Course Lab Book - AI for the Arts and Humanities (A)

Getting Started

In this course, we are going to learn some Python in order to understand basic programming concepts, and how Python is used in a typical machine learning pipeline.

In a typical computer science programme, you would learn how to write code in one course, and learn to develop your own AI model in another course. In contrast, we are taking a **shortcut**, to focus on how to **read** existing code instead of how to write code, and, to develop skills on how to **present** code to a broader arts and humanities audience to share your interpretation of the model instead of how to develop the models. This will help us reflect more deeply on the more human-centric **social and cultural implications of AI**.

Your objective: to not freak out when you see code and to help other to not freak out when they see code!

Why learn Python?

Python is a highly popular programming language in software development. Statista's 2024 report found 51% of respondents to be using Python, surpassed only by JavaScript (62.3%) and HTML/CSS (52.9%) – way more popular than languages such as Java (30.3%) and C++ (23%). It is a **versatile** language finding extensive use in web development, artificial intelligence, and software engineering.

Python **runs on a wide variety of platforms** including Windows, Mac, Linux, and Raspberry Pi. Its syntax **resembles English**, often requiring fewer lines than languages like Java for the same task. Python is a high level language that you **can quickly execute** to verify functionality. This makes it **ideal for a broad audience** and user base wanting to learn interactively about programming, machine learning, and AI.

> **Did you know?** The name Python comes from Monty Python. When Guido van Rossum was creating Python, he was also reading the scripts from BBC's *Monty Python's Flying Circus*. He thought the name Python was appropriately short and slightly mysterious.

Writing vs Reading Code

This course is not a computing science course. You will not become a proficient programmer or AI developer by taking this course. It will take much more than a semester-long course to achieve that. Instead, we will focus on learning to **read**

existing code and to critically reflect on how it works and their limitation. To achieve this, you will get help in the form of **AI-assisted programming.**

Your objective: Interact with a conversational AI to acquire an **understanding of basic programming concepts** such as printing things onto the screen, defining functions, and handling different data types (e.g. text, image, sound).

Coding Environments for AI

Tools for the Course

For the course, we will be using the resources made available to us by GitHub, GitHub Codespaces and Microsoft Copilot. These resource will allow you to host, share, manage and write code with the assistance of AI and present all the content you produce as part of the course**. GitHub Codespaces is limited to provide each Free plan account, each month, 60 hours of run time for a 2 core codespace, plus 15 GB of storage.** So you should not wait until the last month of the course to cram all your portfolio work into the final part of the course schedule. Planning will be necessary – a transferable skill in any profession!

Tools of the trade and profession (for information only)

If your goal is to become a serious data scientist, you will need to go beyond the tools of the course to set up an environment that supports your development workflow locally and remotely. This could involve setting up something like Anaconda 3 - a popular open-source distribution of Python and R programming languages designed specifically for data science, machine learning, artificial intelligence (AI), and scientific computing. While something like Anaconda simplifies the management and deployment of data science projects by providing a comprehensive ecosystem of tools and libraries, getting to grips with how to set it up and use it can be overwhelming. Nevertheless, it supports integration with other popular Integrated Development Environments (IDE), like Jupyter Notebook, PyCharm, and VS Code, so if you are serious about programming, check it out! It is already installed on the University Glasgow Anywhere remote desktop (see section below on this topic).

GitHub Codespaces and Jupyter Notebook

GitHub Codespaces will allow us to access the essential features of **Jupyter Notebook**. The features of open source Jupyter Notebook from the Jupyter Project is attuned to the objectives of this course. It combines human-readable blocks (known as markdown cells) with coding blocks, allowing you to “present” code. It also handles each block independently and interactively, so that, for example, each block can be run independently for outputs, allowing you to showcase different stages of the code and their outputs, as well as, providing the final output.

University resources and Anaconda and/or Jupyter Notebook

If you want to go further than GitHub Codespaces, you can find Anaconda and Jupyter Notebook already be installed on the Glasgow Anywhere remote Student Desktop for your use. You also have the option to use of them to supplement your work on GitHub. An optional extra resource has been made available in the Moodle Lab resources section **Working with Anaconda**, should you want to explore this option.

Quick Start Guide to GitHub

What is GitHub?

GitHub is a cloud-based platform designed for version control and collaboration. Among other things, it allows developers to manage their code repositories and collaborate on software development projects. GitHub provides a user-friendly interface and a set of tools to facilitate workflows. GitHub hosts millions of open-source and private projects, making it one of the most popular platforms for developers and organizations worldwide. According to 6sense, at the time of checking, it has been estimated that GitHub has a market share of over 87% in the Source Code Management Category.

