PYTHON
For Data Science

The Ultimate Step-by-Step Guide to Learn Python In 7 Days & NLP, Data Science from Scratch with Python (Master The basics of Data Science and Improve Artificial Intelligence)

RICHARD WILSON

PYTHON For Data Science

The Ultimate Step-by-Step Guide to Learn Python In 7 Days & NLP, Data Science from Scratch with Python (Master The basics of Data Science and Improve Artificial Intelligence)

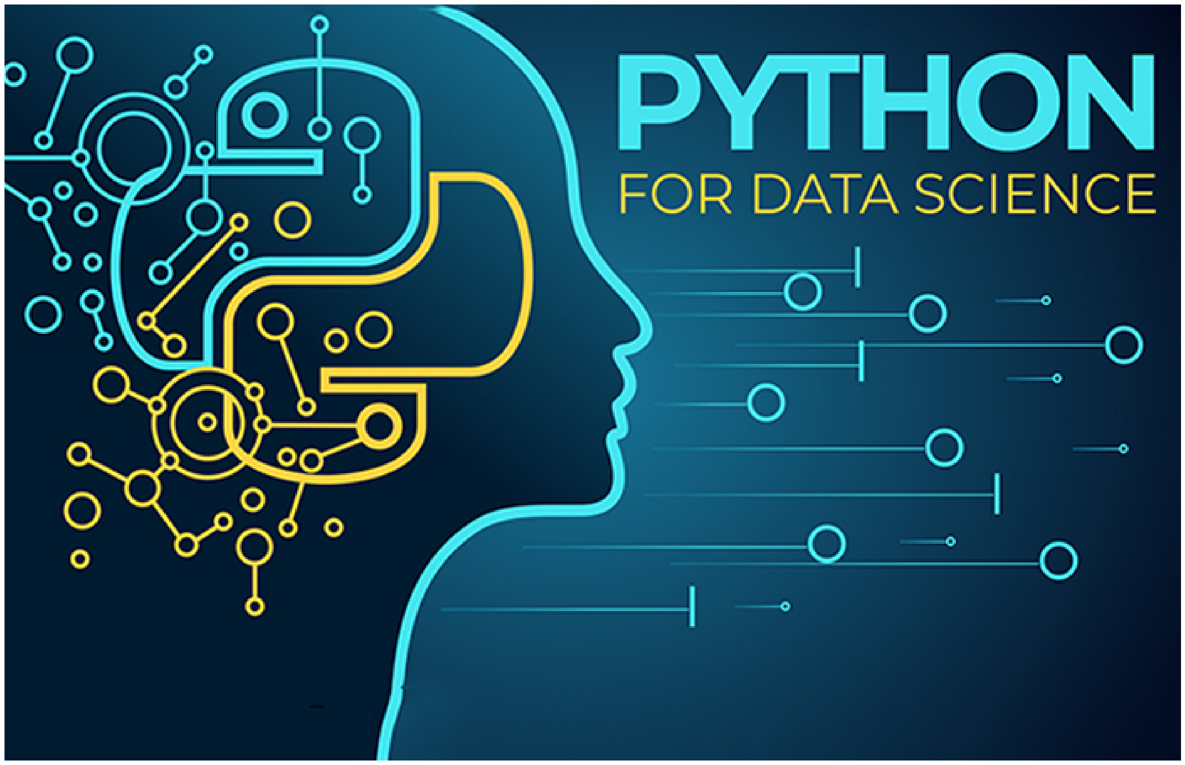
RICHARD WILSON

PYTHON

For Data Science

The Ultimate Step-by-Step Guide to Learn Python In 7 Days & NLP, Data Science from Scratch with Python (Master The basics of Data Science and Improve Artificial Intelligence)

RICHARD WILSON


PYTHON
FOR DATA SCIENCE

CONTENTS

CHAPTER 1

Data Scientist: This Is How It Is And How One Is Formed In This Increasingly Demanded Profession

The popular wisdom is clear; a data scientist is a "statistician who works in San Francisco." And, for a few years, this profession is fashionable thanks, in part, to the startup world. But data science goes much further and is becoming one of today's most promising professions.

The data fever has made us begin to hear about this discipline everywhere. But, we can't help wondering if it's a fad or data, scientists have come to stay. We review what exactly that is about data science, its job opportunities and the possibilities that exist to train.

What is a data scientist?

Another way to see it is that of Josh Wills. Wills uses another definition that seems much more accurate and intuitive to me: " Data scientist: Person who knows more about statistics than any programmed and at the same time knows more about programming than any statistic ". A little more seriously, a data scientist is simply a professional dedicated to analyzing and interpreting large databases. Or what is the same, one of the most important professionals in any internet company today.

Why has it become fashionable?

The answer was given by Javi Pastor: current technology not only needs the best talent but also data, lots of data. That is to say, that fashion for the open and the turn towards data is nothing more than the net mask of the same corporate spirit of always looking for the next site. And what is valid for artificial intelligence and machine learning environments is valid for almost any technology.

The funny thing is that this great value of the data contrasts with that precisely the data is the most abundant resource on the planet (it is estimated that 2.5 trillion bytes of new information is created per day). They don't seem easy to make things compatible. How is it possible that something so abundant is so valuable? Even if it was pure supply and demand, accumulating data should be trivial. And it is, the complex thing is to process them.

Until relatively recently we simply couldn't do it. At the end of the 90s, the field of machine learning began to take on an autonomous entity, our ability to work with immense amounts of data was reduced and the social irruption of the internet did the rest. For a few years we have faced the first great 'democratization' of these techniques. And with that, the boom of data scientists: nobody wants to have an untapped gold mine.

In search of a data scientist

The problem is that, suddenly, there has been a great demand for a profile that until now practically did not exist. Remember that you need statistical knowledge that a programmer does not usually have and computer knowledge that a statistician does not usually even imagine.

Most of the time it has been solved with self-taught training that completes the basic skills that the training program should have but does not have. That is why, today, we can find a great diversity of professional profiles in the world of data science. According to Burtch Works , 32% of active data, scientists comes from the world of mathematics and statistics, 19% from computer engineering and 16% of other engineering.

How to train

Degrees

Today, there are some double degrees in computer engineering and mathematics (Autonomous University of Madrid, Granada, Polytechnic University of Madrid, Polytechnic University of Catalonia, Complutense, Murcia Autonomous University of Barcelona) or in computer science and statistics (University of Valladolid) that seem the best option if we consider this specialization. In fact, this option seems more interesting than the possible 'degrees in data science' that could arise in the future: the possibilities are broader, the formation more diverse and allows us not to typecast.

Postgraduate

The postgraduate is a very diverse world. We can find postgraduate, masters or specialized courses in almost all universities and a truly excessive private offer. To give some examples we have postgraduate degrees at the UGR, the UAB , the UAM , the UPM or the Pompeu Fabra. However, in postgraduate courses it is more difficult to recommend a specific course. The key is to seek to complement our previous training and, in that sense, diversity is good news.

What we can find in the postgraduate training that we cannot find in the previous training is the ' business orientation ' component. We must not forget that most of the work of data scientists is in companies that seek to make their databases, profitable, because what market orientation is highly recommended. In fact, many of the masters 'big data' are offered by business schools such as OEI or Instituto Empresa.

MOOCS

One of the most interesting resources you can find are the moocs (you know, the massive open online courses). In fact recently, we saw that this self-training option could have a lot of future . Starting with the specialization program in Big Data of Coursera , we can find online courses from the best universities in the world. All this without mentioning the numerous tools to learn languages like Python or R .

Certificates and other options

There are also a series of certificates or accreditations that allow us to guarantee our knowledge in data science: Certified Analytics Professional (CAP), Cloudera Certified Professional: Data Scientist (CCP: DS), EMC: Data Science Associate (EMCDSA) or more specific certificates like those of SAS. Some of these certificates have very hard requirements, but are a good alternative if we have been working in this field before.

Other interesting resources are the associations (such as R Hispanic or Python Spain) and informal groups such Databeers so successful are having throughout the country. It is true that the ecosystem of events and meetings in data science is beginning to develop, but with the experience accumulated in other areas it is sure to be updated soon.

What languages should be learned?

In reality, as any initiate knows, in programming the choice of one language or another is always complicated. In this election, they intervene from technical or formative factors to simple personal preferences. What is clear is that there are some languages more popular than others.

The three musketeers of Data Science

An irreplaceable

- SQL: 68% of data scientists use SQL and if we include all databases, we would complete almost 100 percent of respondents. It is necessary not only because of the immense amount of data we are talking about but because most of the data used by a professional data scientist comes from the internet.

The great division

- A: Around 52% of the dateros use R for their usual work. It has in its favor that it has been the statistical language par excellence for many years and we can find codes and packages for almost anything we can think of. He has against him that his syntax is older, complex and ugly than other more modern languages that push hard. It is the language of those who approach from a scientific background.
- Python: 51% percent of Daterists use Python on a regular basis. It is the nemesis of R in this case: it has a very good and modern syntax, but there is still much work to be done to develop its ecosystem. However, to be fair, Python is

becoming more competitive and initiatives like SciPy are making things very difficult for R. It is the language of those who approach from a computer background.

Although common sense tells us that each language is better for certain things, in practice there is a certain rivalry . Personally, I use R but I usually recommend Python. Not only because it is prettier, but because it is multipurpose and that is always an advantage.

The little D'Artagnan

Julia : Julia is the white hope of data science (although as the years go by, she seems to have lost her great opportunity). A language designed to conserve the power of languages such as Fortran or C combined with the ease of the syntax of the new languages. If I had to bet, today, I wouldn't do it for Julia. He has a lot of work left if he wants to be more than the shelter of the fortraneros most open to change.

Other tools

A fireproof

- Excel: It is not a language and usually does not like those who work with professional data. Or so they say when asked why polls say otherwise: 59% percent of respondents routinely use excel. So, finally, the application of Office spreadsheets is still a lot of war .

The corporate brother and other languages and programs

- Some languages or environments enjoy some success driven by corporate inertia: it is the case of the classic Matlab, but progressively it is losing weight and use up to only 6%.
- If we examine the surveys we can find many more languages that obey more particular needs of the practice of data scientists (or the programs they use): Scala (17%), Slack (10%), Perl (12%), C # (6%), Mahout (3%), Apache Hadoop (13%) or Java (23%).
- Also, although it is possible that we should talk about them separately, there are many specific programs (free or proprietary) that are used in data science with different uses. For example, we could talk about Tableau , RapidMiner or Weka .

The labor market: salaries and opportunities

Salaries, as in general in the world of software development, change a lot depending on the place, the functions and the employer. However, right now it is a well paid expertise . On a general level and according to the annual KdNuggets survey, salaries / incomes average $ 141,000 for freelancers, 107,000 for employees, 90,000 for government workers or in the non-profit sector; 70,000 dollars for work in universities.

However, these average salaries must be taken with great caution. While the average salary in the United States is between $ 103,000 and $ 131,000, in Western Europe it is between $ 54,000 and $ 82,000. In Spain, we are in similar numbers because, despite our (increasingly smaller) deficit of product companies, we have large companies (especially banks) that have turned in this field.

What differentiates data science from the rest of the development world is perhaps the shortage of professionals. This phenomenon makes salaries relatively inflated and, as more dater profiles appear, they adjust. Therefore, it can be said that it is time to get on the wave of data science. Within a couple of years the market will have matured and the opportunities will be elsewhere.

CHAPTER 2

Python introduction

Take 5 minutes to catch up quickly with this scientific computing language

Are you looking for an easy-to-learn programming language to help you with your scientific work? Look no further than Python. We will introduce you to the basic concepts you need to know to start with this simple programming language and show you how Python is used to do everything from executing algebraic calculations to generating a graphical representation of your data.

Review of scientific computing

Scientific computing involves the use of computers to solve scientific problems. More specifically, it is used to solve equations, of everything, from simple nonlinear equations (finding roots) to systems of linear algebraic equations (linear numerical algebra), through solving systems of nonlinear partial differential equations (computational physics).

Historically, numerical algorithms to solve those problems were programmed in languages such as C / C ++ and Fortran - and still being done. So where does Python fit in? Python is excellent for quickly deploying and testing new (or old) algorithms and for gathering various physical codes, which is often done in the best US labs. UU. Python is easy, fun to learn and quite powerful. So what are you waiting for? Let us begin!

Download Python

Python is fully available for all computers running Linux or macOS operating systems. Python can be run even on an iPad by using Pythonista . You can also download a Windows version of Python.org . But, if you are going to do scientific computing, and even if you are not going to do it, I recommend that you download and install Anaconda. Anaconda provides a complete installation of Python and many of the excellent packages (or modules, as I call them) for scientific computing. It also offers easy access to the Spyder integrated development environment.

Python is at your service

When you have installed Anaconda, you can click on the Anaconda Navigator icon and start having some fun. In the lower right window is the command prompt. Just place the mouse on the right side of this symbol and start entering Python commands. If you use the traditional route to learn a new programming language, start typing print("¡Hola Mundo!"), then press Back .

You can use the command prompt to enter one or more commands to quickly test the code icons or to generate the results of the job. When it comes to using more than a few lines of code, it is better to generate and save a program file separately.

The other option, at least on Linux and MacOS, is to open a terminal window and type Python in the command prompt. When you do this, the Python command prompt starts, where you can start typing commands and execute the Python code. If, instead, it is written idle in the terminal window, a new window appears that presents the Idle Python editor - which is suitable for writing new Python scripts and for executing them using the powerful command F5.

What do the names contain?

Now that you have Python installed and that you know how to start typing commands, you can continue and do some math and science. To program computers to solve equations, you have to use variables and manipulate the numbers that represent these variables. For example, define a variable in Python by typing the following command at the command prompt:

```
One:  >>>   X0 =1.5
Two:   >>>   X1  =2.0
```

Congratulations! They have simultaneously created two new variables with their names x0and x1and assigned the values 1.5y 2.0, respectively. To see it in action, type:

```
One:  >>>   X0,X1
```

Or you can use the function print:

```
One:  >>>   Print   (x0,x1)
```

It is not necessary to declare these variables as real (floating point numbers) or as integers (whole numbers) because Python is a dynamically written language; determines the type of variables on the fly, based on the values assigned to it.

Computers and Algebra

You have two variables that have been assigned to two numbers, so right now you can do some simple algebraic calculations with them; You can add them, subtract them, multiply them or divide them as you wish. This is what computers are better at. To see this algebra running, type the following command at the command prompt:

```
One:  >>>   yp   =   x0 +  x1
Two:  >>>   ym   =   x1  -x0
3:   >>>   yt   =   x0*x1
4:   >>>   yd   =   x1/x0
5:   >>>   print (yp,ym,yt,yd)
```

Now you are officially done scientific calculations.

Computers and logic

If computers could only do algebra calculations, their impact on scientific computing would be very limited. The fact that computers are also good in logic is what makes it possible to create complex programs. You may be familiar with the logic, if this occurs, then that (IFTTT). This is not exactly what I am talking about, but it looks like it. I refer to program flow control, or the ability to execute a line or block (group of lines) of code only under certain conditions and other lines or blocks of code under other conditions. To see what this really means, type the following command:

```
One: >>> x1 = 2.0
Two: >>> if x1 < 0:
3: >>> x = 0
4: >>> print (“Negativo Cambiado a c
ero”)
5: >>> elif x1 == 0:
6: >>> print (“cero”)
7: >>> else:
8: >>> print (“x1 Es positivo”)
```

That code is an example of a block if, where it elifis the abbreviation for else if, and else is executed if the tests of the two (or those that you need) blocks of previous code fail. For a more detailed explanation, take a look at More Flow Control Tools in the Python documentation.

The power behind many scientific computing algorithms is related to the ability to execute the same blocks of code several times with different data. This is where the loops are useful. Consider the following code icon, which uses the built-in function range Python to generate a list of 10 integers, starting with 0:

```
One: >>> x0 = 1.5
Two: >>> for i in range(10):
3: >>> x1 = x0 + i
```

This code executes the calculation x1 = x0 + i 10 times, starting with i = 0 and ending with i = 9.

What is your function?

The functions initiate the important programming process of breaking down a large number of programming tasks into small sets of subtasks, or functions. Python has built-in external libraries and functions that I will explain later. You can also create your own functions. Functions are created using the Python keyword def, as shown below, for the called function f, which receives the input variable x and returns the value resulting from the evaluation of the programmed algebraic expression:

```
One: >>> def f(x):
Two:     return x**3 - 5*x**2 + 5*x +
 1
3: >>> x0 = 1
4: >>> print (“f(x0) = “, f(x0))
```

To create a function to calculate the analytical derivative of the previous function, type:

```
One: >>> def fp(x):
Two:    return 3*x**2 - 10*x + 5
```

```
3:  >>>  x1  =  2
4:  >>>  print  (“La  derivada  de  f
(x)  Es:  “,  fp (2))
```

Archive this

So far you have entered all your Python commands at the command prompt, which is fine for a short and disposable code. But if you are working on a larger project or if you want to save your work for later, I recommend that you create a new Python file or script . You can do this from the terminal window with the text editor of your choice. For example, with vi, you just have to write vi newton.pyto create an empty text file called newtonputting .py as the value of the file extension to let everyone (and the computer, in particular) know that this is a Python file. Then, with the file open, you can start writing Python commands.

Note: Python uses blanks to indicate blocks of code. The standard convention is to use four spaces to indent a block of new code, such as lines of code that constitute a function, or a block if-then.

Another important aspect of writing programs is the comment lines that explain to the one who is reading the file what the script does. Single line comments begin with the pound symbol (#); To create multi-line comments, start with a backslash followed by # (\#), and write them down. #\. After entering the code, save it and exit the editor. Then run the code from the command line of the terminal window (assuming you are in the same directory as the file you saved) by typing python newton.py.

In scientific computing it is usually a good idea to break down a problem or task into smaller problems or subtasks. In Python, these subtasks are known as modules. The modules are only additional Python files (which have the. by file extension) that contain definitions and declarations. Module models are also available. You can use any module within your program by importing it with the keyword import. For example, the mathematical module contains basic mathematical functions such as sine and cosine; It can be used by typing the keyword import math.

How to import the power of scientific computing into Python

Two powerful modules of scientific computing that you may want to use in Python are NumPy and SciPy . Bumpy contains many powerful features, but the interesting one for this example is the ability to create collections of numbers of the same type of data and assign them to a single variable, called an array.It also has extensive linear algebra, Fourier transforms and random number creation capabilities. Zippy is the dominant scientific computing ecosystem that contains bumpy and many other packages, such as matplotlib. The following code provides a quick example of how to import NumPy and how to use it in a code icon:

```
One:   import numpy as np

Two:   /*   Ahora  Creamos  y asignamos   una lista de numeros
enteros de 0 a 9 para las

One:   import  numpy  as  np

Two:   /*   Ahora  Creamos   y   asignamos  una  lista  de  nume
ros  enteros  de  0  a  9  para  las

3:   variables  de  x[0]  a

4:   x[9],  lo  que  en  la  practica  crea  un  array  unidimensiona
l  de  Numpy  */

5:   x  =   np.Linspace (10)

6:   print(x)
```

How to use Python to generate a graphical representation

Generating an effective graphic representation from scientific computing data is the key to understanding and informing the results. The standard Python package to achieve this important goal is the matplotlib module. It can be easily accessed and used, as you can see here:

```
One:  >>>  import  numpy  as  np
Two:  >>>  import  matplotlib.pyplot  as  plt
3:  >>>  x  =  np.arange(0,  5,  0.1)
4:  >>>  y  =np.sin(x)
5:  >>>  plt.plot(x,y)
6:  >>>  plt.show()
```

The matplotlib module has commands to control the type of line, colors and style, and also to save the diagram.

CHAPTER 3

Languages do you need to learn from data science

When it comes to advanced data science, you can only reinvent the wheel every time. Learn to master the different packages and modules offered in the language of your choice. The extent to which this is possible depends primarily on the domain-specific packages that are available to you!

Generality

A high-level data scientist will have good programming skills, as well as the ability to reduce numbers. The bulk of the work daily in data science is devoted to research and processing of raw data or "data cleansing". For this, no amount of sophisticated machine learning packages will be useful.

Productivity

In the rapidly changing world of commercial data science, there is a lot to be said for doing the job quickly. However, this is what allows the technical debt to creep - and it is only through sound practices that it can be minimized.

Performance

In some cases, it is essential to optimize the performance of your code, especially when dealing with large volumes of critical data. The compiled languages are generally much faster than those interpreted; similarly, static typing languages are considerably more reliable than those with dynamic typing. The obvious compromise is against productivity.

To a certain extent, they can be seen as a couple of axes (specificity, generality, performance-productivity). Each of the languages below is somewhere on these spectrums.

Keeping these basic principles in mind, let's look at some of the most popular languages used in data science. What follows is a combination of research and personal experience, but it is by no means definitive! In order of approximate popularity, here is:

What do you want to know?

Released in 1995 as a direct descendant of the old S programming language, R has become more and more powerful. Written in C, Fortran and himself, the project is currently supported by the R Foundation for Statistical Computing.

License

Free!

Advantages

- Excellent range of high quality, domain-specific and open source open source packages. R offers a package for almost every quantitative and statistical application imaginable. This includes neural networks, nonlinear regression, phylogenetic, advanced plotting, and many others.
- The basic installation includes comprehensive and integrated statistical functions and methods. R also handles matrix algebra very well.
- Visualization of data is a major advantage with the use of libraries such as ggplot2.

The inconvenient

- Performance. There are no two ways, R is not a fast language .
- Domain specificity. R is fantastic for statistics and data science. But less for general programming.
- Oddities. R has some unusual features that might appeal to experienced programmers in other languages. For example: indexing from 1, using multiple assignment operators, unconventional data structures.

Verdict - "brilliant for what it was designed for"

R is a powerful language that excels in a wide variety of statistical and data visualization applications, and being open source enables a very active community of contributors. Its recent growth in popularity is testimony to its effectiveness.

Python

What do you want to know?

Guido van Rossum introduced the Python language in 1991. It has since become a versatile language that is extremely popular and widely used by the IT community. The main versions are currently 3.6 and 2.7.

License

Free!

Advantages

- Python is a very popular and very popular general-purpose programming language. It offers a wide range of specially designed modules and community support. Many online services provide a Python API.
- Python is an easy language to learn. The low barrier to entry makes it an ideal first language for beginners in

programming.
- Packages such as pandas, scikit-learn and Tensorflow make Python a solid option for advanced machine learning applications.

The inconvenient

- Security type: Python is a dynamically typed language, which means you have to be very careful. Type errors (such as passing a string as an argument to a method that expects an integer) must be expected from time to time.
- For statistical purposes and specific data analysis, R’s wide range of packages gives it a slight advantage over Python. For general-purpose languages, there are faster and safer alternatives to Python.

Verdict - "excellent to do everything"

Python is a very good choice of language for data science, not just for beginners. Much of the data science process revolves around the ETL (Extraction-Transformation-Loading) process. This makes the generality of Python perfectly adapted. Libraries such as Google's Tensorflow make Python a very interesting language in which to work in machine learning.

SQL

What do you want to know

SQL ('Structured Query Language') defines, manages, and queries relational databases . The language appeared in 1974 and has since undergone many implementations, but the basic principles remain the same.

License

Varies - some implementations are free, other owners

Advantages

- Very effective for querying , updating and manipulating relational databases.
- The declarative syntax makes SQL a language that is often very readable. There is no ambiguity about what we
- SELECT name FROM users WHERE age > 18
- are supposed to do!
- The SQL language is widely used in many applications, making it a very useful language to know. Modules such as SQLAlchemy make it easy to integrate SQL with other languages.

The inconvenient

- The analytical capabilities of SQL are rather limited - beyond aggregation and the sum, count and average data, your options are limited.

- For programmers coming from an imperative context, the declarative syntax of SQL can present a learning curve.
- There are many different implementations of SQL, such as PostgreSQL , SQLite , MariaDB . They are all different enough to make interoperability a puzzle.

Verdict - "timeless and effective"

SQL is more useful as a computer language than as an advanced analytical tool. Yet much of the data science process relies on ETL, and the longevity and efficiency of SQL prove that it is a very useful language for the data specialist to know.

Java

What do you want to know?

Java is an extremely popular, multi-purpose language that runs on the Java Virtual Machine (JVM). It is an abstract computer system that allows transparent portability between platforms. Currently supported by Oracle Corporation .

License

Version 8 - Free! Old versions, owners.

Advantages

- Ubiquity. Many modern systems and applications rely on a Java back-end. The ability to integrate data science methods directly into the existing code base is a powerful tool .
- Strongly typed. Java is a good way to guarantee the security of types. For strategic Big Data applications, this is invaluable.
- Java is a powerful compiled language , for general use. This makes it suitable for efficient ETL production code writing and computationally intensive machine learning algorithms.

The inconvenient

- For ad-hoc analytics and more dedicated statistical applications, Java's verbosity makes it an unlikely first choice. Dynamic typing scripting languages such as R and Python lend themselves to much higher productivity.
- Compared to domain-specific languages such as R, there are not many libraries available for advanced statistical

methods in Java.

Verdict - "a serious contender for data science"

There is a lot to say about learning Java as the language of choice for data science. Many companies will appreciate the ability to seamlessly integrate the Data Science production code directly into their existing code base. You'll find that the performance and security of Java are real benefits. However, you will not have the range of statistics-specific packages available in other languages. That said, there is no doubt that you must take this into account, especially if you already know R / and / or Python.

Scale

What do you want to know?

Developed by Martin Odersky and published in 2004, Scala is a language that runs on the JVM. It is a multi-paradigm language, allowing both an object-oriented approach and a functional approach. Cluster computing framework Apache Spark is written in Scala.

License

Free!

Advantages

- Scala + Spark = High performance cluster computing. Scala is an ideal language choice for those who work with large data sets .
- Multi-paradigmatic: Scala programmers can have the best of both worlds. Functional and object-oriented programming paradigms at their disposal.
- Scala is compiled into Java bytecode and runs on a Java virtual machine. This allows interoperability with the Java language itself, making Scala a very powerful, versatile language, while also being suitable for data science.

The inconvenient

Scale is not an easy language to use if you are just starting out. Your best bet is to download sbt and configure an IDE such as Eclipse or IntelliJ with a specific Scala plug-in.

Syntax and type system are often described as complex . This creates a steep learning curve for those derived from dynamic languages such as Python.

Verdict - "perfect for big data"

When it comes to using cluster computing to work with Big Data, Scala + Spark are fantastic solutions. If you have experience in Java and other static languages, you will also appreciate these features of Scala. However, if your application does not process the data volumes that justify the additional complexity of Scala, your productivity will likely be much higher using other languages such as R or Python.

Julia

What do you want to know?

Released a little over 5 years ago, Julia made a strong impression in the world of digital computing. Its profile has been strengthened by the rapid adoption of several large organizations, including many financial sector companies.

License

Free!

Advantages

- Julia is a JIT compiled language ("just in time"), which allows her to perform well. It also offers the simplicity, dynamic typing and scripting features of an interpreted language such as Python.
- Julia has been specifically designed for digital analysis. He is also capable of general programming.
- Readability. Many language users cite this as a key benefit

The inconvenient

- Maturity. As a new language, some Julia users experienced instability when using packages. But the basic language itself would be stable enough to be used in production.
- Limited packs are another consequence of the youth of the language and the small development community. Unlike R and Python, Julia does not have (for the moment) the choice of packages.

Verdict - "one for the future"

The main problem with Julia is a problem that cannot be faulted. As a newly developed language, it is not as mature and ready for production as its main alternatives, Python and R. But if you want to be patient, you have every reason to pay particular attention to the evolution of language in the years to come.

MATLAB

What do you want to know?

