Chapter 1

Introduction to Computers and Python Part II

expanded by J. Goetz, 2021

Jozef Goetz contribution 2021

1.11 Internet and World Wide Web

- ***ARPAnet***
  - **Implemented** in late **1960's by ARPA** (Advanced Research Projects Agency of DOD)
  - **Networked computer systems** of a dozen universities and institutions with **56KB** communications lines
  - **Grandparent** of today's Internet
  - Intended to **allow computers to be shared**
  - Became clear that **key benefit** was allowing **fast communication between researchers** – *electronic-mail* (***email***)

1.11 The Internet and World Wide Web

- **Internet**
  - Developed more than **four** decades ago with **DOD** Department of Defense funding
  - Originally for connecting **few main computer systems**
  - Now accessible by **hundreds of millions of computers**

- **World Wide Web (WWW)**
  - Allows for locating/viewing **multimedia-based documents**

- **ARPA – The ARPANET was the predecessor to the Internet established by the United States Department of Defense Advanced Research Projects Agency (ARPA),**
- **ARPA's goals**
  - Allow **multiple users** to **send** and **receive info at same time**
  - Network operated ***packet switching*** technique
    - **Digital data sent** in **small packages** called ***packets***
    - **Packets contained**
      - **data**,
      - **address** info,
      - **error-control** info and
      - **sequencing** info
    - Greatly **reduced transmission costs** of dedicated communications lines

  - **Network designed** to be **operated without centralized control**
    - **If portion of network fails**, remaining portions still **able to route** packets

A brief history of the Internet.

- **ARPANET** (**50s** and **60s**, some universities)
- **NSFNET** (late **70s**, all universities)
- **TCP/IP** (invention **'74**) became the official protocol in 1983.
  - **When NSFNET and the ARPANET were connected, the growth became exponential**
  - Many regional networks (Canada, Europe, the Pacific) joined up
  - **In mid-8 or 100s people began viewing the collection of networks as the Internet**
  - The **glue that holds the Internet together is the TCP/IP reference model and TCP/IP protocol stack**
- **ANS** (Advanced Networks and Service) by MERIT, MCI, and IBM took over NSFNET in **1990** and form **ANSNET**
- **ANSNET** sold to **American Online** in **1995**.

1.11 Internet and World Wide Web

- ***Was used Transmission Control Protocol*** (**TCP**)
  - Name of protocols for communicating over ARPAnet
  - Ensured that **messages** were properly **routed** and that they arrived intact
- Organizations implemented own networks
  - Used both for intra-organization and communication

1.11 Internet and World Wide Web

- Huge **variety of networking hardware** and **software** appeared
  - **ARPA** achieved **inter-communication** between **all platforms** with development of the ***IP***
    - ***Internetworking Protocol***
    - **Current** architecture of Internet
  - Combined set of protocols called ***TCP/IP***

- The Internet
  - **Limited to universities** and **research institutions**
  - **Military** became a big user
  - Next, **government** decided to access Internet for **commercial** purposes

1.11 Internet and World Wide Web

- **Internet traffic grew**
  - **Businesses** spent heavily to improve Internet
    - Better service their clients
  - Fierce **competition among communications carriers** and **hardware** and **software suppliers**
    - Result
      - ***Bandwidth*** (info carrying capacity) of Internet **increased** tremendously
      - **Costs** plummeted (**dropped**)

1.11 Internet and World Wide Web - WWW

- In '90 **Tim Berners-Lee** at **CERN** (Switzerland) **invented** the **WWW (World Wide Web)**
- **Tim Berners-Lee**
  - Developed **information system based on hyperlinked text documents**
    - **Berners-Lee** called his invention the **HyperText Markup Language (HTML)**
      - allows computer users to **locate** and **view multimedia-based documents**
    - He developed **communication protocols HyperText Transf Protocol (HTTP)** as backbone
- **WWW today**
  - Makes **info instantly accessible**
  - **Merges computing** and **communication technologies**

1.11 Internet Usage

- **GUI** (**graphical user interface**) allows computer users to **locate** and **view multimedia-based documents**
- In **'93** development of **Mosaic** by **Marc Anderson**, the first **graphics-based web browser at NCSA**
  - This created an interface to the Web that was easy to use – just **point** and **click** instead of remembering **text commands**
  - This set the stage for **easier information sharing** and **retrieval**
- The ch-r of the network **was changed from an academic and military playground to a public utility**

1.11 World Wide Web Consortium (W3C)

- **W3C - 1994**
  - Founded in 1994 **by** **Tim Berners-Lee**
    - **Devoted** to developing **non-proprietary** and **interoperable technologies** for the **World Wide Web** and **making the Web universally accessible**
    - One of the W3C's goals is to make the **web accessible to everyone** regardless of disabilities, language or culture.

- **W3C Goals**
  - **Specify the role, syntax** and **rules of a technology**
    - **User Interface** Domain
    - **Technology** and Society Domain
    - **Architecture** Domain and **Web Accessibility** Initiatives

  - **Standardization**
    - W3C *Recommendations*: **technologies standardized by W3C** include
      - **E**xtensible **H**yper**T**ext **M**arkup **L**anguage (**XHTML**),
      - Cascading Style Sheets (**CSS**) and
      - the Extensible Markup Language (**XML**)

    - **Document must pass through**
      - ***Working Draft*** (specify an evolving draft),
      - ***Candidate Recommendation*** (industry can **begin** to implement) and
      - ***Proposed Recommendation*** (i.e. has been **implemented** and tested over a period of time) phases before considered for
      - ***W3C Recommendation (standards)***

1.11 World Wide Web Consortium (W3C) (II)

- **W3C Structure**
  - Comprised of **3 Hosts**
    - Massachusetts Institute of Technology (**MIT**)
    - **France's** INRIA (Institut National de Recherche en Informatique et Automatique)
    - **Keio** University of Japan
  - **400 Members** (including Deitel & Associates) that provide the primary financing

- W3C homepage at www.w3.org

Extensible Markup Language (XML).

