Data & AI

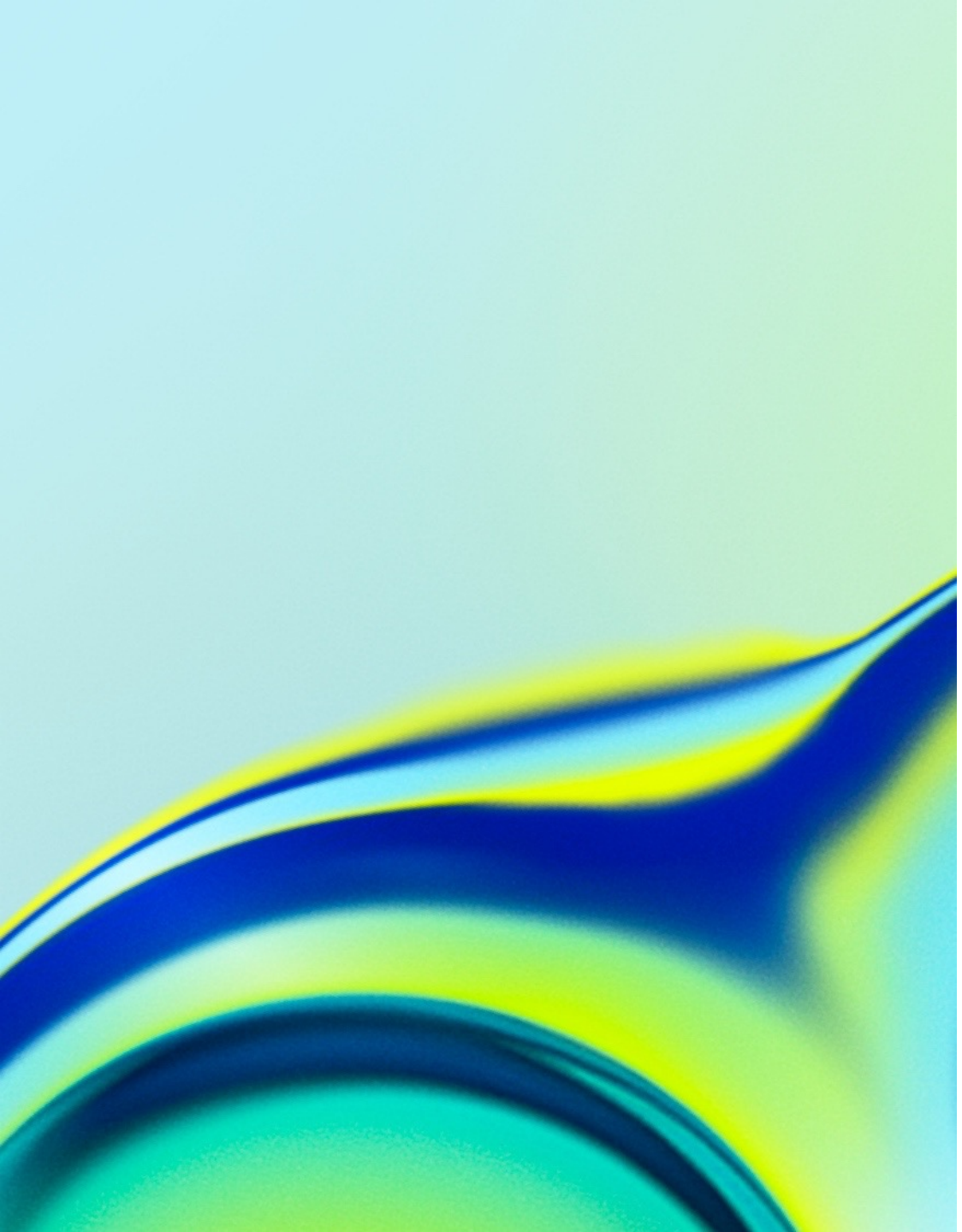
Python Bootcamp

Introduction to Python

Sandy Young

Deloitte.