MATLAB is a well-known digital computer language used in academic and industrial settings. It is developed and licensed by MathWorks, a company created in 1984 to commercialize the software.

License

Owner - the price varies depending on your use case

Advantages

- Designed for digital computing. MATLAB is well suited for quantitative applications with sophisticated mathematical requirements such as signal processing, Fourier transformations, formal computation and image processing.
- Visualization of data. MATLAB has great built-in tracing capabilities.
- MATLAB is often taught in many undergraduate courses in quantitative subjects such as physics, engineering and applied mathematics. As a result, it is widely used in these areas.

The inconvenient

- Owner license. Depending on your use case (academic, personal or business), you may need to pay an expensive license . There are free alternatives such as Octave. This is something you should really think about.

- MATLAB is not an obvious choice for general purpose programming.

Verdict - "ideal for math-intensive applications"

The widespread use of MATLAB in a range of quantitative and digital domains in industry and academia makes it a serious option for data science. The clear use case would be when your application or your daily role requires advanced and intensive math features; Indeed, MATLAB has been specially designed for this.

The key here is to understand your usage requirements in terms of generality versus specificity, as well as your preferred development style of performance versus productivity.

CHAPTER 4

How to use Python to analyze SEO data: A reference guide Web Developer Cheat Sheet

Primarily aimed at SEO professionals who are new to programming; it will be useful for those who already have experience working with software or Python, but are looking for easy-to-analyze information to use in projects where data analysis is needed.

Python is pretty easy to learn, and I recommend that you learn it from the official tutorial. I am going to focus on practical SEO applications.

When writing programs in Python, you can choose between Python 2 or Python 3. It is better to write new programs in Python 3, but it is possible that your system may come with Python 2 already installed, especially if you are using a Mac. Please also install Python 3 to be able to use this cheat sheet.

You can check your version of Python using:

```
$ python --version
```

Using virtual environments

When you complete your work, it is important to make sure that other people in the community can reproduce your results. They will need to be able to install the same third-party libraries that you use, often using exactly the same versions.

Python involves creating virtual environments for this.

If your system comes with Python 2, please download and install Python 3 using the Anaconda distribution and follow these steps on the command line .

```
  $ sudo easy_install pip
  $ sudo pip install virtualenv
$ mkdir seowork
$ virtualenv -p python3 SEO work
```

If you are already using Python 3, follow these alternative steps on the command line:

```
  $ mkdir SEO work
  $ python3 -m van SEO work
```

The following steps allow you to work with any version of Python and allow you to use virtual environments.

```
  $ CD SEO work
  $ source bin / Activ
  (SEO work) $ Deactivate
```

When you deactivate the environment, you return to the command line, and the libraries installed in this environment will not work.

Useful data analysis libraries

Whenever I start a data analysis project, I would like to have at least the following libraries installed:

- Matplotlib.
- Requests-html .
- Pandas .
- The Requests .

Most of them are included in the Anaconda distribution. Let's add them to our virtual environment.

```
  (SEO work) $ pip3 install requests
  (SEO work) $ pip3 install matplotlib
  (SEO work) $ pip3 install requests -html
```

```
(SEO work) $ pip3 install pandas
```

You can import them at the beginning of your code as follows:

```
Import requests
From request_html import HTMLSession
Import pandas as pied
```

Since you need more third-party libraries in your programs, you need an easy way to track them and help others easily customize your scripts.

You can export all libraries (and their version numbers) installed in your virtual environment using:

```
(SEO work) $ pip3 freeze> needs.txt
```

When you share this file with your project, any other member of the community can install all the necessary libraries with this simple command in their virtual environment:

```
(Peer-SEO work) $ pip3 install -r needs.txt
```

Using Jupyter Notebooks

When analyzing data, I prefer to use Jupiter Notebooks, as they provide a more convenient environment than the command line. You can check the data you work with and write your programs in a research manner.

```
(SEO work) $ pip3 install chapter
```

Then you can run Jupyter using:

```
(SEO work) $ jupyter notebook
```

You will get a URL to open in your browser.

Alternatively, you can use the Google Colab Laboratory, which is part of GoogleDocs and requires no configuration.

String formatting

You will spend a lot of time in your programs, preparing lines for input in various functions. Sometimes you need to combine data from different sources or convert from one format to another.

Say you want to programmatically retrieve Google Analytics data. You can create an API URL uses Google Analytics Query Explorer and replace the parameter values for the API placeholders using parentheses. For example:

```
  api_uri = "https://www.googleapis.com/analytics/v3/data/ga?ids={gaid}&" \
"Start-date = {start} & end-date = {end} & metrics = {metrics} & "\
" Dimensions = {dimensions} & segment = {segment} & access_token = {token} & "\
" Max-results = {max_results} "
```

{God} is a Google account, that is, "ga: 12345678"

{Start} - start date, that is, "2018-06-01"

{End} is the end date, that is, "2019-06-30".

{Metrics} for a list of numeric parameters, that is, "ga: users", "ga: new Users"

{ Dimensions} is a list of categorical parameters, that is, "ga: landingPagePath", "ga: date"

{ Segment} are marketing segments. For SEO, we want an organic search called “gaid :: - 5”

{ Token} is the most secure access token you get from Google Analytics Query Explorer. It expires in an hour, and you need to run the request again (during authentication) to get a new one.

{max_results} is the maximum number of results to return up to 10,000 rows.

You can define Python variables to store all of these parameters. For example:

```
gaid = "ga: 12345678"
Start = "201 8 -06-01"
End = "201 9 -06-30"
```

This makes it fairly easy to retrieve data from multiple websites or data ranges.

Finally, you can combine the parameters with the API URL to create a valid API request for the call.

```
  api_uri = api_uri.Format (
gaid = gaid,
start = start,
end = end,
metrics = metrics,
 Dimensions = dimensions,
 Segment = segment,
token = token,
max_results = max_results
)
```

Python will replace each placeholder with the corresponding value from the variables that we pass.

String coding

Coding is another common string manipulation technique. Many APIs require strings that are formatted in a certain way.

For example, if one of your parameters is an absolute URL, you must encode it before inserting it into the placeholder API string.

```
  from urllib import parse
url = "https://www.searchenginejournal.com/"
parse.quote (url)
```

The output will look like this: "https% 3A // www.searchenginejournal.com /", which will safely pass to the API request.

Another example: let's say you want to generate heading tags that include an ampersand (&) or angle brackets (<,>). They must be avoided to avoid confusion of HTML parsers.

```
  import html
Title = "SEO <News & Tutorials>"
Html.Escape (title)
```

The output will look like this:

```
  'SEO & lt; News & amp; Tutorials & gt; '
```

Similarly, if you are reading data that is encoded, you can return it back.

```
  Html. Unescape (escaped_title)
```

The output will be read again as the original.

Date formatting

Time series data are very often analyzed, and date and time values can have various formats. Python supports converting dates to strings and vice versa.

For example, after we get the results from the Google Analytics API, we might want to parse dates in datetime objects. This will make them easier to sort or convert from one string format to another.

```
From datetime import datetime
dt = datetime.strptime ('Jan 5 2018 6:33 PM', '% b% d% Y% I:% M% p')
```

Here% b,% d, etc. are directives supported by strptime (used when reading dates) and strftime (used when writing them).

Making API Requests

Now that we know how to format strings and create the correct API requests, let's see how we actually execute such requests.

```
R = queries.get (api_uri)
```

We can check the answer to make sure that we have valid data.

```
Print (r. status_code)
Print (r. Headers ['content-type'])
```

You should see a 200 response code. The content type of most APIs is usually JSON.

When you check redirection chains, you can use the redirection history parameter to see the full chain.

```
Print (r. history)
```

To get the final URL, use:

```
Print (r. url)
```

Data extraction

Most of your work is getting the data you need for analysis. Data will be available from various sources and formats. Let's look at the most common.

Read from JSON

Most APIs will return results in JSON format. We need to analyze the data in this format in Python dictionaries. You can use the standard JSON library for this.

```
Import Jason

json_response = '{"website_name": "Cheat sheet for web developer", "website_url": "https://www.searchenginejournal.com/"}'

parsed_response = Jason.Loads (json_response)
```

Now you can easily access any data you need. For example:

```
Print (parsed_response ["website_name"])
```

The output will be:

```
  " Cheat sheet for web developer "
```

When you use the query library to make API calls, you do not need to do this. The response object provides a convenient property for this.

```
  parsed_response = r.json ()
```

Reading from HTML pages

Most of the data we need for SEO will be on client sites. Despite the fact that there is no shortage of SEO parsers, it is important to learn how to crawl to your needs in order to do things such as automatic grouping by page type.

```
From requests_html import HTMLSession

Session = HTMLSession ()

R = session.get ('https://www.searchenginejournal.com/')
```

You can get all absolute links using this:

```
Print (r.html.absolute_links)
```

Partial output would look like this:

```
{'http://jobs.searchenginejournal.com/',
'https://www.searchenginejournal.com/what-i-learned-about-seo-this-
year/281301/', ...}
```

Next, let's pick some common SEO tags using XPATHs :

Page Title (title)

```
r.Html.xpath ('// title / text ()')
```

Output:

```
  [' Softobzor - Cheat Sheet for Web Developer ']
```

Meta Description

```
r.html.xpath ("// meta [@ name = 'description'] / @ content")
```

Note that I changed the style of quotes from single to double, or I received an encoding error.

Output:

```
  ['Cheat sheet for web developer']
```

rel canonical

```
  r.html.xpath ("// link [@ rel = 'canonical'] / @ href")
```

Output:

```
  [' https://softobzor.com.ua/ ']
```

AMP URL

```
  r.html.xpath ("// link [@ rel = 'amphtml'] / @ href")
```

Softobzor does not have an AMP URL.

Meta Robots

```
  r.html.xpath ("// meta [@ name = 'ROBOTS'] / @ content")
```

Output:

```
  ['NOODP']
```

H1

```
  r.html.xpath ("// h1")
```

The homepage does not have h1.

HREFLANG attribute values

```
  r.html.xpath ("// link [@ rel = 'alternate'] / @ hreflang")
```

Softobzor has no hreflang attributes.

Google Site Check

```
  r.html.xpath ("// meta [@ name = 'google-site-verification'] / @ content")
```

Output:

Softobzor does not have verification attributes , because it is confirmed through analytics.

Javascript rendering

If the page being analyzed requires JavaScript rendering, you only need to add an extra line of code to support this.

```
from requests_html import HTMLSession
session = HTMLSession ()
r = session.get ('https://www.searchenginejournal.com/')
```

r.html.render ()

The first run of render () will take some time, because Chromium will load. Javascript rendering is much slower than without rendering.

Reading from server logs

Google Analytics provides a lot of data, but does not record or show visits to most search engine crawlers. We can get this information directly from the server log files.

Let's see how we can parse server log files using regular expressions in Python.

Import re

log_line = '66 .249.66.1 - - [06 / Jan / 2019: 14: 04: 19 +0200] "GET / HTTP / 1.1" 200 - "" "Mozilla / 5.0 (compatible; Googlebot / 2.1; + http: //www.google.com/bot.html) "'

Regex =' ([(\ d \.)] +) - - \ [(. *?) \] \" (. *?) \ "(\ d +) - \ "(. *?) \" \ "(. *?) \" '

Groups = re.Match (regex, line). Groups ()

Print (groups)

The output breaks down each element of the log entry well:

('66 .249.66.1 ', '06 / Jan / 2019: 14: 04: 19 +0200', 'GET / HTTP / 1.1', '200', '', 'Mozilla / 5.0 (compatible; Googlebot / 2.1; + http://www.google.com/bot.html) ')

You get access to the user agent string in the sixth group, but lists in Python start from scratch, so that's five.

Print (groups [5])

Output:

'Mozilla / 5.0 (compatible; Googlebot / 2.1; + http: //www.google.com/bot.html)'

You can learn about regular expressions in Python here . Be sure to check out the section on greedy and non-greedy expressions. I use non greedy when creating groups.

Googlebot Check

When performing log analysis to understand the behavior of search bots, it is important to exclude any fake queries, since anyone can pretend to be a Google robot by changing the user agent string.

Google provides a simple approach to do it. Let's see how to automate this with Python.

```
Import socket

bot_ip = "66.249.66.1"

Host = socket.Gethostbyaddr (bot_ip)

Print (host [0])
```

You will get crawl-66-249-66-1.googlebot.com

```
  IP = socket.Gethostbyname (host [0])
```

You will get a '66 .249.66.1 ′, which indicates that we have a real Googlebot IP address, since it matches our original IP address, which we extracted from the server log.

Read from a URL

A frequently overlooked source of information is the actual URL of the web page. Most websites and content management systems contain extensive URL information. Let's see how we can extract this.

You can break down URLs into their components using regular expressions, but it’s much simpler and more reliable to use the standard earlobe library for this.

```
 From urllib.Parse import urlparse
```

Url = "https://www.searchenginejournal.com/?s=google&search orderby=relevance&searchfilter=0&search-date-from=January+1%2C+2016&search-date-to=January+7% 2C + 2019 "

```
parsed_url = urlparse (url)

Print (parsed_url)
```

Output:

ParseResult (scheme = 'https', netloc =' www.searchenginejournal.com ', path =' / ', params =' ', query =' s = google & search-orderby = relevance & searchfilter = 0 & search-date-from = January + 1% 2C + 2016 & search-date-to = January + 7% 2C + 2019 ', fragment =' ')

For example, you can easily get the domain name and directory path using:

```
 Print (parsed_url.netloc)

Print (parsed_url. path)
```

This will output what you expect.

We can optionally split the query string to get the URL parameters and their values.

```
  parsed_query = parse_qs (parsed_url.query)

print (parsed_query)
```

You get a Python dictionary as output .

```
  {'s': ['google'],

'search-date-from': ['January 1, 2016'],

'search-date-to': ['January 7, 2019'],

'search-orderby' : ['relevance'],
```

'searchfilter': ['0']}

We can continue and analyze date strings in Python date and time objects, which allow you to perform date operations, for example, calculate the number of days between ranges.

Another common method used in your analysis is to split a part of the path by URL into “/” to get the parts. This is easy to do with the split function.

```
  URL = " https://softobzor.com.ua/category/laravel/ "
parsed_url = urlparse (URL)
parsed_url.path.split ("/")
```

The output will be:

```
  ['', 'category', 'digital-experience', '']
```

When you separate URL paths this way, you can use this to group a large group of URLs into their top directories.

For example, you can find all products and all categories on an e-commerce website if the URL structure allows it.

Perform Basic Analysis

You will spend most of your time getting the data in the right format for analysis. Part of the analysis is relatively simple if you know the right questions.

Let's start by loading the Screaming Frog scan into the pandas dataframe.

```
 Import pandas as pied
df = pd.DataFrame (pd.read_csv ('internal_all.csv', header = 1, parse_dates = ['Last Modified']))
print (df.dtypes)
```

The output shows all the columns available in the Screaming Frog file and their Python types. Using pandas, we analyze the last modified column in a Python date and time object.

Let's do some analysis examples.

Top Level Directory, Grouping

First, let's create a new column with a page type by dividing the URL path and extracting the first directory name.

```
 Df ['Page Type'] = df ['Address']. Apply (lambda x: urlparse (x). path. Split ("/") [1])
aggregated_df = df [[' Page Type ',' Word Count ']]. Grubby ([' Page Type ']). Agg (' sum ')
Print (aggregated_df)
```

After creating the “Page Type” column, we group all pages by type and total number of words. The output partitioning looks like this:

```
SEO-guide 736
SEO-internal-links-best-practices 2012
SEO-keyword-audit 2104
SEO-risks 2435
Seo-tools 588
SEO-trends 3448
SEO-trends-2019 2676
SEO-value 1528
```

Grouping by status code

```
status_code_df = df [['Status Code', 'Address']]. groupby (['Status Code']). agg ('count')
print (status_code_df)
```

```
200 218
```

```
301 6
302 5
```

Enumeration of temporary redirects

```
  temp_redirects_df = df [df ['Status Code'] == 302] ['Address']
print (temp_redirects_df)
```

```
50 https: //www.searchenginejournal.com/wp-content ...
116 https: //www.searchenginejournal.com/wp-content ...
128 https://www.searchenginejournal.com/feed
154 https: //www.searchenginejournal.com/wp-content ...
206 https: //www.searchenginejournal.com/wp-content ...
```

Listless Pages

```
  no_content_df = df [(df ['Status Code'] == 200) & (df ['Word Count']
== 0)] [['Address', 'Word Count']]
```

```
7 https: //www.searchenginejournal.com/author/cor ... 0
9 https: //www.searchenginejournal.com/author/vic ... 0
66 https: //www.searchenginejournal.com/author/ada ... 0
70 https: //www.searchenginejournal.com/author/ron ... 0
```

Publishing activity

Let's see what time of the day most articles are published in SEJ.

```
lastmod = pd.DatetimeIndex (df ['Last Modified'])
writing_activity_df = df.groupby ([lastmod.hour]) ['Address']. count ()
```

```
0.0 19
1.0 55
```

```
2.0 29
10.0 10
11.0 54
18.0 7
19.0 31
20.0 9
21.0 1
22.0 3
```

Interestingly, during normal business hours there are not many changes.

We can build it directly from pandas.

```
writing_activity_df.plot.bar ()
```

A bar chart showing the time of day of publication of articles. A bar chart was generated using Python 3.

Saving and exporting results

Now we move on to the easy part - saving the results of our hard work.

Saving in Excel

```
writer = pd.ExcelWriter (no_content.xlsx ')
no_content_df.to_excel (writer,' Results')
writer.save ()
```

Save to CSV

```
  temporary_redirects_df.to_csv ('temporary_redirects.csv')
```

Machine Learning Algorithms

The scikit-learn library implements a host of machine learning algorithms.

Some machine learning algorithms implemented in scikit-learn:

Basic methods of classes that implement machine learning algorithms

All algorithms are executed in the form of classes possessing at least the following methods:

Note that the parameters of learning algorithms can be set both in the class constructor and using the method set_params(**params).

The table of the presence / absence of methods for the basic machine learning algorithms:

K NN - Nearest Neighbor Method

Let's start with one of the simplest machine learning algorithms - the k method nearest neighbors (kNN).

For a new object, the algorithm looks for k in the training set the closest object and relates the new object to the class to which most of them belong.

Number of neighbors kmatches the parameter n_neighbors. Default n_neighbors = 5.

First, we will train the model:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier ()
knn . fit ( X_train , y_train )
KNeighborsClassifier (algorithm = 'auto', leaf_size = 30, metric =
'minkowski',
```

metric_params = None, n_neighbors = 5, p = 2, weights = 'uniform')

After the model is trained, we can predict the value of the target attribute by the input characteristics for new objects. This is done using the method predict.

We are interested in the quality of the constructed model, therefore, we will predict the value of the output feature on the data for which it is known: on the training and (more importantly) test samples:

```
y_train_predict  =  knn . predict ( X_train )

y_test_predict  =  knn . predict ( X_test )

err_train  =  np . mean ( y_train  ! =  y_train_predict )

err_test   =  np . mean ( y_test   ! =  y_test_predict )

print  err_train ,  err_test
```

```
0.146997929607 0.169082125604
```

err_trainand err_test- these are errors in the training and test samples. As we can see, they amounted to 14.7% and 16.9% .

For us, the error on the test sample is more important, since we must be able to predict the correct (if possible) value on new objects that were not available during training.

Let's try to reduce the test error by varying the parameters of the method.

The main parameter of method knearest neighbors is k.

The search for optimal parameter values can be accomplished with a class GridSearchCV- search for the best set of parameters that minimizes the error cross-checking (cross-validation). By default, 3x cross control is considered.

For example, find the best value of kamong the values [1, 3, 5, 7, 10, 15]:

```
from  sklearn.grid_search  import  GridSearchCV
n_neighbors_array  =  [ 1 ,  3 ,  5 ,  7 ,  10 ,  15 ]
knn  =  KNeighborsClassifier ()
grid   =   GridSearchCV  (  knn  ,   param_grid  =  {  'n_neighbors'  :
n_neighbors_array })
grid . fit ( X_train ,  y_train )

best_cv_err  =  1  -  grid . best_score_
best_n_neighbors  =  grid . best_estimator_ . n_neighbors
print  best_cv_err ,  best_n_neighbors
```

```
0.207039337474 7
```

As an optimal method, I chose the value of kequal to 7. The error of the cross control was 20.7%, which is even more than the error in the test sample for the 5 nearest neighbors. This may be to the fact that not all data is used to build models in the framework of the cross-control scheme.

Let us check what the errors in the training and test samples are equal for this parameter value

```
knn  =  KNeighborsClassifier ( n_neighbors = best_n_neighbors )
knn . fit ( X_train ,  y_train )

err_train  =  np . mean ( y_train  ! =  knn . predict ( X_train ))
err_test   =  np . mean ( y_test   ! =  knn . predict ( X_test ))
print  err_train ,  err_test
```

```
0.151138716356 0.164251207729
```

As we see, the method of nearest neighbors in this problem gives not very satisfactory results.

SVC - support vector machine

The following method, which we will try - support vector machine (SVM - support vector machine or SVC - support vector classifier) - immediately leads to more optimistic results.

Already with the default parameter value (in particular, the kernel is radial rbf) we get a lower error in the training set:

```
from  sklearn.svm  import  SVC

svc  =  SVC ()

svc . fit ( X_train ,  y_train )


err_train  =  np . mean ( y_train  ! =  svc . predict ( X_train ))

err_test   =  np . mean ( y_test   ! =  svc . predict ( X_test ))

print  err_train ,  err_test
```

```
0.144927536232 0.130434782609
```

So, the test sample received an error of 13% .

Using the selection of parameters, we will try to reduce it even further.

Radial core

First, let's try to find the best parameter values for the radial core.

```
from  sklearn.grid_search  import  GridSearchCV
C_array  =  np . logspace ( - 3 ,  3 ,  num = 7 )
gamma_array  =  np . logspace ( - 5 ,  2 ,  num = 8 )
svc  =  SVC ( kernel = 'rbf' )
grid  =  GridSearchCV ( svc ,  param_grid = { 'C' : C_array ,  'gamma'
:  gamma_array })
grid . fit ( X_train ,  y_train )
print  'CV error =' ,  1  -  grid . best_score_
print  'best C =' ,  grid . best_estimator_ . C
print  'best gamma =' ,  grid . best_estimator_ . gamma
```

```
CV error = 0.138716356108
best C = 1.0
best gamma = 0.01
```

Got a cross-control error of 13.9%.

Let's see what the error in the test sample is for the found values of the algorithm parameters:

```
svc  =  SVC ( kernel = 'rbf' ,  C = grid . best_estimator_ . C ,  gamma
= grid . best_estimator_ . gamma )
svc . fit ( X_train ,  y_train )

err_train  =  np . mean ( y_train  ! =  svc . predict ( X_train ))
err_test   =  np . mean ( y_test   ! =  svc . predict ( X_test ))
print  err_train ,  err_test
```

```
0.134575569358 0.111111111111
```

The error in the test sample is 11.1% . Noticeably better than kNN!

Line core

Now consider the linear core.

```
from  sklearn.grid_search  import  GridSearchCV
C_array  =  np . logspace ( - 3 ,  3 ,  num = 7 )
svc  =  SVC ( kernel = 'linear' )
grid  =  GridSearchCV ( svc ,  param_grid = { 'C' :  C_array })
grid . fit ( X_train ,  y_train )
print  'CV error =' , 1  -  grid . best_score_
print  'best C =' ,  grid . best_estimator_ . C
```

```
CV error = 0.151138716356
best C = 0.1
```

Got a cross-control error of 15.1%.

Let's see what the error in the test sample is for the found values of the algorithm parameters:

```
svc  =  SVC ( kernel = 'linear' ,  C = grid . best_estimator_ . C )
svc . fit ( X_train ,  y_train )

err_train  =  np . mean ( y_train  ! =  svc . predict ( X_train ))
err_test   =  np . mean ( y_test   ! =  svc . predict ( X_test ))
print  err_train ,  err_test
```

```
0.151138716356 0.125603864734
```

The error in the test sample was 12.6% .

Polynomial kernel

Let's also try the polynomial core:

```
from  sklearn.grid_search  import  GridSearchCV

C_array  =  np . logspace ( - 5 ,  2 ,  num = 8 )

gamma_array  =  np . logspace ( - 5 ,  2 ,  num = 8 )

degree_array  =  [ 2 ,  3 ,  4 ]

svc  =  SVC ( kernel = 'poly' )

grid  =  GridSearchCV ( svc ,  param_grid= { 'C' :  C_array ,  'gamma'
:  gamma_array ,  'degree' :  degree_array })

grid . fit ( X_train ,  y_train )

print  'CV error =' ,  1  -  grid . best_score_

print  'best C =' ,  grid . best_estimator_ . C

best_estimator_   .   gamma   print   'best   degree   ='   ,   grid   .
best_estimator_ . print  'best gamma =' ,  grid .
  degree
```

```
CV error = 0.138716356108

best C = 0.0001

best gamma = 10.0

best degree = 2
```

Got a cross-control error of 13.9%.

Let's see what the error in the test sample is for the found values of the algorithm parameters:

```
svc  =  SVC ( kernel = 'poly' ,  C = grid . best_estimator_ . C ,
          gamma = grid . best_estimator_ . gamma ,  degree = grid .
best_estimator_ . degree )
```

```
svc . fit ( X_train ,  y_train )

err_train  =  np . mean ( y_train  ! =  svc . predict ( X_train ))
err_test   =  np . mean ( y_test   ! =  svc . predict ( X_test ))
print  err_train ,  err_test

