A logo of a company

Description automatically generated

Data Advanced Analytics Google Certification

Course 2 Get Started with Python

**Module 1:**

**Computer programming**

The process of giving instructions to a computer to perform an action or set of actions.

**What you’ll learn**

-Variables

-Data Types

-Functions

-Conditional statements

-Loops

-Strings

-Data structures

**What you’ll learn**

-Collaborate on data projects

-Jupyter Notebooks

-Object-oriented programming

-Variables

-Naming conventions

-Data types

**Helpful resources and tips**

As a learner, you can choose to complete one or multiple courses in this program. However, to obtain the certificate, you must complete all the courses in the program. This reading describes what is required to obtain a scertificate and best practices for you to have a good learning experience on Coursera.

**Obtain the Google Advanced Data Analytics Certificate**

To receive your official Google Advanced Data Analytics Certificate, you must:

* Pass all graded assignments in all 7 courses of the certificate program. Each graded assignment is part of a cumulative graded score for the course, and the passing grade for each course is 80%.

AND **one**of the following:

* Pay the [course certificate fee](https://www.coursera.support/s/article/209818963-Payments-on-Coursera?language=en_US),
* Be approved for [Coursera Financial Aid](https://www.coursera.support/s/article/209819033-Apply-for-Financial-Aid-or-a-Scholarship?language=en_US), **or**
* Complete the certificate through an educational institution, employer, or agency that's sponsoring your participation.

**Healthy habits for course completion**

Here is a list of best practices that will help you complete the courses in the program in a timely manner:

* **Plan your time:** Setting regular study times and following them each week can help you make learning a part of your routine. Use a calendar or timetable to create a schedule, and list what you plan to do each day in order to set achievable goals. Find a space that allows you to focus when you watch the videos, review the readings, and complete the activities.
* **Work at your own pace:** Everyone learns differently, so this program has been designed to let you work at your own pace. Although your personalized deadlines start when you enroll, feel free to progress through the program at the speed that works best for you. There is no penalty for late assignments; to earn your certificate, all you have to do is complete all of the work. You can extend your deadlines at any time by going to **Overview** in the navigation panel and selecting **Switch Sessions**. If you have already missed previous deadlines, select **Reset my deadlines** instead.
* **Be curious:** If you find an idea that gets you excited, act on it! Ask questions, search for more details online, explore the links that interest you, and take notes on your discoveries. The steps you take to support your learning along the way will advance your knowledge, create more opportunities in this high-growth field, and help you qualify for jobs.
* **Take notes:** Notes will help you remember important information in the future, especially as you’re preparing to enter a new job field. In addition, taking notes is an effective way to make connections between topics and gain a better understanding of those topics.
* **Review exemplars:** Exemplars are completed assignments that fully meet an activity's criteria. Many activities in this program have exemplars for you to compare to your own work. Although there are often many ways to complete an assignment, exemplars offer you guidance and inspiration about how to complete the activity.
* **Chat (responsibly) with other learners:** If you have a question, chances are, you’re not alone. Use the [discussion forums](https://www.coursera.org/learn/get-started-with-python/discussions) to ask for help from other learners taking this program. You can also visit Coursera’s [Global Online Community](https://coursera.community/). Other important things to know while learning with others can be found in the [Coursera Honor Code](https://learner.coursera.help/hc/en-us/articles/209818863-Coursera-Honor-Code) and [Code of Conduct](https://learner.coursera.help/hc/en-us/articles/208280036-Coursera-Code-of-Conduct).
* **Update your profile:** Consider [updating your profile](https://www.coursera.org/account/profile) on Coursera to include your photo, career goals, and more. When other learners find you in the discussion forums, they can click on your name to access your profile and get to know you better.

**Documents, spreadsheets, presentations, and labs for course activities**

To complete certain activities in the program, you will need to use digital documents, spreadsheets, presentations, and/or labs. Data analytics professionals use these software applications to collaborate within their teams and organizations. If you need more information about using a particular tool, refer to these resources:

* [Microsoft Word: Help and learning](https://support.microsoft.com/en-us/word): Microsoft Support page for Word
* [Google Docs](https://support.google.com/docs/topic/9046002?hl=en&ref_topic=1382883): Help Center page for Google Docs
* [Microsoft Excel: Help and learning](https://support.microsoft.com/en-us/excel): Microsoft Support page for Excel
* [Google Sheets](https://support.google.com/docs/topic/9054603?hl=en&ref_topic=1382883): Help Center page for Google Sheets
* [Microsoft PowerPoint: Help and learning](https://support.microsoft.com/en-us/powerpoint): Microsoft Support page for PowerPoint
* [How to use Google Slides](https://support.google.com/docs/answer/2763168?hl=en&co=GENIE.Platform%3DDesktop): Help Center page for Google Slides
* [Common problems with labs](https://support.google.com/qwiklabs/answer/9133560?hl=en&ref_topic=9134804): Troubleshooting help for Qwiklabs activities

**Module, course, and certificate glossaries**

This program covers a lot of terms and concepts, some of which you may already know and some of which may be unfamiliar to you. To review terms and help you prepare for graded quizzes, refer to the following glossaries:

* **Module glossaries**: At the end of each module’s content, you can review a glossary of terms from that module. Each module’s glossary builds upon the terms from the previous modules in that course. The module glossaries are not downloadable; however, all of the terms and definitions are included in the course and certificate glossaries, which are downloadable.
* **Course glossaries**: At the end of each course, you can access and download a glossary that covers all of the terms in that course.
* **Certificate glossary**: The certificate glossary includes all of the terms in the entire certificate program and is a helpful resource that you can reference throughout the program or at any time in the future.

You can access and download the certificate glossaries and **save them on your computer.** You can **always find the** course and certificate glossaries through each course’s Resources section. To access the **Advanced Data Analytics Certificate glossary**, click the link below and select *Use Template*.

* Link to the glossary: [Advanced Data Analytics Certificate glossary](https://docs.google.com/document/d/193-AtS7MlB2w4buwiCyPjBoOhIbbByKgHWPpYnSR9VI/template/preview)

OR

* If you don’t have a Google account, you can download the glossary directly from the following attachment.

[Advanced Data Analytics Certificate glossary](https://d3c33hcgiwev3.cloudfront.net/hh5fTi8iRiSd_QFMpbgudg_6afdc847dcb2418a8d6626674ce6a5f1_Advanced-Data-Analytics-Certificate-glossary.docx?Expires=1719878400&Signature=FNclnNKZpv8DFlGkeSrK4qKrPXAo8KobJIO0UwkpRA4W~keDDof~OTBYXeec-e75RfFKPta2XNTFpB5I3NKXOAJe2qeL6HxHeYAAoqtRT77UX65IvQvve1ztew210fkA1KPW51XZyeilFJUAbyS6oxrC-mhgIl6-hKNkU-ZZfH8_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

[DOCX File](https://d3c33hcgiwev3.cloudfront.net/hh5fTi8iRiSd_QFMpbgudg_6afdc847dcb2418a8d6626674ce6a5f1_Advanced-Data-Analytics-Certificate-glossary.docx?Expires=1719878400&Signature=FNclnNKZpv8DFlGkeSrK4qKrPXAo8KobJIO0UwkpRA4W~keDDof~OTBYXeec-e75RfFKPta2XNTFpB5I3NKXOAJe2qeL6HxHeYAAoqtRT77UX65IvQvve1ztew210fkA1KPW51XZyeilFJUAbyS6oxrC-mhgIl6-hKNkU-ZZfH8_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

**Data Analytics Certificate glossary**

If you completed the original [Google Data Analytics Certificate](https://www.coursera.org/professional-certificates/google-data-analytics?utm_source=google&utm_medium=institutions&utm_campaign=gwgsite-gDigital-paidha-sem-bk-gen-exa-glp-br-null&_ga=2.170664992.1625030801.1661901112-1742325342.1661901112), you may recognize some overlap with several of the glossary terms in this program. Refer to the Data Analytics Certificate glossary, linked in the [Resources](https://www.coursera.org/learn/get-started-with-python/resources/E23KB) tab, to review these foundational terms and concepts. The definitions of some terms in the Data Analytics Certificate glossary differ from the definitions of the same terms in this program since the Advanced Data Analytics Certificate builds upon the concepts taught in the previous program.

**Course feedback**

Providing feedback on videos, readings, and other materials is easy. With the resource open in your browser, you can find the thumbs-up and thumbs-down symbols.

* Click **thumbs-up** for materials you find helpful.
* Click **thumbs-down** for materials that you do not find helpful.

If you want to flag a specific issue with an item, click the flag icon, select a category, and enter an explanation in the text box. This feedback goes back to the course development team and isn’t visible to other learners. All feedback received helps to create even better certificate programs in the future.

For technical help, visit the [Learner Help Center](https://learner.coursera.help/hc/en-us).

“There were a number of transferable skills that I learned from my previous career as a nurse: first being critical thinking, second problem solving, and third assessment.”

“Another skill that I was able to bring over from nursing is soft skills or interpersonal skills, which are critical in advanced data analytics as you try to work with others in a collaborative space.”

“Data, and the ability to manipulate and leverage it, is going to be critical.”

**Course 2 overview**

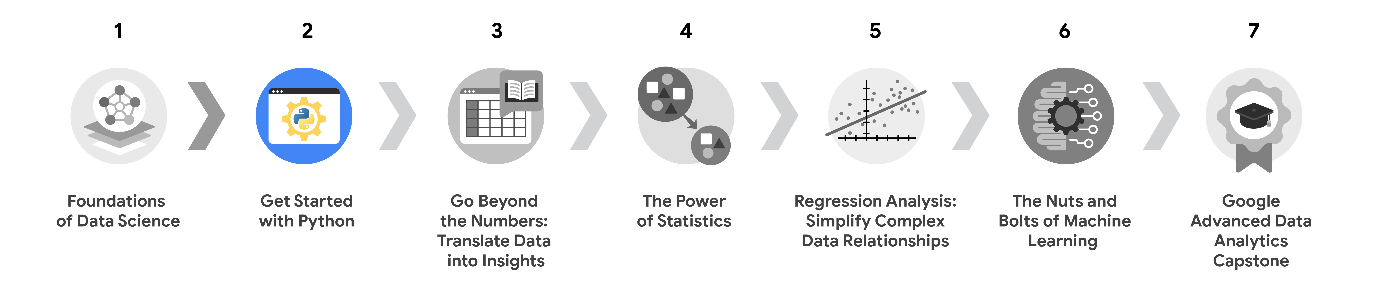


Hello, and welcome to **Get Started with Python**, the second course in the Google Advanced Data Analytics Certificate. You’re on an exciting journey!

Throughout this course, you will develop an understanding of Python syntax, logic, data types, objects, and object-oriented programming. For many professionals, Python is the key to unlocking advanced analytics, machine learning, and the world of data science. By the end of this course, you will better understand how data scientists use programming on the job and how Python will be an important tool throughout your career as a data analytics professional.

**Course descriptions**

The Google Advanced Data Analytics Certificate has seven courses. **Get Started with Python** is the second course.

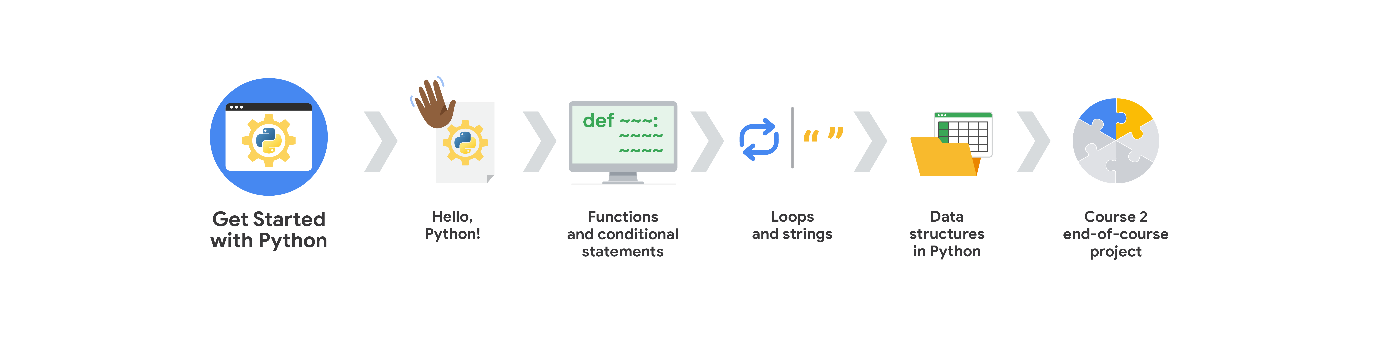


1. [**Foundations of Data Science**](https://www.coursera.org/learn/foundations-of-data-science/home/week/1) — Learnhow data professionals operate in the workplace and how different roles in the field of data science contribute to an organization’s vision of the future. Then, explore data science roles, communication skills, and data ethics.
2. [**Get Started with Python**](https://www.coursera.org/learn/get-started-with-python/home/week/1) —*(current course)* Discover how the programming language Python can power your data analysis. Learn core Python concepts, such as data types, functions, conditional statements, loops, and data structures.
3. [**Go Beyond the Numbers: Translate Data into Insights**](https://www.coursera.org/learn/go-beyond-the-numbers-translate-data-into-insight/home/week/1) — Learn the fundamentals of data cleaning and visualizations and how to reveal the important stories that live within data.
4. [**The Power of Statistics**](https://www.coursera.org/learn/the-power-of-statistics/home/week/1) — Explore descriptive and inferential statistics, basic probability and probability distributions, sampling, confidence intervals, and hypothesis testing.
5. [**Regression Analysis: Simplify Complex Data Relationships**](https://www.coursera.org/learn/regression-analysis-simplify-complex-data-relationships/home/week/1) — Learn to model variable relationships, focusing on linear and logistic regression.
6. [**The Nuts and Bolts of Machine Learning**](https://www.coursera.org/learn/the-nuts-and-bolts-of-machine-learning/home/week/1) — Learn unsupervised machine learning techniques and how to apply them to organizational data.
7. [**Google Advanced Data Analytics Capstone**](https://www.coursera.org/learn/google-advanced-data-analytics-capstone/home/week/1) — Complete a hands-on project designed to demonstrate the skills and competencies you acquire in the program.

**Course 2 content**

Each course of this certificate program is broken into modules. You can complete courses at your own pace, but the module breakdowns are designed to help you finish the entire Google Advanced Data Analytics Certificate in about six months.

What’s to come? Here’s a quick overview of the skills you’ll learn in each module of this course.



**Module 1: Hello, Python!**

You’ll begin by exploring the basics of Python programming and why Python is such a powerful tool for data analysis. You’ll learn about Jupyter Notebooks, an interactive environment for coding and data work. You’ll investigate how to use variables and data types to store and organize your data; and, you'll begin practicing some important coding skills.

**Module 2: Functions and conditional statements**

Next, you’ll discover how to call functions to perform useful actions on your data. You’ll also learn how to write conditional statements to tell the computer how to make decisions based on your instructions. And you’ll practice writing clean code that can be easily understood and reused by other data professionals.

**Module 3: Loops and strings**

After that, you’ll learn how to use iterative statements, or loops, to automate repetitive tasks. You’ll also learn how to manipulate strings using slicing, indexing, and formatting.

**Module 4: Data structures in Python**

Then, you’ll explore fundamental data structures such as lists, tuples, dictionaries, sets, and arrays. Lastly, you’ll learn about two of the most widely used and important Python tools for advanced data analysis: NumPy and pandas.

**Module 5: Course 2 end-of-course project**

At the end of this course, you will put everything you have learned about Python so far into practice with an end-of-course project. You will select a business problem from a list of options and use the given data to solve the problem. This project is an opportunity to demonstrate your skills and build a professional portfolio you can use to showcase your work to potential employers.

**What to expect**

Each course offers many types of learning opportunities:

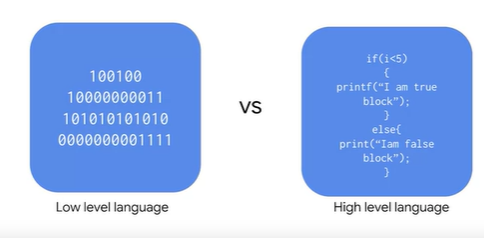
* **Videos** led by Google instructors teach new concepts, introduce the use of relevant tools, offer career support, and provide inspirational personal stories.
* **Readings** build on the topics discussed in the videos, introduce related concepts, share useful resources, and describe case studies.
* **Discussion prompts** explore course topics for better understanding and allow you to chat and exchange ideas with other learners in the [**discussion forums**](https://www.coursera.org/learn/get-started-with-python/discussions).
* **Self-review activities** and **labs** give you hands-on practice in applying the skills you are learning and allow you to assess your own work by comparing it to a completed example.
* **In-video quizzes** help you check your comprehension as you progress through each video.
* **Practice quizzes** allow you to check your understanding of key concepts and provide valuable feedback.
* **Graded quizzes** demonstrate your understanding of the main concepts of a course. You must score 80% or higher on each graded quiz to obtain a certificate, and you can take a graded quiz multiple times to achieve a passing score.

**Tips for success**

* It is strongly recommended that you go through the items in each lesson in the order they appear because new information and concepts build on previous knowledge.
* Participate in all learning opportunities to gain as much knowledge and experience as possible.
* If something is confusing, don’t hesitate to replay a video, review a reading, or repeat a self-review activity.
* Use the additional resources that are referenced in this course. They are designed to support your learning. You can find all of these resources in the [**Resources**](https://www.coursera.org/learn/get-started-with-python/resources/GnHn1) tab.
* When you encounter useful links in this course, bookmark them so you can refer to the information later for study or review.
* Understand and follow the [Coursera Code of Conduct](https://www.coursera.support/s/article/208280036-Coursera-Code-of-Conduct?) to ensure that the learning community remains a welcoming, friendly, and supportive place for all members.

**Programming Languages**

The words and symbols we use to write instructions for computers to follow



**Test your knowledge: Get started with the course**

Question 1

Fill in the blank: Computer \_\_\_\_\_ refers to the process of giving instructions to a computer to perform an action or set of actions.

Programming

Question 2

In Python, what is a library?

A reusable collection of code

Question 3

Python is a low-level programming language.

False

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to *Section 1.* *Discover more about Python*.

I'm ready to use the follow-along guide while watching this video.

**Python versus other programming languages**

Python is one of the most popular programming languages for data professionals, which makes it a great addition to your data analytics toolbox! As we’ve previously investigated, Python’s use of syntax to communicate commands and perform tasks mirrors spoken language. This makes Python a much easier programming language to learn. Python’s structure is similar to many other programming languages, but there are some key differences to consider as well.

In this reading, you’ll learn how Python compares to other programming languages data professionals use, including R, Java, and C++.

**Five considerations of programming languages**

Python isn’t the only programming language used for data analysis, but it is one of the most widely used and most powerful. Many data professionals even use more than one programming language. Every language has benefits and drawbacks. For the purposes of this course, examine the following considerations: speed, approachability, variables, data science focus, and programming paradigm.

**Speed**

There are many factors that contribute to the speed of a program’s execution, including compile time, runtime, hardware, installed dependencies, and the efficiency of the code itself. In general, low-level programming languages are faster, but they’re more difficult to learn and work with.

**Approachability**

Approachability refers to how easy it is for new learners to start using a language. Learning new programming languages can be challenging depending on their syntax and overall structure. The **syntax** is the structure of code words, symbols, placement, and punctuation. Semantics builds meaning into those structures by using variables and objects. Additionally, those variables help add flexibility to the programs and objects where data is housed.

**Variables**

Information in code is stored in variables. A **variable** is a named container which stores values in a reserved location in the computer’s memory. The way a programming language uses variables will have an effect on a system's core operations or kernel speed. Some languages use static variables to maintain a value throughout the entire run of a program. Others approach variables as dynamic, allowing values to be determined when a program is run. Some languages even allow declarative variables, which enable a program to determine where a variable should be placed.

**Data science focus**

Programming languages have individual characteristics and can better serve different tasks in data analysis; this means programmers often use them for specific data science tasks.

**Programming paradigm**

Programming languages can be object-oriented, functional, or imperative. Object-oriented programming languages are modeled around data objects. Functional programming languages are modeled around functions. Imperative languages are modeled around code statements that can alter the state of the program itself.

**Programming language comparisons**

Python, R, Java and C++ are four of the most commonly used programming languages for data analysis. The following chart compares them using five considerations: speed, accessibility, variable, data science focus, and programming paradigm.

| **Features by Software** | **Python** | **R** | **Java** | **C++** |
| --- | --- | --- | --- | --- |
| **Speed** | Slower | Depends on configuration and add-ons | Faster | Very fast |
| **Approachability** | Easy to learn | Complex | Easy to learn | Complex |
| **Variable** | Dynamic | Dynamic | Static | Declarative |
| **Data science focus** | Machine learning and automated analysis | Exploratory data analysis and building extensive statistical libraries | Used across projects with open-source assets | Not as widely used but very powerful implementations |
| **Programming Paradigm** | Object-oriented | Functional language | Object-oriented | Multi-paradigm (imperative & object-oriented) |

**Key takeaways**

There are a number of different programming languages that can be used for data analysis. Each language has its own benefits and drawbacks. Learning to work with different languages will give you the opportunity to broaden your data skills and access new tools for your analysis. However, in this certificate program, Python will be your sole focus. As mentioned previously, Python is an easy to learn, object-oriented programming language that engages dynamic variables; though it sometimes requires a longer time to execute, it is a great tool for machine learning and automated analysis.

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to *Section 2.* *Jupyter Notebook*.

I’m ready to use the follow-along guide while watching this video.

**Jupyter Notebook**

An open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text.

**Cells**

The modular code input/output fields into which Jupyter Notebooks are partitioned.

**Activity Overview**





By now, you have been introduced to spreadsheets and their role in data analysis. In this activity, you will work with a spreadsheet in Google Sheets. You will create and edit a spreadsheet, share the sheet with others, and add comments to the sheet. Google Sheets is a cloud-based spreadsheet application. You can use Sheets to organize and analyze data from any online device. All the changes you make are automatically saved in the cloud.

By the time you complete this activity, you will be more familiar with some of the key features of Google Sheets. Knowing how to work with spreadsheet applications is an essential skill for any data analyst. Spreadsheets are powerful tools because they let you store, organize, analyze, and share data.

Step-By-Step Instructions

As you progress in the program, you will become more familiar with spreadsheets. You’ll learn how to use functions, formulas, pivot tables, and more to organize and analyze data. For now, you’ll begin with the basics: how to create, edit, and format your own spreadsheet.

Follow the instructions to complete each step of the activity. Then answer the questions at the end of the activity before going to the next course item

Step 1: Access a spreadsheet application

There are many excellent spreadsheet applications available to data analysts, such as Google Sheets, Microsoft Excel, and more. This activity uses Google Sheets. If you want to follow along with a different spreadsheet application, the steps will be similar.

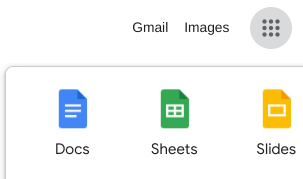
If you want to use Google Sheets, you will need a Google account. If you don’t yet have a Google account, you can follow the instructions from the [Google account support page](https://support.google.com/accounts/answer/27441?hl=en). Once you have your Google account set up, you can start working with Google Sheets!

Step 2: Create a new preadsheet

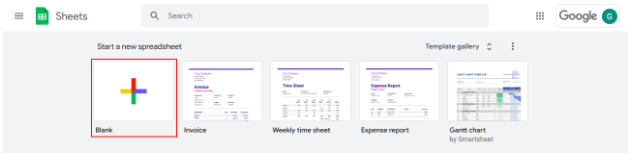
1. To start, go to [www.google.com](http://www.google.com/).

2. Click the **Google apps** icon.

3. Then, click the **Sheets** icon.



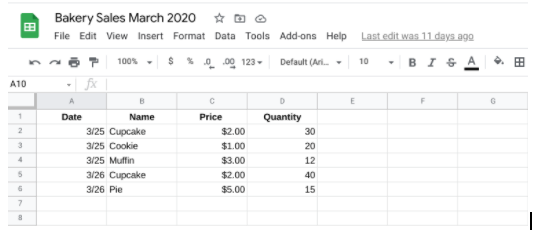
4. In the **Start a new spreadsheet** section, click **Blank** to create a new blank spreadsheet.



Now you’ve got a new spreadsheet that you can use to enter your data!

Step 3: Edit and format your spreadsheet

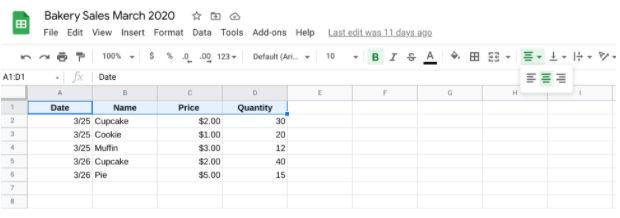
For this activity, you want to create a spreadsheet that contains sales data for a local bakery. Here’s an example of the spreadsheet you will create:



1. After you create your sheet, give it a title. In the upper left corner of your sheet, click **Untitled spreadsheet** and enter a title. Make your title clear and concise. It should describe what the data in the spreadsheet is about. For example, you could use the title **Bakery Sales March 2020** or something similar.

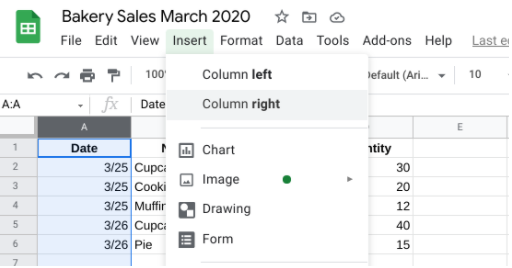
2. Next, enter your column headers. The first row of the spreadsheet is for data attributes, which is another name for column headers. An **attribute** is a characteristic or quality of data used to label a column in a table. It's basically labeling the type of data in each column. In this case, you want to enter data for transaction date, product name, price, and quantity. Click in cell A1 and enter your first header: Date. Click in cell B1 and enter your next header: Name. Repeat this process for the rest of the attributes.

3. Next, format your column headings to make them stand out clearly. For example, you can make the headings stand out from the rest of the rows by using bold and center align. Click on cell A1 and drag the handle across to cell D1 (you have now selected the range A1:D1). Next, click the **bold** icon on the toolbar. Then, click the **Center align** icon on the toolbar.



4. Now, enter relevant data for each column (**Date, Name, Price, Quantity**). Feel free to use the data contained in the example above, or create your own.

5. If you want to add another column between two existing columns, you can insert a new column. First, click on a column to select it. Then, click **Insert** on the menu bar and choose where to add the column. You can do the same thing to insert rows.



6. If you want to move an existing row or column, click the row number or column letter to select it. Then, drag it to a new location.

7. If you want to delete an existing row or column, right-click on the row number or column letter that you want to delete. Then, select **Delete** in the popup menu.

8. **(Optional)** Feel free to explore some of the other features in Google Sheets. Later in the course, we’ll cover many of these features in detail. For example, you can click **Insert** on the menu bar and experiment with charts, images, drawings, and more.

9. As mentioned earlier, your data is saved automatically as you are working.

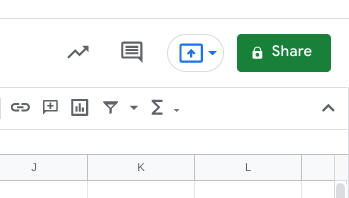
Now that you’ve set up your spreadsheet, share it with others.

Step 4: Share your spreadsheet

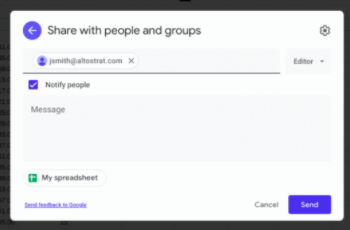
Collaborating with team members is an important part of being a data analyst. When you collaborate, people can make changes at the same time, and you can see their changes as they happen.

In Google Sheets, you can only share files that you own or have edit access to. Because you created your spreadsheet, you are the default owner.

1. To start, click the **Share** icon.

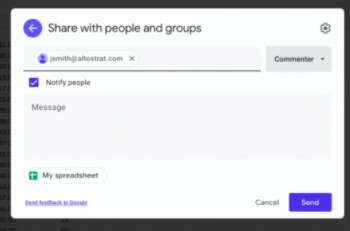


2. Under **Share with people and groups** in the pop-up window, enter the email address of your collaborator.



3. In Google Sheets, users have different access privileges. As owner of the sheet, you can choose the access level for your collaborators. Click **Editor** in the dropdown and choose the access level:

* **Can edit**—collaborators can add and edit content or comments. Choose this access level if you want them to be able to make changes to your spreadsheet.
* **Can comment**—collaborators can add comments, but can't edit content. Choose this level if you just want their feedback.
* **Can view**—collaborators can view the file, but cannot edit or add comments. Choose this level if you want to share a spreadsheet as an optional resource and you don’t need feedback.



4. **(Optional)** By default, Google Sheets will send an email notification to the person or group you share your spreadsheet with to let them know they now have access. You can add a note in that email by entering your text in the **Message** box. For example, you may want to include a description of what data your spreadsheet contains and why you want that person to check it out. If you don’t want to send an email notification, uncheck the **Notify people** box. This is particularly useful during documentation, when you don’t need people to review your work right away, but may want them to in the future.

5. Click **Send**.

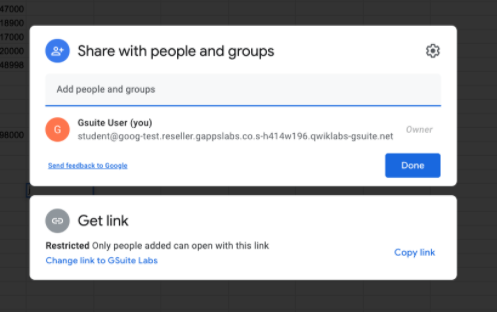
Step 5: Share a link to a file or folder

You can send other people a link to a file or folder so that anyone with the link can open it. This is useful when you want to share your file or folder with a large group and do not want to type in everyone’s individual email addresses. You can share files that you own or have edit access to.

1. To start, click the **Share** icon.

2. In the **Get link** section of the pop-up window, you’ll notice that the default is **Restricted** so that only users with whom you’ve shared the file or folder via email can access it.

3. But, if you want to allow others to access your file or folder without having to add their email addresses, click **Change**.



* By default, your organization is selected. Now choose an access level.

4. Click **Copy link**.

5. Click **Done**.

6. Paste the link in an email or any place you want to share it. If the recipient is in your organization, they will be able to access your file or folder without you having to grant them individual access.

Step 6: Unshare your spreadsheet

You can also stop sharing a spreadsheet that you own at any time. You may want to do this if someone switched jobs or teams and should no longer be looking at your data.

1. To start, click the **Share** icon.

2. Click on the dropdown menu that shows the access level for the person you want to stop sharing the file with, then click **Remove**.

3. Finally, click **Save**.

Step 7: Comment on your spreadsheet

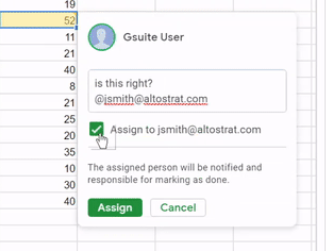
Google Sheets lets you and your collaborators add comments to your sheet and reply to those comments. As a data analyst, this is a great way to share feedback with your teammates.

1. In your sheet, select the cell or cells you'd like to comment on.

2. Do one of the following:

* Right click on the cell and click on **Comment** in the pop-up menu.
* Or click the **Comment icon** in the menu bar at the top of your sheet.

3. Next, enter your comment in the box.



4. **(Optional)** To direct your comment to a specific person, enter an **at sign (@)** followed by their email address. You can add as many people as you want. Each person will get an email with your comment and a link to the spreadsheet file. If that person does not currently have access to your file, Google Sheets will notify you in a pop-up window and ask if you want to grant that user access..

5. **(Optional)** To assign the comment to a specific person, check the **Assign to** box.

6. Finally, click **Comment** or **Assign**.

**1.**

Question 1

**Reflection**

In Google Sheets, what access level should you select if you want your collaborator to comment on your spreadsheet, but not edit the content?

1 point

Can comment

Can edit

Can view

Can delete

**2.**

Question 2

In this activity, you had the opportunity to learn about some of the basic features in Google Sheets. In the text box below, write 2-3 sentences (40-60 words) in response to each of the following questions:

* What do you think are the main advantages of using Google Sheets to organize, analyze, and share your data?
* How do you think the comment feature in Google Sheets can increase collaboration among teammates?

1 point

Your answer cannot be more than 10000 characters.

**How to use Jupyter Notebooks**

Jupyter Notebook is an open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text. This is a great tool to develop and present code in a standardized text block format that is interactive and shareable. You can create code, mathematical formulas, data visualizations, and even freestyle text—all in Jupyter notebooks!

You will be using Jupyter notebooks to write, execute, and present your own code throughout this program. This reading will guide you through using your own notebook. Note, however, that for this certificate program you do not need to download any software. You can complete all activities with the tools provided on the Coursera platform.

**Jupyter Notebook**

You can access Jupyter Notebook directly from your browser or download the desktop application onto your device to work with over 100 programming languages, including some you might already know like R and Python. There is [JupyterLab](https://jupyterlab.readthedocs.io/en/latest/), which is the full suite of tools for working with computational notebooks. There is also [Jupyter Notebook](https://jupyter-notebook.readthedocs.io/en/stable/), which is a more streamlined and simplified tool that nonetheless offers powerful ways to perform interactive computing. Again, for this certificate program, we recommend working within the Jupyter Notebook interface provided by Coursera. Activities that use Jupyter notebooks will be labeled as labs, and you will find relevant instructions for each activity on its landing page.

**Why Jupyter Notebook?**

Notebooks are particularly useful for working with data. Here are some ways that Jupyter notebooks excel:

1. **Modular/interactive computing:** You can write and execute individual chunks of code in small, manageable chunks, which are called cells. You can run a cell without necessarily having to run the whole notebook. This is especially helpful for data exploration and experimentation. Cells are also helpful with debugging, because they provide a user-friendly way to make a mistake, notice that you made the mistake, and iterate back to correct your mistake, without having to re-execute a whole script.
2. **Integration of code and documentation:** Notebooks allow you to combine code, textual explanations, and visualizations like charts, graphs, and tables—all in a single document.
3. **Support for multiple languages:** The Advanced Data Analytics program will use Python, but Jupyter notebooks support many other languages, making them powerful and versatile.
4. **Data exploration and analysis:** The notebook simplifies working with data by offering tools to load, clean, analyze, and examine it in an elegant interface.
5. **Cloud-based services:** Many cloud computing platforms host Jupyter notebooks, which makes it easy to run and share notebooks without setting up a local environment. This is very useful for collaboration.
6. **Libraries and extensions:** There is a rich ecosystem of extensions and plugins that enhance functionality for whatever type of project you’re working on.

**How to use Jupyter notebooks**

Once you’ve opened a Jupyter notebook, it’s time to use it! Here are some tips to get started.

**Command/edit mode**

Notebooks have two working modes: command mode and edit mode. Command mode is used to interact with the notebook as a whole and perform actions like adding, moving, and deleting cells. Edit mode is used to type code or markdown text into a particular cell.

Command mode is indicated by a blue bar on the left side of the current cell.

A screenshot of a computer

Description automatically generated

Edit mode is indicated by a green bar on the left and also a thin green border around the active cell.

