**Course 2 overview**

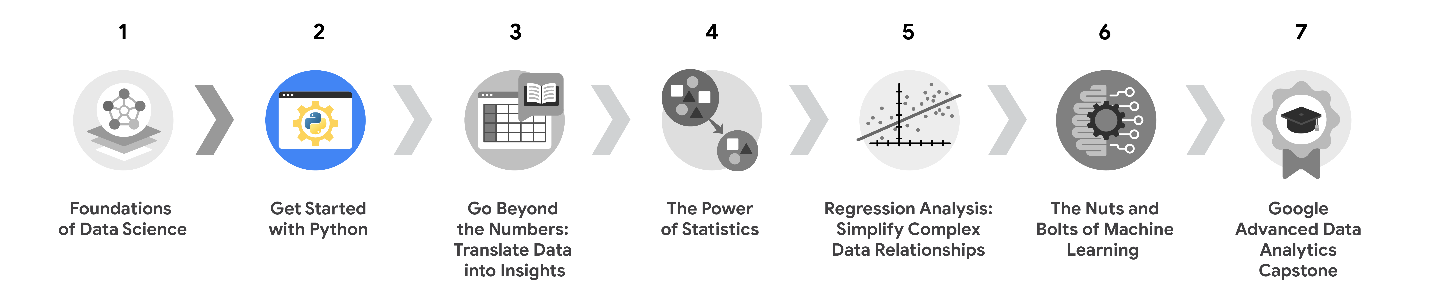


Hello, and welcome to **Get Started with Python**, the second course in the Google Advanced Data Analytics Certificate. You’re on an exciting journey!

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.

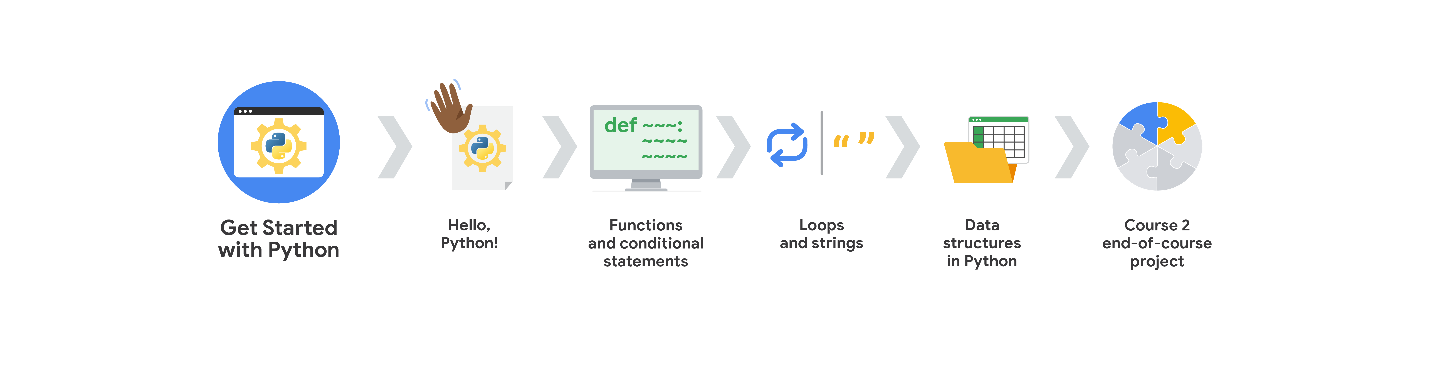


1. [**Foundations of Data Science**](https://www.coursera.org/learn/foundations-of-data-science/home/week/1) — Learnhow data professionals operate in the workplace and how different roles in the field of data science contribute to an organization’s vision of the future. Then, explore data science roles, 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 weeks. You can complete courses at your own pace, but the weekly 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 week of this course.



**Week 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.

**Week 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.

**Week 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.

**Week 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.

**Week 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.

# 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 |
| **Accessibility** | 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.