0.0973084886128 0.12077294686
```

The error in the test sample was 12.1% .

Mathematics for Data Analysis

we will show you how to study math for data analysis and machine learning without long and expensive courses.

The number of mathematical problems in daily work will greatly depend on the position that you will be occupied as an analyst. Keep reading to find out what concepts you need to master in order to succeed for your goals.

Prerequisites: Basic Python Skills

To complete this tutorial, you will need at least basic Python programming skills. We will study mathematics at an applied, practical level.

Mathematics required for data analysis

First of all, every data analyst needs to know Statistics. For it, we have a separate guide, " Statistics for data analysis. "

And what else? There are nuances here: it all depends on how diverse the tasks you work with will be.

In practice, you will most often use ready-made solutions, especially at the initial stage. For data analysis, many libraries (packages) have been written in many programming languages.

However, during the interview you can be tested for knowledge of the basics of linear algebra and multidimensional analysis . That's because at some point your team may need to adapt the solution for the technology stack or expand its functionality. To do this, you will need to understand how machine learning algorithms work.

Development tasks

Some tasks require much more extensive knowledge. For example, if you need to translate a mathematical idea from paper into working code. Or if you are faced with non-trivial tasks.

In other words, you will develop algorithms from scratch. In such cases, knowledge of linear algebra and multivariate analysis is a must.

The Right Way to Learn Math for Data Analysis

We will study the problems of linear algebra, applying them in real algorithms.

You need to familiarize yourself or refresh the underlying theory. No need to read the whole textbook, focus on key concepts.

Here are 3 steps to learning the math needed for analysis and machine learning:

- Linear algebra for data analysis : matrices and eigenvectors
- Mathematical Analysis : Derivatives and Gradients
- Gradient descent: creating a simple neural grid from scratch

Step 1: Linear Algebra for Data Analysis

Many machine learning concepts are related to linear algebra. For instance, for the Principal Component System, you need to know the eigenvectors, and for the regression, matrix multiplication is required.

In addition, machine learning often works with high dimensional data (data with various variables). This data kind is best represented by matrices.

Here are few of the best free resources we've found for learning linear algebra for data analysis:

Khan Academy provides short practical lessons in linear algebra. They cover the most important topics.

2. MIT OpenCourseWare offers a linear algebra course. All video lectures and teaching materials are included.

Additionally:

3. Linear Algebra Review for the Machine Learning (Video Series) These are additional videos that summarize the concepts of linear algebra for a machine learning course. The entire series of 6 parts can be viewed in less than 1 hour. Recommended if you only need to refresh your knowledge of linear algebra.

4. The Matrix Cookbook (PDF) - An excellent reference resource for matrix algebra.

Step 2: Mathematical Analysis

Mathematical analysis underlies many machine learning algorithms. Derivatives and gradients are needed for optimization problems.

For instance, one of the most common optimization methods is gradient descent.

Here are some of the best free resources we have found to study math analysis:

At the Khan Academy has a course with a short, practical lessons on data analysis. It is enough to get acquainted with the basic concepts.

MIT OpenCourseWare offers a course in mathematical analysis. All video lectures and teaching materials are included.

Additionally:

3. Multivariable Calculus Review (Video) - This is a short course of mathematical analysis in the format of solving practical problems. It is recommended if you have studied the subject before and just want to refresh your knowledge.

Step 3: a simple neural network from scratch

This is the most interesting part of the entire guide.

One of the right ways to learn mathematics in analysis and machine learning is to build a simple neural network from scratch.

You will use linear algebra to represent the network and the mathematical analysis to optimize it. You will create a gradient descent from the scratch.

Do not worry about the nuances of neural networks. This is nice if you just follow the instructions and write the code. We will take a closer look at machine learning in another guide. And here we look more at mathematical practice.

Follow the step-by-step instructions and learn the theory. In the final, you will have a cool project that you can add to your resume.

Here are some good walkthroughs:

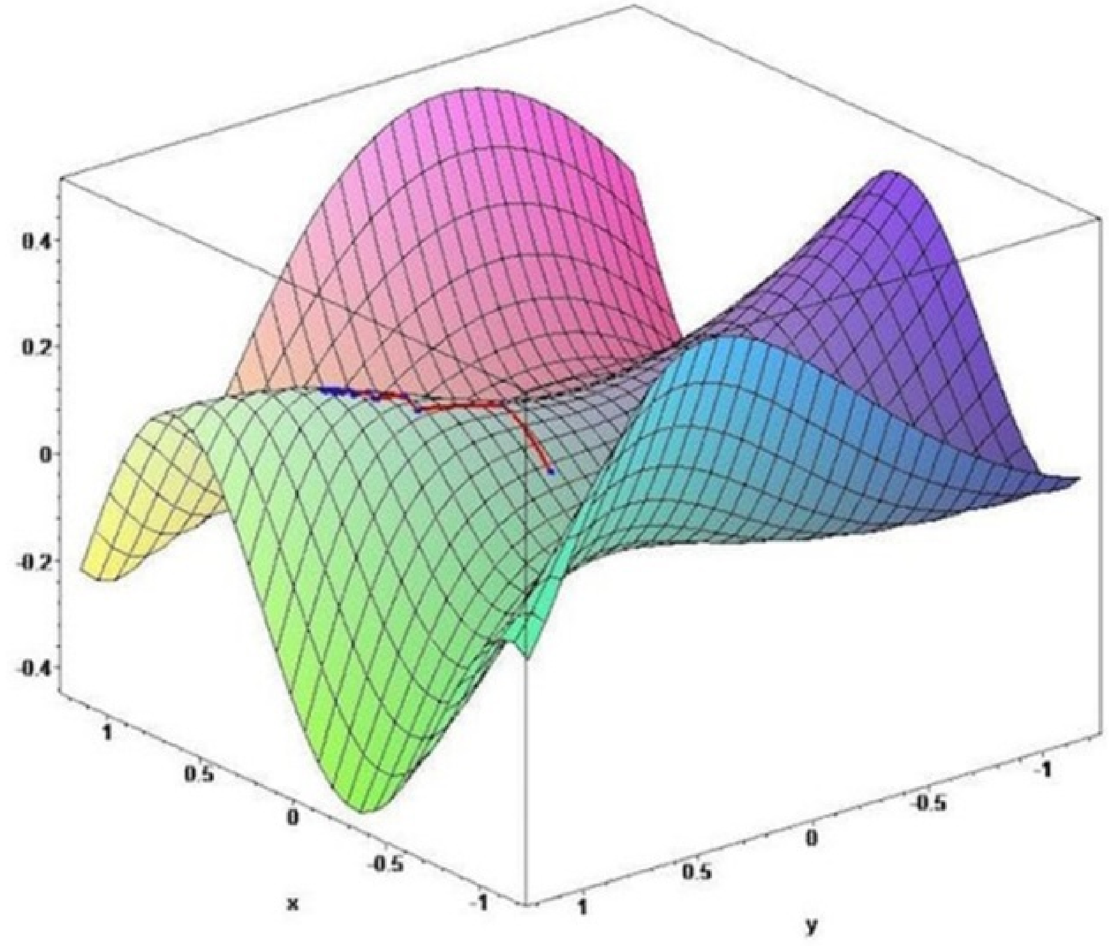
Neural Network in Python - This is a great tutorial in which you can build a simple neural grid from start to finish. In it you will find useful illustrations and learn how gradient descent fits.

Neural Nets to Recognize Handwritten Digits Recommended This Resource! This is a free online book that introduces you to the use of neural networks. After reading, you understand the principles of operation of algorithms at an intuitive level, and this is the most detailed textbook on this list.

Implementing a Neural Network from Scratch A shorter tutorial that will also help you learn neural networks step by step.

Total: Data Analyst is a unique and very interesting specialty. In your work, you will often encounter problems for the successful solution of which you need, on the one hand, to study the theory, and on the other hand, to begin to practice the use of various approaches and models.

The data science community is constantly growing due to the entry of people from completely different fields. They are united by an interest in solving complex and non-standard problems. Our company also contributes to the development of the community, we publish articles, as well as vacancies and events in our group . For us, the most important thing is the willingness to learn and learn something new.



0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
x
1
0.5
0
-0.5
-1
y

CHAPTER 5

Why python?

One of the difficulties that developers face when working with joint processing in the Python programming language (in particular, CPython, the reference implementation of Python written in C) is GIL. The GIL mechanism is a deadlock mechanism that protects access to Python objects, preventing the simultaneous execution of Python byte codes in multiple threads. This locking is necessary mainly because CPython's memory management is not thread-safe. CPython uses a reference counter to implement its memory management. This results in the fact that multiple threads can access and execute Python code at the same time; this situation is undesirable and it can cause incorrect data processing and we say that this type of memory management is not oriented towards multi-threaded management. To solve this problem, there is a GIL, as its name indicates, a certain lock that allows only one thread to access the code and Python objects. However, this also means that in order to implement programs with multiple threads in CPython, developers must be aware of the GIL and circumvent it. It is for this reason that many who have problems implementing co-processing systems in Python.

So why do we even use Python for collaborative processing? Even though GIL does not allow CPython programs with multiple threads to get all the advantages of multiprocessor systems in certain situations, most of the operations with blocking or long-term execution, such as input / output, image processing, as well as grinding numbers in NumPy, occur outside the existing one. Gil Thus, the GIL itself becomes a potential bottleneck only for programs with many threads that spend significant time inside the GIL. As you will see in subsequent chapters, multithreading is just some kind of co-processing programming, and although the GIL does make certain calls for multithreading CPython programs that allow more than one thread to access shared resources, other forms of parallel programming do not have this problem. For example, applications with many processes that do not share any common resources between processes, such as input / output, image processing, or grinding NumPy numbers, can work seamlessly with GIL. We will discuss GIL itself and its place in the Python ecosystem in more detail in Chapter 15, Global Interpreter Lock .

In addition, Python has gained increasing popularity in the programming community. Thanks to user-friendly syntax and general readability, more and more people believe that it is relatively easy to use Python in their development, whether it’s a beginner learning a new programming language, users with an average level of training in searching for available modern Python functionality, or experienced Programmers using Python to solve complex problems. There are estimates that Python code development can be up to 10 times faster than C / C ++ coding.

A large number of developers using Python have come about as a result of a powerful, still growing community. Every day, Python libraries and packages are developed and released, supplying various tasks and technologies. Currently, Python supports an incredibly wide range of programming - namely, software development, GUI workstations, video game design, web and Internet development, as well as scientific and numerical calculations. In recent years, Python has also grown as one of the top tools in data science, Big Data, and machine learning, competing with a long-term player in the field, R.

The huge number of development tools available in Python has prompted an increasing number of developers to start programming in Python, which makes Python even more popular and easy to use; I call it the vicious circle of Python . David Robinson, DataCamp's research supervisor, wrote a blog post about Python's incredible growth and called it the most popular programming language.

However, Python is slow, at least slower than other popular programming languages. This is because Python is a dynamic typing language that interprets in which values are not stored in tight buffers, but in scattered objects. This is a direct result of the readability and usability of Python. Fortunately, there are various options for making your Python program run faster, with concurrency being one of the most difficult among them; and this is exactly what we are going to master in this book.

Customizing Your Python Environment

Before we move on, let's go through a number of descriptions about how to configure the necessary tools that you will use in this book. In particular, we will discuss the process of obtaining a Python distribution for your system and a suitable development environment, as well as how to download the code that is used in the examples contained in all chapters of this book.

General installation

Let's look at the actual process of obtaining a certain Python distribution for your system and the corresponding development environment:

All developers can get their own Pyton distribution from https: / / www.python. Org / downloads / .

Even though both versions of Python 2 and Python 3 are supported and maintained, throughout this book we will use Python 3.

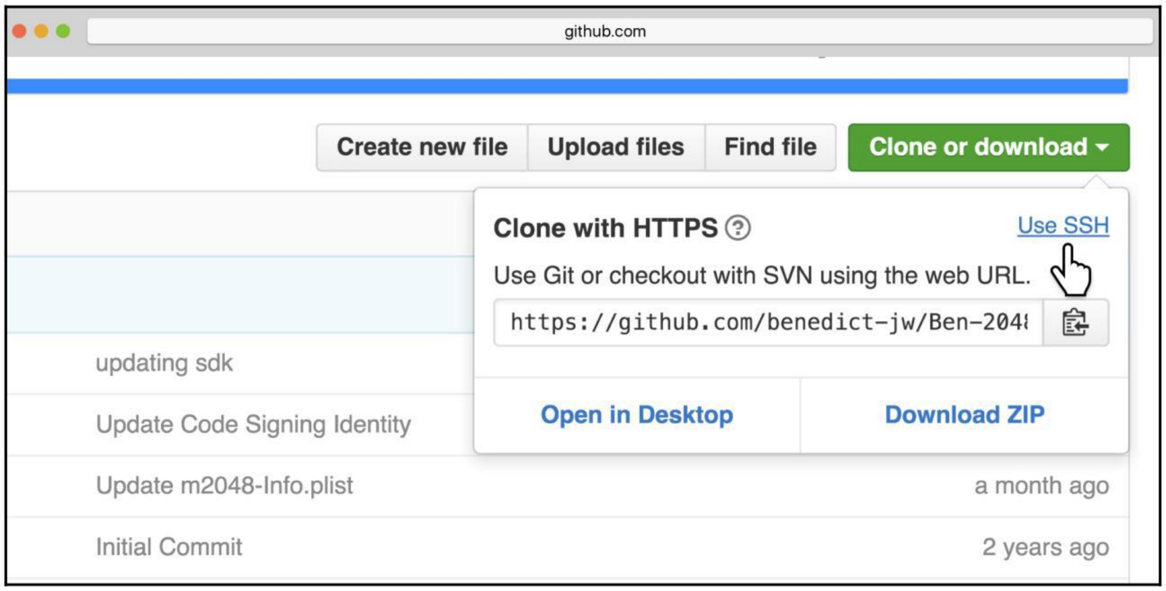
For this book, a flexible option would be an IDE (integrated development environment). Although it is technically possible to develop Python applications with a minimal text editor such as Notepad or TextEdit, it is usually much easier to read and write code using an IDE purposefully developed for Python. They include: IDLE , PyCharm , Sublime Text, and Atom .

Sample Code Unloading

To get the code that has been used throughout this book, you can download the repository from GitHub, which contains all the examples and project code described in the book:

To get started, visit https: / / github.com/ PacktPublishing / Mastering- Concurrency-in-Python .

To upload the corresponding repository, simply click on the Clone or the download button in the upper right corner of your window. Select Download ZIP to download the necessary compressed repository to your computer:



Use Git or checkout with SVN using the web URL.

```
https://github.com/benedict-jw/Ben-204
```

updating sdk
Update Code Signing Identity
Update m2048-Info.plist a month ago
Initial Commit 2 years ago

To download the necessary repository, click on Download ZIP

Unzip the downloaded file to create the folder we are looking for. This folder should have a name Mastering-Concurrency-in-Python.

Inside the resulting folder will be located separate folders, entitled ChapterXX, indicating the chapter that contains the corresponding code in this folder

findings

Now you have learned the basics of the concepts of joint and parallel programming. All this relates to the design and structural programming of commands and instructions so that different sections of your program can be executed in some kind of effective way, and at the same time share the same resources. Since when some commands and instructions are executed at the same time, time is saved, parallel programming provides a significant improvement in the execution time of the program when compared with traditional sequential programming.

However, when designing a parallel program, various factors should be taken into account. While there are special tasks that can easily be divided into independent sections that can be executed in parallel (stunning parallel tasks), others require different types of coordination between the existing program commands so that such shared resources can be applied correctly and efficiently. There are also tasks with sequential processing that is intrinsically intrinsic to them, in which no joint processing and parallelism can be applied to obtain program acceleration. You should be aware of such fundamental differences between these tasks so that you can carefully design your co-processing programs.

In recent times, a paradigm shift has occurred that facilitated the implementation of joint processing in most aspects of the existing programming universe. Now joint processing can be found everywhere: desktop and mobile applications, video games, web and Internet development, AI and so on. Collaborative processing is still growing and is expected to continue to grow in the future. Therefore, it is extremely important for any experienced programmer to understand the joint processing and related concepts, as well as to know how to integrate these concepts into their applications.

Python, on the other hand, is one of the most (if not the most) popular programming languages. It provides options in most programming sub-sectors. Combining collaborative processing and Python in this way is one of the most important topics to learn and master in programming.

CHAPTER 6

Working with Threads in Python

An Advanced Introduction to Joint and Parallel Programming, you saw an example of threads used in joint processing and parallel programming. In this chapter, you will get some introduction to the actual formal definition of a stream, as well as a module threading from Python. . We will consider a number of ways to work with threads in some Python program, including Action such as creating new threads, synchronizing threads, and working with multi-threaded queues with priorities, and with specific examples. We will also discuss the basic concept of blocking when synchronizing threads, and we will also implement some kind of multithreaded application with blocking to better understand the benefits of thread synchronization.

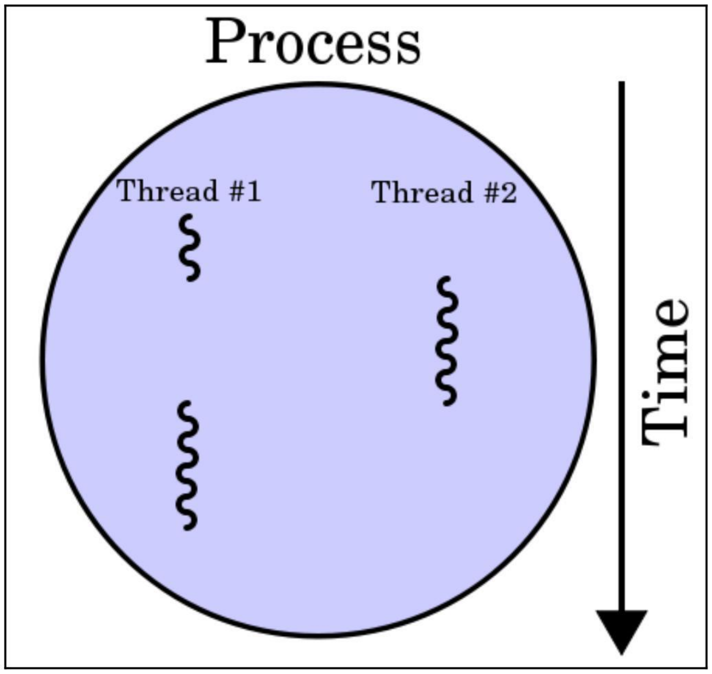
Low concept

In the field of computer science, a certain flow of execution is the smallest element of programming commands (code) that some scheduler (usually as part of the operating system) can process and manage. Depending on the specific operating system, the implementation of threads and processes (which we will discuss in our next chapter) varies, but a thread is usually a certain element (component) in any process.

Mapping Threads and Processes

Within the same process, more than one thread can be implemented, most likely running jointly and accessing / sharing the same resources. Streams in the same process share all subsequent instructions (their code) and context (those values that their variables refer to at any given moment).

The main key difference between these two concepts is that a certain thread is usually a component of any process. Thus, one process can contain many threads that can be executed simultaneously. The threads also usually allow the sharing of resources, such as memory and data, while processes rarely do this. In short, a certain flow is some kind of independent component the calculations, which is similar to a process, however, a flow within a certain process can share the address space and, thus, the data of such a process itself:



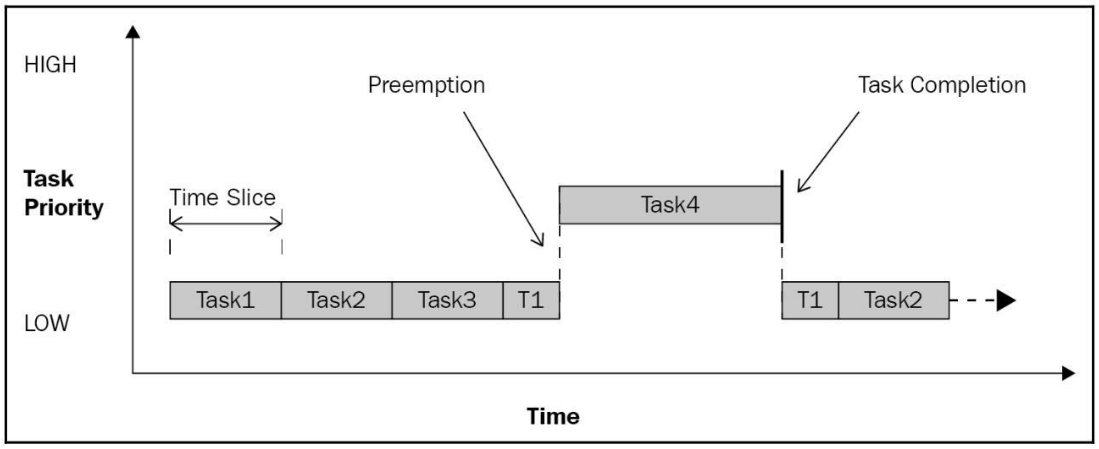
A process with two threads running on the same processor

The first mention of the use of threads for a variable number of tasks in OS / 360 multi-programming, which is a discontinued batch processing system dating back to 1967 after its development by IBM. At that time, developers called streams tasks, and the term stream became popular later and is attributed to Victor A. Vysotsky, a mathematician and research scientist in the field of computing, who was the founding director of the Digital Research Laboratory in Cambridge.

Multithreading

In computer science, a single stream is similar to traditional sequential processing, executing a certain separate command at a certain point in time. Multithreading implements more than one thread existing and executing in some separate process, and at the same time. By allowing multiple threads to have access and shared resources / contexts and execute independently, this technique can help applications accelerate their independent tasks.

Multithreading can initially be achieved in two ways. In systems with a single processor, multithreading is usually implemented by dividing the time , a technique that allows an existing CPU to switch between different software running in different threads. When dividing the time, the CPU itself switches its execution so quickly and so often that users usually perceive that their software is running in parallel (for example, when you open two different programs at the same time in a computer with a single processor):



Some example of time sharing with the name of carousel planning

In contrast to systems with a single processor, systems with many processors or cores can easily implement multithreading, moreover, executing each thread in a separate processor or core and at the same time. In addition, time sharing is also an option, since such systems with multiple processors and multiple cores can have only one processor / single core for switching between tasks - although this is usually not the most practical technique.

Multithreaded applications have a number of advantages over conventional serial applications; some of them are:

- Faster execution time: One of the main advantages of multi-threading collaboration is the acceleration achieved. Separate threads in the same program can be executed jointly or in parallel, when they are sufficiently independent from each other.
- Speed of response: A certain program with a single thread at a time can process only one piece of input; thus, if its main thread of execution is blocked in some task with a long execution time (for example, some part of the input, which requires intensive calculations and processing), the whole program will not be able to continue other input and, therefore, will seem frozen. By using separate threads to perform the calculations and remaining executable to receive another user's input at the same time, a multi-threaded program can provide much better responsiveness.
- Efficiency in resource consumption: As mentioned earlier, many threads within the same process can share the same resources and access them. As a result, multi-threaded programs can serve and process many client requests for data for joint processing, using much less resources than would be required when using single-threaded or multi-process programs. It also leads to faster interaction between threads.

At the same time, multithreaded programs also have their own drawbacks, namely:

- Crash: Even though a certain process can contain many threads, a single invalid operation within the same thread may adversely affect the processing of all other threads in this process and may result in the entire program crashing.
- Synchronization: Even though sharing the same resources may be some kind of advantage over regular sequential

programming or programs with many processes, such sharing of resources also requires careful consideration of the details so that the joint data is calculated correctly and processed correctly. Intuitively obscure problems that may be caused by careless coordination of flows include deadlocks, freezes and a state of competition, each of which will be discussed in subsequent chapters.

Some Python example

To demonstrate the very concept of starting multiple threads in the same process, let's look at some quick Python example. Let's look at the file below Chapter03/my_thread.py:

```
# Chapter03/my_thread.py
import threading
import time

class MyThread(threading.Thread):
    def __init__(self, name, delay):
        threading.Thread.__init__(self)
        self.name = name
        self.delay = delay

    def run(self):
        print('Starting thread %s.' % self.name)
        thread_count_down(self.name, self.delay)
        print('Finished thread %s.' % self.name)

def thread_count_down(name, delay):
    counter = 5

    while counter:
        time.sleep(delay )
        print('Thread %s counting down: %i...' % (name, counter))
        counter -= 1
```

In this file, we use the appropriate module threading from Python as the necessary basis for our class MyThread. The smeared object from this class has a certain name parameter delay. An existing function run(), which is called as soon as a new thread is initialized and started, prints a start message and, in turn, calls the corresponding function thread_count_down(). This function counts down 5to 0, and falls asleep between iterations for several seconds, which are determined by the defined delay parameter.

The main point in this example is to show the existing nature of joint processing by simultaneously launching more than one object from our class MyThread. We know that as soon as each thread starts, a countdown based on time also starts. In a traditional sequential program, a separate countdown will be executed separately, in order (that is, some new countdown will not start until the current one is completed). As you will find, all separate countdowns for separate streams are executed together.

Let's look at the following file Chapter03/example1.py:

```
# Chapter03/example1.py

from my_thread import MyThread

thread1 = MyThread('A', 0.5)
thread2 = MyThread('B', 0.5)

thread1.start()
thread2.start()

thread1.join()
thread2.join()
```

```
print('Finished.')
```

Here we performed the initialization and launched two threads together, and each of them has seconds as a parameter . Using your Python interpreter, run this script. You should get the following output:delay 0.5

```
> python example1.py
Starting thread A.
Starting thread B.
Thread A counting down: 5...
Thread B counting down: 5...
Thread B counting down: 4...
Thread A counting down: 4...
Thread B counting down: 3...
Thread A counting down: 3...
Thread B counting down: 2...
Thread A counting down: 2...
Thread B counting down: 1...
Thread A counting down: 1...
Finished thread B.
Finished thread A.
Finished.
```

Exactly as expected, the conclusion we received tells us that our two countdowns for existing flows were executed together; Instead of completing our very first countdown stream and then launching the second available countdown stream, our program executed these two countdowns at almost the same time. Without the inclusion of any overhead and various announcements, such a technique for organizing threads can almost double the speed improvement for our previous program.

There is one additional point that should be taken into account in our previous conclusion. After the very first countdown at number, 5we can notice that stream B actually overtook stream A in execution, even though we know that stream A was initialized and started before stream B. This change actually allowed stream B to complete earlier than stream A. This phenomenon is a direct result of collaboration with multithreading; since these two threads were initialized and started almost simultaneously, there is a high probability that one thread will be ahead of the other at execution.

If you run this script many times, it is likely that you will get a different output in terms of the execution order and completion of your countdowns. The following are two snippets of output that I got by running this script over and over. The very first conclusion shows a certain uniformity and invariance of the order of execution and completion, in which these two countdowns were executed hand to hand. The second one shows a variant in which stream A executed slightly faster than stream B; even ending before thread B counted the number 1. This output option will further illustrate the fact that our threads were perceived and executed by Python the same way.

The following code shows one possible output of our program:

```
> python example1.py

Starting thread A.

Starting thread B.

Thread A counting down: 5...

Thread B counting down: 5...

Thread A counting down: 4...

