1.7 Python

- Python is an object-oriented scripting language
- Released publicly in 1991
- Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam.
- Has rapidly become one of the world's most popular programming languages
- Particularly popular for educational and scientific computing
- Recently surpassed the programming language R as the most popular data-science programming language

1.7 Python (cont.)

- Some reasons why Python is popular:
  - Open source, free and widely available with a massive open-source community
  - Easier to learn than many other languages, enabling novices and professional developers to get up to speed quickly
  - Easier to read than many other popular programming languages
  - Widely used in education.
  - Enhances developer productivity with extensive standard libraries and third-party open-source libraries
    - Programmers can write code faster and perform complex tasks with minimal code
  - Massive numbers of free open-source Python applications
  - Popular in web development (e.g., Django, Flask)
  - Supports popular procedural, functional-style and object-oriented programming
  - Build anything from simple scripts to complex apps with massive numbers of users, such as Dropbox, YouTube, Reddit, Instagram and Quora
  - Popular in artificial intelligence, which is enjoying explosive growth, in part because of its special relationship with data science
  - Widely used in the financial community
  - Extensive job market for Python programmers across many disciplines, especially in data-science-oriented positions, and Python jobs are among the highest paid of all programming jobs

Anaconda Python Distribution

- Easy to install on Windows, macOS and Linux and supports the latest versions of Python, the IPython interpreter and Jupyter Notebooks
- Also includes other software packages and libraries commonly used in Python programming and data science
- IPython interpreter

Zen of Python

- Tim Peters' *The Zen of Python* summarizes Python creator Guido van Rossum's design principles for the language
- List can be viewed in IPython with the command `import this`
- The Zen of Python is defined in Python Enhancement Proposal (PEP) 20
  - "A PEP is a design document providing information to the Python community, or describing a new feature for Python or its processes or environment"

1.8 It’s the Libraries!

- Existing libraries to help you avoid “reinventing the wheel”
- Leverage your program-development efforts
- Perform significant tasks with modest amounts of code

1.8.1 Python Standard Library

- The **Python Standard Library** provides rich capabilities for
  - text/binary data processing, mathematics
  - functional-style programming
  - file/directory access
  - data persistence
  - data compression/archiving
  - cryptography
  - operating-system services
  - concurrent programming
  - interprocess communication
  - networking protocols
  - JSON/XML/other Internet data formats
  - multimedia
  - internationalization
  - GUI
  - debugging
  - profiling
  - and more

Some of the Python Standard Library modules we use in the book
`collections` —Additional data structures beyond lists, tuples, dictionaries and sets.
`csv` —Processing comma-separated value files.
`datetime` , `time` —Date and time manipulations.
`decimal` —Fixed-point and floating-point arithmetic, including monetary calculations.
`doctest` —Simple unit testing via validation tests and expected results embedded in docstrings.
`json` —JavaScript Object Notation (JSON) processing for use with web services and NoSQL document databases.
`math` —Common math constants and operations.
`os` —Interacting with the operating system.
`queue` —First-in, first-out data structure.
`random` —Pseudorandom numbers.
`re` —Regular expressions for pattern matching.
`sqlite3` —SQLite relational database access.
`statistics` —Mathematical statistics functions like `mean` , `median` , `mode` and `variance` .
`string` —String processing.
`sys` —Command-line argument processing; standard input, standard output and standard error streams.
`timeit` —Performance analysis.

1.8.2 Data-Science Libraries

- Enormous and rapidly growing community of open-source developers in many fields
- One of the biggest reasons for Python's popularity is the extraordinary range of open-source libraries developed by its open-source community
- The following table lists various popular data-science libraries
  - You'll use many of these as you work through our data-science examples
  - For visualization, we'll use Matplotlib, Seaborn and Folium, but there are many more
    - A nice summary of Python visualization libraries (http://pyviz.org/)

Scientific Computing and Statistics

NumPy (Numerical Python)—Python does not have a built-in array data structure. It uses lists, which are convenient but relatively slow. NumPy provides the high-performance `ndarray` data structure to represent lists and matrices, and it also provides routines for processing such data structures.

