in rezultat spet prejel .Net.
2. `PdfExtractor`: primer, kjer bomo Python kodi podali PDF dokument, Python bo v dokumentu prebral vso besedilo in nam ga vrnil kot niz znakov.
3. `ShowGraph`: primer, kjer bomo s pomočjo Python knjižnic `seaborn` in `matplotlib` ustvarili graf ter ga kot niz `byte`-ov vrnili .Net kodi, ki ga bo prikazala.
4. `OpenAI`: primer, kjer se bo Python preko knjižnice `openai` povezal z `OpenAI` ter generiral kratko otroško zgodbico o zmajčku.
5. `HTMLExtractor`: v tem primeru pa bomo z uporabo Python-ovih knjižnic `selenium`, `bs4` in programom `webdriver` pridobili žive podatke o cenah delnic na Ljubljanski borzi.

.Net del programa bo razvit v treh projektih:

1. Glavni del programa bo .Net knjižnica, ki bo v resnici skrbela za povezavo in predajo podatkov med .Net in Python-om.
2. Projekt s programom, ki teče v ukazni vrstici ter kliče prej omenjeno knjižnico.
3. REST API projekt s petimi API funkcijami, ki posledično kličejo prej omenjenih pet funkcij v Python-u.

2 Povezave med Python in .Net

Danes različne tipe programske opreme ali pa programsko opremo, ki je razvita v različnih ogrodjih, z različnimi programskimi jeziki, največkrat kličemo z medsebojno uporabo REST API funkcij ali pa slednje povežemo z uporabo sporočilnih vrst.

Vendar pa, kot je omenjeno že na začetku, danes ne bomo govorili o tovrstni povezavi, temveč bomo .Net in Python povezali neposredno v eno celoto.

Za povezavo med .Net in Python lahko uporabimo različne knjižnice, jaz pa bom danes omenil tri ter prikazal kratko tabelo z razlikami med njimi.

Tabela 1: Pregled razlik med tremi knjižnicami, uporabnimi za povezavo python in .Net.

Značilnost	IronPython	Python.Net	CSnakes
Podprta različica Pythona	Primarno Python 2.7; omejena podpora za Python 3	Podpira Python 2.x in 3.x	Podpira Python od 3.9 dalje
Integracija z .Net	Python koda teče na .Net-ovem Common Language Runtime (CLR)	Omogoča Python kodi dostop do .Net knjižnic; uporablja standardni CPython interpreter	Python v .Net teče z uporabo direktnih klicev Python-ovega C-API
Zmogljivost	Relativno najslabša hitrost delovanja med temi tremi	Na splošno boljša zmogljivost; neposreden dostop do .Net knjižnic	Visoka zmogljivost; neposredna integracija na C-nivoju
Podpora za Python knjižnice tretjih oseb	Omejena podpora, zlasti za module s C-razširitvami	Dobra podpora; omogoča uporabo večine Python knjižnic, tudi s C-razširitvami	Popolna podpora Python knjižnicam, vključno s C-razširitvami in virtualnimi okolji
Nitnost in GIL	Nima globalnega interpreter locka (GIL); boljša podpora večnitnosti	Podvržen GIL; omejena resnična večnitnost	Podpira CPython 3.13 način "free-threading"
Razvojno okolje	Integracija z Visual Studiom; podpora za .Net orodja	Zahteva standardna orodja za Python razvoj	Podpora za Visual Studio in druge IDE-je; omogoča "hot reload" Python kode
Primernost za uporabo	Idealno za aplikacije, ki močno temeljijo na .Net in vključujejo nekaj Python skriptiranja	Primerno za Python aplikacije, ki potrebujejo dostop do .Net knjižnic	Primerno za .Net aplikacije, ki zahtevajo visoko zmogljivo integracijo s Python knjižnicami

S pregledom na tabelo lahko sami ocenite primernost posamezne knjižnice za vaše potrebe, jaz pa se bom tokrat osredotočil na knjižnico CSnakes.