Step 1: Why GitHub?

Goal of this step: to get a feel for what GitHub is.

GitHub is built on **Git**. Git is an open-source distributed version control system developed by Linus Torvalds.

1. Watch the **shot video** "Why use Git?" from Net Ninja to understand what it means for Git to be a distributed version control system, and to understand why it is useful. This video is about Git but conveys the ethos behind GitHub.
2. Key points to grasp while watching the video:
    - GitHub is a platform for version control and collaboration enhanced with the ability for you to host, share, and manage code from a web interface.
    - It integrates Git, a version control system, to track changes in projects.

Step 2: Create a GitHub Account

Goal: Set up your GitHub account.

1. Go to GitHub.com and click "Sign up."
2. Choose an email – **you can use your personal email if you want to be able to access your account after you leave University of Glasgow**.

3. Choose a password and username and click continue.
4. You will be asked to prove you are a human, and/or verify your email address. Follow the instructions to complete this.
5. Spend a minute exploring the GitHub dashboard and settings after logging in.

Step 3: Understand the Basics of Repositories

Goal: Learn about repositories (repos) and how to work with them.

1. Read the page Hello World on GitHub which gives a very good overview of how to create and manage your first repository. Follow the steps to create a repository on the GitHub account.
2. Once you have familiarised yourself with the steps, modify necessary elements, where appropriate, to:
    - Create a new course repository called `ai-for-the-arts`.
    - Make sure you tick the box to include a **README file** in your repository.
    - Read the information at choosealicense.com to select an appropriate license. If you are unsure, just for the course, choose MIT license.
    - Create a **branch** `readme-edit` and edit the README file to write about the purpose of repository, and what your goals are for the end of the course. Don't worry about formatting and style – we will learn more about that in a moment.
    - Create a **pull request** and **merge** the changes you made in the `readme-edit` branch to be integrated into the `main` branch of the repository.
3. Browse a few popular repositories on GitHub in general to see how others have structured their repository.

Step 5: Style your README File (With Markdown Basics)

Goal: Practice editing files in GitHub and learn basic Markdown syntax for editing README files and notebooks.

Step 5-1: Learn Markdown Basics

Markdown is a lightweight language used for formatting text in GitHub README files. It is also used in editing markdown cells in Jupyter Notebooks more generally. The same syntax can be used in Google Colab notebook markdown cells.

If you are familiar with HTML, then you might be interested to know that almost everything you learned in HTML can be used for editing the README.md file and Jupyter Notebook markdown cells.

1. Refer to the markdown syntax primer at https://www.markdownguide.org/basic-syntax/.
2. **Key points:** how do you make headings, bullet points, and emphasise text?

Step 5-2: Edit Your README File (3 Minutes)

1. Go to your repository and open the README.md file.
2. Click the pencil icon (i.e. edit button) to make changes.
3. Use the markdown syntax to add:
    - A title heading (e.g., AI for the Arts and Humanities Portfolio).
    - A short paragraph describing the purpose of your repository.
    - A bulleted list of goals (e.g., - Learn GitHub basics).

Step 5-3: Save Your Changes

1. Click **Commit changes** – usually found upper righthand corner of the editing window.
2. A box will appear asking you to write a short description of your changes (e.g., "Added project description and goals"). **This step is important** because it helps you and the system keep track of the changes you make in your repository. For example, it would help if you decide to roll back your repository to an earlier version.

Step 6: Uploading Content to Your Repository

Later in GitHub Codespaces you will be able to upload files. But it is useful to be able to upload from the repository itself. This can help you save the time you spend on GitHub Codespaces – remember you have a limited time allowance every month on Github Codespaces.

- **Navigate to the Repository**:
    1. Go to GitHub.
    2. Log in and navigate to your repository or create a new one.
- **Upload Files**:
    1. Click the **Add file** button.

2. Select **Upload files**.
3. Drag and drop files or use the **Choose your files** button to select files from your computer.
4. Add a commit message describing the changes – for example, this could be the names of files you added.
5. Click **Commit changes** to upload the files. **Remember that it could take a little time for these changes to take effect.**

Step 7: Remove Content from Your Repository

- **Navigate to the Repository**:
    1. Open the file or folder you want to delete in the repository.
- **Delete the File**:
    1. Click on the file.
    2. Click the **three dots** in the upper right corner and select **delete file.**
    3. Add a commit message describing the deletion.
    4. Click **Commit changes** to confirm.

Step 8: Checking the Status of Your Repository for Changes

Navigate to Your Repository:

- Go to GitHub and log in (if you are not logged in already).
- Open your repository (if you are not already in your repository).