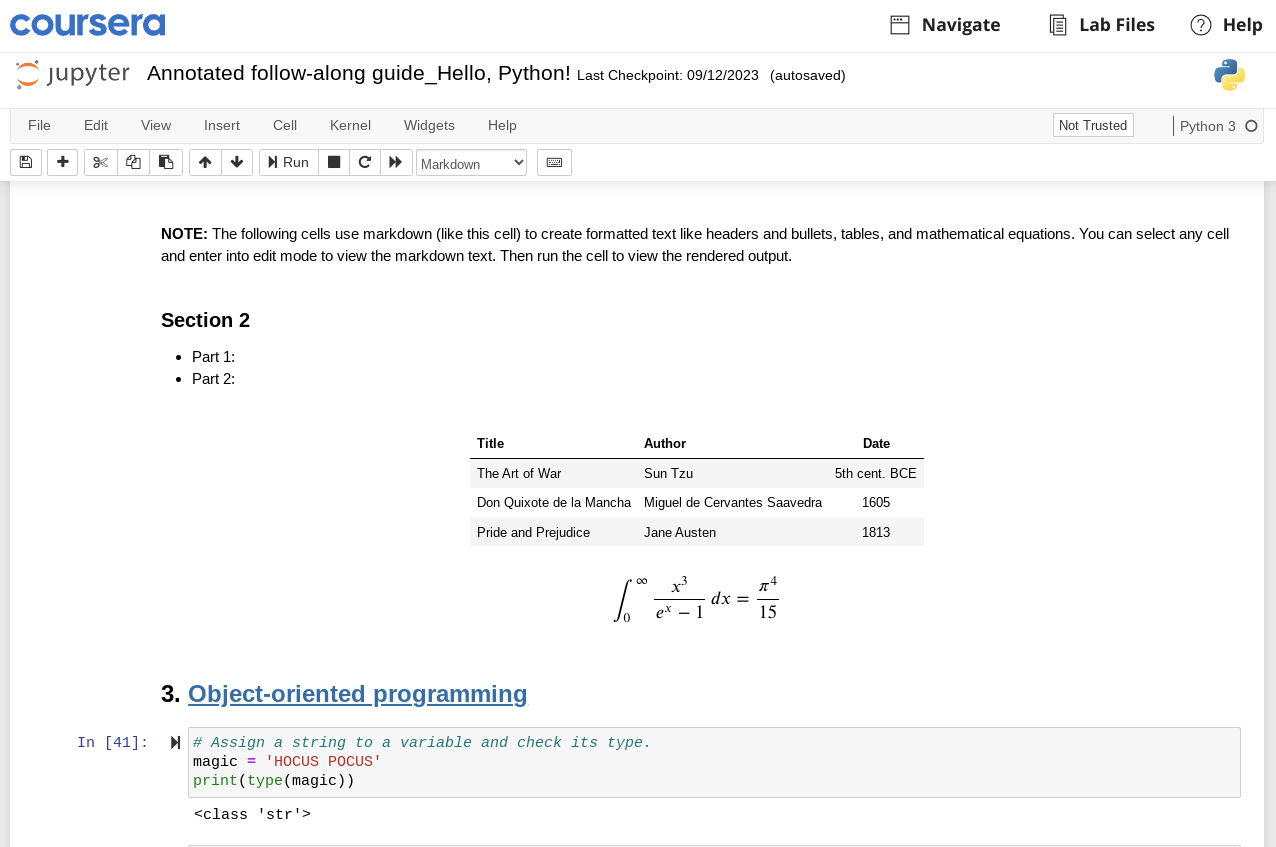
A screenshot of a computer

Description automatically generated

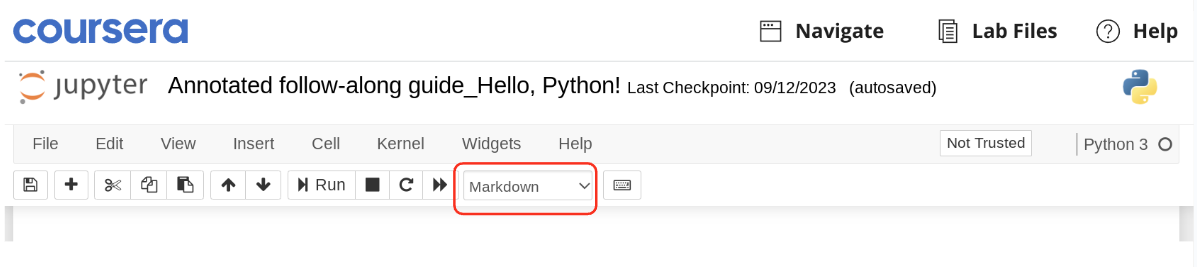
To enter into edit mode, simply click into a cell to insert your cursor there or use the navigation arrows on your keyboard to select a cell and press Enter. To revert back to command mode, click anywhere outside the cell or press the escape key.

**Markdown mode**

Jupyter notebooks allow you to toggle cells between coding mode and Markdown mode. Markdown is a markup language that lets you add formatting elements to plain text. It’s useful because it’s ubiquitous, future-proof, and platform independent. In Jupyter notebooks, Markdown text is used to provide written explanations, analysis, and context to explain the code and its output. In the following example, all of the text above **In [41]:** was written and formatted using Markdown.



To toggle between code and Markdown mode, go to the menu at the top of the page and click where it says “Code.” Then, select “Markdown” from the dropdown menu that appears.



For more information about how to use Markdown, refer to [Markdown guide for basic syntax](https://www.markdownguide.org/basic-syntax/) and the [Markdown guide for extended syntax](https://www.markdownguide.org/extended-syntax/).

**Common actions**

Most actions can be performed using both a mouse/graphic interface and keyboard shortcuts. Here are some of the most common actions.

**Add a new cell**

* Click on Insert in the menu bar at the top of the notebook. Options are to insert a new cell above or below the current cell.
* Keyboard shortcuts (while in command mode):
  + **a:** Insert a cell above the current cell
  + **b:** Insert a cell below the current cell

**Delete a cell**

* Use command mode to select a cell or group of cells.
* Click on Edit in the menu bar at the top of the notebook and select Delete Cells from the dropdown menu.
* Keyboard shortcut (while in command mode):
  + **dd** (press D two times)

**Move a cell**

* Use command mode to select a cell or group of cells.
* Click on the up arrow button or down arrow button in the menu bar at the top of the notebook to move the selected cell(s) up or down

**Run a cell**

* Select a cell and click the **Run** button in the menu bar at the top of the notebook.
* Keyboard shortcuts:
  + **Ctrl + Enter:** Run selected cell
  + **Shift + Enter:** Run selected cell and select next cell
  + **Alt + Enter:** Run selected cell and insert new cell below
* You can run cells from both command mode and edit mode

Press **h** while in command mode for a pop-up window with all available keyboard shortcuts. You can also check out [Jupyter Notebook interface components](https://jupyter-notebook.readthedocs.io/en/stable/ui_components.html) for more detailed descriptions of various notebook features.

**Troubleshooting**

You will use Jupyter notebooks throughout the Advanced Data Analytics certificate program. At times, you might encounter difficulty accessing or running the notebook. Here are some troubleshooting steps to help you if this happens.

**Browser compatibility**

Make sure your internet browser is updated regularly. It is best to use the latest version of Google Chrome, Firefox, or Microsoft Edge. If your browser is outdated or you are using a browser that is not supported by Coursera, you may encounter a problem. If your browser is up to date and you are using one of the browsers listed above and still encountering problems, try restarting your browser or clearing your browser’s cache and cookies. You can also use incognito mode, which prevents your browser from storing cookies and other temporary data.

**Internet connection**

Coursera requires a stable internet connection. If you are experiencing problems starting or running a Jupyter notebook, your internet connection may be slow or unreliable. Some signs of an unstable internet connection may be pages failing to load, freezing labs, or the inability to type or enter commands within the lab environment.

**Pro Tip:** If you are unable to complete a lab on one device, try using another device.

**Troubleshooting steps**

To summarize, here are the troubleshooting steps to try if you encounter a problem with Jupyter notebooks in Coursera.

1. Make sure you are using the latest version of a supported browser: Google Chrome, Firefox, or Microsoft Edge.
2. Restart your browser and clear your browser’s cache and cookies. You can also use incognito mode.
3. Check your internet connection and make sure it is stable. You can try restarting your router and modem to regain a stable connection.
4. Try restarting the lab again.

If all this fails, it’s possible that Coursera is performing maintenance or experiencing a service interruption. In that case, wait a little while and try again.

**Key takeaways**

Jupyter Notebook provides a coding platform where you can develop and debug your own code. Knowing how to use and interact with notebooks will prepare you for upcoming activities where you will try out new Python skills and prepare for the end-of-course project. Python will be a great tool in your toolkit—it will open up more advanced analytics tools like machine learning and automated analysis. And, using Jupyter Notebook will be a great way to build your Python knowledge!

**Resources for more information**

* [Jupyter Notebooks interface training](https://jupyter-notebook.readthedocs.io/en/stable/ui_components.html)
* [Jupyter software homepage](https://jupyter.org/)
* [Jupyter documentation](https://docs.jupyter.org/en/latest/)
* [Jupyter Notebooks cloud](https://jupyter.org/try-jupyter/notebooks/?path=notebooks/Intro.ipynb) (online)
* [Jupyter community forum](https://discourse.jupyter.org/)
* [Jupyter notebooks community forum](https://discourse.jupyter.org/c/notebook/3)
* [Python community forum](https://www.python.org/community/forums/)
* [StackOverflow questions](https://stackoverflow.com/) (crowdsource forum to help solve problems)
* [Jupyter Notebooks installation](https://test-jupyter.readthedocs.io/en/latest/install.html)

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to *Section 3.* *Object-oriented programming*.

I’m ready to use the follow-along guide while watching this video.

**Object-Oriented Programming**

A programming system that is based around objects, which can contain both data and code that manipulates that data.

**Object**

An instance of a class; a fundamental building block of Python

**Core Python Classes**

Integers

Floats

Strings

Booleans

Lists

Dictionaries

-Tuples

-Sets

-Functions

-Ranges

-None

-Custom-defined

**Attribute**

A value associated with an object or class which is referenced by name using dot notation.

**Question**

There are four fundamental concepts in object-oriented programming, including objects. What are the other three concepts? Select all that apply.

- Methods

- Classes

- Attributes

**More about object-oriented programming**

**Note:** This reading contains only a brief introduction to object-oriented programming. A more detailed discussion about the nuances of object oriented programming is beyond the scope of this course.

Previously, we identified object-oriented programming as a programming paradigm that is based around objects, which can contain both data and code that manipulates that data. You may recall that a class is an object’s data type that bundles data and functionality together, and you’ve encountered some examples of this class-specific functionality in the form of methods and attributes. In this reading, you’re going to learn more about object-oriented programming and how it works. Although this certificate program will not require you to define your own classes, having a basic understanding of how this process works will be very helpful when you encounter these concepts along your learning journey.

**Review: Attributes and methods**

Python classes are powerful and convenient because they come with built-in features that simplify common data analysis tasks. These features are known as attributes and methods.

* **Attribute**: A value associated with an object or class which is referenced by name using dot notation.
* **Method**: A function that belongs to a class and typically performs an action or operation.

A simpler way of thinking about the distinction between attributes and methods is to remember that attributes are *characteristics* of the object, while methods are *actions* or *operations*.

For example, if the class were Spaceship, then attributes might be:

**name**

**kind**

**speed**

**tractor\_beam**

These attributes could be accessed by typing:

**Spaceship.name**

**Spaceship.kind**

**Spaceship.speed**

**Spaceship.tractor\_beam**

Notice that these characteristics are accessed using only a dot.

On the other hand, methods of the Spaceship class might be:

**warp()**

**tractor()**

These methods could be used by typing:

**Spaceship.warp()**

**Spaceship.tractor()**

Notice that methods are followed by parentheses, and it’s possible for them to take arguments. For example, **Spaceship.warp(7)** could change the speed of the ship to warp seven.

**Defining classes with unique attributes and methods**

Python lets you define your own classes, each with their own special attributes and methods. This helps all different kinds of programmers to build reusable code that makes their work more efficient. You can even build the Spaceship class mentioned previously. The example, here, demonstrates how to do this.

**Note:** The following code block is not interactive.

class Spaceship:

   # Class attribute

   tractor\_beam = 'off'

   # Instance attributes

   def \_\_init\_\_(self, name, kind):

       self.name = name

       self.kind = kind

       self.speed = None

  # Instance methods

   def warp(self, warp):

       self.speed = warp

       print(f'Warp {warp}, engage!')

   def tractor(self):

       if self.tractor\_beam == 'off':

           self.tractor\_beam = 'on'

           print('Tractor beam on.')

       else:

           self.tractor\_beam = 'off'

           print('Tractor beam off')

For this course you don’t have to learn the syntax to create classes. Just notice that the class itself is defined first, and then indented beneath it are the attributes and methods. This is what it means when an attribute or method “belongs to a class.” Attributes and methods are defined in the code for that class.

A class is like a blueprint for all things that share characteristics and behaviors. In this case, the class is Spaceship. There can be all different kinds of spaceships. They can have different names and different purposes. Whenever you create an object of a given class, you’re creating an **instance** of that class. This is also known as **instantiating** the class. In the code above, every time you instantiate an object of the Spaceship class it will start with its tractor beam set to off. The tractor beam is a class attribute. All instances of the Spaceship class have one. There are also instance attributes. These are attributes that you can assign when you instantiate the object.

# Create an instance of the Spaceship class (i.e. "instantiate")

ship = Spaceship('Mockingbird','rescue frigate')

# Check ship's name

print(ship.name)

# Check what kind of ship it is

print(ship.kind)

# Check tractor beam status

print(ship.tractor\_beam)

The next block of code uses the **warp()** method to set the warp speed to seven. Then it checks the current speed of the ship using the speed attribute.

# Set warp speed

ship.warp(7)

# Check speed

ship.speed

This final block of code uses the **tractor()** method to toggle the tractor beam. Then it checks the current status of the tractor beam using the **tractor\_beam** attribute.

# Toggle tractor beam

ship.tractor()

# Check tractor beam status

print(ship.tractor\_beam)

This is just a basic example meant to demonstrate some of the fundamental ways that classes, attributes, and methods work and how they relate to each other, but classes can be very complex and have many attributes and methods. Depending on the work you’re doing as a data professional, knowledge about object-oriented programming will be helpful as you define your own classes, attributes, and methods to investigate the patterns, relationships, and meaning data holds.

**Key takeaways**

Classes comprise the core objects of Python, which is why Python is known as an object-oriented language. Class objects are powerful because they contain unique tools designed specifically for that class packaged within them. Methods are functions that belong to a class; they perform actions or operations, and they use parentheses. Attributes are values or characteristics associated with a class or class instance; they do not use parentheses. And, while there are many classes, attributes, and methods pre-built into Python, there is a high level of customization offered in the object-oriented programming paradigm.

“It’s a form of art for me, that you get to create something from scratch that does not exist and you put it into production and it is used by about a hundred million users.”

“Python has helped me become a better data scientist, a better machine learning engineer.”

**Test your knowledge: The power of Python**

Review Learning Objectives

Question 1

Fill in the blank: Jupyter Notebook is an open source \_\_\_\_\_ for creating and sharing documents containing live code, mathematical formulas, visualizations, and text.

web application

Question 2

In object-oriented programming, objects can contain both data and useful code that manipulates that data.

Question 3

What are examples of classes in Python? Select all that apply.

- Integers

- Floats

- Strings

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to *Section 4.* *Variables and data types*.

I'm ready to use the follow-along guide while watching this video.

**Data type**

An attribute that describes a piece of data based on its values, its programming language, or the operations it can perform.

**Variable algorithm questions**

-What’s the variable’s name?

-What’s the variable’s type?

-What’s the variable’s starting value?

Assignment

The process of storing a value in a variable.

**Expression**

A combination of numbers, symbols, or other variables that produce a result when evaluated.

**Dynamic typing**

Variables can point to object of any data type

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to *Section 5. Create precise variable names*.

**Naming conventions**

Consistent guidelines that describe the content, creation date, and version of a file in its name.

**Naming restrictions**

Rules built into the syntax of the language itself that must be followed.

**Keyword**

A special word that is reserved for a specific purpose and that can only be used for that purpose.

**Naming restrictions for variables**

-Only include letters, numbers, and underscores

-Must start with a letter or underscore

-Case-sensitive

-Cannot include parentheses

Example variable names

any\_a\_variable

any\_a\_variable\_2

1\_is\_a\_number

Apples\_&\_oranges

**Question**

Fill in the blank: Naming \_\_\_\_\_ are rules built into the syntax of the language itself that must be followed.

restrictions

**Explore Python syntax**

Python is a flexible programming language used in a wide range of fields, including software development, machine learning, and data analysis. Python is one of the most popular programming languages for data professionals, so getting familiar with its fundamental syntax and semantics will be useful for your future career. In this reading, you will learn about Python’s syntax and semantics, as well as where to find resources to further your learning.

**The Language of Python**

People use language to communicate and give instructions to each other. Computers do the same thing, except computers use languages like Python, C++, and Java. So, in order to communicate instructions to the computer, programmers need to arrange ideas and concepts into a language it will understand.

Python syntax includes words that represent objects and commands, as well as punctuation that gives the words structure, hierarchy, and context. Together, the words and punctuation communicate ideas and processes; this is known as semantics. Semantics is the meaning conveyed by the syntax. The best way to learn syntax and semantics is through exposure. Practice coding and become familiar and comfortable with reading other people’s code. In addition, there are some general conventions that practitioners use to help maintain stylistic uniformity within the language.

Coding languages are similar to spoken languages in that they have a way to classify words according to their function. For example, English sentences are composed of nouns, verbs, prepositions, etc.

Here are some of the basics:

* **Variables:** Represent data stored as strings, tuples, dictionaries, lists, and objects (note: future readings explain these categories)
  + Example: **student\_name**
* **Keywords:** Special words that are reserved for specific purposes and that can only be used for those purposes
  + Examples:
    - **in**
    - **not**
    - **or**
    - **for**
    - **while**
    - **return**
* **Operators:** Symbols that perform operations on objects and values
  + Examples:
    - **+** Addition
    - **-** Subtraction
    - **\*** Multiplication
    - **/** Division
    - **\*\*** Exponentiation
    - **%** Modulo (returns the remainder after a division). Example: **10 % 3** = **1**
    - **//** Floor division (divides the first operand by the second operand and rounds the result down to the nearest integer. Example: **5 // 2** = **2**
    - **>** Greater than (returns a Boolean of whether the left operand is greater than the right operand)
    - **<** Less than (returns a Boolean of whether the left operand is less than the right operand)
    - **==** Equality (returns a Boolean of whether the left operand is equal to the right operand)
* **Expressions:** A combination of numbers, symbols, and variables to compute and return a result upon evaluation
  + Example: **[1, 2, 3] + [2, 4, 6]**
* **Functions:** A group of related statements to perform a task and return a value
  + Example:

1

2

3

4

5

def to\_celsius(x):

   '''Convert Fahrenheit to Celsius'''

   return (x-32) \* 5/9

to\_celsius(75)

RunReset

* **Conditional statements:** Sections of code that direct program execution based on specified conditions
  + Example:

1

2

3

4

5

6

7

8

number = -4

if number > 0:

   print('Number is positive.')

elif number == 0:

   print('Number is zero.')

else:

   print('Number is negative.')

RunReset

As you’ll surely discover, Python generates syntax errors for incorrectly used keywords and syntax.

Example:

1

print(This will throw an error because I didn’t make it a string.)

RunReset

**Naming rules and conventions**

When assigning names to objects, programmers adhere to a set of rules and conventions which help to standardize code and make it more accessible to everyone. Here are some naming rules and conventions that you should know:

* Names cannot contain spaces.
* Names may be a mixture of upper and lower case characters.
* Names can’t start with a number but may contain numbers after the first character.
* Variable names and function names should be written in snake\_case, which means that all letters are lowercase and words are separated using an underscore.
* Descriptive names are better than cryptic abbreviations because they help other programmers (and you) read and interpret your code. For example, student\_name is better than sn. It may feel excessive when you write it, but when you return to your code you’ll find it much easier to understand.

Tim Peters, a Python programmer, wrote this now-famous “poem” of guiding principles for coding in Python:

**The Zen of Python**

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!

Finally, it’s helpful to bookmark the [PEP 8 Style Guide for Python](https://peps.python.org/pep-0008/) so you can reference it as needed. This reading is limited in scope, and PEP 8 is a more exhaustive resource for style-related matters. PEP stands for Python Enhancement Proposals. These are a running catalog of ways to improve or standardize Python as a language. Because Python is open source, PEP offers a framework to guide developers and build consensus around ideas. It’s a useful and trusted resource.

**Key takeaways**

Syntax and semantics are what give form and meaning to a language, including Python.  A large part of learning a new language is familiarizing yourself with its syntax and semantics. Much of this comes through exposure and practice, but there are a few guiding principles and resources that can help you along the way. If you learn the rules about naming objects and build a bank of resources that you can reference for guidance, you’ll surely make progress as a Python learner. As you get more familiar with Python, you’ll be able to communicate more efficiently with computers and do more with your data analysis tools!

**Resources for more information**

Here are a few useful resources to help you get more familiar with Python:

* Python [Reference Library](https://docs.python.org/3/library/)
  + [Built-in Data types](https://docs.python.org/3/library/stdtypes.html)
  + [Built-in functions](https://docs.python.org/3/library/functions.html#built-in-functions)
* [Python operators](https://python-reference.readthedocs.io/en/latest/docs/operators/index.html)

**Question**

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [*Annotated follow-along guide: Hello, Python!*](https://www.coursera.org/learn/get-started-with-python/ungradedLab/JnPjr/annotated-follow-along-guide-hello-python) in a new browser and navigate to Section 6. *Data types and conversions*.

I’m ready to use the follow-along guide while watching this video.

**String**

A sequence of characters and punctuation that contains textual information.

**Immutable data type**

A data type in which the values can never be altered or updated.

**Integer**

A data type used to represent whole numbers without fractions.

**Question**

Fill in the blank: The \_\_\_\_\_ data typeis a sequence of characters and punctuation that contains textual information.

string

**Implicit conversion**

Python automatically converts one data type to another without user involvement.

**Explicit conversion**

Users convert the data type of an object to a required data type.

Instructions

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: Using Python syntax

### 1.

Question 1

Variables can only store values of numeric data types.

1 / 1 point

True

False

Correct

Variables can store values of any data type.

### 2.

Question 2

What are best practices when assigning a new variable? Select all that apply.

1 / 1 point

Determine the variable’s starting value

Correct

Best practices when assigning a new variable include determining the variable’s name, data type, and starting value.

Determine the variable’s keyword

Determine the variable’s data type

Correct

Best practices when assigning a new variable include determining the variable’s name, data type, and starting value.

Determine the variable’s name

Correct

Best practices when assigning a new variable include determining the variable’s name, data type, and starting value.

### 3.

Question 3

Fill in the blank: An \_\_\_\_\_ is a combination of numbers, symbols, or other variables that produce a result when evaluated.

1 / 1 point

object

attribute

expression

argument

Correct

An expression is a combination of numbers, symbols, or other variables that produce a result when evaluated.

### 4.

Question 4

Which data type represents numbers that contain decimals?

1 / 1 point

Float

Boolean

Integer

String

Correct

The float data type represents numbers that contain decimals.

**Review**

Introduction to Python

Jupyter Notebook

Object-oriented programming

Variables

Naming conventions

Data Types

# Glossary terms from Module 1

# Terms and definitions from Course 2, Module 1

**Argument**: Information given to a function in its parentheses

**Assignment**: The process of storing a value in a variable

**Attribute**: A value associated with an object or class which is referenced by name using dot notation

**Cells**: The modular code input and output fields into which Jupyter Notebooks are partitioned

**Class**: An object’s data type that bundles data and functionality together

**Computer programming**: The process of giving instructions to a computer to perform an action or set of actions

**Data type**: An attribute that describes a piece of data based on its values, its programming language, or the operations it can perform

**Dot notation**: How to access the methods and attributes that belong to an instance of a class

**Dynamic typing**: Variables that can point to objects of any data type

**Explicit conversion**: The process of converting a data type of an object to a required data type

**Expression**: A combination of numbers, symbols, or other variables that produce a result when evaluated

**Float**: A data type that represents numbers that contain decimals

**Immutable data type**: A data type in which the values can never be altered or updated

**Implicit conversion**: The process Python uses to automatically convert one data type to another without user involvement

**Integer**: A data type used to represent whole numbers without fractions

**Jupyter Notebook**: An open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text

**Keyword**: A special word in a programming language that is reserved for a specific purpose and that can only be used for that purpose

**Markdown**: A markup language that lets the user write formatted text in a coding environment or plain-text editor

**Method**: A function that belongs to a class and typically performs an action or operation

**Naming conventions**: Consistent guidelines that describe the content, creation date, and version of a file in its name

**Naming restrictions**: Rules built into the syntax of a programming language

**Object**: An instance of a class; a fundamental building block of Python

**Object-oriented programming**: A programming system that is based around objects which can contain both data and code that manipulates that data

**Programming languages**: The words and symbols used to write instructions for computers to follow

**String**: A sequence of characters and punctuation that contains textual information

**Syntax**: The structure of code words, symbols, placement, and punctuation

**Typecasting:** Converting data from one type to another (see **explicit conversion**)

**Variable**: A named container which stores values in a reserved location in the computer’s memory

# Module 1 challenge

### 1.

Question 1

A manager explores which programming language their team can use to perform coding work most effectively. They want a language with thousands of open-source libraries for reference. Ultimately, they select Python. What are some other benefits of using Python for coding? Select all that apply.

Python has a large and supportive user community.

Python is based on simple syntax.

Python is versatile.

### 2.

Question 2

Fill in the blank: The open-source web application \_\_\_\_\_ enables users to share documents containing live code, mathematical formulas, visualizations, and text. It also puts all output into a single document, which is very useful when first learning about programming.

Jupyter Notebook

### 3.

Question 3

FIll in the blank: In Python, a \_\_\_\_\_ is an object’s data type that bundles data and functionality together.

class

### 4.

Question 4

In Python, what does assignment refer to?

The process of storing a value in a variable

### 5.

Question 5

A data professional names a variable in Python, then receives an error message. What might they have included in the name incorrectly? Select all that apply.

Reserved functions

### 6.

Question 6

Which of the following is an effective variable name?

**able\_variable**

### 7.

Question 7

In Python, which data type represents whole numbers without fractions?

Integer

### 8.

Question 8

A data professional converts floats into strings using a predefined function. This is an example of what type of conversion?

Explicit conversion

**Module 2:**

**What you’ll learn**

Functions

Write clean code

Commenting

Operators

Conditional statements

“From that, I took away just learning to admit your mistakes, facing it head-on, and not being afraid to talk it through with somebody.”

**Tips for beginners**

* 1. Start practicing to truly understand coding
  2. Learn to debug
  3. Start compartmentalizing to accelerate your learning journey.

# Annotated follow-along guide: Functions and conditional statements

Instructions

All of the instructional videos with onscreen coding demonstrations have a corresponding follow-along guide that is available to you. The follow-along guide is an annotated Jupyter notebook organized to match the content from each module. It contains the same code shown in the videos for that module. This guide is provided for your reference; you do not need to add any text or run the code yourself. If you would like to run the code, you will need to run each cell sequentially for the code to function as intended.

In addition to content that is identical to what is covered in the videos, you’ll often find additional information throughout the guide to explain the purpose of each concept covered, why the code is written in a certain way, and tips for running the code.

The landing page for each follow-along notebook also provides information about data sources (when relevant) and tips on how to access and use these guides.

# Data dictionary

This module does not use any imported data, but when data is imported this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

## Set up a split screen

While watching the videos that follow this notebook, you may find it helpful to track the instructor’s progress by following along in your own Jupyter Notebook. It can be helpful to review the code notebook alongside the video, especially if you’re new to coding in Python.

To do so:

1. Open the video in one browser window.
2. Then, open the annotated follow-along guide in a separate window.
3. Arrange your screen so that the video and the follow-along guide are both visible.

A screenshot of a computer

Description automatically generated

## Tips for working with the follow-along guide

Follow these suggestions to enhance your learning experience:

* Reference the [Jupyter notebooks reading](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks) before starting if you need more information on working with the Jupyter notebooks.
* Watch for an in-video message to advise that the video you are viewing contains coding instruction and examples.
* Go to the section of the follow-along guide for the current module's content. The follow-along guide has different sections for each video included in the module's content. The in-video message will direct you to the relevant section in the guide for the specific video you are viewing.
* Follow along in the notebook as the instructor discusses the code.

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Functions and conditional statements](https://www.coursera.org/learn/get-started-with-python/ungradedLab/qxjEj/annotated-follow-along-guide-functions-and-conditional-statements) in a new browser and navigate to Section 1. Defining functions and returning values.

I’m ready to use the follow-along guide while watching this video

**Function**

A body of reusable code for performing specific processes or tasks.

## Question

Fill in the blank: A \_\_\_\_\_ is a body of reusable code for performing specific processes or tasks.

function

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Functions and conditional statements](https://www.coursera.org/learn/get-started-with-python/ungradedLab/qxjEj/annotated-follow-along-guide-functions-and-conditional-statements) in a new browser and navigate to Section 2. Write clean code.

I’m ready to use the follow-along guide while watching this video.

**Modularity**

The ability to write code in separate components that work together and that can be reused for other programs.

**Refactoring**

The process of restructuring code while maintaining its original functionality.

**Self-documenting code**

Code written in a way that is readable and makes its purpose clear.

## Question

Which Python feature enables data professionals to define code once, then use it many times without having to rewrite it?

Reusability

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Functions and conditional statements](https://www.coursera.org/learn/get-started-with-python/ungradedLab/qxjEj/annotated-follow-along-guide-functions-and-conditional-statements) in a new browser and navigate to Section 3. Use comments to scaffold your code.

I’m ready to use the follow-along guide while watching this video.

## Question

Fill in the blank: Lines of code that begin with a \_\_\_\_\_ serve as comments and don’t get executed.

hashtag

# Reference guide: Functions

As you’ve been learning, functions are bodies of reusable code for performing specific processes or tasks. They help you do more work with less code. Function examples include:

* A specific calculation or measurement, such as converting Fahrenheit to Celsius
* An inventory utility to iterate quantities and calculate the total cost of goods in stock
* Building a DataFrame from a series or dictionary data
* An application utility such as a spell checker

In this reading, you will learn how to define, build, and call functions.

## Save this course item

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the following link and select “Use Template.”

[Reference guide: Functions](https://docs.google.com/document/d/1Kxm7hv3w6ddQ6C2-m1ZNWD-EWte5QDlF6it0unmLjaw/template/preview?resourcekey=0-BeaGUzArCDKD0NLcRvzSGw)

OR

If you don’t have a Google account, you can download the item directly from the following attachment.

## Function syntax

Define functions using the following syntax and format:

**Note:** The following code block is not interactive.

1

2

3

4

5

6

7

8

def my\_function(parameters):

    '''

    Docstring.

    Summarize the function's behavior and explain its arguments and return values.

    '''

    code block

    return value

1. Begin with the def keyword followed by the function’s name, then put its parameters/arguments in parentheses, ending with a colon.
   1. Python convention is to use snake\_case (lowercase words separated by underscores) for function names.
2. For important functions or functions whose purposes or operations are not very obvious, include a docstring. Write the docstring between three opening and closing quotation marks.
   1. The docstring should be in the form of a command (e.g., “Add two numbers” as opposed to “Adds two numbers”).
   2. The docstring should summarize the function’s behavior and explain its arguments and return values.
   3. The docstring should be indented four spaces from the definition statement.
3. Write the body of the function.
   1. All code should be indented at least four spaces from the definition statement, but there can be many levels of indentation depending on the complexity of the code.
4. Finally, use a return statement to return a value or a print statement to print something to the console and complete the function. This line should also be indented four spaces.

## **return vs. print**

Sometimes the difference between return statements and print statements isn’t clear to new learners of Python. It’s important to understand what each action is and when to use it. Return statements give you a result that you can use for something else. It doesn’t have to be something that prints when the function is run. Print statements print something to the console and nothing more. Think of it like this: a return statement is like your brother going to the market and bringing you back a bag of potatoes. A print statement is like your brother going to the market, coming home, and telling you what kind of potatoes were for sale. With the return statement, you have some potatoes to cook. With the print statement, you just know what potatoes are available, but you don’t have any potatoes.

## **Functions vs. methods**

Functions and methods are very similar, but there are a few key differences. Methods are a specific type of function. They are functions that belong to a class. This means that you can use them—or “call” them—by using dot notation.

**Method example:**

1

2

my\_string = 'The eagles filled the sky.'

my\_string.split()

RunReset

The split method is a function that belongs to the string class. It splits strings on their whitespaces.

Standalone functions do not belong to a particular class and can often be used on multiple classes.

**Note:** The following code block is not interactive.

**Function example:**

1

2

 sum([6, 3])

9

You can review [Python’s list of built-in functions](https://docs.python.org/3/library/functions.html) and research how other people use them in the [Jupyter forum](https://discourse.jupyter.org/), [StackOverflow](https://stackoverflow.com/), and other online communities.

## Resources for more information

For more information on functions, consider the Python [Reference Library](https://docs.python.org/3/library/), [Data types,](https://docs.python.org/3/library/stdtypes.html) [Functions](https://docs.python.org/3/library/functions.html#built-in-functions), [Symbols](https://wiki.python.org/moin/PythonGlossary?action=AttachFile&do=view&target=PySymbols.html)

* [Built-in functions](https://docs.python.org/3/library/functions.html#built-in-functions):
  + [enumerate](https://docs.python.org/3/library/functions.html#enumerate)()
  + [isinstance](https://docs.python.org/3/library/functions.html#isinstance)()
  + [dict](https://docs.python.org/3/library/functions.html#func-dict)()
  + [type](https://docs.python.org/3/library/functions.html#type)()
  + [len](https://docs.python.org/3/library/functions.html#len)()
  + [set](https://docs.python.org/3/library/functions.html#func-set)()
  + [zip](https://docs.python.org/3/library/functions.html#zip)()
* [Docstring conventions](https://peps.python.org/pep-0257/): PEP 257 guide to writing docstrings

# Instructions

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: Functions

### 1.

Question 1

A data professional wants to define their own Python function. What keyword should they use at the start of the function block?

1 / 1 point

**return**

**for**

**with**

**def**

Correct

To define a function, use the keyword **def** at the start of the function block.

### 2.

Question 2

Modularity is the ability to write code in separate components that work together.

1 / 1 point

True

False

Correct

Modularity is the ability to write code in separate components that work together. Modularity allows data professionals to reuse code for other programs.

### 3.

Question 3

Why do data professionals use comments for their Python code? Select all that apply.

1 / 1 point

To execute code

To outline the steps of a process

Correct

Data professionals use comments for their Python code to provide helpful explanations, outline the steps of a process, and document their work for teammates.

To provide helpful explanations

Correct

Data professionals use comments for their Python code to provide helpful explanations, outline the steps of a process, and document their work for teammates.

To document their work for teammates

Correct

Data professionals use comments for their Python code to provide helpful explanations, outline the steps of a process, and document their work for teammates.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Functions and conditional statements](https://www.coursera.org/learn/get-started-with-python/ungradedLab/qxjEj/annotated-follow-along-guide-functions-and-conditional-statements) in a new browser and navigate to Section 4. Make comparisons using operators.

I’m ready to use the follow-along guide while watching this video.

**Boolean**

A data type that has only two possible values, usually true or false

**Comparators**

Operators that compare two values and produce Boolean values (True/False)

**Logical operators**

Operators that connect multiple statements together and perform more complex comparisons.

-and

-or

-not

The and **operator** needs both expressions to be true to return a True result.

If we use the or **operator** the expression will be True if either of the expressions is true, and False only when both expressions are false.

