

Leksioni 1

Endri Raco

21 April, 2024





Section 1

Hyrje



- Instalimi i Python
- Konceptet bazë të programimit në Python
- Analizimi i të dhënave me Python
- Vizualizimi i të dhënave në Python



- Procesimi i të dhënave vektor
- Procesimi i të dhënave raster
- Vizualizimi i të dhënave gjeografike
- Lidhja me burimet gjeografike online



- Interpolimi hapësinor
- Analiza e rrjetit hapësinor
- Analiza e terrenit



Section 2

Instalimi



- Python dhe libraritë e tij mund të instalohen lehtësisht duke përdorur paketa të ndryshme.
- Për të instaluar Python-in, **Miniconda** është një zgjedhje e mirë sepse ofron një mjedis të qëndrueshëm dhe mënjanon konfliktin e librarive.



Menaxhimi i Varësive midis librarive (dependency)

- Python ka një numër të madh librarish të disponueshme që mund të kenë varësi të ndërsjella.
- Është e rëndësishme që libraritë dhe versionet e tyre të punojnë mirë së bashku.
- Menaxhimi i librarive (package managers)



Pluset e përdorimit të Miniconda

- Miniconda përfshin një menaxher librarish që lehtëson instalimin dhe përditësimin.
- Ka support shumë të mirë
- Falas
- Ofron ndërfaqe grafike për lehtësi përdorimi



Mjediset Virtuale (Virtual environments)

- Mjediset virtuale krijojnë një hapësirë të izoluar për projektet tona Python.
- Krijimi i mjedisëve virtuale ndihmon për të shmangur konfliktet midis librarive dhe instalimeve të ndryshme.
- Mund të krijojmë mjedisë të shumta dhe të kalojmë lehtësisht mes tyre.



- Përdorim skedarët **YAML** për të dokumentuar konfigurimet e mjedisëve që krijojmë.
- Në skedarët YAML, mund të përcaktojmë specifikat e mjedisit, përfshirë versionin e Python-it dhe libraritë që do përdorim.



- Format i tipik për mjediset Conda/Mamba është **environment.yaml**



- Është një praktikë e mirë të instalojmë të gjitha libraritë (kur është e mundur) nga i njëjti kanal Conda, si p.sh., **conda-forge**, dhe të mos përziejmë **Conda** dhe **Pip** për instalime nëse nuk është e domosdoshme.



Çfarë është një Kanal Conda

- Një kanal Conda është një vendndodhje/server me një adresë të dedikuar në internet, ku ruhen libraritë.
- Kanali shërben si bazë për strehimin(repository) e librarive, dhe menaxherët e paketave (si Conda/Mamba) kërkojnë dhe shkarkojnë libraritë nga këto kanale.



Windows:

- Shkarkojmë versionin Miniconda të bazuar në Python 3 që është i përshtatshëm për sistemin operativ ku do punojmë.

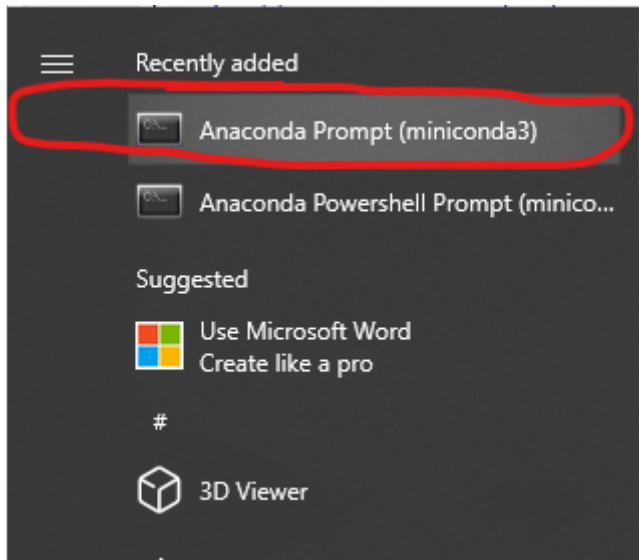
<https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links>

- Ndjekim udhëzimet e instalimit nga faqja e Miniconda.



Kontrolli i instalimit

- Hapim **Terminalin** ose **Anaconda Prompt**



- Për të siguruar që conda është instaluar siç duhet, ekzekutojmë komandën:



```
Anaconda Prompt (miniconda3)
(base) C:\Users\ENDRI>conda --version
conda 24.3.0
(base) C:\Users\ENDRI>_
```



- Mamba është një menaxher librarish për Miniconda.
- Për të instaluar **mamba**, hapim **Terminalin** ose **Command Prompt** në Windows si administrator.
- Ekzekutojmë komandën:

```
conda install mamba -n base -c conda-forge
```



- Do përdorim skedarin **environment.yml** që përmban listën e librarive të nevojshme
- Hapim Terminalin dhe shkoni te direktoria ku keni shkarkuar environment.yml