Thread B counting down: 4...
Thread A counting down: 3...
Thread B counting down: 3...
Thread A counting down: 2...
Thread B counting down: 2...
Thread A counting down: 1...
Thread B counting down: 1...
Finished thread A.
Finished thread B.
Finished.
```

And here is another possible conclusion:

```
> python example1.py
Starting thread A.
Starting thread B.
Thread A counting down: 5...
Thread B counting down: 5...
Thread A counting down: 4...
Thread B counting down: 4...
Thread A counting down: 3...
Thread B counting down: 3...
Thread A counting down: 2...
Thread B counting down: 2...
Thread A counting down: 1...
Finished thread A.
Thread B counting down: 1...
Finished thread B.
Finished.
```

Overview of the available threading module

There are various alternatives when it comes to implementing multi-threaded programs in Python. One of the most common methods to work with threads in Python is to use a module threading. Before we dive into the existing application model and its syntax, let's first examine the model itself thread, which was originally the most basic thread-based development module in Python.

Thread module in Python 2

Before the module became popular threading, the primary thread-based development module was thread. If you are using the older versions of Python 2, you can use this module as-is. However, according to the module’s documentation page, this module was actually renamed to Python 3 _thread.

For those readers who have worked with this module thread to build multi-threaded applications and are considering porting their code from Python 2 to Python 3, the 2to3 tool may be some kind of solution. The 2to3 tool handles most of the revealed incompatibilities between different versions of Python, while parsing the resulting source code and traversing this source tree to convert Python 2.x code to Python 3.x code. Another trick to achieve the conversion is to change your imported code from an import thread to import _thread as a thread in your Python program.

The main property of this module thread is its speed and the sufficiency of the method of creating a new thread for the execution of functions: the corresponding function thread.start_new_thread(). In addition, simple locking objects are provided for synchronization purposes (for example, mutual exceptions - mutexes and semaphores - semaphores).

Python threading module 3

The old module was thread considered by Python developers as having lost its relevance for a long time, mainly because of its functions at a rather low level and limitation of use. On the other hand, the module threading is built on top of the existing module thread, providing simpler ways to work with threads through powerful top-level APIs. Python users actually approved the use of this new module threading instead of the module thread in their programs.

In addition, the module itself threadconsiders each thread as a function; when called a thread.start_new_thread(), it actually gets some kind of separate function as its main argument to spawn a new thread. Nevertheless, the new module is threadingdesigned to provide a user-friendly interface for those who come from the paradigm of object-oriented software, treating each created stream as an object.

In addition to all the functionality for working with streams that the module provides thread, the new module threadingsupports a number of additional methods, such as:

- Threading.Active Count (): This function returns the total number of currently active stream objects in this program.
- Threading.CurrentThread (): This function returns the total number of stream objects in this current stream controlled by the call side.
- Threading.Enumerate (): The function returns a list of all currently active stream objects in the given program.

Following the paradigm of developing object-oriented software, the new module threading also provides a certain class Thread that supports the necessary object-oriented implementation of threads. The following methods are supported in this class:

- Run (): This method is executed when a new thread is initialized and started.
- Start (): This method starts the initialized caller of the thread by calling the corresponding method. Run ()
- Join (): Such a method waits for the completion of the corresponding calling thread object before continuing with the rest of the program.

- IsAlive (): This method returns a Boolean value indicating whether the calling thread object is currently executing
- GetName (): This method returns the actual name of the caller of the stream.
- SetName (): This method sets the corresponding name of this caller to the thread.

Creating a new thread in Python

Providing some overview of the new module threading and its differences from the existing old module thread in this section, we will study a number of examples of creating new threads using these tools in Python. As mentioned earlier, the new module threading is probably the most common way to work with threads in Python. Special cases require the use of the old module thread, and maybe also other tools, and it is important for us to be able to detect such situations.

Starting a thread with thread

In the module thread for the execution of functions, new threads are jointly created. As we already mentioned, the main way to accomplish this is to use a function thread.start_new_thread():

```
Thread.start_new_thread(function, args[, kwargs])
```

When this function is called, a new thread is generated to execute the function that is defined in the passed parameters, and when this function finishes its execution, the identifier value of the created thread is returned. The value of the parameter function is the name of the function to be executed, and the list of parameters ages (which must be a list or a tuple) contain those arguments that must be passed to this defined function. An optional parameter kwargs, on the other hand, contains a separate dictionary of additional arguments with keywords. After that, a return is made from the generated function thread.start_new_thread(), this thread will also quietly end.

Let's take a look at some example of using a module thread in a Python program. If you already have unloaded the necessary code for this book from our site GitHub, go on and navigate to the folder Chapter03 of the file Chapter03/example2.py. In this example, we will look at a function is_prime()that we have already used in our previous chapters:

```
# Chapter03/example2.py
From math import sqrt
Def is_prime (x):
    if x < 2:
        Print ('%i is not a prime number.' % x)

    Elif x == 2:
        Print ('%i is a prime number.' % x)

    Elif x % 2 == 0:
        Print ('%i is not a prime number.' % x)

    Else:
        Limit = int (sqrt (x)) + 1
        For I in range (3, limit, 2):
            If x % I == 0:
                Print ('%i is not a prime number.' % x)

        Print ('%i is a prime number.' % x)
```

You may notice that the result of its calculation of this function is_prime(X) is a completely different way to return the result; instead of returning true or false to indicate whether the parameter is a prime number, this function is_prime()directly prints this result. As we said earlier, our function thread.start_new_thread()executes the function received by the parameter by generating a new thread, but in practice it returns the value of the received stream identifier. By printing the resulting result of our function, is_prime()we bypass the task of gaining access to the very result of this function in the module stream thread.

In the most basic part of our program, we organize a cycle according to some list of potential candidates to be prime numbers and we will call our function thread.start_new_thread()for the calculating function is_prime()and each number from this list as follows:

```
# Chapter03/example2.py
import _thread as thread

my_input = [2, 193, 323, 1327, 433785907]

for x in my_input:
    thread.start_new_thread(is_prime, (x, ))
```

You can note that in this file Chapter03/example2.pythere is a line of code for receiving input from the client at the very end:

```
a = input('Type something to quit: \n')
```

Now let's comment on this very last line. Then, when we execute the entire Python program, it will appear that our program ended without printing any results; in other words, our program ended before its threads can complete its execution. This happens for the reason that when a new thread is generated from a function thread.start_new_thread() to process a number from our input list, the program itself continues to loop through all subsequent numbers while such newly created threads are executed.

Therefore, by the time our Python interpreter reaches the very end of its program, if no thread has completed execution (in our case, these are all threads), such a thread will be ignored and terminated and no output will be printed. However, from time to time one of the output is

2 is a prime number that it is the result printed before the completion of this program, since the processing number of the 2thread has the ability to complete until this point.

The very last line of code is another workaround for our module thread this time to solve the above problem. This line keeps our program from exiting until the user enters any keystroke to complete all running threads (that is, to complete processing of all numbers in the input list). Remove the comment from the very last line and execute this file, as a result you will get your output, which will look like the following:

```
> python example2.py

Type something to quit:

2 is a prime number.

193 is a prime number.

1327 is a prime number.

323 is not a prime number.

433785907 is a prime number.
```

As you can note, the line Type something to quitthat corresponds to the very last line of code from our program was output before we got the output from the function is_prime(); this is consistent with the fact that this line is executed before all other threads have completed their execution, in most cases. I specify exactly “in most cases”, because when our thread that processes the very first input (number 2) finishes execution before the Python interpreter reaches the very last line, our output can sometimes look similar to the following:

```
> Python example2.py

2 is a prime number.

Type something to quit:

193 is a prime number.
```

323 is not a prime number.

1327 is a prime number.

433785907 is a prime number.

Starting a thread using threading

Now you know how to start a certain thread using the module thread, and also you know about its limitations and the use of threads at a low level, in addition to the need to consider an intuitively clear workaround when working with it. In this section, we will examine the more preferred module threadingand its advantages over threadthe implementation of multi-threaded programs in Python.

In order to create and customize some new stream using the new module threading, there are certain steps, which should be as follows:

In your program, define a subclass of the general class threading.Thread

Inside the resulting subclass, rewrite the default method __init__(self [,args])to add the necessary individual arguments to this class

Inside the same subclass, rewrite the default method run(self [,args])for personalizing the existing behavior of this class of threads when initializing and starting some new thread

In fact, you already saw a certain example in the very first example from this chapter. To recall, the following is what we have to use to individualize a subclass threading.Threadto perform a five-step countdown with a certain personal delay between each steps:

# Chapter03/my_thread.py

Import threading

```
Import time

Class MyThread (threading. Thread):
    def __init__(self, name, delay):
        Threading. Thread.__init__(self)
        self.name = name
        self.delay = delay

    def run(self):
        print('Starting thread %s.' % self.name)
        thread_count_down(self.name, self.delay)
        print('Finished thread %s.' % self.name)

def thread_count_down(name, delay):
    counter = 5

    while counter:
        time.sleep(delay)
        print('Thread %s counting down: %i...' % (name, counter))
        counter -= 1
```

In our next example, we will consider our task of identifying whether a given number is prime. This time we are implementing some kind of multi-threaded Python program through a new module threading. Go to the folder Chapter03 to the file example3.py. Let's focus on our subclass first MyThread:

```
# Chapter03/example3.py
Import threading

class MyThread(threading.Thread):
    def __init__(self, x):
        threading.Thread.__init__(self)
        self.x = x

    def run(self):
        print('Starting processing %i...' % x)
        is_prime(self.x)
```

Every instance of our class MyThreadhas a parameter with a name xthat defines a candidate number to be simple. As you can see, when a certain instance of this class is initialized and launched (or rather, in a function run(self)), our function is_prime(), which is the same function to check whether a number is prime, which we used in our previous example, with the specified parameter x, and before that, in our function, run()to indicate the beginning of this process, a message is also printed.

In our main program, we still have the same input list to check for membership in primes. We intend to go through all the numbers from this list, generating and launching a new instance of this class MyThreadwith this number and adding this instance MyThreadto a separate list. Such a list of created threads is necessary for the reason, because after that we will have to call a method join()for all these threads, which will certify all these threads have successfully completed their execution:

```
my_input = [2, 193, 323, 1327, 433785907]

Threads = []

For x in my_input:
    temp_thread = MyThread(x)
    temp_thread.start( )

    Threads. Append (temp_thread)
```

```
for thread in threads:
    Thread. Join ()

Print ('Finished.')
```

Note that unlike the case when we have been using the old module thread, this time we do not need to invent some sort of workaround to ensure that all threads successfully complete their execution. Again, this is done by the specified method join(), which is provided by the new module threading. This is just one instance of the many benefits of using the more powerful top-level API of the new module threadingregarding the use of the traditional module thread.

Thread synchronization

As you saw in our previous examples, the new module threading has many advantages over its predecessor, the traditional module thread, in terms of functionality and top-level API calls. Even though some people recommend sophisticated Python developers to know how to implement multi-threaded applications using both modules, most likely you will use the module to work with Python threads threading. We will look into the use of the module threading when synchronizing threads.

The concept of thread synchronization

Before we jump to some real Python example, let's explore the very concept of synchronization in computer science. As you saw in previous chapters, it is sometimes undesirable to have all portions of a program executed in parallel. In fact, in the most modern joint programs, even within some joint part, some form of coordination between different threads / processes is also required.

Thread / Process Synchronization is a certain concept from computer science, which defines various mechanisms of guaranteeing that at a time no more than one simultaneously executed thread / process can process and execute some specific part of the code; this part of the code is called the critical section.

In a specific program, when a thread accesses / executes the corresponding critical section of a given program, all other threads must wait until this thread completes execution. The most typical goal of thread synchronization is to avoid potential data inconsistency when organizing access by multiple threads of resources shared by them; allowing only one thread to execute this critical section of your program at a time to ensure that no conflicts with data occur in a multi-threaded application. { Note trans.: an example illustrating such a conflict is given in our translation of the Asyncio Serial Bots Restaurant section of Python 3 Tsaleba Hattingha. }

Class threading.Lock

One of the most common ways to use thread synchronization is to implement some kind of locking mechanism. In our module threading, the class threading.Lock provides a certain example of an intuitive approach to creating locks and working with them. Its main use includes such methods:

Threading.Lock(): This method initializes and returns a certain lock object.

Acquire (blocking): When calling this method, all available threads will be launched synchronously (that is, only one thread can execute this critical section at a certain point in time):

An optional argument blocking allows us to determine whether a given current thread should wait to receive a given lock.

When blocking = 0, this current thread does not wait for this lock and simply returns 0 if such a lock cannot be obtained by this thread or 1in the opposite case.

When blocking = 1, this current thread executes blocking and waits until this lock is released and receives it after that.

release(): When this method is called, its blocking is released.

Some Python example

Let's look at a specific example. In this example, we will view our file Chapter03/example4.py. We will return back to our example of a countdown from five to one, which we already examined at the very beginning of this chapter; seize the moment to go back and remember the problem statement. In this example, we will adjust our class MyThread as follows:

```
# Chapter03/example4.py

Import threading
Import time

Class MyThread (threading. Thread):
    Def __init__ (self, name, delay):
        Threading. Thread.__init__(self)
        Self. Name = name
        self.delay = delay

    Def runs (self):
        Print ('Starting thread %s.' % self. name)
        thread_lock.acquire()
        thread_count_down(self.name, self.delay )
        thread_lock.release()
        print('Finished thread %s.' % self.name)

Def thread_count_down (name, delay):
    Counter = 5

    While counter:
        Time. sleep(delay)
        Print ('Thread %s counting down: %i...' % (name, counter))
        Counter -= 1
```

In contrast to the very first example in this chapter, a class MyThreadapplies some kind of object lock (whose variable name has a name thread_lock) inside its function run() . In particular, this lock object is obtained immediately before the function is called thread_count_down()(that is, when the countdown begins), and this locked object is released immediately after its completion. Theoretically, this specification will allow us to change the behavior of our flaws, which we saw in the very first example; instead of the simultaneous available countdown, our program now executes the available flows separately, and each subsequent countdown will take place after the previous one.

Finally, we initialize our variable thread_lockand also run two separate instances of the class MyThread :

```
thread_lock = threading.Lock()

thread1 = MyThread('A', 0.5)
thread2 = MyThread('B', 0.5)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print('Finished.')
```

The output will be like this:

```
> python example4.py
Starting thread A.
Starting thread B.
Thread A counting down: 5...
Thread A counting down: 4...
Thread A counting down: 3...
Thread A counting down: 2...
Thread A counting down: 1...
Finished thread A.
Thread B counting down: 5...
Thread B counting down: 4...
Thread B counting down: 3...
Thread B counting down: 2...
Thread B counting down: 1...
Finished thread B.
Finished.
```

Priority multi-threaded queue

The concept of informatics that is widely used both in non-parallel and in joint programming is the use of queues. A queue is an abstract data structure, which is a collection of various elements to accompany a certain established order; these elements may be other objects in some program.

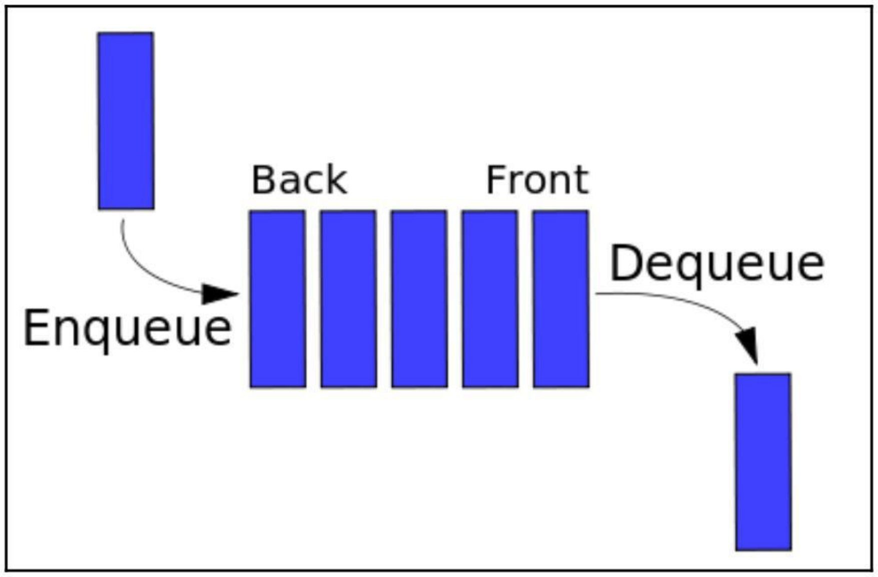
The relationship of the real world and programmable queues

Queues are an intuitive concept that can easily be correlated with everyday practice, for example, when you stand at a counter in a line at the airport. In a real chain of people, you observe the following:

- People usually get up on one side of this chain and leave it on the other side.
- If person A enters this chain before person B, then person A will also leave this chain until person B (unless person B has a higher priority).
- When everyone is accommodated on the plane, no one will be left in this queue. In other words, this chain will be empty.

In computer science, the queue works in much the same way:

- Elements can be added at the very end of a certain queue; this task is called queuing (enqueue).
- Elements can also be deleted from the very beginning of such a queue; this task is called fetching from the queue (dequeue).
- In the FIFO queue (First In First Out , First In, First Out), that element that is added first will also be deleted first (location and name, FIFO). This is the opposite of another common data structure in computer science called the stack , in which the most recently added item is deleted first. What is also referred to as LIFO (Last In, First Out, last come - first go).
- If all elements have been deleted inside a queue, such a queue will be empty and there will be no way to delete further elements from such a queue. Similarly, if a given queue reaches its maximum capacity in terms of the number of elements that it can contain, there is no way to add any elements to this queue:



Queue module

A module queueing Python provides some simple implementation of such a queue data structure. Each queue from an existing class queue.Queuecan contain a certain number of elements and can have the following methods as its top-level API:

- Get(): This method returns the next element by calling the object queue and removing it from this object queue
- Put (): This method adds a new item to the given called object. Queue
- Qsize (): This method returns the total number of current elements in the given called object queue(i.e. Its size).
- Empty(): This method returns a Boolean value indicating whether the called object is queue empty.
- Full (): This method returns some Boolean value indicating whether the called object is queued full.

Queuing in parallel programming

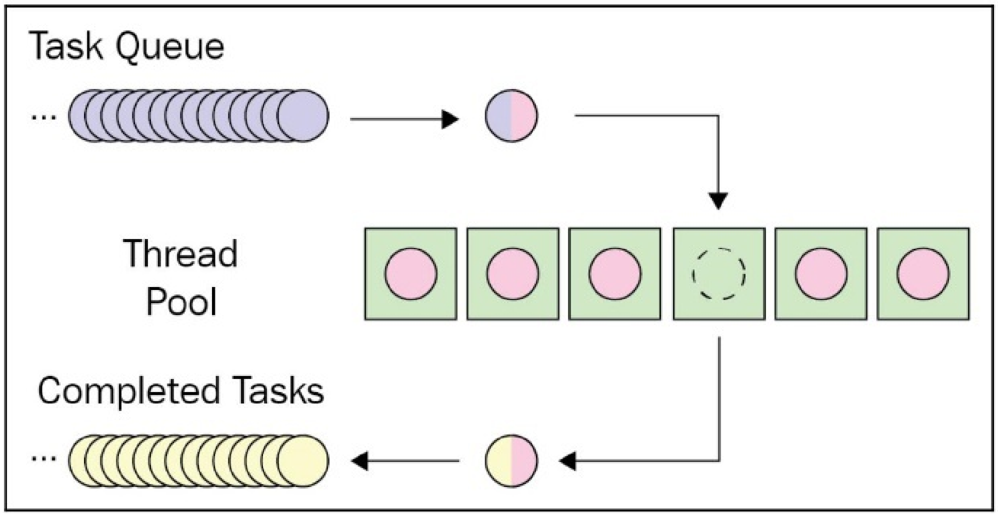
The concept of queue under discussion is even more common in the subregion of joint programming, especially when we need to implement some fixed number of threads in our program to interact with a variable number of shared resources.

In our previous examples, we studied the purpose of a certain specific task to some new thread. This means that the total number of tasks that need to be processed will be dictated by the total number of threads that our program should generate (for example, in its file Chapter03/example3.py we had five numbers as our input and we therefore created five threads - each took one the number from the input and processed it.)

Sometimes it is not advisable to have as many threads as there are tasks for processing. Say you have a large number of processes to be processed, then it would not be enough to efficiently generate the same number of threads and have each thread perform only one task. Having a fixed number of threads (usually referred to as a certain pool of threads) that could work with existing tasks based on cooperation can give more advantages.

Here the concept of queue comes onto the scene. We have the opportunity to design a structure in which our thread pool will not save any information related to the tasks that it should perform, but instead the tasks themselves are stored in a queue (in other words, create a task queue), and the corresponding elements from this queue will feed the personal participants of the existing thread pool. As the selected task ends with a certain member of our thread pool, if the present task queue still contains the elements to be processed, then the next element from this queue will be sent to the thread that has just been freed.

The diagram below illustrates this setup:



Queues when processing in threads

Let's look at a quick Python example to illustrate this point. Browse to the appropriate file Chapter03/example5.py. In this example, we will consider the problem of deriving all the positive factors of some elements in a given list of positive integers. We still adhere to our class MyThread, but with some settings:

```
# Chapter03/example5.py
Import queue
Import threading
Import time

Class MyThread (threading. Thread):
    Def __init__ (self, name):
        threading.Thread.__init__(self)
        Self. name = name

    Def runs (self):
        Print ('Starting thread %s.' % self. name)
        process_queue( )
        Print ('Exiting thread %s.' % self. name)
```

```
Def process_queue ():
    While True:
        Try:
            x = my_queue.get(block=False)
        Except queue.Empty:
            Return
        Else:
            print_factors(x)

        time.sleep(1)

Def print_factors (x):
    result_string = 'Positive factors of %i are: ' % x
    For I in range (1, x + 1):
        If x % I == 0:
            result_string += str(i) + ' '
    result_string += '\n' + '_' * 20

    Print (result_string)

# setting up variables
input_ = [1, 10, 4, 3]

# filling the queue
my_queue = queue.Queue()
for x in input_ :
    my_queue.put(x)

# initializing and starting 3 threads
```

```
thread1 = MyThread('A')
thread2 = MyThread('B')
thread3 = MyThread('C')

thread1.start()
thread2.start()
thread3.start()

# joining all 3 threads
thread1.join()
thread2.join()
thread3.join()

print('Done.')
```

There are a lot to look at, so let's break this program down into smaller parts. First, let's look at our key function:

```
# Chapter03/example5.py

def print_factors(x):
    result_string = 'Positive factors of %i are: ' % x
    for i in range(1, x + 1):
        if x % i == 0:
            result_string += str(i) + ' '
    result_string += '\n' + '_' * 2 0

    print(result_string)
```

This function beats a certain parameter, x then iterates over all positive numbers between 1and its value to check whether this number is a certain factor x. At the end, a formatted message is printed, which contains all the necessary information that has accumulated in our cycle.

In our new class MyThread, after a new instance is initialized and started, the corresponding function will be called process_queue(). This function will first try to get the next required element from the existing queue object, which is contained in the variable in my_queuesome way without blocking by calling the corresponding method get(block=False). If a certain exceptional situation occurs queue.Empty(which indicates that this queue does not contain values), then we complete the execution of this function. Otherwise, we simply pass the element that we just received into our function print_factors().

```
# Chapter03/example5.py
Def process_queue ():
    while True:
        try:
            x = my_queue.get(block=False)
        Except queue.Empty:
            Return
        Else:
            print_factors(x)

        Time. sleep(1)
```

A variable is my_queuedefined in our main function as an object Queue from a module queue that contains all the elements in its listinput_:

```
# setting up variables
input_ = [1, 10, 4, 3]
```

```
# filling the queue
my_queue = queue.Queue(4)
For x in input_:
    my_queue.put(x)
```

In the remainder of our main program, we simply initiate and run such separate threads until they all complete their respective executions. In this case, we stopped at creating only three threads to simulate our architecture, which we discussed above - a certain fixed number of input queue processing threads, whose number of elements can be changed independently:

```
# initializing and starting 3 threads
thread1 = MyThread('A')
thread2 = MyThread('B')
thread3 = MyThread('C')

thread1.start()
thread2.start()
thread3.start()

# joining all 3 threads
thread1.join()
thread2.join()
thread3.join()

Print ('Done.')
```

Run this program and get the following output:

```
> Python example5.py
Starting thread A.
Starting thread B.
Starting thread C.
Positive factors of 1 are: 1
_____________________
Positive factors of 10 are: 1 2 5 10
_____________________
Positive factors of 4 are: 1 2 4
_____________________
Positive factors of 3 are: 1 3
_____________________
Exiting thread C.
Exiting thread A.
Exiting thread B.
Done.
```

In this example, we implemented the structure that we discussed earlier: a task queue that stores all tasks to be executed and a certain pool of threads (threads A? \, B and C) that interact with an existing queue to process its elements.

Priority multi-threaded queue

All elements in a queue are processed in the order in which they were added to this queue; in other words, the very first item added to the queue leaves this queue first (FIFO). Even though such an abstract data structure imitates real life in many cases, depending on the application itself and its goals, sometimes we need to redefine / change the existing order of the elements in the queue dynamically. It is in such a situation that the concept of priority queue becomes convenient.

Priority queue abstract data structure similar to the already discussed queue data structure (and even the aforementioned concept of the stack), but each element in a priority queue, as its name anticipates, has a certain priority associated with it; in other words, when an element is added to the priority queue, it is necessary to determine its priority. Unlike a regular queue, the principle of selecting from a priority queue relies on the priority value of the available elements: those elements with a higher priority are processed earlier than competitors with a lower priority.

This priority queue concept is used in a variety of different applications - namely, bandwidth management, Dijkstra's algorithm, best first search algorithms, and the like. Each of these applications usually uses a system of points with a finite number of values / function to determine the priority value of its elements. For example, when managing bandwidth, priority exchanges, such as exchanges in real-time streams, are processed with the least delay and the lowest probability of failure. In the best search algorithms that are used to find the shortest path between two defined nodes of a graph, a priority queue is implemented to track unexplored routes; routes with a shorter estimated length have a higher priority in such a queue.

Findings

Some thread execution is the smallest element of command programming. In computer science, multithreaded applications allow multiple threads to exist within the same process at the same time to implement simultaneous processing and parallelism. Multithreading provides various advantages at runtime, in the form of responsiveness, as well as the effectiveness of resource utilization.

A module threading in Python 3. Which is usually seen as covering an earlier module thread, provides an efficient, powerful and yet top-level API for working with threads when implementing multi-threaded Python applications, including options for spawning new threads dynamically, as well as for synchronizing threads through various mechanisms.

Queues and queues with prioritization are important data structures in the informatics industry, and they are an essential concept in synchronous processing and parallel programming. They make it possible for multi-threaded applications to efficiently execute and accurately terminate their threads, ensuring that shared resources are handled in a special way and dynamically.

CHAPTER 7

Using the with statement in threads

The operator within Python sometimes confuses both beginners and experienced Python programmers. This chapter gives in-depth explanations of the main idea behind the operator with acting as a context manager and its application in joint and parallel programming, in particular, regarding the specific use of locks for thread synchronization. This section also provides specific instances of how this operator is most often used to.

This section will cover the following topics:

The very concept of context management and the options that this operator with provides as a kind of context manager, especially in joint and parallel programming.

Actually the operator syntax withand how to efficiently and effectively apply it.

Different ways to use the operator within co-programming.

Technical requirements

Here is a list of prerequisites for this chapter:

Make sure Python 3 is already installed on your computer

Unload the required repository from GitHub

Throughout this chapter, we will work with a subfolder named Chapter04

Check out the following Code in Action videos

Context management

The operator was with the first proposed in Python 2.5 and has been used for quite some time. However, there is still confusion in its use even by experienced Python programmers. This operator with is mainly used as a context manager that properly manages resources, and this is especially important in joint and parallel programming, when resources are shared by various entities in a particular joint or parallel application.

We start with control files

As an experienced Python user, you have probably already seen that this operator with is used to open and read external files from within Python programs. Considering this task at a certain lower level, this operation of opening some external file in Python consumes a certain resource - in this case, some file descriptor - and your operating system will set a certain limit on this resource. This means that there is some upper limit on how many files can be opened at the same time by a separate process running on your system.

Let's quickly look at an example to illustrate this point further. Let's look at the file Chapter04/example1.pythat is shown in the following code:

```
# Chapter04/example1.py

n_files = 10
files = []

for i in range(n_files):
    files.append(open('output1/sample%i.txt' % i, 'w'))
```

This program is in a hurry, simply creates 10 text files in the appropriate folder output1: sample0.txt, sample1.txt, ..., sample9.txt. What may be more interesting for us is the fact that these files were open inside our cycle for but were not closed - this is a bad programming practice, which will be discussed later. Now, let's say we wanted to reassign the variable n_filesto a larger number — say, 10,000 — as shown in this code:

```
# Chapter4/example1.py

n_files = 10000
Files = []

# method 1
For I in range (n_files):
    Files. Append (open ('output1/sample%i.txt' % I, 'w'))
```

We will get some error similar to the following:

```
> Python example1.py
Traceback (most recent call last):
  File "example1.py", line 7, in <module>
OSError: [Errno 24] Too many open files: 'output1/sample253.txt'
```

Taking a closer look at this error message, we can see that my laptop is capable of processing only 253 open files at a time (as a note: if you work in a UNIX-like operating system, the call limit -and will give you the number of files that your system can process {Note: For Windows, use Testlimit.exe -h, more ... }). More generally, this situation has arisen because of what is called a file descriptor leak. When Python opens a certain file inside the program, this open file is actually represented by some integer value. This whole acts as a kind of pointer that the program can use to access this file this program completes the management of the underlying file itself.

Opening numerous files at the same time, our program assigns too many file descriptors to manage such open files, hence the corresponding error message. Leaking file descriptors can lead to many complex problems - especially with joint or parallel programming - namely, unauthorized I / O operations in already open files. The main solution for this is to simply close open files in a certain consistent way. Let's look at your file Chapter04/example1.py with its second method. In our loop forwe will do the following:

```
# Chapter04/example1.py
n_files = 1000
files = []