The not **operator** inverts the value of the expression that follows it.

## Question

Which of the following words are examples of logical operators? Select all that apply.

not

and

or

# Reference guide: Python operators

You’ve encountered many Python operators already. Many of them likely feel very familiar to you. After all, there’s nothing novel about addition and subtraction in Python. But there are many more operators than the ones used for basic arithmetic! Operators are characters that enact specific arithmetic, logical actions, or processes. Data professionals use operators all the time in their work, and they’re a rudimentary part of Python programming, so it’s important to learn them. This reading is a guide to the various operators available to you in Python.

## Save this course item

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the following link and select “Use Template.”

Reference guide: [Python operators](https://docs.google.com/document/d/1R6CU4cNKvztLqMNCferbGzYMk7KTRv0ZU4eGDlHJ_y8/template/preview)

OR

If you don’t have a Google account, you can download the item directly from the following attachment.

## Comparators

In Python, you can use comparison operators to compare values. When a comparison is made, Python returns a Boolean result—True or False. Python uses the following comparators:

| **Operation** | **Operator** |
| --- | --- |
| greater than | **>** |
| greater than or equal to | **>=** |
| less than | **<** |
| less than or equal to | **<=** |
| not equal to | **!=** |
| equal to | **==** |

**Notes:**

* The single equals sign (**=**) is reserved for assignment statements. If you use a single equal sign to make a comparison, the computer will return a **SyntaxError**.
* If you try to compare data types that aren’t compatible, like checking if a string is greater than an integer, Python will throw a **TypeError**.

## Logical operators

Python also has three logical operators that can be combined with comparators to create more complex statements.

These operators are:

* **and**
  + evaluates to True only if both statements are true
* **or** 
  + evaluates to True if one or both of the statements are true
* **not** 
  + reverses the evaluation
  + If the statement evaluates to True, returns False; if the statement evaluates to False, returns True

Examples:

1

2

3

4

5

x = 3

my\_list = [3, 4, 6, 10]

print(x < 3 and x != 0)

print(x >= len(my\_list) or x == min(my\_list))

print(x not in my\_list)

RunReset

## Arithmetic operators

Python is also capable of performing mathematical operations using a set of built-in operators. These arithmetic operators are:

| **Operation** | **Operator** | **Example** |
| --- | --- | --- |
| Addition | **+** | **[IN]  5 + 2**  **[OUT] 7** |
| Subtraction | **-** | **[IN] 5 - 2**  **[OUT] 3** |
| Multiplication | **\*** | **[IN] 5 \* 2**  **[OUT] 10** |
| Division | **/** | **[IN] 5 / 2**  **[OUT] 2.5** |
| Modulo (the remainder of a division) | **%** | **[IN] 5 % 2**  **[OUT] 1** |
| Exponentiation | **\*\*** | **[IN] 5 \*\* 2**  **[OUT] 25** |
| Floor division  (the number of times the denominator can fully go into the numerator) | **//** | **[IN] 5 // 2**  **[OUT] 2** |

There are many other mathematical operations that can be performed in Python using functions from special libraries, which you’ll learn about later. Python uses a core set of operators to make comparisons, perform logical operations, and compute arithmetic operations. These operators can be combined in statements to perform an infinite number of tasks and operations.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Functions and conditional statements](https://www.coursera.org/learn/get-started-with-python/ungradedLab/qxjEj/annotated-follow-along-guide-functions-and-conditional-statements) in a new browser and navigate to Section 5. Use if/elif/else statements to make decisions.

I’m ready to use the follow-along guide while watching this video.

## Question

Fill in the blank: \_\_\_\_\_ describes the ability of a program to alter its execution sequence.

Branching

**Using of branching**

-Bin data based on its value

-Backup files

-Restrict login access

# Reference guide: Conditional statements

Conditional statements are an essential part of programming. They allow you to control the flow of information based on certain conditions. In Python, **if, elif,** and **else** statements are used to implement conditional statements. Using conditional statements to branch program execution is a core part of coding for most data professionals, so it’s important to understand how they work. This reading is a reference guide to conditional statements.

## Save this course item

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the following link and select “Use Template.”

Reference guide: [Conditional statements](https://docs.google.com/document/d/1DnAA9MY1xW-Xw2UrBaAFbHNkkfQxi1ZAYzYazwiCaHo/template/preview?resourcekey=0-nyt9oGjsbubU0Ky8oeff5Q#heading=h.xr97bf74ctnq)

OR

If you don’t have a Google account, you can download the item directly from the following attachment.

[Reference guide\_ Conditional statements](https://d3c33hcgiwev3.cloudfront.net/8AjHHn8BQz2FjQJu6_ZseQ_15fb6f2e66ff4f08b2accb3bf3cabbf1_Reference-guide_-Conditional-statements.docx?Expires=1720137600&Signature=jPIFZ7b2sCMjMQ7RcU2sSSuVDr5XyFr5SQwayxVBV2PSoLwXu4mrUdNKoh2qHwhSHwj5Bt~f7UbW1l3phLrhbkBOHEKSOqaRlxseUDVfK--bHUWJ6vq8MHK0Z7zZkQJ5JnZujokRqiJIjn~xA3WBn9aEsxDLlhI-QjqEbEu7wJ8_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

[DOCX File](https://d3c33hcgiwev3.cloudfront.net/8AjHHn8BQz2FjQJu6_ZseQ_15fb6f2e66ff4f08b2accb3bf3cabbf1_Reference-guide_-Conditional-statements.docx?Expires=1720137600&Signature=jPIFZ7b2sCMjMQ7RcU2sSSuVDr5XyFr5SQwayxVBV2PSoLwXu4mrUdNKoh2qHwhSHwj5Bt~f7UbW1l3phLrhbkBOHEKSOqaRlxseUDVfK--bHUWJ6vq8MHK0Z7zZkQJ5JnZujokRqiJIjn~xA3WBn9aEsxDLlhI-QjqEbEu7wJ8_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

## Conditionals syntax

In earlier videos, you learned some built-in Python operators that allow you to compare values, and some logical operators that you can use to combine values. You also learned how to use operators in **if-elif-else** blocks.

**Note**: The following code block is not interactive.

The basic syntax of **if-elif-else** statements in Python is as follows:

1

2

3

4

5

6

7

8

9

10

if condition1:

   # block of code to execute if the condition evaluates to True

elif condition2:

   # block of code to execute if condition1 evaluates to False

   # and condition2 evaluates to True

else:

   # block of code to execute if BOTH condition1 and condition2

   # evaluate to False

Here, **condition1** and **condition2** are expressions that evaluate to either True or False. If the condition in the if statement is true, then the block of code that follows is executed. Otherwise, it is skipped.

The **elif** statement stands for “else if,” and it is used to specify an alternative condition to check if the first condition is false. You can have any number of **elif** statements in your code. If the preceding condition is false and the **elif** condition is true, then the block of code that follows the **elif** statement is executed.

The **else** statement is used to specify what code to execute if both the if statement and any subsequent **elif** statements are false.

Here is an example that uses all three kinds of statements:

1

2

3

4

5

6

7

x = 8

if x > 5:

   print('x is greater than five')

elif x < 5:

   print('x is less than five')

else:

   print('x is equal to five')

RunReset

Some important things to note about conditional statements in Python:

* The **elif** and **else** statements are optional. You can have an **if** statement by itself.
* You can have multiple **elif** statements.
* You can only have one **else** statement, and only at the end of your logic block.
* The conditions must be an expression that evaluates to a Boolean value (True or False).
* Indentation matters! The code associated with each conditional statement must be indented below it. The typical convention for data professionals is to indent four spaces. Indentation mistakes are one of the most common causes of unexpected code behavior.

# Activity: Conditional statements

# Instructions

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: Conditional statements

### 1.

Question 1

Fill in the blank: Comparators are operators that compare two values and produce \_\_\_\_\_ values.

1 / 1 point

boolean

float

integer

string

Correct

Comparators are operators that compare two values and produce boolean values. Booleanis a data type that has only two possible values, usually true or false.

### 2.

Question 2

When comparing two expressions, what logical operator requires both expressions to be true to return a **True** result?

1 / 1 point

**not**

**or**

**with**

**and**

Correct

When comparing two expressions, the **and** logical operator requires both expressions to be true to return a **True** result. Logical operators can connect multiple statements and perform complex comparisons.

### 3.

Question 3

Which of the following is a reserved keyword that sets up a condition in Python?

1 / 1 point

**as**

**with**

**else**

**if**

Correct

**if** is a reserved keyword that sets up a condition in Python.

### 4.

Question 4

The **elif** keyword only works when there are exactly two possible conditions in the code.

1 / 1 point

True

False

Correct

The **elif** keyword allows for more than two possible conditions in the code—up to an unlimited number of comparison cases.

**Review**

-Functions

-Writing clean code

-Commenting

-Operators

-Conditional statements

# Glossary terms from module 2

# Terms and definitions from Course 2, Module 2

**Algorithm**: A set of instructions for solving a problem or accomplishing a task

**Boolean**: A data type that has only two possible values, usually true or false

**Branching**: The ability of a program to alter its execution sequence

**Comparator**: An operator that compares two values and produces Boolean values (True/False)

**def**: A keyword that defines a function at the start of the function block

**Docstring**: A string at the beginning of a function’s body that summarizes the function’s behavior and explains its arguments and return values

**elif**: A reserved keyword that executes subsequent conditions when the previous conditions are not true

**else**: A reserved keyword that executes when preceding conditions evaluate as False

**Function**: A body of reusable code for performing specific processes or tasks

**if**: A reserved keyword that sets up a condition in Python

**Logical operator**: An operator that connects multiple statements together and performs complex comparisons

**Modularity**: The ability to write code in separate components that work together and that can be reused for other programs

**Modulo**: An operator that returns the remainder when one number is divided by another

**Refactoring**: The process of restructuring code while maintaining its original functionality

**return**: A reserved keyword in Python that makes a function produce new results which are saved for later use

**Reusability**: The capability to define code once and using it many times without having to rewrite it

**Self-documenting code**: Code written in a way that is readable and makes its purpose clear

# Terms and definitions from the previous module

## A

**Argument**: Information given to a function in its parentheses

**Assignment**: The process of storing a value in a variable

**Attribute**: A value associated with an object or class which is referenced by name using dot notation

## C

**Cells**: The modular code input and output fields into which Jupyter Notebooks are partitioned

**Class**: An object’s data type that bundles data and functionality together

**Computer programming**: The process of giving instructions to a computer to perform an action or set of actions

## D

**Data type**: An attribute that describes a piece of data based on its values, its programming language, or the operations it can perform

**Dot notation**: How to access the methods and attributes that belong to an instance of a class

**Dynamic typing**: Variables that can point to objects of any data type

## E

**Explicit conversion**: The process of converting a data type of an object to a required data type

**Expression**: A combination of numbers, symbols, or other variables that produce a result when evaluated

## F

**Float**: A data type that represents numbers that contain decimals

## I

**Immutable data type**: A data type in which the values can never be altered or updated

**Implicit conversion**: The process Python uses to automatically convert one data type to another without user involvement

**Integer**: A data type used to represent whole numbers without fractions

## J

**Jupyter Notebook**: An open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text

## K

**Keyword**: A special word in a programming language that is reserved for a specific purpose and that can only be used for that purpose

## M

**Markdown**: A markup language that lets the user write formatted text in a coding environment or plain-text editor

**Method**: A function that belongs to a class and typically performs an action or operation

## N

**Naming conventions**: Consistent guidelines that describe the content, creation date, and version of a file in its name

**Naming restrictions**: Rules built into the syntax of the language itself that must be followed

## O

**Object**: An instance of a class; a fundamental building block of Python

**Object-oriented programming**: A programming system that is based around objects which can contain both data and code that manipulates that data

## P

**Programming languages**: The words and symbols used to write instructions for computers to follow

## S

**String**: A sequence of characters and punctuation that contains textual information

**Syntax**: The structure of code words, symbols, placement, and punctuation

## V

**Variable**: A named container which stores values in a reserved location in the computer’s memory

# Module 2 challenge

### 1.

Question 1

A data professional wants to define a function to calculate the square of a number. What code should they begin with?

1 / 1 point

**else find\_square(n):**

**if find\_square(n):**

**def find\_square(n):**

**return find\_square(n):**

Correct

### 2.

Question 2

Fill in the blank: A data professional can use the \_\_\_\_\_ keyword to make a Python function produce new results and save the results for later use.

1 / 1 point

**return**

**if**

**else**

**def**

Correct

### 3.

Question 3

What are best practices for writing clean code? Select all that apply.

1 / 1 point

Reusability

Correct

Redundancy

Modularity

Correct

Clarity

Correct

### 4.

Question 4

Fill in the blank: Refactoring is used to create \_\_\_\_\_, which is code that is written in a way that is readable and makes its purpose clear.

1 / 1 point

open source code

self-documenting code

branching code

immutable code

Correct

### 5.

Question 5

Fill in the blank: A data professional can add a \_\_\_\_\_ to the beginning of a function’s body to summarize the function’s behavior and explain its arguments and return values.

1 / 1 point

logical operator

comparator

conditional statement

docstring

Correct

### 6.

Question 6

What is the Python comparator for not equal to?

1 / 1 point

**==**

**<=**

**!=**

**>=**

Correct

### 7.

Question 7

A data professional writes the following code: **print(33 > 12 or 9 < 7)**. What result will Python display when they run the code?

1 / 1 point

True

False

Equal

Not equal

Correct

### 8.

Question 8

Fill in the blank: In Python, the \_\_\_\_\_ statement branches the execution based on a specific condition being true.

1 / 1 point

**else**

**then**

**if**

**elif**

Correct

### 9.

Question 9

In Python, when does an **else** statement execute a piece of code?

0 / 1 point

When the **if** statement contains numeric data

When the **if** statement contains a false condition

When the **if** statement contains text data

When the **if** statement contains a true condition

Incorrect

Review [the video about branching.](https://www.coursera.org/learn/get-started-with-python/lecture/6JsS8/use-if-elif-else-statements-to-make-decisions)

**Module 3:**

**What you’ve learned**

-Variables

-Data types

-Functions

-Write clean code

-Conditional statements

**What you’ll learned**

-Loops

-Strings

“Imposter syndrome is a very real thing. I think everybody experiences it, and so did I.”

# Instructions

All of the instructional videos with onscreen coding demonstrations have a corresponding follow-along guide that is available to you. The follow-along guide is an annotated Jupyter notebook organized to match the content from each module. It contains the same code shown in the videos for that module. This guide is provided for your reference; you do not need to add any text or run the code yourself. If you would like to run the code, you will need to run each cell sequentially for the code to function as intended.

In addition to content that is identical to what is covered in the videos, you’ll often find additional information throughout the guide to explain the purpose of each concept covered, why the code is written in a certain way, and tips for running the code.

The landing page for each follow-along notebook also provides information about data sources (when relevant) and tips on how to access and use these guides.

# Data dictionary

This module references [The Zen of Python](https://peps.python.org/pep-0020/), a widely known poem written by programmer Tim Peters.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the Lab Files dropdown menu at the top of the page, clicking into the /home/jovyan/work folder, selecting the relevant data file, and clicking Download.

## Set up a split screen

While watching the videos that follow this notebook, you may find it helpful to track the instructor’s progress by following along in your own Jupyter Notebook. It can be helpful to review the code notebook alongside the video, especially if you’re new to coding in Python.

To do so:

1. Open the video in one browser window.
2. Then, open the annotated follow-along guide in a separate window.
3. Arrange your screen so that the video and the follow-along guide are both visible.

A screenshot of a computer

Description automatically generated

## Tips for working with the follow-along guide

Follow these suggestions to enhance your learning experience:

* Reference the [Jupyter notebooks reading](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks) before starting if you need more information on working with the Jupyter notebooks.
* Watch for an in-video message to advise that the video you are viewing contains coding instruction and examples.
* Go to the section of the follow-along guide for the current module's content. The follow-along guide has different sections for each video included in the module's content. The in-video message will direct you to the relevant section in the guide for the specific video you are viewing.
* Follow along in the notebook as the instructor discusses the code.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 1. Introduction to while loops.

I'm ready to use the follow-along guide while watching this video.

**Loops**

A block of code used to carry out iterations

**Iterable**

An object that’s looped, or iterated, over

## Question

Fill in the blank: A loopis a block of code used to carry out \_\_\_\_\_.

Iterations

# Loops, break, and continue statements

You’ve learned about while loops in Python and have explored some examples. While loops are useful because they allow you to perform an action or evaluation repeatedly until a given condition or requirement is met, and then they stop. This is an important process in computer programming, not just in Python, but in most other languages too. Data professionals use while loops to process data, so it’s important for you to familiarize yourself with them as you grow your skills. This reading is a review of the fundamental concepts of while loops.

## While loop syntax

A while loop is a control structure that allows you to repeatedly execute a block of code for as long as a certain condition is true.

**Note**: The following code block is not interactive.

The basic syntax of a while loop is as follows:

1

2

while condition:

   # Code block to execute

The **condition** is a Boolean expression that is evaluated at the beginning of each iteration of the loop. If the condition is true, the code block executes. After the code block executes, the condition is evaluated again. This process continues until the condition is false, at which point the loop terminates and the program continues with the next statement after the loop.

Here is an example of a basic while loop:

1

2

3

4

x = 1

while x < 100:

   print(x)

   x = x\*2

RunReset

## Activity: While loops

## Instructions

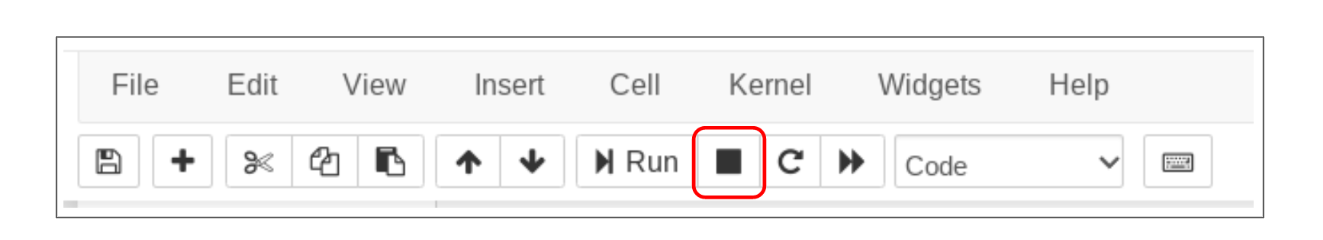
In this example, **x** equals one when the loop begins. Because **x** is less than 100, the program prints the value of **x**, then multiplies **x** by two. Then the condition is checked again, and because it is still **True**, the code inside the loop executes again. This process continues until **x** becomes 128, at which point the condition becomes **False** and the loop terminates.

## Infinite loops

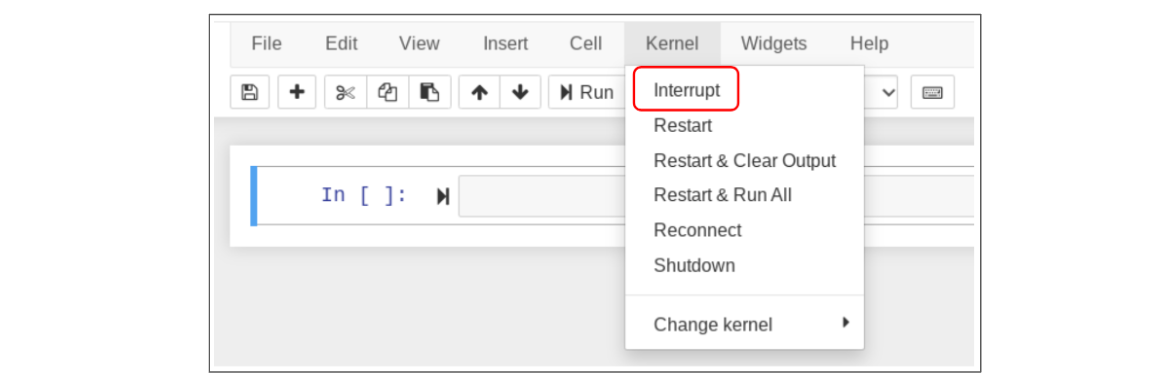
Be careful with while loops because if you make a mistake with your logic or syntax, it could result in an infinite loop that never terminates. In the previous example, if x = x\*2 were accidentally not indented to be in the body of the while loop, the loop would reach the print statement and cycle back to check the conditional statement, which of course would still be true because the value of x would never change from one.

If you get stuck in an infinite loop, don’t worry. You can break out of it by interrupting the kernel. There are several ways to do this:

1. Use the stop button in the menu at the top of the notebook.



2. Go to Kernel in the menu bar at the top of the notebook and select Interrupt from the drop-down menu.



3. While in command mode, press i twice.

## break & continue

It is possible to end a loop even if the conditional statement is still true. To do this, use a **break** statement.

Here’s an example:

1

2

3

4

5

6

7

8

x = 1

i = 0

while x < 100:

   if i == 5:

       break

   print(i, x)

   x = x\*2

   i += 1

RunReset

In this example, there is a variable **i** that acts as a counter. For each iteration of the loop, the program:

1. Checks if **x** is less than 100.
2. If it is, then the program checks if **i** equals five.
3. If it does, the loop terminates because of the break statement. Otherwise, it prints the values of both **i** and **x**, doubles the value of **x**, and increments the value of **i** by one.
4. Repeats until **x** ≥ 100 or **i** = 5. In this case, the loop breaks when **i** becomes 5.

It’s also possible to skip an iteration of the loop without executing the rest of the code inside the loop for the current iteration. To do this, use a **continue** statement.

Here’s an example:

1

2

3

4

5

6

7

i = 0

while i < 10:

    if i % 3 != 0:

        print(i)

        i += 1

        continue

    i += 1

RunReset

This example is a loop that prints all the numbers from zero through 9 that are not divisible by three. For each iteration of the loop, the program:

1. Checks if **i** is less than 10.
2. If it is, then the program uses the modulo operator to check if **i** is evenly divisible by three.
3. If it is not, then the program prints **i**, increments the value of **i** by one, and then cycles back to the beginning to check that **i** is less than 10. This happens because of the continue statement. The final **i += 1** does not execute, thus avoiding a double incrementation of **i**.
4. But if step 2 evaluates **i** as evenly divisible by three, nothing in the if block executes (so there’s no print statement) and **i** is incremented by one.
5. Repeats until **i** becomes 10.

## Key takeaways

A **while** loop allows you to repeatedly execute a block of code while a certain condition is true. You can use the **break** statement to exit the loop prematurely, and the **continue** statement to skip to the next iteration of the loop without executing the rest of the code in the current iteration.

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Exemplar: While loops

# Test your knowledge: While loops

### 1.

Question 1

Fill in the blank: A **while** loop instructs your computer to continuously execute your code based on the value of a \_\_\_\_\_.

1 / 1 point

condition

A **while** loop instructs your computer to continuously execute your code based on the value of a condition. The loop will keep iterating as long as the condition remains true.

### 2.

Question 2

A data professional wants to set up a **while** loop that will iterate as long as the variable **x** is less than 7. They assign the value 0 to the variable **x**. What code should they write next?

1 / 1 point

**while x < 7:**

Correct

The code **while x < 7:** sets a condition for a **while** loop stating that the variable **x** must be less than 7. The code begins with the distinguishing keyword **while**. And, like functions and other expressions that start a distinct code block, it ends with a colon. The while loop will iterate until the condition is false.

### 3.

Question 3

In Python, the keyword **break** lets you escape a loop without triggering any **else** statement that follows it in the loop.

1 / 1 point

True

In Python, the keyword **break** lets you escape a loop without triggering any **else** statement that follows it in the loop.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 2. Introduction to for loops.

I’m ready to use the follow-along guide while watching this video.

**for loop**

A piece of code that iterates over a sequence of values.

**range()**

A Python function that returns a sequence of numbers starting from zero, increments by 1 by default, and stops before the given number.

Range function

* 1. A range of numbers will start with the value 0 by default
  2. The list of numbers generated will be one less than the given value

## Question

In Python, what type of loop iterates over a sequence of values?

for loop

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 3. Loops with multiple range parameters.

I'm ready to use the follow-along guide while watching this video.

**Parameters of the range function**

-Start value

-Stop value

-Step value

Use **for loops** when there’s a sequence of elements that you want to iterate over.

Use **while loops** when you want to repeat an action until a Boolean condition changes.

Boolean are a data type that represents one of two possible states: **True** or **False**

## Question

Python’s **range()** function includes which of the following parameters? Select all that apply.

Stop value

Step value

Start value

# For loops

You’ve learned about for loops in Python and have explored some examples. For loops are like while loops, but instead of looping continuously until a condition is met, for loops iterate over each element of an iterable sequence, allowing you to perform an action or evaluation with each iteration. This is an important process in computer programming, not just in Python, but in most other languages too. Data professionals use for loops to process data, so it’s important for you to familiarize yourself with them as you grow your skills. This reading is a review of the fundamental concepts of for loops.

## For loop syntax

A for loop is a control structure that allows you to execute a block of code the same number of times as there are elements in an iterable sequence. You’ll learn more about iterable sequences later in this course, but some examples of iterable data types include:

Strings: **‘chimichurri’**

Lists: **[1, 2, 3, 4, 5, 6]**

Tuples: **(1, 2, 3, 4, 5)**

Dictionaries: **{‘Name’: ‘Anita’, ‘Age’: 77}**

Sets: **{1, 4, 14, 33}**

**Note**: The following code block is not interactive.

The basic syntax of a for loop is as follows:

1

2

for item in iterable\_sequence:

   # Code block to be executed for each value in iterable\_sequence

The **iterable\_sequence** variable can be any iterable data type, and **item** is a variable whose name is arbitrary —you decide it. However, there are some conventions that you’ll encounter when naming this variable. For example, if you’re iterating over characters in a string, you’ll frequently encounter the variable **char**. If you’re iterating over a list of numbers, you’ll find **n** or **num**. It’s helpful to give this variable a name so readers of your code understand what kind of information is being looped over. So, for a variable called **names** that contains a list of people’s names, you might write: **for name in names:**.

A note about the behavior of this variable — its value is reassigned for each iteration of the loop, and it persists even after the loop terminates.

Here’s an example:

1

2

3

4

5

6

num = 5

y = [1, 2, 3]

for num in y:

   print(num)

print(num)

RunReset

Notice that **num** exists as a variable before the for loop begins. The for loop’s first iteration reassigns its value with that of the first element in the sequence. This reassignment occurs with each iteration of the loop. When the loop terminates, the variable persists, and it contains the value it had after the final iteration of the loop.

## The range() function

The **for** loop allows you to create a loop that performs exactly the number of iterations needed for the data structure you’re looping over. In other words, whether your iterable sequence contains two, 1,000, or a million elements, you can use the same syntax and don’t have to specify the number of iterations you want. However, sometimes you need to perform a task a set number of times, but you don’t already have an iterable object to loop over. Or, sometimes you need to generate a known, regular sequence of numbers. This is where the **range()** function is useful.

The **range()** function is a function that takes three arguments: start, stop, step. Its output is an object belonging to the range class. If you only include one argument, it will be interpreted as the stop value. The start and step values by default will be zero and one, respectively. If you include two arguments, they will be interpreted as the start and stop values (again, with step being one by default). Note that the stop value is not included in the range that is returned.

Here are some examples:

**A.**

1

2

for i in range(3):

   print(i)

RunReset

**B.**

1

2

for n in range(2, 5):

   print(n)

RunReset

**C.**

1

2

for even\_num in range(2, 11, 2):

   print(even\_num)

RunReset

You’ll find that the **range()** function is very useful, for example, when creating numbered lists or performing operations on certain indices of an object. You’ll learn more about indexing later.

## Nested loops

Sometimes you’ll need to extract information from nested structures—for example, from a list of lists. One way of doing this is by using nested loops. A nested loop is a loop inside of another loop. You can have an infinite number of nested loops, but it becomes more confusing to read and understand the more nested loops you add.

Here’s an example of one loop nested in another:

1

2

3

4

5

students = [['Igor', 'Sokolov'], ['Riko', 'Miyazaki'], ['Tuva', 'Johansen']]

for student in students:

   for name in student:

       print(name)

   print()

RunReset

In this example, the students variable contains a list of three lists. Each inner list contains two elements: a given name and a surname. The first for loop iterates over the inner lists. The second (nested) for loop iterates over each name in each inner list and prints the name. After each iteration of the outer loop, the program uses an empty print statement to print a new line.

## Key takeaways

A **for** loop allows you to execute a block of code the same number of times as there are elements in an iterable sequence. The **range()** function is useful for creating a defined iterable sequence. And nested loops are loops within loops that give you even greater power and control over how your code may execute. These are powerful tools that can be used in many different ways to solve a variety of problems that you’ll encounter as a data professional.

# **Instructions**

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

The activity imports a file called **ada\_c2\_labs.py**, which is used to generate test cases so you can verify whether your code successfully completes the given tasks. If you are working in a notebook on the Coursera platform, you do **not** have to do anything. If you are working in a notebook that is not on the Coursera platform, you will need to download this file to run the test cases.

You can [download ada\_c2\_labs.py](https://drive.google.com/file/d/1PEeoEYVzu0M9snRWKeXPWyQx1mT3PQ5Y/view?usp=sharing) here or from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: For loops

### 1.

Question 1

A data professional can use a **for** loop to perform which of the following tasks?

1 / 1 point

To iterate over a series of numbers

A data professional can use a **for** loop to iterate over a series of numbers. In Python, a **for** loop is a piece of code that iterates over a sequence of values, such as numbers in a list or characters in a string.

### 2.

Question 2

A data professional wants to set up a **for** loop. They write the following code: **for x in range(3):** . What values will the variable **x** take?

1 / 1 point

0, 1, and 2

### 3.

Question 3

What parameter of Python’s **range()** function specifies the size of the increments in a sequence of numbers?

1 / 1 point

Step value

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 4. Work with strings.

I’m ready to use the follow-along guide while watching this video.

**Concatenate**

To link or join together

**Escape character**

A character that changes the typical behaviour of the characters that follow it

## Question

Fill in the blank: \_\_\_\_\_ strings makes a single longer string from two or more shorter ones.

Concatenating

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 5. String slicing.

I’m ready to use the follow-along guide while watching this video.

**Indexing**

A way to refer to the individual items within an iterable by their relative position.

**Indexing can be used on:**

-strings

-lists

-tuples

-most other iterable data types

index()

A string method that outputs the index number of a character in a string.

## Question

Python uses one-based indexing.

False

**String slice**

A portion of a string, also known as a substring, that can contain more than one character.

To check whether or not a substring is contained in a string, use the keyword in.

# String indexing and slicing

As you know, strings are an important class of data because they represent text. Data professionals encounter strings all the time, so it’s important to become familiar with different ways of manipulating and working with them. This reading will review the string formatting techniques you’ve learned and also introduce you to regular expressions.

## String formatting

Indexing refers to accessing a single element of a sequence by its position. In Python, the first element of any sequence has an **index of zero.** This means Python uses zero-based indexing. Numerous other programming languages also use zero-based indexing, but not all of them do. Some languages use one-based indexing, such as R, Julia, and SAS.

Use square brackets to perform indexing. Here are some examples:

1

2

3

4

5

6

7

my\_string = 'Mississippi half-step'

print(my\_string[0])

my\_list = [1, 'unladen', 'swallow']

print(my\_list[1])

print(my\_list[-1])

RunReset

In these examples, there are two sequence variables: a string and a list. Indexing is used to access the character at index zero of the string, which is its first character—M. The list is selected at index one, which contains the word “unladen.” The list is also selected at its final position using negative indexing.

**Note:** If you try to select an index that is out of range of what the object contains, you’ll get an IndexError.

1

2

my\_list = [1, 'unladen', 'swallow']

my\_list[3]

RunReset

## Slicing

Slicing refers to accessing a range of elements from a sequence. Use square brackets containing two indices separated by a colon.

Here are some examples:

1

2

3

new\_string = 'pining for the fjords'

print(new\_string[0:3])

print(new\_string[:3])

RunReset

These two examples, each with slightly different syntax, are being used to produce the same result. Notice two things: (1) the resulting slice includes the starting index and excludes the ending index; (2) when the starting index is omitted it’s implied to be zero, as shown in the second print line.

The process follows the same logic when the ending index is omitted:

1

2

3

4

new\_string = 'pining for the fjords'

print(new\_string[6:21])

print(new\_string[6:])

print(len(new\_string))

RunReset

Again, there are two statements that are syntactically different but still produce the same substring. When the ending index is omitted, its implied value is the length of the sequence.

Finally, the code will throw an IndexError if you try to index a sequence at an index number outside the scope of the elements; this is not the case for slicing.

For example:

1

2

new\_string = 'pining for the fjords'

print(new\_string[6:100])

RunReset

Although the ending index was 100—far beyond the scope of the indices in the string—the computer returned a substring that ended with the string’s final element.

## Key takeaways

Indexing and slicing are powerful tools in Python that allow you to access specific elements or parts of a sequence. Both indexing and slicing use square brackets. Remember that in a slice the starting index is inclusive and the stopping index is exclusive, and that negative indices count from the end of the sequence. With these tools, you can manipulate strings and other iterable sequences to perform a wide variety of operations, making you a more proficient data professional.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Loops and Strings](https://www.coursera.org/learn/get-started-with-python/ungradedLab/1r1mX/annotated-follow-along-guide-loops-and-strings) in a new browser and navigate to Section 6. Format strings.

I’m ready to use the follow-along guide while watching this video.

format()

Formats and inserts specific substrings into designated places.

# String formatting and regular expressions

As you’ve learned, strings are a crucial class of data because they represent textual information. Data professionals encounter strings all the time, so it’s important to become familiar with different ways of manipulating and working with them. This reading will review the string formatting techniques you’ve learned, and also introduce you to regular expressions.

## String formatting

String formatting uses the **format()** method, which belongs to the string class. This method formats and inserts specific substrings into designated places within a larger string. It’s useful when you have reusable template text into which you want to insert specific changeable values, for example. The **format()** method is also useful when assigning the strings used to label charts and graphs you make.

Here’s an example:

1

2

3

4

5

x = 'values'

y = 100

print('''String formatting lets you insert {} into strings.

They can even be numbers, like {}.'''.format(x, y))

RunReset

Notice the syntax. The **format()** function inserts its arguments into the braces within the string that it’s attached to. The order of insertion follows the order of the arguments. Also, this example includes a helpful trick. Sometimes you’ll encounter a very long string. Many editors will allow the string to keep extending to the right on a single line. This is impractical unless you have a very wide monitor, but 79 characters is a conventional maximum length for a single line of Python code. Enclosing your string in triple quotes lets you break the string over multiple lines.

The **format()** function can also insert values into braces using explicitly assigned keyword names, which allow you to mix up the order of the function’s arguments without changing the order of their insertion into the final string.

For example:

1

2

3

var\_a = 'A'

var\_b = 'B'

print('{a}, {b}'.format(b=var\_b, a=var\_a))

RunReset

Because the arguments were named, it didn’t matter that they were entered with **var\_b** first and **var\_a** last; they still were inserted into the string in the order specified.

You can also include the arguments’ index numbers within the braces to indicate which arguments get inserted in specific spots:

1

2

3

4

var\_a = 'A'

var\_b = 'B'

print('{1}, {0}'.format(var\_a, var\_b))

print('{0}, {1}'.format(var\_a, var\_b))

RunReset

You can have as many arguments as you want:

1

print('{}, {}, {}, {}, {}, {} ...'.format(1, 2, 3, 4, 5, 6))

RunReset

And you can repeat arguments’ indices:

1

print('{0}{1}{0}'.format('abra', 'cad'))

RunReset

The string **format()** method is a versatile and convenient way to take values that are stored in different variables and insert them into a string.