- Krijojmë mjedisin Python duke ekzekutuar:

```
mamba env create -f environment.yml
```



Për të aktivizuar mjedisin e ri:

```
conda activate pythongis
```



- Për të filluar **JupyterLab**, ekzekutojmë komandën në **Terminal** ose **Command Prompt**:

```
jupyter lab
```



- JupyterLab duhet të hapet automatikisht në një faqe browser



Për të instaluar paketa të reja, përdorim komandën:

```
mamba install -c conda-forge package-name
```



- Një shembull për instalimin e librarisë pandas nga kanali conda-forge:

```
mamba install -c conda-forge pandas
```



- Në rast se shfaqet ndonjë gabim, kontrollojmë versionet dhe kanalet e librarive ekzistuese me komandën:

```
mamba list
```



Çfarë është JupyterLab?

- JupyterLab është një mjet programimi i bazuar në shfletues (browser) për programim dhe data science.
- Ofron një mjedis të integruar që përfshin interpretues Python, redaktor teksti, terminal, dhe shumë më tepër.



Përbërësit Kryesorë të JupyterLab

• File Browser (Paneli i Navigimit):

The screenshot shows the JupyterLab interface with several panels and annotations:

- File Browser Panel (Left):** A red box highlights the left sidebar. An arrow points to the "+" button with the text "Open a 'Launcher' panel (+) or create a new folder". Below this, a table lists files in the directory "/chapter-01/nb/".
- Launcher Panel (Top Right):** A red box highlights the "Launcher" panel. It contains buttons for "Notebook", "Console", and "Other".
- Terminal Panel (Bottom):** A red box highlights the terminal window at the bottom, showing a command prompt.

Annotations:

- Navigation panel showing all the files in a directory** (points to the File Browser panel)
- A "Launcher" panel:** (points to the Launcher panel)
- Button for creating a new Notebook** (points to the Notebook button)
- Button for launching a Python interpreter** (points to the Console button)
- Buttons for opening a Terminal or creating files** (points to the Other button)
- Window for running terminal commands** (points to the terminal window)

Name	Last Modified
00-motivation.ipynb	3 months ago
01-computers-and-programs.ipynb	24 days ago
02-why-python.ipynb	an hour ago
03-writing-and-running-python-code.ipynb	an hour ago
04-using-jupyterlab.ipynb	11 minutes ago
05-installation.ipynb	an hour ago
06-references.ipynb	3 months ago

```
hertenka@hertenka-XPS-12 X
(base) hertenka@hertenka-XPS-15-7590:~/edu/Python-GIS-book/source$
```

Përbërësit Kryesorë të JupyterLab

The screenshot shows the JupyterLab web interface. On the left is the 'Launcher' panel, which contains a file browser and buttons for creating new content. The main area on the right shows a 'part1/chapter-01/nb' directory with buttons for creating a new Notebook, launching a Python interpreter, and opening a Terminal or creating files. At the bottom is a terminal window for running commands.

Open a "Launcher" panel (+) or create a new folder

A "Launcher" panel:

Navigation panel showing all the files in a directory

Button for creating a new Notebook

Button for launching a Python interpreter

Buttons for opening a Terminal or creating files

Window for running terminal commands

- **Launcher Panel (Paneli i Nisjes):**

- I vendosur në të djathtë të ndërfaqes.
- Përdoret për të krijuar elemente të rinj, si Jupyter Notebooks, skedarë të rinj teksti, ose terminale të reja.
- Mund të krijojmë dokumente të reja ose sesione të reja për programim nga ky panel.



Përbërësit Kryesorë të JupyterLab

Buttons to save the Notebook, create a new cell (+), cut/paste cells, run a cell, or interrupt the kernel

The screenshot shows the JupyterLab interface. At the top, there's a 'Launcher' tab and a notebook tab titled '00-python-basics.ipynb'. Below the tabs is a toolbar with icons for saving, creating a new cell, cutting, pasting, running a cell, and interrupting the kernel. A red box highlights these icons. To the right of the toolbar is a 'Cell Type Selector' dropdown menu with options: 'Code', 'Markdown', and 'Raw'. A red box highlights this menu. The main content area shows the notebook 'Basic elements of Python'. It contains two code cells. The first cell is highlighted with a blue bar on the left and contains the code `[1]: 1 + 1`. The second cell contains the code `[1]: 2`. Below these are two more code cells with the code `[2]: 5 * 7` and `[2]: 35`. A red box highlights the first two cells. Below the code cells is a markdown cell with the text 'If you want to edit and re-run some code, simply make changes to the Python cell and press **Shift-Enter** to execute the modified code.' and 'A Markdown cell with text that can be edited by double clicking'. Below this is a section titled 'Functions' with text about using Python for more advanced math by using a `function`. Functions are pieces of code that perform a single action such as printing information to the screen (e.g., the `print()` function). Functions exist for a huge number of operations in Python. Let's try out a few simple examples using functions to find the sine or square root of a value using the `sin()` and `sqrt()` functions.