# method 2
for i in range(n_files):
    f = open('output1/sample%i.txt' % i, 'w')
    files.append(f)
    f.close()
```

The with statement as a context manager

In real-life applications, it is even easier to lose control over open files in your programs, forgetting to close them; this can sometimes also serve as the case when it is impossible to say whether a given program has finished processing a certain file, and we, programmers, will not be able to decide where exactly to put the close statement of this file in the proper way. This situation is even more common with joint and parallel programming, in which the order of execution often changes between different elements.

One of the possible solutions to this problem, which is also common in other programming languages, is to use the block try...except...finally whenever we want to interact with a certain external file. This solution still requires the same level of management and significant overhead and does not even provide a worthy improvement regarding the simplicity and readability of our programs. This is where the Python operator comes into play with.

This operator withgives us a simple way to ensure that all open files are properly managed and cleaned when our program stops using them. The most noticeable advantage of using the operator withis the fact that even if our code is successfully completed or it returns some kind of error, the operator itself withalways processes open files and manages them properly through the context. For example, let's take a look at the file in Chapter04/example1.pymore detail:

```
# Chapter04/example1.py

n_files = 254
files = []

# method 3
for i in range(n_files):
    with open('output1/sample%i.txt' % i, 'w') as f:
        Files. Append (f)
```

Although this method performs the same task as our second method, which we saw earlier, it additionally performs some cleaning and greater readability for managing open files that our program interacts with. More specifically, this operator with helps us to indicate the scope of certain variables — in this case, those variables that point to our open files — and therefore their context.

For example, in this third method in our previous code, the value of a variable findicates a specific current open file inside this block withat each iteration of the general cycle for, and as soon as our program leaves this block with (which happens outside the scope in which the variable acts f), therefore there is no other way to access it. Such a construction ensures that all cleanups associated with a certain file descriptor occur properly. This operator withtherefore called some kind of context manager.

Syntax with

Cbynfrcbc j, Ce; Iftvjuj jgthfnjhf with can be intuitive and fairly straightforward. In order to wrap a given execution unit, with methods are defined by a certain context manager, and it is compiled in such a simple way:

```
With [expression] (as [target]):
    [Code]
```

Note that part of as [target] the operator under discussion with is not actually required, as we will discover later. In addition, a given operator with can also process more than one element on the same line. In particular, the corresponding context managers are considered as a set of operators with nested one by one. For example, the following code:

```
With [expression1] as [target1], [expression2] as [target2]:
    [Code]
```

It can be interpreted as follows:

```
With [expression1] as [target1]:
    With [expression2] as [target2]:
        [Code]
```

With statement in parallel programming

Obviously, opening and closing external files is not very similar to collaborative processing. However, we mentioned earlier that our operator with, acting as a context manager, is not only used to manage file descriptors, but usually most resources. And if you have already actually found that controlling the locking of objects from a class is threading.Lock()similar to managing external files when studying Chapter 2, Amdahl's Law , then it's convenient to use their comparison.

As a reminder, locks are mechanisms in joint and parallel programming that are typically used to synchronize threads in multi-threaded applications (that is, to prevent more than one thread from accessing a critical section at the same time). However, as we discuss again in Chapter 12, Hangs , locks are also a common source of deadlocks , during which a thread gets hold of some sort of blocking but never releases it due to circumstances that cannot be processed, whereby it stops the entire program entirely.

Mutual lock processing example

Let's look at a quick example. To do this, look at the corresponding file Chapter04/example2.py, which is shown in the following code:

```
# Chapter04/example2.py
From threading import Lock

my_lock = Lock()

def get_data_from_file_v1(filename):
    my_lock.acquire()

    with open(filename, 'r') as f:
        data.append(f.read())

    my_lock.release()

data = []

try:
    get_data_from_file('output2/sample0.txt')
except FileNotFoundError:
    print('Encountered an exception...')

my_lock.acquire()
```

print('Lock can still be acquired.')

In this example, we have a certain function get_data_from_file_v1()that gets the corresponding path to some external file, reads data from it and appends data from a previously defined list with the name to its end data. Inside this function, with the help of my_locka certain blocking of the object, which is also defined before the call to this function, it is reached and released when the given parametric file is read before and after, respectively.

With our main program, we will try to call get_data_from_file_v1 () for some non-existent file, which is one of the most common programming errors. At the very end of this program, we will also take possession of the corresponding object lock again. The main point is to find out whether our programming is able to handle this error reading a nonexistent file properly and accurately only by applying the block try... except that we have.

After you run this script you'll observe that your program will print any error message, which is defined in the block try... except, Encountered an exception... as expected, because the file cannot be found. However, this program will also refuse to execute all the remaining code; it will never reach the very last line of code - print('Lock acquired.')- and hang forever (or until you press Ctrl + to force quit this program).

This is precisely the situation of mutual deadlock, which, again, occurs when we take possession of my_lockour function inside get_data_from_file_v1(), but since our program encountered some error before executing my_lock.Release (), this blocking is never released. This in turn causes the corresponding line my_lock.Acquire () to hang at the very end of our program, since blocking cannot be obtained in any way. Therefore, our program will not be able to reach their most recent code line print('Lock acquired.').

However, this problem can be solved quickly and effortlessly with the help of some operator with. In your file, example2.Pyjust set the comment on the call line get_data_from_file_v1()and remove the comment from the line get_data_from_file_v2(), and you will get the following:

```
# Chapter04/example2.py

From threading import Lock

my_lock = Lock()

Def get_data_from_file_v2 (filename):
    With my_lock, open (filename, 'are') as f:
        Data. append(f.read())

Data = []

Try:
    get_data_from_file_v2('output2/sample0.txt')

Except :
    print('Encountered an exception...')
my_lock.acquire()
Print ('Lock acquired.')
```

In our function, get_data_from_file_v2()we have the necessary equivalent of a pair of such nested operators with:

```
With my_lock:
    With open (filename, 'are') as f:
        Data. append(f.read())
```

Since objects Lock is context managers, a simple application with my_lock: it will guarantee the receipt and release of the necessary blocking, even if a certain exception is raised inside this block. After running this script, you will get the following output:

```
> Python example2.py
Encountered an exception...
Lock acquired.
```

We can make sure that this time our program was able to take hold of the necessary lock and gently reach the very end of this scenario without any errors.

Findings

The Python operator with offers an intuitive way to manage resources while ensuring that errors and the exceptions are handled rightly. This ability to manage the resources is even more important for joint and parallel programming, etc., which are used by different resources in different entities - in particular, by using the corresponding operator with objects threading. Lock that is used to synchronize different threads in some multi-threaded application.

In addition to better error handling and guaranteed task cleaning, the operator under discussion withal so provides additional readability for your programs, which is one of the strengths that Python offers its developers.

CHAPTER 8

Working with processes in Python

This chapter will cover some of the most common ways to work with processes using the API of this module multiprocessing, such as a class Process, a class, Pool and interprocess communication tools, such as a class Queue. This chapter will also look at the key differences between multithreading and many processes in collaborative programming.

This chapter will cover the following topics:

The very concept of a process in the context of joint programming in computer science

Python Module API Basics multiprocessing

How to interact with processes and those advanced functions that the module provides multiprocessing

How the module multiprocessing supports inter-process communication

Key Differences Between Multiprocessing and Multithreading in Joint Programming

Technical requirements

Here is a list of prerequisites for this chapter:

Make sure Python 3 is already installed on your computer

Unload the required repository from GitHub

Throughout this chapter, we will work with a subfolder named Chapter06

Check out the following Code in Action videos

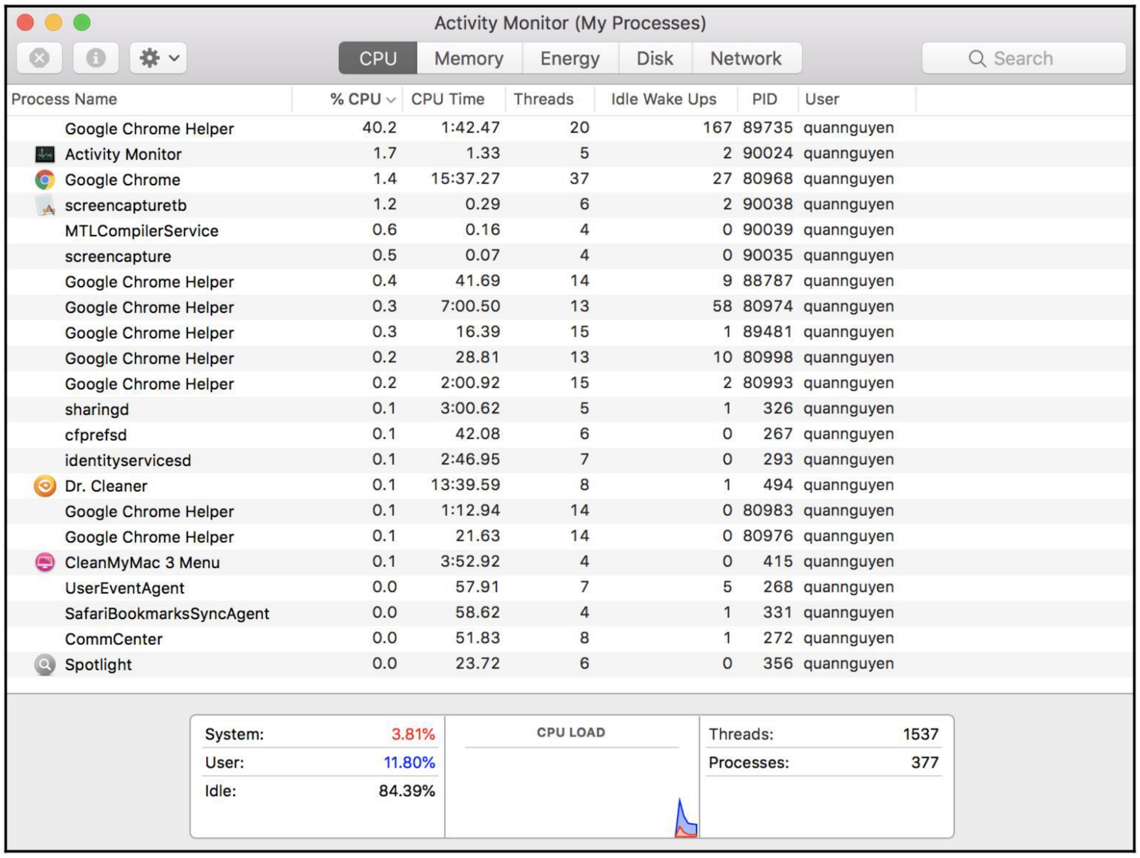
Process concept

In the field of informatics, a certain execution process is some kind of instance of a certain computer program, or software, which is subject to execution by the operating system. A certain process contains both the program code itself and its current operations and interAction with other entities. Depending on the operating system, the implementation of the process itself can be performed from multiple threads of execution, which can execute instructions together or in parallel.

It is important to note that a certain process is not equivalent to any computer program. While a certain program is just some static set of instructions (program code), a certain process is instead a real execution of such instructions. It also means that the same program can be launched simultaneously by spawning multiple processes. These processes execute the same code from one parent program.

For example, the Internet browser Google Chrome usually controls a certain process called Google Chrome Helper as its main program to provide web browsing and other processes with help for various purposes. The easiest way to see the various processes of your system is to start and manage, involving the use of Task Manager for Windows, Activity Monitor for iOS and System Monitor for Linux operating systems.

Below is a screen shot of my Activity Monitor . In this list you can find many processes called Google Chrome Helper . The PID column (which is an abbreviation for process ID) reports the value of the unique identifier that each such process has: A simple list of processes



Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID	User
Google Chrome Helper	40.2	1:42.47	20	167	89735	quannguyen
Activity Monitor	1.7	1.33	5	2	90024	quannguyen
Google Chrome	1.4	15:37.27	37	27	80968	quannguyen
screencapturetb	1.2	0.29	6	2	90038	quannguyen
MTLCompilerService	0.6	0.16	4	0	90039	quannguyen
screencapture	0.5	0.07	4	0	90035	quannguyen
Google Chrome Helper	0.4	41.69	14	9	88787	quannguyen
Google Chrome Helper	0.3	7:00.50	13	58	80974	quannguyen
Google Chrome Helper	0.3	16.39	15	1	89481	quannguyen
Google Chrome Helper	0.2	28.81	13	10	80998	quannguyen
Google Chrome Helper	0.2	2:00.92	15	2	80993	quannguyen
sharingd	0.1	3:00.62	5	1	326	quannguyen
cfprefsd	0.1	42.08	6	0	267	quannguyen
identityservicesd	0.1	2:46.95	7	0	293	quannguyen
Dr. Cleaner	0.1	13:39.59	8	1	494	quannguyen
Google Chrome Helper	0.1	1:12.94	14	0	80983	quannguyen
Google Chrome Helper	0.1	21.63	14	0	80976	quannguyen
CleanMyMac 3 Menu	0.1	3:52.92	4	0	415	quannguyen
UserEventAgent	0.0	57.91	7	5	268	quannguyen
SafariBookmarksSyncAgent	0.0	58.62	4	1	331	quannguyen
CommCenter	0.0	51.83	8	1	272	quannguyen
Spotlight	0.0	23.72	6	0	356	quannguyen

System:	3.81%
User:	11.80%
Idle:	84.39%

Threads:	1537
Processes:	377

Process and Thread Mapping

One of the most common mistakes that programmers make when developing joint or parallel applications is to confuse the existing structure and functions of processes and threads. As we saw in Chapter 3, Working with Threads in Python , a thread is the smallest piece of program code and usually it is some kind of process component. Moreover, more than one thread can be implemented inside the same process for accessing and sharing memory and other resources, while different processes do not interact in this way. This cordial relationship is shown in the following diagram:

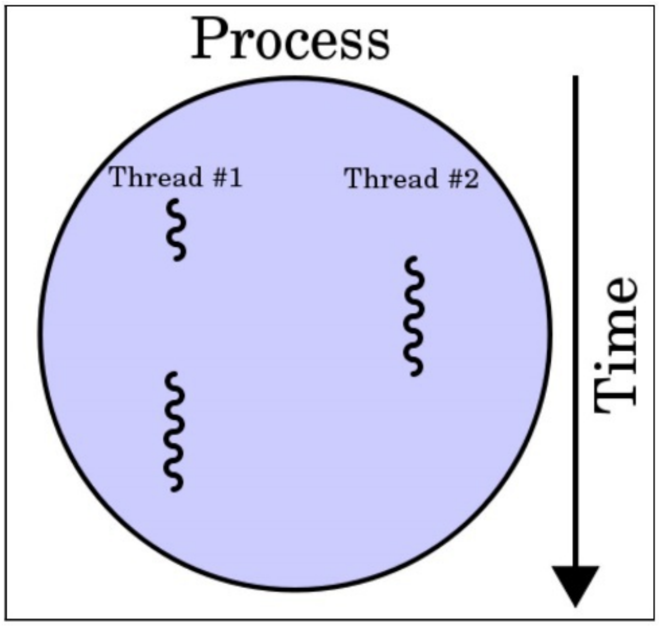


Diagram of two threads executing in one process

Since a certain process is an element of a larger size than a stream, it is also more complex and composed of a larger number of software components. A process, therefore, also requires large resources, while in a thread this is not so and sometimes it is called a process with a low weight. In a typical process of a computing system, there are a number of basic resources displayed in the following list:

A certain image (or copy) of the code that must be executed from its parent program.

The memory associated with a certain instance of the program. It should contain executable code, input and output data for this particular process, a call stack for managing program events, and a certain heap that contains the data generated by the calculations and used by this process during execution time.

Descriptors for those resources that are allocated to this particular process by its operating system. We saw some example of this - file descriptors - in Chapter 4, Using the with operator in streams .

The security components of a certain process, namely, the owner of this process and its authority, as well as valid operations.

The actual state of the processor, also referred to as the context of this process. The data of such a context of a process are often found in the registers of the processor, the RAM used by this processor or in the control registers used by its operating system to control the process itself.

Since each process has a certain state allocated to it, processes, support more information about the state than threads; many processes within a process, in turn, share process states, memory, and various other resources. For similar reasons, processes interact with each other exclusively through interprocess communication methods provided by the system, while threads can easily interact with each other by sharing resources.

In addition, context switching - a certain action to save the current state of the data of a process or thread to interrupt the execution of a task and resume it later - requires more time between different processes than between different threads within the same process. Nevertheless, despite the fact that we have already observed that the interaction between threads requires careful memory synchronization to guarantee the correct processing of the data, due to the fact that there are interaction between separate processes, processes require less memory synchronization if they are not without such.

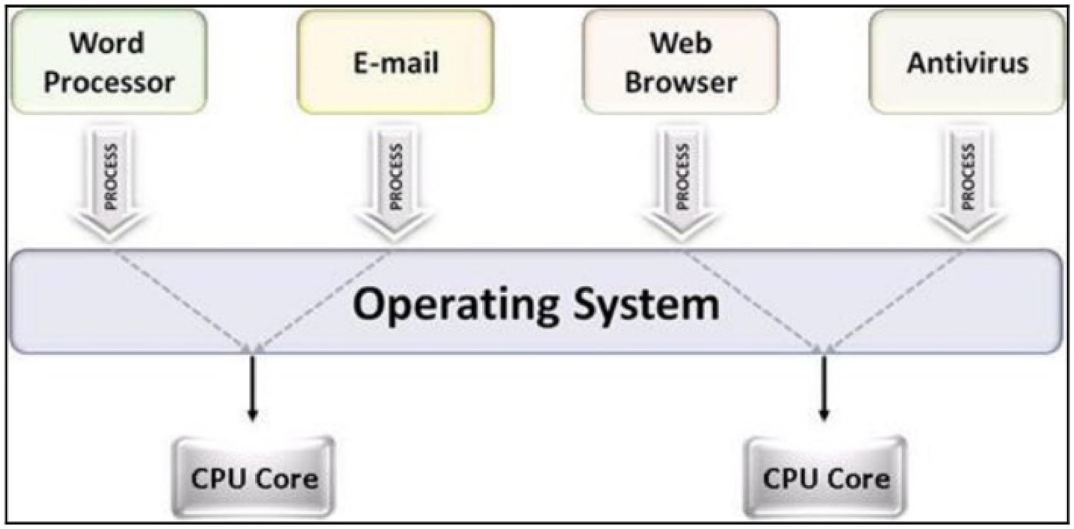
Multiplicity of processes

Multitasking in computer science is a common concept. At some multitasking operating system simply switches between different processes at high speed, giving the appearance that these processes are executed at the same time, despite the fact that usually takes place only pursuant to a single process in a single CPU (the CPU , central processing unit) at any given time. In contrast, the multiple-process method employs more than one CPU to carry out a task.

Although there are a number of various ways of applying this term of multiprocessing, in our context of joint execution and parallelism, multiprocessing refers to the execution of many joint processes in the operating system, in which each process is executed in a separate CPU, in contrast to the fact that only a separate process is executed at a certain point in time. By the very nature of this process, the operating system needs to have two or more CPUs in order to be able to implement tasks with many processes, since it needs to support many processors at the same time and divide them between tasks accordingly.

This cordial relationship is shown in the following diagram:

Example of a process interaction scheme using two cores



We already saw in Chapter 3, Working with Threads in Python, that multithreading shares something similar to the definition of multiprocessing. Multithreading means that only one processor is used, and the system itself switches between tasks within that processor (which is also called time slicing, time slicing), while multiprocessing is usually expressed in real joint / parallel execution of many processes by using multiple processors.

Applications with many processes are gaining popularity in the fields of joint and parallel programming. Here is a listing of some reasons:

Faster execution time: As we already know, proper joint processing always provides additional acceleration to your programs, provided that some parts are able to execute independently.

Exemption from synchronization : Given the fact that individual processes do not share resources among themselves in any application with many processes, and developers often have to spend their time coordinating such shared resources and synchronizing them, unlike applications with many processes where they are not required efforts to guarantee correct data processing.

Crash safety: Since the processes are independent of each other both in terms of computational procedures and in terms of input / output, the failure of one of the processes does not affect the execution of the other in a program with multiple processes, with proper processing. This implies that programmers may be able to spawn a large number of processes (which their system is still capable of processing) and the resulting chances of crashing the entire application do not grow.

Having said all this, it is also worth mentioning the disadvantages of using many processes that should be taken into account and which are listed below:

Many processors are required: Again, multiprocessing requires the operating system to have more than one CPU. Even though many processors are quite common in computer systems these days, if you do not have more than one available, then this implementation of many processes will be impossible.

Processing time and space: As mentioned earlier, there are many complex components involved in the implementation of a process and its resources. Therefore, it requires considerable computational time and power to generate and control processes in comparison with the same tasks with threads.

Python introductory example

To illustrate an example of starting many processes on an operating system, let's look at a quick Python example. Let's dwell on our file Chapter06/example1.Pydisplayed below:

```
# Chapter06/example1.py

From multiprocessing import Process
Import time

def count_down(name, delay):
    Print ('Process %s starting...' % name)

    counter = 5

    While counter:
        time.sleep(delay)
        print('Process %s counting down: %i...' % (name, counter))
        counter -= 1

    print('Process %s exiting...' % name)

if __name__ == '__main__':
    process1 = Process(target=count_down, args=('A', 0.5))
    process2 = Process(target=count_down, args=('B', 0.5))

    process1.start()
    process2.start()

    process1.join()
    process2.join()
    print('Done.')
```

In this file we intend to go back to our countdown example, which we already covered in Chapter 3, Working with Threads in Python , when we looked at the concept of threads. Our function count_down()received a certain string as a process identifier and some kind of delay range. Then she performed a countdown from 5 to 1, falling asleep between iterations for the number of seconds that was determined in her parameter delay. This function also printed at each iteration a certain message with the identifier of the corresponding process.

As we said in Chapter 3, Working with threads in Python , the main point demonstrated by this example is to demonstrate the nature of the simultaneous execution of individual tasks at the same time. And this time going through excellent processes using a class Process from a module multiprocessing. In our main program, we initialize two processes simultaneously in order to implement two simultaneous separate countdowns based on time.

After running this Python script, you will get some output that should be similar to the following:

```
> Python example1.py
Process A starting...
Process B is starting...
Process B is counting down: 5...
Process A counting down: 5...
Process B is counting down: 4...
Process A counting down: 4...
Process B is counting down: 3...
Process A counting down: 3...
Process B counting down: 2...
Process A counting down: 2...

Process A counting down: 1...

Process B counting down: 1...

Process A exiting...
```

Exactly as expected, this conclusion tells us that these two countdowns were executed simultaneously; instead of completing the first countdown and then launching the second, our program executed these two countdowns almost simultaneously. Even though the processes are more costly and contain more additional overhead than threads, multiprocessing also shows a double improvement in terms of speed of program execution compared to the previous one.

Remember that with multiple streams, we observe a phenomenon in which the resulting print order changes with different starts of this program. In particular, sometimes process B can overtake process A during its countdown and end before process A, even if it was initialized later. It is this, again, the direct impact of the implementation and launch of two processes that perform the same function almost simultaneously. Running this scenario many times you will observe that most likely you will get various results in the sense of the order of the countdown being performed and the completion of such countdowns.

Multiprocessing Module Overview

The module multiprocessingis one of the most common implementations of multiprocess programming in Python. It offers methods for the generation of processes and interact with them using the same API module API threading(as we have already discovered this to the methods start()and join()in its previous sample). According to his documentation on the website, this method allows both local and remote joint processing and effectively avoids the global interpreter lock (GIL) in Python (which will be discussed in more detail in Chapter 15, Global Interpreter Lock) by using subprocesses instead of threads.

Class process

In a module, multiprocessingprocesses are usually spawned in a class Process and controlled by it. Each object Process represents a certain action that executes a separate process. For convenience, a class object is Processequivalent to methods and APIs that can be found in the corresponding class threading.Thread.

In particular, using the object-oriented programming approach, the class Process of multiprocessingprovides the following resources:

Run (): This method executes when initializing and starting a new process.

Start(): This method is started and initialized by calling the object Process by calling the appropriate method run().

Join (): This method waits for the completion of this called object Process before continuing with the rest of the program.

IsAlive (): The method returns a Boolean value indicating whether this call object is currently executing Process.

Name: This attribute contains the actual name of this call object Process.

PID: This attribute contains the process identifier of the called object Process.

Terminate (): The method terminates the execution of this call object Process.

As you can see in the previous example, when initializing an object, Processwe can pass parameters to some function and execute it in a separate process, defining itself target(for such an objective function) and parameters args(as arguments of the objective function). Note also that you can override the default constructor Process()and implement your own function run().

Since he is the main player in the module multiprocessingand in the joint processing of Python as a whole, we will look at the class first Processin our next section again.

Pool class

In the module, the multiprocessingclass Poolis mainly used to implement a certain pool of processes, each of which will take care of the tasks assigned to a certain object Pool. Usually a class is Poolmore convenient than a class Process, especially if you need to organize the data received from your joint applications.

In particular, we saw that the resulting completed order for various elements in the list with a high degree of probability changes, being placed in a certain function of joint processing when you run your program again and again. This leads to difficulties in ordering the resulting output of this program with respect to the initial order of the input data it receives. One of the possible solutions is to create process tuples and their conclusions with their subsequent sorting by process identifier.

This problem is solved by a class discussion Pool: available techniques Pool.Map () and Pool.Apply () follows the conventions of traditional Python methods map() and apply()ensuring that all returned values are arranged in exactly the order in which they are input. However, these methods block their main program until the process completes processing. As a consequence, the class Pool also has a function map_async () and apply_async () to better serve the simultaneous processing and parallelism.

Determining the value of the current process, waiting and terminating processes

The class Process provides a number of ways to more easily interact with processes in a co-processing program. In this section we will study the control parameters of various processes by determining the value of the current process, waiting and terminating the process.

Determining the value of the current process

Working with processes is sometimes very complex and therefore requires significant debugging. One of the main methods for debugging a program is to identify the process that encountered the error. As a reminder, in our previous countdown example, we passed a parameter name to our function count_down () to determine where each process is located during a particular countdown.

However, there is no need for each object to Process have a certain parameter name (with some default value) that can be changed. The naming process is the best way to leave executable processes in the eye rather than pass a certain identifier to its target function on its own (as we did before), especially in applications with different types of processes running at the same time. One of the powerful functions provided by the module multiprocessingis its method current_process(), which will return the object Process that is currently being executed anywhere in the program. This is another way to efficiently and effortlessly track running processes.

Let's take a closer look at this with some example. Browse to the file Chapter06/example2.pyshown in the following code:

```
# Chapter06/example2.py
From multiprocessing import Process, current_process
Import time

def f1():
    pname = current_process().name
    print('Starting process %s...' % pname)
    time.sleep(2)
    print('Exiting process %s...' % pname)

def f2():
    pname = current_process().name
    print('Starting process %s...' % pname)
    time.sleep(4)
    print('Exiting process %s...' % pname)
if __name__ == '__main__':
    p1 = Process(name='Worker 1', target=f1)
    p2 = Process(name='Worker 2', target=f2)
    p3 = Process(target=f1)

    p1.start()
    p2.start()
    p3.start()

    p1.join()
    p2.join()
    p3.join()
```

In this example, we have two dummy functions, f1()and f2()each of which prints the name of the process that performed this function before and after sleep for a given period of time. In our main program, we initialized three separate processes. We entitled the first two Worker 1 and Worker 2, and the last one, we deliberately left blank to provide it with a default name (i.e., 'Process-3'). After running this script, you should get output similar to this:

```
> python example2.py
Starting process Worker 1...
Starting process Worker 2...
Starting process Process-3...
Exiting process Worker 1...
Exiting process Process-3...
Exiting process Worker 2...
```

We may find that our current_process () successfully helped us access the correct process that performed all the functions and our third process, which was assigned the default name by default Process-3. Another way to track all the processes running in your program is to view the corresponding individual process identifiers using the module os. Let's look at a modified example from a file Chapter06/example3.py, as shown in the following code:

```
# Chapter06/example3.py
from multiprocessing import Process, current_process
import time
import os

def print_info(title):
    print(title)

    if hasattr(os, 'getppid'):
        print('Parent process ID: %s.' % str(os.getppid()))

    print('Current Process ID: %s.\n' % str(os.getpid()))

def f():
    print_info('Function f')

    pname = current_process().name
    print('Starting process %s...' % pname)
    time.sleep(1)
    print('Exiting process %s...' % pname)
```

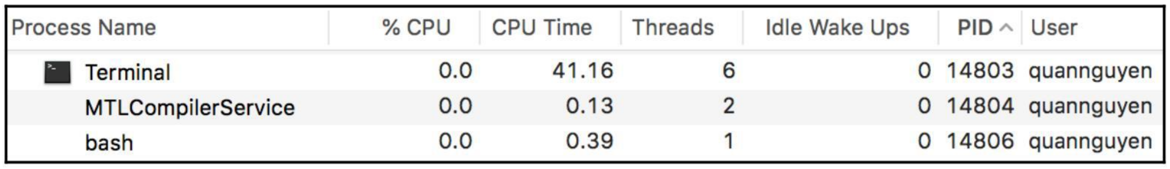
```
if __name__ == '__main__':
    print_info('Main program')

    p = Process(target=f)
    p.start()
    p.join()

    print('Done.')
```

Our focus for this example is a function print_info()that applies a function os.Getpid () and OS.Getppid () to identify the current process using its process ID. In particular, it os.Getpid () returns the process identifier itself, and the OS.Getppid () (which is available only on Unix systems) returns the value of the identifier of its parent process. Below is my conclusion after running this script:

```
> python example3.py
Main program
Parent process ID: 14806.
Current Process ID: 29010.