- **XML**
  - Resulted from **HTML's** limitations
  - Data independence
- **HTML (Hypertext Markup Language) limitations**
  - **Lack of extensibility**
    - **Inability to add** or **change features**
      - Developers become **frustrated**
      - Code becomes **erroneous**
  - Led to more development on HTML
    - **W3C** created **Cascading Style Sheets (CSS)** as temporary solution
      - New technology **for formatting documents**
    - Led to research for a **standardized** extensible language
      - **W3C** developed **Extensible Markup Language (XML)**
        - Combined power of Standard Generalized Markup Language **(SGML) with simplicity of HTML**
        - Developed XML-based standards for **style-sheet**s and advanced **hyperlinking**

Extensible Markup Language (XML)

Here's an example of the above data in XML:

```
<Customers>
    <Customer>
        <LastName>JONES</LastName>
        <FirstName>JOHN</FirstName>
        <Telephone>5555551212</Telephone>
        <Address>9902 BROADWAY</Address>
        <City>NEW YORK</City>
        <State>NY</State>
        <Zip>10010</Zip>
    </Customer>
    <Customer>
        <LastName>SMITH</LastName>
        <FirstName>MABEL</FirstName>
        <Telephone>5555559999</Telephone>
        <Address>674 ANYSTREET</Address>
        <City>CHICAGO</City>
        <State>IL</State>
        <Zip>60614</Zip>
    </Customer>
</Customers>
```

Extensible Markup Language (XML) (II)

- Become the **universal technology** for data representation
- XML features
  - **Data independence**
    - **Separation** of **content** from its **presentation**
      - Because **an XML document describes data in ASCII, any application** can **process XML documents**
        - **Improves Web functionality and interoperability**
        - XML documents can be **easy manipulated** by **any app** that can process text so it **reduces server load** and network **traffic**
    - **Integration with applications** not only via Web services
      - **Communication between applications employ XML**
      - **Structure** allows **easy integration with database applications**

Extensible Markup Language (XML) (III)

- **Communication using XML**
  - **Simple Object Access Protocol (SOAP)**
    - **Technology for transmissions** of **objects over** the **Internet**.
    - A **Framework** for
      - expressing application semantics,
      - **encoding that data** and
      - **packing** it in modules
    - Structured into **three parts**
      - **Envelope**
        - Describes **content** and **recipient** of SOAP message
      - **Encoding rules**
        - Which are XML-based
      - **Remote Procedure Call** (**RPC**) **representation**
        - **Commands** other computers to perform a task

Extensible Markup Language (XML) (III)

- **Since SOAP's foundation** are in
  - **XML** and
  - **HTTP (Hypertext Transfer Protocol)**
    - The **key** communication **protocol** of the **Web**

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

1.10 Test-Drives: Using IPython and Jupyter Notebooks

•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`

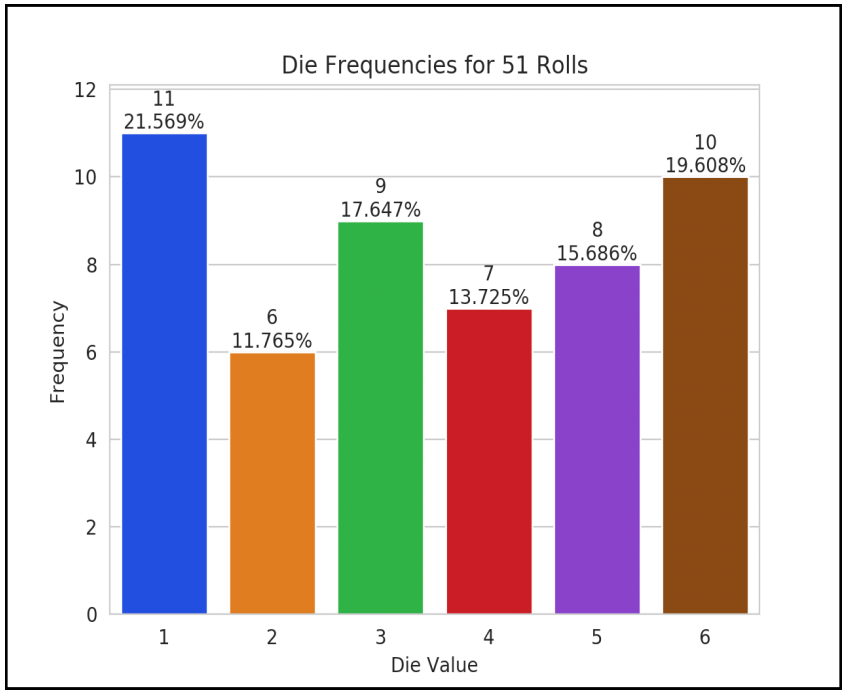
•Open your system's command-line window
•Use the `cd` ("change directory") command to change to the `ch01` folder:
- •On macOS/Linux, type `cd ~/Documents/examples/ch01`, then press *Enter*
- •On Windows, type `cd C:\Users\`*YourAccount*`\Documents\examples\ch01`, then press *Enter*

