

ID#220205

PUBLISHED ON  
MARCH 15, 2022

## Automating Bureaucracy with Python: The Case of the Springfield Bail Fund

BY C. DANIEL GUETTA\*

### The Bail System in the United States

Anyone arrested in the United States<sup>i</sup> is entitled to an initial hearing to review whether they should be detained until trial. During this hearing, the court may grant bail—a conditional pretrial release in which the accused promises to appear in court when required.

When a court sets bail, it usually requires a dollar amount the defendant must pay to be released to guarantee the defendant will show up for their trial. At the conclusion of the case, when the defendant has fulfilled all attendance obligations, the money is then returned. In this sense, bail acts as an incentive to encourage the defendant to return to court when required. (In some cases, a judge may choose to release the defendant on their own recognizance, without any payment required.)<sup>1</sup> Bail amounts vary widely, with a nationwide median of around \$10,000 for felonies (much higher for serious charges) and less for misdemeanors.<sup>2</sup> In some cases, bail is set as low as one dollar. This usually happens when a defendant has two cases open simultaneously; the defense attorney will ask for one-dollar bail to be set on the lesser of the offenses so time spent incarcerated for the more serious crime can count as time spent for the less serious crime.<sup>3</sup> Due to the administrative difficulty of paying bail, some defendants remain incarcerated on these “dollar bails” for long periods.<sup>4</sup>

Fewer than 5% of pretrial releases involve the defendant paying the full bail amount.<sup>5</sup> More commonly, the defendant will pay a fee—often around 10% of the bail amount—to a commercial bail agent. The bail agent agrees to pay the full bail (which is retrieved once the case is over), but the fee is not returned to the defendant. Bail agents often purchase insurance against the possibility the defendant will not show up and thus forfeit the bail. Only about 10

---

<sup>i</sup> Bail systems exist in other countries, but this case will focus on the system as it is implemented in the United States.

#### Author Affiliation

\* Columbia Business School

#### Acknowledgements

The author is grateful to Will Dunn, Drew Feldman, and Royce Fillmore for providing research, writing, and data analytic support, and to Rebecca Borison, Chris Correa, Joseph Pate and Cameron Pipe for the time they spent talking to the author about bail in the United States.

#### Copyright Information

© 2022 by The Trustees of Columbia University in the City of New York.

This case is for teaching purposes only and does not represent an endorsement or judgment of the material included.

This case cannot be used or reproduced without explicit permission from Columbia CaseWorks. To obtain permission, please visit [www.gsb.columbia.edu/caseworks](http://www.gsb.columbia.edu/caseworks), or e-mail [ColumbiaCaseWorks@gsb.columbia.edu](mailto:ColumbiaCaseWorks@gsb.columbia.edu).

insurers underwrite a significant majority of the approximately \$14 billion in bail bonds issued in the United States each year, and they collect around 10% of the premiums bail agents charge for this service.<sup>6</sup> This for-profit cash bail system only exists in two countries in the world: the United States and the Philippines.<sup>7</sup>

Though the exact factors and conditions that determine whether a pretrial release is granted and specific bail amounts vary by jurisdiction, in general the four main criteria used to set bail are the seriousness of the alleged offense(s), the defendant's flight risk (probability of reappearing in court), criminal history, and community ties.<sup>8</sup>

## Cash Bail on Trial

Proponents of the bail system argue that bail props up the entire court system by allowing certain defendants to avoid incarceration before trial while keeping those who might pose a danger to society incarcerated. They also note that giving defendants a financial incentive to return to court reduces the number of defendants who fail to show up for their appearances, thus easing the burden of having to find them.

This system, however, also faces considerable criticism. The US justice system theoretically presumes that defendants are innocent until proven guilty; however, according to some "cash bail systems violate the rights of [the accused] who have been charged, but not convicted, of crimes," by limiting the practice of this right to only those who can pay.<sup>9</sup> Indeed, as of 2020, hundreds of thousands of people were jailed in the United States despite not having been convicted of a crime—around 20% of the incarcerated population.<sup>10</sup> These decisions are sometimes made in mere minutes<sup>11</sup> and can sometimes be highly dependent on the judge that happens to preside over the bail hearing.<sup>12</sup>

This kind of incarceration can lead to dire consequences. Defendants unable to make bail are more likely to lose their employment, become distant from family members, and be exposed to dangers and violence inside jail. Pretrial release also has implications for the result of a case. For various reasons those unable to post bail are more than three times more likely to be sentenced to prison than those released pretrial.<sup>13</sup> Pretrial detention also has a disproportionate impact on communities of color; Black and Hispanic individuals are much more likely to be held on bail than white individuals.<sup>14</sup> It is even unclear whether bail keeps society safer. In a study of two large jurisdictions, nearly half of the defendants considered "high risk" were released simply because they could afford bail.<sup>15</sup>

It is perhaps a mark of the strong opinions about this issue that politicians across the spectrum have called for reform of the bail system. A 2017 editorial on the subject in the *New York Times* was coauthored by Kamala Harris and Rand Paul<sup>16</sup>—unlikely bedfellows in any other context. Steps towards reform, however, have been slow, and there is disagreement as to how the system should be reformed.

## Reforming the Bail System

The concerns surrounding bail began to be pointed out as early as the 1920s. By the 1960s, studies showed that assigning bail was unnecessary; the vast majority of people would come back to court for their hearings even without the financial incentive.<sup>17</sup> In 1966 the Bail Reform Act<sup>18</sup> required a person to be released under the “least restrictive conditions” that would ensure their appearance at court hearings. In the 1970s and 1980s, fears of rising crime led to another law: the Bail Reform Act of 1984.<sup>19</sup> This allowed the judge in a federal court to detain someone if they were a serious threat to public safety or if they were likely to abscond if released. In 1987 the Supreme Court affirmed that detaining a person for the sake of public safety was constitutional.<sup>20</sup> This set the foundation for the bail system as we know it today, and in the years since, the use of cash bail in felony cases has risen from 37% to 61%.<sup>21</sup>

Recent attempts to reform the system have focused both on lawsuits to argue cash bail is unconstitutional and on state-specific changes in the law.

An example of statewide reform is the New Jersey Criminal Justice Reform Act, signed by then-Governor Chris Christie, which took effect in 2017. The act essentially eliminated cash bail in the state. Every defendant is granted the presumption of pretrial release, and the prosecution must convince the judge that the defendant absolutely needs to be detained, either to ensure public safety or their return to the court. An algorithmic Public Safety Assessment (PSA) system is used to determine the risk a person will re-offend. The PSA was created using a database of more than 1.5 million cases drawn from more than 300 jurisdictions.<sup>22</sup> After the reform was enacted, the pretrial population was reduced by 19%, which corresponds to approximately 3,000 defendants.<sup>23</sup>

New York also saw changes to its bail system in 2020, eliminating pretrial detention and cash bail as an option in an estimated 90% of arrests (most misdemeanors and nonviolent felonies). The law specified judges would have to release individuals charged with those crimes either on their own recognizance or with release conditions designed to ensure their return to court (such as pretrial supervision and text message reminders of court dates). In the remaining cases comprising the most serious crimes (including violent felonies), the law still gave the judge the option to set cash bail.<sup>24</sup> One important way New York’s measure differed from New Jersey’s is that since 1971, New York has not allowed judges to consider the “dangerousness” of an individual in deciding whether to set bail.<sup>25</sup> An initial version of the new law gave judges the ability to make this determination, but legislators did not include it in the final bill.<sup>26</sup>

In addition to these two initiatives, Washington, DC, and some states have also passed laws to reform their bail systems, including Massachusetts, Alaska, and Illinois. Lawsuits challenging the practice have also been filed in Texas, Alabama, Illinois, and California.<sup>27</sup>

These changes have not come without concerns. One week after bail reform was introduced in New York, prosecutors, law enforcement officials, and numerous state leaders, including then-Governor Andrew Cuomo, called for changes to the new law. They were concerned offenders might be released without bail and later re-offend. In one example, a Brooklyn

woman was arrested after being accused of slapping three Orthodox Jewish women in the face and yelling “F—k you, Jews!”<sup>28</sup> She was released without bail but was rearrested after police said she assaulted another woman the next day.<sup>29</sup> In another, a 22-year-old man accused of burglary was released without having to post bail on New Year’s Eve; he was shortly rearrested after police said he committed yet another burglary just hours after being set free.<sup>30</sup> In the first two months following the reform, 482 people charged with a felony were released without bail only to be rearrested for new crimes 846 times.<sup>31</sup>

Advocates of the law pushed back, accusing opponents of cherry-picking a few cases to stoke public anxiety. “For every story that ends up in the press that is concerning, there are dozens of people who have been accused of very low-level crimes that are now home with their families,” said Senator Michael Gianaris, a Queens Democrat who served as deputy majority leader at the time.<sup>32</sup> One disturbing example of the bail issue was Bronx teenager Kalief Browder, who in 2010 was accused of stealing a backpack. Denied bail, he was held for three years on Rikers Island without ever being tried. His subsequent suicide galvanized the push for reform in New York.<sup>33</sup> Even the broader statistics, advocates pointed out, painted a more nuanced story—in the period following the introduction of the law, New York City saw fewer murders and rapes, for example.<sup>34</sup> They also pointed to the result of New Jersey’s reforms,<sup>35</sup> which four years later were widely acknowledged to have been successful,<sup>36</sup> so much so that even detractors of New York’s reform pointed to New Jersey as a success story.<sup>37</sup>

Ultimately, the New York law was reformed three months after its introduction, allowing cash bail to be set on a broader range of crimes. In addition, the new law allowed judges to set bail based on a person’s legal history and status. For example, for persistent offenders, cash bail could be set for any new felony charge.<sup>38</sup>

Other concerns around these reforms have focused on the Public Safety Assessment (PSA) system used in New Jersey and other jurisdictions.<sup>39</sup> The nonprofit Pretrial Justice Institute, which initially strongly recommended the use of these systems in New Jersey, has since reversed itself, claiming these tools perpetuate racial inequities.<sup>40</sup> These concerns have since been echoed by many,<sup>41</sup> many,<sup>42</sup> others. There is also concern around exactly how these algorithms are used by judges, and if some judges might selectively ignore the results of these algorithms, thus introducing additional bias into the system.<sup>43</sup>

It is finally worth noting that commercial bail agents also have a major incentive to influence any reforms to cash bail. Bail insurance is a large industry, with profit margins of 83%, compared with 33% in home and auto insurance, and insurance companies have spent \$17 million to defeat proposals to weaken or abolish it—spending that has grown tenfold since 2010.<sup>44</sup> Among other efforts, the bail industry has supported legal challenges against the New Jersey law eliminating cash bail. One involved a man shot 22 times by a felon who had been released on bail for a weapons violation. Another centered on a man who would have had the means to pay cash bail but was instead released and fitted with an electronic ankle bracelet under the new law.<sup>45</sup> The New Jersey law was eventually upheld.<sup>46</sup>

## The Advent of Bail Funds

The bail system is reforming, but for some, it is reforming too slowly. Bail funds are organizations that collect money and use it to post bail for some members of the community, based on internal eligibility criteria. The first bail fund in the United States appeared in the early 1920s, spearheaded by the American Civil Liberties Union to bail out political dissidents suspected of conspiring in a Communist revolution during the First Red Scare. Bail funds continued to grow in prominence during the Second Red Scare in the 1950s, the Civil Rights movement in the 1960s, the Vietnam War protests in the early 1970s, and the AIDS epidemic in the late 1990s. Attention to paying bail in low-profile cases also grew as a response to movements to counter mass incarceration, jail overcrowding, child detention, and immigration waves from the 1970s to early 2000s.<sup>47</sup>

Typically, bail funds identify potential clients from jail rosters or referrals from the community. Once a candidate has been identified and fulfills internal criteria, the bail fund pays the bail amount and any associated fees to the court and the defendant is released. When the case is closed, if the client has maintained all attendance obligations, the money is released back into the fund where it can be used to pay bail for another candidate.<sup>48</sup>

Many bail funds now exist throughout the United States at both state and local levels.<sup>49</sup> One of the largest is the Bail Project ([bailproject.org](http://bailproject.org)), whose mission is to “restore the presumption of innocence, reunite families, and challenge a system that criminalizes race and poverty.” It has operations in over 30 cities across the country and has provided bail assistance and pretrial support for tens of thousands of low-income individuals to date. The fund is charge agnostic, meaning the charge alone does not determine eligibility. Instead, the organization looks at the reliability of provided contact information, prior court appearance track records, and any situational specifics that may affect the likelihood of a candidate appearing in court. The Bail Project also provides support to defendants to ensure they are able to return to court when required.<sup>50</sup>

## A Major Logistical Challenge

Bail funds face many challenges, from raising money to the logistical challenges of managing the bail process. We will focus on the latter.

