

Jupyter Notebooks

Madhavan Mukund

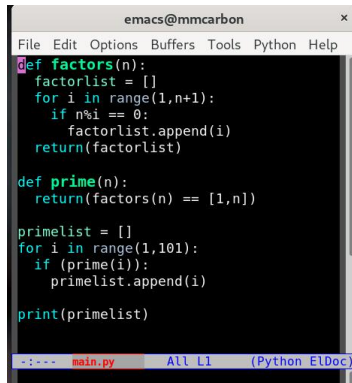
<https://www.cmi.ac.in/~madhavan>

Programming, Data Structures and Algorithms using Python
Week 1

Writing and running code

■ Manual

- Text editor to write code
- Run at the command line



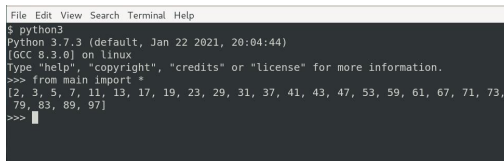
```
emacs@mmcarbon
File Edit Options Buffers Tools Python Help
def factors(n):
    factorlist = []
    for i in range(1,n+1):
        if n%i == 0:
            factorlist.append(i)
    return(factorlist)

def prime(n):
    return(factors(n) == [1,n])

primelist = []
for i in range(1,101):
    if (prime(i)):
        primelist.append(i)

print(primelist)

-:--- main.py All L1 (Python ElDoc)
```



```
File Edit View Search Terminal Help
$ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from main import *
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
>>>
```

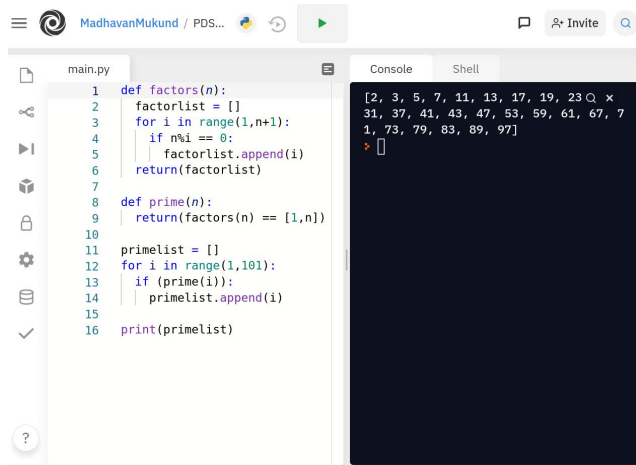
Writing and running code

■ Manual

- Text editor to write code
- Run at the command line

■ Integrated Development Environment (IDE)

- Single application to write and run code
- On desktop or online, [replit](#)
- Quick update-run cycle
- Debugging, testing, ...



The screenshot shows a Jupyter Notebook interface with a file named `main.py`. The code defines two functions: `factors(n)` which returns a list of factors of `n`, and `prime(n)` which returns a boolean indicating if `n` is prime. The code then finds all primes up to 101 and prints them.

```
1 def factors(n):
2     factorlist = []
3     for i in range(1,n+1):
4         if n%i == 0:
5             factorlist.append(i)
6     return(factorlist)
7
8 def prime(n):
9     return(factors(n) == [1,n])
10
11 primelist = []
12 for i in range(1,101):
13     if (prime(i)):
14         primelist.append(i)
15
16 print(primelist)
```

The console output shows the list of primes: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`.

Writing and running code

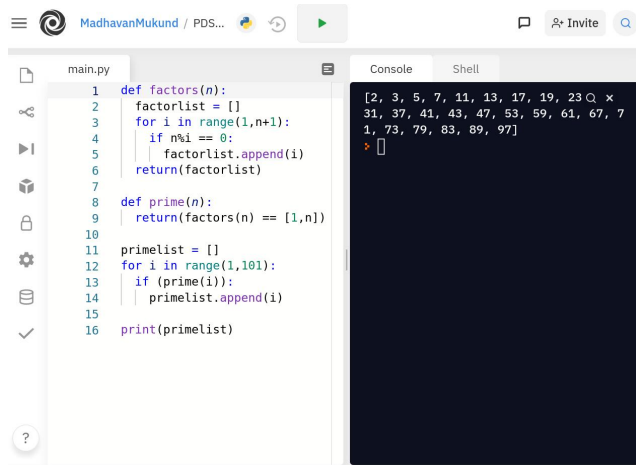
■ Manual

- Text editor to write code
- Run at the command line

■ Integrated Development Environment (IDE)

- Single application to write and run code
- On desktop or online, [replit](#)
- Quick update-run cycle
- Debugging, testing, ...