## **Literal string interpolation (f-strings)**

Another string formatting technique that you’ll often encounter when using Python version 3.6+ is literal string interpolation, also known as f-strings. F-strings further minimize the syntax required to embed expressions into strings. They’re called f-strings because the expressions always begin with f (or F—they’re the same).

For example:

1

2

3

4

5

var\_a = 1

var\_b = 2

print(f'{var\_a} + {var\_b}')

print(f'{var\_a + var\_b}')

print(f'var\_a = {var\_a} \nvar\_b = {var\_b}')

RunReset

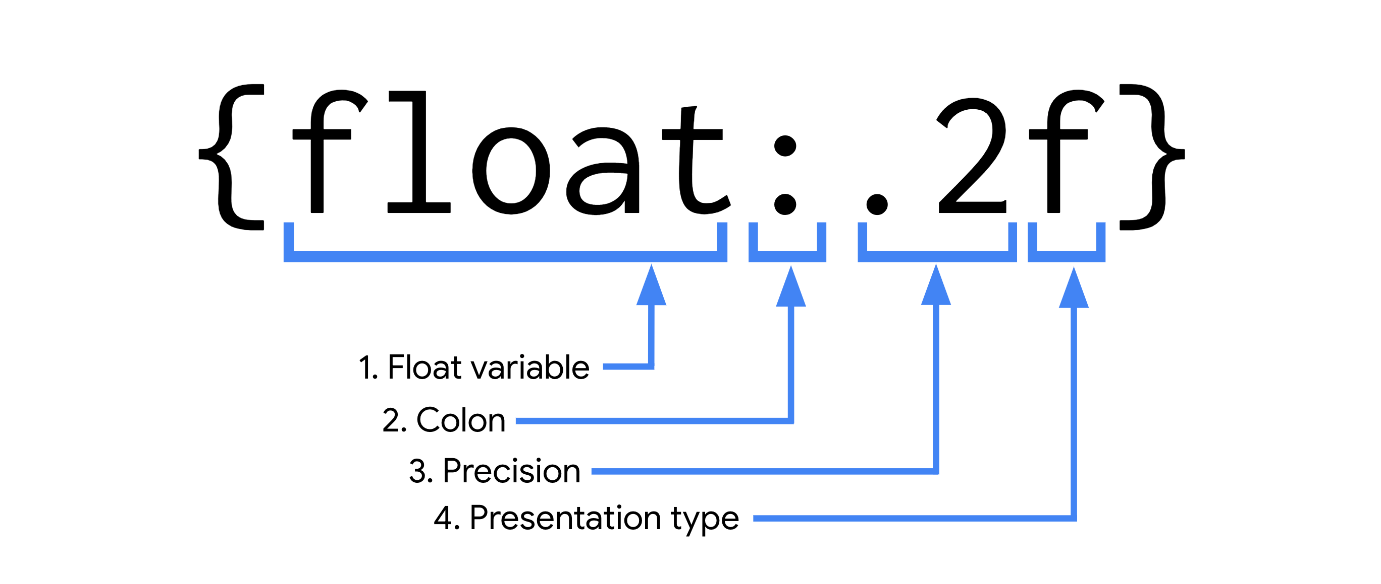
In these examples, the braces still function as the way to indicate where values should be inserted into the string, but they allow you to make the insertion directly, without having to call the **format()** method.

## **Float formatting options**

In addition to inserting expressions into strings, string formatting can format their appearance. There are too many options to list here, but the [Python string documentation](https://docs.python.org/3/library/string.html) is a good place to review these techniques. Here are some of the most useful.

### **To use these options, build your expression within braces as follows.**

1. The float variable is what’s being formatted
2. A colon (**:**) separates what’s being formatted from the syntax used to format it
3. **. number** indicates the desired precision
4. A letter indicates the presentation type



Example:

1

2

num = 1000.987123

f'{num:.2f}'

RunReset

This example uses the **f** presentation type to specify that the number contained in the **num** variable should be rounded to two places beyond the decimal.

Here are some of the most common presentation types:

| **Type** | **Meaning** |
| --- | --- |
| **'e'** | Scientific notation. For a given precision **p**, formats the number in scientific notation with the letter ‘**e**’ separating the coefficient from the exponent. The coefficient has one digit before and **p** digits after the decimal point, for a total of **p + 1** significant digits. With no precision given, **e** uses a precision of **6** digits after the decimal point for [float](https://docs.python.org/3/library/functions.html#float), and shows all coefficient digits for [decimal](https://docs.python.org/3/library/decimal.html#decimal.Decimal). |
| **'f'** | Fixed-point notation. For a given precision **p**, formats the number as a decimal number with exactly **p** digits following the decimal point. |
| **'%'** | Percentage. Multiplies the number by 100 and displays in fixed (**'f'**) format, followed by a percent sign. |

Here are some examples:

1

2

3

4

5

num = 1000.987123

print(f'{num:.3e}')

decimal = 0.2497856

print(f'{decimal:.4%}')

RunReset

## **String methods**

As one of the primary object classes in Python, strings have many built-in methods designed to facilitate working with them. There are too many of these methods to cover all of them here in depth, but some of the most useful include:

**str.count(sub[, start[, end]])**

Return the number of non-overlapping occurrences of substring **sub** in the range **[start , end]**.

1

2

3

4

my\_string = 'Happy birthday'

print(my\_string.count('y'))

print(my\_string.count('y', 2, 7))

RunReset

### str.find(sub)

Return the lowest index in the string where substring **sub** is found. Return -1 if **sub** is not found.

1

2

3

my\_string = 'Happy birthday'

my\_string.find('birth')

RunReset

### str.join()

Return a string which is the concatenation of the strings in iterable. The separator between elements is the string providing this method.

1

2

3

4

separator\_string = ' '

iterable\_of\_strings = ['Happy', 'birthday', 'to', 'you']

separator\_string.join(iterable\_of\_strings)

RunReset

### str.partition(sep)

Split the string at the first occurrence of **sep** , and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings.

1

2

3

my\_string = 'https://www.google.com/'

my\_string.partition('.')

RunReset

### str.replace(old, new[, count])

Return a copy of the string with all occurrences of substring **old** replaced by **new**. If the optional argument **count** is given, only the first **count** occurrences are replaced.

1

2

3

my\_string = 'https://www.google.com/'

my\_string.replace('google', 'youtube')

RunReset

### str.split([sep])

Return a list of the words in the string, using **sep** (optional) as the delimiter string. If no **sep** is given, whitespace characters are used as the delimiter. Any number of consecutive whitespaces would indicate a split point, so **' '** (a single whitespace) would split the same way as **' '** (two or more whitespaces).

1

2

3

my\_string = 'Do you know the muffin man?'

my\_string.split()

RunReset

Note that some of these methods have additional optional parameters. This reading covers only the most rudimentary ones. Reference the full [string methods documentation](https://docs.python.org/2/library/stdtypes.html#string-methods) for more information on these functions and other methods not included here.

## Regular expressions

Regular expressions, also known as regex, refer to techniques that advanced data professionals use to modify and process string data. This certificate program will not require you to use regular expressions in your work, but it’s important for you to be aware of the concept. As always, you’re encouraged to explore regular expressions on your own.

Regex works by matching patterns in Python. It allows you to search for specific patterns of text within a string of text. Regex is used extensively in web scraping, text processing and cleaning, and data analysis.

The first step in working with regular expressions is to import the **re** module. This module provides the tools necessary for working with regular expressions. Once you have imported the module, you can start working with regular expressions.

**Note**: The following code block is not interactive.

The basic syntax for a regular expression is:

1

2

3

4

5

import re

pattern = 'regex\_pattern'

match = re.search(pattern, string)

Here is a basic example:

1

2

3

4

5

import re

my\_string = 'Three sad tigers swallowed wheat in a wheat field'

re.search('wall', my\_string)

RunReset

This example returns a match object that contains information about the search. In this case, it tells you that the substring **‘wall’** does occur in the string from indices 18–22.

Regex is especially useful because it allows you a very high degree of customization when performing your searches.

Here’s another example:

1

2

3

4

5

import re

my\_string = 'Three sad tigers swallowed wheat in a wheat field'

re.search('[bms]ad', my\_string)

RunReset

This example will search for “bad,” “mad,” and “sad.” Again, these are very basic examples.

Regex has a large catalogue of special expressions that let you search for substrings that will only match if, for example, they are followed by certain characters, or if they don’t contain a certain set of characters. It can get very complex. Depending on the work you do as a data professional, you may find yourself exploring regular expressions to analyze and process your data.

## Key takeaways

String formatting is the process of inserting specific substrings into designated places within a larger string. Often, the inserted substrings get processed and formatted a certain way. There are multiple ways of using string formatting to help you process strings. These include the **format()** method, literal string interpolations—or f-strings—and regular expressions, also known as regex. The methods you use will depend on what your data demands and your own personal preferences, but it’s important to be familiar with the most common techniques used by data professionals.

# Instructions

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/teach/google-beginning-your-python-journey/authoringBranch~wJKPci8PEe-YcA77IVc2yQ/content/item/supplement/C2ll0).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: Strings

### 1.

Question 1

In Python, what is the term for the portion of a string that can contain more than one character? Select all that apply.

Substring

String slice

### 2.

Question 2

If you’re reading from left to right, what is the index of the first character in a string?

1 / 1 point

0

### 3.

Question 3

A data professional wants to insert specific substrings in a larger string. What method can they use to do so?

**format()**

Review

-While loops

-For loops

-Strings

# Glossary terms from module 3

# Terms and definitions from Course 2, Module 3

**break:** A keyword that lets a user escape a loop without triggering any ELSE statement that follows it in the loop

**Concatenate**: To link or join together

**Escape character**: A character that changes the typical behavior of the characters that follow it

**For loop**: A piece of code that iterates over a sequence of values

**format()**: A string method that formats and inserts specific substrings into designated places within a larger string

**index()**: A string method that outputs the index number of a character in a string

**Indexing**: A way to refer to the individual items within an iterable by their relative position

**Iterable**: An object that’s looped, or iterated, over

**Iteration**: The repeated execution of a set of statements, where one iteration is the single execution of a block of code

**Loop**: A block of code used to carry out iterations

**range()**: A Python function that returns a sequence of numbers starting from zero, increments by 1 by default, and stops before the given number

**String slice**: A portion of a string that can contain more than one character; also referred to as a substring

**While** **loop**: A loop that instructs the computer to continuously execute the code based on the value of a condition

# Terms and definitions from previous modules

## A

**Algorithm**: A set of instructions for solving a problem or accomplishing a task

**Argument**: Information given to a function in its parentheses

**Assignment**: The process of storing a value in a variable

**Attribute**: A value associated with an object or class which is referenced by name using dot notation

## B

**Boolean**: A data type that has only two possible values, usually true or false

**Branching**: The ability of a program to alter its execution sequence

## C

**Cells**: The modular code input and output fields into which Jupyter Notebooks are partitioned

**Class**: An object’s data type that bundles data and functionality together

**Comparator**: An operator that compares two values and produces Boolean values (True/False)

**Computer programming**: The process of giving instructions to a computer to perform an action or set of actions

## D

**Data type**: An attribute that describes a piece of data based on its values, its programming language, or the operations it can perform

**def**: A keyword that defines a function at the start of the function block

**Docstring**: A string at the beginning of a function’s body that summarizes the function’s behavior and explains its arguments and return values

**Dot notation**: How to access the methods and attributes that belong to an instance of a class

**Dynamic typing**: Variables that can point to objects of any data type

## E

**elif**: A reserved keyword that executes subsequent conditions when the previous conditions are not true

**else**: A reserved keyword that executes when preceding conditions evaluate as False

**Explicit conversion**: The process of converting a data type of an object to a required data type

**Expression**: A combination of numbers, symbols, or other variables that produce a result when evaluated

## F

**Float**: A data type that represents numbers that contain decimals

**Function**: A body of reusable code for performing specific processes or tasks

## I

**if**: A reserved keyword that sets up a condition in Python

**Immutable data type**: A data type in which the values can never be altered or updated

**Implicit conversion**: The process Python uses to automatically convert one data type to another without user involvement

**Integer**: A data type used to represent whole numbers without fractions

## J

**Jupyter Notebook**: An open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text

## K

**Keyword**: A special word in a programming language that is reserved for a specific purpose and that can only be used for that purpose

## L

**Logical operator**: An operator that connects multiple statements together and performs complex comparisons

## M

**Markdown**: A markup language that lets a user write formatted text in a coding environment or plain-text editor

**Method**: A function that belongs to a class and typically performs an action or operation

**Modularity**: The ability to write code in separate components that work together and that can be reused for other programs

**Modulo**: An operator that returns the remainder when one number is divided by another

## N

**Naming conventions**: Consistent guidelines that describe the content, creation date, and version of a file in its name

**Naming restrictions**: Rules built into the syntax of a programming language

## O

**Object**: An instance of a class; a fundamental building block of Python

**Object-oriented programming**: A programming system that is based around objects which can contain both data and code that manipulates that data

## P

**Programming languages**: The words and symbols used to write instructions for computers to follow

## R

**Refactoring**: The process of restructuring code while maintaining its original functionality

**return**: A reserved keyword in Python that makes a function produce new results which are saved for later use

**Reusability**: The capability to define code once and using it many times without having to rewrite it

## S

**Self-documenting code**: Code written in a way that is readable and makes its purpose clear

**String**: A sequence of characters and punctuation that contains textual information

**Syntax**: The structure of code words, symbols, placement, and punctuation

## V

**Variable**: A named container which stores values in a reserved location in the computer’s memory

### 1.

Question 1

A data professional can use a **while** loop to perform which of the following tasks?

1 / 1 point

To repeat a specific block of code until a condition is no longer met

### 2.

Question 2

Fill in the blank: The Python **range()** function returns a sequence of numbers starting from zero; then increments by \_\_\_\_\_, by default; then stops before the given number.

one

### 3.

Question 3

What Python code instructs the computer to loop through values from 750 to 850 (inclusive)?

**for x in range(750, 851):**

### 4.

Question 4

A data professional wants to set up a **for** loop. They write the following code: **for x in range(100, 501, 20):** . What is the step value of the **range()** function?

20

### 5.

Question 5

What Python code can a data professional use to concatenate the strings **'air'** and **'plane'**?

**‘air’ + ‘plane’**

### 6.

Question 6

Fill in the blank: In Python, the **index()** method interprets a string as a \_\_\_\_\_.

sequence of characters

### 7.

Question 7

A data professional assigns the string **'palm and pine'** to the variable **trees**. What Python code can they use to find the index of the character **'m'**?

**trees.index(‘m’)**

### 8.

Question 8

A data professional assigns the string **'penguin'** to the variable **animal**. What Python code will return the slice **'pen'**?

**animal[ :3]**

### 9.

Question 9

Fill in the blank: In Python, a data professional can use the \_\_\_\_\_ method to insert specific substrings in a larger string.

format()

**Module 4:**

**What you’ve learned**

-Variables

-Data types

-Functions

-Operators

-Write clean code

-Conditional statements

-Loops

**Data structures**

Collections of data values or objects that contain different data types

**What you’ll learn**

-List

-Tuples

-Dictionaries

-Sets

-Arrays

-NumPy

-pandas

# Instructions

All of the instructional videos with onscreen coding demonstrations have a corresponding follow-along guide that is available to you. The follow-along guide is an annotated Jupyter notebook organized to match the content from each module. It contains the same code shown in the videos for that module. This guide is provided for your reference; you do not need to add any text or run the code yourself. If you would like to run the code, you will need to run each cell sequentially for the code to function as intended.

In addition to content that is identical to what is covered in the videos, you’ll often find additional information throughout the guide to explain the purpose of each concept covered, why the code is written in a certain way, and tips for running the code.

The landing page for each follow-along notebook also provides information about data sources (when relevant) and tips on how to access and use these guides.

# Data dictionary

This notebook uses a file called **train.csv**, which contains data from the [Titanic dataset on Kaggle](https://www.kaggle.com/competitions/titanic/data). Note that it is not a complete dataset, but the training component of a full dataset that has been split into training/testing sets.

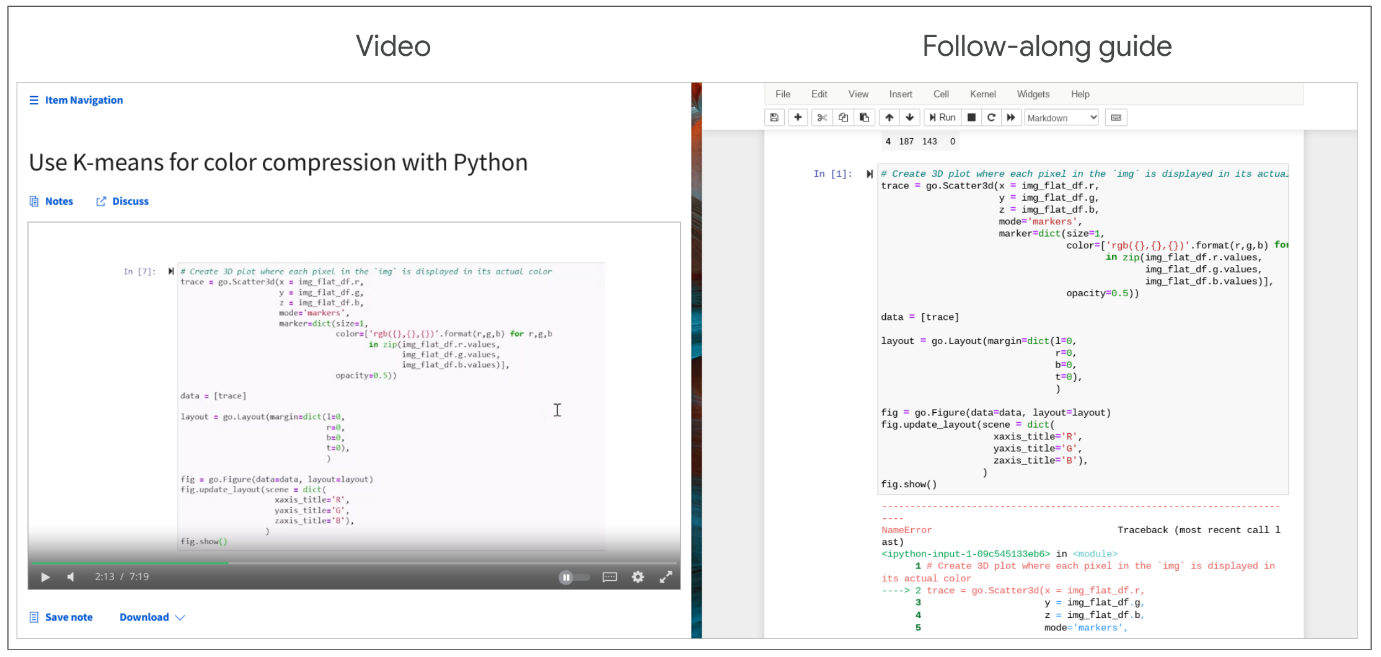
Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

## Set up a split screen

While watching the videos that follow this notebook, you may find it helpful to track the instructor’s progress by following along in your own Jupyter Notebook. It can be helpful to review the code notebook alongside the video, especially if you’re new to coding in Python.

To do so:

1. Open the video in one browser window.
2. Then, open the annotated follow-along guide in a separate window.
3. Arrange your screen so that the video and the follow-along guide are both visible.



## Tips for working with the follow-along guide

Follow these suggestions to enhance your learning experience:

* Reference the [Jupyter notebooks reading](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks) before starting if you need more information on working with the Jupyter notebooks.
* Watch for an in-video message to advise that the video you are viewing contains coding instruction and examples.
* Go to the section of the follow-along guide for the current module's content. The follow-along guide has different sections for each video included in the module's content. The in-video message will direct you to the relevant section in the guide for the specific video you are viewing.
* Follow along in the notebook as the instructor discusses the code.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 1. Introduction to lists.

I’m ready to use the follow-along guide while watching this video.

**Data structure**

A collection of data values or objects that contain different data types

**List**

A data structure that helps store and manipulate an ordered collection of items.

**Sequence**

A positionally ordered collection of items

**Mutability**

The ability to change the internal state of s data structure.

**Immutability**

A data structure or element’s values can never be altered or updated.

## Question

Fill in the blank: Mutability refers to the ability to \_\_\_\_\_ the internal state of a data structure.

change

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 2. Modify the contents of a list.

I’m ready to use the follow-along guide while watching this video.

**append()**

Method that adds an element to the end of a list

**insert()**

Function that takes an index as the first parameter and an element as the second parameter, then inserts the element into a list at the given index.

**remove()**

A method that removes an element from a list

**pop()**

A method that extracts an element from a list by removing it at a given index.

# Reference guide: Lists

You’ve been learning that lists are important data structures in Python. A list is a data structure that helps store and manipulate an ordered collection of items. These items can be of any data type such as integers, floats, strings, and even other lists. Because they are so versatile, data professionals and all Python programmers use lists every day, so it’s important to be familiar with how they work. This reading is a reference guide for lists designed to help you as you learn Python.

## **Save this course item**

You may want to save a copy of this guide for future reference. You can use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the following link and select “Use Template.”

Reference guide: [Lists](https://docs.google.com/document/d/1yZoljX91vuOG2dXAekx0lf3w1DxX1z1NyMtcUSdYEzQ/template/preview#heading=h.387z4nt1u8b3)

OR

If you don’t have a Google account, you can download the item directly from the attachment below.

## Create a list

There are two main ways to create lists in Python:

* Square brackets: **[]**
* The list function: **list()**

When instantiating a list using brackets, separate each element with a comma.

For example, the following code creates a list of strings:

1

2

list\_a = ['olive', 'palm', 'coconut']

print(list\_a)

RunReset

You can also create a list of integers:

1

2

list\_b = [8, 6, 7, 5, 3, 0, 8]

print(list\_b)

RunReset

Or a list of mixed data types:

1

2

list\_c = ['Abidjan', 14.2, [1, 2, None], 'Zagreb']

print(list\_c)

RunReset

To create an empty list, use empty brackets or the **list()** function:

1

2

empty\_list\_1 = []

empty\_list\_2 = list()

RunReset

## Indexing and slicing

Just as with strings, you can access elements in a list using indexing and slicing. The first element of a list has index zero, the second element has index one, and so on. Use square brackets to index:

1

2

phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']

print(phrase[1])

RunReset

You can also use negative indices to access items from the end of a list:

1

2

phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']

print(phrase[-1])

RunReset

Use slicing to extract a sublist. To slice, use square brackets containing a range of indices separated by a colon:

1

2

phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']

print(phrase[1:4])

RunReset

Notice that this code returned a sublist containing the elements at indices one, two, and three of phrase. The ending index of the slice is not included.

Omitting the starting index in a slice implies an index of zero, and omitting the ending index implies an index of **len(my\_list)**:

1

2

3

phrase = ['Astra', 'inclinant', 'sed', 'non', 'obligant']

print(phrase[:3])

print(phrase[3:])

RunReset

## List mutability

Lists are mutable, which means that you can change their contents after they are created. You can change an individual item in a list by specifying its index and assigning a new value to it. For example:

1

2

3

my\_list = ['Macduff', 'Malcolm', 'Duncan', 'Banquo']

my\_list[2] = 'Macbeth'

print(my\_list)

RunReset

You can even change a slice of a list using the same logic. The slice can be of any length. The elements in the new list will be inserted in place of the indicated slice:

1

2

3

my\_list = ['Macduff', 'Malcolm', 'Macbeth', 'Banquo']

my\_list[1:3] = [1, 2, 3, 4]

print(my\_list)

RunReset

## List operations

Lists can be combined using the addition operator (**+**):

1

2

3

num\_list = [1, 2, 3]

char\_list = ['a', 'b', 'c']

num\_list + char\_list

RunReset

They can also be multiplied using the multiplication operator (**\***):

1

2

list\_a = ['a', 'b', 'c']

list\_a \* 2

RunReset

But they cannot be subtracted or divided.

You can check whether a value is contained in a list by using the **in** operator:

1

2

3

num\_list = [2, 4, 6]

print(5 in num\_list)

print(5 not in num\_list)

RunReset

## List methods

Lists are a core Python class. As you’ve learned, classes package data together with tools to work with it. Methods are functions that belong to a class. Lists have a number of built-in methods that are very useful.

## append()

Add an element to the end of a list:

1

2

3

4

my\_list = [0, 1, 1, 2, 3]

variable = 5

my\_list.append(variable)

print(my\_list)

RunReset

## insert()

Insert an element at a given position:

1

2

3

my\_list = ['a', 'b', 'd']

my\_list.insert(2, 'c')

print(my\_list)

RunReset

## remove()

Remove the first occurrence of an item:

1

2

3

my\_list = ['a', 'b', 'd', 'a']

my\_list.remove('a')

print(my\_list)

RunReset

## pop()

Remove the item at the given position in the list, and return it. If no index is specified, **pop()** removes and returns the last item in the list:

1

2

3

my\_list = ['a', 'b', 'c']

print(my\_list.pop())

print(my\_list)

RunReset

## clear()

Remove all items:

1

2

3

my\_list = ['a', 'b', 'c']

my\_list.clear()

print(my\_list)

RunReset

## index()

Return the index of the first occurrence of an item in the list:

1

2

my\_list = ['a', 'b', 'c', 'a']

my\_list.index('a')

RunReset

## count()

Return the number of times an item occurs in the list:

1

2

my\_list = ['a', 'b', 'c', 'a']

my\_list.count('a')

RunReset

## sort()

Sorts the list ascending by default. You can also make a function to decide the sorting criteria:

1

2

3

4

5

6

char\_list = ['b', 'c', 'a']

num\_list = [2, 3, 1]

char\_list.sort()

num\_list.sort(reverse=True)

print(char\_list)

print(num\_list)

RunReset

## **Additional resources**

* For more information about lists, refer to [An Informal Introduction to Python: Lists](https://docs.python.org/3/tutorial/introduction.html#lists).
* For more list methods, refer to [Data Structures: More on Lists](https://docs.python.org/3/tutorial/datastructures.html).

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 3. Introduction to tuples.

I’m ready to use the follow-along guide while watching this video.

**Tuple**

An immutable sequence that can contain elements of any data type

Tuples are expressed with parentheses or the tuple() function.

**tuple()**

Function that transforms input into tuples.

## Question

Atuple is an immutable sequence that can contain elements of any data type

True

# Compare lists, strings, and tuples

You’ve now learned about some of Python’s core iterable sequence data structures, including strings, lists, and tuples. These structures share many similarities, but there are some key differences between them. Data professionals must often decide which data structures work best to solve a particular problem, so understanding the relationship between these classes can help you make informed decisions in your work. This reading is a guide to the similarities and differences between strings, lists, and tuples.

## Strings

### **Syntax/instantiation**

**Note**: The following code block is not interactive.

* Single, double, or triple quotes:

1

2

3

4

5

6

7

8

empty\_str = ''

my\_string1 = 'minerals'

my\_string2 = "martin"

my\_string3 = """

marathon

golfcart

"""

**Note**: Using triple quotes to write a string over multiple lines will insert newlines (**\n**).

1

2

3

4

5

6

my\_string3 = """

marathon

golfcart

"""

my\_string3

RunReset

* The **str()** function can be used for instantiation and conversion.

**Note**: The following code block is not interactive.

1

2

 empty\_str = str()

my\_string = str(125)

### **Content**

* Strings can contain any character—letters, numbers, punctuation marks, spaces—but everything between the opening and closing quotation marks is part of the same single string.

### **Mutability**

* Strings are **immutable**. This means that once a string is created, it cannot be modified. Any operation that appears to modify a string actually creates a new string object.

### **Usage**

* Strings are most commonly used to represent text data.

### **Methods**

The Python **string** class comes packed with many useful methods to manipulate the data contained in strings. For more information on these methods, refer to [Common String Operations](https://docs.python.org/3/library/string.html) in the Python documentation.

## Lists

### **Syntax/instantiation**

* Brackets, with each element separated by a comma:

**Note**: The following code block is not interactive.

1

2

empty\_list = []

my\_list = [1, 2, 3, 4, 5]

* The **list()** function can be used for instantiation and conversion. Note that this function only works on iterable data types.

1

2

print(list('rocks'))

print(list(('stones', 'water', 'underground')))

RunReset

### **Content**

* Lists can contain any data type, and in any combination. So, a single list can contain strings, integers, floats, tuples, dictionaries, and other lists.

**Note**: The following code block is not interactive.

1

my\_list = [1, 2, 1, 2, 'And through', ['and', 'through']]

### **Mutability**

* Lists are **mutable**. This means that they can be modified after they are created.

1

2

3

num\_list = [1, 2, 3]

num\_list[0] = 5446

print(num\_list)

RunReset

### **Usage**

* Lists are very versatile and therefore are used in numerous cases. Some common ones are:
  + Storing collections of related items
  + Storing collections of items that you want to iterate over: Because lists are ordered, you can easily iterate over their elements using a for loop or list comprehension.
  + Sorting and searching: Lists can be sorted and searched, making them useful for tasks such as finding the minimum or maximum value in a list or sorting a list of items alphabetically.
  + Modifying existing data: Because lists are mutable, they are useful for situations in which you know you’ll need to modify your data.
  + Storing results: Lists can be used to store the results of a computation or a series of operations, making them useful in many different programming tasks.

### **Methods**

* You can find methods for the Python list class in [More on Lists](https://docs.python.org/3/tutorial/datastructures.html#more-on-lists) in the Python documentation.

## **Tuples**

### **Syntax/instantiation**

* Parentheses, with each element separated by a comma:

**Note**: The following code block is not interactive.

1

2

empty\_tuple = ()

my\_tuple = (1, 'z')

**Note:** When using parentheses to declare a tuple with just a single element, you must use a trailing comma.

1

2

3

4

5

test1 = (1)

test2 = (2,)

print(type(test1))

print(type(test2))

RunReset

* No parentheses, but each element followed by a comma (even if there’s only one element):

1

2

3

4

5

tuple1 = 1,

tuple2 = 2, 3

print(type(tuple1))

print(type(tuple2))

RunReset

* The **tuple()** function can be used for instantiation, and for conversion of iterable data types.

**Note**: The following code block is not interactive.

1

2

empty\_tuple = tuple()

my\_tuple = tuple([1, 'z'])

### **Content**

* Tuples can contain any data type, and in any combination. So, a single tuple can contain strings, integers, floats, lists, dictionaries, and other tuples.

**Note**: The following code block is not interactive.

1

my\_tuple = (1871, 'all', 'mimsy', ('were', 'the'), ['borogroves'])

### **Mutability**

* Tuples are **immutable**. This means that once a tuple is created, it cannot be modified.

### **Usage**

* Common uses of tuples include:
  + Returning multiple values from a function
  + Packing and unpacking sequences: You can use tuples to assign multiple values in a single line of code.
  + Dictionary keys: Because tuples are immutable, they can be used as dictionary keys, whereas lists cannot. (You’ll learn more about dictionaries later.)
  + Data integrity: Due to their immutable nature, tuples are a more secure way of storing data because they safeguard against accidental changes.

### **Methods**

* Because tuples are built for data security, Python has only two methods that can be used on them:
  + **count()** returns the number of times a specified value occurs in the tuple.
  + **index()** searches the tuple for a specified value and returns the index of the first occurrence of the value.

## Key takeaways

Strings, lists, and tuples are all iterable sequential data structures that share many similarities. They also have fundamental differences that you should be aware of so you can make effective choices in your work as a data professional. When selecting a data structure, consider its manner of instantiation, content, mutability, and the use case.

## Resources:

* For more information about strings, refer to the [Introduction to Python strings documentation](https://docs.python.org/3/tutorial/introduction.html#strings).
* For more information about lists, refer to the [Introduction to Python lists documentation](https://docs.python.org/3/tutorial/introduction.html#lists).
* For more information about tuples, refer to the [Python Standard Data Types tuples documentation](https://docs.python.org/3/library/stdtypes.html#tuples).

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 4. More with loops, lists, and tuples.

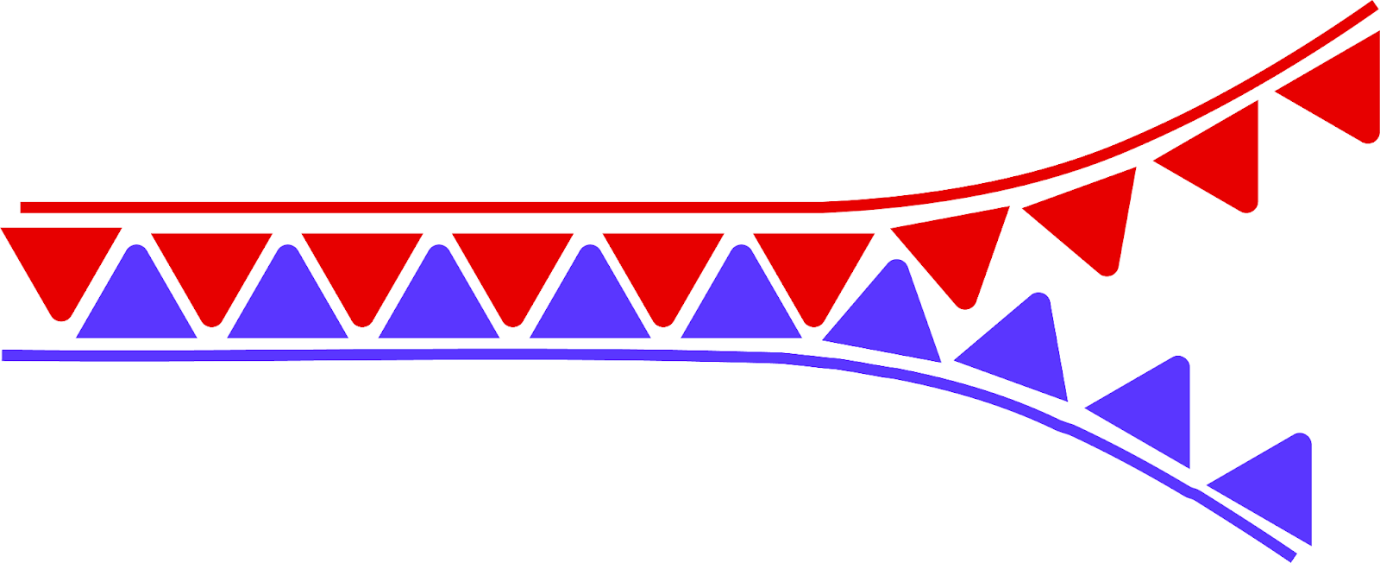
I'm ready to use the follow-along guide while watching this video.

# zip(), enumerate(), and list comprehension

You’ve learned much about iterable objects such as strings, lists, and tuples, and soon you’ll learn more. These objects comprise many of Python’s core data structures and, as a data professional, you’ll work with them constantly. While working in Python, you’ll often need to perform the same tasks and operations many times. This reading will introduce you to three time-saving tools: **zip()**, **enumerate()**, and list comprehension.

## **zip()**

The [zip() function](https://docs.python.org/3/library/functions.html#zip) is a built-in Python function that does what the name implies: It performs an element-wise combination of sequences.



The function returns an **iterator** that produces tuples containing elements from each of the input sequences. An iterator is an object that enables processing of a collection of items one at a time without needing to assemble the entire collection at once. Use an iterator with loops or other iterable functions such as **list()** or **tuple()**. Here’s an example:

1

2

3

4

5

6

7

cities = ['Paris', 'Lagos', 'Mumbai']

countries = ['France', 'Nigeria', 'India']

places = zip(cities, countries)

print(places)

print(list(places))

RunReset

Notice that, in this case, the **list()** function is used to generate a list of tuples from the iterator object. Here are a few things to keep in mind when using the **zip()** function.