Section 3

Programimi bazik në Python



Section 4

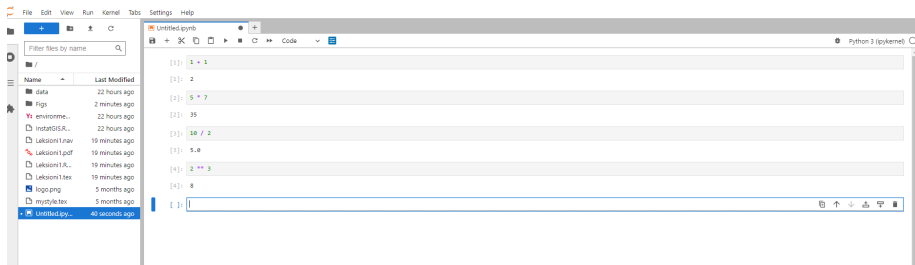
Matematika e Thjeshtë në Python



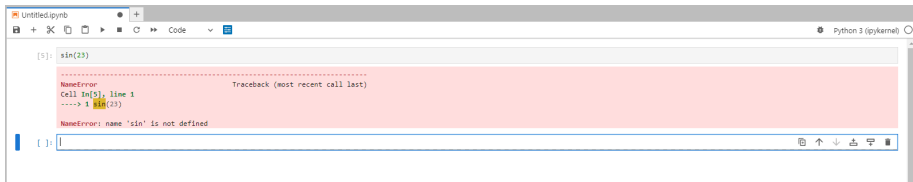
- Python mund të përdoret për të kryer operacione të thjeshta matematikore.
- Shembuj: $1 + 1$, $5 * 7$, $10 / 2$, $2 ** 3$.



- Në Jupyter Notebook, shtypim **Shift-Enter** për të ekzekutuar kodin.



Më shumë Veprime



The screenshot shows a Jupyter Notebook interface with a single code cell. The cell contains the text `sin(23)`. Below the code cell, a red error message is displayed, indicating a `NameError`. The error message includes a traceback showing the error occurred in the current cell at line 1, column 1. The specific error is `NameError: name 'sin' is not defined`. The notebook's title bar shows "Untitled.ipynb" and the kernel is identified as "Python 3 (ipykernel)".

```
[5]: sin(23)

-----
NameError                                Traceback (most recent call last)
Cell In[5], line 1
----> 1 sin(23)

NameError: name 'sin' is not defined
```



Çfarë është Një Funkcion?

- Funkcionet janë pjesë të kodit që kryejnë një veprim të vetëm, si p.sh., printimi i informacionit në ekran.
- Python ka një shumëllojshmëri të madhe funksionesh për operacione të ndryshme.



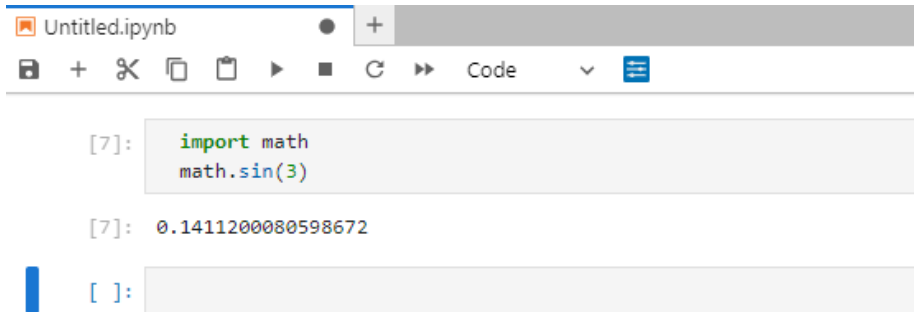
- Funksioni **print()** përdoret për të shfaqur tekst në ekran.
- Funksioni **len()** kthen gjatësinë e një liste ose vargu.
- Funksioni **sum()** llogarit shumën e elementeve në një listë.



- Për të bërë veprime më komplekse, përdorim module të tilla si `math`.
- Importojmë modulin `math` për të përdorur funksione si `math.sin()`, `math.sqrt()`, etj.




```
import math  
math.sin(3)  
math.sqrt(4)
```



The image shows a Jupyter Notebook interface. At the top, there is a tab labeled "Untitled.ipynb". Below the tab is a toolbar with various icons for file operations (save, new, close, copy, paste) and execution (run, step, code, help). The main area contains two code cells. The first cell, labeled "[7]:", contains the code `import math` and `math.sin(3)`. The second cell, also labeled "[7]:", displays the output `0.1411200080598672`. A third cell, labeled "[]:", is currently empty.

```
[7]: import math
      math.sin(3)
```

```
[7]: 0.1411200080598672
```

```
[ ]:
```

Ja përmbledhja e asaj që pamë:

- Çfarë Është Një Modul?
 - Një modul është një grup pjesësh kodi, si p.sh funksione, që janë të lidhura me njëri-tjetrin.
 - Modulet individuale shpesh përfshihen në një grup të quajtur bibliotekë.