The vast majority of a bail fund’s work happens at the state level (this can often come as a surprise to those unfamiliar with the US criminal justice system). Because state laws around pre-trial detention differ, bail funds must adjust its approach accordingly with each jurisdiction. The definition of excessive bail, which is the maximum amount that may be reasonably imposed by the courts, may differ, as well as the way bail is paid, procedures, and other nuances.<sup>51</sup> Different states’ treatments of fees deducted from bail may also affect repayment amount. Put together, these variations create a formidable challenge for funds like the Bail Project trying to juggle defendants across jurisdictions.



Even for bail funds operating in a single jurisdiction, there is another formidable challenge that might not be immediately obvious: the challenge of recovering money as quickly as possible so it can be used to pay bail for other defendants. If a single defendant pays their bail themselves, knowing when to recover their money from the court is relatively simple. As soon as the case closes, they can return to the court and recover the amount they paid (though even that process can involve a number of administrative hurdles).

For a bail fund with potentially hundreds of cases currently in progress, this kind of case management exercise can be very difficult. In theory, a defendant could report back to the bail fund at the conclusion of the case, but in some instances, it might not even be clear to a defendant when bail can be recovered. There is no way to determine the length of a case a priori, which can range from several months to several years, and cases can conclude for many reasons (the defendant could be acquitted or convicted or may reach a plea bargain with the prosecutor). This can lead to money that could be used to bail out other defendants being locked up with the courts.

Some jurisdictions provide digital systems that allow bail funds to check the status of any case. Perhaps even more useful, some jurisdictions will allow interested parties to sign up for e-mail updates—every time the case progresses, an e-mail is sent with the update. New York’s eTrack system is one such system (see Exhibit 1 for a sample e-mail from the system).

However, even with such systems, the task of managing a bail fund’s caseload can be difficult. Since every event in a case can result in an update, eTrack systems will typically produce a very large volume of e-mail. Furthermore, bail funds might begin tracking a case as soon as there is even the slightest chance they will be helping the defendant in that case. Even if they later decide they are unable to pay bail for that defendant, the system will continue sending updates. One data scientist at a bail fund using this technique mentioned it was not unusual to receive hundreds of e-mails a day from the system—far too many to track manually.<sup>52</sup> It is in precisely these circumstances that programming tools like Python can shine; automation can quickly locate any e-mails of interest and identify cases for which moneys might be recoverable from the court system.

## The Springfield Bail Fund

For the purposes of this case, we will consider a fictional bail fund called the Springfield Bail Fund (SBF), created as an amalgam of a number of real bail funds operating across the United States. This bail fund, together with the city of Springfield, the details of its court system, and the cases we will encounter, are completely fictional—no information relating to individual defendants was used to produce the data in this case.

As part of your class, you will have been given access to an e-mail address belonging to this fictional bail fund. This in-box belongs to SBF, and whenever any case comes in, the fund manager signs up to Springfield’s eTrack system using this e-mail address.

There are three kinds of e-mails sent out by the eTrack system:

- Bail paid e-mails confirm bail has been paid for a case.
- Bail reimbursement e-mails confirm bail has been refunded for a case
- Court update alert e-mails apprise the receiver of an update in a specific case. In each case, the e-mail will indicate whether a case is closed. (Bail moneys in Springfield can only be recovered once the case is closed.)

E-mails are sent for every update of a tracked case. Log in to the e-mail account to see a year's worth of e-mails received by the fund, dated January 2021 to December 2021. (See Exhibit 2 for an example of an update email from the Springfield eTrack system.)

A quick scan of the in-box shows the cases therein cover a broad range of offenses and details the multiple steps taken in the lifetime of each case.

**Important note:** *The e-mails and court cases associated with this exercise represent a year's worth of fictional cases in Springfield. For the purposes of this class, these e-mails were generated all at once and sent to SBF's in-box by an automated script. Therefore, the date listed in the e-mail itself (the true date of the event) will not match the date on which the e-mail was sent. Be sure to use the date in the body of the e-mail.*

Ignore this date (it indicates when the e-mail was sent in preparation for class)

Use this as the date attached to a case update

SBF has been struggling with manually keeping track of the e-mails in this in-box, and they have enlisted your help. Appendix 1 contains the transcript of an interview with a manager at the bail fund. In preparation for class, read it and carry out the following exercises:

1. By searching through the e-mails, identify a few cases for which bail was paid by the fund.

2. Find any cases in which SBF has paid bail for a given case that has since closed. (These are the cases highlighted in green in the sample of hand-written notes in Appendix 1.)
3. Before class, start thinking of how—if you were able to obtain the text of these e-mails in Python—you could automatically extract the information from them that SBF would require to produce the reports it needs.

The three tasks above should make it clear how tedious this kind of tracking is. When we meet in class, we will discuss how Python can be used to automate this process.

Do Not Copy



## Exhibits

### Exhibit 1

#### Sample E-mail from New York's eTrack System (Identifying Details Have Been Obscured)

no-reply@nycourts.gov via gmail.com  
to erf.etrack

Case Number: [REDACTED]  
The following case which you have subscribed to in eTrack has been updated. Changes from the last update are shown in red and are annotated.

Court: Richmond Criminal Court  
Case Number: [REDACTED]  
Summons/Ticket Number:  
Defendant: [REDACTED]  
Defendant Status:  
Arrest Date: October [REDACTED], 2020 [REDACTED]  
Incident Date: July [REDACTED], 2020  
Criminal Justice Tracking Number: [REDACTED]  
NYSID Number: [REDACTED]  
Arrest Number: [REDACTED]  
Issuing Officer Agency: NYPD  
Issuing Officer Command:

Defense Attorney: [REDACTED]  
Attorney Type: Legal Aid  
Assignment Date: May [REDACTED], 2021

Assistant District Attorney: Richmond County District Attorney,  
Assignment Date: October [REDACTED], 2020

**Last Appearance: 06 [REDACTED], 2021 --- Information updated**  
Part: APV  
Judge: [REDACTED]  
Calendar Section: Pending  
Arraignment / Hearing Type: No Type  
Court Reporter: [REDACTED]  
**Outcome / Release Status: Warrant Stayed / --- Information updated**

Next Appearance: 06 [REDACTED], 2021 09:00 AM  
Part: APV  
Judge:  
Calendar Section: Pending  
Arraignment / Hearing Type:

Older appearances may exist but are not shown.

Charges:  
[REDACTED] Aggravated Family Offense \*\* TOP CHARGE\*\*  
E Felony, 1 count, Arrest charge, Arraignment charge  
Disposition/Sentence:  
[REDACTED] Harassment 2nd- Phy Contact  
Violation, 1 count, Arrest charge, Arraignment charge  
Disposition/Sentence:

## Exhibit 2

### Sample Update E-mail from the Springfield eTrack System

Sent Date = 10-26-2019 02:48:19

Case Number = CR-8108672

Defendant Name = Sara Bullard

DOB = 1968-03-18

Sex = F

Charge = Criminal possession of stolen property in the fifth degree, Class A Misdemeanor

Arrest Date = 10-08-2019

Last Appearance = 10-13-2019

Next Appearance

- Purpose = Evidence received 2
- Date = 10-26-2019
- Location = New York County
- Modality = Virtual

Case Closed = NO

--

CBS Python Bail Fund Case



## Appendixes

### Appendix 1

#### Interview with the Springfield Bail Fund

In preparation for the work we will be doing for SBF, Columbia Business School interviewed Operations Manager Sarah Jones to understand how Python automation could help the fund.

The following is a transcript of the interview (edited for clarity):

CBS: Thanks for your time, Sarah.

SJ: No, thank you! Based on our preliminary discussions, I can already tell you're going to be making our lives much easier.

CBS: Let's hope so! I'd like to begin by talking about the Springfield eTrack system. When do you decide to sign up for e-mail alerts for a specific case?


SJ: We generally take a better-safe-than-sorry approach. If a case comes across our radar that we might be interested in, we sign up for alerts. There are many ways this can happen. Sometimes one of our case workers notice the case while they're in court; we might get a referral from the defendant's friends or family; or the defendant might self-refer. In the past, we've even used interns to look for defendants that might benefit from our help and to track their cases wholesale.

This works well for us in that we rarely miss cases, but because we want to make sure we don't miss anything, no one wants to unsubscribe from any case updates. That means we're getting a massive volume of e-mails on a daily basis, and it makes things very difficult to track.

CBS: I can imagine! So you have all these case e-mails hitting your in-box every day. I'm going to look through them a little later, but can you talk me through the kinds of e-mails eTrack sends you?

SJ: You have a fun afternoon in store! There are three kinds of e-mails eTrack might send. The first is a receipt we get when we pay a defendant's bail. These e-mails have a subject line that begins with "Bail paid confirmation." The second is a receipt we get when we get bail refunded. These e-mails have a subject line that begins with "Bail reimbursement confirmation." When we get an e-mail like that, all we really keep track of is the amount paid or reimbursed, which appears in the e-mail, such as in this example:

Bail Reimbursement Confirmation: Case Number CR-8470345 Inbox x

 cbs.python.courts@gmail.com  
to me ▾

Sent Date = 10-07-2019 05:37:10  
Bail Reimbursement Confirmation  
Case Number = CR-8470345  
Amount = \$550.0

CBS: OK. And you said there were three—what’s the third kind?

SJ: The third kind is a case update. Once we sign up for a case, we get an update every time anything happens in the case. There are a lot of those; the subject line always begins with “Court update alert.”

CBS: But is that information relevant?

SJ: Most of it isn’t. But some parts are. For example, we get an update when bail is set or denied. This is important, because even though we might follow a case from the start, it only becomes relevant for us once bail is set and there’s something to pay. If bail is denied—or hasn’t been set yet—there’s nothing for us to do. So it’s important to keep track of that. Of course, we also want to keep track of whether bail was posted. If a defendant was able to post bail without our help, we can focus on other cases.