■ What more could one want?

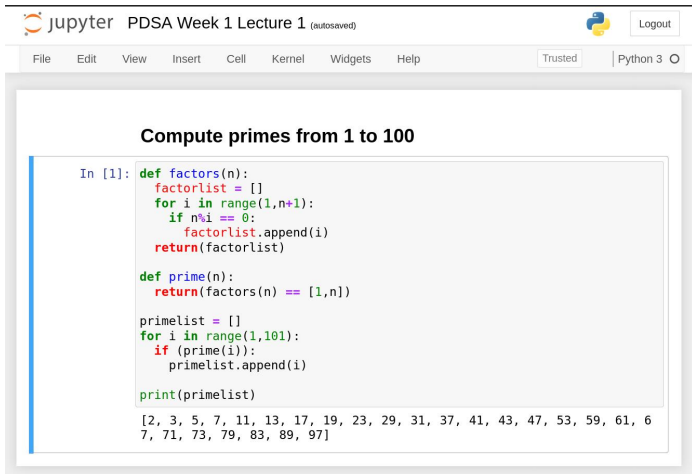


The screenshot shows a Jupyter Notebook interface with a file named `main.py`. The code defines two functions: `factors(n)` which returns a list of factors of `n`, and `prime(n)` which returns a list of prime factors of `n`. The code then finds the prime factors of 101 and prints them.

```
1 def factors(n):
2     factorlist = []
3     for i in range(1,n+1):
4         if n%i == 0:
5             factorlist.append(i)
6     return(factorlist)
7
8 def prime(n):
9     return(factors(n) == [1,n])
10
11 primelist = []
12 for i in range(1,101):
13     if (prime(i)):
14         primelist.append(i)
15
16 print(primelist)
```

The console output shows the list of prime factors of 101: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`.

- Share your code
 - Collaborative development
 - Report your results



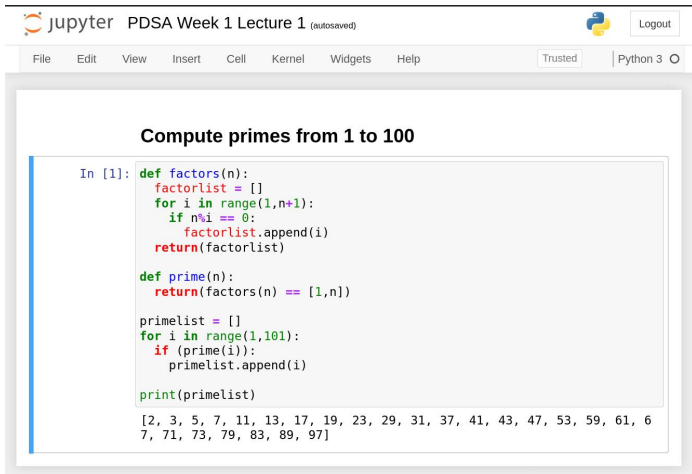
The screenshot shows a Jupyter Notebook window titled "jupyter PDSA Week 1 Lecture 1 (autosaved)". The interface includes a top bar with a "Logout" button and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The notebook content is titled "Compute primes from 1 to 100". It contains a code cell with the following Python code:

```
In [1]: def factors(n):  
        factorlist = []  
        for i in range(1,n+1):  
            if n%i == 0:  
                factorlist.append(i)  
        return(factorlist)  
  
        def prime(n):  
            return(factors(n) == [1,n])  
  
        primelist = []  
        for i in range(1,101):  
            if (prime(i)):  
                primelist.append(i)  
  
        print(primelist)  
  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 6  
7, 71, 73, 79, 83, 89, 97]
```

The output of the code is a list of prime numbers from 2 to 97, displayed across two lines.