Function f
Parent process ID: 29010.
Current Process ID: 29012.

Starting process Process-1...
Exiting process Process-1...
Done.
```

The value of the process identifier can vary from system to system, but their relative interaction should remain the same. In particular, in my conclusion, we can see that while the identifier for the main program was 29010, the identifier of its parent process was value

14806. Using Activity Monitor, I performed the check in a different way and connected it to my Terminal and Bash profile, which makes sense, since I ran this Python script from my Terminal. In the pix below, you can see the displayed results:



Process Name	% CPU	CPU Time	Threads	Idle Wake Ups	PID ^	User
Terminal	0.0	41.16	6	0	14803	quannguyen
MTLCompilerService	0.0	0.13	2	0	14804	quannguyen
bash	0.0	0.39	1	0	14806	quannguyen

Screenshot of Activity Monitor for checking PID from an alternative source

In addition to our main program, we can also call print_info()inside our function f(), for which the process identifier had a value 29012. We can also find that the process that started the execution of our function f() is actually our main process, the identifier of which was the value 29010.

Process pending

Often, we would like to wait until the completion of all our simultaneous processing processes before moving on to the next section of our program. As mentioned earlier, a class Process from a module multiprocessing provides an appropriate module join () for implementing some way of waiting until some process completes its tasks and exits.

However, sometimes developers want to implement processes that run in the background and do not block their main program to exit. Such a prescription is usually applied when there is no easy way for the main program to tell whether it is correct to interrupt a certain process at a given time, or when exiting the main program without completing its executor does not affect the final result.

Such processes are called daemon processes . The class Process also provides some simple parameter for determining whether a certain process will be a daemon using the corresponding attribute daemon, which takes a Boolean value. The default value for such an attribute daemonis False, therefore, setting it to a value True will turn such a process into a kind of daemon. Let's take a closer look at this using the example from the file Chapter06/example4.pyshown in the following code:

```
# Chapter06/example4.py
from multiprocessing import Process, current_process
import time

def f1():
    p = current_process()
    print('Starting process %s, ID %s...' % (p.name, p.pid))
    time.sleep(4)
    print('Exiting process %s, ID %s...' % (p.name, p.pid))

def f2():
    p = current_process()
    print('Starting process %s, ID %s...' % (p.name, p.pid))
    time.sleep(2)
    print('Exiting process %s, ID %s...' % (p.name, p.pid))

if __name__ == '__main__':
    p1 = Process(name='Worker 1', target=f1)
    p1.daemon = True
    p2 = Process(name='Worker 2', target=f2)

    p1.start()
    time.sleep(1)
    p2.start()
```

In this example, we have a function with a long lifespan (presented f1(), in which the sleep period is 4seconds) and a faster function (which is presented f2() with a sleep period of only 2seconds). We also have two separate processes, which are listed in the following list:

A process p1that is a daemon process that is assigned to execute f1().

A process p2that is a daemon process that is assigned to execute f2().

In our main program, we start both processes without calling the corresponding method join()in any of them at the very end of this program. Since the process p1has a long lifespan, it most likely will not complete its execution before it completes p2(which is the faster process of the two). We also know what p1a daemon process is, so our program must exit before it completes execution. After running this Python script, your output should look something like this:

```
> Python example4.py

Starting process Worker 1, ID 33784...

Starting process Worker 2, ID 33788...

Exiting process Worker 2, ID 33788...
```

And again, even though the process identifiers may differ when you run this script yourself, the general format for the output will be the same. As we can see, the resulting conclusion is consistent with the fact that we have been discussing: the two processes, p1and p2have been initialized and are running our main program, with this very program has stopped the execution after the release of the corresponding process, not a demon without waiting for the daemon process completes its execution .

This possibility of terminating your main program without the obligation to wait for a specific task, which is a processing daemon, is actually extremely useful. However, we sometimes wish to wait for the daemon to execute the prescribed number of times before exiting; Thus, if the corresponding instruction of this program allows a certain waiting time before the execution of our process, we could end some process that is potentially a demon, instead of prematurely terminating it.

Such a combination of the daemon process and the corresponding method join() from the module multiprocessingcan help us implement such a solution, in particular, given that while our method join()blocks the execution of our program indefinitely (or at least until the corresponding task is completed), it is also possible to transfer a certain timeout parameter to determine the required number of seconds to wait before exiting. Let's look at some modified version of our previous example in Chapter06/example5.py. With the same features f1()and f2()in its following example, we change the way in which we handle your daemon process in the main program:

```
# Chapter06/example5.py
if __name__ == '__main__':
    p1 = Process(name='Worker 1', target=f1)
    p1.daemon = True
    p2 = Process(name='Worker 2', target=f2)

    p1.start()
    time.sleep(1)
    p2.start()

    p1.join(1)
    print('Whether Worker 1 is still alive:', p1.is_alive())
    p2.join()
```

Instead of stopping the daemon without waiting for its process, in this example we call the method join()for both processes: we give one second forp1 complete, while we block our main program until it finishes p2. If it p1does not complete the execution after this one second, our main program will continue to execute what is left in it, while we see that p1- or Worker 1- is still alive. After executing this Python script, your output will look like this:

```
> python example5.py
Starting process Worker 1, ID 36027...
Starting process Worker 2, ID 36030...
Whether Worker 1 is still alive: True
Exiting process Worker 2, ID 36030...
```

We see that p1in fact it is still alive, while the main program proceeded further after waiting for one second.

Process termination

The method terminates () from the class under discussion multiprocessing.The process offers some way to quickly terminate the process. When this method is called, the exit handlers that complete positions or similar resources prescribed in the corresponding class Processor some overlapping class will not be executed . However, the processes of the descendants will not be stopped. Such processes are called orphaned processes.

Although the termination of processes is sometimes viewed with disapproval, such an attitude is sometimes inevitable, since some processes communicate with interprocess communication resources, such as locks, semaphores, pipelines, or queues and forced shutdown of such processes will cause the destruction of such a resource or its inaccessibility to other processes. However, if the processes in your program never interact with the resources mentioned above, such a method is terminated () quite useful, in particular, when a certain process seems not responding, or is involved in a deadlock.

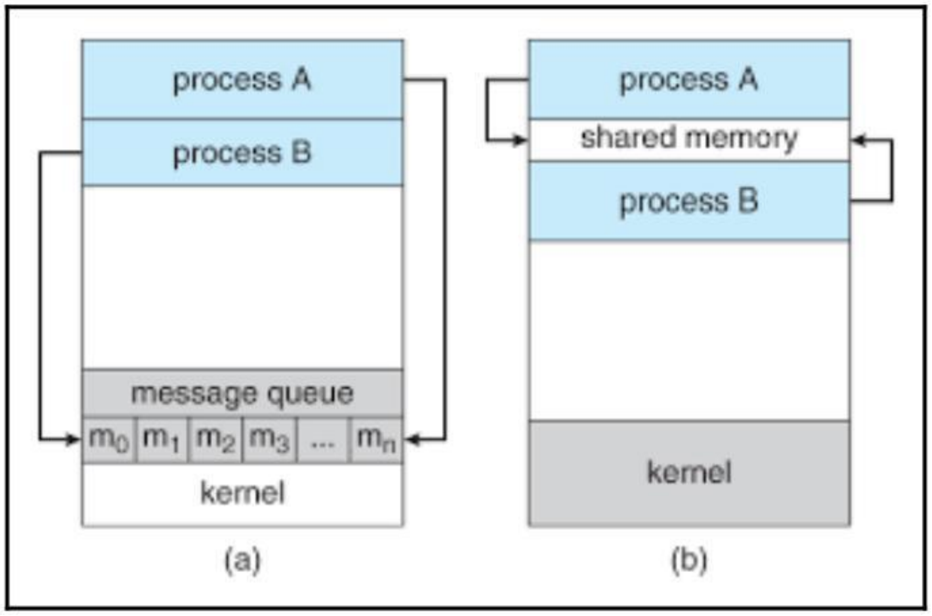
One of the points that should be noted when applying the method terminate()is that even though the corresponding object is Process effectively destroyed after calling this method, it is important that you also call this method as well join(). Since the state of the aliveobject is Process sometimes not updated immediately after the corresponding method terminate() , this practice gives its system the ability to update itself in the background in response to the cessation of its processes.

Process interaction

While locks are one of the common synchronization primitives that are used to interact between threads, pipelines and queues are the main way that various processes interact. In particular, they provide messaging capabilities to facilitate interaction between processes — pipelines for interaction between two processes, and queues for multiple suppliers and consumers.

In this page, we will study the use of queues, in particular the class Queue from the module under discussion multiprocessing. The existing implementation of the class Queueactually saves both the thread and the process, and we have already seen the use of queues in Chapter 3, Working with threads in Python. An object Queue can pass any pickling Python objects; in this section, we will use queues to send messages back and forth between processes.

Using a message queue to communicate between processes is preferable to owning shares resources because if certain processes mismanage and destroy shared memory and resources while these resources are to be shared, then there will be numerous undesirable and unpredictable consequences. However, if a process fails to correctly process its message; other messages in this queue will remain untouched. The diagram below shows the differences in architecture between the use of a queue and shared resources (in particular, memory) during the interaction between processes:



An example of a solution involved in using a message queue and sharing resources between processes

Sending a message to an individual artist

Before we dive into the abyss of our Python example code, first we need to discuss in particular how we use an object Queue in our application with many processes. Suppose we have a class worker that does the heavy computing and does not require significant resources for sharing and interaction. Nevertheless, these instances of the performers still require the possibility of receiving information from time to time in the course of their execution.

This is where a certain line comes into play: when we put all our performers in a certain line. At the same time, we will also have a number of processes that have completed initialization, each of which will go through this queue and process one executor. Looking back at the earlier diagram, we can find that there are two separate processes that continue to snatch and process messages from the queue.

In the object Queuewe will use two main methods, which are listed in the following list:

get(): This method returns the next element in the called object. Queue

put(): The data method adds the parameter passed to it as an additional element to the called object itself Queue

Let's look at a sample script showing an example of using some queue in Python. Navigate to your file Chapter06/example6.pyand open it, as shown in the code below:

```
# Chapter06/example6.py
import multiprocessing

class MyWorker():
    def __init__(self, x):
        self.x = x

    def process(self):
        pname = multiprocessing.current_process().name
        print('Starting process %s for number %i...' % (pname, self.x))

def work(q):
    worker = q.get()
    worker.process()

if __name__ == '__main__':
    my_queue = multiprocessing.Queue()

    p = multiprocessing.Process(target=work, args=(my_queue,))
    p.start( )

    my_queue.put(MyWorker(10))
    my_queue.close()
    my_queue.join_thread()
    p.join()

    print('Done.')
```

In this scenario, we have a certain class MyWorkerthat receives a parameter in some number and performs some calculation based on it (in this case, this is just printing this number). In our main function, we initialize some object Queue from the module multiprocessing and add some object MyWorkerwith the value of the number 10in it. We also have a corresponding functional work () that, when called, will receive the very first element from this queue and process it. Finally, we have a process whose task is to call such a function work().

This structure is designed to transmit a message — in this case, an object MyWorker— into one separate process. The main program, then waits for the completion of such a process. After running this scenario, your output should look like this:

```
> python example6.py
Starting process Process-1 for number 10...
Done.
```

Messaging between multiple artists

As mentioned earlier, our goal is to have a structure in which there are several processes that constantly launch executors from a certain queue, and when a certain process completes the execution of one executor, then it picks up a new one. To do this, we will use a certain subclass Queue with a name JoinableQueuethat will provide its additional methods task_done()and join(), as defined in this list:

task_done(): This method tells our program that the called object is JoinableQueuecomplete.

join(): This method blocks until all elements in its called object JoinableQueueare processed.

Now the main goal here, again, is to have some kind of object JoinableQueuecontaining all the tasks that must be completed - we call it our task queue - as well as a certain number of processes. As long as there are elements (messages) in our task queue, the existing processes will in turn be executed for such elements. We will also have a certain object Queue for saving all the results returned from executable processes - we will call it the result queue.

Let's Chapter06/example7.Pymove on to our file and switch to the consideration of the class Consumer and class Task, which are displayed in the following code:

As we said earlier, in our main program, we created a certain task queue and a result queue. We also created some list of objects Consumer and launched them all; the total number of processes created corresponds to the total number of CPUs available in our system. Further, from a list of input data that require heavy calculations in our class Task, we initialize a certain object Task for each input data and put them all in the task queue. At this point, our processes - our facilities Consumer- will begin to perform these tasks.

Finally, at the very end of our main program, we call join()up tasks in our queue to ensure that all elements have been completed and print the results, bypassing the results queue. After executing this scenario, your conclusion should be like this:

Everything seems to work, but if we take a closer look at the messages that our processes printed, we will notice that most of the tasks were performed either Consumer-2, or Consumer-3, but Consumer-4completed only one task, whileConsumer-1 refused at all. What's going on here?

Essentially, when one of our customers — let’s say Consumer-3— has completed a task, he tries to find another task for execution right after that. In most cases, it will gain priority over other consumers since it has already been executed by our main program. Therefore, while Consumer-2and Consumer-3constantly completed execution of their tasks and grab for execution other tasks Consumer-4could "squeeze" out of himself it only once, while Consumer-1 refused to do it at all.

Running this scenario again and again, you will notice a similar trend: only one or two consumers fulfill most of the tasks, while others refuse this. This state of affairs is undesirable for us, since this program does not use all available processes that were created at the very beginning of this program.

To solve this problem, a certain technique was developed to stop consumers from immediately receiving the next available element in our task queue, called the poison pill . The main idea is that after setting our real tasks to the existing task queue, we also add some kind of dummy task that contains the values “stop” and which will make the current consumer linger and allow other consumers to get the next available items in this task queue first; this clarifies the name "poisonous pill."

To implement this technique, we need to add tasks special objects to our value in our main program, one per consumer. In addition, our class Consumer also requires the implementation of special logic for processing such special objects. Let's look at our file example8.py (some modification of the version of our previous example containing the corresponding implementation of the poison pill technique), in particular in our class Consumer and in our main program, as shown in the following code:

Our class Taskremains the same as in our previous example. We can find that: our poison pill is the value None: in our main program, we add to the None values a number equal to the total number of consumers that we generated in our task queue; in its class Consumer, if the current task to be executed contains a value None, then an object of this class will print a message indicating the poisonous pill, it will call task_done()and complete the execution.

Run this script, your output should be like this:

```
> python example8.py
Spawning 4 consumers...
Consumer-1 processing task: Checking if 2 is a prime or not.
Consumer-2 processing task: Checking if 36 is a prime or not.
Consumer-3 processing task: Checking if 101 is a prime or not.
Consumer-4 processing task: Checking if 193 is a prime or not.
Consumer-1 processing task: Checking if 323 is a prime or not.
Consumer-2 processing task: Checking if 513 is a prime or not.
Consumer-3 processing task: Checking if 1327 is a prime or not.
Consumer-1 processing task: Checking if 100000 is a prime or not.
Consumer-2 processing task: Checking if 9999999 is a prime or not.
Consumer-3 processing task: Checking if 433785907 is a prime or
not.
Exiting Consumer-1...
Exiting Consumer-2...
Exiting Consumer-4...
Exiting Consumer-3...
Result: 2 is a prime number.
Result: 36 is not a prime number.
Result: 323 is not a prime number.
Result: 101 is a prime number.
Result: 513 is not a prime number.
Result: 1327 is a prime number.
Result: 100000 is not a prime number.
Result: 9999999 is not a prime number.
Result: 193 is a prime number.
```

```
Result: 433785907 is a prime number.
Done.
```

This time, along with the fact that we are seeing messages about toxic pills being printed, our output also reflects a much better distribution as to which consumers perform which tasks.

Findings

In the field of computer science, a process is an instance of a special computing program or software that is executed by an existing operating system. A certain process contains both the program code and its current activities and interAction with other entities. To implement access and separation of memory or other resources within the same process, more than one thread can be implemented, while different processes do not interact in this way.

In the context of joint processing and parallelism under discussion, the multiplicity of processes refers to the specific execution of simultaneous processes of a certain operating system, in which each process is executed in a separate CPU, which is completely different from a separate process executed at a given point in time. The multiprocessing Python module provides a powerful and flexible API for spawning processes and managing them in applications with many processes. It also allows sophisticated technologies for interprocess communication through its class Queue.

CHAPTER 9

Python tutorial: Cognitive Services API calls in the Azure Search indexing pipeline

In this tutorial, you learn the programming data enrichment mechanism in Azure Search using cognitive skills. Capabilities are supported by natural language processing (NLP) and image analysis features in cognitive services. You can extract text and text representations of an image or scanned document file through skill composition and configuration. You can also identify language, assets, key phrases and more. The result has rich additional content in an Azure Search index created with AI riches in an indexing process line.

The output is a full-text searchable directory on Azure Search. You can enhance the index with other standard features such as thesauri, scoring profiles, parsers, and filters.

This tutorial runs on a free service, but the number of free transactions is limited to 20 documents per day. If you want to run this tutorial multiple times on the same day, use a smaller set of files so you can follow more runs.

Note

After you expand the scope by increasing the processing frequency, adding more documents, or adding more AI algorithms, you must add a billable cognitive Services resource . Accrued fees for calling APIs in cognitive services and extracting images as part of the document decoding phase within Azure Search. There is no charge for extracting text from documents.

Execution of built-in capabilities is charged at the price of Pay As You Use existing cognitive Services. Image extraction pricing is described on the Azure Search pricing page.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

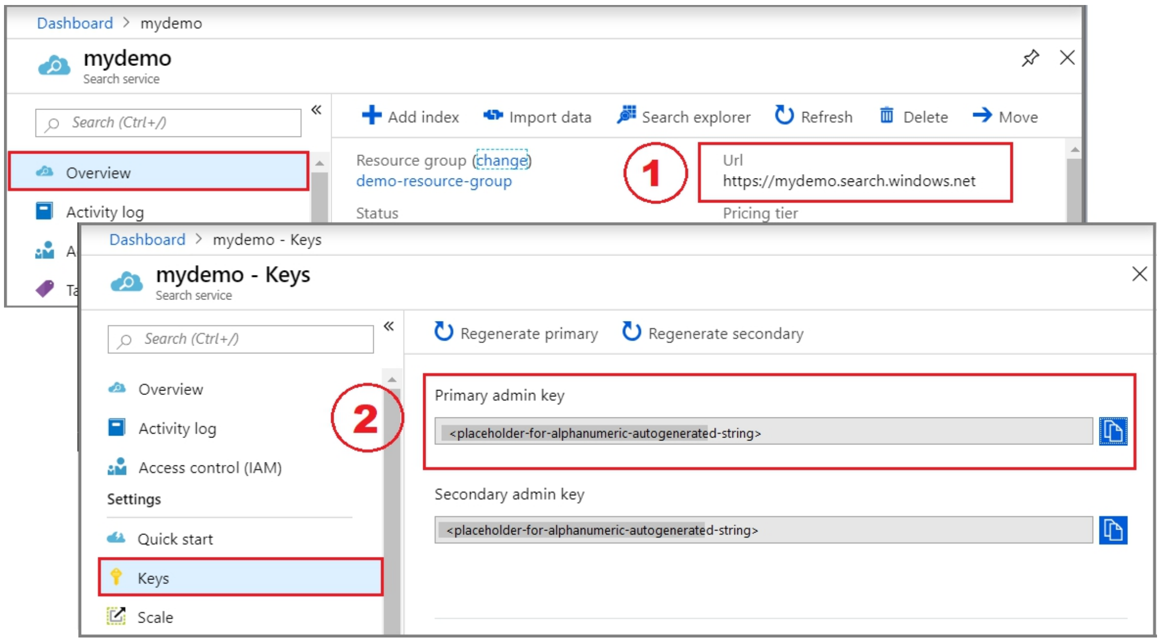
The following services, tools, and data are used in this tutorial.

- Create an Azure storage account to store the sample data . Make sure that the storage account is located in the same zone as Azure Search.
- Anaconda 3. x, which provides Python 3. x and Jupiter notebooks .
- The sample data consists of a small set of files of different types.
- Create an Azure Search service on your current subscription , or find an existing service . You can use a free service for this tutorial.

Get key and URL

To interact with your Azure Search service, you need the service URL and the access key. A search service that includes both is created. Therefore, if you added the Azure Search service to your subscription, follow these steps to get the required information:

1. Sign in to Azure Portal and get the URL on the search service overview page. The sample may appear as an endpoint . https://mydemo.search.windows.net
2. In Settings > Keys , obtain an administrator key for full rights to the service. There are two interchangeable administrator keys for business continuity, which you need to purchase one. You can use the primary or secondary key in requests to add, modify, and delete objects.

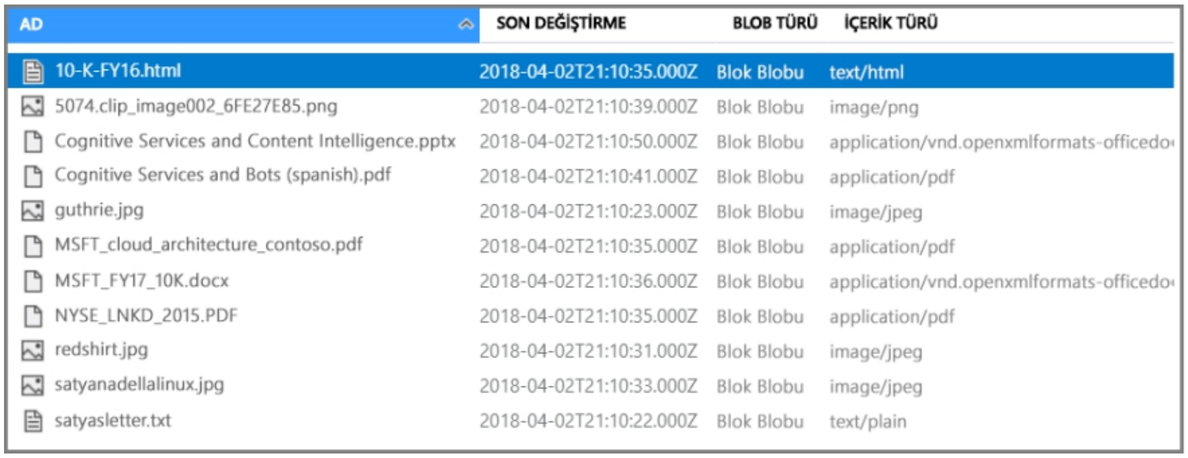


All requests require an API key for each request sent to your service. A valid key, per request, creates trust between the application that sends the request and the service that processes it.

Preparing sample data

The enrichment process line pulls from Azure data sources. The source data must come from a supported data source type of the Azure Search indexer . Azure Table Storage is not supported for cognitive search. In this exercise, we use blob storage to show multiple content types.

1. Log in to Azure Portal , go to your Azure storage account, click Blobs, and then click + Container .
2. Create a blob container that contains the sample data . You can set the public access level to any of its current values.
3. After the container is created, open the file and select Upload on the command bar to upload the sample files that you downloaded in a previous step.



AD	SON DEĞİŞTİRME	BLOB TÜRÜ	İÇERİK TÜRÜ
10-K-FY16.html	2018-04-02T21:10:35.000Z	Blok Blobu	text/html
5074.clip_image002_6FE27E85.png	2018-04-02T21:10:39.000Z	Blok Blobu	image/png
Cognitive Services and Content Intelligence.pptx	2018-04-02T21:10:50.000Z	Blok Blobu	application/vnd.openxmlformats-officedo
Cognitive Services and Bots (spanish).pdf	2018-04-02T21:10:41.000Z	Blok Blobu	application/pdf
guthrie.jpg	2018-04-02T21:10:23.000Z	Blok Blobu	image/jpeg
MSFT_cloud_architecture_contoso.pdf	2018-04-02T21:10:35.000Z	Blok Blobu	application/pdf
MSFT_FY17_10K.docx	2018-04-02T21:10:36.000Z	Blok Blobu	application/vnd.openxmlformats-officedo
NYSE_LNKD_2015.PDF	2018-04-02T21:10:35.000Z	Blok Blobu	application/pdf
redshirt.jpg	2018-04-02T21:10:31.000Z	Blok Blobu	image/jpeg
satyanadellalinux.jpg	2018-04-02T21:10:33.000Z	Blok Blobu	image/jpeg
satyasletter.txt	2018-04-02T21:10:22.000Z	Blok Blobu	text/plain

4. After the sample files are loaded, obtain a connection string and container name for your Blob store. You can do this by going to your storage account in the Azure portal. Click Access keys , and then copy the connection string field.

The connection string will have the following format:**DefaultEndpointsProtocol=https;AccountName=<YOUR-STORAGE-ACCOUNT-NAME>;AccountKey=<YOUR-STORAGE-ACCOUNT-KEY>;EndpointSuffix=core.windows.net**

Keep the connection string handy. This will be needed in a future step.

There are many other ways to specify the connection string, such as providing a shared access signature.

Create a Jupyter notebook

Note:

This article demonstrates how to create a data source, index, indexer, and skill using a series of Python scripts. Go to the Azure-Search-Python-Samples repository to download the full notebook sample .

Use the Anaconda explorer to start Jupyter Notebook and create a new Python 3 Notepad.

Azure Search Connect

In your notebook, run this script to load the libraries used to work with JSON and to add HTTP requests to the formula.

Python:

```
Import json
Import requests
From pprint import pprint
```

Then, define the names for the data source, index, indexer, and skill. Run this script to set the names of this tutorial.

Python:

```
# Define the names for the data source, skillset, index and indexer
datasource_name = "cogsrch-py-datasource"
skillset_name = "cogsrch-py-skillset"
index_name = "cogsrch-py-index"
indexer_name = "cogsrch-py-indexer"
```

Hint

In a free service, you are limited to three indexes, indexers, and data sources. In this tutorial, one of each is created. Make sure you have room to create new objects before continuing.

In the following script, replace the placeholders for your search service (-SEARCH-SERVICE-NAME) and your administrator API key (-ADMIN-API-KEY), and then run to set the search service endpoint.

Python:

```
# Setup the endpoint
```

```
Endpoint = 'https://<YOUR-SEARCH-SERVICE-NAME>.search.windows.net/'
Headers = {'Content-Type': 'application/json',
        'api-key': '<YOUR-ADMIN-API-KEY>'}
Params = {
   'api-version': '2019-05-06'
}
```

Create A Data Source

Now that your services and source files are prepared, you can now start compiling the components of your indexing process line. Start with a data source object that tells Azure Search how to import the external source data.

In the following script, replace the -BLOB-RESOURCE-CONNECTION-STRING placeholder with the Blobun connection string that you created in the previous step. Then, run the script to create a named data source. Cogsrch-pay-datasource

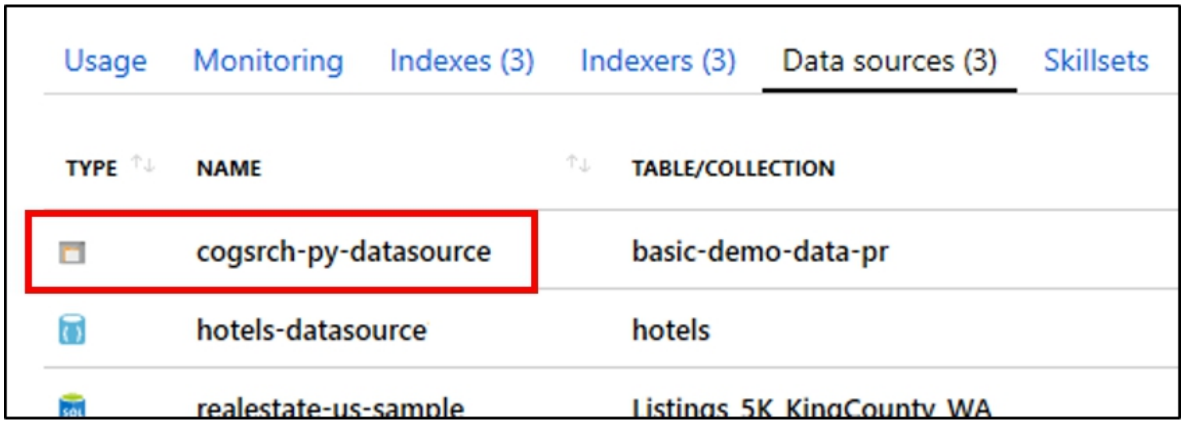
Python:

```
# Create a data source
DatasourceConnectionString = "<YOUR-BLOB-RESOURCE-CONNECTION-STRING>"
datasource_payload = {
   "name": datasource_name,
   "description": "Demo files to demonstrate cognitive search capabilities.",
   "type": "azureblob",
   "credentials": {
      "connectionString": datasourceConnectionString
```

```
    } ,
    "container": {
        "name": "basic-demo-data-pr"
    }
}
r = requests.put(endpoint + "/datasources/" + datasource_name,
                  data=json.dumps(datasource_payload),
headers=headers, params=params)
print(r.status_code)
```

The request must return status, status 201, which confirms success.

On the Azure portal search service dashboard page, verify that cogsrch-Copy-DataSource appears in the data sources list. Click Refresh to update the page.



TYPE	NAME	TABLE/COLLECTION
	cogsrch-py-datasource	basic-demo-data-pr
	hotels-datasource	hotels
	realestate-us-sample	Listings_5K_KingCounty_WA

Creating A Skill Set

In this step, you will define a series of enrichment steps to apply to your data. You can name each enrichment step as a skill and the set of enrichment steps as a skill set . This tutorial uses built-in cognitive skills for skill :

- Language Detection to define the language of the content .
- Split Text to separate large content into smaller heaps before calling key phrase extraction . The key phrase extraction accepts an entry of 50,000 or fewer characters. To comply with this limit, several sample files must be split.
- Entity recognition to extract the names of organizations from the content contained in the Blob container .
- Extract Key Expression to pull top key phrases .

Python script

Cogsrch-pay-skillsetRun the following script to create a called skill.

Python

```
# Create a skillet
skillset_payload = {
    "Name": skillset_name,
    "Description":
    "Extract entities, detect the language and extract key-phrases",
    "Skills":
    [
        {
            "@odata.type":
"#Microsoft.Skills.Text.EntityRecognitionSkill",
            "Categories": ["Organization"],
            "defaultLanguageCode": "en",
            "Inputs": [
                {
                    "Name": "text", "source": "/document/content"
                }
            ],
            "outputs": [
                {
                    "name":            "organizations",            "targetName":
"organizations"
                }
            ]
        },
        {
            "@odata.type":
"#Microsoft.Skills.Text.LanguageDetectionSkill",
            "inputs": [
                {
                    "name": "text", "source": "/document/content"
                }
            ],
            "outputs": [
                {
                    "name": "languageCode",
                    "targetName": "languageCode"
                }
            ]
        },
        {
            "@odata.type": "#Microsoft.Skills.Text.SplitSkill",
            "textSplitMode": "pages",
            "maximumPageLength": 4000,
            "inputs": [
                {
                    "name": "text",
                    "source": "/document/content "
                },
                {
                    "name": "languageCode",
                    "source": "/document/languageCode"
                }
            ],
            "outputs": [
                {
                    "name": "textItems",
                    "targetName": "pages"
                }
            ]
        },
        {
            "@odata.type":  "#Microsoft.Skills.Text.KeyPhraseExtraction
Skill",
            "context": "/document/pages/*",
            "inputs": [
                {
                    "name": "text", "source": "/document/pages/*"
                },
                {
```

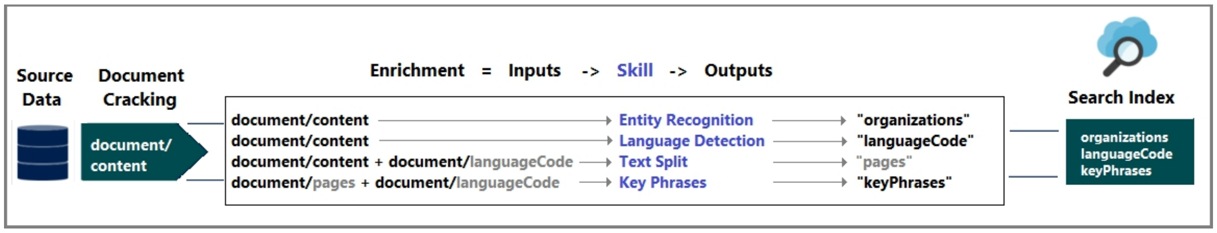
```
                "name":         "languageCode",         "source":
"/document/languageCode"
            }
        ],
        "outputs": [
            {
                "name": "keyPhrases" ,
                "targetName": "keyPhrases"
            }
        ]
    }
  ]
}

r = requests.put(endpoint + "/skillsets/" + skillset_name,
               data=json.dumps(skillset_payload),   headers=headers,
params=params)
print(r.status_code)
```

The request must return status status 201, which confirms success.

The ability to extract the key phrase is applied to each page. By setting its context , you run this richer rich for each member of the document / pages array (for each page in the document). "Document/pages/*"

Each skill is executed in the content of the document. During processing, Azure Search decodes each document to read content from different file formats. The text in the source file is placed in a field for each document . Therefore, set the input to . Content "/document/content"

The graphical representation of the skill set is shown below.



Source Data
Document Cracking
document/ content
Enrichment = Inputs -> Skill -> Outputs
document/content → Entity Recognition → "organizations"
document/content → Language Detection → "languageCode"
document/content + document/languageCode → Text Split → "pages"
document/pages + document/languageCode → Key Phrases → "keyPhrases"
Search Index
organizations
languageCode
keyPhrases

Outputs can be matched in both cases as well as in a directory used as an input to a downstream skill, or both in relation to the language code. A language code in the directory is useful for filtering. As an introduction, the language code is used by text analysis skills to report grammar rules related to word division.

Indexing

In this section, you define the Index schema by specifying the fields to be included in the searchable index and setting the search attributes for each field. Fields have a type and can take attributes that determine how the field is used (searchable, sortable, and so on). The field names in an index need not be exactly the same as the field names in the source. In a next step, you add field mappings in the indexer to link the source-to-target fields. For this step, define the index using the field naming conventions for your search application.

The following fields and field types are used in this study:

domain names:	ID	content	languageCode	keyphrases	Organizations
field-types:	edm.string	edm.string	edm.string	List <edm.string>	List <edm.string>

cogsrch-py-indexRun this script to create the named directory.

# Create an index

```
index_payload = {
    "name": index_name,
    "fields": [
        {
            "name": "id",
            "type": "Edm.String",
            "key": "true",
            "searchable": "true" ,
            "filterable": "false",
            "facetable": "false",
            "sortable": "true"
        },
        {
            "name": "content",
            "type": "Edm.String",
            "sortable": "false",
            "searchable": "true",
            "filterable": "false",
            "facetable": "false"
        },
        {
            "name": "languageCode",
            "type": "Edm.String",
            "searchable": "true",
            "filterable": "false",
            "facetable": "false"
        },
        {
            "name": "keyPhrases",
            "type": "Collection(Edm.String)",
            "searchable": "true",
            "filterable": "false",
            "facetable": "false"
        },
        {
            "name": "organizations" ,
            "type": "Collection(Edm.String)",
            "searchable": "true",
            "sortable": "false",
            "filterable": "false",
            "facetable": "false"
        }
    ]
}
r = requests.put(endpoint + "/indexes/" + index_name,
data=json.dumps(index_payload),                    headers=headers,
params=params)
print(r.status_code)
```

The request must return status status 201, which confirms success.

Creating An Indexer, Mapping Fields, And Executing Transformations

So far, you've created a data source, skills and an index. These three components become part of the indexer, which pulls each part together into a single multistage process . To link these objects together in an indexer, you must define the field mappings.

- If Field Mappings are processed before the Skill element, the source fields are mapped from the data source to the target fields in an index. If the field names and types are the same on both ends, no mapping is required.
- OutputFieldMappings is processed after the Skill and references sourceFieldNames that do not exist until you unravel or enrich the document. TargetFieldName is a field in the directory.

In addition to attaching inputs to the outputs, you can use field mappings to flatten the data structures. For more information. Mapping enriched fields to a searchable directory .

Cogsrch-pay-indexerRun this script to create a named indexer.

```
# Create an indexer

indexer_payload = {

    "name": indexer_name,

    "dataSourceName": datasource_name,

    "targetIndexName": index_name,

    "skillsetName": skillset_name,

    "fieldMappings": [

        {

            "sourceFieldName": "metadata_storage_path",

            "targetFieldName": "id",

            "mappingFunction":

            {"name": "base64Encode"}

        },
        {
            "sourceFieldName": "content",
            "targetFieldName": "content"
        }
    ],
    "outputFieldMappings":
    [
        {
            "sourceFieldName": "/document/organizations",
            "targetFieldName": "organizations"
        },
        {
            "sourceFieldName": "/document/pages/*/keyPhrases/*" ,
            "targetFieldName": "keyPhrases"
        },
        {
            "sourceFieldName": "/document/languageCode",
            "targetFieldName": "languageCode"
        }
    ],
    "parameters":
    {
        "maxFailedItems": -1,
        "maxFailedItemsPerBatch": -1,
        "configuration":
        {
            "dataToExtract": "contentAndMetadata",
            "imageAction": "generateNormalizedImages"
        }
    }
}
r = requests.put(endpoint + "/indexers/" + indexer_name,
                 data=json.dumps(indexer_payload),   headers=headers,
params=params)
print(r.status_code)
```

The request should return status code 201 quickly, but may take several minutes to complete. Although the data set is small, analytical capabilities such as image analysis are computationally intensive and time-consuming.

Use the Check Indexer status script in the next section to find out when the Indexer process is complete.

Hint

When an indexer is created, the process line is called. Access to data is displayed at this stage if there is a problem with the mapping of inputs and outputs or with the processing sequence. You may need to delete objects before you can rerun the process line with code or script changes. For more information. Reset and restart .

Explore The Request Body

By "maxFailedItems"setting its value to -1 , the script instructs the indexing engine to ignore errors during data import. This is useful because there are very few documents in the demo data source. For a larger data source, you set the value to a value that is greater than 0.

Also "dataToExtract":"contentAndMetadata"note the statement in the configuration parameters . This statement tells the indexer to extract content from different file formats and metadata for each file.

When the content is extracted, you imageActioncan set the value to extract the text from the images in the data source . When used in conjunction with the configuration, OCR capability, and text concatenation, it tells the indexer to extract text from images (for example, stops the traffic from logging on to the word "Stop") and tells the embed as part of the content area. This behavior applies to embedded images in documents (consider an image of an image in a PDF) and to images that reside in the data source (for example, a JPG file). "imageAction":"generateNormalizedImages"

Check The Indexer Status

Once the indexer is defined, it will run automatically when you submit the request. Depending on the cognitive skills you define, indexing may take longer than you expect. Run the following script to see if Indexer processing is complete.

Python

```
# Get indexer status
r = requests.get(endpoint + "/indexers/" + indexer_name +
                 "/status", headers=headers, params=params)
pprint(json.dumps(r.json(), indent=1))
```

In the response, follow "lastResult" for the "status" and "End Time" values. Run the script periodically to check the status. When the indexer completes, the status is set to "success", "End Time" is specified, and the response will include all errors and warnings that occurred during the enrichment.

Alerts are global to some source file and skill combinations and do not always indicate a problem. In this tutorial, warnings are harmless. For example, one of the JPEG files that does not contain text shows a warning on this screen image.

Querying Your Directory

After the indexing process is complete, run queries that return the contents of the individual fields. By default, Azure Search returns the first 50 results. The sample data is small for the default value to work correctly. However, when working with large data sets, you may need to include parameters in the query string to return more results. For instructions, see. Create a page of results in Azure Search .

As the validation step, query the index for all fields.

```
# Query the index for all fields
r = requests.get(endpoint + "/indexes/" + index_name,
                 headers=headers, params=params)
Print (Jason. dumps (r. Jason (), indent=1))
```

The results should look similar to the following example. The screenshot shows only part of the response.

The output is an index schema that contains the name, type, and attributes of each field.

organizations"*"Submit a second query to return all the contents of a single field, such as.

Python

# Query the index to return the contents of organizations

r = requests.get(endpoint + "/indexes/" + index_name +

"/docs?&search=*&$select=organizations", headers=headers, params=params)

Print (Jason. dumps (r. Jason (), indent=1))

The results should look similar to the following example. The screenshot shows only part of the response.

Repeat this exercise content, languageCode, key Phrases, and organizations. You can return multiple fields through using a comma-separated list . $select

Reset and restart

In the early stages of process line development, the most practical approach to design iterations is to delete objects from Azure Search and have your code rebuild them. Resource names are unique. When you delete an object, you can re-create it using the same name.

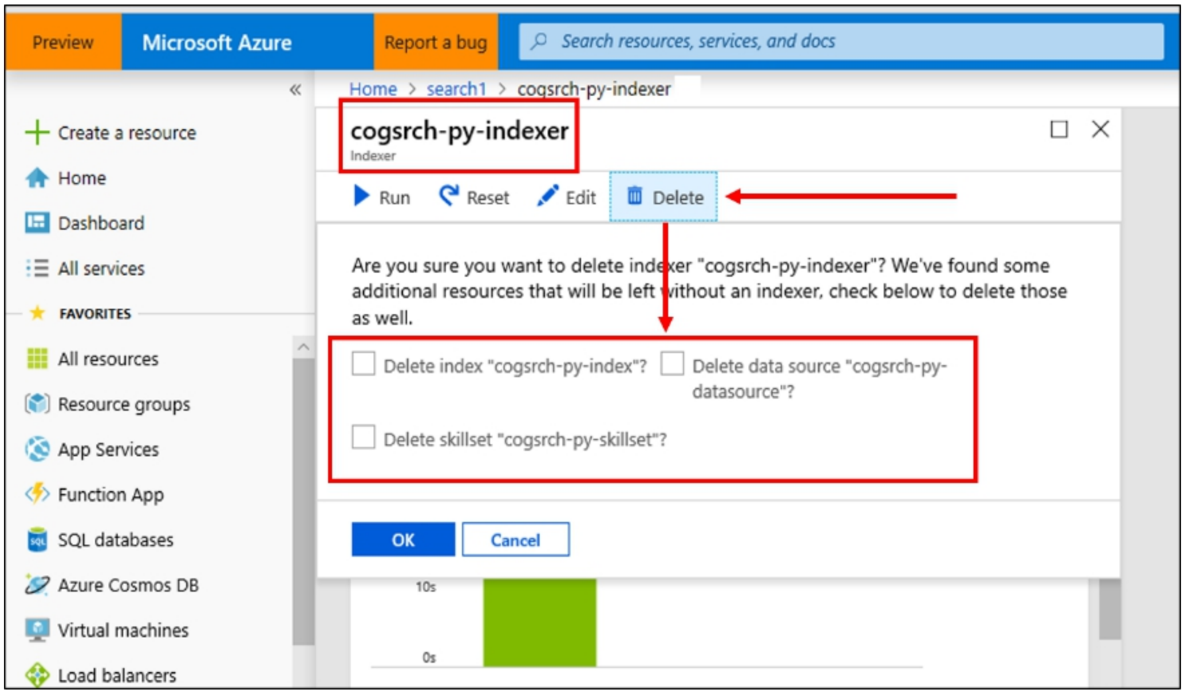
To re-index your documents with new definitions:

Remove persistent data by deleting the directory. Recreate the indexer on your service by deleting it.

Change the Skill and Index definitions.

To run the process line, re-create an index and indexer on the service.

You can use the portal to delete directories, indexers, and skills. When you delete the indexer, you can optionally selectively delete the index, skill, and data source at the same time.



Are you sure you want to delete indexer "cogsrch-py-indexer"? We've found some additional resources that will be left without an indexer, check below to delete those as well.

☐ Delete index "cogsrch-py-index"?
☐ Delete data source "cogsrch-py-datasource"?
☐ Delete skillset "cogsrch-py-skillset"?

You can also delete it by using a script. The following script will erase the skill we have created. You can easily change the request to delete the index, indexer, and data source.

Python:

```
# delete the skillset

r = requests.delete(endpoint + "/skillsets/" + skillset_name,

                    headers=headers, params=params)

pprint(json.dumps(r.json(), indent=1))
```

Packets

This tutorial demonstrates the basic steps to create an enriched indexing process line by creating component parts such as a data source, skill set, index, and an indexer.

Predefined abilities, Skill definitions and a way to chain skills through inputs and outputs are presented. You've learned that in order to route enriched values on the process line to a searchable directory in an Azure Search service, you need it in the indexer definition. OutputFieldMappings

Finally, you learn about testing the results and resetting the system for more iterations. You have learned that when queries against the index are edited, the output generated by the enriched indexing processing line is returned. This publication provides a mechanism for viewing internal structures (enriched documents created by the system). You also learn the Indexer status and which objects to delete before you run a process line again.

Cleaning Resources

The quickest way to clean up after a tutorial is complete is to delete the resource group that contains the Azure Search service and the Azure Blob service. When you place both services in the same group, you can permanently delete everything in it, including services and any stored content you create for this tutorial by deleting the resource group. In the portal, the resource group name is on the Overview page for each service.

Next Steps

Customize or expand the processing line with special skills. Creating a custom skill and adding it to a skill set allows you to add your own text or image analysis.

Automatic Machine Learning in Python Hyperparameter Setting

A complete walk using Bayesian optimization for automatic hyperparameter adjustment in Python

Setting machine learning hyperparameters is very laborious but very important because the performance of an algorithm can be highly dependent on the choice of hyperparameter. The machine moves away from the important steps of the learning pipeline, such as manual tuning, feature engineering, and interpretation of results. Grid and random searches are very convenient, but require long run times because they spend time evaluating areas that do not violate the search area. Increasingly, the hyperparameter setting is done by automated methods aimed at finding optimal hypermeters in less time using the most appropriate hyperparameters, without requiring manual effort beyond the initial setup of informed search .

Bayesian optimization, a model-based method for finding the minimum of a function, has recently been applied to machine learning hyperparameter setting, and results have shown that this approach may require less iteration than random search, while achieving better performance in the test set. What's more, there are now a number of Python libraries that simplify the Bayesian hyperparameter setting for any machine learning model.

In this guide, we will examine an example of a full Bayesian hyperparameter adjustment of a gradient incrementing machine using the Hyperopt library. In an earlier article, I explained the concepts behind this method, so we'll continue to apply it here. As with most machine learning topics, you don't need to understand all the details, but knowing the basic idea can help you use the technique more effectively!

Bayesian Optimization Methods

As a brief start, Bayesian optimization finds the value that minimizes an objective function by creating a proxy function (probabilistic model) based on the historical evaluation results of the target. The carrier is cheaper to improve than the target, so the next input values to be evaluated are selected by applying a criterion to the carrier (usually Expected Improvement). Bayesian methods use past evaluation results to select the next values to be evaluated randomly or from different ones in the grid search . The concept is this: Limit the expensive evaluation of the objective function by selecting the next input values based on what is good in the past.

In the case of hyperparameter optimization, the objective function is the validation error of a machine learning model that uses a series of hyperparameters. The aim is to find the hyperparameters that give the lowest error on the device. Verification set We hope these results are generalized to the test set. Evaluation of the objective function is expensive because it requires training the machine learning model with a particular set of hyperparameters. Ideally, we want a method that can do the search for the search field while limiting the assessment of poor hyperparameter options . The Bayesian hyperparameter setting uses a continuously updated probability model to “concentrate ere on promising hyperparameters based on past results.

Python Options

In Python, there are several Bayesian optimization libraries that differ in the algorithm for the proxy of the objective function. In this article, we will work with Hyperopia using the Wood Parzen Estimator (TPE). Other Python libraries include Spearmint (Gaussian Process proxy) and SMAC (Random Forest Regression). There are many interesting studies in this field. If you are not satisfied with a library, check out the alternatives! The general structure of a problem (hereafter) is a translation only minor differences in syntax between libraries. See this article for a basic introduction to the Hypermeter.

Four Parts of the Optimization Problem

There are four parts to a Bayesian Optimization problem:

Objective function: what we want to minimize is the error of validation of a machine learning model according to hyperparameters in this case

Field Name: hyperparameter values to search

Optimization algorithm: Method for constructing a carrier model and selecting the next hyperparameter value to be evaluated

Outcome history: Results from objective function evaluations of hyperparameters and loss of validation

With these four parts, we can optimize any function that returns an actual value (we can find the minimum). This is a powerful abstraction that allows us to solve many problems as well as to adjust the machine learning hyperparameters.

Data set

In this example, we will use the Caravan Insurance dataset that aims to predict whether a customer will buy an insurance policy. This is a supervised classification problem with 5800 training observations and 4000 test scores. The measurement that we will use to evaluate performance is the ROC AUC, which is an unbalanced classification problem. (A higher ROC AUC is better with 1 point, which shows an excellent model). The dataset is shown below:

Since Hyperopt requires a value to minimize, we will return the 1-ROC AUC from the target function, thus increasing the ROC AUC.

Gradient Enhancement Model

The detailed knowledge of the gradient enhancement machine (GBM) is not necessary for this article, and the basics we need to understand are: GBM is a group upgrade method based on the use of weak learners (almost always decision trees) who is trained in order to form a strong form. Model. There are many hyperparameters in GBM that control both whole community and individual decision trees. One of the most effective methods to select the number of trees (estimators) is early stopping. LightGBM allows for the quick and simple implementation of GBM in Python.

Going beyond the necessary background, let's write four parts of a Bayesian optimization problem for hyperparameter setting.

Purpose function

The goal is what we are trying to minimize. It takes a series of values - hyperparameters for GBM in this case - and gives a true value to minimize the loss of cross validation. Hyperopt sees the objective function as a black box because it only considers what comes in and out. The algorithm does not need to know the contents of the objective function to find input values that minimize loss! At a very high level (in the so-called code), our target function should be:

```
def target (hyperparameters):
"" "Returns the validation score from hyperparameters" ""

model = Classifier (hyperparameters)
validation_loss = cross_validation (model, training_data)

return validation_loss
```

We need to be careful not to use the damage. Since we can only use the test set to test, set it one-time , considering the latest model. Instead, we evaluate hyperparameters in a validation set. We also use KFold cross validation, which should give us a less biased estimate of error in the test set, in addition to maintaining valuable training data, rather than separating training data into a different set of validations.

The basic structure of the objective function for the hyperparameter setting will be the same between models: the function takes hyperparameters and returns the cross validation error using these hyperparameters. Although this example is specific to GBM, the structure can be applied to other methods.

The full purpose function for the Gradient Augmentation Machine using 10-fold cross validation with early stop is shown below.

Main line cv_results = lgb.cv (...). We use the LightGBM function to implement cross validation with early stopping, including Özgeçmişhyperparameters, a training set, a set of curves to be used for cross validation, and several other arguments. We have determined the number of estimators (num_boost_round) from 10000, but this number is not available because early_stopping_roundsstopping training when Verification points have not improved for 100 estimators. Early stop is an effective way to select the number of predictors instead of setting it as another hyperparameter that needs to be set!

After the cross-validation is complete, we get the best score (ROC AUC), and then we want a value to minimize, so we get the best 1 score. This value is then returned. kayıpenter the return dictionary.

We're returning a dictionary of values because this objective function is actually a little more complicated than it should be. For the objective function in Hyperopia, we can return to a dictation with a single value, loss, or minimum keys. "Kayıp"and "Durum". Restoring hyperparameters allows us to investigate the loss from each set of hyperparameters.

Domain Name

The field field represents the range of values we want to evaluate for each hyperparameter. For each iteration of the search, the Bayesian optimization algorithm will select a value for each hyperparameter from the field space. When we search random or grid, the domain area is a grid. The idea outside of this field in Bayesian optimization is the same probability distributions for each hyperparameter rather than discrete values.

Determining the domain is the most difficult part of the Bayesian optimization problem. If we have experience with a machine learning method, we can use it to declare our hyperparameter distribution choices, putting greater possibilities where we think the best values are. However, the optimum model, settings will vary between data sets and it may be difficult to find the interaction between the high dimensionality problem (many hyperparameters) and hyperparameters. If we are not sure about the best values, we can use wide distributions and allow the Bayesian algorithm to make sense to us.

First, we should look at all the hyperparameters in a GBM:

Import lgb

```
# Default gradient incrementing machine classifier
model = lgb.LGBM Classifier ()
model