Another crucial piece of information we get from these e-mails is a notification when the case closes. Once we get that alert, we know it’s time to go and get the bail money back so we can use it to help someone else.


CBS: Got it. So bail set, bail denied, bail posted, and case closed.

SJ: That’s about it. The other information is important too, but we don’t really need to retrieve it regularly. We can just look it up for cases when we need it.

CBS: So talk me through what these update e-mails look like. How would you get this information from them?

SJ: Good question. Here is a sample of a typical update e-mail:

Court Update Alert: Case Number CR-4687019 Inbox x

 **cbs.python.courts@gmail.com**  
to me ▾

**Sent Date = 10-07-2019 04:03:19**  
Case Number = CR-4687019  
Defendant Name = Bertha Ferrera  
DOB = 1985-04-11  
Sex = F  
Charge = Compounding a crime- Class A Misdemeanor  
Arrest Date = 09-14-2019  
Last Appearance = 10-02-2019  
Next Appearance  
Purpose = Pre trial appearance  
Date = 10-07-2019  
Location = Queens County  
Modality = Virtual  
Case Closed = NO

Notice that last line at the end of the e-mail that says “Case Closed=NO.” Not every update will have that line, but when the case closes, an e-mail will be sent with “Case Closed=YES” as that last line.

CBS: Is that always the last e-mail that gets sent for a case?

SJ: Not necessarily. You might get a bail reimbursement e-mail after the case is closed, and in rare cases, you might get some final evidence filings. But once you get an e-mail stating the case is closed, it doesn't matter what comes after—you can claim your money back.

CBS: What about when bail is set, denied, or posted?

SJ: Bail set and bail denied e-mails are a little idiosyncratic. Those phrases are simply listed along with other information. Here's an example showing that bail was set:

Court Update Alert: Case Number CR-4135244 Inbox x

**cbs.python.courts@gmail.com**  
to me ▾  
**Sent Date = 09-19-2019 06:38:28**  
Case Number = CR-4135244  
Defendant Name = Gary Wilson  
DOB = 1992-10-29  
Sex = M  
Charge = Sexual abuse in the second degree- Class A Misdemeanor  
Arrest Date = 09-19-2019  
Last Appearance = 09-19-2019  
Next Appearance  
Purpose = Arraignment  
Date = 09-19-2019  
Location = New York County  
Modality = In Person  
Bail Set  
Bail = 550.0  
Case Closed = NO

Notice how the bail amount is listed as well. A bail denied update would look the same, but without the bail amount, obviously.

CBS: And bail posted?

SJ: That update is listed in the e-mail's purpose entry, such as in this example:

Court Update Alert: Case Number CR-6116704 Inbox x

**cbs.python.courts@gmail.com**  
to me ▾  
**Sent Date = 09-22-2019 08:47:32**  
Case Number = CR-6116704  
Defendant Name = James Holmes  
DOB = 1977-07-22  
Sex = M  
Charge = Stalking in the second degree- Class E Felony  
Arrest Date = 09-16-2019  
Last Appearance = 09-19-2019  
Next Appearance  
Purpose = Bail posted  
Date = 09-22-2019  
Location = Queens County  
Modality = Virtual  
Case Closed = NO

CBS: I see. Now, I notice that it doesn't actually mention the bail amount—I guess there's no need, because it must mean they just paid whatever bail was listed in the bail set e-mail.

SJ: That's right. Though there's one other situation in which we pick up a case on the day in the courtroom *after* bail has been set. In those situations, you'll see a bail posted e-mail, but you won't see a bail set e-mail, because it will have happened *before* we ever started tracking e-mails for the case. In those cases, you'll see the bail amount in a bail paid confirmation e-mail.

CBS: So does it make sense to say I should just ignore the bail set e-mails and look at the payment confirmations?

SJ: Yes and no, because sometimes we'll track a case from the start if we think it might be a good candidate for bail. In those cases, it's useful to know when bail has been set and for how much, and that would come in a bail set e-mail.

CBS: Got it—so keep track of both. One thing confuses me, though. Do you ever get inconsistent information? For example, do you ever find situations in which the bail payment receipt doesn't match the amount in the bail set e-mail? Or suppose you pay bail for someone—you would expect a bail posted update to come soon after. Do you ever get cases in which you post bail and an update never comes? Or could you ever get two bail posted e-mails?

SJ: No, thankfully that doesn't tend to happen. The court's eTrack system is old, but it works pretty consistently. Maybe down the line we would want to triple check for consistency in those ways you describe, but for now we're happy to trust that things will match.

CBS: And, unbelievably, you do all this manually right now?

SJ: Yes! We have someone sift through the in-box every day and update a master Google Sheets document we keep current with the status of every case. We're especially looking out for cases that were closed but that we had paid bail for—those are cases we need to collect on. I actually used to do all this by hand, believe it or not. This is an old version of my notes:

Case #	Closed	Last update	Bail set	Bail amt	Bail posted	objus?	Refunded?	date posted
CR-1060108	✓	9/1/19	Denied					
CR-1534741		9/15/19	✓	\$1,000	✓			
CR-7565517	✓	9/18/19	✓	\$4,000	✓	✓		9/1/19
CR-1746837		9/18/19	Denied					
CR-9854962	✓	9/5/19	✓	\$5,000	✓	✓		8/12/19
CR-9650527		9/22/19	✓	\$6,200	✓	✓		9/21/19
CR-9513722		9/25/19	Denied					
CR-1182...		10/1/19	/	\$7,000	/	/		



Here, I highlighted two cases in green. Those were closed cases for which we had paid the bail. (See the check marks in the “By us?” column and the date under the “Date posted” column, which lists the date we posted bail.) One of them is crossed out, because we got the bail back (see the check mark in the “Refunded” column).

CBS: Wow, we’re definitely going to make this process more efficient! So presumably, you would want the program to be able to output the most up-to-date version of this table at a click?

SJ: That would be a dream! And if you’re doing that anyway, maybe your program could also kick out our status report.

CBS: Status report?

SJ: It’s a weekly PowerPoint presentation we produce that lists some important statistics, including the total amount currently lent out for active cases (and the number of active cases), the total amount we’re eligible to claim back (such as those cases highlighted in green in my sample notes), and the total number of cases we helped over the lifetime of the fund. We then provide a list of every case eligible for refund plus a graph of how many cases we’ve been involved in on a weekly basis in the recent past.

CBS: That must be a nightmare to put together if you do everything manually!

SJ: Tell me about it. But you’re going to make all that better, right?

CBS: We’ll do our best!

## Appendix 2

### Homework

In class we discussed how to read every e-mail in the SBF's e-mail account using Python and save them into an Excel spreadsheet.

We then used these e-mails to update the status of each case being tracked by the fund.

After class, SBF will receive additional e-mails from the court system. Your task is to begin with the files we ended class with and to update them in light of those new e-mails.

Note that you could simply rerun the script from class, re-download every e-mail in the organization's in-box, and update the files that way. This, however, would be wasteful. Why re-download files that were already downloaded before?

The purpose of this homework is to figure out how you can download only new e-mails not previously read by SBF. One way you might go about this is by obtaining every single e-mail ID in the in-box, and then removing those that were already downloaded. You should then be able to download only new e-mails and merge the datasets.

### Parting Words

This concludes our very brief introduction to creating PowerPoint presentations in Python. The `python-pptx` package includes a vast number of additional features not discussed, including adding shapes, changing text and table formatting, adding images and charts, etc. The documentation's step-by-step guide explains how to incorporate these elements.

## Endnotes

---

- <sup>1</sup> Aaron Larson, "How Does Bail Work," *Expert Law*, April 4, 2018, <https://www.expertlaw.com/library/criminal-law/how-does-bail-work>, accessed February 7, 2022.
- <sup>2</sup> Stephanie Wykstra, "Bail Reform, Which Could Save Millions of Unconvicted People from Jail, Explained," *Vox*, October 17, 2018, <https://www.vox.com/future-perfect/2018/10/17/17955306/bail-reform-criminal-justice-inequality>.
- <sup>3</sup> Elise White et al., *Navigating the Bail Payment System in New York City* (New York: Center for Court Innovation, December 2015), <https://www.courtinnovation.org/sites/default/files/documents/Bail%20Payment%20in%20NYC.pdf>.
- <sup>4</sup> "Money Bail Must Be Abolished," Dollar Bill Brigade, accessed February 7, 2022, <https://www.dollarbailbrigade.com/about>.
- <sup>5</sup> Eric Helland and Alexander Tabarrok, "The Fugitive: Public versus Private Law Enforcement: Evidence from Bail Jumping," *Journal of Law and Economics* 47, no. 1 (April 2004), <https://www.jstor.org/stable/10.1086/378694>.
- <sup>6</sup> Katie Unger et al., *Selling Off Our Freedom: How Insurance Corporations Have Taken Over Our Bail System* (ACLU, May 2017), [https://www.aclu.org/sites/default/files/field\\_document/059\\_bail\\_report\\_2\\_1.pdf](https://www.aclu.org/sites/default/files/field_document/059_bail_report_2_1.pdf).
- <sup>7</sup> Louis Jacobson, "Are US, Philippines the Only Two Countries with Money Bail?" *PolitiFact*, October 9, 2018, <https://www.politifact.com/factchecks/2018/oct/09/gavin-newsom/are-us-philippines-only-two-countries-money-bail/>.
- <sup>8</sup> Larson, "How Does Bail."
- <sup>9</sup> Scott Shackford, "Innocence Until Proven Guilty, But Only If You Can Pay," *Reason Magazine*, August 2018, <https://reason.com/2018/07/14/innocent-until-proven-guilty-b/>.
- <sup>10</sup> Wendy Sawyer and Peter Wagner, *Mass Incarceration: The Whole Pie 2020* (Northampton, MA: Prison Policy Initiative, March 24, 2020), <https://www.prisonpolicy.org/reports/pie2020.html>.
- <sup>11</sup> Mustafa Z. Mirza, "Dallas County's Secret Bail Machine," Marshall Project, September 4, 2018, <https://www.themarshallproject.org/2018/09/04/dallas-county-s-secret-bail-machine>.
- <sup>12</sup> Anna Maria Barry-Jester, "You've Been Arrested. Will You Get Bail? Can You Pay It? It May All Depend on Your Judge," *FiveThirtyEight*, June 19, 2018, <https://fivethirtyeight.com/features/youve-been-arrested-will-you-get-bail-can-you-pay-it-it-may-all-depend-on-your-judge/>.
- <sup>13</sup> Megan Stevenson, *Distortion of Justice: How the Inability to Pay Bail Affects Case Outcomes*, (Philadelphia, PA: University of Pennsylvania, November 8, 2016), <https://www.econ.pitt.edu/sites/default/files/Stevenson.jmp2016.pdf>.