Check the Commit History:

1. Click on **Commits** in the repository (quite small in the window – usually it will also have a number of commits you made, for example "12 cimmits").
2. Verify that your recent changes are listed with commit messages.
3. Note that you can also browse the repository at the time of each commit point by clicking on the pointy brackets at the far right. When you do this you will see the ID of that commit in the repository (top left of the repository window).
4. You can create new branches from a selected commit point to have multiple versions of your repository from the commit point – **just don't get crazy with tens of versions and lose track to get confused about the many versions you create!**

Best Practices

- **Commit Messages**: Always provide meaningful commit messages to describe your changes.
- **Backup**: Before deleting or making significant changes, ensure you have a backup or are working in a branch.
- **Use Branches**: Work in branches for development and merge them into the main branch once changes are finalized.

Introducing GitHub Codespaces

Step 1: Introduction to GitHub Codespaces

Goal: Understand what GitHub Codespaces is and why it's useful.

1. **What is GitHub Codespaces?**
    - A cloud-based development environment provided by GitHub.
    - No need to install Python or Jupyter Notebook on your computer—it runs directly in the browser.
    - You can save changes to synchronise the work you do on your repository with that on your codespace.
2. **Why use Codespaces?**
    - Easy setup for coding.
    - Perfect for learning Python with tools like Jupyter Notebook.

Step 2: Preparing Your Repository

Goal: Create a GitHub repository to use with Codespaces. You have already done this!

1. Go to GitHub and log in.
2. Navigate to the GitHub Repository (e.g. `ai-for-the-arts`) that you created in the earlier part of this Course Lab Book.
3. Your repository is already ready to be used with Codespaces.

Step 3: Starting a Codespace for your repository

Goal: Open your repository in GitHub Codespaces.

1. In your repository, click the **Code** button (green button at the top right).
2. Select the **Codespaces** tab and select **Create Codespace** (if you have created branches then you can create to specify a branch or it be on the main branch).
    - This will open a development environment in your browser.
3. Wait for the Codespace to initialize—**this may take a minute, so be patient!**
4. **Note down the name automatically assigned to your code space** (e.g. I have created one named "orange fornight") so that you know which one to open if you should close your codespace and come back at a later point to open it.

Step 4: Starting a Jupyter Notebook file

Goal: Open a Jupyter Notebook file in the Codespace.

1. Inside the Codespace:
    - Look at the **Explorer panel** on the left (shows your files).
    - Click **+ File** and name the new file `coding_exercises.ipynb`.
    - The `.ipynb` extension is required for Jupyter Notebook files.
2. Click the new file to open it in the Jupyter Notebook editor.

Step 5: Adding a Markdown Cell to your notebook

Goal: To understand how to add markdown cells and to edit them – you will use these to explain the context of your notebook to engage a broader audience.

1. In your codespace, if you click on the file you created `coding_exercises.ipynb`, you will see: `+ Code` and `+ Markdown`.
2. To create a markdown cell, click on `+ Markdown`
3. Use what you learned about markdown cells to create a title and text to discuss what this notebook is for (e.g. learning basic programming concepts), and a few bullet points on what you would like to achieve (e.g. specific tasks you always wanted to do with programming).

Step 6: Writing a "Hello, World!" Program

Goal: Write and run your first Python program in Jupyter Notebook.

1. Create a Markdown Cell - click + Markdown. Describe the purpose of this section – for example, first heading could be **Hello, World!** followed by something like "The following code was created as my first attempt at

programming in this course. It is meant to help me understand how to write and run code in GitHub Codespaces."

2. Create a Coding Cell - if you click `+ Code` it will start a coding cell. You can start to write your first Python code to print a string (explained further in the next step and in later parts of the Course Lab Book).
3. In the cell created, type:

```
print("Hello, World!")
```

4. Run the cell:
    - Click the **Run** button (triangle icon) at the top of the notebook.
    - You should see the output after your cell: Hello, World!.

The phrase "Hello, World" is a **data type** called a **string** in Python programming. Strings are always surrounded by quotation marks. In fact, "2" is a string. If you want the integer value 2, then you would be expected to ommit the quotation marks. We will learn more about **data types** later.

Step 7: Saving Your Work

Goal: Save changes you made in the Codespace to your repository.