# Create, upload, and edit Jupyter Notebooks

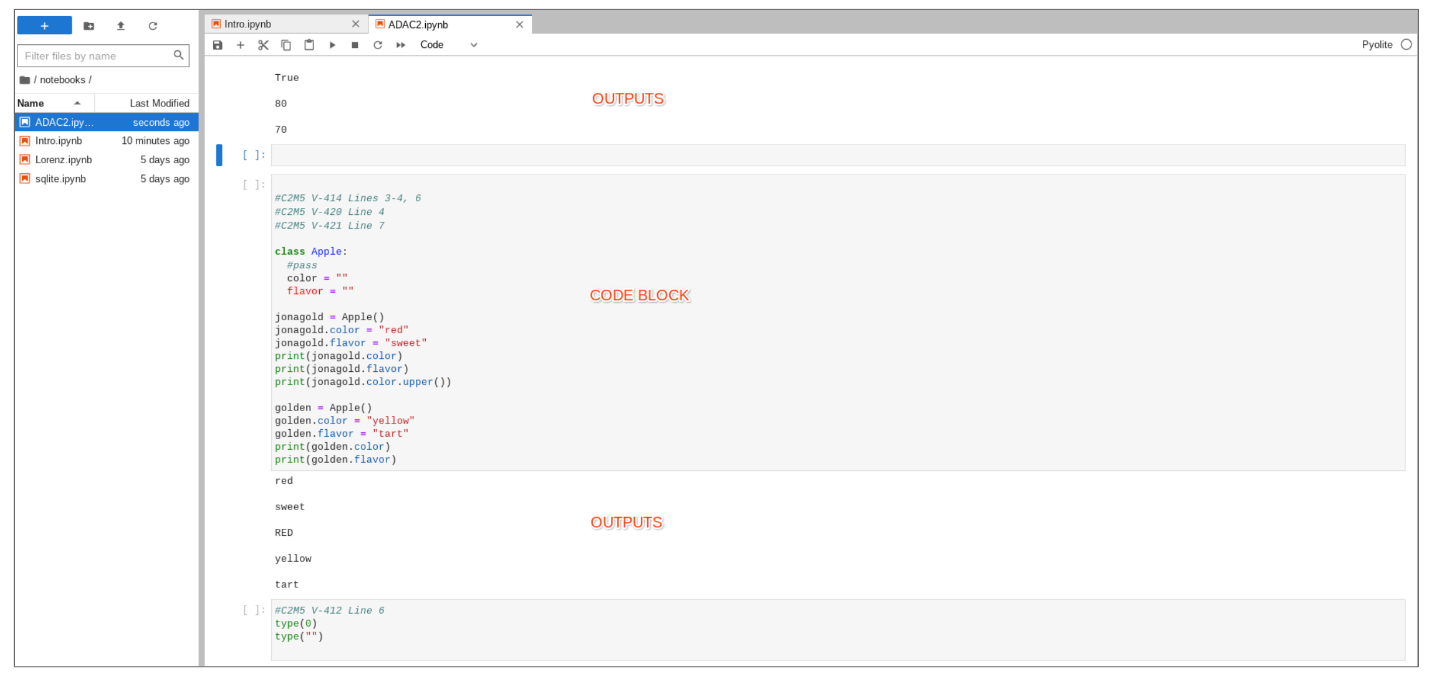
Jupyter Notebook is an open-source web application for creating and sharing documents containing live code, mathematical formulas, visualizations, and text. This is a great tool to practice developing and presenting code in a standardized text block format that is interactive and shareable. You can create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

You will be using Jupyter Notebook to write, execute, and present your own code throughout this program. This reading will guide you through creating and uploading your own notebook; that way, you’ll have everything you need to practice your Python skills in upcoming activities.