- 
- <sup>14</sup> John Mathews II and Felipe Curiel, “Criminal Justice Debt Problems,” *ABA Human Rights Magazine*, November 30, 2019, [https://www.americanbar.org/groups/crsj/publications/human\\_rights\\_magazine\\_home/economic-justice/criminal-justice-debt-problems/](https://www.americanbar.org/groups/crsj/publications/human_rights_magazine_home/economic-justice/criminal-justice-debt-problems/).
- <sup>15</sup> Kamala D. Harris and Rand Paul, “Kamala Harris and Rand Paul: To Shrink Jails, Let’s Reform Bail,” *New York Times*, July 20 2017, <https://www.nytimes.com/2017/07/20/opinion/kamala-harris-and-rand-paul-lets-reform-bail.html>.
- <sup>16</sup> Harris and Paul, “Kamala Harris and Rand Paul.”
- <sup>17</sup> Scott Kohler, “Vera Institute of Justice: Manhattan Bail Project: Ford Foundation, 1962,” Case 29 (Durham, NC: Center for Strategic Philanthropy and Civil Society [Duke], 2007), [https://cspcs.sanford.duke.edu/sites/default/files/descriptive/manhattan\\_bail\\_project.pdf](https://cspcs.sanford.duke.edu/sites/default/files/descriptive/manhattan_bail_project.pdf), accessed February 7, 2022.
- <sup>18</sup> Bail Reform Act of 1966, Pub. L. No. 89–465 (June 22, 1966), via Govinfo website, <https://www.govinfo.gov/content/pkg/STATUTE-80/pdf/STATUTE-80-Pg214.pdf>.
- <sup>19</sup> David N. Adair Jr., *The Bail Reform Act of 1984: Third Edition* (Washington, DC: Federal Judicial Center, 2006), <https://www.fjc.gov/sites/default/files/2012/BailAct3.pdf>.
- <sup>20</sup> United States, Petitioner v. Anthony Salerno and Vincent Cafaro, 481 U.S. 739 (1987), <https://www.law.cornell.edu/supremecourt/text/481/739>.
- <sup>21</sup> Brian A. Reaves, *Felony Defendants in Large Urban Counties, 2009 – Statistical Tables* (Washington, DC: US Department of Justice, 2013), <https://bjs.ojp.gov/content/pub/pdf/fdluc09.pdf>.
- <sup>22</sup> “Pretrial Justice Reform,” ACLU of New Jersey, accessed February 7, 2022, <https://www.aclu-nj.org/theissues/criminaljustice/pretrial-justice-reform>.
- <sup>23</sup> Megan Thompson and Mori Rothman, “New Jersey Eliminates Most Cash Bail, Leads Nation in Reforms,” *PBS News Hour Weekend*, PBS, July 22, 2017, <https://www.pbs.org/newshour/show/new-jersey-eliminates-cash-bail-leads-nation-reforms>, accessed February 7, 2022.
- <sup>24</sup> Taryn A. Merkl, “New York’s Upcoming Bail Reform Changes Explained,” Brennan Center for Justice, December 10, 2019, <https://www.brennancenter.org/our-work/analysis-opinion/new-yorks-upcoming-bail-reform-changes-explained>, accessed February 7, 2022.
- <sup>25</sup> *Preventive Detention in New York: From Mainstream to Margin and Back* (New York: NYU Law/Center on the Administration of Criminal Law, February 2017), [https://www.law.nyu.edu/sites/default/files/upload\\_documents/2017-CACL-New-York-State-Bail-Reform-Paper.pdf](https://www.law.nyu.edu/sites/default/files/upload_documents/2017-CACL-New-York-State-Bail-Reform-Paper.pdf).
- <sup>26</sup> Jesse McKinley and Jeffery C. Mays, “After Anti-Semitic Incidents, New Bail Law in N.Y. Comes Under Attack,” *New York Times*, January 8, 2020, <https://www.nytimes.com/2020/01/08/nyregion/cash-bail-reform-ny.html>.
- <sup>27</sup> Wykstra, “Bail Reform.”

- 
- <sup>28</sup> Israel Salas-Rodriguez and Ruth Weissmann, “Suspect Accused of Attacking Jewish Women Released without Bail,” *New York Post*, December 28, 2019, <https://nypost.com/2019/12/28/suspect-accused-of-attacking-jewish-women-released-without-bail/>.
- <sup>29</sup> McKinley and Mays, “After Anti-Semitic.”
- <sup>30</sup> “Police: Long Island Burglary Spree Suspect Released Under Bail Reform, Then Robs Another Store Hours Later,” *CBS New York*, January 4, 2020, <https://newyork.cbslocal.com/2020/01/04/long-island-burglary-bail-reform/>.
- <sup>31</sup> Todd Maisel, “As Felonies Rise, NYPD Blames Repeat Offenders Released with No Bail,” *AMNY*, March 5 2020, <https://www.amny.com/police-fire/as-felonies-rise-nypd-blames-repeat-offenders-released-with-no-bail/>.
- <sup>32</sup> McKinley and Mays, “After Anti-Semitic.”
- <sup>33</sup> Michael Schwirtz and Michael Winerip, “Kalief Browder, Held at Rikers Island for 3 Years Without Trial, Commits Suicide,” *New York Times*, June 8 2015, <https://www.nytimes.com/2015/06/09/nyregion/kalief-browder-held-at-rikers-island-for-3-years-without-trial-commits-suicide.html>.
- <sup>34</sup> Lauren-Brooke Eisen and Ames Grawert, “Consider All Data — and Coronavirus — Before Changing New York’s New Bail Law,” *Brennan Center for Justice*, March 27, 2020, <https://www.brennancenter.org/our-work/analysis-opinion/consider-all-data-and-coronavirus-changing-new-yorks-new-bail-law>, accessed February 7, 2022.
- <sup>35</sup> Lauren Krisai, Jason Pye, and Norman Reimer, “Bail Reform May Make New York Safer,” *Slate*, January 2, 2020, <https://slate.com/news-and-politics/2020/01/new-york-bail-reform.html>.
- <sup>36</sup> Diana Dabruzzo, “New Jersey Set Out to Reform Its Cash Bail System. Now, the Results Are In,” *Arnold Ventures*, November 14, 2019, <https://www.arnoldventures.org/stories/new-jersey-set-out-to-reform-its-cash-bail-system-now-the-results-are-in/>, accessed February 7, 2022.
- <sup>37</sup> Rafael A. Mangual, “How New Jersey Did Bail Reform Better Than New York,” *New York Post*, January 11, 2020, <https://nypost.com/2020/01/11/how-new-jersey-did-bail-reform-better-than-new-york/>.
- <sup>38</sup> Taryn A. Merkl, “New York’s Latest Bail Law Changes Explained,” *Brennan Center for Justice*, April 16, 2020, <https://www.brennancenter.org/our-work/analysis-opinion/new-yorks-latest-bail-law-changes-explained>, accessed February 7, 2022.
- <sup>39</sup> Tom Simonite, “Algorithms Were Supposed to Fix the Bail System. They Haven’t,” *Wired*, February 19, 2020, <https://www.wired.com/story/algorithms-supposed-fix-bail-system-they-havent/>.
- <sup>40</sup> “Updated Position on Pretrial Risk Assessment Tools,” *Pretrial Justice Institute*, open letter, February 7, 2020, <https://university.pretrial.org/HigherLogic/System/DownloadDocumentFile.ashx?DocumentFileKey=b417a859-9fe9-2a6c-5e12-3f472d0dc997&forceDialog=0>, accessed February 7, 2021.

---

<sup>41</sup> “More Than 100 Civil Rights, Digital Justice, and Community-Based Organizations Raise Concerns About Pretrial Risk Assessment,” Leadership Conference on Civil and Human Rights, July 30, 2018, <https://civilrights.org/2018/07/30/more-than-100-civil-rights-digital-justice-and-community-based-organizations-raise-concerns-about-pretrial-risk-assessment/>, accessed February 7, 2022.

<sup>42</sup> Chelsea Barabas et al., “Technical Flaws of Pretrial Risk Assessments Raise Grave Concerns,” open letter, July 2019, [https://dam-prod.media.mit.edu/x/2019/07/16/TechnicalFlawsOfPretrial\\_ML%20site.pdf](https://dam-prod.media.mit.edu/x/2019/07/16/TechnicalFlawsOfPretrial_ML%20site.pdf), accessed February 7, 2022.

<sup>43</sup> Wykstra, “Bail Reform.”

<sup>44</sup> Alwyn Scott and Suzanne Barlyn, “US Bail-Bond Insurers Spend Big to Keep Defendants Paying,” Reuters, March 26, 2021, <https://www.reuters.com/business/us-bail-bond-insurers-spend-big-keep-defendants-paying-2021-03-26/>.

<sup>45</sup> Alan Feuer, “New Jersey Is Front Line in a National Battle Over Bail,” *New York Times*, August 21, 2017, <https://www.nytimes.com/2017/08/21/nyregion/new-jersey-bail-reform-lawsuits.html>.

<sup>46</sup> Jim Walsh, “Federal Appeals Court Upholds NJ Bail-Reform Measures,” *Courier-Post*, July 9, 2018, <https://www.courierpostonline.com/story/news/crime/2018/07/09/new-jersey-bail-reform-brittan-holland-lexington/768009002/>.

<sup>47</sup> Robin Steinberg, Lillian Kalish, and Ezra Ritchin, *Freedom Should Be Free: A Brief History of Bail Funds in the United States* (Venice, CA: The Bail Project, 2018), via eScholarship, <https://escholarship.org/uc/item/37s1d3c2>, accessed February 7, 2022.

<sup>48</sup> “How It Works,” Massachusetts Bail Fund, accessed February 7, 2022, <https://www.massbailfund.org/how-it-works.html>.

<sup>49</sup> “Bail Funds by State,” B Lab US & Canada, accessed February 7, 2022, <https://usca.bcorporation.net/news/bail-funds-state>.

<sup>50</sup> “The Bail Project Frequently FAQ,” The Bail Project, <https://bailproject.org/faq/>, accessed March 2022.

<sup>51</sup> Alison Smith, “State Money-Bail Systems: Differing Approaches,” Congressional Research Service, November 27, 2018, <https://sgp.fas.org/crs/misc/LSB10220.pdf>.

<sup>52</sup> Data scientist, anonymous by request (a New York City bail fund employee), in discussion with Columbia Business School.



ID#220205SM1

PUBLISHED ON  
MAY 18, 2022

---

## Automating Bureaucracy with Python: The Case of the Springfield Bail Fund

BY C. DANIEL GUETTA

---

### Appendix 3

#### Technical Primer: OAuth 2

##### INTRODUCTION

Most users are accustomed to accessing online services using password authentication—logging in to a website with a username and password. In this case, we will require a Python script to access an e-mail inbox. In principle, we could simply provide our script with the Gmail account’s username and password, giving it all the rights it needs.

However, there are several significant issues with this approach. First and foremost, passwords are an all-or-nothing authentication technique. Anyone with someone’s Gmail credentials can do anything with that account, from reading and sending e-mails to making purchases or even changing its password.

Second, passwords don’t allow granular access controls. Suppose you give access to two or more apps but then decide to revoke only one app’s access. If both of them have your password, there’s no way to do this—you would have to change the password, and then give the new password to all the apps that should still have access.

Third, passwords don’t allow you to know for sure which app did what on your account. Imagine granting access to a number of apps by giving them your password and later discovering that unbeknownst to you a large purchase was made on your account and each app denies having made the purchase. There would be no way for you to know which app is to blame, because each one would have access to your account using the same technique.

This is why Google has not allowed use of this authentication technique since 2015—even if you were to give your username and password to an app, it wouldn’t be able to access your Google account programmatically using these details. (As of the time this case was written, there are ways to get around that, but we won’t go into them here).