SciPy (Scientific Python)—Built on NumPy, SciPy adds routines for scientific processing, such as integrals, differential equations, additional matrix processing and more. `scipy.org` controls SciPy and NumPy.

StatsModels—Provides support for estimations of statistical models, statistical tests and statistical data exploration.

Data Manipulation and Analysis

Pandas—An extremely popular library for data manipulations. Pandas makes abundant use of NumPy's `ndarray` . Its two key data structures are `Series` (one dimensional) and `DataFrames` (two dimensional).

Visualization

Matplotlib—A highly customizable visualization and plotting library. Supported plots include regular, scatter, bar, contour, pie, quiver, grid, polar axis, 3D and text.

Seaborn—A higher-level visualization library built on Matplotlib. Seaborn adds a nicer look-and-feel, additional visualizations and enables you to create visualizations with less code.

Machine Learning, Deep Learning and Reinforcement Learning

scikit-learn—Top machine-learning library. Machine learning is a subset of AI. Deep learning is a subset of machine learning that focuses on neural networks.

Keras—One of the easiest to use deep-learning libraries. Keras runs on top of TensorFlow (Google), CNTK (Microsoft's cognitive toolkit for deep learning) or Theano (Université de Montréal).

TensorFlow—From Google, this is the most widely used deep learning library. TensorFlow works with GPUs (graphics processing units) or Google's custom TPUs (Tensor processing units) for performance. TensorFlow is important in AI and big data analytics—where processing demands are huge. You'll use the version of Keras that's built into TensorFlow.

OpenAI Gym—A library and environment for developing, testing and comparing reinforcement-learning algorithms.

Natural Language Processing (NLP)

NLTK (Natural Language Toolkit)—Used for natural language processing (NLP) tasks.

TextBlob—An object-oriented NLP text-processing library built on the NLTK and pattern NLP libraries. TextBlob simplifies many NLP tasks.

Gensim—Similar to NLTK. Commonly used to build an index for a collection of documents, then determine how similar another document is to each of those in the index.

1.10 Test-Drives: Using IPython and Jupyter Notebooks

- Test-drive the IPython interpreter in two modes:
  - **interactive mode**—enter small bits of code called **snippets** and immediately see their results
  - **script mode**—execute code loaded from a file that has the `.py` extension (short for Python)
    - Called **scripts** or **programs**
- Use browser-based Jupyter Notebook for writing and executing Python code

1.10.1 Using IPython Interactive Mode as a Calculator

- Use IPython interactive mode to evaluate simple arithmetic expressions

Entering IPython in Interactive Mode

- Open a command-line window on your system
  - On macOS, open a **Terminal** from the **Applications** folder's **Utilities** subfolder
  - On Windows, open the **Anaconda Command Prompt** from the start menu
  - On Linux, open your system's **Terminal** or shell (this varies by Linux distribution)
- Type `ipython`, then press *Enter* (or *Return*)

Entering IPython in Interactive Mode (cont.)

- You'll see text like:

```
Python 3.7.0 | packaged by conda-forge | (default, Jan 20 2019, 17:24:52)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

- " `In [1]:` " is a *prompt*, indicating that IPython is waiting for your input
- Can type `?` for help or begin entering snippets, as you'll do momentarily

Evaluating Expressions

- Can evaluate expressions:

```
In [1]:
```

```
45 + 72
```

```
Out[1]:
```

```
117
```

- After you type `45 + 72` and press *Enter*, IPython
  - *reads* the snippet
  - *evaluates* it
  - *prints* its result in `Out[1]`
  - Displays the `In [2]` prompt to show that it's waiting for you to enter your second snippet
- For each new snippet, IPython adds 1 to the number in the square brackets

Evaluating Expressions (cont.)

- Evaluate a more complex expression:

```
In [2]:
```

```
5 * (12.7 - 4) / 2
```

```
Out[2]:
```

```
21.75
```

- Asterisk (`*`) for multiplication and forward slash (`/`) for division
- Parentheses force the evaluation order
- IPython displays result in `Out[2]`
- Whole numbers, like `5` , `4` and `2` , are called **integers**
- Numbers with decimal points, like `12.7` , `43.5` and `21.75` , are called **floating-point numbers**

Exiting Interactive Mode

- To leave interactive mode:
  - Type `exit` and press *Enter* to exit immediately
  - Type *Ctrl* + *d* (or *control* + *d*) then confirm
  - Type *Ctrl* + *d* (or *control* + *d*) twice

1.10.2 Executing a Python Program Using the IPython Interpreter

- Execute a script named `RollDieDynamic.py` that you'll write in Chapter 6
- **`.py` extension** indicates the file contains Python source code
- `RollDieDynamic.py` simulates rolling a six-sided die, presenting a colorful animated visualization that dynamically graphs the frequencies of each die face

Changing to This Chapter's Examples Folder

- In the Before You Begin section you extracted the `examples` folder to your user account's `Documents` folder
- Each chapter has a folder containing that chapter's source code
  - The folder is named `ch##`, where `##` is a two-digit chapter number from `01` to `17`
- Open your system's command-line window
- Use the `cd` ("change directory") command to change to the `ch01` folder:
  - On macOS/Linux, type `cd ~/Documents/examples/ch01`, then press *Enter*
  - On Windows, type `cd C:\Users\` *YourAccount* `\Documents\examples\ch01`, then press *Enter*

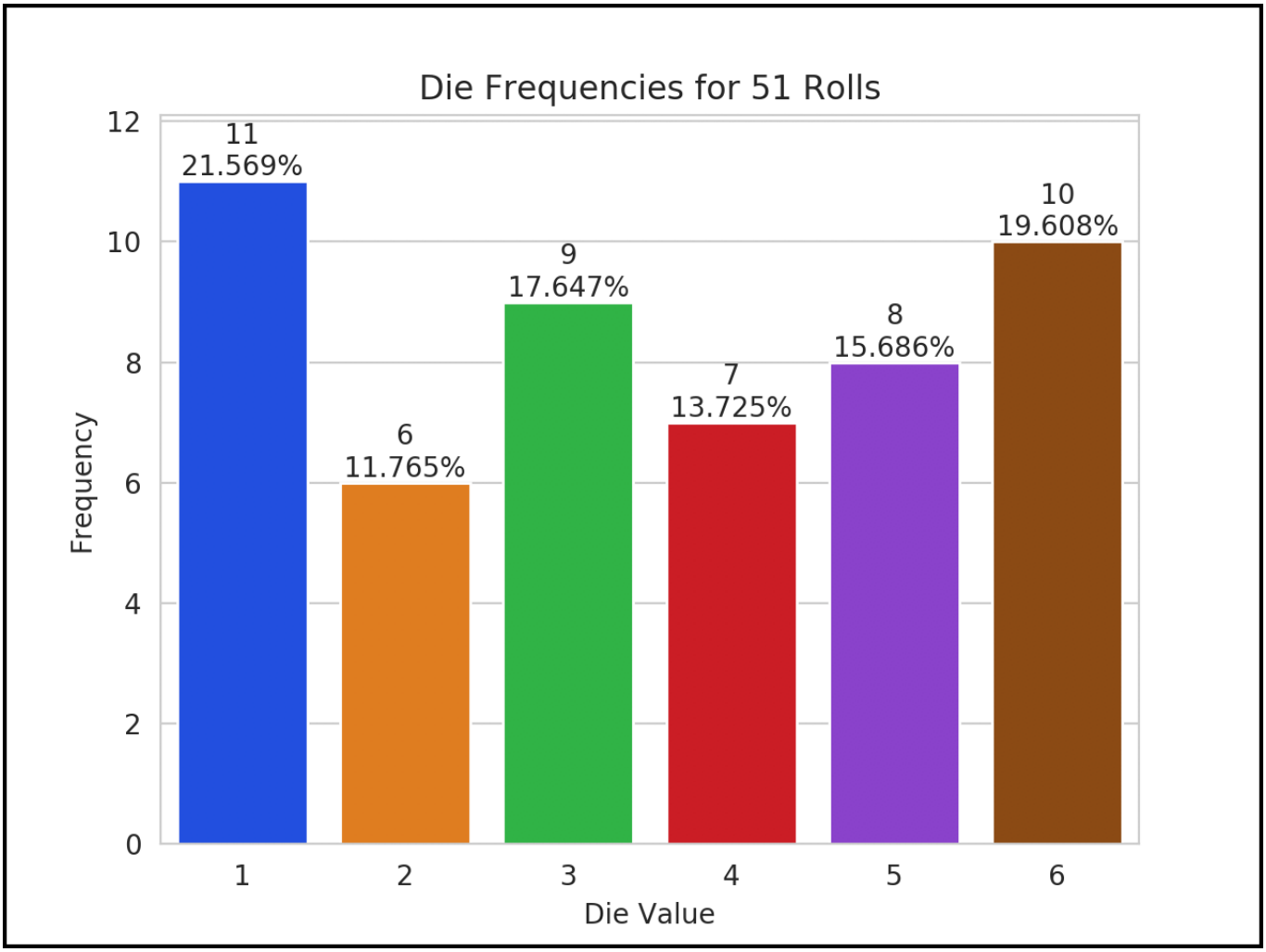
Executing the Script