## 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.

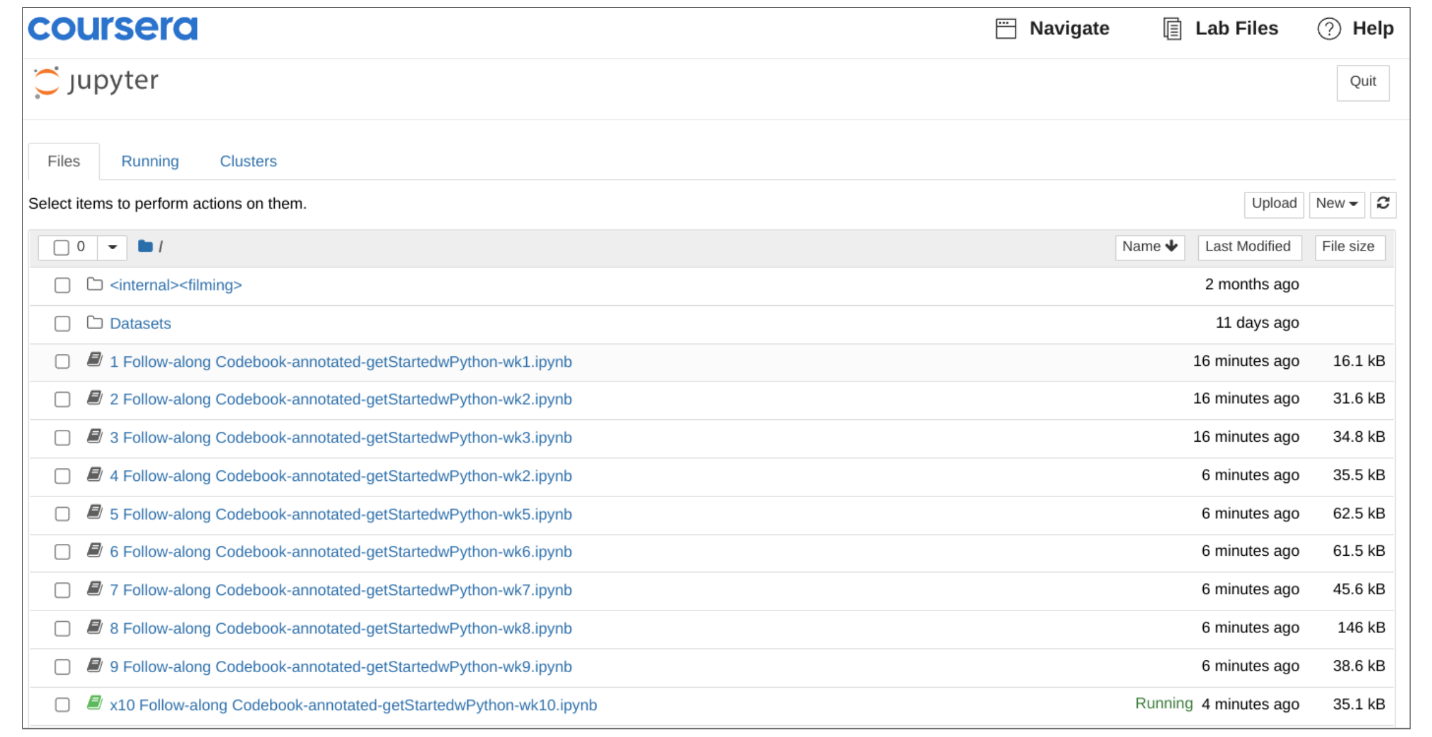
When you first access the Notebook interface, you can find both outputs and code in a horizontal block format. Select **View\Show Line Numbers** to view a code block’s line number identification.



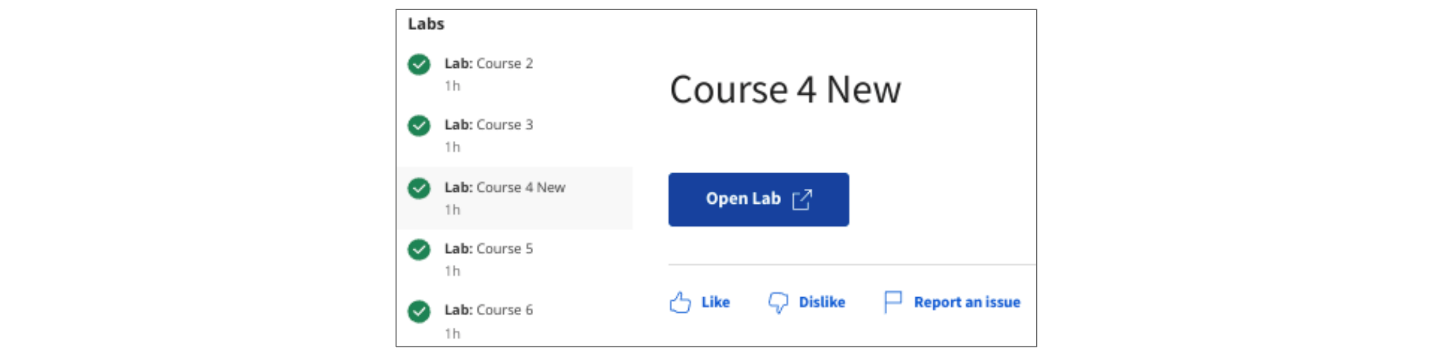
From here, you will be able to create or upload your own notebook to start coding.