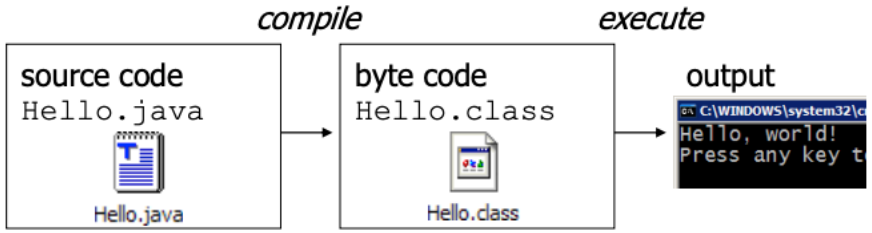
Contents

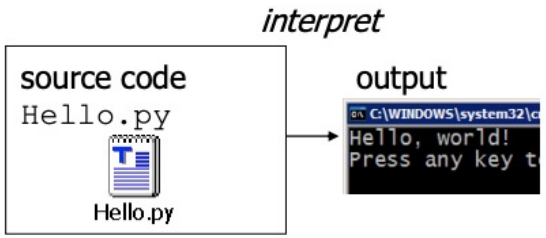


01 Introduction to Python

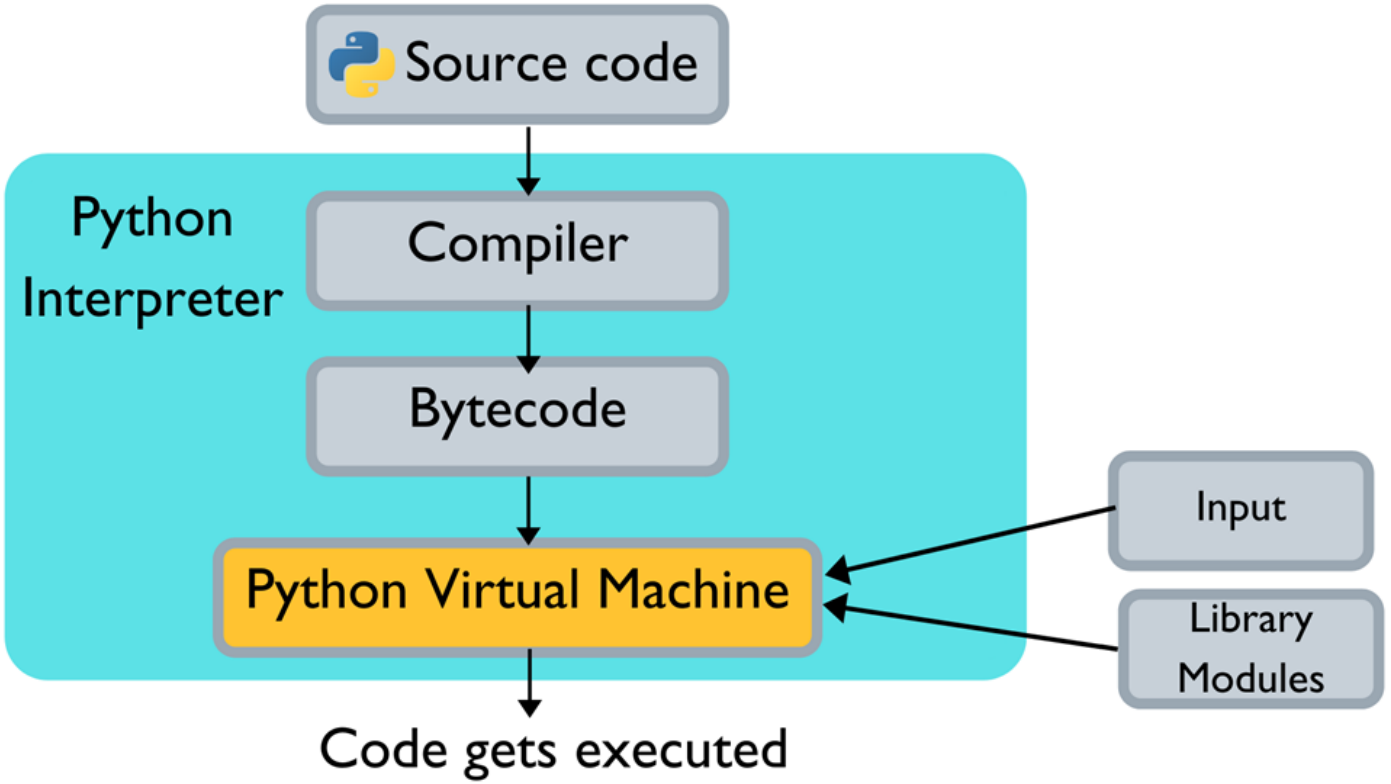
What is Python?