Executing the Script

ipython RollDieDynamic.py `6000 1`

1.10 Test-Drives: Using IPython and Jupyter Notebooks

- **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"



1.10 Test-Drives: Using IPython and Jupyter Notebooks

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

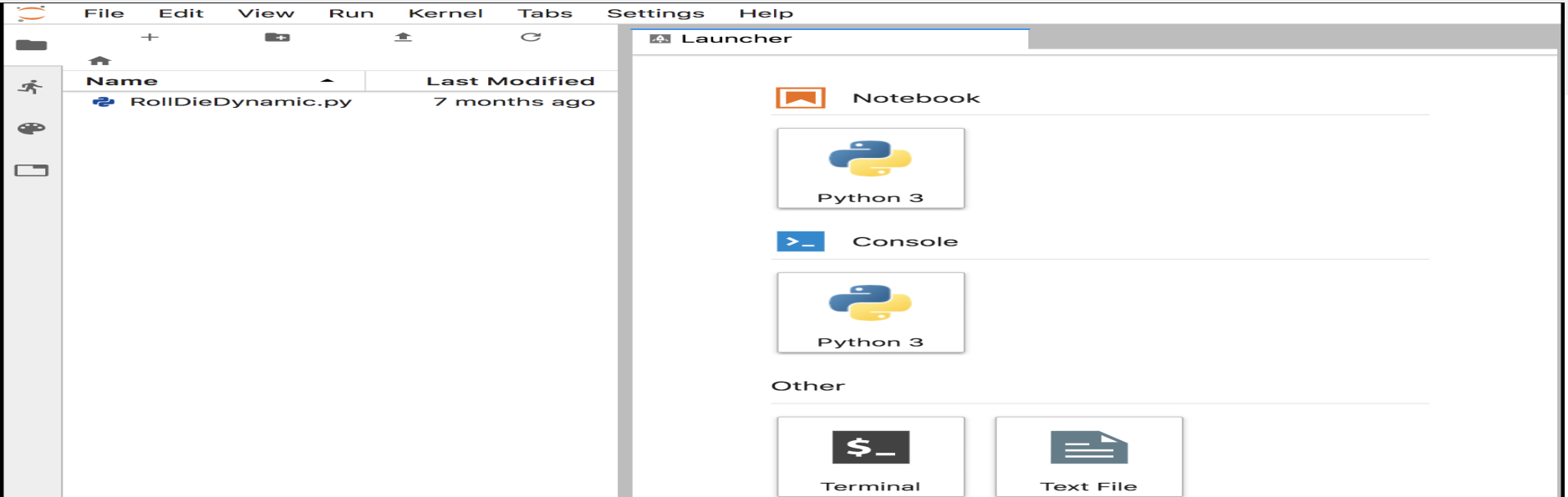
1.10 Test-Drives: Using IPython and Jupyter Notebooks

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



1.10 Test-Drives: Using IPython and Jupyter Notebooks

Opening JupyterLab in Your Browser (cont.)

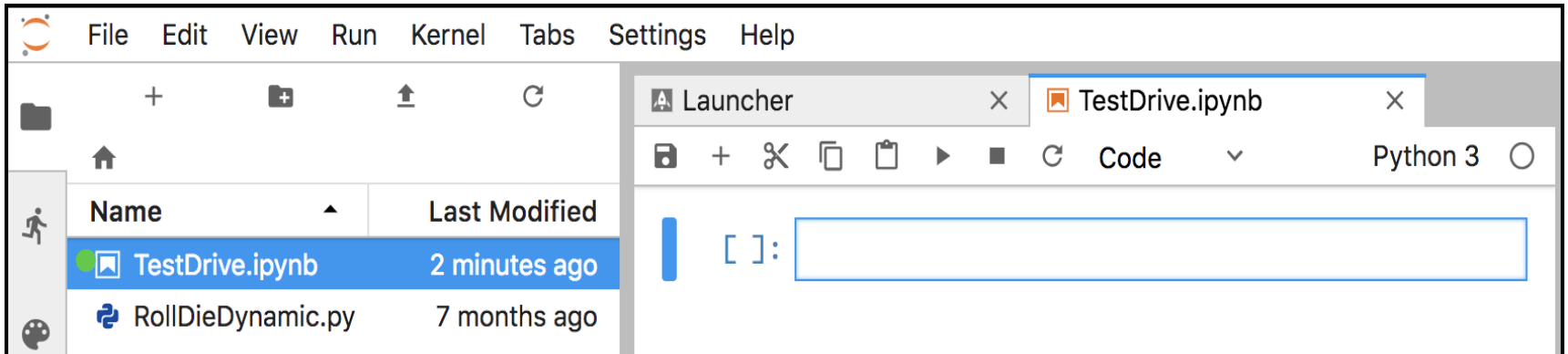
- The Jupyter Notebook server enables you to load and run Jupyter Notebooks in your web browser
- From the **Files** tab, can double-click files to open them in the right side of the window
- Each file you open appears as a separate tab
- If you accidentally close your browser, you can reopen JupyterLab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