3 Postopek

V tem prispevku ne bom opisoval knjižnice kot takšne, saj je podroben opis seveda dostopen na spletu, bom pa opisal konkreten način uporabe ter postopke, ki sem jih uporabljal.

Kot vsako drugo knjižnico, lahko tudi CSnakes uporabite na različne načine in tudi sama omogoča več načinov inicializacije in uporabe. A ne glede na način uporabe, morate nekako skozi par korakov:

1. Priprava Python okolja:
   a. lahko vnaprej instalirate sami ali
   b. prepustite CSnakes knjižnici, da ga avtomatično naloži s spleta in instalira.
2. Priprava potrebnih knjižnic:
   a. lahko vnaprej instalirate sami ali
   b. prepustite CSnakes knjižnici, da jih poišče na spletu in instalira.
3. Uporaba kode v našem .Net programu.

3.2. Priprava Python okolja

V kolikor Python okolja ne pripravite vnaprej, lahko knjižnici zaukažete, naj ga sama poišče in instalira. Pri tem boste naleteli na sledeče težave:

1. Python se fiksno instalira v podmapo uporabnikove glavne mape, torej nekam na: `\users\<ime>\AppData\Roaming\SCnakes…`
   a. Če nič drugega, to pomeni, da bi imel vsak uporabnik vašega programa svojo kopijo Python-a.
2. Uporabnik mora imet dovolj pravic za instalacijo in zagon programov is podmape `\users`.
3. Program mora imeti dostop do spleta.

Seveda pa knjižnica »ni neumna« in pred instalacijo preveri, ali se Python na iskani lokaciji že nahaja in v tem primeru ga seveda ne instalira ponovno.

> 😎 ***Moje priporočilo:*** *Vaš instalacijski program naj vnaprej instalira ustrezno različico Python-a, najbolje kar ob vaš program in CSnakes naj potem uporablja to različico.*

3.3. Priprava Python knjižnic

Podobno kot pri instalaciji samega Python okolja, lahko tudi instalacijo Python knjižnic prepustite CSnakes knjižnici ali pa jih vi sami instalirate že vnaprej.

V primeru avtomatične instalacije Python knjižnic s strani Csnakes-a slednji uporabi program `pip` v Python podmapi in s pomočjo pip-a instalira vse zahtevane knjižnice. Seznam knjižnic mu podate vi preprosto tako, da jih naštejete v dokumentu z imenom `requirements.txt`.

> 😎 ***Moje priporočilo:*** *Vaš instalacijski program naj vnaprej pripravi virtualno Python okolje, kamor vi že vnaprej instalirate vse potrebne Python knjižnice.*

Kaj je Pythonovo virtualno okolje?

Pythonovo virtualno okolje je kot peskovnik za vaš projekt — ločeno, čisto okolje, kjer lahko nameščate knjižnice, ne da bi vplivali na preostali sistem. Ko ustvarite okolje z ukazom:

```
python -m venv moje_okolje
```

pravzaprav rečete Python-u: *Ustvari mi izolirano okolje (z imenom 'moje_okolje', kjer lahko delam, ne da bi karkoli pokvaril drugje).*

To okolje ima svojo kopijo interpreterja in svojo mapo `site-packages`, kjer se nahajajo vse knjižnice — ločeno od sistemske namestitve Python-a.

Zakaj je to uporabno?

1. **Izogibanje konfliktom**: različni projekti pogosto potrebujejo različne verzije knjižnic. Virtualna okolja vam omogočajo, da jih ločite — en projekt lahko brez težav uporablja Django 3.2, drugi pa Django 4.0.
2. **Čistejši razvoj**: ne obremenjujete sistemskega Python-a s knjižnicami, ki jih potrebujete le za vaš projekt. Upravljanje je enostavnejše, odstranjevanje pa brez posledic.
3. **Ponovljivost**: z ukazom `` `pip freeze > requirements.txt` `` lahko zamrznete trenutno stanje okolja, nekdo drug pa ga lahko natančno poustvari z `` `pip install -r requirements.txt` ``.
4. **Varno eksperimentiranje**: želite preizkusiti novo knjižnico ali različico? To lahko storite v virtualnem okolju, brez tveganja za druge projekte.