## Create a notebook

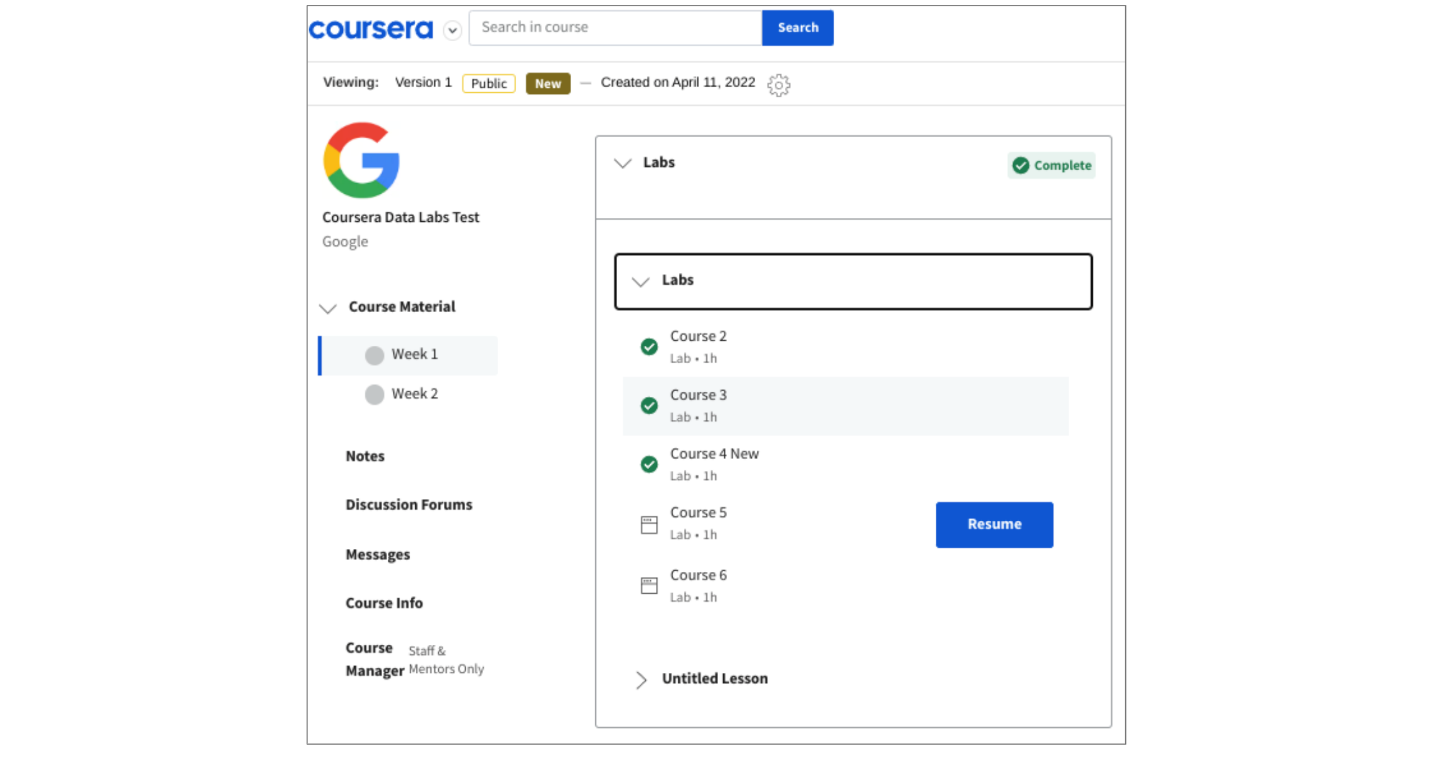
You can create and code new files directly in the Jupyter Notebook interface or upload files to work with from a drive. These notebooks can be accessed from the folder and file window, which displays what notebooks are currently available in your browser. Files uploaded to cloud software are available anywhere the internet is accessible.