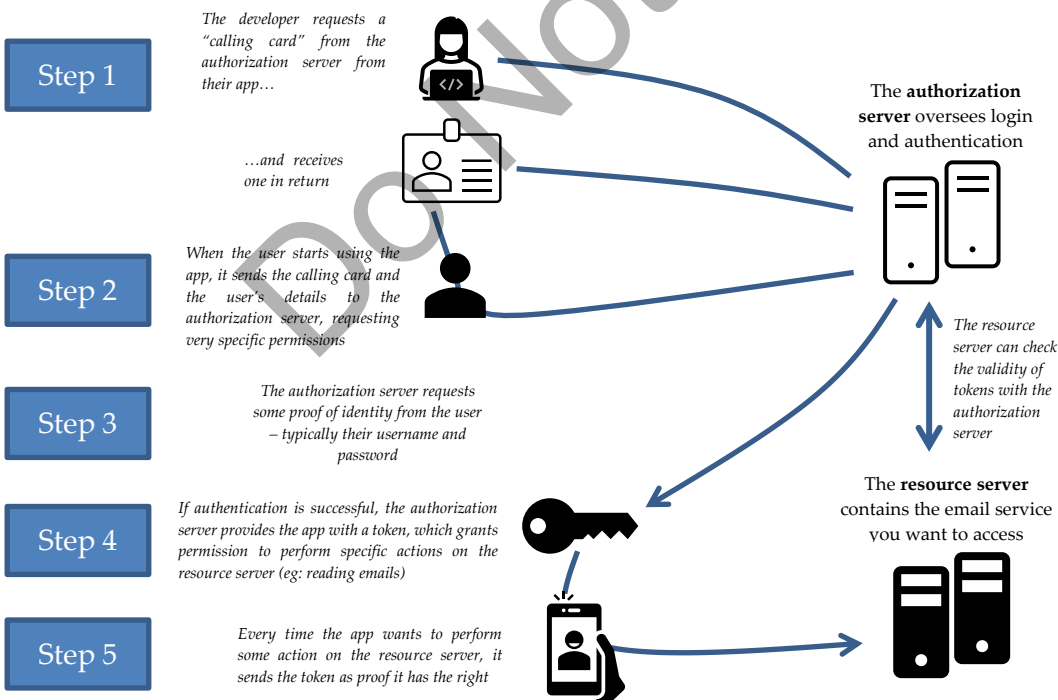
## ENTER OAUTH2

Instead, Google uses a far more robust authentication technique called Open Authorization, Version 2 (OAuth2). It is an open standard for access delegation, commonly utilized when users want to grant limited access to their information on other websites or apps.

The steps for getting an app authorization using OAuth2 are as follows (to see Google's official instructions, click here: [documentation](#)):

1. Request OAuth2 client credentials from Google for your app. These credentials are linked to a specific app/project; in some sense, they are your app's calling card to Google.
2. When your app runs, it presents its calling card to Google and requests very specific permissions (for example, permission to read or send e-mails or create calendar events).
3. Google will then pop up a browser window that asks the user to grant the app the specific permissions it has requested.
4. Assuming the user grants permissions, Google sends a token back to the app.
5. The app then uses this token as a password to do the specific things you gave it permission for.

The following figure illustrates these steps, which are often mediated by two separate servers – an *authorization server*, which oversees login and authentication, and a *resources server*, which contains the actual email service we care about.



Note that OAuth2 is [ubiquitous](#). It is also used by Facebook, Amazon, Microsoft, and Twitter. We will use OAuth2 to gain access to Google's Gmail API on behalf of the bail fund, but you could also use it to gain access to hundreds of other Google APIs, and many more beyond.

In fact, you've almost certainly granted permission to your Facebook profile using an OAuth workflow before. For example, when logging into the Tinder app using Facebook, the app redirects you to Facebook and asks you to grant permission to access your profile information. When it does this, Tinder is requesting a token from Facebook, which it then uses to pull pictures and other data from your Facebook profile.

This protocol prevents all the problems we mentioned above that could occur if you gave an app access to your account directly by providing them with your Google password. Each token only grants access to specific resources, and tokens can be revoked for one app only, with all other tokens remaining valid. In fact, you can see all the tokens you have ever issued to your Google and Facebook accounts and revoke them. Here's how:

- In Google, go to [google.com](https://google.com), click on your picture on the top right-hand corner, click on "Manage your Google account," choose the Security option on the menu on the left, and scroll down to the section entitled "Third-party apps with account access." If you click on "Manage third-party access," you'll see every token you ever issued. To revoke access, just click on any item.
- In Facebook, click on the down arrow at the top right corner of the Facebook page, click on Settings & Privacy, then Settings, and then select Apps and Websites in the menu on the left.

## ACCESSING THE GOOGLE API USING PYTHON

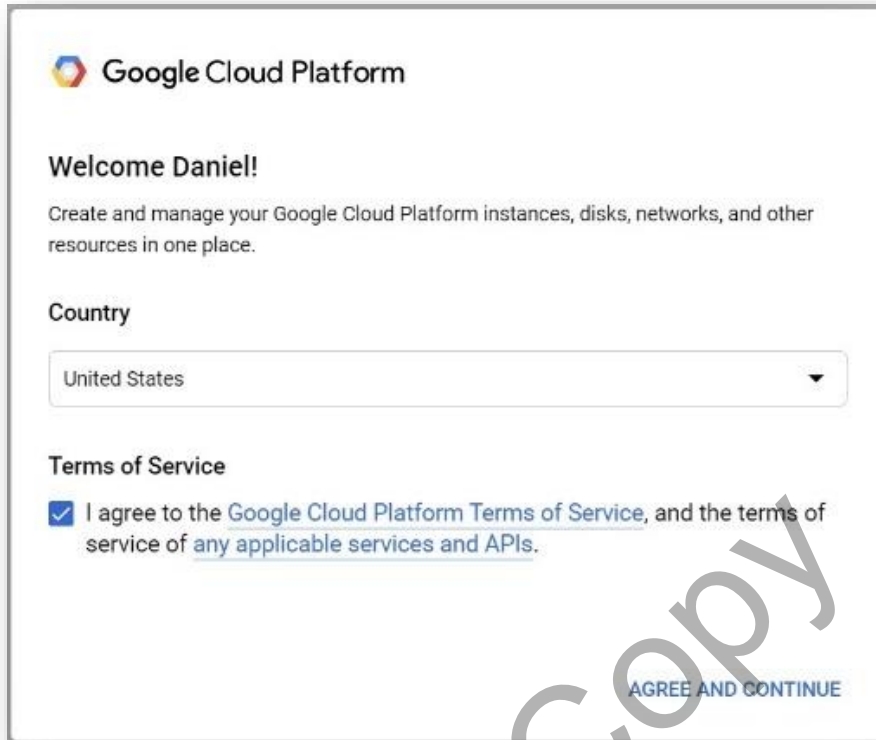
In this section, we will teach you how to register a project with Google, download client credentials, and then use Python to obtain a token using those credentials.

Note that if you are covering the Springfield Bail Fund Case in class, you should have been given a file called `client_credentials.json`, and you can skip to the last subsection in this section.

## Creating a Project

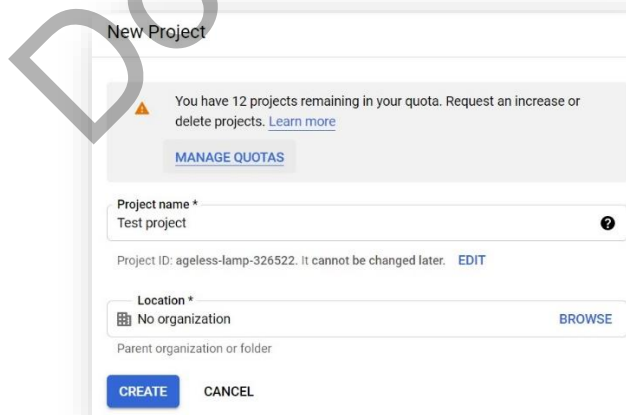
To begin, we need to create a Google Project, which is a way to register your app on the Google Cloud Platform—it is this project that will be requesting a token and access to the Google API.

Open the Google Cloud Platform Console at <https://console.cloud.google.com/>. If this is your first time using the console, you will be asked to select your country and to accept the terms and conditions:



You will then be asked how you intend to use the project (feel free to answer the question or skip it).

Next, create a project by clicking “Select a project” near the top left corner of the page. In the pop-up window, click New Project on the top right. In the resulting dialogue box, choose a project name. Google will automatically generate one for you.



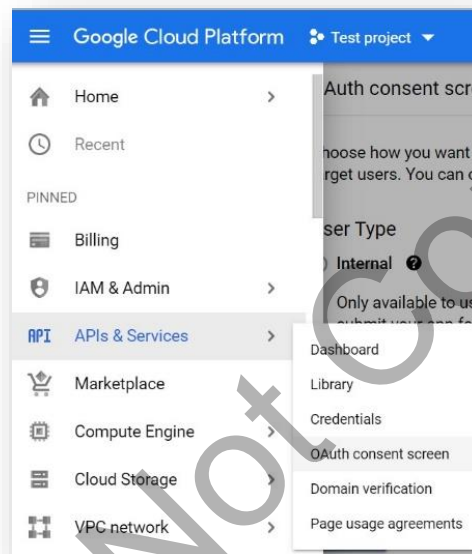
Click Create. A few seconds later, your project will be ready.

## Requesting Client Credentials

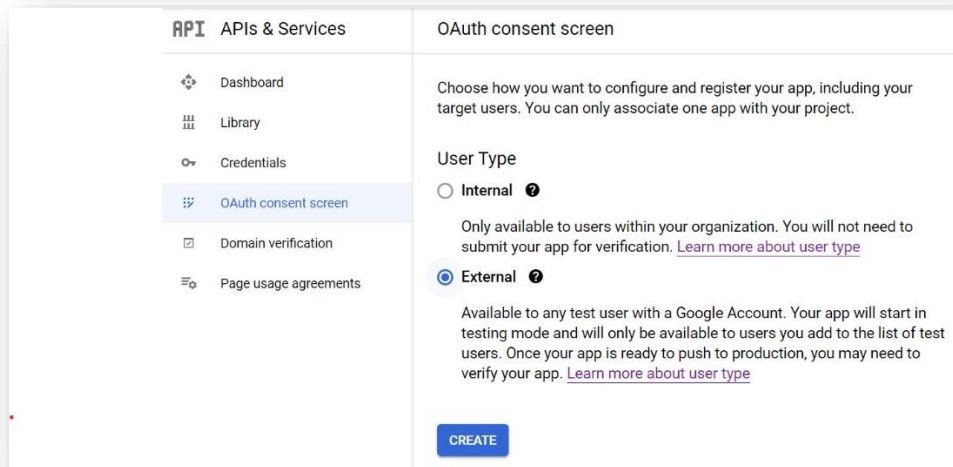
The next step is to download client credentials—your app's calling card. Your Python script will send these credentials to Google to obtain a token.

When you return to the Google Cloud Platform Console at <https://console.cloud.google.com/>, the new project you created should be selected by default.