```
ipython RollDieDynamic.py 6000 1
```

- The script displays a window, showing the visualization
- The numbers `6000` and `1` tell this script the number of times to roll dice and how many dice to roll each time
  - Update the chart `6000` times for `1` die at a time

Executing the Script (cont.)

- For a six-sided die, the values 1 through 6 should each occur with "equal likelihood"—1/6th or about 16.667%
- For 6000 rolls, about 1000 of each face
- *random* so there could be some faces with fewer than 1000, some with 1000 and some with more than 1000
- Experiment with the script by changing the value 1 to 100, 1000 and 10000
- As the number of die rolls gets larger, the frequencies zero in on 16.667%
  - "Law of Large Numbers"



Creating Scripts

- Typically, create in an editor that enables you to type text
- **Integrated development environments (IDEs)** provide tools that support the entire software-development process
  - editors, debuggers for locating logic errors that cause programs to execute incorrectly and more
- Popular Python IDEs include Spyder, PyCharm and Visual Studio Code and many more

Problems That May Occur at Execution Time

- Programs often do not work on the first try
  - An executing program might try to divide by zero (illegal in Python)
  - Would cause the program to display an error message
  - You'd return to the editor, make corrections and re-execute the script to determine whether the corrections fixed the problem(s)
- Errors such as division by zero occur as a program runs, so they're called **runtime errors** or **execution-time errors**
  - **Fatal runtime errors** cause programs to terminate immediately without having successfully performed their jobs
  - **Non-fatal runtime errors** allow programs to run to completion, often producing incorrect results

1.10.3 Writing and Executing Code in a Jupyter Notebook

- The Anaconda Python Distribution comes with the **Jupyter Notebook**
  - Interactive, browser-based environment
  - You can write and execute code and intermix the code with text, images and video
- Widely used in data-science and broader scientific communities
- Preferred means of doing Python-based data analytics studies and *reproducibly* communicating their results
- Supports a growing number of programming languages

1.10.3 Writing and Executing Code in a Jupyter Notebook (cont.)

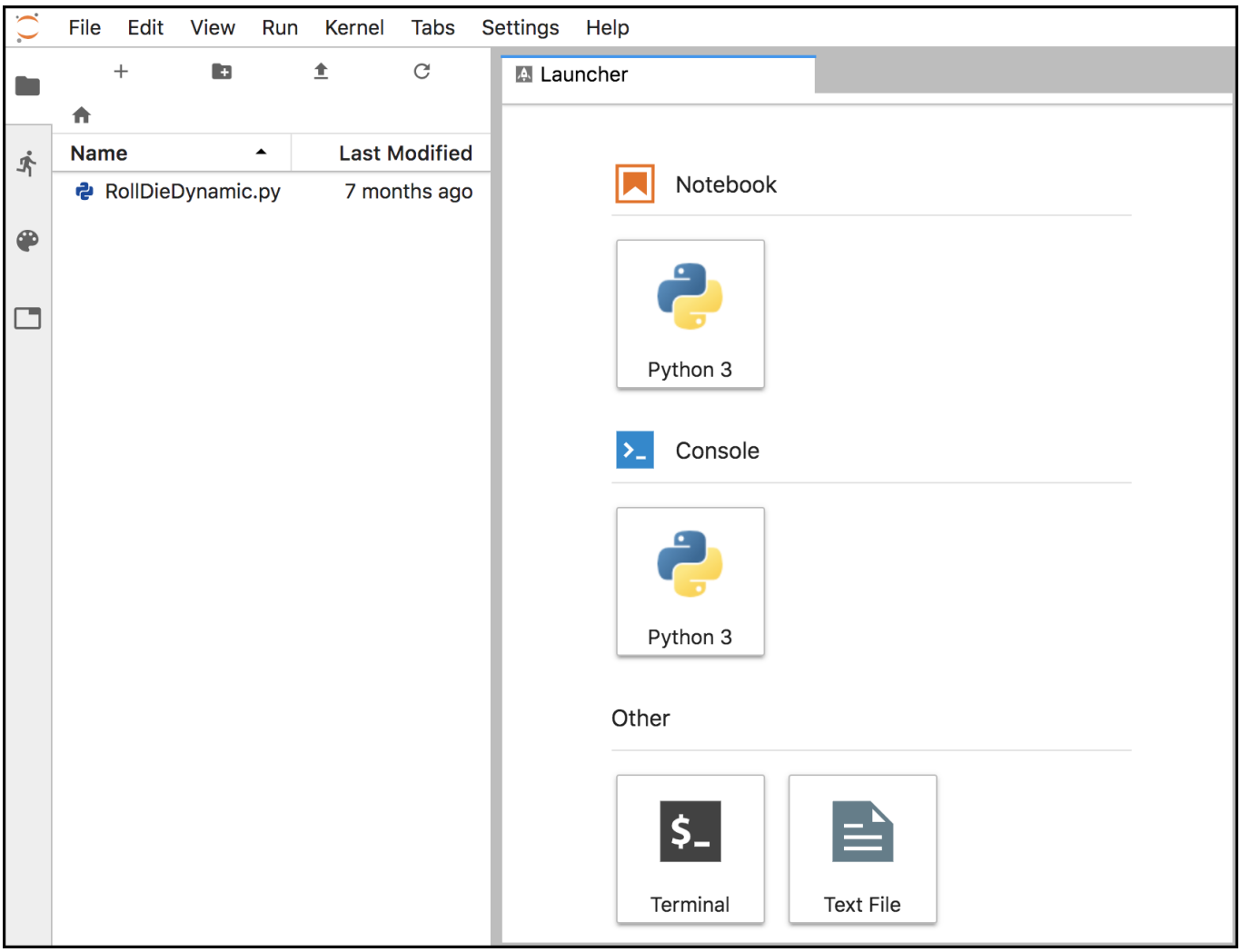
- The **JupyterLab** interface enables you to manage your notebook files and other files that your notebooks use (like images and videos)
  - Makes it convenient to write code, execute it, see the results, modify the code and execute it again
- Coding in a Jupyter Notebook is similar to working with IPython—in fact, Jupyter Notebooks use IPython by default

Opening JupyterLab in Your Browser

- To open JupyterLab, change to the `ch01` examples folder in your Terminal, shell or Anaconda Command Prompt, the execute