Collaboration

- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code



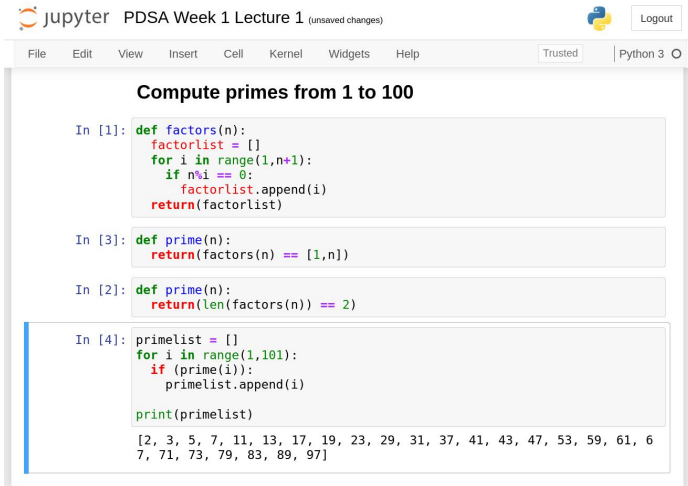
The screenshot shows a Jupyter Notebook window titled "jupyter PDSA Week 1 Lecture 1 (autosaved)". The interface includes a top bar with a "Logout" button and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The notebook content is titled "Compute primes from 1 to 100". It contains a Python code cell with the following code:

```
In [1]: def factors(n):  
        factorlist = []  
        for i in range(1,n+1):  
            if n%i == 0:  
                factorlist.append(i)  
        return(factorlist)  
  
        def prime(n):  
            return(factors(n) == [1,n])  
  
        primelist = []  
        for i in range(1,101):  
            if (prime(i)):  
                primelist.append(i)  
  
        print(primelist)  
  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 6  
7, 71, 73, 79, 83, 89, 97]
```

The output of the code is a list of prime numbers from 2 to 97, displayed across two lines.

Collaboration

- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code
- Switch between different versions of code



The screenshot shows a Jupyter Notebook interface with the title "PDSA Week 1 Lecture 1 (unsaved changes)". The interface includes a top bar with the Jupyter logo, a "Logout" button, and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu bar, there are tabs for "Trusted" and "Python 3". The notebook content is titled "Compute primes from 1 to 100" and contains three code cells:

```
In [1]: def factors(n):  
        factorlist = []  
        for i in range(1,n+1):  
            if n%i == 0:  
                factorlist.append(i)  
        return(factorlist)
```

```
In [3]: def prime(n):  
        return(factors(n) == [1,n])
```

```
In [2]: def prime(n):  
        return(len(factors(n)) == 2)
```

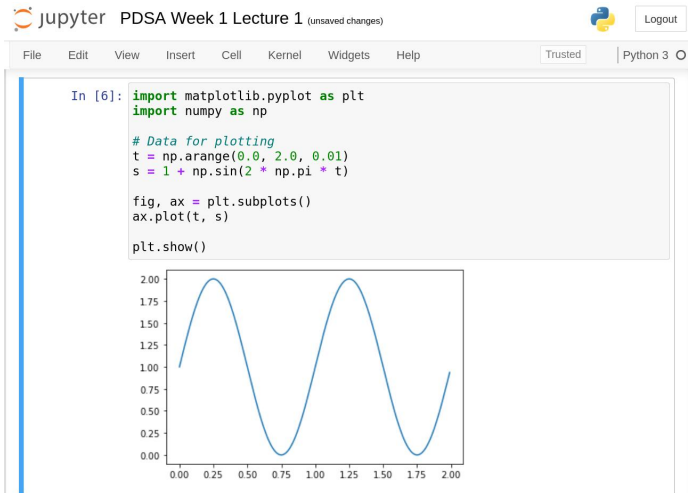
```
In [4]: primelist = []  
        for i in range(1,101):  
            if (prime(i)):  
                primelist.append(i)  
  
        print(primelist)
```

The output of the third cell is displayed below the code:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

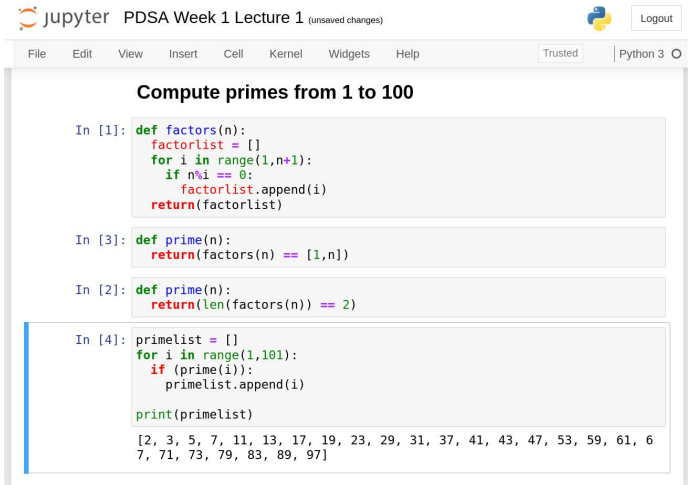
Collaboration

- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code
- Switch between different versions of code
- Export and import your project
- Preserve your output