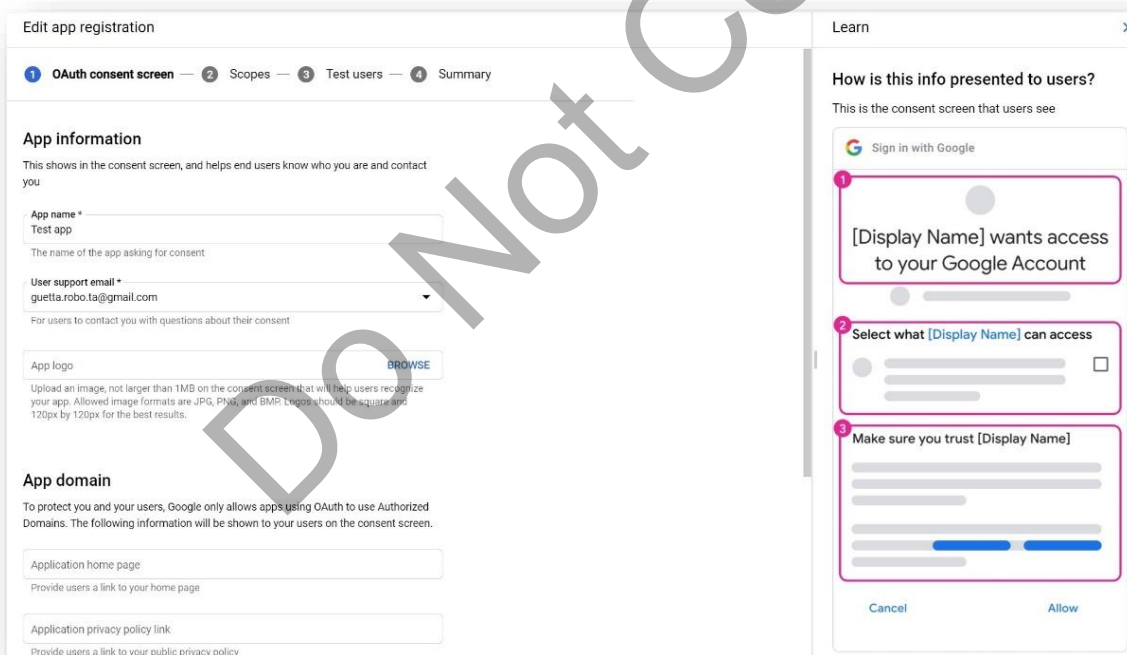
Before downloading credentials, you will need to configure the OAuth consent screen, which is presented to the user when an app requests a token. Click on the three lines at the top left of the window to open the navigation menu, hover over the APIs & Services entry, and select “OAuth consent screen”:



You will then be asked to choose whether the app is an internal or external one. Internal apps are meant for Google Workspace users who want to design apps that will only be accessed by others at their company in the same Google Workspace. You will likely be using a personal account to carry out these steps, so you should select External, then click Create:



The next screen asks you to select a name for the app and an e-mail address users can contact if they have issues with the app. The column on the right shows a preview of what the consent screen. It should look familiar if you've ever granted apps a Google token in the past:



Once you've entered these details, you can leave the rest of the form blank. Scroll to the bottom of the page, enter your e-mail address, and click Save and Continue.

At this point you select the “scopes” you would like the app to have access to. Scopes is Google's name for the specific permissions an app might request. For example, the

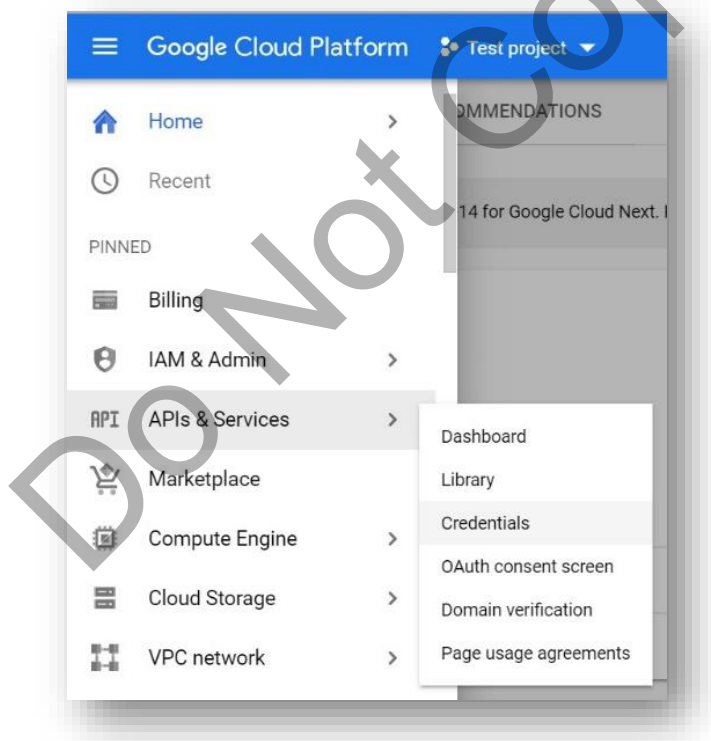


<https://www.googleapis.com/auth/gmail.readonly> scope allows your API to read e-mails but not send them. You can scroll to the bottom of this page and click Save and Continue without selecting any scopes as we will specify the correct scopes in our code.

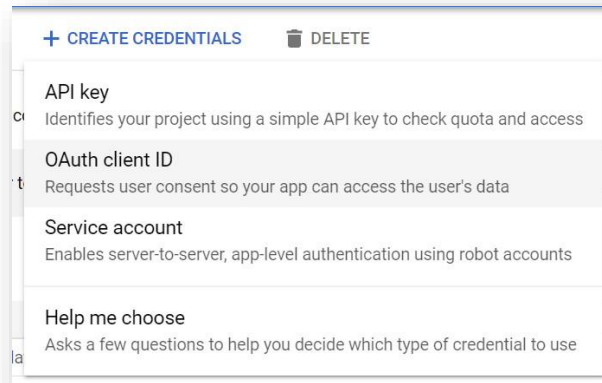
The next screen asks you to specify the users allowed to use the app. In a real app, any user could use the app and request a token from their account (just like any user can sign up for Tinder and give permission for the app to access their account). To prevent people from creating malicious apps, Google will limit the number of users who can use the app to 100 until verified by Google. (In this case, it shouldn't be a problem since you will be employing this app to get programmatic access to one specific e-mail account—the Springfield Bail Fund account). Simply click on Add Users, enter your e-mail address, click Add, and then click Save and Continue.

Scroll to the bottom of the summary screen that appears and click on Back to Dashboard.

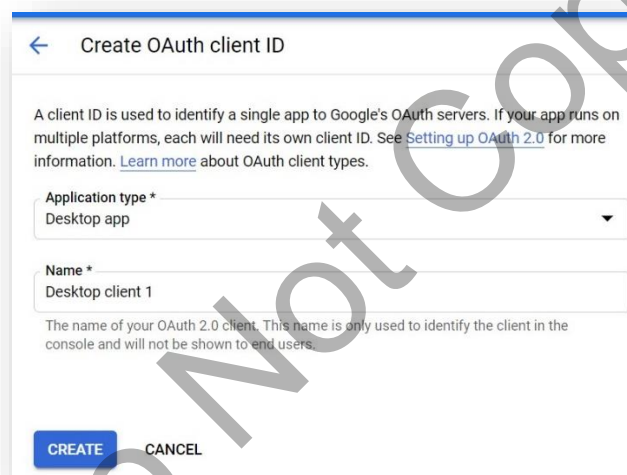
We're now ready to download our credentials. Once again, click on the three lines at the top left of the window to open the navigation menu, hover over APIs & Services, and select Credentials.



At the top of the page, select Create Credentials and OAuth Client ID:



In the Application Type drop-down, select Desktop App (since you will be using this app in a Python script on your desktop). Pick a name and click Create.



You will then see a screen that lists two long pieces of text—these are the client credentials you will use to request a token from Google. Click on Download Json, and save the file as `client_credentials.json` somewhere safe—this file will contain everything you need to make that request. (By itself, however, the file doesn't grant any access; it's simply the app's calling card.)

## Getting a Token in Python

Important note: the steps in this section will generate a token with access to a real Google account. In class, we will only generate tokens for the experimental Google accounts associated with the fictitious Springfield Bail Fund case. *If you generate a token for your own personal account, you do so at your risk and peril, as anyone with that token can access your account.*

You should now have a file called `client_credentials.json`. We will go through the process of requesting a token from Google using these credentials.

First, you will need to install a handful of packages by running the following command wherever you usually run `pip`:

```
pip install --upgrade google-api-python-client google-auth-httplib2 google-auth-oauthlib
```

These packages will provide functions that will allow us to easily interact with the Google API. Import them in your code:

```
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
```

Google scopes—the specific permissions you want to grant for a token—need to be provided to Google as a list of strings, where each string contains one scope. Here, we only want permission granted to read e-mails, and we will define a list with one item only:

```
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']
```

You can see what other scopes are available for access to Gmail [here](#).

Next, we need to create an authentication flow object. This is the process by which the app will present its client credentials to Google and get a token in response. The packages you've just imported make this easy:

```
flow = InstalledAppFlow.from_client_secrets_file('client_credentials.json', SCOPES)
```

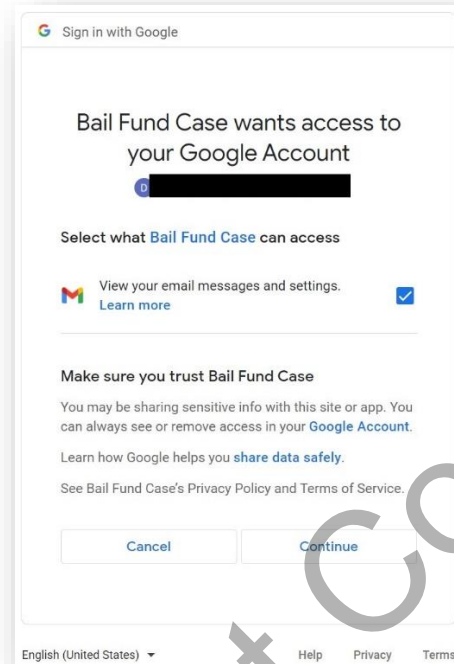
This instructs Python to create an authentication flow object, using the client credentials you downloaded into your `json` file, to request the scopes you specified.

We're now ready to get the token. Run the following line:

```
creds = flow.run_local_server()
```

This should open up a new window that will allow you to carry out the authorization flow. Select the account you would like your app to have authorization to. As you might remember from setting up our client credentials, our app is still in testing mode and can therefore only get access to the specific user accounts you listed when you created the client credentials. This is fine for our purposes, since we'll only be accessing the bail fund e-mail address.

Select that account (or log in to it using the details provided by your instructor). You will then see a blood-curdling warning informing you Google has not verified this app yet but go ahead and click Continue. Finally, you will see a window listing all the scopes your script has requested, with a checkbox next to each one. Put a check next to each box, and click Continue:



When a message appears that the authentication flow has completed, you can then close the window.

When you return to Python, you will find your code has run. The `creds` variable will now contain the token your script will need to access the Google API with the scopes requested. You can then connect to Gmail using this token as follows:

```
service = build('gmail', 'v1', credentials=creds)
```

You can now use `service` to interact with Google (we will discuss how in a later appendix).

One last note: it would be a little inconvenient to have to go through this process every time you restart your Python kernel. Luckily, there is a way to save the token in a file. The first step is to run the following line of code:

```
creds.to_json()
```

The result of this line of code contains the token you just created. *Be incredibly careful—treat this token like a password, because it is.* (Note that it will include an expiration date.)