1. Once you're happy with your code, save your work:
    - Click **Source Control** in the left-hand menu (icon looks like a tree branch).
    - Enter a commit message, e.g., "Added conding_exercises.ipynb."
    - Click **Commit** and then click **Synch Changes** to save the changes to your GitHub repository – **remember that what you see in Codespaces is not always what is archived in your repertory** until you go through Commit and Synch Changes.
    - **Unless you do this last step the changes will not be kept when you next use Codespaces**.
2. Verify your changes:
    - Before leaving Codespaces, go back to your repository (not your Codespace) on GitHub.com.
    - You should see the coding_exercises.ipynb file in the repository. Click on the file to verify the content.

Step 8: Stopping Your Codespace

Goal: Properly close your Codespace.

Remember that you only have 60 hours a month on GitHub codespaces!! – even less if you choose to have more processing power for your Codespace. By stopping your Codespace, you can save your resources.

1. In your Codespace, click the hamburger menu on the top lefthand corner (looks like three lines stacked on top of each other) and select **My Codespaces.** This should open a list of all your codespaces and it should say which ones are currently **active.**
2. Click on the three dots next to any codespace. If it is currently active, it will show an option to select **Stop Codespace**. Select this to pause your Codespace.
3. Your Codespace will be saved, and you can resume later from where you left off.

Always remember to stop your Codespace to save on your resources. Only have it active when necessary for your coding practice!!

It also helps the environment to not have everything running all the time!

Using Microsoft Copilot as an AI-Assistant

Step 1: What is Microsoft Copilot?

Goal: Understand what MIcrosoft Copilot is and why it's helpful.

1. **What is GitHub Copilot, Microsoft Copilot and what is the difference?**

   Microsoft Copilot is an AI-powered assistant developed by Microsoft to enhance user productivity across its suite of applications and services. Leveraging advanced large language models (LLMs), such as OpenAI's GPT-4, Copilot integrates seamlessly into tools like Word, Excel, PowerPoint, Outlook, and Teams, providing real-time assistance to streamline tasks and improve efficiency.

   Microsoft has also extended Copilot's capabilities to other areas. For instance, GitHub Copilot assists developers by providing code suggestions and automating repetitive programming tasks. GitHub Copilot should not be confused with Microsoft Copilot. GitHub Copilot is now included free with your GitHub account and can be integrated into Codespaces, but it is limited to 2,000 code completions and 50 chat

messages per month and, if you use it this way, it can deplete your Codespace allowance. Instead, **in this course we will use Microsoft Copilot** as an assistant for coding – the enterprise version of Microsoft Copilot is available to members of the University of Glasgow, and will give you more resources than GitHub Copilot. It also saves depleting your limited GitHub resources and has added benefits.

2. **Why use Copilot at all?**
    - It's **perfect for beginners** as it helps you learn by suggesting solutions and examples.
    - It will speed up the learning process - help you **learn Python in a hurry.**
    - **Saves time** by automating repetitive tasks and providing insights on coding patterns, leaving you time to work on the fun creative part!

Step 2: Access Microsoft Copilot

Goal: Access the web version of Microsoft Copilot.

1. **Log into Microsoft Copilot**:
    - Navigate to Microsoft Copilot and click the **Sign in** button – might need to click it again in the window that appears.
    - If you are already logged into your University account on the computer, you will likely be directly asked whether you want to log in with that account.
    - If you are not yet logged into the University account, select "Sign-in options" (right at the bottom) and then "Sign in to an organization".
    - Provide the domain name "gla.ac.uk" and click **Next**. Now sign in as normal as you would using your University credentials for any resource at the University.
    - Choose **Work** (not Personal) for the enterprise account.
2. **Test it with a prompt**:
    - **Submit the following prompt to Copilot:** "Could you suggest a python code for 'Hello, world!' to demonstrate the concept of printing to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects."

- Response will look something like the following. Copilot will also suggest follow-on prompts to try at the end of the response. Feel free to try those as well.

Absolutely! Here's a simple Python code to print "Hello, world!" and a step-by-step explanation:

```
print("Hello, world!")
```

Step-by-Step Explanation:

1. **print**: This is a built-in Python function used to display output. Functions are reusable pieces of code that perform a specific task. In this case, print is used to output text to the screen.
2. **("Hello, world!")**: The text inside the parentheses is called a string. A string is a sequence of characters enclosed in quotation marks. Here, "Hello, world!" is the string we want to print.
3. **Quotation Marks**: In Python, strings can be enclosed in either single (') or double (") quotation marks. Both work the same way, but it's important to be consistent.
4. **Parentheses ()**: These are used to pass arguments to functions. In this case, the string "Hello, world!" is the argument passed to the print function.

When you run this code, Python executes the print function, which takes the string "Hello, world!" and displays it on the screen.