- Si Të Ngarkoni Një Modul?
 - Për të ngarkuar një modul, përdorni deklaratën **import**.
 - Funkcionet që janë pjesë e një moduli mund të përdoren duke shkruar **modulename.functionname()**
 - Për shembull, **sin()** është një funksion i modulit **math**, dhe përdoret duke shkruar **math.sin()** me një numër brenda kllapave.



- Përdorimi i Moduleve në Jupyter Notebook

- Në Jupyter Notebook, variablat që përcaktohen në qelizat e mëparshme janë të disponueshme për përdorim në qelizat që pasojnë, për sa kohë që ato janë ekzekutuar më parë.
- Kjo lejon ruajtjen e variablave për përdorim të mëtejshëm gjatë ekzekutimit të kodit. Konstantet në Module
- Modulet gjithashtu mund të përmbajnë konstante si **math.pi**

Për të thirrur konstante, nuk përdoren kllapa; thjesht shkruajmë emrin e konstantes



Section 5

Kombinimi i Funksioneve në Python



Kombinimi i Funksioneve me print()

- Funksioni `print()` shfaq tekstin në ekran.
- Për të printuar rezultatin e një funksioni tjetër, përdorim `print()` brenda kodit:

```
import math  
print(math.sqrt(4))    # Shfaq 2.0
```



Kombinimi i Tekstit me Vlerat e Llogaritura

- Përdorim **print()** për të shfaqur tekst dhe vlera të llogaritura së bashku.

Shembull:

```
print("Dy plus dy është", 2 + 2)  # Shfaq "Dy plus dy është 4"
```



- Kombinojmë funksione të ndryshme për të prodhuar rezultat më të avancuar:

Shembull:

```
print("Rrënja katrore e 4 është", math.sqrt(4))
```



Përdorimi i Variablave në Python

- Për të caktuar vlerën e një variabli, përdorni =:

```
temp_celsius = 10.0
```



Përdorimi i Variablave në Python

- Për të parë vlerën e një variabli, përdorim **print()** ose thjesht emrin e variablit në një qelizë Jupyter Notebook:

```
temp_celsius    # Kthen 10.0
```



- Për të kombinuar tekst dhe vlera të llogaritura, përdorim **print()**

Shembull:

```
print("Temperatura në Fahrenheit:", 9 / 5 * temp_celsius + 32)
```



- Variablat mund të përditësohen me vlera të reja:

Shembull:

```
temp_celsius = 15.0  
print("Temperatura në Celsius është tani:", temp_celsius)
```



- Nëse përpiqeni të përdorni një variabël që nuk është definuar, do të merrni një **NameError**



Gabimi **NameError**

- Ky gabim ndodh kur një variabël ose funksion nuk është definuar.

```
print("Temperature in Celsius:", 5 / 9 * (tempFahrenheit - 32))
```



Trajtimi i Gabimeve të Zakonshme në Python

- Për të zgjidhur këtë gabim, sigurohemi që të gjitha variablat dhe funksionet të jenë të definuara dhe të importuara siç duhet.

```
tempFahrenheit = 9 / 5 * temp_celsius + 32
```



Section 6

Llojet e të Dhënave në Python



Llojet e të Dhënave në Python

- Lloji i të dhënave përcakton karakteristikat e të dhënave në një program.
- Python ka katër lloje bazë të të dhënave: *int*, *float*, *str* dhe *bool*.



- **int:** vlera të plota të numrave të plotë.
- **float:** vlera dhjetore.
- **str:** vargje karakteresh (tekste).
- **bool:** vlera të tipit të vërtetë/false.



int: 4

float: 3.1415

str: 'Hot'

bool: True



- Përdorim funksionin **type()** për të marrë llojin e të dhënave të një variabli.

Shembull:

```
weatherForecast = "Hot"  
type(weatherForecast)  # Kthen "str"
```



- Llojet e të dhënave janë të rëndësishme sepse disa prej tyre nuk janë kompatibël me njëri-tjetrin.
- Për shembull, nuk mund të mbledhim një int me një str.



Gabimi **TypeError**

- Ky gabim ndodh kur përpiqeni të kryejmë operacione me tipe të ndryshme të dhënave që nuk janë kompatibël.

Shembull:

```
tempFahrenheit + 5.0 * "Hot" # Kthen TypeError
```

- Për të shmangur këtë gabim, sigurohemi që të gjitha llojet e dhënave të jenë të kompatibël përpara se të kryejmë veprime.



Çfarë Është Një Listë?

- Një listë është një koleksion vlerash të lidhura së bashku me një variabël të vetme.
- Listat mund të përmbajnë lloje të ndryshme të dhënash, si numra, vargje, ose madje edhe lista të tjera.