* It works with two or more iterable objects. The given example zips two sequences, but the **zip()** function will accept more sequences and apply the same logic.
* If the input objects are of unequal length, the resulting iterator will be the same length as the shortest input.
* If you give it only one iterable object as an argument, the function will return an iterator that produces tuples containing only one element from that iterable at a time.

### Unzipping

You can also unzip an object with the **\*** operator. Here’s the syntax:

1

2

3

4

scientists = [('Nikola', 'Tesla'), ('Charles', 'Darwin'), ('Marie', 'Curie')]

given\_names, surnames = zip(\*scientists)

print(given\_names)

print(surnames)

RunReset

Note that this operation unpacks the tuples in the original list element-wise into two tuples, thus separating the data into different variables that can be manipulated further.

**enumerate()**

The [enumerate() function](https://docs.python.org/3/library/functions.html#enumerate) is another built-in Python function that allows you to iterate over a sequence while keeping track of each element’s index. Similar to **zip()**, it returns an iterator that produces pairs of indices and elements. Here’s an example:

1

2

3

letters = ['a', 'b', 'c']

for index, letter in enumerate(letters):

   print(index, letter)

RunReset

Note that the default starting index is zero, but you can assign it to whatever you want when you call the **enumerate()** function. For example:

1

2

3

letters = ['a', 'b', 'c']

for index, letter in enumerate(letters, 2):

   print(index, letter)

RunReset

In this case, the number two was passed as an argument to the function, and the first element of the resulting iterator had an index of two. The **enumerate()** function is useful when an element’s place in a sequence must be used to determine how the element should be handled in an operation.

## **List comprehension**

One of the most useful tools in Python is [list comprehension](https://docs.python.org/3/tutorial/datastructures.html?highlight=list%20comprehension#list-comprehensions). List comprehension is a concise and efficient way to create a new list based on the values in an existing iterable object. List comprehensions take the following form:

**my\_list = [expression for element in iterable if condition]**

In this syntax:

* **expression** refers to an operation or what you want to do with each element in the iterable sequence.
* **element** is the variable name that you assign to represent each item in the iterable sequence.
* **iterable** is the iterable sequence.
* **condition** is any expression that evaluates to **True** or **False**. This element is optional and is used to filter elements of the iterable sequence.

Here are some examples of list comprehensions:

This list comprehension adds 10 to each number in the list:

1

2

3

numbers = [1, 2, 3, 4, 5]

new\_list = [x + 10 for x in numbers]

print(new\_list)

RunReset

In the preceding example, **x + 10** is the expression, **x** is the element, and **numbers** is the iterable sequence. There is no condition.

This next list comprehension extracts the first and last letter of each word as a tuple, but only if the word is more than five letters long.

1

2

3

words = ['Emotan', 'Amina', 'Ibeno', 'Sankwala']

new\_list = [(word[0], word[-1]) for word in words if len(word) > 5]

print(new\_list)

RunReset

Note that multiple operations can be performed in the expression component of the list comprehension to result in a list of tuples. This example also makes use of a condition to filter out words that are not more than five letters long.

## **Key takeaways**

**zip()**, **enumerate()**, and list comprehension make code more efficient by reducing the need to rely on loops to process data and simplifying working with iterables. Understanding these common tools will save you time and make your process much more dynamic when manipulating data.

# **Instructions**

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity does not use any imported data, but when an activity does work with imported data this section will contain information about its source, what the different variables mean, and other useful information, as applicable.

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



# Test your knowledge: Lists and tuples

### 1.

Question 1

Lists and their contents are immutable, so their elements cannot be modified, added, or removed.

1 / 1 point

True

False

Correct

Lists and their contents are mutable, so their elements can be modified, added, or removed. A listis a data structure that helps store and manipulate an ordered collection of items.

### 2.

Question 2

What Python method adds an element to the end of a list?

1 / 1 point

**append()**

**pop()**

**remove()**

**type()**

Correct

Python’s **append()** method adds an element to the end of a list.

### 3.

Question 3

A data professional wants to instantiate a tuple. What Python elements can they use to do so? Select all that apply.

1 / 1 point

Parentheses

Correct

A data professional can use parentheses or the **tuple()** function to instantiate a tuple. Atuple is an immutable sequence that can contain elements of any data type.

The **insert()** function

The **tuple()** function

Correct

A data professional can use parentheses or the **tuple()** function to instantiate a tuple. Atuple is an immutable sequence that can contain elements of any data type.

Square brackets

### 4.

Question 4

What Python technique formulaically creates a new list based on the values in an existing list?

1 / 1 point

List sequencing

List nesting

List comprehension

List conversion

Correct

A list comprehension formulaically creates a new list based on the values in an existing list. A list comprehension functions like a for loop, but is a more efficient and elegant way to create a new list from an existing list.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 5. Introduction to dictionaries.

I'm ready to use the follow-along guide while watching this video.

**Dictionary**

A data structure that consists of a collection of key-value pairs

**Immutable keys**

-Integers

-Floats

-Tuples

**Mutable data types cannot be used as keys**

-Lists

-Sets

-Other dictionaries

\*In Python versions 3.7 and up, dictionaries retain their order.

## Question

Fill in the blank: Adictionary is a data structure that consists of a collection of \_\_\_\_\_ pairs.

key-value

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 6. Dictionary methods.

I'm ready to use the follow-along guide while watching this video.

**keys()**

A dictionary method to retrieve only the dictionary’s keys

**values()**

A dictionary method to retrieve only the dictionary’s values

**items()**

A dictionary method to retrieve both the dictionary’s keys and values.

# Reference guide: Dictionaries

By now you’ve encountered dictionaries and are discovering their power and utility as a data structure in Python. You’ve also learned that dictionaries provide a way to store and retrieve data using key-value pairs. Data professionals use dictionaries for many tasks, so it’s important to be familiar with how they work. This reading is a reference guide about dictionaries. It’s designed to help you in your Python learning journey.

## **Save this course item**

You may want to save a copy of this guide for future reference. Use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the link below and select “Use Template.”

[Reference guide: Dictionaries](https://docs.google.com/document/d/1H1MZbLW6wA_my7dxF9WhmScSgxHSBbj1GknNLiLT0Ic/template/preview?pli=1)

OR

If you don’t have a Google account, download the item directly from the attachment below.

## **Create a dictionary**

There are two main ways to create dictionaries in Python:

* Braces: **{}**
* The dict function: **dict()**

When instantiating a dictionary using braces, separate each element with a colon. For example, the following code creates a dictionary containing continents as keys and their smallest countries as values:

**Note:** The following code block is not interactive.

1

2

3

4

5

6

7

smallest\_countries = {'Africa': 'Seychelles',

                     'Asia': 'Maldives',

                     'Europe': 'Vatican City',

                     'Oceania': 'Nauru',

                     'North America': 'St. Kitts and Nevis',

                     'South America': 'Suriname'

                     }

To create an empty dictionary, use empty braces or the **dict()** function:

**Note:** The following code block is not interactive.

1

2

empty\_dict\_1 = {}

empty\_dict\_2 = dict()

The **dict()** function uses a different syntax, where keys are entered as the function’s keyword arguments and values are assigned with an equals operator:

**Note:** The following code block is not interactive.

1

2

3

4

5

6

7

smallest\_countries = dict(africa='Seychelles',

                         asia='Maldives',

                         europe='Vatican City',

                         oceania='Nauru',

                         north\_america='St. Kitts and Nevis',

                         south\_america ='Suriname'

)

Notice that, because the keywords cannot be entered as strings, they cannot contain whitespaces.

Some important notes about keys and values:

* **Dictionary keys:** Can be of any immutable data type, such as strings, numbers, or tuples
* **Dictionary values:** Can be of any data type—mutable or immutable—including other dictionaries or objects
* Each key can only correspond to a single value; so, for example, this will throw an error:

1

invalid\_dict = {'numbers': 1, 2, 3}

RunReset

But if you enclose multiple values within another single data structure, you can create a valid dictionary. For example:

1

2

valid\_dict = {'numbers': [1, 2, 3]}

print(valid\_dict)

RunReset

## **Work with dictionaries**

### Access values

To access a specific value in a dictionary, you must refer to its key using brackets:

1

2

3

4

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

print(my\_dict['nums'])

RunReset

To access all values in a dictionary, use the **values()** method:

1

2

3

4

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

print(my\_dict.values())

RunReset

### Assign new keys

Dictionaries are mutable data structures in Python. You can add to and modify existing dictionaries. To add a new key to a dictionary, use brackets:

1

2

3

4

5

6

7

8

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

# Add a new 'floats' key

my\_dict['floats'] = [1.0, 2.0, 3.0]

print(my\_dict)

RunReset

### Check if a key exists in a dictionary

To check if a key exists in a dictionary, use the **in** keyword:

1

2

3

4

5

6

7

8

9

10

11

smallest\_countries = {'Africa': 'Seychelles',

                     'Asia': 'Maldives',

                     'Europe': 'Vatican City',

                     'Oceania': 'Nauru',

                     'North America': 'St. Kitts and Nevis',

                     'South America': 'Suriname'

                     }

print('Africa' in smallest\_countries)

print('Asia' not in smallest\_countries)

RunReset

### Delete a key-value pair

To delete a key-value pair from a dictionary, use the **del** keyword:

1

2

3

4

5

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

del my\_dict['abc']

print(my\_dict)

RunReset

## **Dictionary methods**

Dictionaries are a core Python class. As you’ve learned, classes package data with tools to work with it. Methods are functions that belong to a class. Dictionaries have a number of built-in methods that are very useful. Some of the most commonly used methods include:

### items()

Return a view of the (key, value) pairs of the dictionary:

1

2

3

4

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

print(my\_dict.items())

RunReset

### keys()

Return a view of the dictionary’s keys:

1

2

3

4

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

print(my\_dict.keys())

RunReset

### values()

Return a view of the dictionary’s values:

1

2

3

4

my\_dict = {'nums': [1, 2, 3],

          'abc': ['a', 'b', 'c']

          }

print(my\_dict.values())

RunReset

Note that the objects returned by these methods are view objects. They provide a dynamic view of the dictionary’s entries, which means that, when the dictionary changes, the view reflects these changes. Dictionary views can be iterated over to yield their respective data. They also support membership tests.

## **Additional resources**

* For more information about dictionaries, refer to the [Python dictionary documentation](https://docs.python.org/3/tutorial/datastructures.html#dictionaries).
* For more dictionary methods, refer to the [Python mapping types documentation](https://docs.python.org/3/library/stdtypes.html#mapping-types-dict).
* For more information about view objects, refer to the [Python dictionary view objects documentation](https://docs.python.org/3/library/stdtypes.html#dict-views).

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 7. Introduction to sets.

I'm ready to use the follow-along guide while watching this video.

**Set**

A data structure in Python that contains only ordered, non-interchangeable elements.

Instantiated with set() function or non-empty braces.

To define an empty set, you have to use set().

## Question

In Python, what type of elements does a set contain?Select all that apply.

Non-interchangeable

Unordered

**intersection()**

A function that finds the elements that two sets have in common.

**union()**

A function that finds the elements from both sets.

**difference()**

A function that finds the elements from present in one set, but not the other.

**symmetric\_difference()**

A function that finds elements from both sets that are mutually not present in the other

# Reference guide: Sets

Data professionals depend on sets for separating data and identifying its unique elements. As you have been discovering, set objects are similar to lists and dictionaries, yet they do not have key-value pairs or positional index[i] capability. Additionally, sets contain unique values but have no item order or index behavior. Data professionals compare sets to understand the range of data they contain, where they intersect, and what items are present in either set but not both. Sets are also helpful when cleaning data for analysis. This reading is a reference guide for sets to help you as you continue learning Python.

## **Save this course item**

You may want to save a copy of this guide for future reference. Use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the link below and select “Use Template.”

[Reference guide: Sets](https://docs.google.com/document/d/13ObxgpD7_EMt2rU09dmatLe8_0VVdGf_NZ64qfKFBoI/template/preview)

OR

If you don’t have a Google account, download the item directly from the attachment below.

## **Sets review**

A set is a collection of unique data elements, without duplicates. In Python, it is an object class—in fact, two different classes—which you’ll learn about in this reading. However, sets are not unique to Python or even to computer programming; they are an important concept in general mathematics. Sets provide a simple means to identify unique data elements.

## **Create a set**

Create a set using braces:

**my\_set = {5, 10, 10, 20}**

Note that an empty set cannot be created with braces, as this will be interpreted as an empty dictionary.

There are two functions for creating sets in Python: **set()** and **frozenset()**. Use these on any iterable object. Or use these functions to create empty sets.

### [**set()**](https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset)

* This is a mutable data type.
* Because it’s mutable, this class comes with [additional methods to add and remove data from the set](https://docs.python.org/3/library/stdtypes.html#frozenset.update).
* It can be applied to any iterable object and will remove duplicate elements from it.
* It is unordered and non-indexable.
* Elements in a set must be hashable; generally, this means they must be immutable. (Refer to the additional resources for more information on hashing.)

In the examples that follow, four sets are instantiated using a variety of data types:

1

2

example\_a = [1, 2, 2.0, '2']

set(example\_a)

RunReset

Notice that, in the preceding example, **2** and **2.0** are evaluated as equivalent, even though one is an integer and the other is a float.

1

2

example\_b = ('apple', (1, 2, 2, 2, 3), 2)

set(example\_b)

RunReset

In the preceding example, **(1, 2, 2, 2, 3)** is a tuple, which is hashable (≈ immutable) and thus treated as a distinct single element in the resulting set.

1

2

example\_c = [1.5, {'a', 'b', 'c'}, 1.5]

set(example\_c)

RunReset

The preceding example throws an error because each element of a set must be hashable (≈ immutable), but **{‘a’, ‘b’, ‘c’}** is a set, which is a mutable (unhashable) object.

The following example demonstrates the **add()** method, which is one of the special methods available to sets but not to frozensets.

1

2

3

example\_d = {'mother', 'hamster', 'father'}

example\_d.add('elderberries')

example\_d

RunReset

An element was added to the **example\_d** set, thus modifying it. This is an example of the mutability of the **set** class.

### [frozenset()](https://docs.python.org/3/library/stdtypes.html#frozenset)

Frozensets are another type of set in Python. They are their own class, and they are very similar to sets, except they are immutable.

* This is an immutable data type.
* It can be applied to any iterable object and will remove duplicate elements from it.
* Because they’re immutable, frozensets can be used as dictionary keys and as elements in other sets.

In this example, a frozenset is used within a set.

1

2

example\_e = [1.5, frozenset(['a', 'b', 'c']), 1.5]

set(example\_e)

RunReset

Unlike **example\_c** previously, this set does not throw an error. This is because it contains a frozenset, which is an immutable type and can therefore be used in sets.

## Set methods

Sets are useful to determine which values are contained in a data structure and to eliminate duplicate values. There are numerous set methods—such as intersection, union, difference, and symmetric difference—that add functionality and power to working with sets.

### [union()](https://docs.python.org/3/library/stdtypes.html#frozenset.union)

* Return a new set with elements from the set and all others.
* The operator for this function is the pipe ( **|** ).

1

2

3

4

5

set\_1 = {'a', 'b', 'c'}

set\_2 = {'b', 'c', 'd'}

print(set\_1.union(set\_2))

print(set\_1 | set\_2)

RunReset

### [intersection()](https://docs.python.org/3/library/stdtypes.html#frozenset.intersection)

* Return a new set with elements common to the set and all others.
* The operator for this function is the ampersand (**&**).

1

2

3

4

5

set\_1 = {'a', 'b', 'c'}

set\_2 = {'b', 'c', 'd'}

print(set\_1.intersection(set\_2))

print(set\_1 & set\_2)

RunReset

### [difference()](https://docs.python.org/3/library/stdtypes.html#frozenset.difference)

* Return a new set with elements in the set that are not in the others.
* The operator for this function is the subtraction operator ( **-** ).

1

2

3

4

5

set\_1 = {'a', 'b', 'c'}

set\_2 = {'b', 'c', 'd'}

print(set\_1.difference(set\_2))

print(set\_1 - set\_2)

RunReset

### [symmetric\_difference()](https://docs.python.org/3/library/stdtypes.html#frozenset.symmetric_difference)

* Return a new set with elements in either the set or other, but not both.
* The operator for this function is the caret ( **^** ).

1

2

3

4

5

set\_1 = {'a', 'b', 'c'}

set\_2 = {'b', 'c', 'd'}

print(set\_1.symmetric\_difference(set\_2))

print(set\_1 ^ set\_2)

RunReset

## **Additional resources**

* Refer to the Python documentation for more information about [sets and frozensets](https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset), including a complete list of available class methods.
* For methods unique to sets (and unavailable to frozensets), refer to this [Python set methods documentation](https://docs.python.org/3/library/stdtypes.html#frozenset.update).
* For more examples of sets, refer to the [Python tutorial on sets](https://docs.python.org/3/tutorial/datastructures.html#sets).
* For more information on hash tables, what makes something hashable, and hashing as a concept, refer to this [resource from Runestone Academy](https://runestone.academy/ns/books/published/pythonds/SortSearch/Hashing.html). For an interesting story about the birth of the original hashing algorithms, check out this [IEEE Spectrum article](https://spectrum.ieee.org/hans-peter-luhn-and-the-birth-of-the-hashing-algorithm).

# **Instructions**

To complete this lab, you will open a Jupyter notebook and follow instructions to enter code and written responses where prompted. The Jupyter notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

# **Data Dictionary**

This activity uses air quality index (AQI) data from the United States Environmental Protection Agency (EPA). The data was taken from the EPA's own [repository of AQI data](https://aqs.epa.gov/aqsweb/airdata/download_files.html) and assembled for pedagogical purposes. The data is comprised of 1,725 observations that include state, county, and AQI values. For more information about AQI data, refer to [AirNow](https://www.airnow.gov/aqi/aqi-basics/). Here is a link to a downloadable csv file of the [AQI data](https://drive.google.com/file/d/1qBp4YHMMumE7ELxYpOxM0OpY7558BnBo/view?usp=sharing). You do not need to download this to continue working on the Coursera platform.

This activity also imports a file called **ada\_c2\_labs.py**, which is used to generate test cases so you can verify whether your code successfully completes the given tasks. Again, if you are working in a notebook on the Coursera platform, you do not have to do anything. If you are working in a notebook that is not on the Coursera platform, you will need to download this file to pull the data and run the test cases.

You can [download ada\_c2\_labs.py](https://drive.google.com/file/d/1PEeoEYVzu0M9snRWKeXPWyQx1mT3PQ5Y/view?usp=sharing) here or from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the file, and clicking **Download**.

# Tips

 As you complete the lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.
* **Hints:** Hidden hints provide suggestions you can use to complete your work.

To review how to work in Jupyter notebooks, refer to the reading [Create, upload, and edit Jupyter notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/create-upload-and-edit-jupyter-notebooks).

Be sure to complete this lab before moving on. To get started, click **Open Lab**.

The next course item will provide an exemplar of a completed lab for you to compare to your own work. To access the exemplar, return to the main course menu and click **Next**.



### 1.

Question 1

Fill in the blank: In Python, a dictionary’s \_\_\_\_\_ must be immutable.

1 / 1 point

keys

order

sets

lists

Correct

In Python, a dictionary’s keys must be immutable. Immutable keys include, but are not limited to, integers, floats, tuples, and strings. Lists, sets, and other dictionaries are not included in this category since they are mutable.

### 2.

Question 2

In Python, what does the **items()** method retrieve?

1 / 1 point

Both a dictionary’s keys and values

Only a dictionary’s keys

Only a dictionary’s values

A dictionary’s sets

Correct

In Python, the **items()** method is used to retrieve both a dictionary’s keys and values.

### 3.

Question 3

A data professional is working with two Python sets. What function can they use to find all the elements from both sets?

1 / 1 point

**union()**

**difference()**

**intersection()**

**symmetric\_difference()**

Correct

When working with two Python sets, a data professional can use the **union()** function to find all the elements from both sets.

**Library (or package)**

Broadly refers to a reusable collection of code

**matplotlib**

A library for creating static, animated, and interactive visualizations in Python

**Seaborn**

A visualization library based on matplotlib that provides a simpler interface for working with common plots and graphs

**Python libraries:**

-NumPy

-pandas

**Numpy**

An essential library that contains multidimensional array and matrix data structures and functions to manipulate them

**Pandas**

A powerful library built on top of NumPy that’s used to manipulate and analyze tabular data.

**Module**

A simple Python file containing a collection of functions and global variables.

**Global variables**

Variables that can be accessed from anywhere in a program or script.

**Commonly used Python modules:**

-Math

-Random

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 8. Introduction to NumPy.

I'm ready to use the follow-along guide while watching this video.

**Vectorization**

Enables operations to be performed on multiple components of a data object at the same time

**Import statement**

Uses the import keyword to load an external library, package, module, or function into your computing environment

**Aliasing**

Lets you assign an alternate name-or alias-by which you can refer to something.

## Question

Fill in the blank: In NumPy, \_\_\_\_\_ enables operations to be performed on multiple components of a data object at the same time.

vectorization

# Understand Python libraries, packages, and modules

Recently, you learned about Python libraries, packages, and modules. As you’ve discovered, importing these tools saves data professionals time and enhances their programming. Another benefit of commonly used  libraries is that they are constantly scrutinized and updated by talented and knowledgeable programmers. Thus, you can be confident that the underlying code is high quality.

In this reading, you’ll learn more about the basic features of libraries, packages, and modules; how they are related; and a selection of basic modules you might use as a data professional.

## **Libraries, packages, and modules**

A **library** is a corpus of reusable code modules and their accompanying documentation. Libraries are bundled into **packages** that you install, which can then be imported into your coding environment as needed. You’ll typically encounter the terms “library” and “package” used interchangeably. Generally, this certificate program will refer to both as libraries, but it’s important to be acquainted with both terms.

**Modules** are similar to libraries, in that they are groups of related classes and functions, but they are generally subcomponents of libraries. In other words, a library can have many different modules, and you can choose to import the entire library or just the module you need.

## **Import statements**

Libraries and modules beyond the Python standard library typically must be imported into your working environment on an as-needed basis. Additional libraries are installed first and then imported into your working environment as needed.

To import a library or module, use an import statement. Import statements require particular syntax using the **import** keyword. Here are some examples:

**Note:** The following code block is not interactive.

1

import numpy

This import statement imports the NumPy library into your working environment. After running this command, you’ll have access to all NumPy classes and functions. For instance, to use the **array()** function on **[2, 4, 6]**, you’d write:

**Note:** The following code block is not interactive.

1

numpy.array([2, 4, 6])

Notice that to access the **array()** function, you must precede it with **numpy**, because this indicates that the function is coming from the NumPy library.

### Aliasing

Another time-saver with Python libraries is aliasing. Aliasing helps you avoid typing a library's full name every time you want to access one of its functions. Instead, you’ll assign the library an **alias**. An alias is an abbreviated name, which is designated using the **as** keyword:

**Note:** The following code block is not interactive.

1

import numpy as np

In this case, the NumPy library is imported with the **np** alias. You can assign any abbreviation you like as an alias, but commonly used libraries have common aliases. Therefore, straying from those could cause confusion when sharing code with others. Here are some common libraries and their conventional aliases used by data professionals:

**Note:** The following code block is not interactive.

1

2

3

4

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

NumPy is used for high-performance vector and matrix computations. Pandas is a library for manipulating and analyzing tabular data. Seaborn and matplotlib are both libraries used to create graphs, charts, and other data visualizations.

After running these imports, whenever you want to use a function from one of these libraries, precede the function with the alias. Returning to the example with NumPy’s **array()** function, after aliasing, you’d write:

**Note:** The following code block is not interactive.

1

np.array([2, 4, 6])

### Additional import syntax

#### **Importing modules**

Recall from the previous example:

**Note:** The following code block is not interactive.

1

import matplotlib.pyplot as plt

You may have noticed that this syntax differs slightly from the other examples. In this case, matplotlib is the library and pyplot is a module inside. The pyplot module is aliased as **plt**, and it’s accessed from the matplotlib library using the dot.

#### **Importing functions**

Just as you can import libraries and modules, you can also import individual functions from libraries or from modules within libraries using a specific syntax. Here’s an example depicting a common import when using the scikit-learn library to build machine learning models:

**Note:** The following code block is not interactive.

1

from sklearn.metrics import precision\_score, recall\_score

Again, notice the different syntax. The import statement begins with the **from** keyword, followed by **sklearn.metrics**—the scikit-learn library + the **metrics** module. Next is the **import** keyword followed by the desired functions. In this case, there are two: **precision\_score** and **recall\_score**.

The same syntax can be applied to the example using NumPy’s **array()** function. However, note that you typically would not encounter individual functions being imported from NumPy. It’s much easier and more common to just import the whole library.

**Note:** The following code block is not interactive.

1

from numpy import array

When a function is imported by name, like in this example, you can use it without any preceding syntax to indicate the library or module that it comes from:

**Note:** The following code block is not interactive.

1

array([2,4,6])

### Discouraged syntax

One last syntactical variation that you might encounter is:

**Note:** The following code block is not interactive.

1

from library.module import \*

This imports everything from a particular library or module and allows you to use its functions without any preceding syntax. So, for instance, if you wrote **from numpy import \***,  you’d be able to use all of NumPy’s functions without preceding them with **numpy** or **np**. **This approach is not recommended** because it makes it difficult to track where functions come from. However, it’s helpful to be aware of this because you will likely encounter it in your work as a data professional. And, in specific instances, it might be useful.

## **Commonly used built-in modules**

The Python standard library comes with a number of built-in modules relevant to data professional work such as **math**, **datetime**, and **random**. These can be imported without additional installation. In other words, you can import them directly, as long as you have Python installed. For example:

### [datetime](https://docs.python.org/3/library/datetime.html#module-datetime)

* Provides many helpful date and time conversions and calculations

Example:

1

2

3

4

5

6

7

8

import datetime

date = datetime.date(1977, 5, 8)       # assign a date to a variable

print(date)                            # print date

print(date.year)                       # print the year that the date is in

delta = datetime.timedelta(days=30)    # assign a timedelta of 30 days to a

                                       # variable

print(date - delta)                    # print date of 30 days prior

RunReset

### [math](https://docs.python.org/3/library/math.html#module-math)

* Provides access to mathematical functions

Example:

1

2

3

4

5

import math

print(math.exp(0))          # e\*\*0

print(math.log(1))          # ln(1)

print(math.factorial(4))    # 4!

print(math.sqrt(100))       # square root of 100

RunReset

### [random](https://docs.python.org/3/library/random.html#module-random)

* Useful for generating pseudo-random numbers (refer to the documentation for explanation of pseudo-random number generation)

Example:

1

2

3

4

import random

print(random.random())          # 0.0 <= X < 1.0

print(random.choice([1, 2, 3])) # choose a random element from a sequence

print(random.randint(1, 10))    # a <= X <= b

RunReset

## **Key takeaways**

Libraries, packages, and modules are gateways to Python's countless capabilities. Understanding how to leverage them for your own coding needs will unlock new tools and functions that make your work much more efficient. Check out the Python Package Index at the [PyPI](https://pypi.org/) repository to search for useful libraries. There are packages designed for applications as diverse as chemistry, audio editing, natural language processing, and video games. Whatever it is you’re trying to do, chances are someone has developed a suite of tools to help you!

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 9. Basic array operations.

I'm ready to use the follow-along guide while watching this video.

**N-dimensional array (ndarray)**

The core data object of Numpy.

**ndim**

A NumPy attribute used to check the number of dimensions of an array

**reshape()**

NumPy method used to change the shape of an array.

# Reference guide: Arrays

As you’ve learned, NumPy is a powerful library capable of performing advanced numerical computing. One of its main benefits is the ability to work with arrays, as an operation applied to a vector executes much faster than the same operation applied to a list. Performance increases become further apparent when working with large volumes of data. This reading is a reference guide for working with NumPy arrays.

## **Save this course item**

You may want to save a copy of this guide for future reference. Use it as a resource for additional practice or in your future professional projects. To access a downloadable version of this course item, click the link below and select “Use Template.”

[Reference guide: Arrays](https://docs.google.com/document/d/1NopsXs4caG8W2oDFt0JIsZvVa2MckyaifO0pCqPVzpA/template/preview?resourcekey=0-Q6qzwooboi4PO_vB2hnyhw)

OR

If you don’t have a Google account, download the item directly from the attachment below.

## **Create an array**

As you’ve discovered, to use NumPy, you must first import it. Standard practice is to alias it as **np**.

### [**np.array()**](https://numpy.org/doc/stable/reference/generated/numpy.array.html)

This creates an **ndarray** (n-dimensional array). There is no limit to how many dimensions a NumPy array can have, but arrays with many dimensions can be more difficult to work with.

#### 1-D array:

1

2

3

import numpy as np

array\_1d = np.array([1, 2, 3])

array\_1d

RunReset

Notice that a one-dimensional array is similar to a list.

#### 2-D array:

1

2

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

array\_2d

RunReset

Notice that a two-dimensional array is similar to a table.

#### 3-D array:

1

2

array\_3d = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

array\_3d

RunReset

Notice that a three-dimensional array is similar to two tables.

### [np.zeros()](https://numpy.org/doc/stable/reference/generated/numpy.zeros.html)

* This creates an array of a designated shape that is pre-filled with zeros:

1

2

np.zeros((3, 2))

RunReset

### [np.ones()](https://numpy.org/doc/stable/reference/generated/numpy.ones.html)

* This creates an array of a designated shape that is pre-filled with ones:

1

2

np.ones((2, 2))

RunReset

### [np.full()](https://numpy.org/doc/stable/reference/generated/numpy.full.html)

* And this creates an array of a designated shape that is pre-filled with a specified value:

1

2

np.full((5, 3), 8)

RunReset

These functions are useful for various situations:

* To initialize an array of a specific size and shape, then fill it with values derived from a calculation
* To allocate memory for later use
* To perform matrix operations

## **Array methods**

NumPy arrays have many methods that allow you to manipulate and operate on them. For a full list, refer to the [NumPy array documentation](https://numpy.org/doc/stable/reference/arrays.ndarray.html). Some of the most commonly used methods follow:

### [ndarray.flatten()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.flatten.html)

* This returns a copy of the array collapsed into one dimension.

1

2

3

4

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

print(array\_2d)

print()

array\_2d.flatten()

RunReset

### [ndarray.reshape()](https://numpy.org/doc/stable/reference/generated/numpy.reshape.html#numpy.reshape)

* This gives a new shape to an array without changing its data.

1

2

3

4

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

print(array\_2d)

print()

array\_2d.reshape(3, 2)

RunReset

Adding a value of -1 in the designated new shape makes the process more efficient, as it indicates for NumPy to automatically infer the value based on other given values.

1

2

3

4

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

print(array\_2d)

print()

array\_2d.reshape(3, -1)

RunReset

### [ndarray.tolist()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.tolist.html)

* This converts an array to a list object. Multidimensional arrays are converted to nested lists.

1

2

3

4

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

print(array\_2d)

print()

array\_2d.tolist()

RunReset

### Mathematical functions

NumPy arrays also have many methods that are mathematical functions:

* [ndarray.max()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.max.html): returns the maximum value in the array or along a specified axis.
* [ndarray.mean()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.mean.html): returns the mean of all the values in the array or along a specified axis.
* [ndarray.min()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.min.html): returns the minimum value in the array or along a specified axis.
* [ndarray.std()](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.std.html): returns the standard deviation of all the values in the array or along a specified axis.

1

2

3

4

5

6

7

8

a = np.array([(1, 2, 3), (4, 5, 6)])

print(a)

print()

print(a.max())

print(a.mean())

print(a.min())

print(a.std())

RunReset

## **Array attributes**

NumPy arrays have several attributes that enable you to access information about the array. Some of the most commonly used attributes include the following:

* [ndarray.shape](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.shape.html): returns a tuple of the array’s dimensions.
* [ndarray.dtype](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.dtype.html): returns the data type of the array’s contents.
* [ndarray.size](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.size.html): returns the total number of elements in the array.
* [ndarray.T](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.T.html): returns the array transposed (rows become columns, columns become rows).

1

2

3

4

5

6

7

8

array\_2d = np.array([(1, 2, 3), (4, 5, 6)])

print(array\_2d)

print()

print(array\_2d.shape)

print(array\_2d.dtype)

print(array\_2d.size)

print(array\_2d.T)

RunReset

## **Indexing and slicing**

Access individual elements of a NumPy array using indexing and slicing. Indexing in NumPy is similar to indexing in Python lists, except multiple indices can be used to access elements in multidimensional arrays.

1

2

3

4

5

6

7

a = np.array([(1, 2, 3), (4, 5, 6)])

print(a)

print()

print(a[1])

print(a[0, 1])

print(a[1, 2])

RunReset

Slicing may also be used to access subarrays of a NumPy array:

1

2

3

4

5

a = np.array([(1, 2, 3), (4, 5, 6)])

print(a)

print()

a[:, 1:]

RunReset

## **Array operations**

NumPy arrays support many operations, including mathematical functions and arithmetic. These include array addition and multiplication, which performs element-wise arithmetic on arrays:

1

2

3

4

5

6

7

8

9

10

11

12

13

a = np.array([(1, 2, 3), (4, 5, 6)])

b = np.array([[1, 2, 3], [1, 2, 3]])

print('a:')

print(a)

print()

print('b:')

print(b)

print()

print('a + b:')

print(a + b)

print()

print('a \* b:')

print(a \* b)

RunReset

In addition, there are nearly 100 other useful [mathematical functions](https://numpy.org/doc/stable/reference/routines.math.html#mathematical-functions) that can be applied to individual or multiple arrays.

## **Mutability**

NumPy arrays are mutable, but with certain limitations. For instance, an existing element of an array can be changed:

1

2

3

4

5

6

a = np.array([(1, 2), (3, 4)])

print(a)

print()

a[1][1] = 100

a

RunReset

However, the array cannot be lengthened or shortened:

1

2

3

4

5

6

a = np.array([1, 2, 3])

print(a)

print()

a[3] = 100

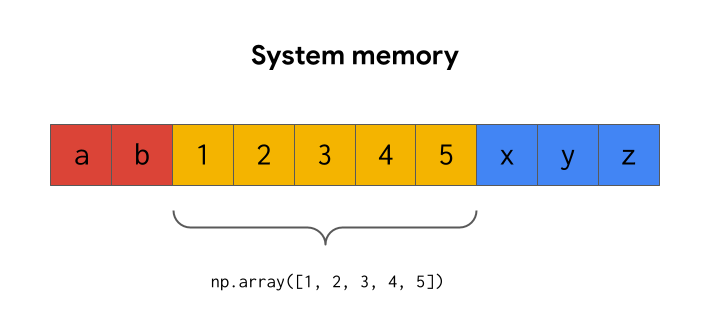
a

RunReset

### How NumPy arrays store data in memory

NumPy arrays work by allocating a contiguous block of memory at the time of instantiation. Most other structures in Python don’t do this; their data is scattered across the system’s memory. This is what makes NumPy arrays so fast; all the data is stored together at a particular address in the system’s memory.

Interestingly, this is also what prevents an array from being lengthened or shortened: The abutting memory is occupied by other information. There’s no room for more data at that memory address. However, existing elements of the array can be replaced with new elements.



The only way to lengthen an array is to copy the existing array to a new memory address along with the new data.

# Test your knowledge: Arrays and vectors with NumPy

### 1.

Question 1

Python libraries and packages include which of the following features? Select all that apply.

1 / 1 point

Modules

Correct

A Python library, or package, broadly refers to a reusable collection of code. Libraries and packages also contain related modules and documentation. You’ll often encounter the terms library and package used interchangeably.