We could copy and paste this token directly into our code, and then use it to authenticate with Google rather than go through the whole flow procedure:

```
# Note: DO NOT DO THIS!
token = {"token": "ya29.a0ARrdaM",
        "refresh_token": "1//0d6hgIQqpt092CgYIARAAGA0",
        "token_uri": "https://oauth2.googleapis.com/token",
        "client_id": "172124637047-
apa2crsnseeugeinle6rp92bml7c77ro.apps.googleusercontent.com",
        "client_secret": "2JSYRxAyrce8zHPj-2pAead",
        "scopes": ["https://www.googleapis.com/auth/gmail.readonly"],
        "expiry": "2021-09-20T18:58:27.496896Z"}

SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']
creds = Credentials.from_authorized_user_info(token, SCOPES)
service = build('gmail', 'v1', credentials=creds)
```

This is, of course, a *terrible idea*. Putting a token in your code means you might inadvertently reveal it to someone else when you send them your code. It would be much better—and safer—to save the token in a file as follows:

```
with open('token.json', 'w') as f:
    f.write(creds.to_json())
```

A file called `token.json` will be created containing the token. You can then authenticate with Google, using the file directly:

```
SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']
creds = Credentials.from_authorized_user_file('token.json', SCOPES)
service = build('gmail', 'v1', credentials=creds)
```

Return to the tokens granted section in the bail fund Gmail account (as described above) to observe this that new token now appears on the list.

## Getting a Token: A More Streamlined Approach

We can combine the steps described in the previous function into `gmail_connect`, a more streamlined function that checks whether a token file currently exists, loads it if it does, and obtains a new token if not. This function also requires the `os.path` package to work (to check whether or not the file exists).

```
import os.path
from googleapiclient.discovery import build
```

```

from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials

def gmail_connect(client_credentials_file='client_credentials.json',
                  token_file='token.json'):
    """
    This function establishes a connection to the Gmail API for reading
    only. It takes two arguments
    - client_credentials_file: a json file containing client credentials
    from Google
    - token_file: a json file containing previously obtained tokens—if
    this file exists and is valid, it will be used to establish a
    connection. If not, a new connection will be established

    The function returns a Google service object that can be used to access
    the Gmail API.
    """

    # Specify the read-only scope we'll want
    SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']

    # Begin with empty credentials
    creds = None

    # Check whether the token file exists
    if os.path.exists(token_file):
        # Attempt to connect using this token file
        creds = Credentials.from_authorized_user_file(token_file, SCOPES)

    # If valid credentials were not obtained, get new ones
    if not creds:
        flow = InstalledAppFlow.from_client_secrets_file(
            client_credentials_file, SCOPES)
        creds = flow.run_local_server(port=0)

        # Save the credentials for future use
        with open(token_file, 'w') as f:
            f.write(creds.to_json())

    # Authenticate to Gmail with this token and return the Google service
    service = build('gmail', 'v1', credentials=creds)
    return service

```



ID#220205SM2

PUBLISHED ON  
MAY 18, 2022

# Automating Bureaucracy with Python: The Case of the Springfield Bail Fund

BY C. DANIEL GUETTA

## Appendix 4

### Technical Primer: Reading E-Mails from Gmail

#### INTRODUCTION

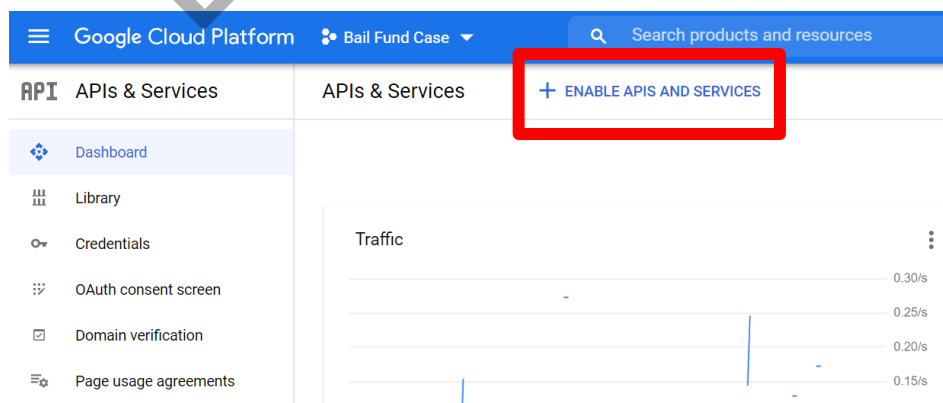
This appendix describes the steps required to connect to the Gmail API and read e-mails from it. These techniques will help streamline the work required for the bail fund case.

Note that the programmatic manipulation of e-mail using Python is an enormous subject. Here, the focus will center on those aspects needed for the case while offering suggestions on where to go for further information. Though this primer covers how to read e-mails, not how to send them, some of the material in this section applies to the method of sending them as well.

#### ENABLING THE GMAIL API


By default, Google disables the ability to access your e-mail in-box using an API—for understandable reasons—and we recommend you leave it disabled for your personal in-box unless you know what you're doing. However, for the bail fund case, the API must be enabled before following the instructions in this appendix.

First, go to <https://console.developers.google.com/>. Near the top right of the page, click on Enable APIs and Services.




In the search box, type “Gmail API” and click on it in the search results.

“gmail api”



**Gmail API**  
Google Enterprise API ⓘ  
The Gmail API lets you view and manage Gmail mailbox data like threads, messages, and labels.

On the resulting page, click Enable.



## Gmail API

Google Enterprise API

Flexible, RESTful access to the user's inbox

**ENABLE** TRY THIS API [↗](#)

### CONNECTING TO THE GMAIL API

Before you begin, you must connect to the Gmail API using the procedures detailed in the technical primer in Appendix 3. The following instructions assume you have already produced a variable called `service` in Python, which allows you to connect to the API. After creating the `gmail_connect` function described above, you can set it up as follows:

```
service = gmail_connect()
```

### LISTING MESSAGES IN A GMAIL USER'S ACCOUNT

To list every message, run the following line in Python:

```
msgs = service.users().messages().list(userId='me').execute()
```

The syntax of this line of code is idiosyncratic to the specific package we're using to access the Gmail API. Notice the single argument to the list function, which specifies that we'd like to

access the in-box as ourselves (this parameter is included for compatibility with other API calls—in this case, it would make no sense to list a different person’s e-mail).

The variable `msgs` will now contain a dictionary with three keys: `messages`, `nextPageToken`, and `resultSizeEstimate`.

Looking at `msgs['messages']`, you’ll find that it is a 100-item list. Each item in the list has the following form:

```
{'id': '17c00950da89e277', 'threadId': '17c00950da89e277'}
```

You’ll see the function has received 100 messages from this Gmail user’s account, along with the ID of the message and the ID of the thread the message belongs to (in Gmail, a thread is a grouping of messages—every line you see in your in-box is a thread).

So far, so good. But clearly, this in-box must contain more than 100 messages. How do we get the rest? The clue is in the `nextPageToken` entry in the `msgs` dictionary. It contains a long number that points to the next page of results.

Why doesn’t the Gmail API return every single message all at once? The answer might be obvious when you realize the average Gmail in-box contains tens of thousands, if not hundreds of thousands, of messages—the list would be far too large to download with one function call.

We now have a pointer to the next page of results. How do we retrieve it? We simply use the list function again:

```
msgs = service.users().messages().list(userId = 'me', pageToken=msgs['nextPageToken']).execute()
```

Notice that `pageToken` argument, to which we pass the next page token. The results in `msgs` now contain the next page of results.

This new set of results will—once again—contain a `nextPageToken`, which we can use to get the next page, and so on. We know we’ve reached the last page when the result doesn’t contain a `nextPageToken` entry.

Again, so far, so good. But if we repeat the line above over and over, the `msgs` variable will simply get overwritten each time. What if we’d like to create a master list of all the messages in our Gmail account? The following code will do the job:

```

msgs = service.users().messages().list(userId='me').execute()
message_list = msgs['messages']

while 'nextPageToken' in msgs:
    msgs = ( service.users()
            .messages()
            .list(userId = 'me',
                  pageToken = msgs['nextPageToken']))
            .execute() )
    message_list.extend(msgs['messages'])

```

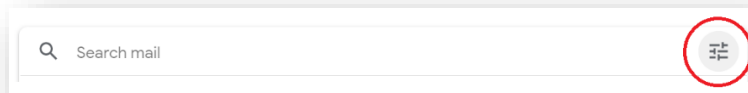
This code works as follows:

- The first line loads the first page of results.
- The next line saves the messages in that first page to a `message_list` variable.
- The next part of the code is a loop that will repeat as long as the `msgs` variable contains a `nextPageToken` argument in its result. During each loop, it'll obtain the next page of results and save the messages to the `message_list` variable. When the `nextPageToken` is `None`, this loop will end.

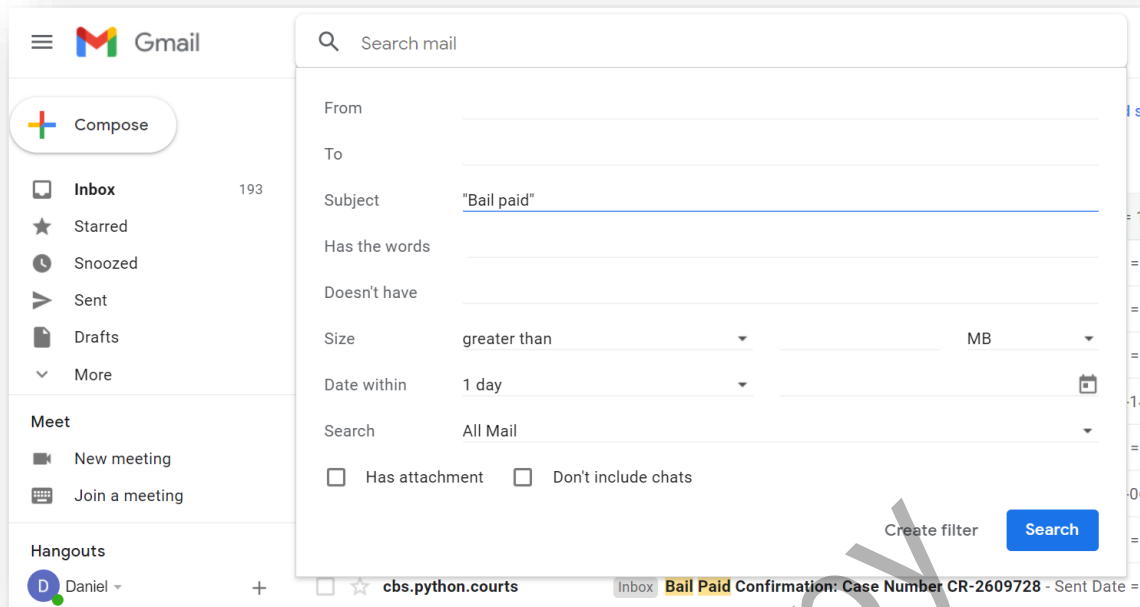
## Filtering Messages Before Retrieving Them

We have thus far discussed how to retrieve every single message in an in-box. This is useful, but sometimes we might want to retrieve only some of the messages. Of course, we could simply download them all and then do the filtering ourselves, but why waste time doing that when Google has far more powerful servers that do the filtering for us? (There are other reasons we wouldn't do it ourselves—not least is the fact Google imposes usage limits on the Gmail API, partly to prevent misuse.)