Krijimi i Një Liste

- Për të krijuar një listë, përdorim kllapat katrore [] dhe ndajmë elementët me presje.
- Shembull:

```
station_names = ["Tirane", "Durrës", "Elbasan", "Sarandë"]
```



- Përdorim funksionin **type()** për të kontrolluar nëse një variabël është një listë.
- Shembull:

```
type(station_names)  # Kthen "list"
```



- Listat mund të përdoren për të ruajtur shumë vlera të lidhura.
- Në Python, listat janë një nga llojet më të zakonshme të koleksioneve.



- Një indeks është një numër që tregon një pozicion në listë.
- Indeksi i parë është gjithmonë 0, prandaj për të marrë elementin e parë të një liste, përdorim indeksin 0.

```
station_names[0]    # Kthen "Tirane"
```



Marrja e Një Vlere nga Një Listë

- Për të marrë një vlerë nga një listë, përdorim indeksin e duhur.
- Shembull:

```
station_names[1]  # Kthen "Durrës"
```



Për të marrë elementë nga fundi i një liste, përdorim indekse negativë.

- Shembull:

```
station_names[-1]  # Kthen "Sarande"
```

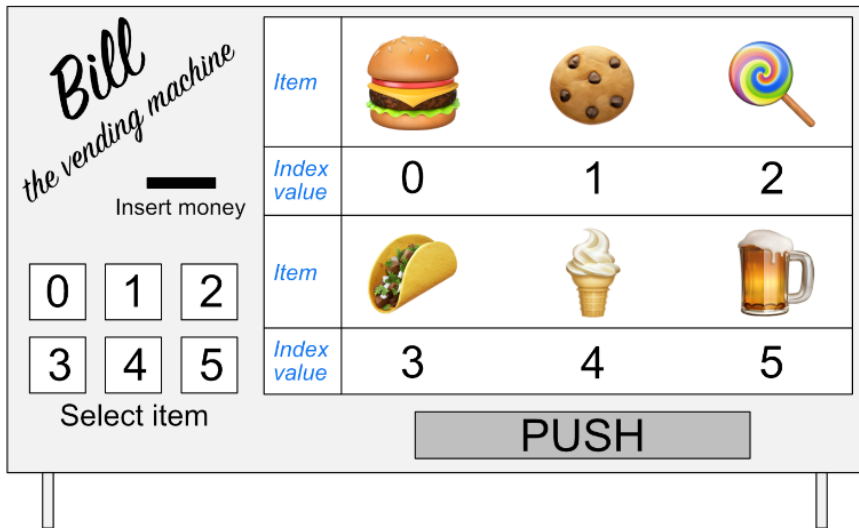


Kujdes me Indekset Jashtë Kufijve

- Nëse përdorim një indeks që është jashtë kufijve të listës, do të marrim një **IndexError**.
- Shembull:

```
station_names[4]    # Kthen "IndexError: list index out of range"
```





- Makina automatike që përmban 6 artikuj.
- Si Python, makina automatike përdor indekse për të zgjedhur artikujt.
- Indeksi i parë është gjithmonë 0, dhe numri rritet me njësi.
- Për të marrë një artikull nga makina automatike, duhet të përdorim indeksin e duhur.



- Për të marrë një taco, do të zgjidhnit butonin 3.
- Në Python, për të marrë një artikull nga një listë, përdorim indeksin përkatës:

```
Bill[3] # Kthen "Taco"
```



Gjetja e Gjatësisë së Një Liste

- Për të marrë gjatësinë e një liste, përdorim funksionin `len()`:

```
qytete = ["Tiranë", "Durrës", "Shkodër", "Vlorë"]  
len(qytete)  # Kthen 4
```



Përdorimi i `len()` për të gjetur vlerën e fundit të një liste

- Duke përdorur **`len()`**, mund të gjejmë indeksin e fundit të një liste.
- Indeksi i fundit është gjithmonë **`len(qytete) - 1`**:

```
qytete[len(qytete) - 1]  # Kthen "Vlorë"
```



Përdorimi i len() për të gjetur vlerën e fundit të një liste

- Për të marrë vlerën e fundit, përdorim indeksin -1:

```
qytete[-1]  # Kthen "Vlorë"
```



- Nëse përdorim një indeks që është jashtë kufijve të listës, do të merrni një `IndexError`.

```
qytete[4]    # Kthen "IndexError: list index out of range"
```



Përdorimi i Indeksimit Negativ

- Indeksi negativ na lejon të marrim elementë nga fundi i një liste.
- Indeksi -1 jep vlerën e fundit, ndërsa indekset me vlera më të mëdha negative shkojnë drejt fillimit të listës:

```
qytete = ["Tiranë", "Durrës", "Shkodër", "Vlorë"]  
qytete[-1]    # Kthen "Vlorë"  
qytete[-2]    # Kthen "Shkodër"
```



Kujdes me Indekset Negativë

Edhe pse indeksimi negativ është i dobishëm, përdorimi i një indeksi jashtë kufijve shkakton **IndexError**:

```
qytete[-5] # Kthen "IndexError: list index out of range"
```



- Indeksi **-len(qytete)** jep vlerën e parë në listë:

```
qytete[-len(qytete)]    # Kthen "Tiranë"
```



- Një nga avantazhet e listave është se ato mund të ndryshohen pasi të krijohen.
- Për të ndryshuar një vlerë në një listë, përdorim indeksin për të përcaktuar pozicionin e vlerës që duam të ndryshojmë.