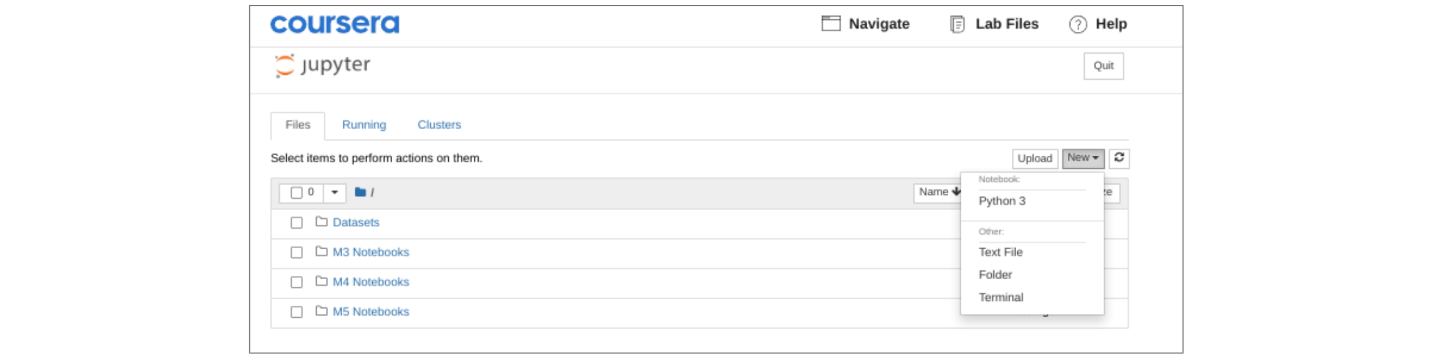
You can access the Jupyter Notebook for this program in Coursera’s Labs section. If you have not previously been in a lab, select ***Open Lab***.



Coursera Labs will either prompt you to start a new notebook or resume an existing one. Access in-progress labs by selecting the ***Resume*** button.



Select **New** to create a new notebook.



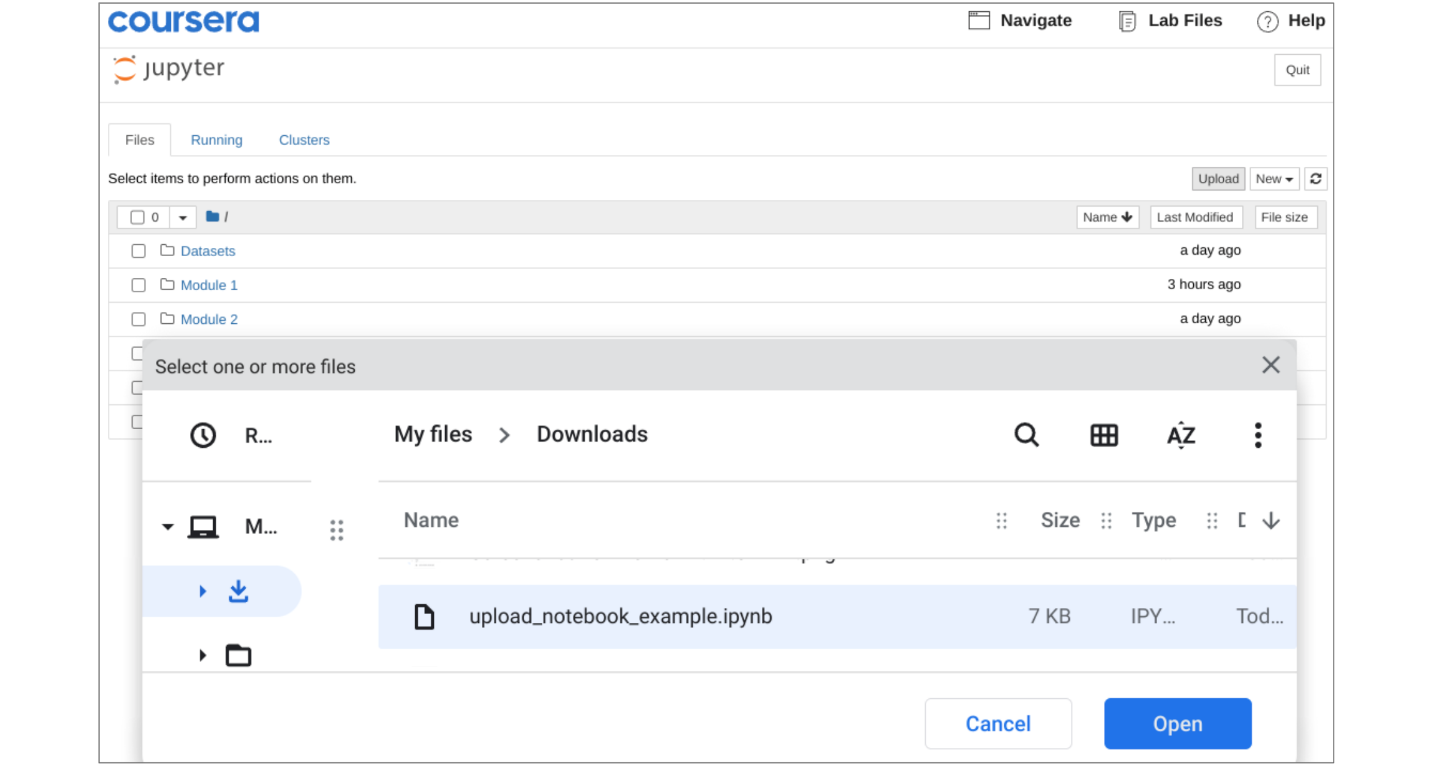
Make sure to select the ***Python 3*** engine for your notebook so that you are coding in the appropriate language.