😎 ***Eureka!*** *Vse te točke skupaj pa so natanko tisto, kar potrebujete vi v vašem programu. Vaš program namreč ne sme vplivati na druge Python in sistemske nastavitve na uporabnikovem računalniku!*

Priprava navideznega Python okolja

Priprava Python virtualnega okolja je skrajno preprosta in tukaj prilagam skripto, ki utvari virtualno okolje z imenom `moj-peskovnik`, ter vanj instalira knjižnice `PyPDF2`, `seaborn` in `matplotlib`:

```
python -m venv moj-peskovnik
call moj-peskovnik\Scripts\activate

call pip install PyPDF2
call pip install seaborn
call pip install matplotlib
```

4 Konkreten primer

V tem poglavju bom prikazal celoten postopek razvoja .NET programa, ki bo uporabil Python za pridobitev besedila iz poljubnih PDF datotek.

4.1. Priprava Python okolja

Že v fazi razvoja celotne aplikacije si lahko – kot .Net razvijalci – pripravimo ločeno Python okolje, kjer bomo zadeve testirali. In natanko to lahko kasneje uporabimo tudi pri instalaciji naše aplikacije na uporabnikov računalnik.

Zatorej je najbolje izbrati kar »embeded« Python, torej Python, ki ga bolj ali manj instaliramo preprosto tako, da razširimo ZIP datoteko v neko mapo.

Prilagam Windows CMD datoteko, ki:

1. prenese Python s spleta,
2. ga razširi v podmapo,
3. ga skonfigurira,
4. instalira virtualna okolja,
5. ustvari delovno virtualno okolje ter ga aktivira,
6. in na koncu še instalira PyPDF2 knjižnico, ki jo potrebujemo za branje PDF datotek:

```
@echo off
setlocal

set PYTHON_FOLDER=Python312
set PYTHON_ZIP=python-embed.zip
set          PYTHON_URL=https://github.com/astral-sh/python-build-standalone/releases/
download/20250626/cpython-3.12.11+20250626-x86_64-pc-windows-msvc-pgo-full.tar.zst
set VENV_NAME=virtual-env

echo Prenos Python paketa...
curl -L -o %PYTHON_ZIP% %PYTHON_URL%

echo Razsirjanje v mapo %PYTHON_FOLDER%...
tar -xf %PYTHON_ZIP%
ren python %PYTHON_FOLDER%
del %PYTHON_ZIP%

echo Instalacija virtualnega okolja...
%PYTHON_FOLDER%\install\python.exe -m pip install virtualenv

echo Ustvarjanje virtualnega okolja...
%PYTHON_FOLDER%\install\python.exe -m virtualenv  %VENV_NAME%

echo Aktivacija virtualnega okolja...
call %VENV_NAME%\Scripts\activate.bat

echo Namestitev PyPDF2...
pip install PyPDF2

echo Koncano! Python je v mapi '%PYTHON_FOLDER%', virtualno okolje pa v '%VENV_NAME%'.

Endlocal
```

😎 ***Dobro je vedeti!*** *Github.com/astral-sh je spletišče, kjer lahko dobite različne 'embeded' Python verzije! Kot rečeno 'embeded' pač pomeni da ni instalacije, temveč samo razpakirate ZIP datoteko in začnete uporabljati Python!*

4.2. Python koda za branje PDF datoteke

Python torej imamo instaliran in lahko napišemo preprosto skripto, ki bo prebrala vse podane PDF datoteke in iz njih izluščila besedilo:

```
import PyPDF2
import sys

def extract_pdf_texts(file_paths: list[str]) -> list[str]:
    contents = []
    for path in file_paths:
        try:
            with open(path, 'rb') as f:
                reader = PyPDF2.PdfReader(f)
                text = ""
                for page in reader.pages:
                    text += page.extract_text() or ""
                contents.append(text)
        except Exception as e:
            contents.append(f"[ERROR reading {path}: {e}]")
    return contents

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python extract.py <pdf1> <pdf2> ...")
    else:
        pdf_paths = sys.argv[1:]
        results = extract_pdf_texts(pdf_paths)
        for i, content in enumerate(results, 1):
            # Prikažemo samo prvih 500 znakov
            print(f"\n--- PDF {i} ---\n{content[:500]}...")
```