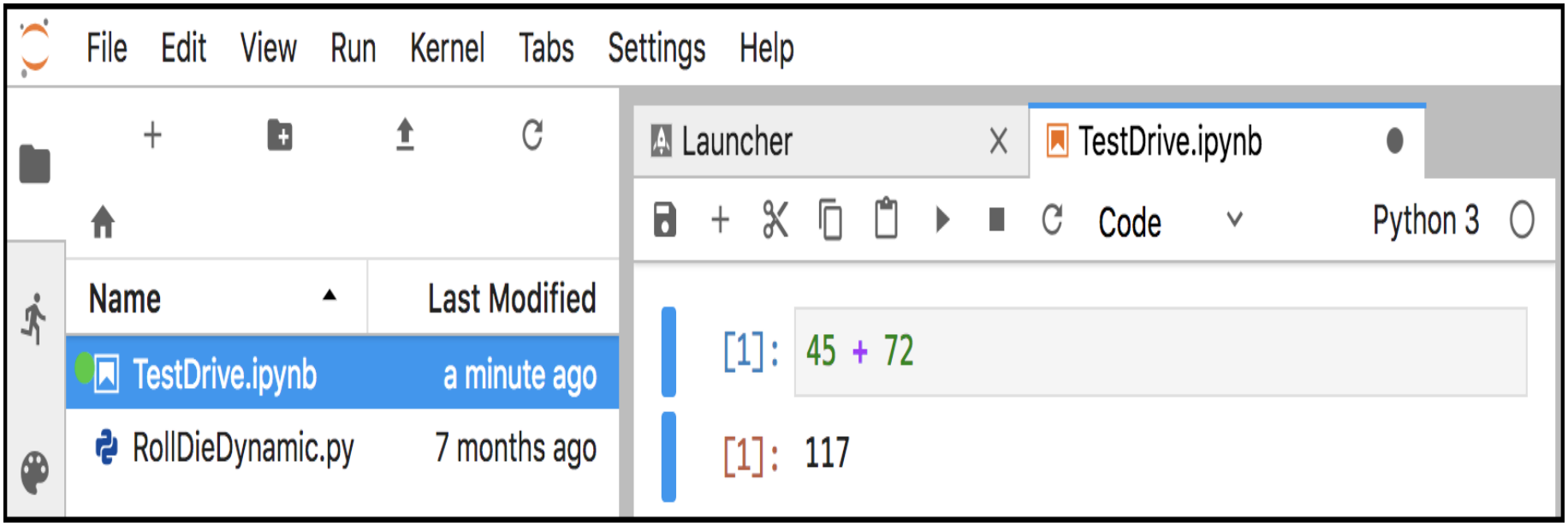
1.10 Test-Drives: Using IPython and Jupyter Notebooks