Documentation

Correct

A Python library, or package, broadly refers to a reusable collection of code. Libraries and packages also contain related modules and documentation. You’ll often encounter the terms library and package used interchangeably.

Cells

Reusable collections of code

Correct

A Python library, or package, broadly refers to a reusable collection of code. Libraries and packages also contain related modules and documentation. You’ll often encounter the terms library and package used interchangeably.

### 2.

Question 2

What is the core data structure of NumPy?

1 / 1 point

Global variable

Array

List

Dictionary

Correct

The array is the core data structure of NumPy. The data object itself is known as an n-dimensional array, or **ndarray** for short. An array can be multidimensional, and all its elements must be of the same data type.

### 3.

Question 3

A data professional wants to confirm the datatype of the contents of array **x**. How would they do this?

1 / 1 point

**x.ndim**

**datatype(x)**

**type(x)**

**x.dtype**

Correct

**dtype** is a NumPy array attribute used to check the data type of the contents of an array.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 10. Introduction to pandas.

I'm ready to use the follow-along guide while watching this video.

**Tabular data**

Data that is in the form of a table, with rows and columns

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 11. pandas basics.

If you want to learn more about pandas, you can also explore the [pandas documentation](https://pandas.pydata.org/docs/).

I'm ready to use the follow-along guide while watching this video.

**DataFrame**

A two-dimensional, labeled data structure with rows and columns

**CSV file**

Stands for “comma-separated values.” A plaintext file that uses commas to separate distinct values from one another

## Question

Fill in the blank: In pandas a dataframe is a \_\_\_\_\_-dimensional, labeled data structure.

two

Correct

In pandas a dataframe is a two-dimensional, labeled data structure. A dataframe is organized into rows and columns.

**NaN**

How null values are represented in pandas, which stands for “not a number”

**loc[ ]**

Used to select pandas rows and columns by name

# The fundamentals of pandas

You’ve learned that Python has many open-source libraries and packages—including NumPy and pandas—that make it one of the most useful coding languages. In this reading, you will review the basics of pandas dataframes and learn more about how to work with them. Understanding the fundamentals of pandas is essential to becoming a capable and competent data professional.

## **Primary data structures**

Pandas has two primary data structures: **Series** and **DataFrame**.

* **Series:** A Series is a one-dimensional labeled array that can hold any data type. It’s similar to a column in a spreadsheet or a one-dimensional NumPy array. Each element in a series has an associated label called an index. The index allows for more efficient and intuitive data manipulation by making it easier to reference specific elements of your data.
* **DataFrame:** A dataframe is a two-dimensional labeled data structure—essentially a table or spreadsheet—where each column and row is represented by a Series.

## **Create a DataFrame**

To use pandas in your notebook, first import it. Similar to NumPy, pandas has its own standard alias, **pd**, that’s used by data professionals around the world:

1

import pandas as pd

Once you’ve imported pandas into your working environment, create a dataframe. Here are some of the ways to create a **DataFrame** object in a Jupyter Notebook.

**From a dictionary:**

1

2

3

d = {'col1': [1, 2], 'col2': [3, 4]}

df = pd.DataFrame(data=d)

df

RunReset

**From a numpy array:**

1

2

3

df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),

                  columns=['a', 'b', 'c'])

df2

RunReset

**From a comma-separated values (csv) file:**

(Note that this cell will not run, but is provided to illustrate syntax.)

1

df3 = pd.read\_csv('/file\_path/file\_name.csv')

## **Attributes and methods**

The **DataFrame** class is powerful and convenient because it comes with a suite of built-in features that simplify common data analysis tasks. These features are known as attributes and methods. An attribute is a value associated with an object or class that is referenced by name using dotted expressions. A method is a function that is defined inside a class body and typically performs an action. A simpler way of thinking about the distinction between attributes and methods is to remember that attributes are characteristics of the object, while methods are actions or operations.

**Common DataFrame attributes**

Data professionals use attributes and methods constantly. Some of the most-used **DataFrame** attributes include:

| **Attribute** | **Description** |
| --- | --- |
| [columns](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.columns.html#pandas.DataFrame.columns) | Returns the column labels of the dataframe |
| [dtypes](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dtypes.html#pandas.DataFrame.dtypes) | Returns the data types in the dataframe |
| [iloc](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html#pandas.DataFrame.iloc) | Accesses a group of rows and columns using integer-based indexing |
| [loc](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html#pandas.DataFrame.loc) | Accesses a group of rows and columns by label(s) or a Boolean array |
| [shape](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html#pandas.DataFrame.shape) | Returns a tuple representing the dimensionality of the dataframe |
| [values](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.values.html#pandas.DataFrame.values) | Returns a NumPy representation of the dataframe |

**Common DataFrame methods**

Some of the most-used **DataFrame** methods include:

| **Method** | **Description** |
| --- | --- |
| [apply()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html#pandas.DataFrame.apply) | Applies a function over an axis of the dataframe |
| [copy()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.copy.html#pandas.DataFrame.copy) | Makes a copy of the dataframe’s indices and data |
| [describe()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe) | Returns descriptive statistics of the dataframe, including the minimum, maximum, mean, and percentile values of its numeric columns; the row count; and the data types |
| [drop()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html#pandas.DataFrame.drop) | Drops specified labels from rows or columns |
| [groupby()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html#pandas.DataFrame.groupby) | Splits the dataframe, applies a function, and combines the results |
| [head(n=5)](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html#pandas.DataFrame.head) | Returns the first n rows of the dataframe (default=5) |
| [info()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html#pandas.DataFrame.info) | Returns a concise summary of the dataframe |
| [isna()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.isna.html#pandas.DataFrame.isna) | Returns a same-sized Boolean dataframe indicating whether each value is null (can also use **isnull()** as an alias) |
| [sort\_values()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sort_values.html#pandas.DataFrame.sort_values) | Sorts by the values across a given axis |
| [value\_counts()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html#pandas.DataFrame.value_counts) | Returns a series containing counts of unique rows in the dataframe |
| [where()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.where.html#pandas.DataFrame.where) | Replaces values in the dataframe where a given condition is false |

These are just a handful of some of the most commonly used attributes and methods—there are many, many more! Some of them can also be used on pandas **Series** objects. For a more detailed list, refer to the [pandas DataFrame documentation](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html), which includes helpful examples of how to use each tool.

## **Selection statements**

Once your data is read into a dataframe, you’ll want to do things with it by selecting, manipulating, and evaluating the data. In this section, you’ll learn how to select rows, columns, combinations of rows and columns, and basic subsets of data.

### Row selection

Rows of a dataframe are selected by their index. The index can be referenced either by name or by numeric position.

#### *loc[]*

**loc[]** lets you select rows by name. Here’s an example:

1

2

3

4

5

6

7

8

df = pd.DataFrame({

   'A': ['alpha', 'apple', 'arsenic', 'angel', 'android'],

   'B': [1, 2, 3, 4, 5],

   'C': ['coconut', 'curse', 'cassava', 'cuckoo', 'clarinet'],

   'D': [6, 7, 8, 9, 10]

   },

   index=['row\_0', 'row\_1', 'row\_2', 'row\_3', 'row\_4'])

df

RunReset

The row index of the dataframe contains the names of the rows. Use **loc[]** to select rows by name:

1

print(df.loc['row\_1'])

RunReset

Inserting just the row index name in selector brackets returns a **Series** object. Inserting the row index name as a list returns a **DataFrame** object:

1

print(df.loc[['row\_1']])

RunReset

To select multiple rows by name, use a list within selector brackets:

1

print(df.loc[['row\_2', 'row\_4']])

RunReset

You can even specify a range of rows by named index:

1

print(df.loc['row\_0':'row\_3'])

RunReset

**Note:** Because you’re using named indices, the returned range includes the specified end index.

#### *iloc[]*

**iloc[]** lets you select rows by numeric position, similar to how you would access elements of a list or an array. Here’s an example.

1

2

3

print(df)

print()

print(df.iloc[1])

RunReset

Inserting just the row index number in selector brackets returns a **Series** object. Inserting the row index number as a list returns a **DataFrame** object:

1

print(df.iloc[[1]])

RunReset

To select multiple rows by index number, use a list within selector brackets:

1

print(df.iloc[[0, 2, 4]])

RunReset

Specify a range of rows by index number:

1

print(df.iloc[0:3])

RunReset

Note that this does not include the row at index three.

### Column selection

#### Bracket notation

Column selection works the same way as row selection, but there are also some shortcuts to make the process easier. For example, to select an individual column, simply put it in selector brackets after the name of the dataframe:

1

print(df['C'])

RunReset

And to select multiple columns, use a list in selector brackets:

1

print(df[['A', 'C']])

RunReset

#### Dot notation

It’s possible to select columns using dot notation instead of bracket notation. For example:

1

print(df.A)

RunReset

Dot notation is often convenient and easier to type. However, it can make your code more difficult to read, especially in longer statements involving method chaining or condition-based selection. For this reason, bracket notation is often preferred.

#### *loc[]*

You can also use **loc[]** notation:

1

2

3

4

print(df)

print()

print(df.loc[:, ['B', 'D']])

RunReset

Note that when using **loc[]** to select columns, you must specify rows as well. In this example, all rows were selected using just a colon (**:**).

#### *iloc[]*

Similarly, you can use **iloc[]** notation. Again, when using **iloc[]**, you must specify rows, even if you want to select all rows:

1

print(df.iloc[:, [1,3]])

RunReset

### Select rows and columns

Both **loc[]** and **iloc[]** can be used to select specific rows and columns together.

#### *loc[]*

1

print(df.loc['row\_0':'row\_2', ['A','C']])

RunReset

Again, notice that when using **loc[]** to select a range, the final element in the range is included in the results.

#### *iloc[]*

1

print(df.iloc[[2, 4], 0:3])

RunReset

Note that, when using rows with named indices, you cannot mix numeric and named notation. For example, the following code will throw an error:

1

print(df.loc[0:3, ['D']])

RunReset

To view rows **[0:3]** at column ‘**D**’ (if you don’t know the index number of column D), you’d have to use selector brackets after an **iloc[]** statement:

1

2

3

4

5

# This is most convenient for VIEWING:

print(df.iloc[0:3][['D']])

# But this is best practice/more stable for assignment/manipulation:

print(df.loc[df.index[0:3], 'D'])

RunReset

However, in many (perhaps most) cases your rows will not have named indices, but rather numeric indices. In this case, you can mix numeric and named notation. For example, here’s the same dataset, but with numeric indices instead of named indices.

1

2

3

4

5

6

7

8

df = pd.DataFrame({

   'A': ['alpha', 'apple', 'arsenic', 'angel', 'android'],

   'B': [1, 2, 3, 4, 5],

   'C': ['coconut', 'curse', 'cassava', 'cuckoo', 'clarinet'],

   'D': [6, 7, 8, 9, 10]

   },

   )

df

RunReset

Notice that the rows are enumerated now. Now, this code will execute without error:

1

print(df.loc[0:3, ['D']])

RunReset

## **Key takeaways**

Pandas dataframes are a convenient way to work with tabular data. Each row and each column can be represented by a pandas **Series**, which is similar to a one-dimensional array. Both dataframes and series have a large collection of methods and attributes to perform common tasks and retrieve information. Pandas also has its own special notation to select data. As you work more with pandas, you’ll become more comfortable with this notation and its many applications in data science.

## Resources for more information

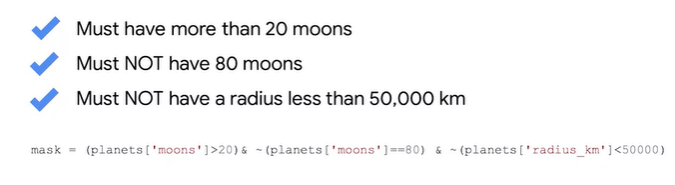
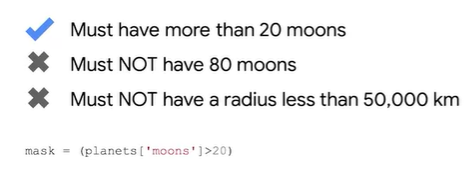
* [pandas DataFrame class documentation](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)
* [pandas Series class documentation](https://pandas.pydata.org/docs/reference/series.html)
* [pandas selection documentation](https://pandas.pydata.org/docs/user_guide/10min.html#selection)

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 12. Boolean masking.

I'm ready to use the follow-along guide while watching this video.

**Boolean masking**

A filtering technique that overlays a Boolean grid onto a dataframe in order to select only the values in the dataframe in order to select only the values in the dataframe that align with the True values of the grid.

# Boolean masking in pandas

Now that you know how to select data in pandas by referring to rows and columns, the next step is to learn how to use Boolean masks. Data professionals use Boolean masks to select data in pandas based on conditions. In this reading, you will discover Boolean masking and how to use pandas’ logical operators to form multi-conditional selection statements. Understanding the fundamentals of pandas will help make your work as a data professional easier and more efficient.

## Boolean masks

You know that Boolean is used to describe any binary variable whose possible values are true or false. With pandas, **Boolean masking,** also called **Boolean indexing**, is used to overlay a Boolean grid onto a dataframe's index in order to select only the values in the dataframe that align with the **True** values of the grid.

Return to the example from the video. Suppose you have a dataframe of planets, their radii, and their number of moons:

| **planet** | **radius\_km** | **moons** |
| --- | --- | --- |
| Mercury | 2,440 | 0 |
| Venus | 6,052 | 0 |
| Earth | 6,371 | 1 |
| Mars | 3,390 | 2 |
| Jupiter | 69,911 | 80 |
| Saturn | 58,232 | 83 |
| Uranus | 25,362 | 27 |
| Neptune | 24,622 | 14 |

Now suppose that you want to keep the rows of any planets that have fewer than 20 moons and filter out the rest. A Boolean mask is a pandas **Series** object indicating whether this condition is true or false for each value in the **moons** column:

|  | **Moons < 20?** |
| --- | --- |
| **0** | **True** |
| **1** | **True** |
| **2** | **True** |
| **3** | **True** |
| **4** | **False** |
| **5** | **False** |
| **6** | **False** |
| **7** | **True** |

The **dtype** contained in this series is **bool**. Boolean masking effectively overlays this Boolean series onto the dataframe’s index. The result is that any rows in the dataframe that are indicated as **False** in the Boolean mask get filtered out, and any rows that are indicated as **True** remain in the dataframe:

| **planet** | **radius\_km** | **moons** |
| --- | --- | --- |
| Mercury | 2,440 | 0 |
| Venus | 6,052 | 0 |
| Earth | 6,371 | 1 |
| Mars | 3,390 | 2 |
| Neptune | 24,622 | 14 |

### Coding Boolean masks in pandas

Here is how to perform this operation in pandas.

Begin with a **DataFrame** object.

1

2

3

4

5

6

7

8

data = {'planet': ['Mercury', 'Venus', 'Earth', 'Mars',

                   'Jupiter', 'Saturn', 'Uranus', 'Neptune'],

       'radius\_km': [2440, 6052, 6371, 3390, 69911, 58232,

                     25362, 24622],

       'moons': [0, 0, 1, 2, 80, 83, 27, 14]

        }

df = pd.DataFrame(data)

df

RunReset

Then, write a logical statement. Remember, the objective is to keep planets that have fewer than 20 moons and filter out the rest.

1

print(df['moons'] < 20)

RunReset

This results in a **Series** object of **dtype: bool** that consists of the row indices, where each index contains a **True** or **False** value depending on whether that row satisfies the given condition. This is the Boolean mask. To apply this mask to the dataframe, simply insert this statement into selector brackets and apply it to your dataframe:

1

print(df[df['moons'] < 20])

RunReset

You can also assign the Boolean mask to a named variable and then apply that to your dataframe:

1

2

mask = df['moons'] < 20

df[mask]

RunReset

Note that this doesn’t permanently modify your dataframe. It only gives a filtered view of it.

1

2

df

RunReset

However, you can assign the result to a named variable:

1

2

3

mask = df['moons'] < 20

df2 = df[mask]

df2

RunReset

And if you want to select just the planet column as a **Series** object, you can use regular selection tools like **loc[]**:

1

2

mask = df['moons'] < 20

df.loc[mask, 'planet']

RunReset

### Complex logical statements

In statements that use multiple conditions, pandas uses logical operators to indicate which data to keep and which to filter out. These operators are:

| **Operator** | **Logic** |
| --- | --- |
| **&** | **and** |
| **|** | **or** |
| **~** | **not** |

**Important: Each component of a multi-conditional logical statement must be in parentheses.** Otherwise, the statement will throw an error or, worse, return something that isn’t what you intended.

For example, here is how to create a Boolean mask that selects all planets that have fewer than 10 moons or greater than 50 moons:

1

2

mask = (df['moons'] < 10) | (df['moons'] > 50)

mask

RunReset

Notice that each condition is self-contained in a set of parentheses, and the two conditions are separated by the logical operator, **|**(or). To apply the mask, call the dataframe and put the statement or the variable it’s assigned to in selector brackets:

1

2

mask = (df['moons'] < 10) | (df['moons'] > 50)

df[mask]

RunReset

Here’s an example of how to select all planets that have more than 20 moons, but not planets with 80 moons and not planets with a radius less than 50,000 km:

1

2

mask = (df['moons'] > 20) & ~(df['moons'] == 80) & ~(df['radius\_km'] < 50000)

df[mask]

RunReset

Note that this returns the same result as the following:

1

2

mask = (df['moons'] > 20) & (df['moons'] != 80) & (df['radius\_km'] >= 50000)

df[mask]

RunReset

Working with pandas dataframes, using their attributes and methods, and selecting data using Boolean masks are some of the core daily activities of a data professional. You’ll soon be using these tools often as you progress on your journey with pandas.

## Key takeaways

A Boolean mask is a method of applying a filter to a dataframe. The mask overlays a Boolean grid over your dataframe in order to select only the values in the dataframe that align with the True values of the grid. To create Boolean comparisons, pandas has its own logical operators. These operators are:

* **&** (and)
* **|** (or)
* **~** (not)

Each criterion of a multi-conditional selection statement must be enclosed in its own set of parentheses. With practice, making complex selection statements in pandas is possible and efficient.

## Resources for more information

* [pandas Boolean indexing documentation](https://pandas.pydata.org/docs/user_guide/indexing.html#boolean-indexing)

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 13. Grouping and aggregation.

I'm ready to use the follow-along guide while watching this video.

**groupby()**

A pandas DataFrame method that groups rows of the dataframe together based on their values at one or more columns, which allows further analysis of the groups.

**agg()**

Short for “aggregate”. A pandas groupby method that allows you to apply multiple calculations to groups of data.

# More on grouping and aggregation

You’ve discovered that pandas is a Python library that facilitates reviewing and manipulating tabular data. In addition, **groupby()** and **agg()** are essential **DataFrame** methods that data professionals use to group, aggregate, summarize, and better understand data. In this reading, you’ll review how these functions work, as well as when and how to apply them.

## groupby()

The [groupby()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html) function is a method that belongs to the **DataFrame** class. It works by splitting data into groups based on specified criteria, applying a function to each group independently, then combining the results into a data structure. When applied to a dataframe, the function returns a groupby object. This groupby object serves as the foundation for different data manipulation operations, including:

* Aggregation: Computing summary statistics for each group
* Transformation: Applying functions to each group and returning modified data
* Filtration: Selecting specific groups based on certain conditions
* Iteration: Iterating over groups or values

Here are some examples that use the **groupby()** function on a dataframe consisting of different articles of clothing:

1

2

3

4

5

6

7

clothes = pd.DataFrame({'type': ['pants', 'shirt', 'shirt', 'pants', 'shirt', 'pants'],

                       'color': ['red', 'blue', 'green', 'blue', 'green', 'red'],

                       'price\_usd': [20, 35, 50, 40, 100, 75],

                       'mass\_g': [125, 440, 680, 200, 395, 485]})

clothes

RunReset

Grouping the dataframe by **type** results in a **DataFrameGroupBy** object:

1

2

3

grouped = clothes.groupby('type')

print(grouped)

print(type(grouped))

RunReset

However, an aggregation function can be applied to the groupby object:

1

2

grouped = clothes.groupby('type')

grouped.mean()

RunReset

In the preceding example, **groupby()** combined all the items into groups based on their type and returned a **DataFrame** object containing the mean of each group for each numeric column in the dataframe. Note: In future versions of pandas it will be necessary to specify a **numeric\_only** parameter when applying certain aggregation functions—like mean—to a groupby object. **numeric\_only** refers to the datatype of each column. In earlier versions of pandas (like the version on this platform) it isn't necessary to specify **numeric\_only=True**, but in future versions this must be done. Otherwise, it will be necessary to indicate the specific columns to be captured.)

In addition, groups may be created based on multiple columns:

1

2

clothes.groupby(['type', 'color']).min()

RunReset

In the preceding example, **groupby()** was called directly on the clothes dataframe. The data was grouped first by **type**, then by **color**. This resulted in four groups—the number of different existing combinations of values for type and color. Then, the **min()** function was applied to the result to filter each group by its minimum value.

To simply return the number of observations there are in each group, use the **size()** method. This will result in a **Series** object with the relevant information:

1

2

clothes.groupby(['type', 'color']).size()

RunReset

### Built-in aggregation functions

The previous examples demonstrated the **mean()**, **min()**, and **size()** aggregation functions applied to groupby objects. There are many available built-in aggregation functions. Some of the more commonly used include:

* **count()**: The number of non-null values in each group
* **sum()**: The sum of values in each group
* **mean()**: The mean of values in each group
* **median()**: The median of values in each group
* **min()**: The minimum value in each group
* **max()**: The maximum value in each group
* **std()**: The standard deviation of values in each group
* **var()**: The variance of values in each group

## agg()

The [agg()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html) function is useful when you want to apply multiple functions to a dataframe at the same time. **agg()** is a method that belongs to the **DataFrame** class. It stands for “aggregate.” Its most important parameters are:

* **func**: The function to be applied
* **axis**: The axis over which to apply the function (default= 0).

Following are some examples of how **agg()** can be used. Note that they demonstrate how this function can be used by itself (without **groupby()**). Note also that, due to platform limitations, some of the following code blocks are not executable. In these cases, output is provided as an image. Here is the original **clothes** dataframe again as a reminder:

1

2

clothes

RunReset

The following example applies the **sum()** and **mean()** functions to the **price** and **mass\_g** columns of the **clothes** dataframe.

1

clothes[['price\_usd', 'mass\_g']].agg(['sum', 'mean'])

**Output:**

A screenshot of a computer

Description automatically generated

Notice the following:

* The two columns are subset from the dataframe before applying the **agg()** method. If you don’t subset the relevant columns first, **agg()** will attempt to apply **sum()** and **mean()** to all of the columns, which wouldn’t work because some columns contain strings. (Technically, **sum()** would work, but it would return something useless because it would just combine all the strings into one long string.)
* The **sum()** and **mean()** functions are entered as strings in a list, without their parentheses. This will work for any built-in aggregation function.

In this next example, different functions are applied to different columns.

1

2

3

clothes.agg({'price\_usd': 'sum',

            'mass\_g': ['mean', 'median']

            })

**Output:**

A screenshot of a graph

Description automatically generated

Notice the following:

* Columns are not subset from the dataframe before applying the **agg()** function. This is unnecessary because the columns are specified within the **agg()** function itself.
* The argument to the **agg()** function is a dictionary whose keys are columns and whose values are the functions to be applied to those columns. If multiple functions are applied to a column, they are entered as a list. Again, each built-in function is entered as a string without parentheses.
* The resulting dataframe contains **NaN** values where a given function was not designated to be used.

The following example applies the **sum()** and **mean()** functions across axis 1. In other words, instead of applying the functions down each column, they’re applied over each row.

1

clothes[['price\_usd', 'mass\_g']].agg(['sum', 'mean'], axis=1)

**Output:**

A screenshot of a cell phone

Description automatically generated

## groupby() **with** agg()

The **groupby()** and **agg()** functions are often used together. In such cases, first apply the **groupby()** function to a dataframe, then apply the **agg()** function to the result of the groupby. For reference, here is the **clothes** dataframe once again.

1

2

clothes

RunReset

In the following example, the items in **clothes** are grouped by **color**, then each of those groups has the **mean()** and **max()** functions applied to them at the **price\_usd** and **mass\_g** columns.

1

2

3

clothes.groupby('color').agg({'price\_usd': ['mean', 'max'],

                             'mass\_g': ['mean', 'max']})

RunReset

## **MultiIndex**

You might have noticed that, when functions are applied to a groupby object, the resulting dataframe has tiered indices. This is an example of **MultiIndex**. MultiIndex is a hierarchical system of dataframe indexing. It enables you to store and manipulate data with any number of dimensions in lower dimensional data structures such as series and dataframes. This facilitates complex data manipulation.

This course will not require any deep knowledge of hierarchical indexing, but it’s helpful to be familiar with it. Consider the following example:

1

2

grouped = clothes.groupby(['color', 'type']).agg(['mean', 'min'])

grouped

RunReset

Notice that **color** and **type** are positioned lower than the column names in the output. This indicates that **color** and **type** are no longer columns, but named row indices. Similarly, notice that **price\_usd** and **mass\_g** are positioned above **mean** and **min** in the output of column names, indicating a hierarchical column index.

If you inspect the row index, you’ll get a **MultiIndex** object containing information about the row indices:

1

2

grouped.index

RunReset

The column index shows a **MultiIndex** object containing information about the column indices:

1

2

grouped.columns

RunReset

To perform selection on a dataframe with a MultiIndex, use **loc[]** selection and put indices in parentheses. Here are some examples on **grouped**, which is a dataframe with a two-level row index and a two-level column index. For reference, here is the **grouped** dataframe:

1

2

grouped

RunReset

To select a first-level (top) column:

1

2

grouped.loc[:, 'price\_usd']

RunReset

To select a second-level (bottom) column:

1

2

grouped.loc[:, ('price\_usd', 'min')]

RunReset

To select first-level (left-most) row:

1

2

grouped.loc['blue', :]

RunReset

To select a bottom-level (right-most) row:

1

2

grouped.loc[('green', 'shirt'), :]

RunReset

And you can even select individual values:

1

2

grouped.loc[('blue', 'shirt'), ('mass\_g', 'mean')]

RunReset

If you want to remove the row MultiIndex from a groupby result, include **as\_index=False** as a parameter to your **groupby()** statement:

1

2

clothes.groupby(['color', 'type'], as\_index=False).mean()

RunReset

Notice how **color** and **type** are no longer row indices, but named columns. The row indices are the standard enumeration beginning from zero.

Again, you will not be expected to do any complex manipulations of hierarchically indexed data in this course, but it’s helpful to have a basic understanding of how MultIndex works, especially because **groupby()** manipulations typically result in a MultiIndex dataframe by default.

## Key takeaways

**groupby()** will be an essential function in your work as a data professional, as it enables efficient combining and analysis of data. Similarly, **agg()** will help you apply multiple functions dynamically across a specified axis of a dataframe. Either on their own or when used together, these tools give data professionals deep access to data and help bring about successful projects.

## Question

The video you are viewing contains coding instruction and examples. To follow along with the instructor, open the [Annotated follow-along guide: Data structures in Python](https://www.coursera.org/learn/get-started-with-python/ungradedLab/T6uuj/annotated-follow-along-guide-data-structures-in-python) in a new browser and navigate to Section 14. Merging and joining data.

I'm ready to use the follow-along guide while watching this video.

**pandas functions**

-concat()

-merge()

**concat()**

A pandas function that combines data either by adding it horizontally as new columns for existing rows, or vertically as new rows for existing columns.

**Keys**

The shared points of reference between different dataframes-what to match on

# Test your knowledge: Dataframes with pandas

Review Learning Objectives

### 1.

Question 1

Fill in the blank: In pandas, a \_\_\_\_\_ is a one-dimensional, labeled array.

1 / 1 point

dataframe

CSV file

key

series

Correct

A series is a one-dimensional, labeled array. Series objects are most often used to represent individual columns or rows of a dataframe.

### 2.

Question 2

In pandas, what is Boolean masking used for?

1 / 1 point

 Merging data in a dataframe

Deleting data from a dataframe

Filtering data in a dataframe

Adding data to a dataframe

Correct

In pandas, Boolean masking is used for filtering data in a dataframe. Boolean masking is a filtering technique that overlays a Boolean grid onto a dataframe in order to select only the values in the dataframe that align with the True values of the grid.

### 3.

Question 3

What is a pandas method that groups rows of a dataframe together based on their values at one or more columns?

1 / 1 point

**keys()**

**groupby()**

**values()**

**agg()**

Correct

**groupby()** is a pandas method that groups rows of a dataframe together based on their values at one or more columns. This allows further analysis of the groups.

### 4.

Question 4

A data professional wants to join two dataframes together. The dataframes contain identically formatted data that needs to be combined vertically. What pandas function can the data professional use to join the dataframes?

1 / 1 point

**type()**

**insert()**

**concat()**

**merge()**

Correct

The data professional can use the **concat()** function to join the dataframes. **concat()** is a pandas function that combines data either by adding it horizontally as new columns for existing rows, or vertically as new rows for existing columns.

**Review**

-Lists

-Tuples

-Dictionaries

-Sets

-Arrays

-NumPy

-pandas

# Glossary terms from module 4

# Terms and definitions from Course 2, Module 4

**agg()**: A pandas groupby method that allows the user to apply multiple calculations to groups of data

**Aliasing**: A process that allows the user to assign an alternate name—or alias—to something

**append()**: A method that adds an element to the end of a list

**Boolean masking**: A filtering technique that overlays a Boolean grid onto a dataframe in order to select only the values in the dataframe that align with the True values of the grid

**concat()**: A pandas function that combines data either by adding it horizontally as new columns for existing rows or vertically as new rows for existing columns

**CSV file**: A plaintext file that uses commas to separate distinct values from one another; Stands for "comma-separated values”

**Data structure**: A collection of data values or objects that contain different data types

**DataFrame**: A two-dimensional, labeled data structure with rows and columns

**dict()**: A function used to create a dictionary

**Dictionary**: A data structure that consists of a collection of key-value pairs

**difference()**: A function that finds the elements present in one set but not the other

**dtype**: A NumPy attribute used to check the data type of the contents of an array

**Global variable**: A variable that can be accessed from anywhere in a program or script

**groupby()**: A pandas DataFrame method that groups rows of the dataframe together based on their values at one or more columns, which allows further analysis of the groups

**iloc[]**: A type of notation in pandas that indicates when the user wants to select by integer-location-based position

**Immutability**: The concept that a data structure or element’s values can never be altered or updated

**Import statement**: A statement that uses the import keyword to load an external library, package, module, or function into the computing environment

**Inner join**: A way of combining data such that only the keys that are in both dataframes get included in the merge

**insert()**: A function that takes an index as the first parameter and an element as the second parameter, then inserts the element into a list at the given index

**intersection()**: A function that finds the elements that two sets have in common

**items()**: A dictionary method to retrieve both the dictionary’s keys and values

**Keys**: The shared points of reference between different dataframes

**keys()**: A dictionary method to retrieve only the dictionary’s keys

**Left join**: A way of combining data such that all of the keys in the left dataframe are included, even if they aren’t in the right dataframe

**Library**: A reusable collection of code; also referred to as a “package”

**List**: A data structure that helps store and manipulate an ordered collection of items

**List comprehension**: Formulaic creation of a new list based on the values in an existing list

**loc[]**: Notation that is used to select pandas rows and columns by name

**matplotlib**: A library for creating static, animated, and interactive visualizations in Python

**merge()**: A pandas function that joins two dataframes together; it only combines data by extending along axis one horizontally

**Module**: A simple Python file containing a collection of functions and global variables

**Mutability**: The ability to change the internal state of a data structure

**N-dimensional array**: The core data object of NumPy; also referred to as “ndarray”

**NaN**: How null values are represented in pandas; stands for “not a number”

**ndim**: A NumPy attribute used to check the number of dimensions of an array

**Nested loop**: A loop inside of another loop

**NumPy**: An essential library that contains multidimensional array and matrix data structures and functions to manipulate them

**Outer join**: A way of combining data such that all of the keys from both dataframes get included in the merge

**pandas**: A powerful library built on top of NumPy that’s used to manipulate and analyze tabular data

**pop()**: A method that extracts an element from a list by removing it at a given index

**remove()**: A method that removes an element from a list

**reshape()**: A NumPy method used to change the shape of an array

**Right join**: A way of combining data such that all the keys in the right dataframe are included—even if they aren’t in the left dataframe

**Seaborn**: A visualization library based on matplotlib that provides a simpler interface for working with common plots and graphs

**Sequence**: A positionally ordered collection of items

**Series**: A one-dimensional, labeled array where the data type must be the same for all the data in a given series

**Set**: A data structure in Python that contains only unordered, non-interchangeable elements

**set()**: A function that takes an iterable as an argument and returns a new set object

**shape**: A NumPy attribute used to check the shape of an array

**symmetric\_difference()**: A function that finds elements from both sets that are mutually not present in the other

**Tabular data**: Data that is in the form of a table, with rows and columns

**Tuple**: An immutable sequence that can contain elements of any data type

**tuple()**: A function that transforms input into tuples

**type()**: A function used to identify the type of data in a list

**union()**: A function that finds all the elements from both sets

**values()**: A dictionary method to retrieve only the dictionary’s values

**Vectorization**: A process that enables operations to be performed on multiple components of a data object at the same time

# Terms and definitions from previous modules

## A

**Algorithm**: A set of instructions for solving a problem or accomplishing a task

**Argument**: Information given to a function in its parentheses

**Assignment**: The process of storing a value in a variable

**Attribute**: A value associated with an object or class which is referenced by name using dot notation

## B

**Boolean**: A data type that has only two possible values, usually true or false

**Branching**: The ability of a program to alter its execution sequence

**break:** A keyword that lets a user escape a loop without triggering any ELSE statement that follows it in the loop

## C

**Cells**: The modular code input and output fields into which Jupyter Notebooks are partitioned

**Class**: An object’s data type that bundles data and functionality together

**Comparator**: An operator that compares two values and produces Boolean values (True/False)

**Computer programming**: The process of giving instructions to a computer to perform an action or set of actions

**Concatenate**: To link or join together

## D

**Data type**: An attribute that describes a piece of data based on its values, its programming language, or the operations it can perform

**def**: A keyword that defines a function at the start of the function block

**Docstring**: A string at the beginning of a function’s body that summarizes the function’s behavior and explains its arguments and return values

**Dot notation**: How to access the methods and attributes that belong to an instance of a class

**Dynamic typing**: Variables that can point to objects of any data type

## E

**elif**: A reserved keyword that executes subsequent conditions when the previous conditions are not true

**else**: A reserved keyword that executes when preceding conditions evaluate as False

**Escape character**: A character that changes the typical behavior of the characters that follow it

**Explicit conversion**: The process of converting a data type of an object to a required data type

**Expression**: A combination of numbers, symbols, or other variables that produce a result when evaluated

## F

**Float**: A data type that represents numbers that contain decimals

**For loop**: A piece of code that iterates over a sequence of values

**format()**: A string method that formats and inserts specific substrings into designated places within a larger string

**Function**: A body of reusable code for performing specific processes or tasks

## I

**if**: A reserved keyword that sets up a condition in Python

**Immutable data type**: A data type in which the values can never be altered or updated

**Implicit conversion**: The process Python uses to automatically convert one data type to another without user involvement

**index()**: A string method that outputs the index number of a character in a string

**Indexing**: A way to refer to the individual items within an iterable by their relative position

**Integer**: A data type used to represent whole numbers without fractions

**Iterable**: An object that’s looped, or iterated, over

**Iteration**: The repeated execution of a set of statements, where one iteration is the single execution of a block of code

## J

**Jupyter Notebook**: An open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text

## K

**Keyword**: A special word in a programming language that is reserved for a specific purpose and that can only be used for that purpose

## L

**Logical operator**: An operator that connects multiple statements together and performs complex comparisons

**Loop**: A block of code used to carry out iterations

## M

**Markdown**: A markup language that lets the user write formatted text in a coding environment or plain-text editor

**Method**: A function that belongs to a class and typically performs an action or operation

**Modularity**: The ability to write code in separate components that work together and that can be reused for other programs

**Modulo**: An operator that returns the remainder when one number is divided by another

## N

**Naming conventions**: Consistent guidelines that describe the content, creation date, and version of a file in its name

**Naming restrictions**: Rules built into the syntax of the language itself that must be followed

## O

**Object**: An instance of a class; a fundamental building block of Python

**Object-oriented programming**: A programming system that is based around objects which can contain both data and code that manipulates that data

## P

**Programming languages**: The words and symbols used to write instructions for computers to follow

## R

**range()**: A Python function that returns a sequence of numbers starting from zero, increments by 1 by default, and stops before the given number

**Refactoring**: The process of restructuring code while maintaining its original functionality

**return**: A reserved keyword in Python that makes a function produce new results which are saved for later use

**Reusability**: The capability to define code once and use it many times without having to rewrite it

## S

**Self-documenting code**: Code written in a way that is readable and makes its purpose clear

**String**: A sequence of characters and punctuation that contains textual information

**String slice**: A portion of a string that can contain more than one character; also referred to as a substring

**Syntax**: The structure of code words, symbols, placement, and punctuation

## V

**Variable**: A named container which stores values in a reserved location in the computer’s memory

## W

**While** **loop**: A loop that instructs the computer to continuously execute the code based on the value of a condition

# Module 4 challenge

### 1.

Question 1

In Python, what data structure helps store and manipulate an ordered collection of items?