LGBM Classifier (boostting_type = 'gbdt', n_estimators = 100,
class_weight = None, colsample_bytree = 1.0,
learning_rate = 0.1,
maximum_depth = -1, min_child_samples = 20,
min_child_weight = 0.001, min_split_gain = 0.0,
n_jobaves = -1, num_le = 31, purpose = None,
random_state = None, reg_alpha = 0.0, reg_lambda = 0.0,
silent = True, subsample = 1.0,
subsample_for_bin = 200000, subsample_freq = 1)
```

I'm not sure there's anyone who knows how this all interacts together! We don't have to set some of these (amaçand for example random_state) and we'll use stopping early to find the best one n_estimators. However, we still have 10 hyperparameters to optimize! When setting up a model for the first time, I usually create a large area centered around the default values and refine it for subsequent searches.

As an example, let's define a simple domain in Hyperopt, which is a separate uniform distribution for the number of leaves in each tree in

GBM:

Hyperopia imported from HP

# Discrete uniform distribution

num_leaves = {'num_leaves': hp.quniform ('num_leaves', 30, 150, 1)}

This is a discrete uniform distribution, distribution because the number of leaves should be an integer (discrete) and each value in the field is likely to be equal (uniform).

Another distribution option is the log uniform, which evenly distributes values on a logarithmic scale. We will use a daily uniform (0.005 to 0.2) for the learning rate, as it varies in several orders of magnitude:

# Learning rate daily uniform distribution

learning_rate = {'learning_rate': hp.Loguniform ('learning_rate',

np.log (0,005),

np.log (0.2)}

Since this is a log-uniform distribution, the values are plotted between exp (low) and exp (high). The following drawing on the left shows the distribution of the individual distribution and the drawing on the right is the log uniform. These are core density prediction graphs, so the y-axis is density, not a number!

Now let's define the entire domain:

Here we use several different domain distribution types:

seçim : categorical variables

quniform : discrete uniform (integers evenly spaced) üniforma: continuous uniform (floating at equal intervals)

loguniform: continuous billet uniform (floating at equal intervals on a billet scale)

(There are other distributions listed in the documentation.)

An important consideration when defining the type of upgrade is:

Here we use the conditional field that expresses the value of a hyperparameter depends on the value of the other. For support type "Goss", gbm cannot use subsampling (only the alt örnekproportion of training observations to be used in each iteration). Therefore, the alt örnekretrofit type is set to 1.0 (without subsampling). "Goss" otherwise, it is 0.5-1.0. This is implemented by using a nested domain.

Conditional nesting can be useful when using different machine learning models with completely separate parameters. Conditionally, it allows us to use sets of different hyperparameters based on a value. seçim.

Now when our domain is defined, we can draw an example to see what a typical example looks like. When we take the example, because alt örnekwe need to assign it to a high-level switch that is initially nested. This is done using the Python dictionary. almakthe default value is 1.0.

{'boostting_type': 'gbdt'

'class_weight': 'balanced',

'number_sample': 0.8111305579351727,

'learning_speed': 0.16186471096789776,

'min_child_samples': 470.0,

'num_leaves': 88.0,

'reg_alpha',

'regatta' : 0.8554826167886239,

'subsample_for_bin': 280000.0,

'subsample': 0.6318665053932255}

(This requires reassigning nested keys because the gradient incrementing machine cannot handle nested hyperparameter dictionaries.)

Optimization Algorithm

Although it is the conceptually most difficult part of Bayesian Optimization, the creation of an optimization algorithm in Hyperopia is a single line. Code to use the Tree Parzen Estimator:

hyperopia from import tpe

# Algorithm

tpe_algorithm = tpe.suggest

That is all! Although Hyperopt only has a TPE option to random searches, the GitHub page says other methods can come. During optimization, the TPE algorithm constructs the probability model using past results and decides the next set of hyperparameters to evaluate the objective function by maximizing the expected improvement.

Result History

Since Hyperopt will do this internally for the algorithm, it is not absolutely necessary to track the results. However, if we want to know what is going on behind the scenes, Denemelerthe object that will store the basic educational information, as well as the glossary (kayıpand parametreler) returned from the objective function . One line I make is a test object:

Trials from Hyperopia Imports

#

Bayes_trials = Trials to test object to monitor progress ()

Another option that allows us to monitor the progress of a long training run is to write a line to the csv file for each search iteration. This also saves all results to disk in the event of something disastrous and we lose the object of trials (speaking from experience). We can do this using. csvlibrary. Before the training we open a new csv file and write the titles:

and then we can add rows to write to csv at each iteration in the objective function (all target function is in the book):

Writing to a csv means we can check the progress by opening the file during training (not in Excel because this will cause an error in Python. Use kuyruk out_file.csvbash to display the last lines of the file).

Optimization

After placing the four parts, the optimization is performed. fmin :

Each iteration, the algorithm selects the new hyperparameter values from the proxy function created based on the previous results and evaluates these values in the objective function. This continues The MAX_EVALSevaluation of the objective function with the proxy function is continuously updated with each new result.

Results

en iyiThe returned object fmincontains hyperparameters with the least loss of the Objective function:

{'boostting_type': 'gbdt'

```
'class_weight': 'balanced',
'number_sample': 0.7125187075392453,
'learning_speed': 0.022592570862044956,
'min_child_samples': 250,
' num_leaves': 49,
'reg_alpha': 0.20352
', 020352 : 0.6455131715928091,
'subsample': 0.983566228071919,
'subsample_for_bin': 200000}
```

Once we have these hyperparameters, we can use them to grow a model about the complete training data, and then we can evaluate the test data (remember, we can only use the test set once when evaluating the last set). For the number of estimators, we can use the number of estimators with the least loss in early stop and cross validation. Final results below:

The best model scored 0.72506 AUC ROC in the test set.

The best cross-validity score was 0.77101 AUC ROC.

This was performed after 413 search replicas.

As a reference, 500 iterations were returned in a random scoring model, which received a 0.7232 ROC AUC test set and 0.76850 in cross validation. The default model without optimization score is 0.7143 ROC AUC Test set.

When we look at the results, there are a few important notes to keep in mind:

Optimal hyperparameters are those who do the best cross-validation and not necessarily those who do the best test data . When we use cross-validation, we hope these results will generalize to the test data.

Even if 10 times cross validation is used, the hyperparameter setting overfits the training data. The best score from cross-validation is much higher than the test data.

Random search can only give better hypermeters, depending on chance (restarting the laptop may change the results). Bayesian optimization is not guaranteed to find better hyperparameters and the objective function may be trapped at the local minimum.

Bayesian optimization is effective, but does not solve all of our tuning problems. As the search progresses, the algorithm switches from discovery - to try new hyperparameter values - to exploitation - using hyperparameter values that cause the lowest loss of objective function. If the algorithm finds a local minimum of the objective function, it may focus on the hyperparameter values around the local minimum, instead of trying different values in the domain area. Do not concentrate on any value because random search does not suffer from this problem!

Another important point is that the benefits of hyperparameter optimization will differ from the data set. This is a relatively small data set (~ 6000 training observations) and there is a small refund for adjusting the hyperparameters (the more time it takes to get more data!). With all these caveats in mind, in this case, with Bayesian optimization we can achieve:

Better performance in the test set

Fewer iterations to set hyperparameters

Bayesian methods can give better tuning results than random searches (although not always). In the next few chapters, we will examine the evolution of Bayesian hyperparameter research and compare it with random search to understand how Bayesian Optimization works.

Visualizing Search Results

Graphing results is an intuitive way to understand what happens during a hyperparameter search. It is also useful to compare Bayesian Optimization with random search, so that we can see how the methods differ. To see how the terrain is made and how the random search is made, see the notebook, but we'll review the results here (as a note, the exact results will vary between iterations, so if you run the notebook, don't be surprised if you get different images. All of these graphics are done with 500 iterations).

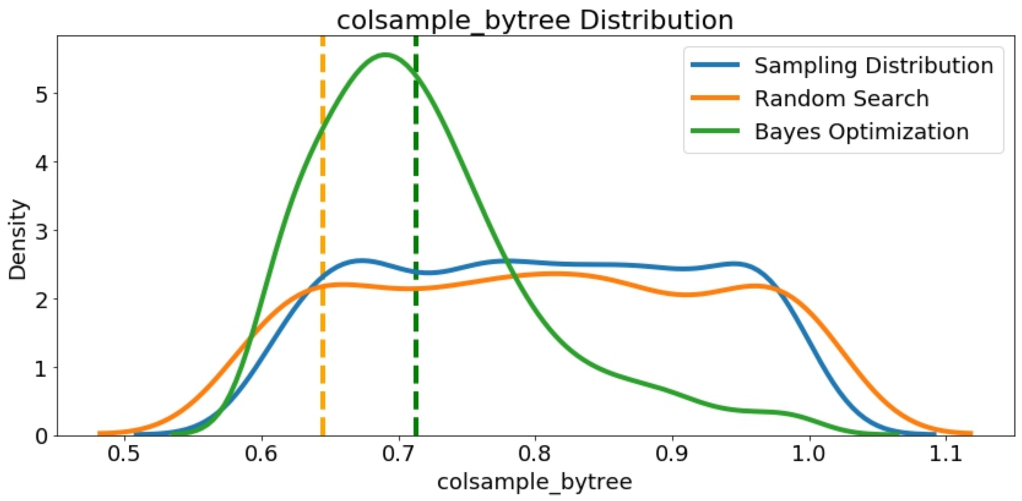
First, we can plot a core density estimate. öğrenme oranırandom search and Bayesian Optimization. As a reference, we can also show the sampling distribution. The vertical dashed lines indicate the best values (relative to cross-validation) for the learning rate.

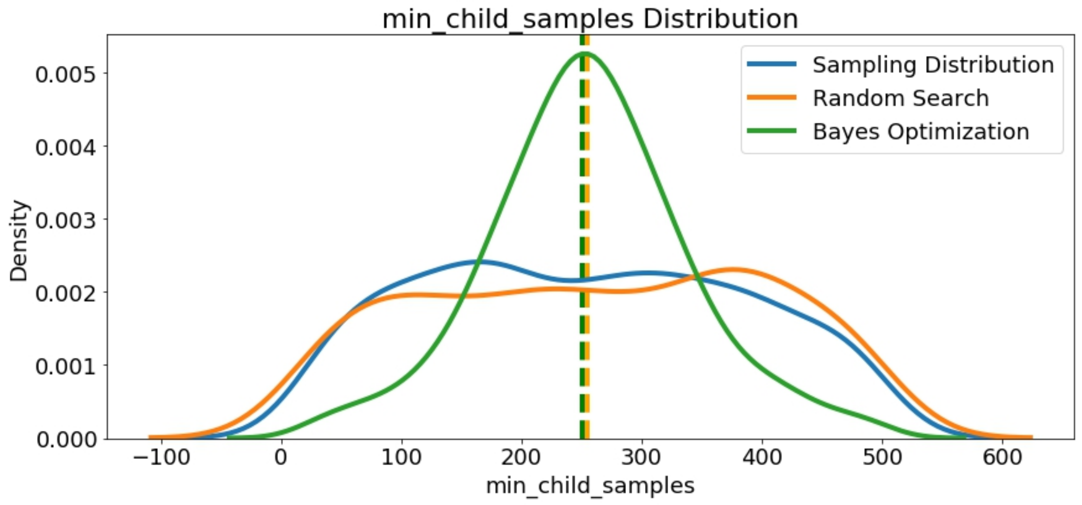
We defined the learning rate as log-normal between 0.005 and 0.2, and the Bayesian Optimization results are similar to the sampling distribution. This tells us that the distribution, we define seems appropriate for the task, but the optimal value is slightly higher than where we put the highest probability. This can be used to inform the domain for further searches.

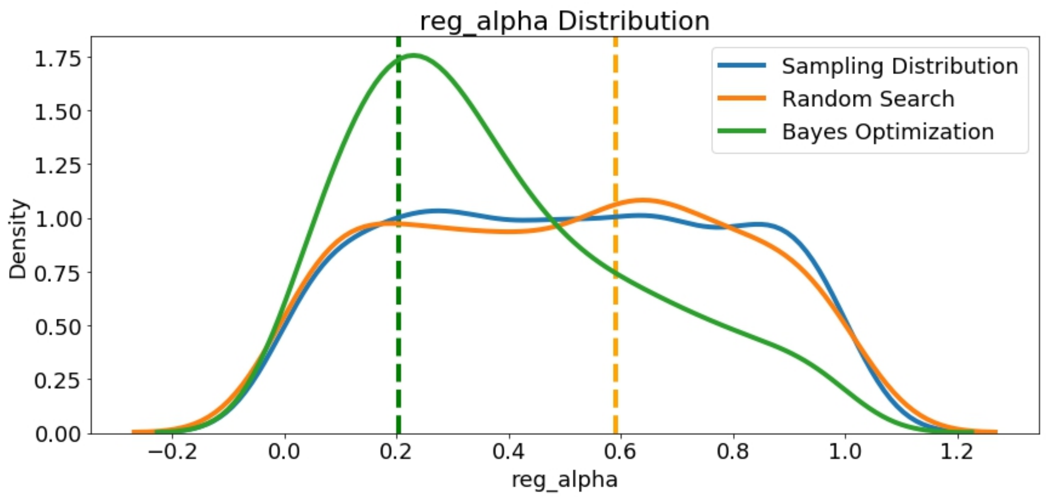
Another hyperparameter is the type of amplification with bar graphs of each type evaluated during random search and Bayesian optimization shown below. Since random search does not pay attention to historical results, we expect roughly the same number of types of each upgrade.

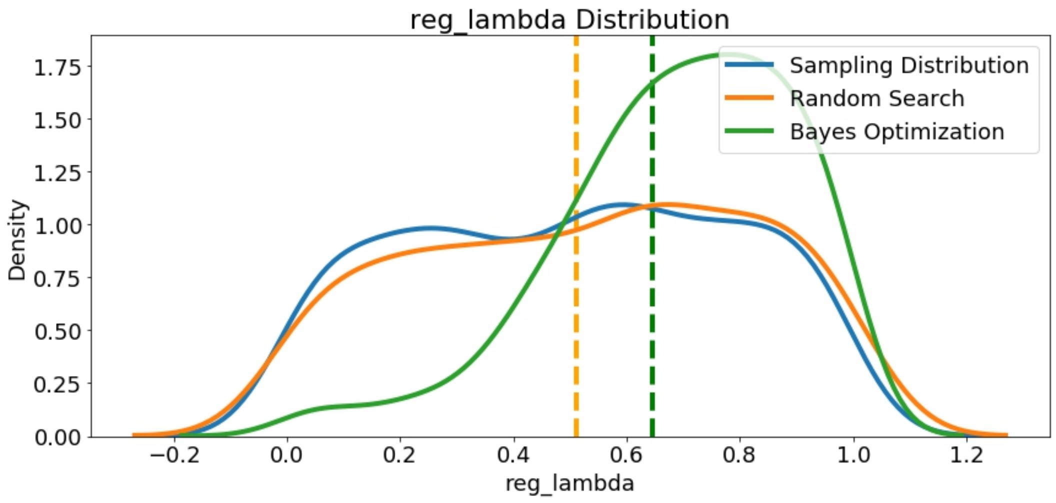
According to the Bayesian algorithm, the gdbttype of boost is more promising Dart oyunuor goss. Again, this can help do more searches, such as Bayesian methods or grid search. If we want to do a more conscious grid search, we can use these results to define a smaller grid that concentrates around the most promising values of hyperparameters.

Since we have them, let's look at all numerical hyperparameters from reference distribution, random search, and Bayesian Optimization. Vertical lines again show the best value of the hyperparameter for each search:







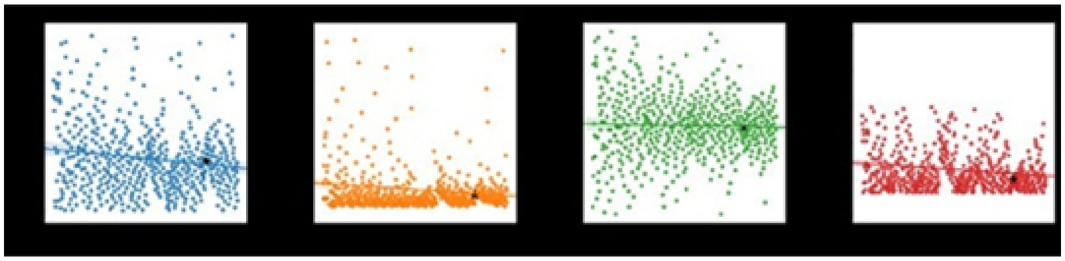


In most cases (except subsample_for_bin), Bayesian optimization research tends to concentrate near hyperparameter values that provide the lowest loss of cross-validation. This demonstrates the basic idea of setting hyperparameters using Bayesian methods: Spend more time evaluating promising hyperparameter values.

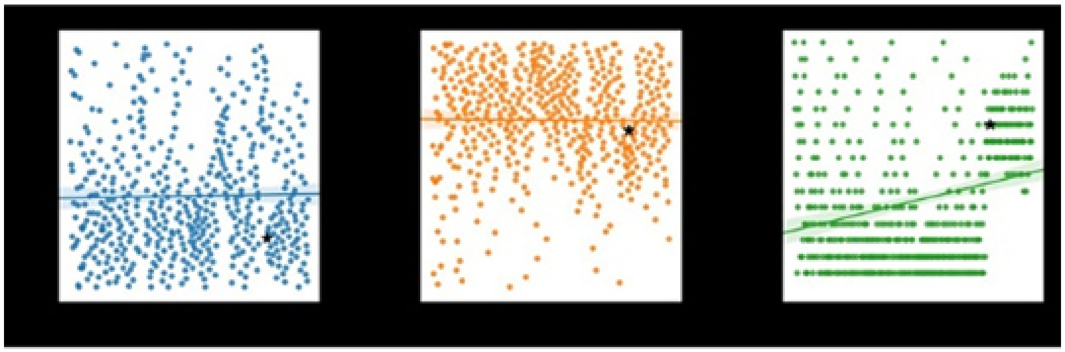
There are also some interesting results that can help us in the future when it is time to define a domain to be investigated. Just as an example, it looks reg_alphaand should reg_lambdacomplement each other: if one is high (close to 1.0), the other should be lower. There is no guarantee that this will apply to problems, but by studying the results we can gain insight into future machine learning problems!

Evolution of Search

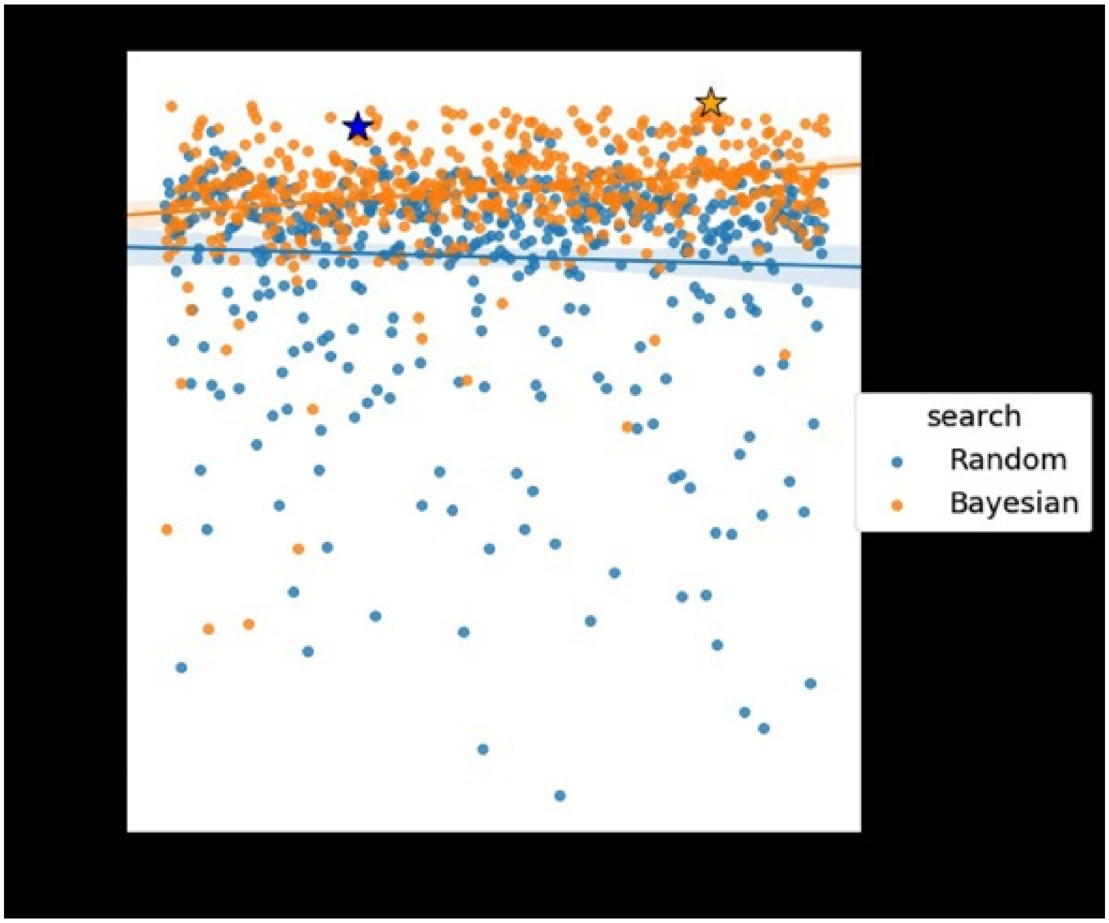
As the optimization progresses, we expect the Bayesian method to focus on the promising values of hyperparameters: those that give the lowest error in cross-validation. We can draw the values of hyperparameters according to iteration to see if there are outstanding trends.



The black star indicates the optimal value. colsample_bytreeand öğrenme oranıwill diminish over time that can guide us in future searches.



Finally, if Bayesian Optimization is running, we expect the average verification score to increase over time (on the contrary, the loss decreases):



Verification scores from Bayesian hyperparameter optimization increase over time, indicating that the method attempts "better" hyperparameter values (note that these are only better than the validation score). Random search shows no improvement in iterations.

Keep Searching

If we are not satisfied with the performance of our model, we can continue searching where we left off using Hyperopt. We just need to switch to the same experimental object and the algorithm will continue to search.

As the algorithm progresses, more exploitation - value collection - and less discovery - which have yielded good results in the past, gather new values. Instead of resuming the search, it may be a good idea to start a completely different search. If the best hyperparameters in the first search are really “optimal, iz we expect subsequent searches to focus on the same values. Given the high dimensionality of the problem and the complex interAction between hyperparameters, another study is unlikely to result in a similar group of hyperparameters.

After 500 retraining , the latest model is in the 0.72736 ROC AUC Test set. (We should really first model evaluation should rely solely on the test set and validation points. The test kit should be used only as an ideal. A time is to measure the performance of the algorithm is deployed to new data). Again, this problem may have decreasing returns to hyperparameter optimization due to the small size of the data set, and consequently will be a plateau in the validation error (there is a natural limit to the performance of any model in the data set due to hidden data variables. Are measured and non-noisy data, Bayes 'Error').

Results

Automatic hyperparameter adjustment of machine learning models can be performed using Bayesian optimization. In contrast to random searches, Bayesian optimization selects subsequent hyperparameters in an informed method to spend more time evaluating promising values. The final result may be less in the evaluation of the objective function and may have a better generalization performance on the test set than in random or grid search.

Conclusion

It is best to define what a data scientist is and the characteristics that, ideally, it should have. I am not going to elaborate much on this part, since there are many websites that deal with this topic and it is not the objective of this post.

We can define data, scientists as professionals who, using large volumes of information and of different types, solve business problems and obtain answers from data.

Although there are no “typical” data science projects, since each project is a world, companies generally start with a structure similar to this:

It is required to solve a business problem or answer a question.

The data scientist should obtain data from the source that was intended to solve that question. Structured data (eg, databases such as SQL), unstructured (eg, images, audios) and semi-structured (eg, texts with a certain structure) come into play. In companies that are starting generally only structured data is used and accessed by a typical query language such as SQL.

A series of techniques, algorithms, etc. are applied to this data. They try to solve the case. Here tools like Python, R, SAS, etc. are used.

The final result obtained can be an analysis, a productivity of some statistical model, results for business, etc.

The main characteristics that a data scientist should have are represented in the following diagram published by Drew Conway in 2010 :

First, there are those called "Hacking Skills", which involve computer skills, data manipulation, command line, programming, etc.

Second, there is the knowledge of mathematics and statistics. It is not necessary to be a doctorate in mathematics or statistics to be a good data scientist, however, it is to have a certain mastery of basic statistical concepts and know how to interpret the algorithms that one uses.

Finally, and something that is not usually given the importance it has, is business knowledge. As important as knowing what algorithm to use or how to program it is knowing what business questions you want to solve, what brings value and what doesn't, or to what extent the problem we want to address can be viable. In this post we discuss different types of problems that can be solved with machine learning techniques.

These three types of skills are what a good data scientist should have. The first thing a data scientist should know is how to program, as well as having some SQL handling.

The steps I recommend to get into a data science are the following:

Learn to program it in R or Python. I prefer Python since it is more general purpose and to learn the concepts of programming is relatively simple. Some knowledge of how to use the command terminal is also very useful .

Learn the basics of SQL and statistics. Generally it is not necessary to be Don Chamberlin but it is essential to know how to make queries to join tables, make groupings, filters, etc. In the same way, it is not necessary to have extensive knowledge to start with data scientist, but concepts such as measures of dispersion, centrality or hypothesis tests are very useful.

Know machine learning algorithms and start programming them, using public open data, competitions, etc.

I don't know how to program, where do I start?

To people who do not know how to program and want to go into a data science, I recommend that they start with some programming language, if possible a general purpose one such as Python. R and

Python are the two most used languages in data science. R is widely used by mathematicians and statesmen, while Python is more used by profiles that come from engineering.

Between the two languages I stay with Python because it is much more versatile than R and the concepts learned are more extrapolated to other types of programming languages. It is true that perhaps at first it may be somewhat more difficult than R, but in the long run it is worth it.

I leave a compilation of quite interesting material for those who need to learn Python and SQL from scratch on their own.

Material to learn Python and SQL on your own:

Online courses:

Introduction to Python from DataCamp : DataCamp is a platform that I highly recommend to do data science courses. There are courses at all levels and themes (programming, algorithm theory, etc.). There are both free and paid courses.

Free Basic Python Course : This course is basic, in Spanish and covers the main language.

Intro to SQL for DataCamp Data Science : another DataCamp course, in this case SQL.

SQLBolt : interactive course that covers the basics of SQL with examples.

Books:

To learn Python, I recommend two books: Python Crash Course and Python 3. The basics of language .

There are also the books "Learn Python on a weekend" and "Learn SQL on a weekend" which, although I have not read them personally, have good references as introductory books.

To learn how to program in Python, I recommend installing Anaconda together with Python and using Jupyter Notebook as a development environment.

I know how to program now what?

Once you have knowledge of programming and SQL, before starting to learn algorithms it would be useful to review basic statistical concepts.

To know the basics of statistics, books like Introduction to Probability and Statistics or courses like Stanford's Probability and Statistics can be useful .

The time has come for the algorithms

At this time we would already have basic knowledge of programming and SQL to start programming the first machine learning algorithms. Before I start programming I suggest learning the concepts of machine learning models. For this there are numerous courses and books that teach us what they are for, what they are based on and how to use them.

As a theoretical initiation material I recommend these two books:

An Introduction to Statistical Learning : This book is basic and explains the concepts in a clear and simple way. It deals with the different types of learning and own algorithms at a low level.

The Elements of Statistical Learning: similar to the previous one, but more rigorously mathematically speaking and broader.

Above all, I recommend two online courses start:

Stanford Statistical Learning : This course follows the previous two books, making it a very good introductory course.

Machine Learning : course par excellence, explained by Andrew Ng. which is something like a Nobel of artificial intelligence. Explain the fundamental concepts in a very clear and simple way. A must.

While doing these courses, it can be interesting to program the algorithms using Scikit-Learn , which is a Python library aimed mainly at learning statistical algorithms. Your documentation is very well explained and loaded with examples.

How do I keep learning?

At this time we would already be able to program our own basic algorithms. From here it is possible to expand knowledge with more advanced books and courses.

In addition, you can also use the Kaggle data science competition's website to learn from other data scientists. I highly recommend this website, you can learn a lot with competitions, kernels (codes uploaded by other users) and even with the learning section .

Finally, I would like to point out three websites that I use a lot to keep up with the latest developments: one is Analytics Vidhya , another Towards Data Science and finally KDnuggets . In them, you can find very valuable material.