## Upload a notebook

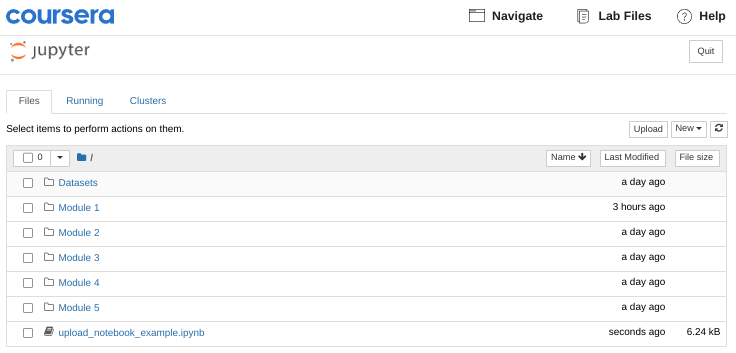
You can upload any Jupyter Notebook file or other file converted to a notebook. If the file doesn’t have an .ipynb extension, it will need to be converted in the application creating the .py extension first. You can check out [StackOverflow](https://stackoverflow.com/questions/62510114/converting-from-py-to-ipynb) for more information about how to convert .py to .ipynb files. To upload a notebook to Coursera, click the **Upload** button and select it from your files.



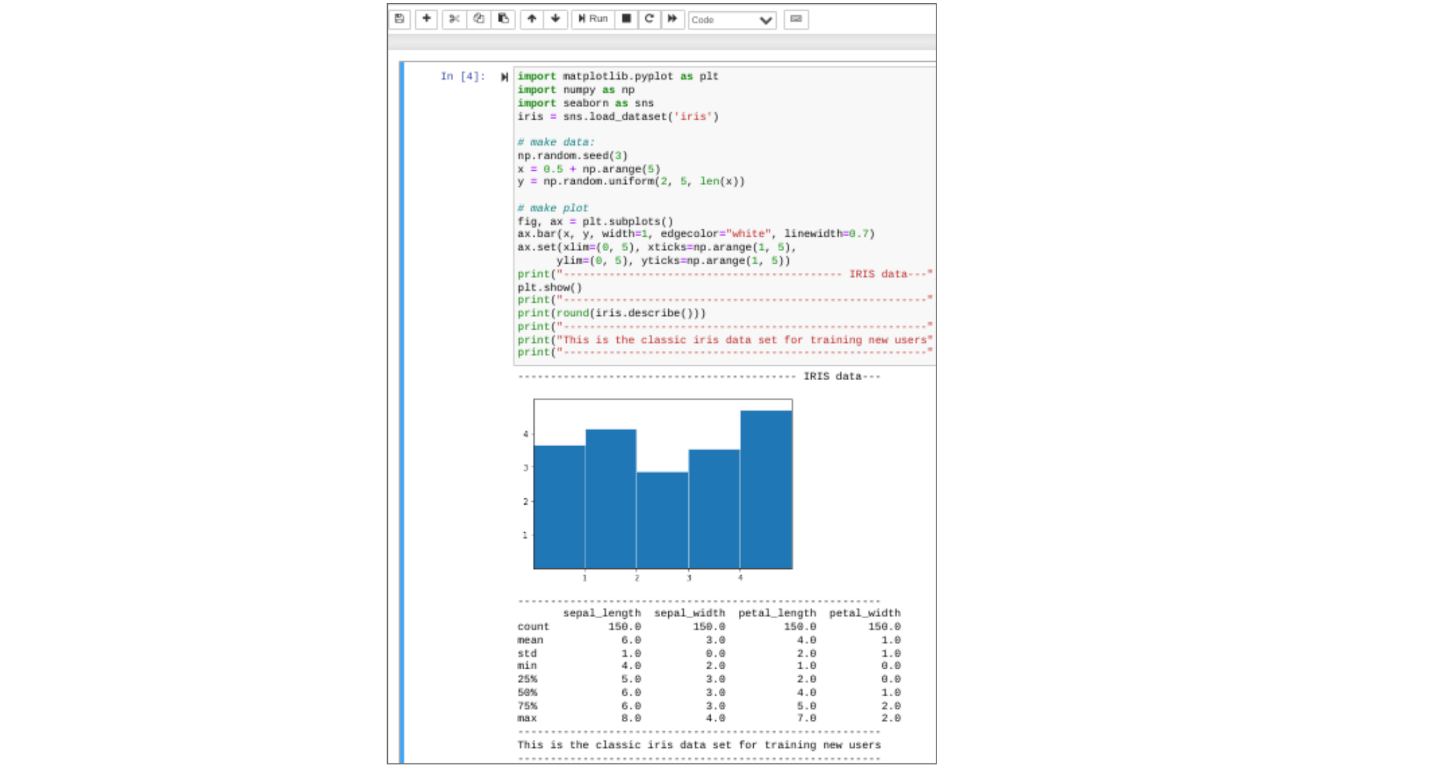
Once you have selected the appropriate file, select open to import it.