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
```

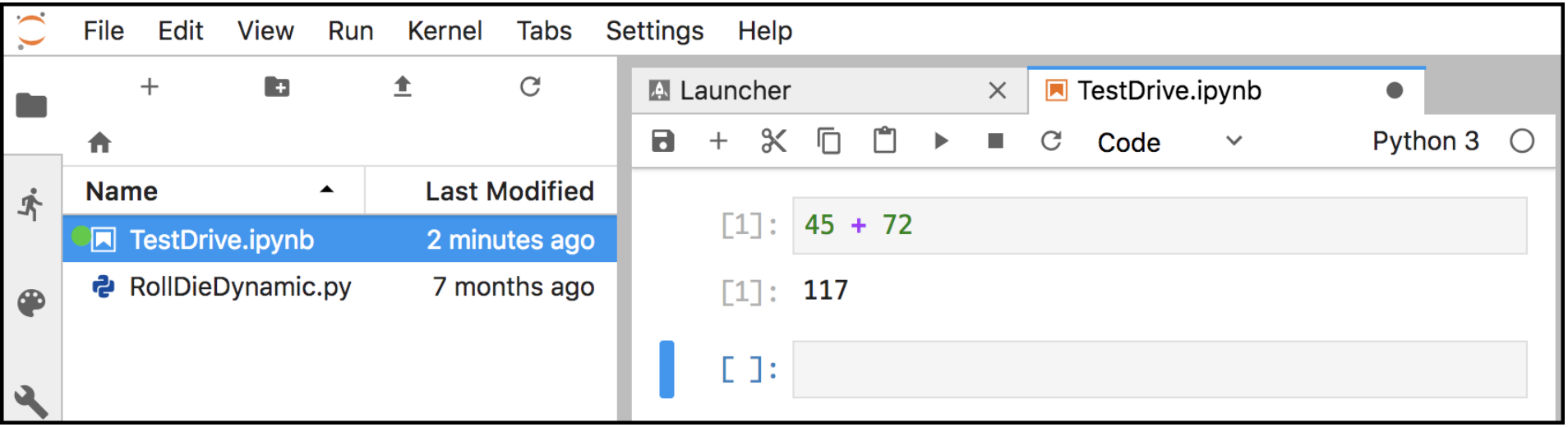
- To execute the current cell's code, type *Ctrl* + *Enter* (or *control* + *Enter*)
- JupyterLab executes the code in IPython, then displays the results below the cell



1.10 Test-Drives: Using IPython and Jupyter Notebooks

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

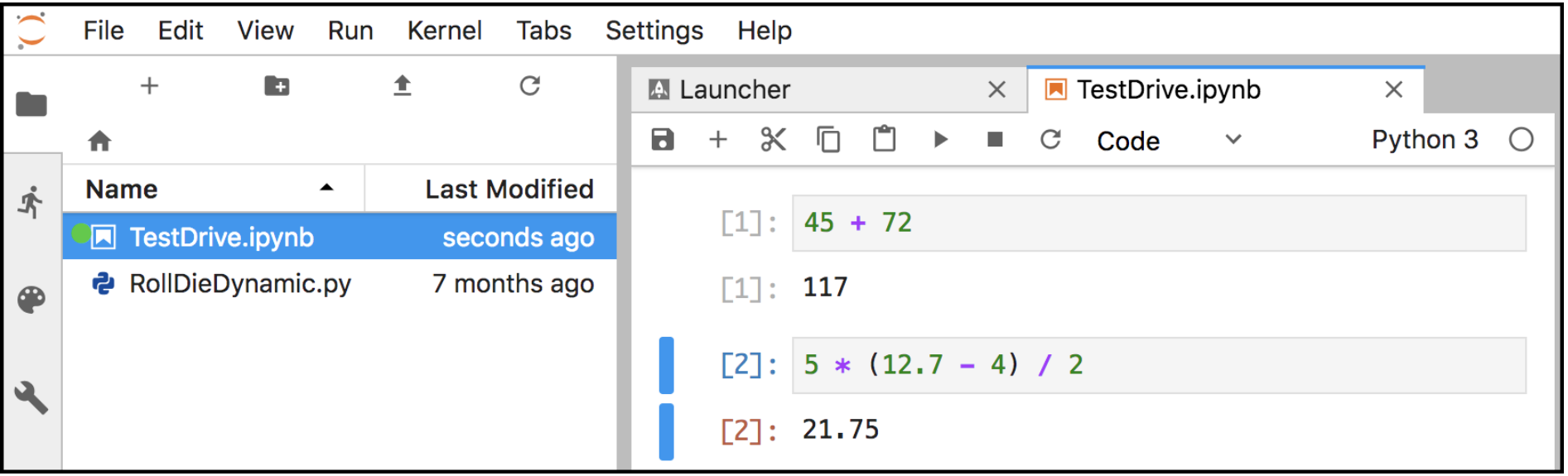


•Click in the new cell, then type the expression

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

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

1.10 Test-Drives: Using IPython and Jupyter Notebooks



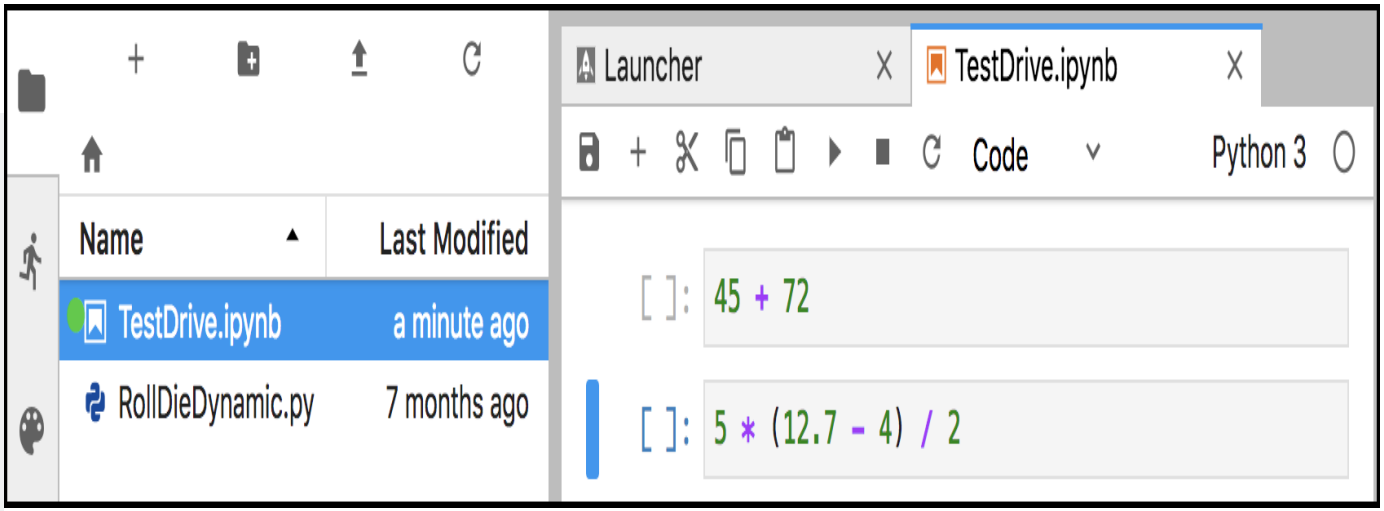
Saving the Notebook

- 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**

1.10 Test-Drives: Using IPython and Jupyter Notebooks

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

1.10 Test-Drives: Using IPython and Jupyter Notebooks

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 notebo
  - 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* 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

- 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.

1.11.3 The Cloud Computing

- **Cloud computing** allows you to use **software** and **data stored** in the **cloud** - i.e., **accessed on remote computers (or servers) via the Internet** and **available on demand** - rather than having it stored on your desktop, notebook computer or mobile device.

- Gives you the **flexibility** to i**ncrease or decrease computing resources** to meet your resource needs at any given time.

- **More cost effective** than **purchasing expensive hardware** to ensure that you have enough storage and processing power at their occasional peak levels.

1.11.3 The Cloud Computing

- Also **saves money by shifting** the burden of **managing** these **apps** to the **service provider**.
- A **service** that provides access to itself over the Internet is known as a **web service**
- Using **cloud-based services** in **Python** often is as simple as creating a software object and interacting with it.

Mashups

- *Mashups* enable you to rapidly develop powerful software applications by combining (often free) complementary web services and other forms of information feeds

- One of the first mashups combined the **real-estate listings** provided by http://www.craigslist.org with the mapping capabilities of Google Maps to offer maps that showed the locations of homes for sale or rent in a given area

- ProgrammableWeb provides a directory of over 20,750 web services and almost 8,000 mashups

- Also provides how-to guides and sample code for working with web services and creating your own mashups

1.11.4 Internet of Things

•The Internet is no longer just a network of *computers*–it′s an **Internet of Things (IoT)**

•A ***thing*** is any **object with an IP address** and the **ability to send**, and in some cases **receive, data automatically over the Internet**, such as

- •a car with a **transponder** for paying tolls
- •monitors for parking-space availability in a garage,
- •a heart **monitor** implanted in a human
- •water quality monitors
- •a smart **meter** that reports energy usage
- •radiation **detectors**
- •item trackers in a warehouse
- •**mobile apps** that can **track your movement** and location
- •**smart thermostats** that adjust room temperatures based on weather forecasts and activity in the home
- •**intelligent home appliances**

•According to https://statista.com, there are already **over 23 billion IoT devices in use today**, and there could be over **75 billion IoT devices in 2025**

1.12 Software Technologies

- Common buzzwords in software development:
- **Refactoring**:
  - Reworking programs to make them clearer and easier to maintain while preserving their correctness and functionality
  - Many IDEs contain built-in refactoring tools
- **Design patterns**:
  - Proven architectures for **constructing flexible and maintainable object-oriented software**
  - The field of **design patterns** tries to enumerate those recurring patterns, encouraging software designers to reuse them to develop better-quality software using less time, money and effort
- **Cloud computing**:
  - Uses software and data stored in the "cloud"—i.e., accessed on remote computers (or servers) via the Internet and available on demand—rather than having it stored locally on your desktop, notebook computer or mobile device
  - Can increase or decrease computing resources to meet your needs at any given time
  - More cost effective than purchasing hardware to provide enough storage and processing power to meet occasional peak demands
  - Saves money by shifting to the service provider the burden of managing these apps (such as installing and upgrading the software, security, backups and disaster recovery)
- **Software Development Kits (SDKs)**:
  - The tools and documentation that developers use to program applications.

1.13 How Big Is Big Data?

- For computer scientists and data scientists, data is now as important as writing programs
- According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily, and 90% of the world's data was created in the last two years
- According to IDC, the global data supply will reach 175 *zettabytes* (equal to 175 trillion gigabytes or 175 billion terabytes) annually by 2025
- **Megabytes (MB)**