- Created by Guido van Rossum and first released in 1991
- Multi-purpose (Web, GUI, Scripting, etc.), object-oriented
- Cross-platform and compatible with major operating systems (OS)
- Focus on readability and productivity
- High-level, interpreted programming language





How does it work?



The Zen of Python

```
import this
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```


Beautiful is better than ugly.

```
# Non-Pythonic code
def add_numbers(a, b): return a + b

# Pythonic code
def add_numbers(a, b):
    return a + b
```


Flat is better than nested.

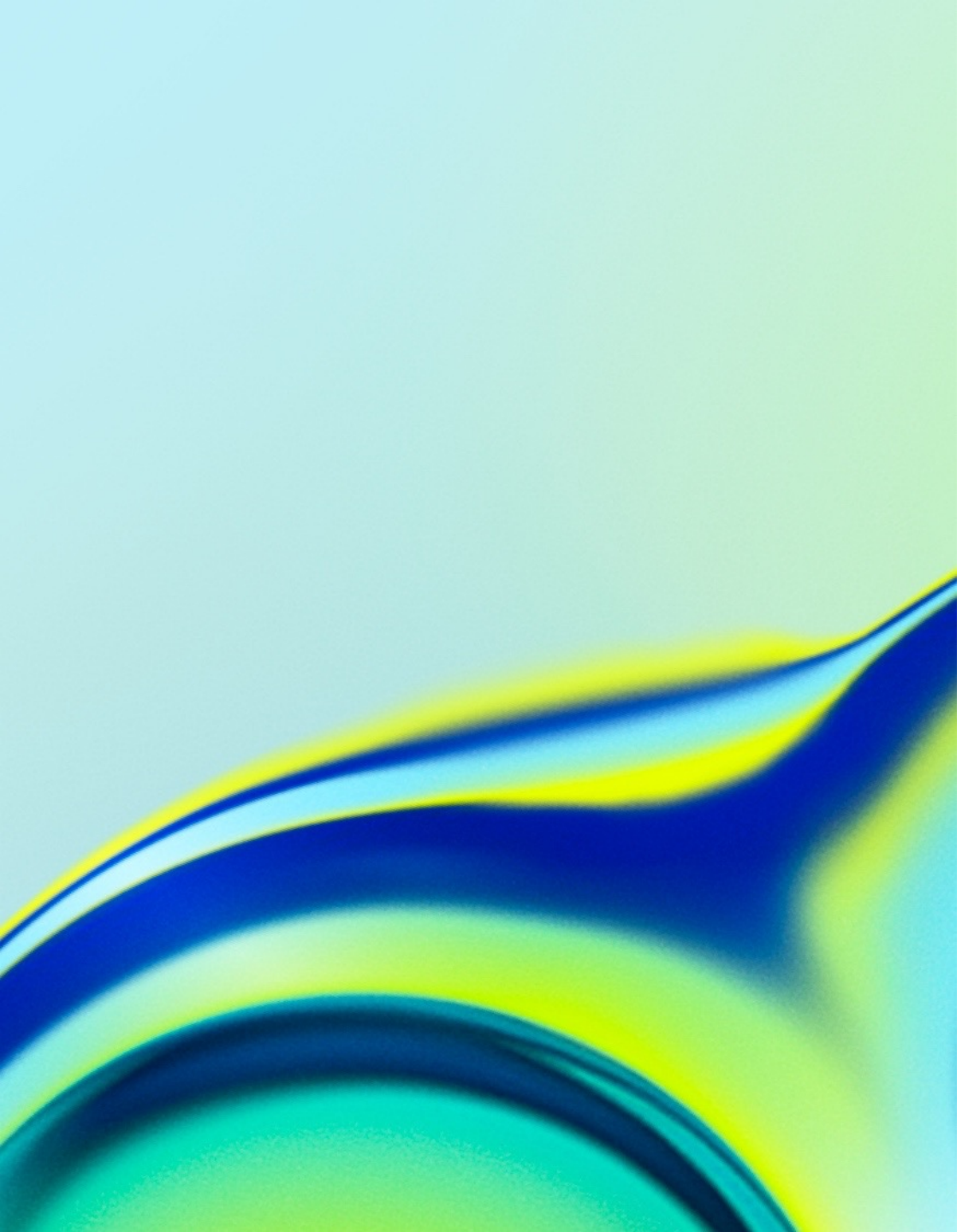
```python
# Nested
def process_data(data):
    result = []
    for item in data:
        if item > 0:
            if item % 2 == 0:
                result.append(item)
    return result