List

### 2.

Question 2

A data professional is working with a list named **cities** that contains data on global cities. What Python code can they use to add the string **'Tokyo'** to the end of the list?

**cities.append(‘Tokyo’)**

### 3.

Question 3

In Python, what types of data can tuples contain? Select all that apply.

- Floats

- Integers

- Strings

### 4.

Question 4

Which of the following statements accurately describe Python dictionaries? Select all that apply.

-Dictionaries consist of collections of key-value pairs.

-Dictionaries are instantiated with the **dict()** function.

### 5.

Question 5

A data professional is working with a dictionary named **employees** that contains employee data for a healthcare company. What Python code can they use to retrieve only the dictionary’s values?

**employees.values()**

### 6.

Question 6

A data professional is working with two Python sets. What function can they use to find the elements present in one set, but not the other?

1 / 1 point

**difference()**

### 7.

Question 7

Fill in the blank: In Python, \_\_\_\_\_ typically contain a collection of functions and global variables.

1 / 1 point

Modules

### 8.

Question 8

A data professional is working with a NumPy array that has three rows and two columns. They want to change the data into two rows and three columns. What method can they use to do so?

**reshape()**

### 9.

Question 9

A data professional is working with a pandas dataframe named **sales** that contains sales data for a retail website. They want to know the price of the least expensive item. What code can they use to calculate the minimum value of the **Price** column?

**sales[‘Price’].min()**

### 10.

Question 10

In pandas, what is the difference between the **iloc[]** and **loc[]** methods?

**iloc[]** selects dataframe rows and columns by index; **loc[]** selects dataframe rows and columns by name.

### 11.

Question 11 (NOT FINAL)

A data professional wants to merge two pandas dataframes. They want to join the data so all of the keys from both dataframes get included in the merge. What technique can they use to do so?

Outer join

**Module 5:**

**Tidy dataset**

-Easy to manipulate, model, and visualize

-Each variable is a column

-Each observation is a row

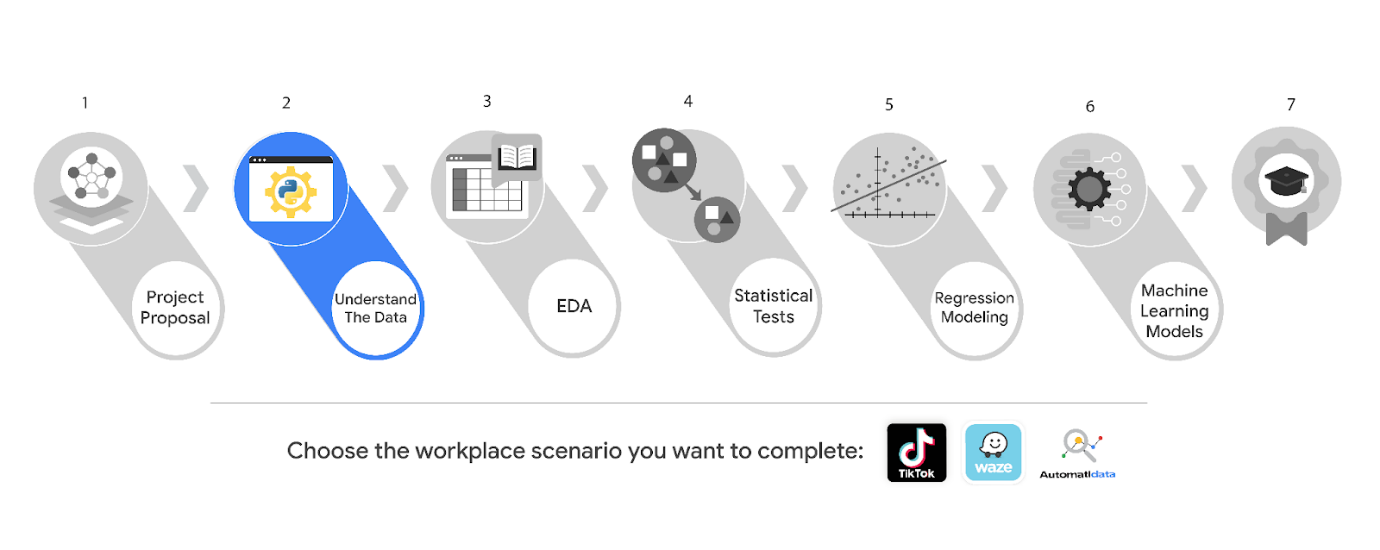
# Explore your Course 2 workplace scenarios

# **Overview**

This certificate offers you a choice of several different workplace scenarios to use when completing each end-of-course project:

* Automatidata, featuring a fictional data consulting firm
* TikTok, created in partnership with the short-form video hosting company
* Waze, created in partnership with the realtime driving directions app

Each scenario offers you an opportunity to apply your skills and create work samples to share when applying for jobs; so, you will be practicing similar skills regardless of the workplace scenario. It is recommended that you work with the same scenario for each end-of-course project to have a cohesive experience. However, you are welcome to investigate any of the workplace scenarios you are interested in as you progress through the program.



***Reminder:*** We recommend that you choose one workplace scenario to follow for all end-of-course projects to ensure end-to-end project development.

The minimum requirement to earn your Advanced Data Analytics Certificate is to complete the end-of-course project, using one workplace scenario, for each course. You may complete the project for as many of the workplace scenarios as you wish. Completing the project for more than one workplace scenario in a single course offers you additional practice and work examples you can add to your portfolio and share with prospective employers during your job search.

This reading offers an overview of all available workplace scenarios. Before moving on, identify the scenario you would like to complete for the Course 2 end-of-course project.

## **Course 2 workplace scenarios**

## Automatidata



**Project goal:**

In this fictional scenario, the New York City Taxi and Limousine Commission (TLC) has approached the data consulting firm Automatidata to develop an app that enables TLC riders to estimate the taxi fares in advance of their ride.

**Background:**

Since 1971, TLC has been regulating and overseeing the licensing of New York City's taxi cabs, for-hire vehicles, commuter vans, and paratransit vehicles.

**Scenario:**

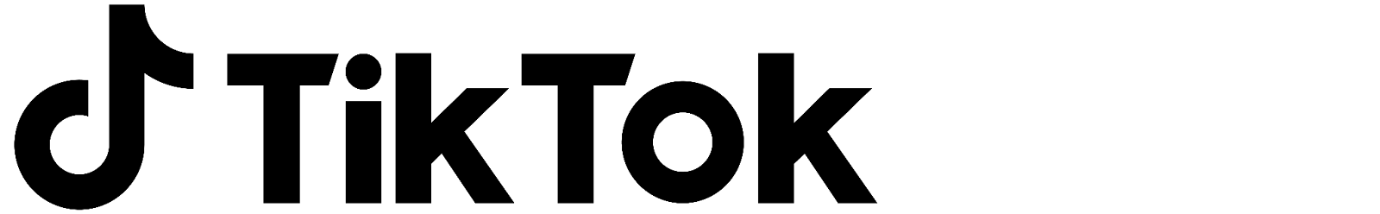
You have received notice that the recently submitted New York City TLC project proposal has been approved. The Automatidata team now has access to the New York City TLC data to analyze, identify key variables, and prepare for exploratory data analysis.

**Course 2 tasks:**

* Load data, explore, and extract the New York City TLC data with Python
* Use custom functions to organize the information within the New York City TLC dataset
* Build a dataframe for the New York City TLC project
* Create an executive summary for Automatidata

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

## **TikTok**



**Project goal:**

The TikTok data team is developing a machine learning model for classifying claims made in videos submitted to the platform.

**Background:**

TikTok is the leading destination for short-form mobile video. The platform is built to help imaginations thrive. TikTok's mission is to create a place for inclusive, joyful, and authentic content–where people can safely discover, create, and connect.

**Scenario:**

As a data analyst on TikTok's data team, you'll help by preparing the data needed for the claims classification project. You’ll build a dataframe, organize the claims data for the process of exploratory data analysis, and update the team on your progress and insights.

**Course 2 tasks:**

* Build a dataframe for the TikTok dataset
* Read in data from TikTok csv file
* Display rows within dataframe
* Examine data type of each column
* Gather descriptive statistics
* Visualize the TikTok data in Python
* Report to TikTok’s data team through an executive summary

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

## **Waze**



**Project goal:**

Waze leadership has asked your data team to develop a machine learning model to predict user churn. Churn quantifies the number of users who have uninstalled the Waze app or stopped using the app. This project focuses on monthly user churn. An accurate model will help prevent churn, improve user retention, and grow Waze’s business.

**Background:**

Waze’s free navigation app makes it easier for drivers around the world to get to where they want to go. Waze’s community of map editors, beta testers, translators, partners, and users helps make each drive better and safer.

**Scenario:**

Your team is in the early stages of their user churn project. Your project proposal has been approved and your team has been given access to Waze’s user data. To get clear insights, the data must first be inspected, organized, and prepared for analysis.

**Course 2 tasks:**

* Import data
* Create a dataframe
* Inspect data
* Identify outliers
* Create a data visualization
* Share an executive summary with the Waze data team

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

## **Key Takeaways**

In Course 2, Get Started with Python, you were introduced to some basics of the Python programming language. You explored syntax, loops, strings, lists, dictionaries, object-oriented programming, and explored how data professionals use code on the job.

### **Course 2 skills:**

* Code with Python
* Create data visualization
* Use comments to enhance code readability
* Work within a Jupyter Notebook
* Share insights and ideas with stakeholders

### **Course 2 end-of-course project deliverables:**

* Build a dataframe
* Create an executive summary

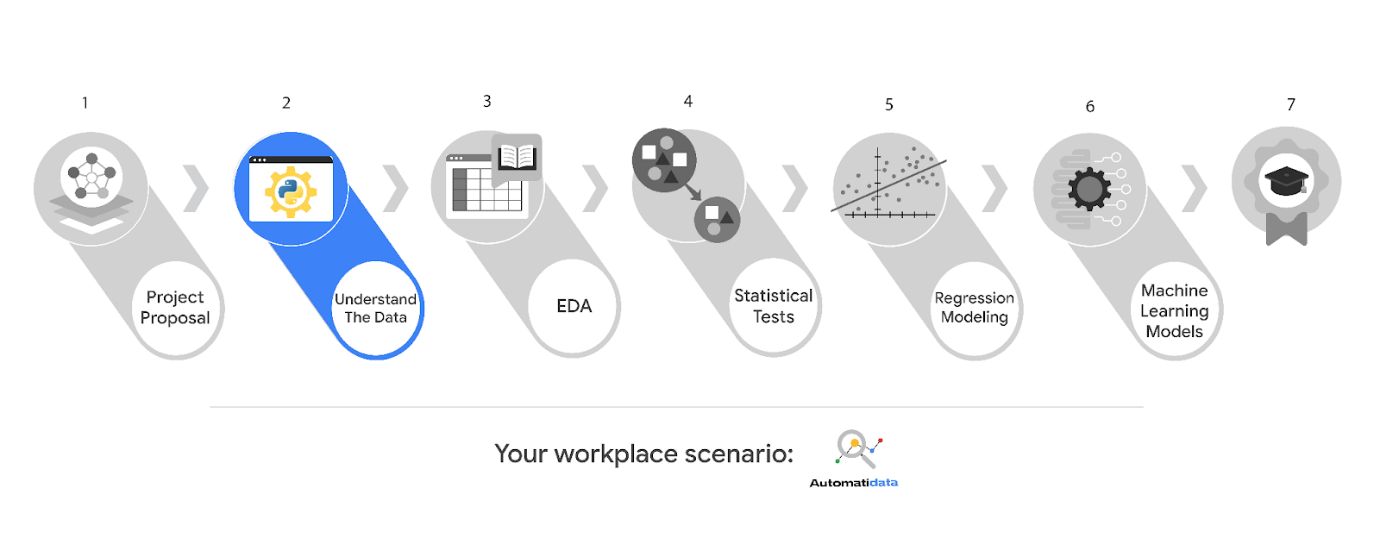
The end-of-course portfolio projects are designed for you to apply your data analytical skills within a workplace scenario. No matter which scenario you work with, you will practice your ability to discuss data analytic topics with coworkers, internal team members, and external clients.

As a reminder, you are required to complete one project for each course. To gain additional practice, or to add more samples to your portfolio, you may complete as many of the scenarios as you wish.

# Course 2 end-of-course portfolio project overview: Automatidata

# **Learn about the Course 2 Automatidata workplace scenario!**

The end-of-course project in Course 2 focuses on your ability to understand the data needed for a project. As a reminder, in Course 1 you developed a project proposal that outlined milestones, which progress with each of the end-of-course projects. A visual representation is provided in the graphic shown here:



Learn more about the project, your role, and expectations in this reading.

## **Background on the Automatidata scenario**

Automatidata works with its clients to transform their unused and stored data into useful solutions, such as performance dashboards, customer-facing tools, strategic business insights, and more. They specialize in identifying a client’s business needs and utilizing their data to meet those business needs.

Automatidata is consulting for the New York City Taxi and Limousine Commission (TLC). New York City TLC is an agency responsible for licensing and regulating New York City's taxi cabs and for-hire vehicles. The agency has partnered with Automatidata to develop a regression model that helps estimate taxi fares before the ride, based on data that TLC has gathered.

The TLC data comes from over 200,000 taxi and limousine licensees, making approximately one million combined trips per day.

**Note:** This project's dataset was created for pedagogical purposes and may not be indicative of New York City taxi cab riders' behavior.

## **Team members at Automatidata and the New York City TLC**

### **Automatidata Team Members**

* Udo Bankole, Director of Data Analysis
* Deshawn Washington, Data Analysis Manager
* Luana Rodriquez, Senior Data Analyst
* Uli King, Senior Project Manager

Your teammates at Automatidata have technical experience with data analysis and data science. However, you should always be sure to keep summaries and messages to these team members concise and to the point.

### **New York City TLC Team Members**

* Juliana Soto, Finance and Administration Department Head
* Titus Nelson, Operations Manager

The TLC team members are program managers who oversee operations at the organization. Their roles are not highly technical, so be sure to adjust your language and explanation accordingly.

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

### **Project background**

Automatidata is in the earliest stages of the TLC project. The following tasks are needed before the team can begin the data analysis process:

* Build a dataframe for the TLC dataset
* Examine data type of each column
* Gather descriptive statistics

### Your assignment

You will build a dataframe for the TLC data. After the dataframe is complete, you will organize the data for the process of exploratory data analysis, and update the team on your progress and insights.

## **Specific project deliverables**

With this end-of-course project, you will gain valuable practice and apply your new skills as you complete the following:

* Complete the questions in the Course 2 PACE strategy document
* Answer the questions in the Jupyter notebook project file
* Complete coding prep work on project’s Jupyter notebook
* Summarize the column Dtypes
* Communicate important findings to DeShawn and Luana in the form of an executive summary

Good luck with this project! Automatidata looks forward to seeing how you communicate your creative work and approach problem-solving!

# **Key takeaways**

The Google Advanced Data Analytics Certificate end-of-course project is designed for you to practice and apply course skills in a fictional workplace scenario. By completing each course’s end-of-course project, you will have work examples that will enhance your portfolio and showcase your skills for future employers.

# Activity: Create your Course 2 Automatidata project

To pass this practice quiz, you must receive 100%, or 1 out of 1 point, by completing the following activity. You can learn more about the graded and practice items in the [course overview](https://www.coursera.org/learn/get-started-with-python/supplement/UD00Z/course-2-overview).



## Activity Overview



In this activity, you will complete a project that showcases your ability to use Python to import, inspect, and organize data. You will also update team members through an executive summary, demonstrating your ability to organize and communicate key information.

For additional information on how to complete this activity, review the previous readings: [End-of-course portfolio project introduction](https://www.coursera.org/learn/foundations-of-data-science/supplement/9Opfe/end-of-course-portfolio-project-introduction) and [Course 2 end-of-course portfolio project overview: Automatidata](https://www.coursera.org/learn/get-started-with-python/supplement/D0f72/course-2-end-of-course-portfolio-project-overview-automatidata).

Be sure to complete this activity before moving on. The next course item will provide you with completed exemplars to compare to your own work. You will not be able to access the exemplars until you have completed this activity.

## Scenario



You are the newest member of Automatidata’s data analytics team. Your team is still in the early stages of their project for the New York City Taxi & Limousine Commission (TLC).

Previously, you were asked to complete a project proposal by your supervisor, Deshawn Washington. You have received notice that your project proposal has been approved and that TLC has given the Automatidata team access to their data for research purposes. Congratulations! To get clear insights, TLC's data must be analyzed, key variables identified, and the dataset ensured it is ready for analysis.

You discover two new emails in your inbox: one from your supervisor, Deshawn Washington, and one from your teammate, Luana Rodriguez. Review the emails, then follow the provided instructions to complete the PACE strategy document, the code notebook, and the executive summary.

***Note:*** Team member names used in this workplace scenario are fictional and are not representative of the New York City TLC.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Email from Deshawn Washington, Data Analysis Manager**

**Subject:** Help with coding notebook?

**From:** “Deshawn Washington,” Deshawn@automatidata.com

**Cc:** “Luana Rodriquez” Luana@automatidata.com

Good morning team,

I have a couple of updates on the TLC project. The project proposal that you completed previously has been approved. Thanks for all of your great work so far.  Additionally, I just received an email from our Senior Project Manager, Uli King, that TLC has given our team access to their data.

Before we begin the process of Exploratory Data Analysis (EDA), we could really use your help with coding and prepping the data. During your interview you mentioned that you worked with Python during the Google Career Certificate program. That experience sounds applicable here.

Luana (Cc’d) started a Jupyter notebook with the relevant dataset from TLC (attached). She is busy in the final stages of another project currently. I’m sure she could use your assistance in completing the coding and setting up the notebook for the TLC project.

Luana, do you mind sharing the details?

Humblest regards,

Deshawn Washington

Data Analysis Manager

Automatidata

**Email from Luana Rodriquez, Senior Data Analyst**

**Subject:** RE:Help with coding notebook?

**From:**  “Luana Rodriquez” Luana@automatidata

**Cc:** “Deshawn Washington,” Deshawn@automatidata

Nice to meet you (virtually)!

Hope you have enjoyed your first few weeks!

The project proposal you helped prepare covered the major points of this project, so I’ll get right to how you can assist the team. There are a number of us making adjustments to the machine learning developed for the last client, so your help is greatly appreciated!

Until we finish the prior project, there is no need to do a full EDA on this data. We will get to that soon. Do you mind reviewing the TLC data we received for the team? It would be fantastic if you could include a summary of the column Dtypes, data value nonnull counts, relevant and irrelevant columns, along with anything else code related you think is worth sharing in the notebook? It would be really helpful if you can create meaningful variables by combining or modifying the structures given.

Thanks,

Luana Rodriquez

Senior Data Analyst

Automatidata

## Step-By-Step Instructions



Follow the instructions to complete the activity. Then, go to the next course item to compare your work to a completed exemplar.

### **Step 1: Access the templates**



To use the templates for this course item, click each link of the following links and select Use Template. Links to templates:

* [Course 2 PACE strategy document](https://docs.google.com/document/d/1JM7h5MAQkD9uxoUhgpVBeT5R15y85LRaMo30I7bIFdk/template/preview)
* [Executive summary templates](https://docs.google.com/presentation/d/1Pps5GKxi1V31y2oRHRzU-xhJubkEYzCgEIfNjlEY3Og/template/preview#slide=id.g1e3a6309cc6_3_311)

OR

If you don’t have a Google account, you can download the template directly from the following attachments.

[Activity Template\_ Course 2 PACE strategy document](https://d3c33hcgiwev3.cloudfront.net/HOFJkwQOQDaAoqAePLgCKQ_1040f4b9e2b84bf6b9e56a62b424e0f1_Activity-Template_-Course-2-PACE-strategy-document.docx?Expires=1720137600&Signature=DXqUIvRX4RIBF7L9XwgkXkL~UBewtVzvXOPY~BZYSnDUpltgDFVymzY66vb~5hTuIrZUOvvYimOS81ElFNuAT3lO247atFiBMfHtFzv2cpYsOlCgo2z2yY5X~eW92V23PW1BHETSnYnMYJVv94AHzlk5uuzlylpjtdnH5q0nI4k_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

[DOCX File](https://d3c33hcgiwev3.cloudfront.net/HOFJkwQOQDaAoqAePLgCKQ_1040f4b9e2b84bf6b9e56a62b424e0f1_Activity-Template_-Course-2-PACE-strategy-document.docx?Expires=1720137600&Signature=DXqUIvRX4RIBF7L9XwgkXkL~UBewtVzvXOPY~BZYSnDUpltgDFVymzY66vb~5hTuIrZUOvvYimOS81ElFNuAT3lO247atFiBMfHtFzv2cpYsOlCgo2z2yY5X~eW92V23PW1BHETSnYnMYJVv94AHzlk5uuzlylpjtdnH5q0nI4k_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

[Activity Templates\_ Executive summaries](https://d3c33hcgiwev3.cloudfront.net/GUpAEib9S5GCZOJ0tvhVGA_4439ae7a596a40b5bd57ac42fe8b05f1_Activity-Templates_-Executive-summaries.pptx?Expires=1720137600&Signature=dl74ev5TFldvJmrLK5PFVSPJpmE0WRyCS3v~Ilrnt5WdnU1h4MuRXcuOjmz1K~vATrJ7r52RREOwEdeERwbaYPvyIHy9VhsTTIrMFX472dHEwgdc~MPVA0hRlDZebcIy25~hLwduu-KrwEbDL01cDv4Imka7B8r9vjv6gBpy2CM_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

[PPTX File](https://d3c33hcgiwev3.cloudfront.net/GUpAEib9S5GCZOJ0tvhVGA_4439ae7a596a40b5bd57ac42fe8b05f1_Activity-Templates_-Executive-summaries.pptx?Expires=1720137600&Signature=dl74ev5TFldvJmrLK5PFVSPJpmE0WRyCS3v~Ilrnt5WdnU1h4MuRXcuOjmz1K~vATrJ7r52RREOwEdeERwbaYPvyIHy9VhsTTIrMFX472dHEwgdc~MPVA0hRlDZebcIy25~hLwduu-KrwEbDL01cDv4Imka7B8r9vjv6gBpy2CM_&Key-Pair-Id=APKAJLTNE6QMUY6HBC5A" \t "_blank)

| **Column name** | **Description** |
| --- | --- |
| ID | Trip identification number |
| VendorID | A code indicating the TPEP provider that provided the record.  **1= Creative Mobile Technologies, LLC;**  **2= VeriFone Inc.** |
| tpep\_pickup\_datetime | The date and time when the meter was engaged. |
| tpep\_dropoff\_datetime | The date and time when the meter was disengaged. |
| Passenger\_count | The number of passengers in the vehicle.  This is a driver-entered value. |
| Trip\_distance | The elapsed trip distance in miles reported by the taximeter. |
| PULocationID | TLC Taxi Zone in which the taximeter was engaged |
| DOLocationID | TLC Taxi Zone in which the taximeter was disengaged |
| RateCodeID | The final rate code in effect at the end of the trip.  **1= Standard rate**  **2=JFK**  **3=Newark**  **4=Nassau or Westchester**  **5=Negotiated fare**  **6=Group ride** |
| Store\_and\_fwd\_flag | This flag indicates whether the trip record was held in vehicle memory before being  sent to the vendor, aka “store and forward,”  because the vehicle did  not have a connection to the server.  **Y= store and forward trip**  **N= not a store and forward trip** |
| Payment\_type | A numeric code signifying how the passenger paid for the trip.  **1= Credit card**  **2= Cash**  **3= No charge**  **4= Dispute**  **5= Unknown**  **6= Voided trip** |
| Fare\_amount | The time-and-distance fare calculated by the meter. |
| Extra | Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour  and overnight charges. |
| MTA\_tax | $0.50 MTA tax that is automatically triggered based on the metered rate in use. |
| Improvement\_surcharge | $0.30 improvement surcharge assessed trips at the flag drop. The  improvement surcharge  began being levied in 2015. |
| Tip\_amount | Tip amount –  This field is automatically populated for credit card tips. Cash tips are not included. |
| Tolls\_amount | Total amount of all tolls paid in trip. |
| Total\_amount | The total amount charged to passengers. Does not include cash tips. |



***Note***: The following lab is also the next course item. Once you complete and submit your end-of-course project activity, return to the lab instructions’ page and click ***Next*** to continue on to the exemplar reading.

To access the end-of-course project lab, click the following link and select Open Lab.

* [Course 2 Automatidata project lab](https://www.coursera.org/learn/get-started-with-python/ungradedLab/FrE1A/activity-course-2-automatidata-project-lab)

Your Python notebook for this project includes a guided framework that will assist you with the required coding. Input the code and answer the questions in your Python notebook to inspect and organize your data. You’ll find helpful reminders for tasks like:

* Importing data
* Loading necessary packages
* Identifying relevant data structures and summarizing data
* Extracting information from columns
* Combining or modifying data structures to create meaningful variables

You will also discover questions in this Python notebook designed to help you gather the relevant information you’ll need to write an executive summary for your team.

Use your completed PACE strategy document and Python notebook to help you prepare your executive summary.

### **Data Dictionary**



This project uses a dataset called [2017\_Yellow\_Taxi\_Trip\_Data.csv](https://drive.google.com/drive/u/1/folders/1epK0e7weuin_qUwfwDA2DhrN1JcQt5na). It contains data gathered by the New York City Taxi & Limousine Commission. For each trip, there are many different data variables gathered.

The dataset contains:

**408,294 rows** – each row represents a different trip

**18 columns**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### **Step 3: Complete your PACE strategy document**



The **Course 2 PACE strategy document** includes questions that will help guide you through the Course 2 Automatidata workplace scenario project. Answer the questions in your PACE strategy document to prepare for using Python to inspect and organize your data.

As a reminder, the PACE strategy document is designed to help you complete the contents for each of the templates provided. You might navigate back and forth between the PACE strategy document and the Python notebook. Make sure your PACE strategy document is complete before preparing your executive summary.

### **Step 4: Prepare an executive summary**



Your executive summary will keep your teammates at Automatidata informed of your progress. The one-page format is designed to respect teammates and stakeholders who may not have time to read and understand an entire report.

First, select one of the executive summary design layouts from the provided template. Then, add the relevant information. Your executive summary should include the following:

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

Complete your executive summary to effectively communicate your results to your teammates.

## Pro Tip: Save the templates

Finally, be sure to save a blank copy of the templates you used to complete this activity. You can use them for further practice or in your professional projects. These templates will help you work through your thought processes and demonstrate your experience to potential employers.

## What to Include in Your Response



Later, you will have the opportunity to self assess your performance using the criteria listed below. Be sure to address the following elements in your completed activity.

**Course 2 PACE strategy document:**

* Answer the questions in the PACE strategy document

**Course 2 Automatidata project lab:**

* Import, inspect, and organize data

**Course 2 executive summary:**

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

### 1.

Question 1

## Did you complete this activity?

1 / 1 point

Yes

## Assessment of Exemplar



### Course 2 Automatidata project lab

Compare the exemplar to the Python notebook you completed. Your responses might differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the Python notebook. Yours might differ in certain ways, such as your specific code input or responses to questions. What's important is that you have an overall understanding of the purpose and functionality of a Python notebook for data analysis.

Your Python notebook should:

* Include the correct code for inspecting and organizing your data
* Clearly communicate your responses to questions about code input and results



### Course 2 executive summary

Compare the exemplar to your completed executive summary. Your responses might differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the executive summary. Yours might differ in certain ways, such as your specific language, answers to questions or the layout you selected from the template offerings. What’s important is that you have an overall understanding of the purpose and organization of executive summaries for data projects.

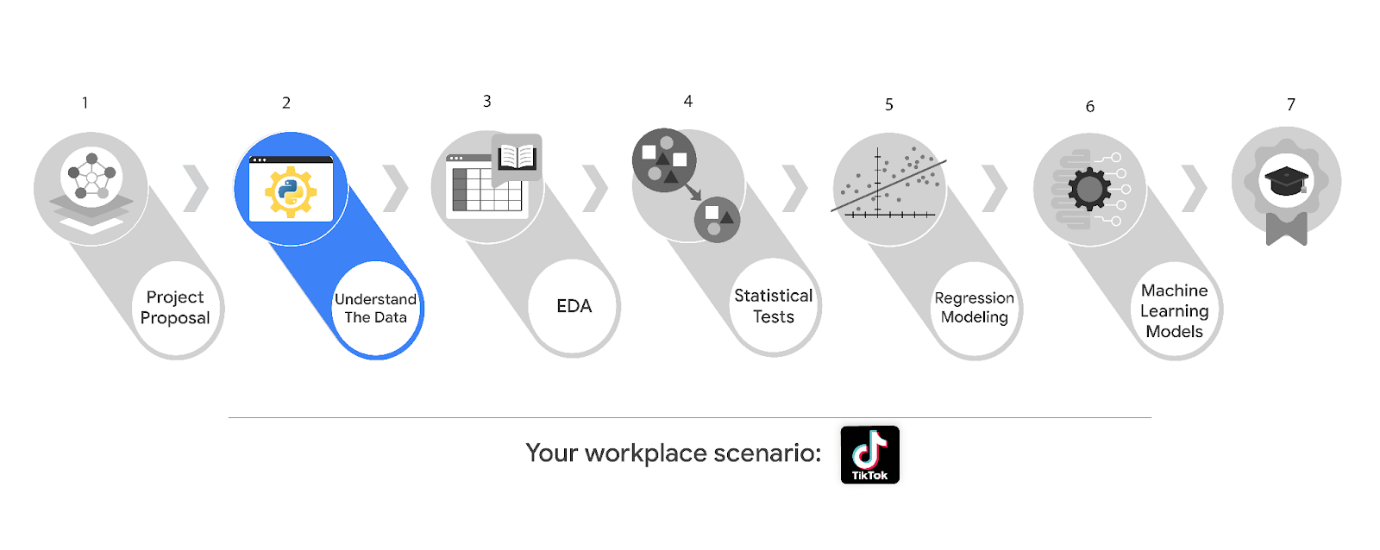
Your executive summary should:

* Include key information that you want to share with teammates and/or stakeholders
* Use clear and concise language to effectively communicate your results

# Course 2 end-of-course portfolio project overview: TikTok

# **Learn about the Course 2 TikTok workplace scenario!**

The end-of-course project in Course 2 focuses on your ability to understand the data needed for a project. As a reminder, in Course 1 you developed a project proposal that outlined milestones, which progress with each of the end-of-course projects. A visual representation is provided in the graphic shown here:



Learn more about the project, your role, and expectations in this reading.

## **Background on the TikTok scenario**

At TikTok, our mission is to inspire creativity and bring joy. Our employees lead with curiosity and move at the speed of culture. Combined with our company's flat structure, you'll be given dynamic opportunities to make a real impact on a rapidly expanding company and grow your career.

TikTok users have the ability to submit reports that identify videos and comments that contain user claims. These reports identify content that needs to be reviewed by moderators. The process generates a large number of user reports that are challenging to consider in a timely manner.

TikTok is working on the development of a predictive model that can determine whether a video contains a claim or offers an opinion. With a successful prediction model, TikTok can reduce the backlog of user reports and prioritize them more efficiently.

### **Project background**

TikTok’s data team is in the earliest stages of the claims classification project. The following tasks are needed before the team can begin the data analysis process:

* Build a dataframe for the TikTok dataset
* Examine data type of each column
* Gather descriptive statistics

### Your assignment

You will build a dataframe for the claims classification data. After the dataframe is complete, you will organize the claims data for the process of exploratory data analysis, and update the team on your progress and insights.

## **Team members at TikTok**

### **Data team roles**

* Willow Jaffey- Data Science Lead
* Rosie Mae Bradshaw- Data Science Manager
* Orion Rainier- Data Scientist

The members of the data team at TikTok are well versed in data analysis and data science. Messages to these more technical coworkers should be concise and specific.

### **Cross-functional team members**

* Mary Joanna Rodgers- Project Management Officer
* Margery Adebowale- Finance Lead, Americas
* Maika Abadi- Operations Lead

Your TikTok team includes several managers, who oversee operations. It is important to adjust your general correspondence appropriately to their roles, given that their responsibilities are less technical in nature.

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

## **Specific project deliverables**

With this end-of-course project, you will gain valuable practice and apply your new skills as you complete the following:

* Course 2 PACE Strategy Document to plan your project while considering your audience members, teammates, key milestones, and overall project goal.
* Answer the questions in the Jupyter notebook project file
* Complete coding prep work on project’s Jupyter notebook
* Summarize the column Dtypes
* Communicate important findings in the form of an executive summary

TikTok's data team needs you to problem-solve and communicate your findings. Good luck on your tasks!

## **Key takeaways**

The Google Advanced Data Analytics Certificate end-of-course project is designed for you to practice and apply course skills in a fictional workplace scenario. By completing each course’s end-of-course project, you will have work examples that will enhance your portfolio and showcase your skills for future employers.

# Activity: Create your Course 2 TikTok project

To pass this practice quiz, you must receive 100%, or 1 out of 1 point, by completing the activity below. You can learn more about the graded and practice items in the [course overview](https://www.coursera.org/learn/get-started-with-python/supplement/UD00Z/course-2-overview).



## Activity Overview



In this activity, you will complete a project that showcases your ability to use Python to import, inspect, and organize data. You will also update team members through an executive summary, demonstrating your ability to organize and communicate key information.

For additional information on how to complete this activity, review the previous readings: [End-of-course project introduction](https://www.coursera.org/learn/foundations-of-data-science/supplement/9Opfe/end-of-course-portfolio-project-introduction) and [Course 2 end-of-course portfolio project overview: TikTok](https://www.coursera.org/learn/get-started-with-python/supplement/50gsf/course-2-end-of-course-portfolio-project-overview-tiktok).

Be sure to complete this activity before moving on. The next course item will provide you with completed exemplars to compare to your own work. You will not be able to access the exemplars until you have completed this activity.