Za potrebe .Net aplikacije seveda ne potrebujemo dela kode, ki se izvaja, če skripto poženemo v ukazni vrstici. Za potrebe .NET integracije potrebujemo samo in izključno funkcijo `extract_pdf_texts`, ki jo bomo v C# kodi poklicali.

> 😎 ***Opomba:*** *Če pogledate podpis funkcije extract_pdf_texts, boste videli tudi podatkovne tipe za vhodne in izhodne podatke. Funkcija torej sprejme seznam stringov (list[str]) (imena PDF datotek) in tudi vrne seznam stringov (vsebina posameznih datotek).*
>
> *S stališča Python kode to ni potrebno, Python je zelo svobodomiseln, kar se tiče podatkovnih tipov. Je pa to zelo priporočljivo za lažje povezovanje te kode v .Net okolje.*
>
> *Na ta način bo lahko razvojno okolje .Net takoj ugotovilo pričakovan podatkovni tip in ponudilo IReadOnlyList<string> kot vhod in izhod.*
>
> *To je nekaj podobnega kot JavaScript in TypeScript. Tudi JavaScript ne potrebuje oznak podatkovnih tipov, pa smo sedaj začeli uporabljati TypeScript, ki pa jih ima in je nam lažje … računalniku je tako ali tako vseeno* 😳

Če torej izvedemo prvo skripto, ki nam vzpostavi Python okolje, lahko takoj preizkusimo to Python skripto s klicem: `python.exe extract.py test1.pdf test2pdf test3pdf`.

4.3. Priprava .NET okolja

Kot omenjeno že na začetku, bom v članku prikazal minimalno delujočo kodo, na GitHub-u (https://github.com/MPrtenjak/OTS2025-NET-And-Python) pa lahko najdete popolno kodo z več primeri ter tako .NET REST API aplikacijo kot tudi preprosto aplikacijo v ukazni vrstici.

Tukaj bomo zatorej razvili samo aplikacijo v ukazni vrstici - z minimalno kode!

V ukazni vrstici lahko pripravimo prazen projekt z imenom `ExtractPdf2Txt`:

```
dotnet new console -n ExtractPdf2Txt -f net9.0
cd ExtractPdf2Txt
```

Dodati je potrebno še CSnakes knjižnico:

```
dotnet add package csnakes.runtime
```

In že lahko pričnemo z delom.

4.4. Python del našega projekta

Python kodo smo že zapisali v datoteki `extract.py` in zato to datoteko tudi skopiramo v naš projekt `ExtractPdf2Txt`.

> 😎 ***Opomba:*** *Python kode seveda ni potrebno imeti v glavni mapi vašega .Net projekta. Lahko se nahaja kjerkoli, je pa smiselno, da je nekako skupaj s projektom.*
>
> *Osebno imam navado vso Python kodo dajati v podmapo Python znotraj mojega .Net projekta.*

Ker naša Python funkcija za delovanje potrebuje zunanjo knjižnico `PyPDF2`, moramo v projekt dodati tudi datoteko `requirements.txt`, kjer naštejemo vse potrebne zunanje Python knjižnice.

> 😎 ***Opomba:*** *Datoteka requirements.txt je zelo pomembna, saj z njo knjižnici CSnakes naročite, katere zunanje knjižnice naj naloži, da bo vaš program deloval.*
>
> *V kolikor vi vnaprej pripravite virtualno okolje, to sicer ni pomembno, škodi pa tudi ne* 😊*.*
>
> *Seveda se lahko datoteka imenuje tudi drugače, vendar pa morate potem to novo ime sporočiti knjižnici CSnakes, v kolikor se držite standarda, pa vam ni potrebno storiti nič več.*

.Net okolje se mora zavedati Python kode, da jo lahko pregleda

Velika pridobitev CSnakes knjižnice je v dejstvu, da se zelo dobro integrira v .Net razvojno okolje, kar pomeni, da pozna podatkovne tipe, pozna vse funkcije v vaših Python datotekah in podobno.