Shembull: Listë e Qyteteve Shqiptare

- Le të krijojmë një listë që përfshin qytetet kryesore shqiptare:

```
qytete = ["Tiranë", "Durrës", "Shkodër", "Vlorë"]
```



- Për të ndryshuar qytetin e tretë në listë, përdorim indeksin përkatës:

```
qytete[2] = "Elbasan"  # Ndryshon Shkodrën në Elbasan
```



Pas modifikimit të vlerës, mund të printojmë listën për të parë ndryshimin:

```
print(qytete)  # Kthen ["Tiranë", "Durrës", "Elbasan", "Vlorë"]
```



- Indeksi i parë është gjithmonë 0, kështu që për të ndryshuar vlerën e parë, përdorim indeksin 0:

```
qytete[0] = "Korçë"  # Ndryshon Tiranën në Korçë
```



- Për të ndryshuar vlerën e fundit, përdorim indeksin -1:

```
qytete[-1] = "Sarandë"  # Ndryshon Vlorën në Sarandë
```



Përdorimi i Indekseve për Modifikim

Pas modifikimit, printoni listën për të verifikuar ndryshimin:

```
print(qytete)  # Kthen ["Korçë", "Durrës", "Elbasan", "Sarandë"]
```



Kujdes me Indekset Jashtë Kufijve

Nëse përdorni një indeks që është jashtë kufijve të listës, do të marrim një **IndexError**.



Shembuj të Tjerë të Modifikimit

Për të ndryshuar vlerat e mesme, përdorim indeksin përkatës:

```
qytete[1] = "Fier"  # Ndryshon Durrësin në Fier
```



Përdorimi i Indekseve për Modifikim

Pas modifikimit, printoni listën për të verifikuar ndryshimin:

```
print(qytete)  # Kthen ["Korçë", "Fier", "Elbasan", "Sarandë"]
```



Lista që Përmban Tipa të Ndryshëm të Dhënave

- Një nga përfitimet e një liste është se ajo mund të përmbajë lloje të ndryshme të dhënash në të njëjtin koleksion.
- Për shembull, le të krijojmë një listë që përmban emrin e një qyteti shqiptar, kodin e postës, koordinatat gjeografike, dhe një atribut tjetër.

```
city_name = "Tiranë"  
postal_code = 1001  
city_lat = 41.3275  
city_lon = 19.8189  
major_landmark = "Sheshi Skënderbej"
```



Lista që Përmban Tipa të Ndryshëm të Dhënave

- Kombinojmë këto variabla në një listë që përmban lloje të ndryshme të dhënash:

```
city_info = [city_name, postal_code, city_lat, city_lon, major]
```



- Përdorim **type()** për të konfirmuar se është një listë:

```
type(city_info)  # Kthen "list"
```



Kontrollimi i Llojit të Listës

- Për të kontrolluar llojet e të dhënave brenda listës, përdorim indekse të ndryshme:

```
type(city_info[0])  # Kthen "str"  
type(city_info[1])  # Kthen "int"  
type(city_info[2])  # Kthen "float"
```



Kujdes me Përzierjen e Llojeve të Dhënave

- Edhe pse një listë mund të përmbajë lloje të ndryshme, mund të jetë problematike në disa raste.
- Rekomandohet që listat të kenë lloje të ngjashme për të shmangur probleme në analizat e të dhënave.



Shtimi dhe Heqja e Vlerave nga Listat

- Le të kemi një listë me emrat e disa qyteteve kryesore në Shqipëri:

qytete = ["Tiranë", "Durrës", "Vlorë", "Shkodër"]



- Për të hequr vlerën e parë nga lista, përdorni `del` me indeksin e duhur:

```
del qytete[0]    # Heq "Tiranë"  
print(qytete)   # Kthen ['Durrës', 'Vlorë', 'Shkodër']
```



- Metoda **remove()** mund të përdoret për të hequr një element specifik nga lista pa përdorur indeksin:

```
qytete.remove("Durrës") # Heq "Durrës"  
print(qytete) # Kthen ['Vlorë', 'Shkodër']
```



- Për të shtuar vlera në fund të listës, përdorni `append()`:

```
qytete.append("Elbasan")  
qytete.append("Korçë")  
print(qytete)    # Kthen ['Vlorë', 'Shkodër', 'Elbasan', 'Korçë']
```



- Metoda `append()` është projektuar për listat, duke shtuar elementë në fund të një liste.
- Nuk ka kuptim ta përdorim për lloje të tjera të dhënash si `int`, sepse një numër i plotë nuk është një koleksion.