By default, files are sorted alphabetically, but you can toggle this to sort by your most recent files instead.



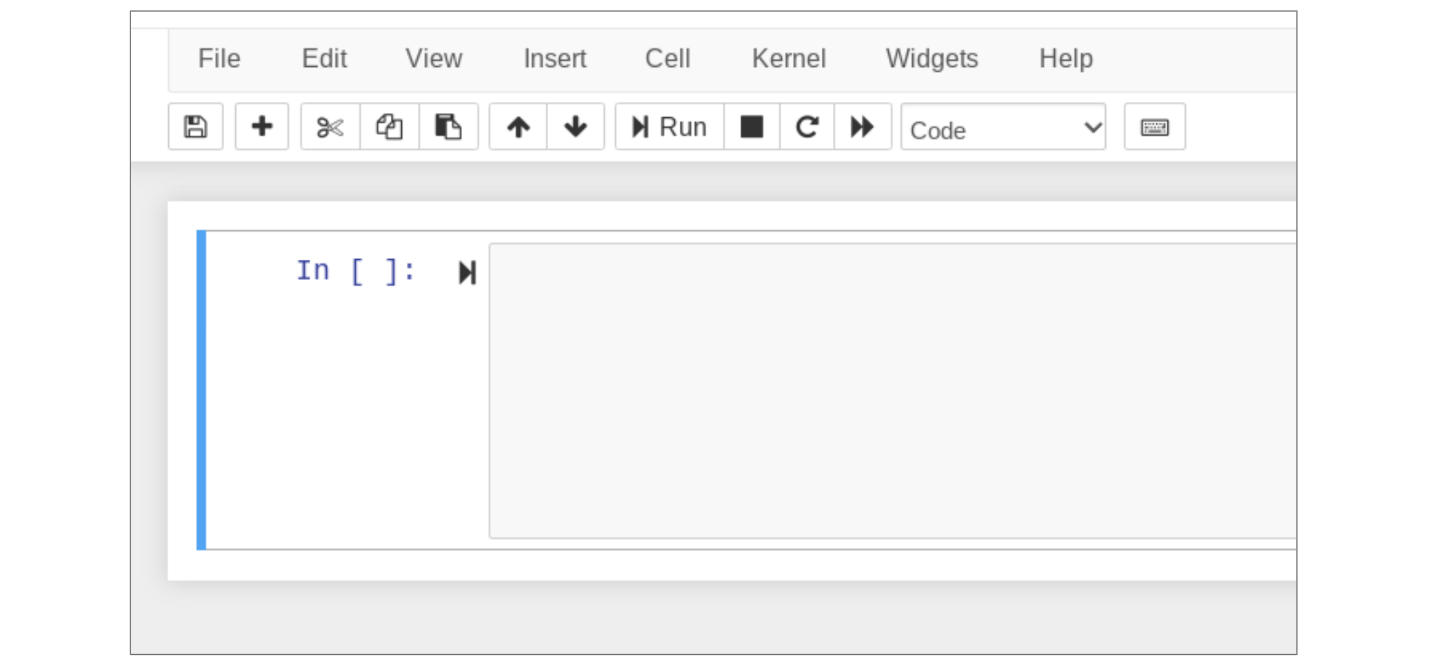
Double-click on the new notebook file to load it. **Click** the ***Run*** button when row [1] is selected to generate sample data. The following example illustrates lessons in Get Started with Python, including import libraries, using the [NumPy](https://numpy.org/) library to generate random numbers, graph outputs, and view descriptive statistics.