Jupyter notebook

- A sequence of cells
 - Like a one dimensional spreadsheet



Jupyter PDSA Week 1 Lecture 1 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Compute primes from 1 to 100

```
In [1]: def factors(n):
        factorlist = []
        for i in range(1,n+1):
            if n%i == 0:
                factorlist.append(i)
        return(factorlist)

In [3]: def prime(n):
        return(factors(n) == [1,n])

In [2]: def prime(n):
        return(len(factors(n)) == 2)

In [4]: primelist = []
        for i in range(1,101):
            if (prime(i)):
                primelist.append(i)

        print(primelist)
```

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

Jupyter notebook

- A sequence of cells
 - Like a one dimensional spreadsheet
- Cells hold code or text
 - Markdown notation for formatting
 - <https://www.markdownguide.org/>

Jupyter PDSA Week 1 Lecture 1 (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Compute primes from 1 to 100

```
In [1]: def factors(n):
        factorlist = []
        for i in range(1,n+1):
            if n%i == 0:
                factorlist.append(i)
        return(factorlist)

In [3]: def prime(n):
        return(factors(n) == [1,n])

In [2]: def prime(n):
        return(len(factors(n)) == 2)

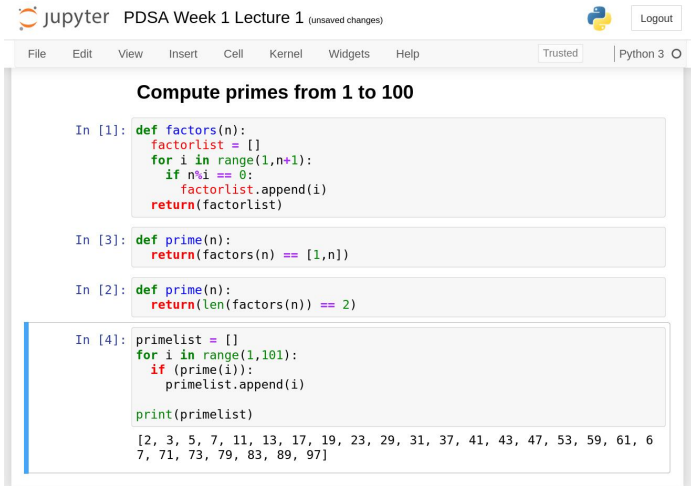
In [4]: primelist = []
        for i in range(1,101):
            if (prime(i)):
                primelist.append(i)

        print(primelist)
```

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

Jupyter notebook

- A sequence of cells
 - Like a one dimensional spreadsheet
- Cells hold code or text
 - Markdown notation for formatting
 - <https://www.markdownguide.org/>
- Edit and re-run individual cells to update environment



The screenshot displays a Jupyter Notebook window titled "PDSA Week 1 Lecture 1 (unsaved changes)". The interface includes a top bar with the Jupyter logo, the title, a "Logout" button, and a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu bar, there are tabs for "Trusted" and "Python 3". The notebook contains four code cells, each starting with "In [X]:". The first cell defines a function `factors(n)` that returns a list of factors for `n`. The second cell defines a function `prime(n)` that returns `[1, n]` if `n` is prime and an empty list otherwise. The third cell defines a function `prime(n)` that returns the number of prime factors of `n`. The fourth cell defines a function `primelist` that generates a list of primes up to 101 and prints it. The output of the fourth cell is displayed below the code: `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]`.

```
def factors(n):
    factorlist = []
    for i in range(1,n+1):
        if n%i == 0:
            factorlist.append(i)
    return(factorlist)

def prime(n):
    return(factors(n) == [1,n])

def prime(n):
    return(len(factors(n)) == 2)

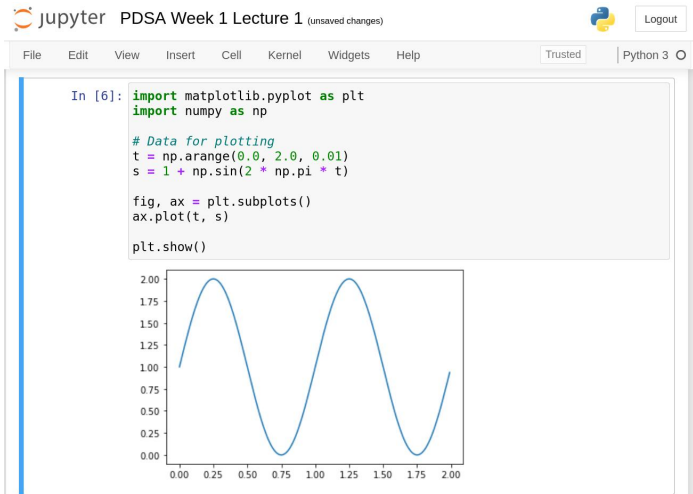
primelist = []
for i in range(1,101):
    if (prime(i)):
        primelist.append(i)

print(primelist)
```