- Le të kemi një listë me emrat e disa qyteteve shqiptare:

```
qytete = ["Tiranë", "Durrës", "Vlorë", "Shkodër"]
```



Kur gjatësia e listës llogaritet, rezultati është një int:

```
qytete_length = len(qytete)  # Kthen 4
```



Çfarë ndodh kur përdorim `append` me `int`?

- Duke qenë se `int` është një lloj i dhënash që përfaqëson vlera të plota, nuk mund të shtojmë vlera në mënyrë sekuenciale si te një listë.
- Përpjekja për të përdorur **`append()`** me një `int` shkakton një **`AttributeError`**:

```
qytete_length.append(1)  # Kthen "AttributeError: 'int' object"
```



Zgjidhje për Shtimin e Vlerave të Tipit int

- Për të rritur një int, përdorim operacionet aritmetike të zakonshme:

```
qytete_length += 1  # Rrit vlerën me 1
```



Numërimi i Një Vlere në Listë

- Përdorim metodën `count()` për të parë sa herë një vlerë shfaqet në një listë.
- Shembull:

```
qytete = ["Tiranë", "Durrës", "Vlorë", "Shkodër", "Tiranë"]  
qytete.count("Tiranë")  # Kthen 2
```



Gjetja e Indeksit të Një Vlere

- Metoda *index()* na lejon të gjejmë indeksin e parë ku shfaqet një vlerë.
- Shembull:

```
qytete.index("Vlorë") # Kthen 2
```



Kthimi i Renditjes së Një Liste

- Për të kthyer renditjen e listës, përdorim metodën *reverse()*:

```
qytete.reverse()  
print(qytete)    # Kthen ['Shkodër', 'Vlorë', 'Durrës', 'Tiranë']
```



Kthimi i Renditjes së Një Liste

Kujdes kur përdorni *reverse()* dhe mos caktoni rezultatin në të njëjtën listë, pasi kjo do të shkaktojë një None:

```
qytete = qytete.reverse()  # Gabim! Kthen None dhe fshin përmbajtjen
```



Renditja e Një Liste Alfabetikisht

- Metoda *sort()* mund të përdoret për të renditur listën në mënyrë alfabetike:

```
qytete.sort()  
print(qytete)    # Kthen ['Durrës', 'Shkodër', 'Tiranë', 'Vlorë']
```



- Për të shmangur gabimet, mos caktoni rezultatin e `sort()` në listën e njëjtë, pasi kjo do të shkaktojë një `None`:

```
qytete = qytete.sort()  # Gabim! Kthen None dhe fshin përmbajtjen
```



Definimi i Variablave me Tipa të Ndryshëm

- Në këtë shembull, kemi pesë variabla që përfaqësojnë të dhëna të ndryshme për një qytet ose një stacion:

```
station_name = "Tiranë"  
station_id = 1001  
station_lat = 41.3275  
station_lon = 19.8189  
station_type = "Kryeqytet"
```



- Përdorim funksionin *type()* për të kontrolluar llojin e një variabli:

```
type(station_name)  # Kthen "str"  
type(station_id)    # Kthen "int"  
type(station_lat)   # Kthen "float"
```



- Nëse përpiqeni të bashkoni një varg me një numër të plotë, do të shkaktoni një `TypeError`:

```
station_name + station_id  # Kthen "TypeError: can only concat
```



Konvertimi i Të Dhënave për Përputhshmëri

- Për të kombinuar një varg me një numër të plotë, duhet të konvertojmë numrin në një varg duke përdorur funksionin **str()**:

```
station_id_str = str(station_id)
station_name + station_id_str  # Kthen "Tiranë1001"
```



Konvertimi i Të Dhënave për Përputhshmëri

- Në mënyrë të ngjashme, mund të konvertojmë një varg ose një numër dhjetor në një numër të plotë duke përdorur *int()*, ose në një numër dhjetor duke përdorur *float()*: python

```
float("3.14")  # Kthen 3.14 (numër dhjetor)  
int("1001")   # Kthen 1001 (numër i plotë)
```



Section 7

Manipulimi i tekstit



Kombinimi i Tekstit dhe Numrave në Python

- Një mënyrë e zakonshme për të kombinuar vargje dhe numra është përdorimi i operatorit të shtimit +.
- Në këtë shembull, ne kombinojmë një emër qyteti dhe një kod postar për të krijuar një varg të ri:

```
city_name = "Tiranë"  
postal_code = 1001  
city_info = city_name + " - " + str(postal_code)  
city_info  # Kthen "Tiranë - 1001"
```



Përdorimi i F-Strings për Manipulimin e Tekstit

- Një metodë më e avancuar për të bashkuar vargjet dhe numrat është përdorimi i f-strings.
- F-strings lejojnë futjen e variablave brenda vargjeve në mënyrë dinamike.
- Shembull:

```
temp = 18.56789876
f_string_example = f"Temperatura në {city_name} është {temp:.2f}"
f_string_example # Kthen "Temperatura në Tiranë është 18.57"
```



- Përdorni shkronjën `f` para vargut për të krijuar një f-string.
- Variablat mund të futen brenda vargut duke përdorur kllapat `{}`.