Zato morate prevajalniku namigniti, naj tovrstne datoteke pregleda. To storite tako, da Python datoteke in datoteko `requirements.txt` označite kot dodatne datoteke, ki jih bo pregledal C# prevajalnik. V vašo `.csproj` datoteko morate zatorej dodati sledeče:

```
<ItemGroup>
  <AdditionalFiles Include="requirements.txt">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </AdditionalFiles>
  <AdditionalFiles Include="extract.py">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </AdditionalFiles>
</ItemGroup>
```

4.5. Glavni program

In sedaj smo prišli do dela, kjer moramo zares napisati naš C# program, ki bo sprejel seznam PDF datotek in izpisal besedilno vsebino podanih datotek.

Program mora torej narediti sledeče:

1. Preveriti vhodne parametre in za vsakega izmed njih preveriti, ali ima končnico PDF in ali obstaja datoteka s podanim imenom.
2. V kolikor datotek ni, to javiti uporabniku in končati program.
3. Inicializirati knjižnico CSnakes.
4. Pridobiti kazalec na Python datoteko `extract.py`.
5. V prej omenjeni datoteki izvesti funkcijo `extract_pdf_texts`, ji podati seznam datotek in sprejeti vrnjeno besedilo.
6. Izpisati prejeto besedilo.

Program

Prilagam program, ki natanko izvede omenjenih 6 korakov.

> 😎 ***Opomba:*** *To je seveda minimalen, a povsem delujoč program. Želja je, da vam prikažem pomembne dele programa, zato je preverjanje parametrov, delitev programske kode na funkcije in podobno pač opuščeno!*

```
using CSnakes.Runtime;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Reflection;

internal class Program
{
  private static void Main(string[] args)
  {
    var pdfFiles = args
        .Where(arg => File.Exists(arg) &&
                      Path.GetExtension(arg).Equals(".pdf",
StringComparison.OrdinalIgnoreCase));

    if (!pdfFiles.Any())
    {
      Console.WriteLine("No PDFs to convert.");
      return;
    }

    var exeFolder = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location)!;
    var baseFolder = Path.GetFullPath(Path.Combine(exeFolder, @"..\..\..\"));

    var host = Host.CreateDefaultBuilder(args)
      .ConfigureServices(s => s
        .WithPython()
        .FromFolder(Path.Combine(baseFolder, @"..\Python312\install"), "3.12")
        .WithVirtualEnvironment(Path.Combine(baseFolder, @"..\virtual-env"))
        .WithPipInstaller())
      .Build();

    var env = host.Services.GetRequiredService<IPythonEnvironment>();
    var module = env.Extract();
    var texts = module.ExtractPdfTexts(pdfFiles.ToArray());

    foreach (var text in texts)
      Console.WriteLine($"------------\n{text}\n------------\n");
  }
}
```

Pomembnejši deli programa

Začetek programa je preprost in gre samo za preverjanje vhodnih podatkov.

Nato določimo mapo, kjer se nahaja naš izvršljiv program (`exeFolder`) in na podlagi tega določimo, kje se nahaja naš projekt (`baseFolder`).

> 😎 ***Opomba:*** *Kot že omenjeno, je to minimalni program katerega namen je prikazati delovanje knjižnice in zato predvidevam (če sledite navodilom v tem prispevku), da program preizkušate v razvojnem okolju in da slednje program prevede v podmape bin/debug/net9.0.*
>
> *S tem namenom se moram za iskanje našega projekta (baseFolder) vrniti tri nivoje višje.*

Inicializacija knjižnice CSnakes

Knjižnico inicializiramo tako kot je v .Net dokaj standardno, namreč da najprej definiramo gostitelja (`host`) in ga nato zgradimo (`Build`).