- One megabyte is about one million (actually 2^{20}) bytes
- Many of the files we use on a daily basis require one or more MBs of storage
  - MP3 audio files—High-quality MP3s range from 1 to 2.4 MB per minute
  - Photos—JPEG format photos taken on a digital camera can require about 8 to 10 MB per photo
  - Video—Smartphone cameras can record video at various resolutions
    - Each minute of video can require many megabytes of storage
    - On one of our iPhones, the **Camera** settings app reports that 1080p video at 30 frames-per-second (FPS) requires 130 MB/minute and 4K video at 30 FPS requires 350 MB/minute
- **Gigabytes (GB)**
- One gigabyte is about 1000 megabytes (actually 2^{30} bytes
- A dual-layer DVD can store up to 8.5 GB, which translates to:
  - as much as 141 hours of MP3 audio
  - approximately 1000 photos from a 16-megapixel camera
  - approximately 7.7 minutes of 1080p video at 30 FPS
  - approximately 2.85 minutes of 4K video at 30 FPS
- Highest-capacity Ultra HD Blu-ray discs can store up to 100 GB of video

1.13 How Big Is Big Data?

Petabytes, Exabytes and Zettabytes

- There are nearly four billion people online creating about 2.5 quintillion bytes of data each day
  - 2500 petabytes (each petabyte is about 1000 terabytes) or 2.5 exabytes (each exabyte is about 1000 petabytes)
- According to a March 2016 *AnalyticsWeek* article, within five years there will be over 50 billion devices connected to the Internet and by 2020 we'll be producing 1.7 megabytes of new data every second *for every person on the planet*
- At today's numbers (approximately 7.7 billion people), that's about
  - 13 petabytes of new data per second
  - 780 petabytes per minute
  - 46,800 petabytes (46.8 exabytes) per hour
  - 1,123 exabytes per day—that's 1.123 zettabytes (ZB) per day (each zettabyte is about 1000 exabytes)
- That's the equivalent of over 5.5 million hours (over 600 years) of 4K video every day or approximately 116 billion photos every day!

1.13 How Big Is Big Data?

Additional Big-Data Stats

- For an entertaining real-time sense of big data, check out https://www.internetlivestats.com, with various statistics, including the numbers so far today of
  - Google searches
  - Tweets
  - Videos viewed on YouTube
  - Photos uploaded on Instagram

1.13 How Big Is Big Data?

Additional Big-Data Stats (cont.)

•Every hour, YouTube users upload 24,000 hours of video, and almost 1 billion hours of video are watched on YouTube every day

•Every second, there are 51,773 GBs (or 51.773 TBs) of Internet traffic, 7894 tweets sent, 64,332 Google searches and 72,029 YouTube videos viewed

•On Facebook each day there are 800 million "likes," 60 million emojis are sent, and there are over two billion searches of the more than 2.5 trillion Facebook posts since the site's inception

Additional Big-Data Stats (cont.)

•In June 2017, Will Marshall, CEO of Planet, said the company has 142 satellites that image the whole planet's land mass once per day

- •They add one million images and seven TBs of new data each day
- •They're using machine learning on that data to improve crop yields, see how many ships are in a given port and track deforestation
- •With respect to Amazon deforestation, he said: "Used to be we'd wake up after a few years and there's a big hole in the Amazon. Now we can literally count every tree on the planet every day."

Additional Big-Data Stats (cont.)

Domo, Inc. has a nice infographic called "Data Never Sleeps 6.0" showing how much data is generated *every minute*, including:

* 473,400 tweets sent. * 2,083,333 Snapchat photos shared. * 97,222 hours of Netflix video viewed. * 12,986,111 million text messages sent. * 49,380 Instagram posts. * 176,220 Skype calls. * 750,000 Spotify songs streamed. * 3,877,140 Google searches. * 4,333,560 YouTube videos watched.

Computing Power Over the Years

•Data is getting more massive and so is the computing power for processing it

•Performance of today's processors is measured in terms of FLOPS (floating-point operations per second)

•In the early to mid-1990s, the fastest supercomputer speeds were measured in gigaflops (109 FLOPS)

•Late 1990s: Intel produced the first teraflop (10^{12} FLOPS) supercomputers

•Early-to-mid 2000s: Speeds reached hundreds of teraflops

•2008: IBM released the first petaflop (10^{15} FLOPS) supercomputer

•Currently, the fastest supercomputer—the IBM Summit, located at the Department of Energy's (DOE) Oak Ridge National Laboratory (ORNL)—is capable of 122.3 petaflops

Computing Power Over the Years (cont.)

•Distributed computing can link thousands of personal computers via the Internet to produce even more FLOPS

•2016: The Folding@home network—a distributed network in which people volunteer their personal computers' resources for use in disease research and drug design—was capable of over 100 petaflops

•Companies like IBM are now working toward supercomputers capable of exaflops (10^{18} FLOPS)

1.13 How Big Is Big Data?

Computing Power Over the Years (cont.)

•Quantum computers now under development theoretically could operate at 18,000,000,000,000,000,000 times the speed of today's "conventional computers"!

•In one second, a quantum computer theoretically could do staggeringly more calculations than the total that have been done by all computers since the world's first computer appeared.

- •Could wreak havoc with blockchain-based cryptocurrencies like Bitcoin