```
jupyter lab
```

- Launches the Jupyter Notebook server on your computer
- Opens JupyterLab in your default web browser, showing the `ch01` folder's contents in the **File Browser** tab



Opening JupyterLab in Your Browser (cont.)

- The Jupyter Notebook server enables you to load and run Jupyter Notebooks in your web browser
- From the **Files** tab, can double-click files to open them in the right side of the window
- Each file you open appears as a separate tab
- If you accidentally close your browser, you can reopen JupyterLab by entering the following address in your web browser

```
http://localhost:8888/lab
```

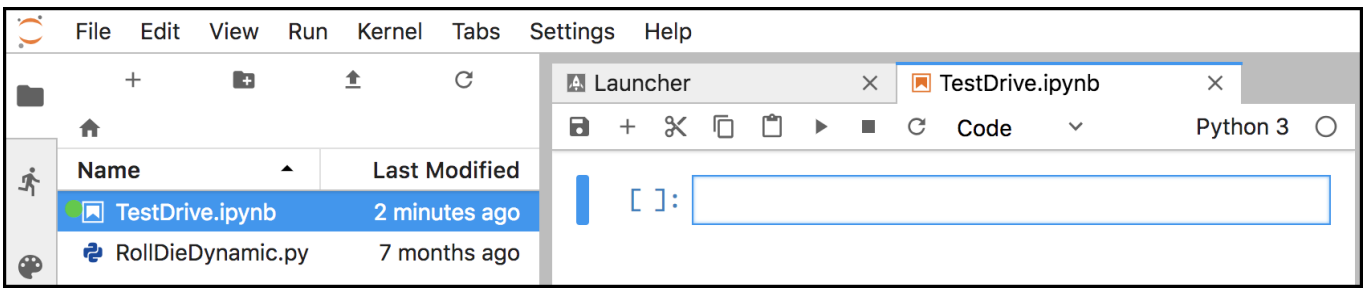
Creating a New Jupyter Notebook

- In the **Launcher** tab under **Notebook**, click the **Python 3** button to create a new Jupyter Notebook named `Untitled.ipynb`
- The file extension `.ipynb` is short for IPython Notebook—the original name of the Jupyter Notebook

Renaming the Notebook

- Rename `Untitled.ipynb` as `TestDrive.ipynb`:

  1. Right-click the `Untitled.ipynb` tab and select **Rename Notebook...**.
  2. Change the name to `TestDrive.ipynb` and click **RENAME**.

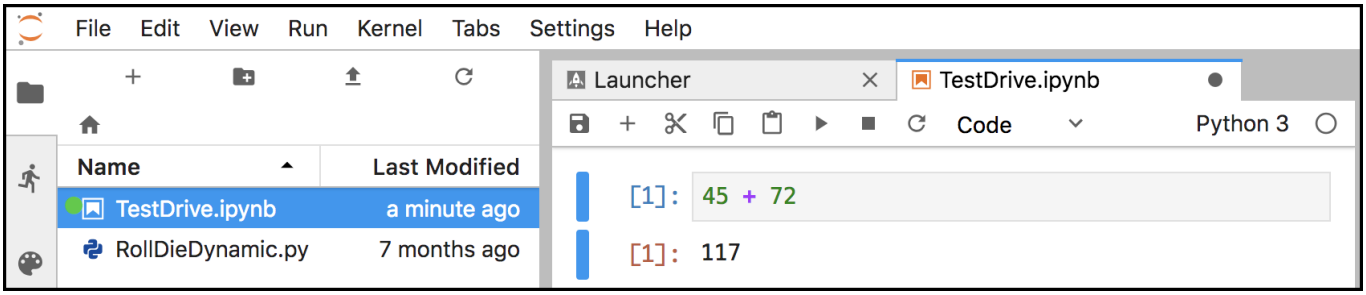


Evaluating an Expression

- Unit of work in a notebook is a **cell** in which you can enter code snippets
- By default, a new notebook contains one cell, but you can add more
- To the cell's left, the notation `[ ]:` is where the Jupyter Notebook will display the cell's snippet number *after* you execute the cell
- Click in the cell, then type the expression

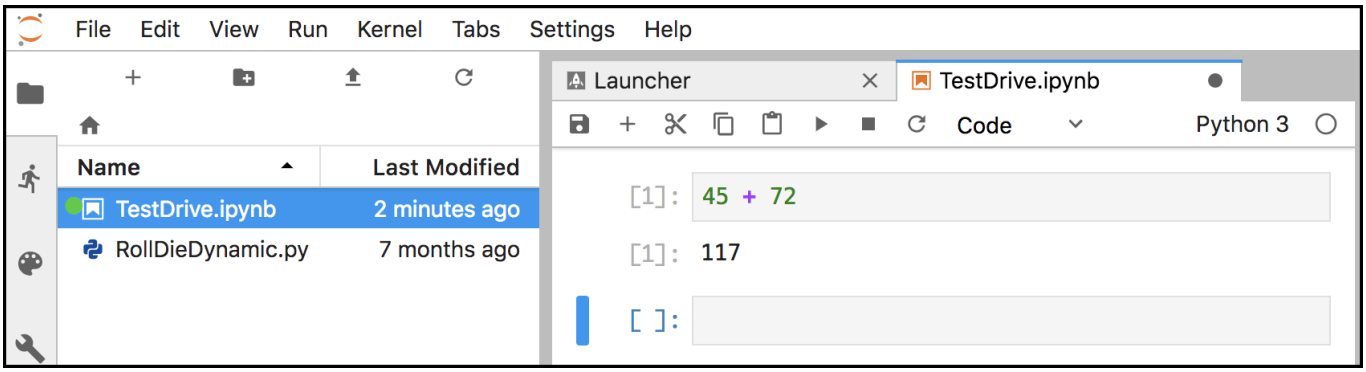
```
45 + 72
```

- To execute the current cell's code, type *Ctrl* + *Enter* (or *control* + *Enter*)
- JupyterLab executes the code in IPython, then displays the results below the cell



Adding and Executing Another Cell

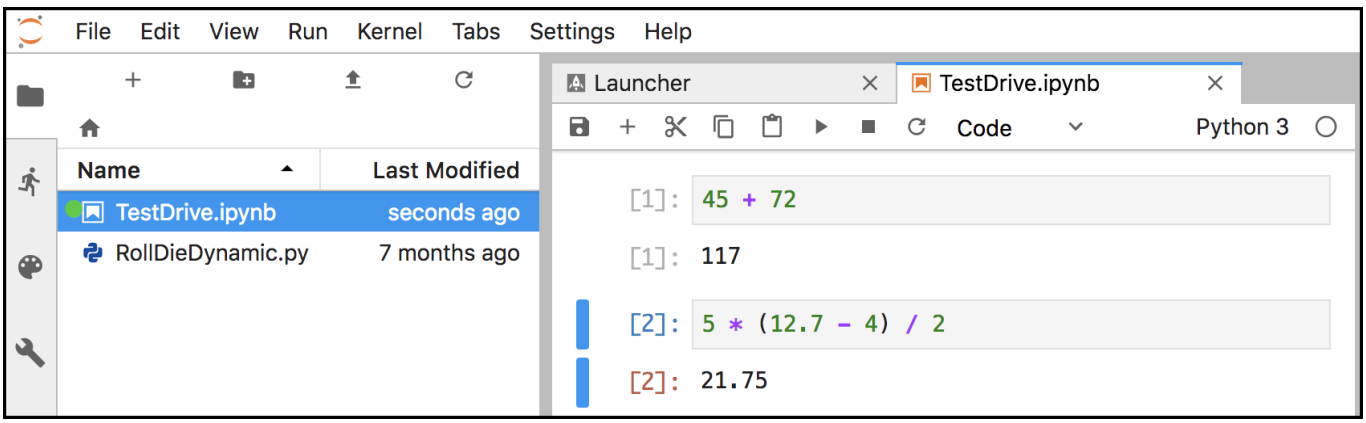
- Evaluate a more complex expression
- Click the **+** button in the toolbar above the notebook's first cell—this adds a new cell below the current one



- Click in the new cell, then type the expression