Key Aspects:

- **Simplicity**: This code is very simple and is often the first program written by beginners to understand the basic syntax of a programming language.
- **Function Call**: Understanding how to call functions and pass arguments is fundamental in programming.
- **Strings**: Learning how to work with strings is essential since they are used frequently in programming.

Feel free to try running this code in a Python environment to see the output! If you have any more questions or need further clarification, just let me know. 😊

Step 3: Using Microsoft Copilot with Codespaces

Goal: Familiarise yourself with navigating between your GitHub Codespace and Microsoft Copilot

1. Leave the Microsoft Copilot page open.
2. Open a another tab in your browser to log into GitHub.
3. Navigate to the repository you created earlier in this Course Lab Book.
4. Start the codespace for the repository and navigate to your file `coding_exercises.ipynb`.
5. Present your work in your notebook.
    a) Create a markdown cell (click `+ Markdown`). Create a heading (for example, Printing). Copy the prompt you submitted to Copilot. Describe what your key learning outcomes were.
    b) Create a coding cell (click `+ Code`). Copy and paste the code(s) suggested by Copilot into a Code cell into your notebook on your Codespace. Run it (click the triangle at the top) to see if it works. If it does not work, copy the error message you get in codespace and the code and submit it to Copilot to ask it to trouble shoot the code with "step-by-step" explanation.
    c) Create markdown cells and coding cells to repeat the above two steps for any follow-on prompts you tried from the prompts that Copilots suggested at the end of your last prompt.
6. **Remember that the objective is to understand the concepts and the code - why or why not code works – not to have code that works in your notebook!**
7. **Stop the Codespace after your exercise session** (as explained earlier in the Lab Book) to ensure no unnecessary usage depletes your allowance.

Always remember to stop your Codespace to save on your resources. Only have it active when necessary for your coding practice!!

It also helps the environment to not have everything running all the time!

Learning Basic Programming Concepts – Python in a Hurry with GitHub Codespaces and GitHub Copilot

You've already written your first Python code using the concept of printing. Pat yourself on the back! You now know how to print a variety of texts on the screen - one of the most important parts of programming! Printing allows the computer to communicate to you. A key aspect of troubleshooting, for example – if it cannot talk to you, you have no clue to what it is doing and where it goes wrong.

In this section we will run through some other key concepts in programming using the Hello World example. We will do this by using Copilot. We will start with the earlier Hello World but to introduce concepts other than printing.

Step 1: Run Some Prompts in Microsoft Copilot

Submit the following prompts into Microsoft Copilot one after another.

1. **Variables**

- **Initial Prompt:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **variables** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Prompts suggested by Copilot:** Make a note of prompts suggested by Copilot at the end of its response. Feel free to try them out by clicking on them and submitting them to Copilot.
- **Asking for more explanation:** If anything is unclear in Copilot's response, feel free to ask for alternative examples, elaboration on explanations, and definitions of unfamiliar terms. The key phrase "step-by-step" is useful for getting more information.
- **Make a note of all your progress:** prompts (initial prompt, suggested prompts, your own prompts) and your key learning outcomes which you will need to present in your notebook.

2. **Functions**

- **Initial Prompt:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **functions** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Prompts suggested by Copilot:** Make a note of prompts suggested by Copilot at the end of its response. Feel free to try them out by clicking on them and submitting them to Copilot.

- **Asking for more explanation:** If anything is unclear in Copilot's response, feel free to ask for alternative examples, elaboration on explanations, and definitions of unfamiliar terms. The key phrase "step-by-step" is useful for getting more information.
- **Make a note of all your progress:** prompts (initial prompt, suggested prompts, your own prompts) and your key learning outcomes which you will need to present in your notebook.

3. Other Initial prompts to explore

Apply the same process as that for Variables and Functions described above to explore the initial prompts listed below. Make notes about learning outcomes, as well as the benefits and limitations of using Copilot for learning Python.

- **Initial prompt to understand loops:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **loops** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to understand data structures such as lists:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **lists** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to understand data structures such as dictionaries:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **dictionaries** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to understand data structures such as arrays:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **arrays** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to understand how to import libraries that extend functionality:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **importing libraries** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to understand how to use conditional if, else, elif statements:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concepts of **if**, **elif**, and **else** statements to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.