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

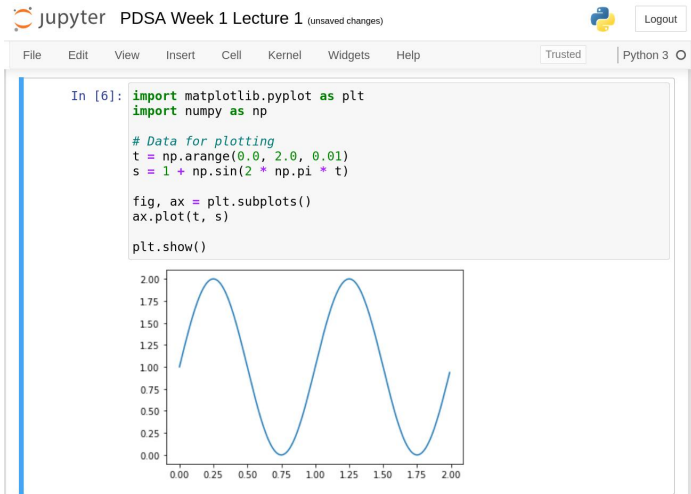
Jupyter notebook ...

- Supports different kernels
 - Julia, Python, R
- We will use it only for Python



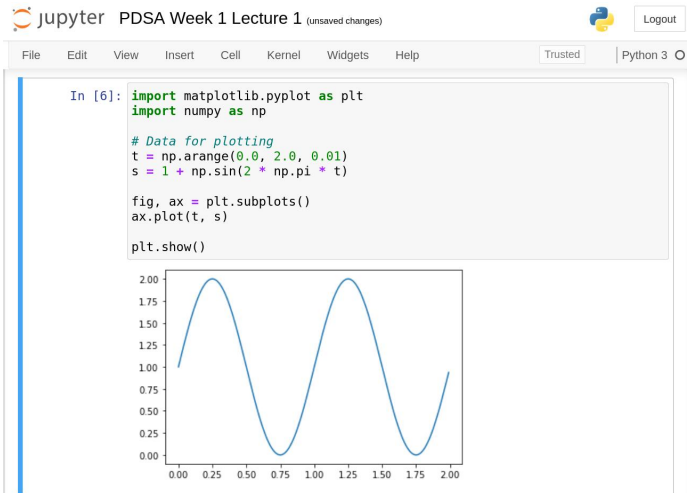
Jupyter notebook ...

- Supports different **kernels**
 - **Julia, Python, R**
 - We will use it only for Python
- Widely used to document and disseminate ML projects
 - Solutions to problems posed on platforms like Kaggle <https://www.kaggle.org>

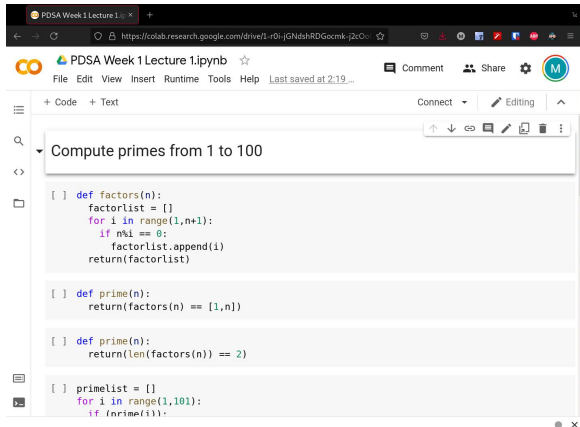


Jupyter notebook ...