# Flat
def process_data(data):
    result = []
    for item in data:
        if item > 0 and item % 2 == 0:
            result.append(item)
    return result
```


Namespaces are one honking great idea - let’s do more of those!

```
# Without namespaces
def calculate_square_area(side_length):
    return side_length * side_length

def calculate_circle_area(radius):
    return 3.14 * radius * radius

square_area = calculate_square_area(4)
circle_area = calculate_circle_area(3)

# With namespaces
class Geometry:
    @staticmethod
    def calculate_square_area(side_length):
        return side_length * side_length

    @staticmethod
    def calculate_circle_area(radius):
        return 3.14 * radius * radius

square_area = Geometry.calculate_square_area(4)
circle_area = Geometry.calculate_circle_area(3)
```

Why Python?

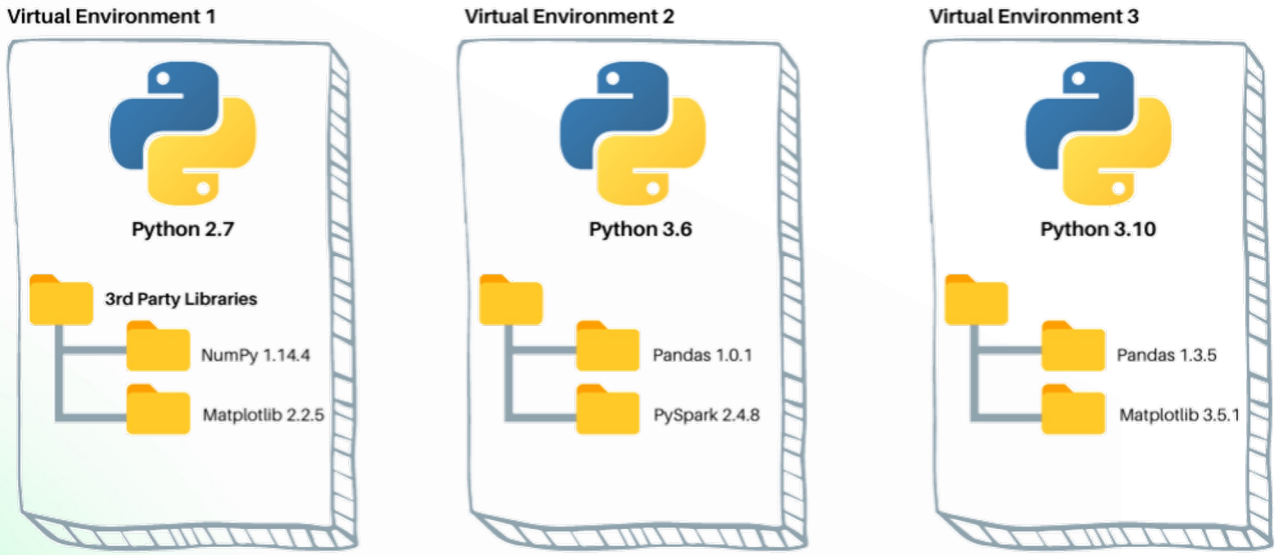
- Versatile language for diverse applications
- Extensive standard library and ecosystem of third-party packages
- Promotes clean and maintainable code
- Widely integrated with cloud platforms such as AWS, GCP and Azure
- Python libraries for data science:
  - **Data wrangling** - Pandas, GeoPandas, NumPy
  - **Machine learning** - Scikit-Learn
  - **Deep learning** - Tensorflow, Pytorch, OpenCV, Hugging Face
  - **Big data processing** - PySpark
  - **Data visualisation** - Matplotlib, Seaborn
- However, not as good for:
  - Super fast, real-time functionality
  - Specific tasks like RPA (robotic process automation)



02 Virtual Environments

What are Virtual Environments?

- An isolated workspace that allows developers to manage and isolate dependencies for different Python projects
- Prevents conflicts and ensures consistent and reproducible dev environments
- Two essential components:
  - Python interpreter
  - Folder containing third party libraries



Creating a Virtual Environment - macOS

1. Create a folder containing your Python project