- **Initial prompt to understand how to use operators in Python:** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of operators to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects. Include discussions about addition and multiplication between strings and numbers.
- **Initial prompt to understand how to use advanced concepts such as classes, objects, and methods (try one term at a time if it is too much to handle at once):** Could you suggest a modification of the Python code for "Hello, world!" to demonstrate the concept of **classes**, **objects** and **methods** to a complete beginner in programming? Please explain the code step-by-step highlighting any key aspects.
- **Initial prompt to learn more about basic Python syntax:** Could you use the examples you have provided above to explain **basic syntax of Python** to a complete beginner in programming? Please explain step-by-step highlighting any key aspects. Please include well known references to Python styling guides.
- **Initial prompt to learn more about best practice in Python to promote principles of simplicity, security and inclusion:** Could you suggest three different Python implementations of the Hello World program to emphasise, respectively, principles of **simplicity**, **security**, and **inclusion**? Please explain step-by-step highlighting any key aspects.

Step 2: Try the Code in Codespace

- Access your GitHub codespace.
- For each programming concept represented by the initial prompts:
    a) Create markdown cells in your file `coding_exercises.ipynb` to create a heading. Add a short piece of text to explain the purpose and to include the prompt.
    b) Create a coding cell, and copy paste the codes in the responses from Copilot to run it.
    c) Do they work? If not, discuss with your peers to figure out why not? What other code examples did your peers get from Copilot. Compare to discuss how each concept has been demonstrated in different ways.
    d) Do the same for prompts other than the initial prompts you explored – for example, prompts suggested by Copilot and or prompts of your own submitted to interrogate Copilot.
    e) Summarise your learning outcomes and your experience learning the Python programming concept using Copilot.

Step 3: Exploring Other Prompts and Concepts

- Submitting the same prompt twice can result in different responses. Feel free to experiment to see if any of the examples provided seem easier to understand than others. Comment on this, if relevant, in your notebook.
- On the Moodle, you will find a link Areas of Python to Explore with Prompts. Have a go formulating prompts similar to the ones above to explore the topic areas listed there!

Step 4: Stop your codespace

Before leaving the lab and before moving away from GitHub stop your codespace – click on the three bar menu at the top lefthand corner and select My Codespaces and stop any active codespaces.

Always remember to stop your Codespace to save on your resources. Only have it active when necessary for your coding practice!!

It also helps the environment to not have everything running all the time!

Resources for Going Further with Python

If you are dead serious about learning Python, check out Learn Python 3 the hard way in the course reading list on the Moodle. There are also less intensive resources such as the W3Schools Python Tutorial and the Kaggle Intro to Programming and Python courses. Also check out Learn Python which is a recommended resource.

The Kaggle courses are especially good as a starting point for supplementing the work you did with the Copilot in this Lab Book.

What are Python Libraries?

You already saw in the last section how libraries can be imported for use in Python code. Python libraries are collections of pre-written code that provide functionality for specific tasks. They help developers avoid "reinventing the wheel" by offering reusable modules for common tasks. Some common libraries:

- **Mathematics and Data Analysis**: numpy, pandas
- **Machine Learning**: scikit-learn, tensorflow, pytorch
- **Web Development**: flask, django
- **Visualization**: matplotlib, seaborn

Libraries are often distributed via **Python Package Index (PyPI)** and can be installed using package managers like `pip`.

Installing Python Libraries on GitHub Codespaces

GitHub Codespaces provides a cloud-based development environment with pre-installed tools but some Python libraries that you want might not be pre-installed.

When you see code that has statements that start with the keyword `import`, check that the named package following the keyword is installed.

If you try to install the packages, you will be told if it is already installed so you might as well try to install the packages. Here's how you can install Python libraries:

1. Access Your Codespace

- Open your GitHub repository.
- Click the **Code** button and select **Open with Codespaces**.
- Launch your Codespace environment.

2. Open the Terminal

In the Codespace environment, locate the terminal at the bottom of the screen. If it's not visible, open it via **View > Terminal**.

3. Use pip to Install Libraries

The pip tool is used to install Python packages. To install a library, use the command:

```
pip install <library_name>
```

For example:

```
pip install numpy pandas
```

In some cases, the named package in the code is not the package that should be named for installation. For example, the package `scikit-learn` - the name for installation – which we will learn about later in the lab book, is named `sklearn` when importing within the Python code.

Fun with YouTube

To finish off this whirlwind introduction to Python, we will learn how to display an embedded YouTube video in Jupyter Notebook as output of code. This is one of the ways you can include multimedia with your code, potentially, to make it more engaging for your audience. The codes provided in this exercise are displayed as an image so you will

not be able to copy and paste the lines – that's right, you need to start your Codespace and type it into your file `coding_exercises.ipynb`!