- Supports different **kernels**
 - **Julia, Python, R**
 - We will use it only for Python
- Widely used to document and disseminate ML projects
 - Solutions to problems posed on platforms like Kaggle <https://www.kaggle.org>
- ACM Software Systems Award 2017



- Google Colaboratory (Colab)
 - `colab.research.google.com`
 - Free to use



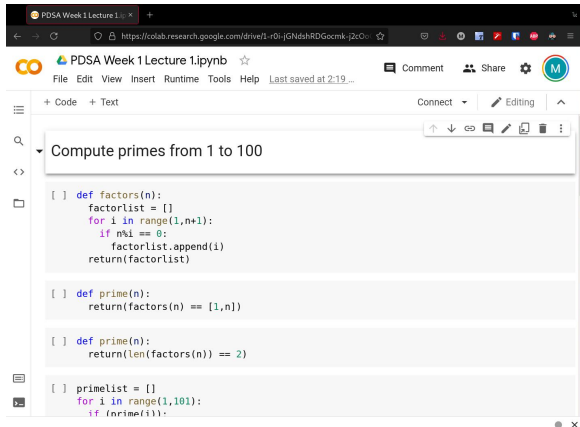
```
[ ] def factors(n):
    factorlist = []
    for i in range(1,n+1):
        if n%i == 0:
            factorlist.append(i)
    return(factorlist)

[ ] def prime(n):
    return(factors(n) == [1,n])

[ ] def prime(n):
    return(len(factors(n)) == 2)

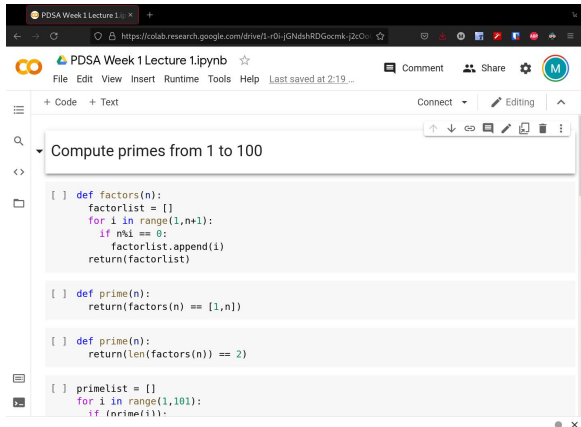
[ ] primelist = []
for i in range(1,101):
    if (prime(i)):
```

- Google Colaboratory (Colab)
 - `colab.research.google.com`
 - Free to use
- Customized Jupyter notebook



```
[ ] def factors(n):  
    factorlist = []  
    for i in range(1,n+1):  
        if n%i == 0:  
            factorlist.append(i)  
    return(factorlist)  
  
[ ] def prime(n):  
    return(factors(n) == [1,n])  
  
[ ] def prime(n):  
    return(len(factors(n)) == 2)  
  
[ ] primelist = []  
    for i in range(1,101):  
        if (prime(i)):
```


- Google Colaboratory (Colab)
 - `colab.research.google.com`
 - Free to use
- Customized Jupyter notebook
- All standard packages required for ML are preloaded
 - `scikit-learn`, `tensorflow`
 - Access to GPU hardware



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL `https://colab.research.google.com/drive/1-r0i-jGNdshRDGocmk-j2cO...`. The notebook title is "PDSA Week 1 Lecture 1.ipynb". The code is written in a Jupyter-style cell with the following Python code:

```
[ ] def factors(n):  
    factorlist = []  
    for i in range(1,n+1):  
        if n%i == 0:  
            factorlist.append(i)  
    return(factorlist)  
  
[ ] def prime(n):  
    return(factors(n) == [1,n])  
  
[ ] def prime(n):  
    return(len(factors(n)) == 2)  
  
[ ] primelist = []  
    for i in range(1,101):  
        if (prime(i)):
```

Summary

- Jupyter notebook is a convenient interface to develop Python code

Summary

- Jupyter notebook is a convenient interface to develop Python code
- Incrementally update and run

Summary

- Jupyter notebook is a convenient interface to develop Python code
- Incrementally update and run
- Embed documentation using Markdown

Summary

- Jupyter notebook is a convenient interface to develop Python code
- Incrementally update and run
- Embed documentation using Markdown
- Preserve outputs when exporting

Summary

- Jupyter notebook is a convenient interface to develop Python code
- Incrementally update and run
- Embed documentation using Markdown
- Preserve outputs when exporting
- Useful for collaboration, sharing

Summary

- Jupyter notebook is a convenient interface to develop Python code
- Incrementally update and run
- Embed documentation using Markdown
- Preserve outputs when exporting
- Useful for collaboration, sharing
- Google Colab — free to use version configured for ML