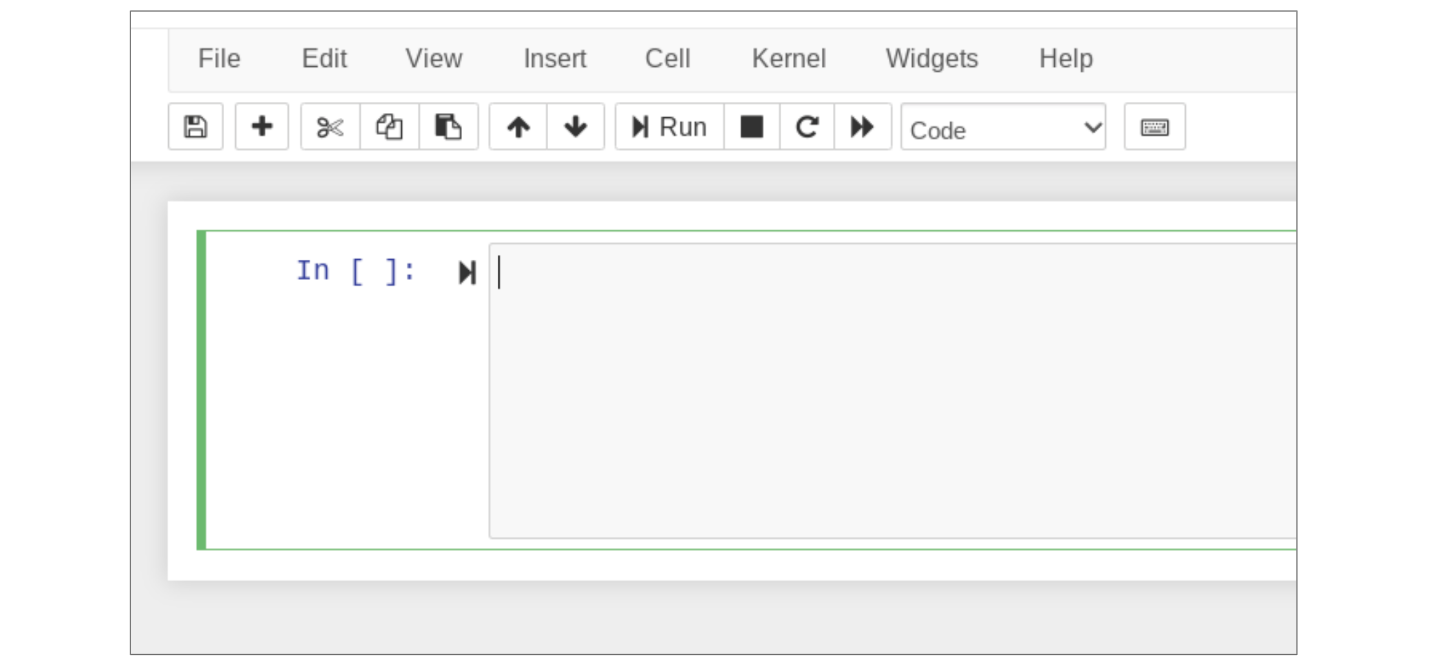
## Command/edit mode

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.



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



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.

## 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.

## 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/retro/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)

# 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 schema 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.

## 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 schema.

# 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:
    - **+**
    - **-**
    - **\***
    - **/**
    - **\*\***
    - **%**
    - **//**
    - **>**
    - **<**
    - **==**
* **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

## 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)

# Glossary terms from Week 1

# Terms and definitions from Course 2, Week 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

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