Task 1-6: A first look at importing library and packages

Create another cell in your notebook. As a first step, you need to get the library/package/module that has the command for displaying videos. The library in this case is called `IPython.display`. **If it is not installed in your codespace, install it** (following the instructions in the last section). To import a specific package from the library you use the key word `from` followed by the name of the library (in our case `IPython.display`) and then, on the same line, the keyword `import` followed by the name of the specific package you want to import (Figure 12). When you use an asterisk after keyword `import`, then this means that you want all the packages, command and/or functions available in the library.

```
In [2]: from IPython.display import *
```

Figure 12. How to import all the commands from the library IPython.display on Jupyter Notebook.

Importing everything is an overkill for our purposes here, but we introduce it so that you get a feel for how it is done. In this course, we will be importing a number of libraries, models etc. So it is important to get used to this process.

If you should get an importing error, or if you are not sure it did the job, compare with your peers to see if it worked for them and compare the code. Remember, Python is case sensitive,

Task 1-7: Find and Display a YouTube Video

- Now use your web browser to search YouTube for a video for which the licence permits you to display it for private/educational purposes. In the example below I have chosen a video of the cartoon character "Bugs Bunny" screaming. Normally, it would be better to choose something related to your message in the previous task. Perhaps a video of hello in a selected language? Feel free to be creative!
- Now simply use the following command to display it in your notebook. The ID in the quotation code below – "05PKG_pWsVY" - is the ID of the Bugs Bunny video. You will need to locate the ID for your own video – this is usually included in the URL. Go ahead and embed your chosen YouTube video by using the syntax `YouTubeVideo(video_ID)` , replacing `video_ID` with that provided on YouTube.

```
In [ ]: YouTubeVideo("05PKG_pWsVY")
```

> You can combine this command and the `import` command in one cell to run together in one cell or have separate cells for each command – the choice is yours. **When you are first starting out with coding, separating it can help you to understand where the first error is if your code does not work.**

Once you have run the commands, **discuss it with your peer group**.

If someone is having trouble help each other out.

- See what other people have chosen for their video.

There are also commands in the library to embed other media such as images and audio. Check out the examples at:

https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html

Understanding a Program: Parsing and Reading

So, You've learned about importing libraries, variables, and printing. These are what I think of as the three most fundamental concepts of programming. The following program takes as input a site URL, and a date, and searches the Internet Archive for a version closest to the date you specified and displays it in your browser:

```
import webbrowser
import requests


print("Shall we hunt down an old website?")
site = input("Type a website URL: ")
era = input("Type year, month, and date, e.g., 20150613: ")
url =
"http://archive.org/wayback/available?url=%s&timestamp=%s" %
(site, era)
response = requests.get(url)
data = response.json()
try:
    old_site = data["archived_snapshots"]["closest"]["url"]
    print("Found this copy: ", old_site)
    print("It should appear in your browser.")
    webbrowser.open(old_site)
```

```
except:
    print("Sorry, could not find the site.")
```

Do not expect to understand everything now – it is a bigger program than ones you experienced in the previous section, so can be a lot if you are only starting out with Python! However, you should be able to identify how many libraries were imported, how many variables were used, how many strings appear in the programme, and how many times something was printed? What was printed?

Optional Task 1-8: Run the code

- Create a markdown cell, titled and described to be appropriate for the program you are exploring.
- Create a code cell and type the code above into your Python notebook. **The indents are important and part of the python syntax!** Use 4 spaces per indentation level. This is what the PEP 8 Style Guide for Python recommends.
- Once you have typed it into your cell, run it to see what it does.

Optional Task 1-9: Annotate each line with a comment

- Python comments are preceded by a hashtag #. These are lines of the code that will be ignored when you run it. Use comments to annotate the code in your notebook to say whether it is printing something, assigning a value to a variable, and/or importing a library.

Hint: there are only three lines that do not fall into any of these categories. Can you which these are and what they are meant to do?

Closing Down Your Codespace

Well done if you made it up to this point! **Don't forget to Stop your Codespace before leaving the lab!**

Always remember to stop your Codespace to save on your resources. Only have it active when necessary for your coding practice!!

It also helps the environment to not have everything running all the time!

Thinking about Media as Data

One of the things that you need to get your head around in Data Science and AI in the Arts and Humanities is how arts and Humanities content (e.g. text, image, audio and video as well as physical objects) can be digitally transformed into data that can be processed by a computer.

For example, those of you who have taken the course 2D digitisation have already experienced a number ways text and images can be digitised. But this is only the first step. There are a number of steps to follow for these to become machine readable. In the case of scanned manuscripts, for example, a popular approach to get started is to apply optical character recognition and/or handwritten text recognition. In the case of scanned photographs, the pixels, colour maps, and filters can be useful. Read my reviews of selected past student projects (both in the course AI for the Arts (B) and postgraduate dissertations) on my blog Forensic Search for Godot. This will help you get a feel for these kinds of steps in action. To highlight in advance: **you will not be expected to do anything this extensive in this course, so don't panic!**

In this section, however, as a taster of how you process media, and as pre-requite for abstract thinking about how media is consumed by machine learning, we will review some basic Python code snippets that process text, images and audio.