```
mkdir <python-project>
```

2. cd into the Python project directory

```
cd path/to/your/directory
```

3. Install `venv` to your host Python

```
pip3 install virtualenv
```

4. Use the `venv` command to create a virtual environment inside the project folder

```
python3 -m venv <env-name>
```

5. Activate the virtual environment

```
source <env-name>/bin/activate
```

6. To deactivate, run the below command

```
deactivate
```

Creating a Virtual Environment - Windows

1. Create a folder containing your Python project

```
mkdir <python-project>
```

2. cd into the Python project directory

```
cd path/to/your/directory
```

3. Install `venv` to your host Python

```
pip install virtualenv
```

4. Use the `venv` command to create a virtual environment inside the project folder

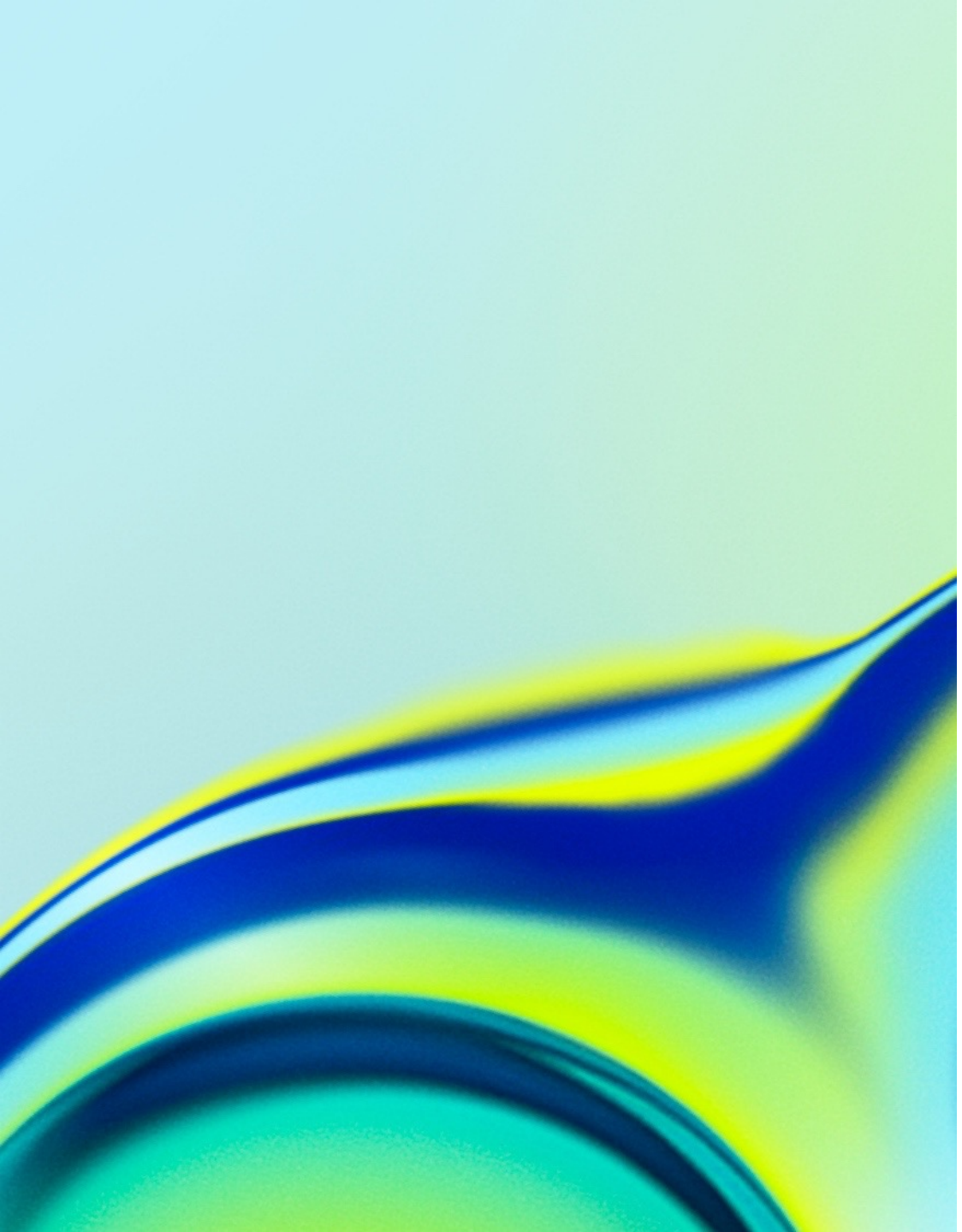
```
python –m venv <env-name>
```

5. Activate the virtual environment