```
5 * (12.7 - 4) / 2
```

- Execute the cell by typing *Ctrl* + *Enter* (or *control* + *Enter*)



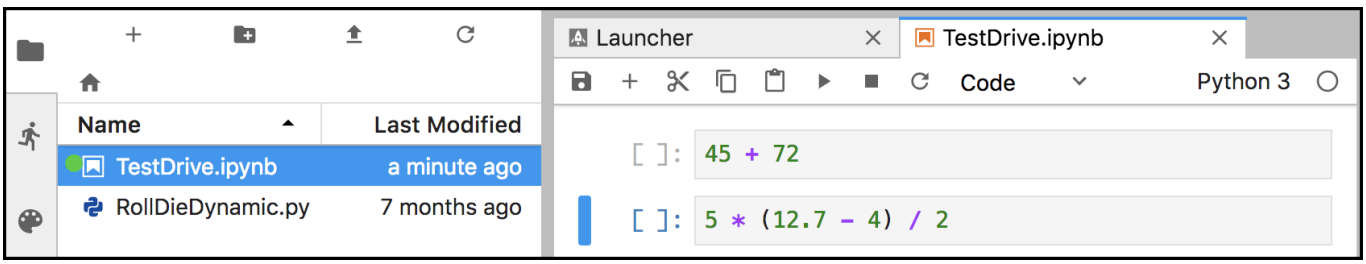
Saving the Notebook

- If your notebook has unsaved changes, the **X**in the notebook's tab will change to

- To save the notebook, select the **File** menu in JupyterLab (not at the top of your browser's window), then select **Save Notebook**

Notebooks Provided with Each Chapter's Examples

- For your convenience, each chapter's examples also are provided as ready-to-execute notebooks without their outputs
- Enables you to work through them snippet-by-snippet and see the outputs appear as you execute each snippet
- So that we can show you how to load an existing notebook and execute its cells, let's reset the `TestDrive.ipynb` notebook to remove its output and snippet numbers
  - This will return it to a state like the notebooks we provide for the subsequent chapters' examples
- From the **Kernel** menu select **Restart Kernel and Clear All Outputs…**, then click the **RESTART** button
  - Also is helpful whenever you wish to re-execute a notebook's snippets
- The notebook should now appear as follows



- From the **File** menu, select **Save Notebook**, then click the `TestDrive.ipynb` tab's **X** button to close the notebook

Opening and Executing an Existing Notebook

- When you launch JupyterLab from a given chapter's examples folder, you'll be able to open notebooks from that folder or any of its subfolders
- Once you locate a specific notebook, double-click it to open it
- Open the `TestDrive.ipynb` notebook again now
- Once a notebook is open, you can execute each cell individually, as you did earlier in this section, or you can execute the entire notebook at once
  - To do so, from the **Run** menu select **Run All Cells**

Closing JupyterLab

- When you're done with JupyterLab, you can close its browser tab, then in the Terminal, shell or Anaconda Command Prompt from which you launched JupyterLab, type *Ctrl* + *c* (or *control* + *c*) twice

JupyterLab Tips

- While working in JupyterLab, you might find these tips helpful:
  - If you need to enter and execute many snippets, you can execute the current cell *and* add a new one below it by typing *Shift* + *Enter*, rather than *Ctrl* + *Enter* (or *control* + *Enter*).
  - As you get into the later chapters, some of the snippets you'll enter in Jupyter Notebooks will contain many lines of code. To display line numbers within each cell, select **Show line numbers** from JupyterLab's **View** menu.

More Information on Working with JupyterLab

- JupyterLab has many more features that you'll find helpful
- Read the Jupyter team's introduction to JupyterLab at:

> https://jupyterlab.readthedocs.io/en/stable/index.html (https://jupyterlab.readthedocs.io/en/stable/index.html)

- For a quick overview, click **Overview** under **GETTING STARTED**
- Under **USER GUIDE** read the introductions to **The JupyterLab Interface**, **Working with Files**, **Text Editor** and **Notebooks** for many additional features

---