Your goal: Apply your Python knowledge to:

- Break these code snippets down into the concepts you explored in your earlier Copilot-assisted coding (for example, printing, variable, functions, loops and more).
- **Do not run the code snippets in Codespace.** This part of the Course Lab Book is to help you familiarise yourself with parsing and interpreting code.
- This will prepare you for the more complex codes in machine learning.
- You are not required to copy these code snippets into your notebook. Instead create a markdown cell to discuss the similarities and differences between these three different media types considered as data.

If you decide that you want to try the code out, you will need to make modifications to the code, for example, to install packages on your Codespace, to replace the names of the text file, image file, and audio file with the name of an actual plain text file, jpeg file, and wav file which you have uploaded to your GitHub repository in the right location.

Make sure you have permission to use any file you use for this.

Upload these files to your GitHub repository - to the same location as your file that has the code, for example, this could be `coding_exercises.ipynb`).

You might also need to install some Python libraries (for example, `librosa`)

Text Processing Example: Word Frequency Counting

Concept: Given a text document, count the frequency of each word and display the most common words.
Key Demonstration:

- Easy access to raw data (plain text is human-readable).
- Simple tokenization and counting operations.
- Demonstrates basic natural language processing tasks (like cleaning and normalizing text).

Example Code:

```
from collections import Counter
import re


def process_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as f:
        text = f.read().lower()
    # Remove punctuation and split into words
    words = re.findall(r'\w+', text)
    word_counts = Counter(words)
    for word, count in word_counts.most_common(10):
        print(f"{word}: {count}")


process_text ("some_text.txt")
```

Image Processing Example: Grayscale Conversion and Basic Feature Extraction

Concept: Load an image, convert it to grayscale, and compute basic statistics such as average intensity.
Key Demonstration:

- Unlike text, images require pixel-level operations and numerical arrays (e.g., NumPy arrays).
- Simple transformations (e.g., grayscale) show how easily images can be manipulated.
- Extracting basic features (average pixel intensity) can be analogous to calculating word frequencies in text—both reduce complex data into simple measurable quantities.

Example Code:

```
from PIL import Image
import numpy as np


def process_image(file_path):
    img = Image.open(file_path).convert("L")  # convert to grayscale
    img_array = np.array(img)
    avg_intensity = np.mean(img_array)
    print(f"Average intensity: {avg_intensity}")
    img.save("processed_image.png")


process_image("sample_image.jpg")
```

Audio Processing Example: Waveform and Frequency Analysis

Concept: Load a short audio file, compute its amplitude envelope or average volume, and optionally compute a frequency spectrum to identify dominant frequencies.
Key Demonstration:

- Like images, audio is numeric data, but it is 1D (or low-dimensional) and time-series based.

- Tools such as pydub or librosa let you load audio and perform transformations.
- Similar to text tokenization or image pixel-level processing, audio processing involves fundamental transformations (time-domain waveform, frequency-domain spectrum) before extracting features.

Example Code (using librosa):

```
import librosa
import numpy as np


def process_audio(file_path):
    # Load audio at a default sample rate
    y, sr = librosa.load(file_path, sr=None)


    # Compute average amplitude (a rough measure of volume)
    avg_amplitude = np.mean(np.abs(y))
    print(f"Average amplitude: {avg_amplitude}")


    # Compute a short-time Fourier transform (STFT) for frequency analysis
    D = np.abs(librosa.stft(y))
    freq_bins = np.mean(D, axis=1)
    dominant_frequency_bin = np.argmax(freq_bins)
    print(f"Dominant frequency bin: {dominant_frequency_bin}")


process_audio("sample_audio.wav")
```

The importance of abstract thinking

The code snippets in the previous section allow you to familiarise yourself with the practice of parsing and interpreting code even without running it! It also gets you started thinking abstractly about how all the sensory data around you can be transformed into machine readable data. You can appreciate the amount of data available for machine learning and AI.

To repeat: **remember to Stop your codespace** if you have activated it in the last section.

Always remember to stop your Codespace to save on your resources. Only have it active when necessary for your coding practice!!

It also helps the environment to not have everything running all the time!

Next to come in this Course Lab Book

More to come in the next weeks to explore:

- Understanding machine learning workflows and AI ethics
- Designing creative projects with large language models