Knjižnici CSnakes povemo:

1. da bomo uporabili Python (`WithPython`),
2. da se Python nahaja v podmapi `Python312` in da ima verzijo `3.12` (`FromFolder`),
3. da želimo uporabiti virtualno okolje z imenom `virtual-env` (`WithVirtualEnvironment`),
4. ter da želimo naj knjižnica uporabi `pip`, v kolikor je potrebno v virtualno okolje še instalirati kakšno knjižnico (`WithPipInstaller`).

Izvajanje Python kode

Z izgradnjo našega gostitelja lahko v kodi kadarkoli dosežemo objekt, ki je namenjen interakciji s Python kodo (`IPythonEnvironment`).

S tem objektom pridobimo možnost dostopa do vseh Python datotek, v našem primeru pač (`env.Extract`)

> 😎 ***Opomba:*** *Python datoteka, s katero delamo, se imenuje extract.py, zato se tudi naš modul znotraj .Net imenuje Extract. V kolikor bi v programu imeli še druge Python datoteke, recimo statistika.py ali numericne_metode.py, bi do slednjih v .Net dostopali kot env.Statistika() oz. env.NumericneMetode().*

Ko imamo dostop do modula, pa lahko v njem izvedemo katero koli funkcijo, ki se v modulu nahaja. V našem modulu se nahaja samo funkcija `extract_pdf_texts`, zato je tudi edina metoda, ki nam je v .Net dosegljiva pač `ExtractPdfTexts`.

> 😎 ***Opomba:*** *Kot ste lahko že opazili, knjižnica CSnakes inteligentno prevaja Python standarde »besede z malimi začetnicami in ločene s podčrtaji« v .net standarde »besede z velikimi začetnicami«.*

In to je vse. Klic Python funkcije nam vrne besedila vseh podanih PDF datotek in vse, kar je še potrebno storiti, je to izpisati uporabniku.

S tem smo preprosto in učinkovito povezali naše .Net okolje, naš C# program s Python okoljem.

5 Demonstracijska aplikacija dosegljiva na GitHub-u

V tem prispevku sem prikazal minimalni delujoč primer, na GitHub-u pa si lahko naložite demonstracijsko aplikacijo, ki prikaže razvoj sledečih modulov:

1. `HelloWorld`: klasičen programski primer 'Pozdravljen svet! Pozdravlja te <ime>.', kjer bo spremenljivko '`ime`' podal .Net del, sporočilo generiral Python in rezultat spet prejel .Net.
2. `PdfExtractor`: primer, kjer bomo Python kodi podali PDF dokument, Python bo v dokumentu prebral vso besedilo in nam ga vrnil kot niz znakov.
3. `ShowGraph`: primer, kjer bomo s pomočjo Python knjižnic `seaborn` in `matplotlib` ustvarili graf, ter ga kot niz `byte`-ov vrnili .Net kodi, ki ga bo prikazala.
4. `OpenAI`: primer, kjer se bo Python preko knjižnice `openai` povezal z `OpenAI` ter generiral kratko otroško zgodbico o zmajčku.
5. `HTMLExtractor`: v tem primeru pa bomo z uporabo Python-ovih knjižnic `selenium`, `bs4` in programom `webdriver`, pridobili žive podatke o cenah delnic na Ljubljanski borzi.

Literatura

[1] CSnakes, https://tonybaloney.github.io/CSnakes/, pridobljeno 1. 8. 2025

[2] Install .NET on Windows - .NET | Microsoft Learn, https://learn.microsoft.com/en-us/dotnet/core/install/windows, pridobljeno 1. 8. 2025

[3] Welcome to Python.org, https://www.python.org/, pridobljeno 1. 8. 2025

[4] Programska koda celotne aplikacije, https://github.com/MPrtenjak/OTS2025-NET-And-Python, pridobljeno 1. 8. 2025