- •Engineers are already rethinking blockchain to prepare for such massive increases in computing power

Computing Power Over the Years (cont.)

•Computing power's cost continues to decline, especially with cloud computing

•People used to ask the question, "How much computing power do I need on my system to deal with my *peak* processing needs?"

•That thinking has shifted to "Can I quickly carve out on the cloud what I need *temporarily* for my most demanding computing chores?"

- •Pay for only what you use to accomplish a given task

Processing the World's Data Requires Lots of Electricity

- Data from the world's Internet-connected devices is exploding, and processing that data requires tremendous amounts of energy.
- According to a recent article, energy use for processing data in 2015 was growing at 20% per year and consuming approximately three to five percent of the world's power
  - That total data-processing power consumption could reach 20% by 2025

Processing the World's Data Requires Lots of Electricity (cont.)

- Another enormous electricity consumer is the blockchain-based cryptocurrency Bitcoin
  - Processing just one Bitcoin transaction uses approximately the same amount of energy as powering the average American home for a week!
  - The energy use comes from the process Bitcoin "miners" use to prove that transaction data is valid

Big-Data Opportunities

- Big data's appeal to big business is undeniable given the rapidly accelerating accomplishments
- Many companies are making significant investments and getting valuable results through technologies in this book, such as big data, machine learning, deep learning and natural-language processing
- Forcing competitors to invest as well, rapidly increasing the need for computing professionals with data-science and computer science experience

1.13 How Big Is Big Data?

1.13.1 Big Data Analytics

•The term "data analysis" was coined in 1962, though people have been analyzing data using statistics for thousands of years going back to the ancient Egyptians
•Big data analytics is a more recent phenomenon–the term "big data" was coined around 2000
•Four of the V's of big data:

1. Volume–the amount of data the world is producing is growing exponentially.
2. Velocity–the speed at which that data is being produced, the speed at which it moves through organizations and the speed at which data changes are growing quickly.
3. Variety–data used to be alphanumeric (that is, consisting of alphabetic characters, digits, punctuation and some special characters)–today it also includes images, audios, videos and data from an exploding number of Internet of Things sensors in our homes, businesses, vehicles, cities and more.
4. Veracity–the validity of the data–is it complete and accurate? Can we trust that data when making crucial decisions? Is it real?

1.13.1 Big Data Analytics (cont.)

•Most data is now being created digitally in a *variety* of types, in extraordinary *volumes* and moving at astonishing *velocities*
•Digital data storage has become so vast in capacity, cheap and small that we can now conveniently and economically retain *all* the digital data we're creating

1.13.1 Big Data Analytics (cont.)

To get a sense of big data's scope in industry, government and academia, check out the high-resolution graphic
http://mattturck.com/wp-content/uploads/2018/07/Matt_Turck_FirstMark_Big_Data_Landscape_2018_Final.png

1.13.2 Data Science and Big Data Are Making a Difference: Use Cases

•The data-science field is growing rapidly because it's producing significant results that are making a difference
•Some data-science and big data use cases in the following table see p. 39-40

1.14 Case Study

1.14 Case Study—A Big-Data Mobile Application

•Google's Waze GPS navigation app, with its 90 million monthly active users, is one of the most successful big-data apps
•Early GPS navigation devices and apps relied on static maps and GPS coordinates to determine the best route to your destination
•They could not adjust dynamically to changing traffic situations
•Waze processes massive amounts of **crowdsourced data** that's continuously supplied by their users and their users' devices worldwide
•They analyze this data as it arrives to determine the best route to get you to your destination in the least amount of time

1.14 Case Study—A Big-Data Mobile Application (cont.)

•Waze relies on your smartphone's Internet connection
•Automatically sends location updates to their servers (assuming you allow it to)
•They use that data to dynamically re-route you based on current traffic conditions and to tune their maps
•Users report other information, such as roadblocks, construction, obstacles, vehicles in breakdown lanes, police locations, gas prices and more
•Waze then alerts other drivers in those locations

1.14 Case Study—A Big-Data Mobile Application (cont.)

•Waze uses many technologies to provide its services
•Below we list technologies Waze probably uses—many of which you'll see in this book
•Most apps created today use at least some open-source software (used throughout the book)
•Waze communicates information over the Internet between their servers and their users' mobile devices
- Such data often is transmitted in JSON (JavaScript Object Notation) format (Chapter 9 and subsequent chapters)
- JSON data is typically hidden from you by the libraries you use