- Është gjithashtu e mundur të përdorni formatime për të rregulluar precizionin e numrave:

```
f_string_example = f"Temperatura është {temp:.2f} gradë Celsius"
```



Teknikat e Tjera për Manipulimin e Tekstit

- Përveç f-strings, ekzistojnë edhe metodat **.format()** dhe **%** për manipulimin e tekstit.
- Megjithatë, f-strings janë metoda e rekomanduar për shkak të thjeshtësisë dhe fleksibilitetit.
- Shembull me **.format()**:

```
format_example = "Temperatura në {} është {:.2f} gradë".format(18.57, "Tiranë")  
format_example # Kthen "Temperatura në Tiranë është 18.57 gradë"
```



Shembull me % (më pak i rekomanduar):

```
percent_example = "Temperatura në %s është %.2f gradë" % (city, temp)
percent_example  # Kthen "Temperatura në Tiranë është 18.57 gradë"
```



Ndarja e Një Vargu me .split()

- Metoda `.split()` lejon ndarjen e një vargu bazuar në një karakter të caktuar.
- Shembull:

```
tekst = "Qytete: Tiranë, Durrës, Shkodër"  
pjesë_te_ndara = tekst.split(":")  
pjesë_te_ndara # Kthen ['Qytete', ' Tiranë, Durrës, Shkodër']
```



Zëvendësimi i Një Fjale me `.replace()`

- Metoda `.replace()` lejon zëvendësimin e një fjale ose grupi karakteresh me një tjetër.
- Shembull:

```
tekst_ndryshuar = pjese_te_ndauro[1].replace("Tiranë", "Elbasan")  
tekst_ndryshuar # Kthen ' Elbasan, Durrës, Shkodër'
```



Prerja e Një Vargu për të Hequr Hapësirat e Panevojshme

- Për të prerë karakteret e panevojshme nga fillimi ose fundi i një vargu, mund të përdorim metodën `.strip()` ose të përdorim prerjen:

```
tekst_prere = pjese_te_ndaura[1][1:] # Hiq hapësirën në fillim  
tekst_prere.strip() # Hiq hapësirat nga fillimi dhe fundi
```



Ndryshimi i Shkronjave me `.upper()` dhe `.lower()`

- Për të ndryshuar të gjitha shkronjat në shkronja të mëdha, përdorni metodën **`.upper()`**:

```
tekst_prere.upper()    # Kthen 'ELBASAN, DURRËS, SHKODËR'
```



Ndryshimi i Shkronjave me `.upper()` dhe `.lower()`

- Për të kthyer të gjitha shkronjat në të vogla, përdorni metodën `.lower()`:

```
tekst_prere.lower()  # Kthen 'elbasan, durrës, shkodër'
```



- Për të kapitalizuar vetëm shkronjën e parë, përdorni metodën **.capitalize()**:

```
tekst_prere.capitalize()  # Kthen 'Elbasan, durrës, shkodër'
```



Section 8

Ciklet “For”



- Ciklet “for” na lejojnë të përsërisim një bllok kodi disa herë, si iterimi mbi një listë dhe kryerja e një veprimi për çdo element.
- Ato janë të dobishme për të automatizuar veprimet që përsëriten, për të shmangur gabimet dhe për të përmirësuar shkallëzueshmërinë e kodit.



Si shembull, le të përpiqemi të marrim qytetet shqiptare me indeksim manual:

```
qytete_shqiptare = ["Tiranë", "Durrës", "Shkodër", "Vlorë"]
qytete_shqiptare[0]    # 'Tiranë'
qytete_shqiptare[1]    # 'Durrës'
qytete_shqiptare[2]    # 'Shkodër'
qytete_shqiptare[3]    # 'Vlorë'
qytete_shqiptare[4]    # Kjo do të shkaktojë IndexError
```



- Shembulli i përdorimit të indeksimit manual në listë
- Mund të shkaktojë gabime `IndexError` dhe nuk është i shkallëzueshëm për lista të gjata.



Përdorimi i ciklit “for” për të iteruar mbi një listë

- Kjo metodë është më fleksibël dhe e shkallëzueshme.
- Shembulli i përdorimit të ciklit “for” për të iteruar mbi listën e qyteteve shqiptare



Përdorimi i ciklit “for” për të iteruar mbi një listë

```
qytete_shqiptare = ["Tiranë", "Durrës", "Shkodër", "Vlorë"]  
for qytet në qytete_shqiptare:  
    print(qytet)
```