## Scenario



The team’s latest project is in its early stages of developing a machine learning model to classify claims in videos.

Previously, you were asked to complete a project proposal by your supervisor, Rosie Mae Bradshaw. You have received notice that the project proposal submitted by the team has been approved and your team has been given access to TikTok’s user data. To get clear insights, the data must be inspected, organized, and prepared for analysis.

You discover two new emails in your inbox: one from your supervisor, Rosie Mae Bradshaw, and one from Willow Jaffey, the data team’s Data Science Lead. Review the emails, then follow the provided instructions to complete the PACE strategy document, the code notebook, and the executive summary.

***Note:*** Team member names used in this workplace scenario are fictional and are not representative of TikTok.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Email from Rosie Mae Bradshaw, Data Science Manager**

**Subject:** Help with coding notebook?

**From:** “Bradshaw, Rosie Mae” —rosiemaebradshaw@tiktok

**Cc:** “Rainier, Orion”—orionrainier@tiktok

Good morning,

I have a couple of updates on our latest project. The leadership team has approved the project proposal that we completed previously. Thanks for all of your great work so far. Additionally, I just received an email from our Project Management Officer, Mary Joanna Rodgers that the data team is clear to proceed.

Before we begin the process of Exploratory Data Analysis (EDA), we could really use your help with coding and prepping the data. During your interview you mentioned that you worked with Python specifically in the Google certificate program you completed. That experience sounds applicable here.

Orion Rainier (Cc’d above) started a Jupyter notebook with the relevant dataset (attached). Orion is very involved in the final stages of another project. I’m sure your assistance in completing the coding and setting up the notebook for the project would be greatly appreciated.

Orion, do you mind sharing the details?

Humblest regards,

Rosie Mae Bradshaw

Data Science Manager

TikTok

[Learn about TikTok’s Trust & Safety team](https://newsroom.tiktok.com/en-us/safety)

**Email from Orion Rainier, Data Scientist**

**Subject:** RE:Help with coding notebook?

**From:** “Rainier, Orion”—orionrainier@tiktok

**Cc:** “Bradshaw, Rosie Mae” —rosiemaebradshaw@tiktok

Nice to meet you (virtually)!

Hope you have enjoyed your first few weeks!

With the project proposal approved, we are ready to begin the process of preparing the claim classification data. The goal of this project is to ultimately build a machine learning model that can streamline the claims process by identifying whether statements made in videos are claims or opinions.

A claim refers to information that is either unsourced or from an unverified source. For example, “The news reported that someone revealed that around 50% of the mined gold on Earth comes from one source.”

Opinions refer to the personal beliefs or thoughts of a group or an individual. Here’s an example, “In my opinion the most productive work day of the week is Tuesday.”

There are a number of data team members committed to adjusting the machine learning developed for the last project, so your help is greatly appreciated!

Until we finish the prior project, there is no need to do a full EDA on this data. We will get to that soon. Do you mind importing the data (attached) and reviewing it for the team? It would be fantastic if you could include a summary of the column Data types, data value nonnull counts, relevant and irrelevant columns, along with anything else code related you think is worth sharing/showing in the notebook? You’ll need to select a couple of variables to focus on. Include their minimum and maximum values. I haven’t looked closely at the data yet, but it would be really helpful if you can create meaningful variables by combining or modifying the structures given.

Thanks,

Orion Rainier

Data Scientist

TikTok

–

“Big data isn’t about bits, it’s about talent.” — Douglas Merrill

## Step-By-Step Instructions



Follow the instructions to complete the activity. Then, go to the next course item to compare your work to a completed exemplar.

## Step 1: Access the templates



To use the templates for this course item, click the following links and select Use Template.

Links to templates:



***Note***: The following lab is also the next course item. Once you complete and submit your end-of-course project activity, return to the lab instructions’ page and click ***Next*** to continue on to the exemplar reading.

To access the end-of-course project lab, click the following link and select Open Lab.

* [Course 2 TikTok project lab](https://www.coursera.org/learn/get-started-with-python/ungradedLab/emRkC/activity-course-2-tiktok-project-lab)

Your Python notebook for this project includes a guided framework that will assist you with the required coding. Input the code and answer the questions in your Python notebook to inspect and organize your data. You’ll find helpful reminders for tasks like:

* Importing data
* Loading necessary packages
* Identifying relevant data structures and summarizing data
* Extracting information from columns
* Combining or modifying data structures to create meaningful variables

You will also discover questions in this Python notebook designed to help you gather the relevant information you’ll need to write an executive summary for your team.

Use your completed PACE strategy document and Python notebook to help you prepare your executive summary in the next step.

### **Data Dictionary**



This project uses a dataset called tiktok\_dataset.csv. It contains synthetic data created for this project in partnership with TikTok. Examine each data variable gathered.

**19,383 rows** – Each row represents a different published TikTok video in which a claim/opinion has been made.

**12 columns**

| **Column name** | **Type** | **Description** |
| --- | --- | --- |
| # | int | TikTok assigned number for video with claim/opinion. |
| claim\_status | obj | Whether the published video has been identified as an “opinion” or a “claim.”  In this dataset, an “opinion” refers to an individual’s or group’s personal belief or thought.  A “claim” refers to information that is either unsourced or from an unverified source. |
| video\_id | int | Random identifying number assigned to video upon publication on TikTok. |
| video\_duration\_sec | int | How long the published video is measured in seconds. |
| video\_transcription\_text | obj | Transcribed text of the words spoken in the published video. |
| verified\_status | obj | Indicates the status of the TikTok user who published the video in terms of their verification,  either “verified” or “not verified.” |
| author\_ban\_status | obj | Indicates the status of the TikTok user who published the video in terms of their permissions: “active,” “under scrutiny,” or “banned.” |
| video\_view\_count | float | The total number of times the published video has been viewed. |
| video\_like\_count | float | The total number of times the published video has been liked by other users. |
| video\_share\_count | float | The total number of times the published video has been shared by other users. |
| video\_download\_count | float | The total number of times the published video has been downloaded by other users. |
| video\_comment\_count | float | The total number of comments on the published video. |

### **Step 3: Complete your PACE strategy document**



The **Course 2 PACE strategy document** includes questions that will help guide you through the Course 2 TikTok workplace scenario project. Answer the questions in your PACE strategy document to prepare to use Python to inspect and organize your data.

As a reminder, the PACE strategy document is designed to help you complete the contents for each of the templates provided. You may navigate back and forth between the PACE strategy document and the Python notebook. Make sure your PACE strategy document is complete before preparing your executive summary.

### **Step 4: Prepare an executive summary**



Your executive summary will keep your teammates at TikTok informed of your progress. The one-page format is designed to respect teammates and stakeholders who may not have time to read and understand an entire report.

First, select one of the executive summary design layouts from the provided template. Then, add the relevant information. Your executive summary should include the following:

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

Complete your executive summary to effectively communicate your results to your teammates.

## Pro Tip: Save the templates

Finally, be sure to save a blank copy of the templates you used to complete this activity. You can use them for further practice or in your professional projects. These templates will help you work through your thought processes and demonstrate your experience to potential employers.

## What to Include in Your Response



Later, you will have the opportunity to self assess your performance using the criteria listed below. Be sure to address the following elements in your completed activity.

**Course 2 PACE strategy document**:

* Answer the questions in the PACE strategy document

**Course 2 TikTok project lab**:

* Import, inspect, and organize data

**Course 2 executive summary**:

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

### 1.

Question 1

## Did you complete this activity?

1 / 1 point

Yes

No

Correct

Thank you for completing this activity! Using Python to inspect and organize your data is a key step in any data analysis project. Further, effectively communicating your findings with an executive summary is an important skill for any data professional. Go to the next course item to compare your work to completed exemplars.

In this lab portion of the end-of-course project, you will open a Jupyter Notebook and follow instructions to enter code and written responses where prompted.

# Data dictionary

This project uses a dataset called **tiktok\_dataset.csv**. It contains synthetic data created for this project in partnership with TikTok.

The dataset contains:

**19,383 rows** – Each row represents a different published TikTok video in which a claim/opinion has been made.

**12 columns**

| **Column name** | **Type** | **Description** |
| --- | --- | --- |
| # | int | TikTok assigned number for video with claim/opinion. |
| claim\_status | obj | Whether the published video has been identified as an “opinion” or a “claim.” In this dataset, an “opinion” refers to an individual’s or group’s personal belief or thought. A “claim” refers to information that is either unsourced or from an unverified source. |
| video\_id | int | Random identifying number assigned to video upon publication on TikTok. |
| video\_duration\_sec | int | How long the published video is measured in seconds. |
| video\_transcription\_text | obj | Transcribed text of the words spoken in the published video. |
| verified\_status | obj | Indicates the status of the TikTok user who published the video in terms of their verification, either “verified” or “not verified.” |
| author\_ban\_status | obj | Indicates the status of the TikTok user who published the video in terms of their permissions: “active,” “under scrutiny,” or “banned.” |
| video\_view\_count | float | The total number of times the published video has been viewed. |
| video\_like\_count | float | The total number of times the published video has been liked by other users. |
| video\_share\_count | float | The total number of times the published video has been shared by other users. |
| video\_download\_count | float | The total number of times the published video has been downloaded by other users. |
| video\_comment\_count | float | The total number of comments on the published video. |

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# **Access the end-of-course project lab**

***Note***: Click the ***Open Lab*** button to start your end-of-course project lab. Once you complete this activity, click ***Next*** to continue on to the exemplar reading.

**(Lab) Activity Instructions:**

The Jupyter Notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

As you complete the end-of-course project lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.

To review how to work in Jupyter Notebooks, refer to the reading [Practice Python skills in Jupyter Notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/revised-content-only-practice-python-skills-in-jupyter-notebooks).

Be sure to complete this lab before moving on. The next course item will explain how to review an exemplar of a completed end-of-course project lab. You can compare the code and text responses in the exemplar to your own.



# Activity Exemplar: Create your Course 2 TikTok project

Here are completed exemplars along with an explanation of how the exemplars fulfill the expectations for the activity.

## Completed Exemplars



To review the exemplar for the Course 2 executive summary, click the following link and select Use Template.

* [Course 2 executive summary](https://docs.google.com/presentation/d/1Vwo9pPwFLOFEu69x0qLEv_StHMSofU5E59AQkW0c_f8/template/preview?resourcekey=0-cEC-fVSgozZqLR-IS0hPZg)

OR

If you do not have a Google account, you can download the exemplar directly from the following attachment.

***Note***: The following lab is also the next course item.

To access the exemplar for the end-of-course project lab, click the following link and select Open Lab.

* [Course 2 TikTok project lab](https://www.coursera.org/learn/get-started-with-python/ungradedLab/MEiVL/exemplar-course-2-tiktok-project-lab)

## Assessment of Exemplars



### Course 2 TikTok project lab

Compare the exemplar to the Python notebook you completed. Your responses may differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the Python notebook. Yours may differ in certain ways, such as your specific code input or responses to questions. What's important is that you have an overall understanding of the purpose and functionality of a Python notebook for data analysis.

Your Python notebook should:

* Include the correct code for inspecting and organizing your data
* Clearly communicate your responses to questions about code input and results



### Course 2 executive summary

Compare the exemplar to your completed executive summary. Your responses may differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the executive summary. Yours might differ in certain ways, such as your specific language, answers to questions or the layout you selected from the template offerings. What’s important is that you have an overall understanding of the purpose and organization of executive summaries for data projects.

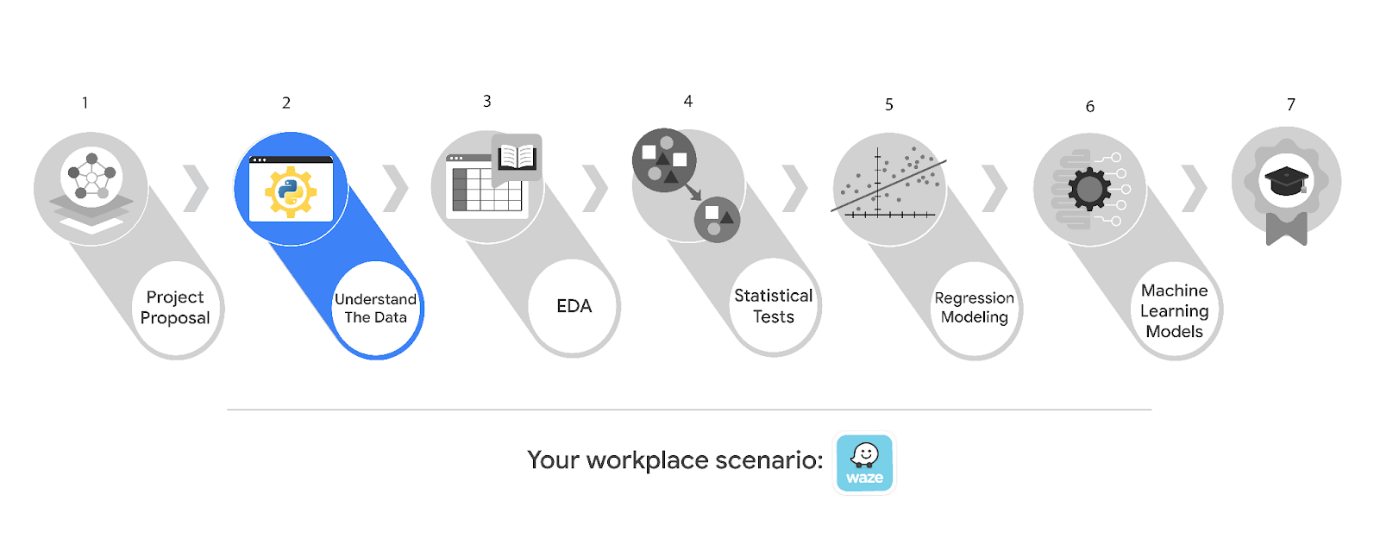
Your executive summary should:

* Include key information that you want to share with teammates and/or stakeholders
* Use clear and concise language to effectively communicate your results

# Course 2 end-of-course portfolio project overview: Waze

# **Learn about the Course 2 Waze workplace scenario!**

The end-of-course project in Course 2 focuses on your ability to understand the data needed for a project. As a reminder, in Course 1 you developed a project proposal that outlined milestones, which progress with each of the end-of-course projects. A visual representation is provided in the graphic shown here:



Learn more about the project, your role, and expectations in this reading.

## **Background on the Waze scenario**

Waze’s free navigation app makes it easier for drivers around the world to get to where they want to go. Waze’s community of map editors, beta testers, translators, partners, and users helps make each drive better and safer. Waze partners with cities, transportation authorities, broadcasters, businesses, and first responders to help as many people as possible travel more efficiently and safely.

You’ll collaborate with your Waze teammates to analyze and interpret data, generate valuable insights, and help leadership make informed business decisions. Your team is about to start a new project to help prevent user churn on the Waze app. Churn quantifies the number of users who have uninstalled the Waze app or stopped using the app. This project focuses on monthly user churn.

This project is part of a larger effort at Waze to increase growth. Typically, high retention rates indicate satisfied users who repeatedly use the Waze app over time. Developing a churn prediction model will help prevent churn, improve user retention, and grow Waze’s business. An accurate model can also help identify specific factors that contribute to churn and answer questions such as:

* Who are the users most likely to churn?
* Why do users churn?
* When do users churn?

For example, if Waze can identify a segment of users who are at high risk of churning, Waze can proactively engage these users with special offers to try and retain them. Otherwise, Waze may lose these users without knowing why.

Your insights will help Waze leadership optimize the company’s retention strategy, enhance user experience, and make data-driven decisions about product development.

### **Project background**

Waze’s data team is in the earliest stages of the churn project. The following tasks are needed before the team can begin the data analysis process:

* Build a dataframe for the churn dataset
* Examine data type of each column
* Gather descriptive statistics
* Your assignment

### **Your assignment**

You will build a dataframe for the churn data. After the dataframe is complete, you will organize the data for the process of exploratory data analysis, and update the team on your progress and insights.

## **Team members at Waze**

### **Data team roles**

* Harriet Hadzic - Director of Data Analysis
* May Santner - Data Analysis Manager
* Chidi Ga - Senior Data Analyst
* Sylvester Esperanza - Senior Project Manager

Data team members have technical experience with data analysis and data science. However, you should always be sure to keep summaries and messages to these team members concise and to the point.

### **Cross-functional team members**

* Emrick Larson - Finance and Administration Department Head
* Ursula Sayo - Operations Manager

Your Waze team includes several managers overseeing operations. It is important to adapt your communication to their roles since their responsibilities are less technical.

***Note:*** The story, all names, characters, and incidents portrayed in this project are fictitious. No identification with actual persons (living or deceased) is intended or should be inferred. And, the data shared in this project has been created for pedagogical purposes.

## **Specific project deliverables**

With this end-of-course project, you will gain valuable practice and apply your new skills as you complete the following:

* Complete the questions in the Course 2 PACE strategy document
* Answer the questions in the Jupyter notebook project file
* Complete coding prep work on project’s Jupyter notebook
* Summarize the column Dtypes
* Communicate important findings in the form of an executive summary

Good luck with this project! Your Waze team members are looking forward to seeing how you communicate your creative work and approach problem-solving!

## **Key takeaways**

The Google Advanced Data Analytics Certificate end-of-course project is designed for you to practice and apply course skills in a fictional workplace scenario. By completing each course’s end-of-course project, you will have work examples that will enhance your portfolio and showcase your skills for future employers.

# Activity: Create your Course 2 Waze project

To pass this practice quiz, you must receive 100%, or 1 out of 1 point, by completing the activity below. You can learn more about the graded and practice items in the [course overview](https://www.coursera.org/learn/get-started-with-python/supplement/UD00Z/course-2-overview).



## Activity Overview



In this activity, you will complete a project that showcases your ability to use Python to import, inspect, and organize data. You will also update team members through an executive summary, demonstrating your ability to organize and communicate key information.

For additional information on how to complete this activity, review the previous readings: [End-of-course project introduction](https://www.coursera.org/learn/foundations-of-data-science/supplement/9Opfe/end-of-course-portfolio-project-introduction) and [Course 2 end-of-course portfolio project overview: Waze](https://www.coursera.org/learn/get-started-with-python/supplement/SqRqL/course-2-end-of-course-portfolio-project-overview-waze).

Be sure to complete this activity before moving on. The next course item will provide you with completed exemplars to compare to your own work. You will not be able to access the exemplars until you have completed this activity.

## Scenario



Your team is still in the early stages of their project to develop a machine learning model to predict user churn.

Previously, you were asked to complete a project proposal by your supervisor, May Santner. Now, you have received notice that your project proposal has been approved and your team has been given access to Waze’s user data. To get clear insights, the data must be inspected, organized, and prepared for analysis.

You discover two new emails in your inbox: one from May Santner, and one from your teammate, Chidi Ga. In the email, May asks for your help reviewing the data and completing a code notebook, and Chidi shares the details of the notebook. Review the emails, then follow the provided instructions to complete the PACE strategy document, the code notebook, and the executive summary.

***Note:*** Team member names used in this workplace scenario are fictional and are not representative of Waze.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Email from May Santner, Data Analysis Manager**

**Subject:** Help with coding notebook?

**From:** “May Santner,” May@waze

**Cc:** “Chidi Ga,” Chidi@waze

Good morning!

I have a couple of updates on the user churn project. First off, the project proposal you completed has been approved. Thanks for all your great work so far. Second, I just received an email from our Senior Project Manager, Sylvester Esperanza, that our team has been given access to the Waze user data.

Before we begin the process of exploratory data analysis (EDA), we could really use your help with coding and prepping the data. During your interview, you mentioned that you worked with Python in your Google certificate program. You can draw on your Python skills for this task.

Chidi Ga (cc’d above) started a Jupyter notebook with the relevant dataset (imported). Right now, Chidi is busy finishing up a previous project. I’m sure he could use your assistance in completing the coding and setting up the notebook for the user churn project.

Chidi, do you mind sharing the details?

Best,

May Santner

Data Analysis Manager

Waze

- –  - – - - - - - — - - - - - - - - - - - - - - - - – - - - - - - - - -  - -

**Email from Chidi Ga, Senior Data Analyst**

**Subject:** RE:Help with coding notebook?

**From:** “Chidi Ga,” Chidi@waze

**Cc:** “May Santner,” May@waze

Nice to meet you (virtually)!

Hope you’ve enjoyed your first few weeks at Waze!

The project proposal you helped prepare covered the major points of this project, so I’ll get right to how you can assist the team. Right now, a number of us are busy making adjustments to the machine learning model for another project, so your help is greatly appreciated!

Until we finish our previous project, there is no need to do a full EDA on our new user data. We’ll get to that soon. Meanwhile, do you mind reviewing the imported data for the team? It would be fantastic if you could include a summary of the data types for each variable, where missing values exist in the data, key descriptive statistics, and anything else code-related you think is worth sharing in the notebook. I haven’t had a chance to explore the data, so I really appreciate you getting an early start on this.

Thanks,

Chidi Ga

Senior Data Analyst

Waze

## Step-By-Step Instructions



Follow the instructions to complete the activity. Then, go to the next course item to compare your work to a completed exemplar.

### **Step 1: Access the templates**



To use the templates for this course item, click each link below and select Use Template.

Link to templates:

* [Course 2 PACE strategy document](https://docs.google.com/document/d/1JM7h5MAQkD9uxoUhgpVBeT5R15y85LRaMo30I7bIFdk/template/preview)
* [Executive summary templates](https://docs.google.com/presentation/d/1Pps5GKxi1V31y2oRHRzU-xhJubkEYzCgEIfNjlEY3Og/template/preview)

OR

If you do not have a Google account, you can download the templates directly from the following attachments:

Step 2: Access the end-of-course project lab

***Note***: The following lab is also the next course item. Once you complete and submit your end-of-course project activity, return to the lab instructions’ page and click ***Next*** to continue on to the exemplar reading.

To access the end-of-course project lab, click the following link and select Open Lab.

* [Course 2 Waze project lab](https://www.coursera.org/learn/get-started-with-python/ungradedLab/RTLD1/activity-course-2-waze-project-lab)

Your Python notebook for this project includes a guided framework that will assist you with the required coding. Input the code and answer the questions in your Python notebook to inspect and organize your data. You’ll find helpful reminders for tasks like:

* Importing data
* Loading necessary packages
* Identifying relevant data structures and summarizing data
* Extracting information from columns
* Combining or modifying data structures to create meaningful variables

You will also discover questions in this Python notebook designed to help you gather the relevant information you’ll need to write an executive summary for your team.

Use your completed PACE strategy document and Python notebook to help you prepare your executive summary in the next step.

### **Data Dictionary**



This project uses a dataset called waze\_dataset.csv. It contains synthetic data created for this project in partnership with Waze. Examine each data variable gathered.

The dataset contains:

**14,999 rows** – each row represents one unique user

**13 columns**

| **Column name** | **Type** | **Description** |
| --- | --- | --- |
| ID | int | A sequential numbered index |
| label | obj | Binary target variable (“retained” vs “churned”) for if a user has churned anytime during  the course of the month |
| sessions | int | The number of occurrence of a user opening the app during the month |
| drives | int | An occurrence of driving at least 1 km during the month |
| device | obj | The type of device a user starts a session with |
| total\_sessions | float | A model estimate of the total number of sessions since a user has onboarded |
| n\_days\_after\_onboarding | int | The number of days since a user signed up for the app |
| total\_navigations\_fav1 | int | Total navigations since onboarding to the user’s favorite place 1 |
| total\_navigations\_fav2 | int | Total navigations since onboarding to the user’s favorite place 2 |
| driven\_km\_drives | float | Total kilometers driven during the month |
| duration\_minutes\_drives | float | Total duration driven in minutes during the month |
| activity\_days | int | Number of days the user opens the app during the month |
| driving\_days | int | Number of days the user drives (at least 1 km) during the month |

### **Step 3: Complete your PACE strategy document**



The **Course 2 PACE strategy document** includes questions that will help guide you through the Course 2 Waze workplace scenario project. Answer the questions in your PACE strategy document to prepare to use Python to inspect and organize your data.

As a reminder, the PACE strategy document is designed to help you complete the contents for each of the templates provided. You may navigate back and forth between the PACE strategy document and the Python notebook. Make sure your PACE strategy document is complete before preparing your executive summary.

### **Step 4: Prepare an executive summary**



Your executive summary will keep your teammates at Waze informed of your progress. The one-page format is designed to respect teammates and stakeholders who may not have time to read and understand an entire report.

First, select one of the executive summary design layouts from the provided template. Then, add the relevant information. Your executive summary should include the following:

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

Complete your executive summary to effectively communicate your results to your teammates.

## Pro Tip: Save the templates

Finally, be sure to save a blank copy of the templates you used to complete this activity. You can use them for further practice or in your professional projects. These templates will help you work through your thought processes and demonstrate your experience to potential employers.

## What to Include in Your Response



Later, you will have the opportunity to self assess your performance using the criteria listed below. Be sure to address the following elements in your completed activity.

**Course 2 PACE strategy document**:

* Answer the questions in the PACE strategy document

**Course 2 Waze project lab**:

* Import, inspect, and organize data

**Course 2 executive summary**:

* A summary of your tasks
* Information regarding the results of your data variable assessment
* Identify recommended next steps in order to build a predictive model

### 1.

Question 1

## Did you complete this activity?

1 / 1 point

Yes

No

Correct

Thank you for completing this activity! Using Python to inspect and organize your data is a key step in any data analysis project. Further, effectively communicating your findings with an executive summary is an important skill for any data professional. Go to the next course item to compare your work to completed exemplars.

# Instructions

In this lab portion of the end-of-course project, you will open a Jupyter Notebook and follow instructions to enter code and written responses where prompted.

# Data dictionary

This project uses a dataset called **waze\_dataset.csv**. It contains synthetic data created for this project in partnership with Waze.

The dataset contains:

**14,999 rows** – each row represents one unique user

**13 columns**

| **Column name** | **Type** | **Description** |
| --- | --- | --- |
| ID | int | A sequential numbered index |
| label | obj | Binary target variable (“retained” vs “churned”) for if a user has churned anytime during the course of the month |
| sessions | int | The number of occurrence of a user opening the app during the month |
| drives | int | An occurrence of driving at least 1 km during the month |
| device | obj | The type of device a user starts a session with |
| total\_sessions | float | A model estimate of the total number of sessions since a user has onboarded |
| n\_days\_after\_onboarding | int | The number of days since a user signed up for the app |
| total\_navigations\_fav1 | int | Total navigations since onboarding to the user’s favorite place 1 |
| total\_navigations\_fav2 | int | Total navigations since onboarding to the user’s favorite place 2 |
| driven\_km\_drives | float | Total kilometers driven during the month |
| duration\_minutes\_drives | float | Total duration driven in minutes during the month |
| activity\_days | int | Number of days the user opens the app during the month |
| driving\_days | int | Number of days the user drives (at least 1 km) during the month |

Remember, you can access and download the data for any Jupyter notebook activity from within the notebook itself by navigating to the **Lab Files** dropdown menu at the top of the page, clicking into the **/home/jovyan/work** folder, selecting the relevant data file, and clicking **Download**.

# **Access the end-of-course project lab**

***Note***: Click the ***Open Lab*** button to start your end-of-course project lab. Once you complete this activity, click ***Next*** to continue on to the exemplar reading.

**(Lab) Activity Instructions:**

The Jupyter Notebook will autosave as you work, or you can manually save it by clicking the **Save and Checkpoint** button or by selecting **Save and Checkpoint** from the **File** menu.

A screenshot of a computer

Description automatically generated

As you complete the end-of-course project lab, note the following features:

* **Sections:** Step-by-step instructions in each section lead you through the lab.
* **Code blocks:** Code blocks allow you to practice key Python coding concepts. Add code where prompted and then click the **Run** button to execute your code and view any possible output.



* **Questions:** Thought questions offer moments to pause and think about concepts and your output as you move through the lab.

To review how to work in Jupyter Notebooks, refer to the reading [Practice Python skills in Jupyter Notebooks](https://www.coursera.org/learn/get-started-with-python/supplement/2poER/revised-content-only-practice-python-skills-in-jupyter-notebooks).

Be sure to complete this lab before moving on. The next course item will explain how to review an exemplar of a completed end-of-course project lab. You can compare the code and text responses in the exemplar to your own.



# Activity Exemplars: Create your Course 2 Waze project

Here are completed exemplars along with an explanation of how the exemplars fulfill the expectations for the activity.

## Completed Exemplars

To review the exemplar for the Course 2 executive summary, click the following link and select Use Template.



* [Course 2 Executive summary](https://docs.google.com/presentation/d/1OwW2mHUZ6vjWwOGIUv_Wr7msvTOHDJaepjKh3gjewjY/template/preview)

OR

If you do not have a Google account, you can download the exemplar directly from the following attachment.

***Note***: The following lab is also the next course item.

To access the exemplar for the end-of-course project lab, click the following link and select Open Lab.

* [Course 2 Waze project lab](https://www.coursera.org/learn/get-started-with-python/ungradedLab/IMizQ/exemplar-course-2-waze-project-lab)

## Assessment of Exemplar



### Course 2 Waze project lab

Compare the exemplar to the Python notebook you completed. Your responses might differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the Python notebook. Yours might differ in certain ways, such as your specific language, answers to questions or the layout you selected from the template offerings. What's important is that you have an overall understanding of the purpose and functionality of a Python notebook for data analysis.

Your Python notebook should:

* Include the correct code for inspecting and organizing your data
* Clearly communicate your responses to questions about code input and results



### Course 2 executive summary

Compare the exemplar to your completed executive summary. Your responses might differ from the exemplar, but that is to be expected. What did you do well? Where can you improve? Use your answers to these questions to guide you as you progress through the end-of-course projects in the certificate.

***Note:*** The exemplar represents one possible way to complete the executive summary. Yours might differ in certain ways, such as your specific language and or layout selected from the template offerings. What’s important is that you have an overall understanding of the purpose and organization of executive summaries for data projects.

Your executive summary should:

* Include key information that you want to share with teammates and/or stakeholders
* Use clear and concise language to effectively communicate your results

**What we covered**

-Importance of communication

-Data professional tools

-Python for dataset management

**Keep in mind**

-Who your audience is and what their goals are

-What they already know

-What they need to know

# Assess your Course 2 end-of-course project

This is the rubric for the Course 2 end-of-course project. The rubric was designed to be applicable to all of the project scenarios. You will use this rubric to review and grade your own work. The rubric grading process is an important part of the learning experience because it allows you to objectively assess your end-of-course project against a set of criteria.

There are a total of 9 points for the end-of-course project and 9 items in this rubric. Each rubric item is worth 1 point. The items are grouped by topic and correspond to each step you completed for the Course 2 end-of-course project.

To use the rubric, first open your workplace scenario notebook, executive summary, and PACE strategy document. Next, review each rubric item’s grading criteria. Then respond to each statement by marking “yes” or “no.”

When you complete and submit the rubric, you will receive a percentage score. This score will help you confirm whether you completed the required steps of the end-of-course project; the recommended passing grade for this project is 80% (or 8/9 points). If you want to increase your score, you can revise your project and then resubmit this rubric to reflect any changes you make. Try to achieve at least 8 points on this rubric before continuing on to the next course.

## Imports

The following rubric items assess the imports for your end-of-course project.

### 1.

Question 1

Applicable packages and libraries were imported to the code notebook.

1 / 1 point

Yes

No

Correct

## **Data Analysis**

The following rubric items assess the data analysis work you completed for your end-of-course project.

### 2.

Question 2

The head() function was used to analyze the data.

1 / 1 point

Yes

No

Correct

### 3.

Question 3

The info() function was used to analyze the data.

1 / 1 point

Yes

No

Correct

### 4.

Question 4

The describe() function was used to analyze the data.

1 / 1 point

Yes

No

Correct

## Results and/or Evaluation

The following rubric items assess the concluding steps of your end-of-course project, including evaluation and summary of findings.

### 5.

Question 5

All questions in the code notebook were answered.

1 / 1 point

Yes

No

Correct

### 6.

Question 6

All questions in the PACE strategy document were answered.

1 / 1 point

Yes

No

Correct

### 7.

Question 7

The executive summary mentioned the tasks completed for this end-of-course project.

1 / 1 point

Yes

No

Correct

### 8.

Question 8

The executive summary included information regarding the results of the data variable assessment.

1 / 1 point

Yes

No

Correct

### 9.

Question 9

The executive summary identified recommended next steps in order to build a predictive model.

1 / 1 point

Yes

No

Correct

ANWSER ALL Yes

In your second end-of-course project, you continued in your role as a data analyst. Within each of the available scenarios, you were given the opportunity to use Python to load, explore, and organize data.

For this discussion prompt, consider the following:

* Which features or tools of the Python programming language did you find to be the most useful as you were working on this project?
* How did using the Python programming language help you learn more about your data?
* What’s an element of Python and/or the data analysis process you discovered for the first time during this project?

Please write 3–4 sentences (80–100 words) responding to these questions. Then, visit the [discussion forums](https://www.coursera.org/learn/get-started-with-python/discussions) to read what other learners have written, engage with two or more posts, and share your feedback.

During my Course 2 project, I found Python's Pandas library to be the most useful tool. It allowed me to easily load, explore, and manipulate the data through its DataFrame structure and built-in functions like groupby, merge, and pivot\_table. Using Python, I could quickly visualize data trends and patterns with Matplotlib and Seaborn, enhancing my understanding of the data. One new element I discovered was the apply function in Pandas, which enables the application of custom functions to DataFrame columns, adding a layer of flexibility to the analysis process.

**Review**

-Variables

-Data types

-Functions

-Operators

-Conditional statements

-Writing clean code

-Loops

-Strings

-Data structures

-NumPy

-pandas

Since starting this course, my views on programming and the Python language have evolved significantly. I now see Python as an incredibly versatile and powerful tool for data analysis, beyond just basic scripting. My understanding of Python has expanded to include object-oriented programming, which has helped me write more modular and reusable code. Additionally, I discovered the utility of libraries like Pandas for data manipulation and Matplotlib for data visualization, which I had not used extensively before. These tools have greatly enhanced my ability to analyze and interpret data effectively.

# Get started on the next course

Congratulations on completing the second course in the Google Advanced Data Analysis certificate! Next, you’ll learn how to find and tell meaningful stories using data.

The entire program has seven courses:

1. **Foundations of Data Science:** Learnhow data professionals operate in the workplace and how different roles within the data-career space contribute to an organization’s vision of their future. You’ll explore data science roles, communication skills, and data ethics.
2. **Get Started with Python:** Discover how Python can power your data analysis. Learn core Python concepts such as data types, functions, conditional statements, loops, and data structures. (This is the course you just completed. Well done!)
3. **Go Beyond the Numbers: Translate Data into Insights:** Learn the fundamentals of data cleaning and visualizations, and how to reveal the important stories that live within data.
4. **The Power of Statistics:** Learn descriptive and inferential statistics, basic probability and probability distributions, sampling, confidence intervals, and hypothesis testing.
5. **Regression Analysis: Simplify Complex Data Relationships:** Learn to model variable relationships, focusing on linear and logistic regression.
6. **The Nuts and Bolts of Machine Learning:** Learn unsupervised machine learning techniques and how to apply them to organizational data.
7. **Google Advanced Data Analytics Capstone:** Complete a hands-on project designed to allow you to use the skills and competencies acquired in the certificate.

Now that you’ve completed this course, you’re ready to move on to the next course: [**Go Beyond the Numbers: Translate Data into Insights**](https://www.coursera.org/learn/go-beyond-the-numbers-translate-data-into-insight/home/week/1)

Keep up the great work!

**END OF THE COURSE**