1.14 Case Study—A Big-Data Mobile Application (cont.)

•Waze uses speech synthesis to speak driving directions and alerts to you, and speech recognition to understand your spoken commands (Chapter 14)
•Once Waze converts a spoken natural-language command to text, it must determine the correct action to perform, which requires natural language processing (NLP) (Chapter 12 and subsequent chapters)
•Waze displays dynamically updated visualizations such as alerts and maps
- Waze also enables you to interact with the maps by moving them or zooming in and out (we create dynamic visualizations throughout the book, and display interactive maps in Chapters 13 and 17)

1.14 Case Study

1.14 Case Study—A Big-Data Mobile Application (cont.)

•Waze uses your phone as a streaming Internet of Things (IoT) device (Chapter 17)
- Each phone is a GPS sensor that continuously streams data over the Internet to Waze

•Waze receives IoT streams from millions of phones at once
- Must process, store and analyze that data immediately to update your device's maps, to display and speak relevant alerts and possibly to update your driving directions
- Requires massively parallel processing capabilities implemented with clusters of computers in the cloud (Chapter 17 in introduces various big-data infrastructure technologies)

1.14 Case Study—A Big-Data Mobile Application (cont.)

•Waze uses artificial-intelligence capabilities to perform the data-analysis tasks that enable it to predict the best routes based on the information it receives (Chapters 15 and 16)
•Waze probably stores its routing information in a graph database
- Such databases can efficiently calculate shortest routes (we overview graph databases in Chapter 17)

•Many cars are now equipped with devices that enable them to "see" cars and obstacles around them
- Used, for example, to help implement automated braking systems and are a key part of self-driving car technology
- Rather than relying on users to report obstacles and stopped cars on the side of the road, navigation apps could take advantage of cameras and other sensors by using deep-learning computer-vision techniques to analyze images "on the fly" and automatically report those items (deep learning for computer vision introduced Chapter 16)

1.14 Intro to Data Science: Artificial Inteliligence at the intersection of CS and Data Science

- When a baby first opens its eyes, does it "see" its parent's faces?
- Does it understand any notion of what a face is—or even what a simple shape is?
- Babies must "learn" the world around them
- That's what artificial intelligence (AI) is doing today
- It's looking at massive amounts of data and learning from it
- AI is being used to
  - play games
  - implement a wide range of computer-vision applications
  - enable self-driving cars
  - enable robots to learn to perform new tasks
  - diagnose medical conditions
  - translate speech to other languages in near real time
  - create chatbots that can respond to arbitrary questions using massive databases of knowledge
  - much more
- The ultimate goal of all this learning is **artificial general intelligence**—an AI that can perform intelligence tasks as well as humans

1.14 Intro to Data Science: Artificial Inteliligence at the intersection of CS and Data Science

Artificial-Intelligence Milestones

- Several milestones captured people's attention and imagination, made the general public start thinking that AI is real and made businesses think about commercializing AI
- 1997: In a match between **IBM's DeepBlue** computer system and chess Grandmaster Gary Kasparov, DeepBlue became the first computer to beat a reigning world chess champion under tournament conditions
  - IBM loaded DeepBlue with hundreds of thousands of grandmaster chess games
  - DeepBlue was capable of using *brute force* to evaluate up to 200 million moves per second!
- 2011: **IBM's Watson** beat the two best human Jeopardy! players in a $1 million match
  - Watson simultaneously used hundreds of language-analysis techniques to locate correct answers in 200 million pages of content (including all of Wikipedia)
  - Watson was trained with **machine learning** and **reinforcement-learning techniques**
- Go—a board game created in China thousands of years ago—is widely considered to be one of the most complex games ever invented with 10^{170} possible board configurations
  - To give you a sense of how large a number that is, it's believed that there are (only) between 10^{78} and 10^{87} atoms in the known universe!
  - 2015: **AlphaGo**—created by Google's DeepMind group—used *deep learning* with two neural networks to beat the European Go champion Fan Hui
  - Go is considered to be a far more complex game than chess

1.14 Intro to Data Science: Artificial Inteliligence at the intersection of CS and Data Science

Artificial-Intelligence Milestones (cont.)

- More recently, Google generalized its AlphaGo AI to create **AlphaZero**—a game-playing AI that *teaches itself to play other games*
  - December 2017: AlphaZero learned the rules of and taught itself to play chess in less than four hours using reinforcement learning
  - It then beat the world champion chess program, Stockfish 8, in a 100-game match—winning or drawing every game
  - After *training itself* in Go for just eight hours, AlphaZero was able to play Go vs. its AlphaGo predecessor, winning 60 of 100 games.