```
.\<env-name>\Scripts\activate
```

6. To deactivate, run the below command

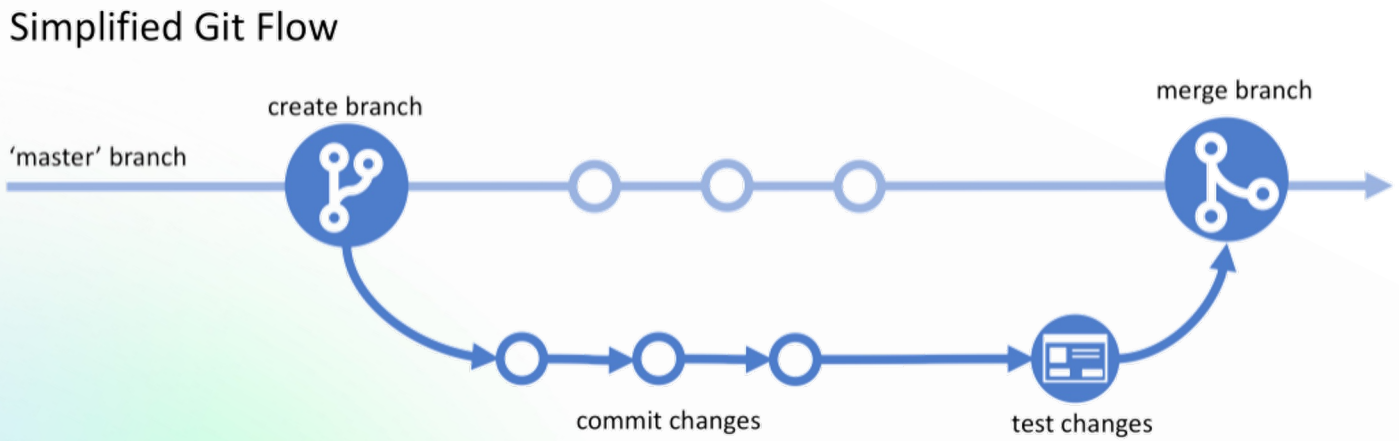
```
deactivate
```

03 Git

What is Git?

- Git is a version control system used for tracking changes in computer files
- Key features and terms:
  - **Repository** - storage location where Git stores and manages files and their revision history
  - **Branch** - a parallel line of development within a repository
  - **Commit** - a snapshot of changes made to files in a repository at a specific point in time
  - **Merge** - the process of combining changes from one branch into another
  - **Origin** – the remote repo from which a local repo was cloned
  - **Remote** - reference to a repo hosted in a different location



Git vs GitHub

Git

- Git operates primarily on your local computer
- Often used through command line interface (CLI)

GitHub

- GitHub is a platform that provides hosting for Git repositories
- Adds a layer of collaboration, project management, and additional tools on top of Git
- Facilitates collaboration among developers

Using Git

Cloning a repository

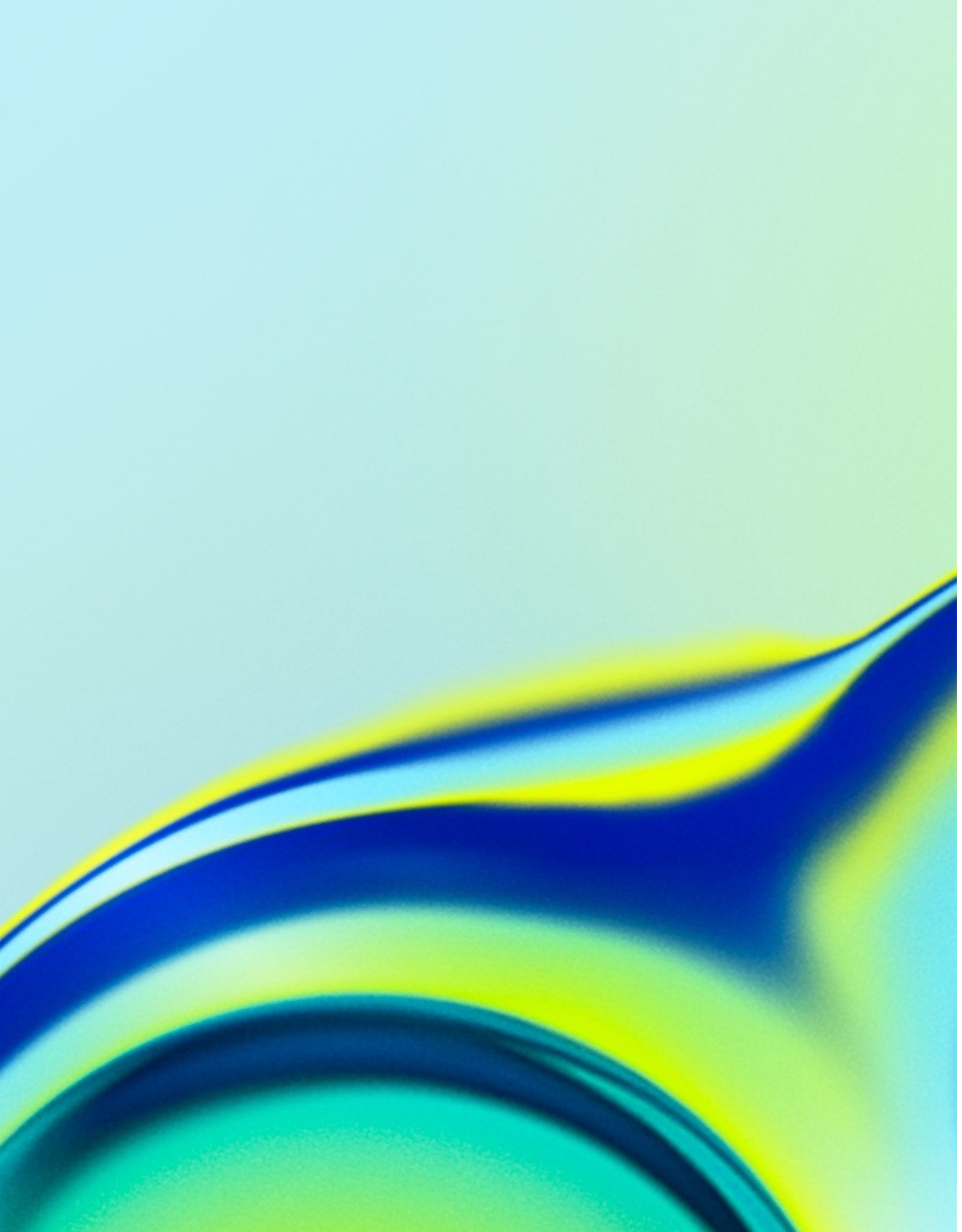
1. Navigate to the URL of the repository
2. Open your terminal and change the directory (**cd**) to where you want to clone the repo
3. Type **git clone** and past the URL you copied

Creating a branch

1. cd into your repository
2. Create a new branch using **git checkout -b <branch-name>**
3. To delete your branch, check out of the branch, and use **git branch --delete <branch-name>** or **git branch -D < branch-name>** if your branch hasn't fully merged (i.e., still has changes on it)

Committing changes

1. After making your changes, use **git add** to stage changes or **git add .** to stage all changes to your local repository
2. To stage your changes with a message, use **git commit -m "Your commit message here"**
3. To stage your changes to a remote repository, use **git push -u <remote-name> <branch-name>**
   For example, this may be **git push -u origin my-feature-branch**



04 Beginner Tips

Tips for Beginners

- Learn the basics first
- Practice regularly – small projects
- Comment your code
- Use descriptive variable names
- Explore the standard library
- Get familiar with version control

Resources

- Official Python documentation
- Stack Overflow
- Online courses from Udemy, Coursera
- Online articles and forums
- YouTube tutorials
- ChatGPT
- Your dev friends and colleagues