Suppose, for example, we wanted to download all messages that corresponded to bail payments. Peruse the Gmail in-box and you'll notice all those e-mails have the words “Bail paid” in the subject. How you would search for these e-mails directly in Gmail? You could easily do this by clicking in the search bar at the top of Gmail, then clicking on the little toolbox at the right of the bar, as shown here:



Since we are looking for e-mails with the exact words “Bail paid” in the subject line, the search terms would be those listed below:



Once you click search, the search bar will now contain the search string you need:



To retrieve only the messages that match the results of this search in Python, insert this string inside the list function:

```
msgs = service.users().messages().list(userId='me', q='subject:(\"Bail Paid\")').execute()
```

Done! If the results extend to more than one page, the `nextPageToken` trick described earlier can be used again.

It is, however, essential for you to pass the same `q` parameter in every `list()` call you use to retrieve later pages.

### READING RETRIEVED E-MAILS

The methods we have discussed allow us to retrieve e-mail and thread IDs, but we—as of yet—have no way to figure out what is inside these e-mails. In this section, we discuss how to retrieve these details.

But before we can do this, we have to understand how e-mails are structured.

## Understanding the MIME E-Mail Format

In an ideal world, every e-mail would simply consist of text. We could then easily retrieve that text using Python. In reality, things aren't quite that simple—e-mails can also contain attachments, images, and many other components. The Internet standard used to transmit complex e-mails is the Multipurpose Internet Mail Extensions (MIME) format.

MIME messages consist of multiple MIME parts—think of these as blocks that make up a message. There are many types of blocks, including:

- `multipart/alternative` contains lots of other blocks.
- `text/plain` and `text/html` contain the actual text of e-mails. When you write in a modern e-mail program, it usually uses HTML format with formatting (e.g., fonts, bolding, images). Your e-mail client will send this HTML code in a `text/html` block (unless it contains no formatting at all). Some older e-mail clients cannot read these HTML blocks (rare these days but more common historically). To ensure these older clients can still read the e-mail, your e-mail client will also produce a plain text version (without any formatting) and transmit it in a `text/plain` block as well. These are basically two copies of the same e-mail—one formatted and one unformatted.
- `application/pdf` contains a PDF attachment.
- `application/jpeg` contains an image attachment.

How can you retrieve all these parts from Gmail given a specific message ID? Rather than go through every single kind of e-mail we might receive (which would take hours and not be particularly illuminating), we'll look at retrieving specific e-mails from the bail fund e-mail address. This will give you a solid base for compiling more complex e-mails.

## Reading an E-Mail from Gmail

Let's practice with the first e-mail in the bail fund in-box, which we retrieved above. Looking at `message_list[0]` returns something like this (your specific message ID might be different, of course):

```
{'id': '17c00950da89e277', 'threadId': '17c00950da89e277'}
```

Armed with the message ID, we can retrieve it from Gmail:

```
msg = service.users().messages().get(userId='me', id=message_list[0]['id']).execute()
```

The `msg` string will now contain a dictionary with a number of keys, including:

- `id` and `threadId`, which contain those two IDs
- `labelIds`, which contain the IDs of all the labels assigned to this e-mail (This primer doesn't really get into label management, but you should be able

to figure out how to use them using Stack Overflow and the documentation here.)

- snippet, which contains the short line of body text from the e-mail that displays in your in-box when you preview the e-mail
- payload (the bit we're really interested in), which contains a single MIME part of type `multipart/alternative`—the block containing all the data in our e-mail

We now need to dig into the `payload` key, which is going to contain the actual content of our e-mail. If you initially look at that dictionary, it's going to look like a hot mess. This is because it contains a lot of information, such as content of the e-mail, how it was delivered, through what servers, and the steps Gmail took to authenticate the sender. (All useful information but unnecessary for our task.)

The following diagram should help you navigate this payload dictionary:



Let's examine what's going on here:

- As mentioned above, the payload is a single MIME part, which is represented by a dictionary. Notice that the `mimeType` entry of the dictionary contains



multipart/alternative; this tells us the e-mail building block will contain lots of other building blocks within it.

- The `MIME` part also contains a `headers` entry. This is a list that contains a bunch of dictionaries, each with a `name` entry and a `value` entry. As you'll see above, these dictionaries contain essential information about the e-mail: sender, recipient, subject, and date. If someone was cc'd on the e-mail, there would also be a `Cc` header.
- The `parts` entry is a list that will contain all the subblocks in the e-mail. In this case, there are only two subblocks; one contains the plain text version of the e-mail, the other contains the full formatted html version of the e-mail (which we discussed above). If there were any attachments, they would appear as additional blocks here.
- Each of these blocks contains a lot of information, but we'll focus on the `body` element and within it the `data` element—this contains the actual message.

In theory, we now have a way to retrieve our message! If we want the full HTML version, all we need to do is go into the payload, access the second part (which is `[1]`, the html version), and look at the data in the body:

```
msg['payload']['parts'][1]['body']['data']
```

## Decoding the E-Mail Text

At this stage, you might be a little befuddled to see that after all the hard work put in, the line above does not, at last, give us the content. Instead, you get a bunch of gobbledygook.

There's a good reason for this. In most of the work so far, Python strings have only involved Latin characters, numbers, and symbols (such as `%`, `$`, `&`, etc.) But there are many more characters that often appear in e-mails—characters from different languages (Chinese, Hebrew, Hindi, etc.), emojis, and all kinds of other characters. These are harder to include in a simple string. For that reason, Gmail uses a more complex way to represent characters that can encompass all these possibilities.

Explaining the ins and outs of the encoding technique goes beyond the scope of this note. For our purposes, it will be enough to use the following function to convert the gobbledygook to an intelligible message. It uses the `Base64` package to do the heavy lifting:

```
import base64
def decode_email(s):
    return base64.b64decode(s.replace('-', '+').replace('_', '/')).decode('utf-8')
```

Armed with this function, we can finally get our desired outcome:

```
decode_email(msg['payload']['parts'][1]['body']['data'])
```

Success! The result will be a piece of HTML that you can analyze using web scraping techniques you may already be familiar with. You could also replace the [1] in the line with a [0] to access the plain text version of the e-mail without the HTML formatting.

Do Not Copy

ID#220205SM3

PUBLISHED ON  
MARCH 15, 2022

# Automating Bureaucracy with Python: The Case of the Springfield Bail Fund

BY C. DANIEL GUETTA

## Appendix 5

### Technical Primer: Creating PowerPoint Files in Python

In your introduction to Pandas, you should have been introduced to techniques for saving Pandas DataFrames as Excel files.

In this primer, we will discuss how to create PowerPoint files in Python using the `python-pptx` package. As with the other primers in this case, we will only introduce the basics of the package. This will set up a solid foundation for further exploration of the package's documentation.

#### INSTALLING AND IMPORTING THE PYTHON POWERPOINT PACKAGE

You can install the package using a standard `pip` command (remember to add an exclamation point before the statement if you are working from Jupyter notebook):

```
pip install python-pptx
```

You can then import it:

```
import pptx
```

#### CREATING A NEW POWERPOINT PRESENTATION

Creating a presentation in Python is as easy as running a single line of code:

```
pres = pptx.Presentation()
```

This simply creates a new, blank presentation, containing no slides. If there were any slides in the deck, you could access them using `pres.slides`. Running `len(pres.slides)` now will return 0, since there are no slides.

To open an existing PowerPoint presentation, simply insert the file name inside the parenthesis above. By the end of this primer, you will know how to create and edit a presentation.

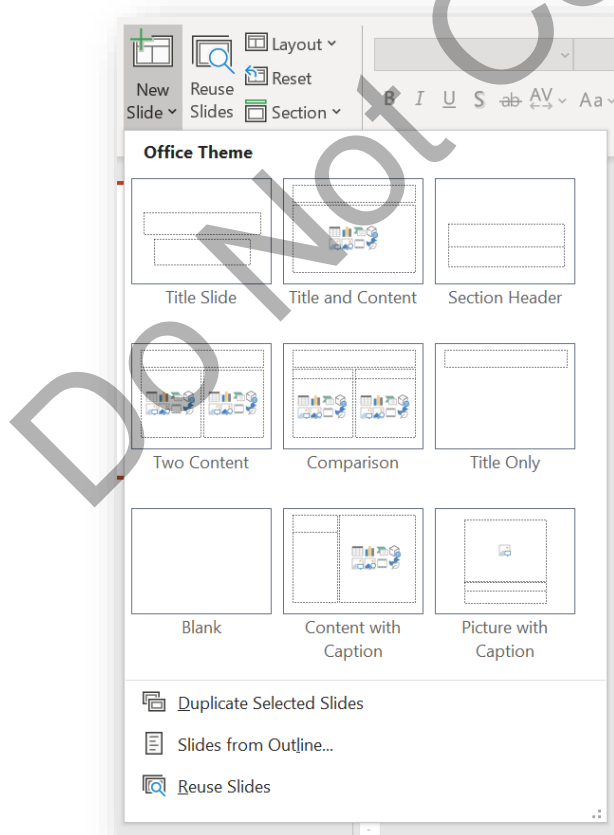
To save it, use the following string:

```
pres.save('file_name.pptx')
```

At this stage, we have nothing in our presentation, so it wouldn't make much sense to save it, but it's good to get used to saving as you go along. Note that if a file with the name you specify already exists, it will be overwritten without warning, so be careful with this line of code. If the file you are trying to overwrite is already open, it will return an error—don't let that confuse you!

### ADDING SLIDES

Let's create our first slide. Before doing so, we have to understand the context of a slide layout. Go to PowerPoint (independent of Python) and create a blank presentation. Go to the Home tab and click on the arrow next to New Slide. You should see a menu that looks something like this:



Each of these options is called a slide layout (these can be edited in the document's slide master, but this is more PowerPoint knowledge than Python knowledge, so we won't discuss it here).

These layouts will be available in Python from the `pres` object. For example, to get the title slide layout, we would run this string:

```
title_layout = pres.slide_layouts[0]
```

0 refers to the fact this is the first layout. Looking at the slide options in the figure on page 2, 1 would be a Title and Content slide, 2 would be a Section Header slide, and so on.

Now that we have the layout, we can add a slide with this layout to the deck:

```
slide = pres.slides.add_slide(title_layout)
```

Try running `len(pres.slides)`. The result should be 1.

Let's add one more slide, a blank layout this time. Looking again at the figure on page 2, the Blank slide layout is numbered as 6 (assuming we start counting from 0), so we can write the following:

```
blank_slide = pres.slides.add_slide(pres.slide_layouts[6])
```

Try running `len(pres.slides)`. The result should be 2.

### ADDING AND EDITING CONTENT ON SLIDES

We're now ready to add content to our slides. You can access anything on the slide using `.shapes`. Type the following two lines:

```
print(len(slide.shapes))
print(len(blank_slide.shapes))
```

The first line should return 2 because there are two objects on the title slide: the title and the subtitle. The second should return 0 because there are no objects on the blank slide.

Note that you could also have accessed each of these slides using the slide number. Try this:

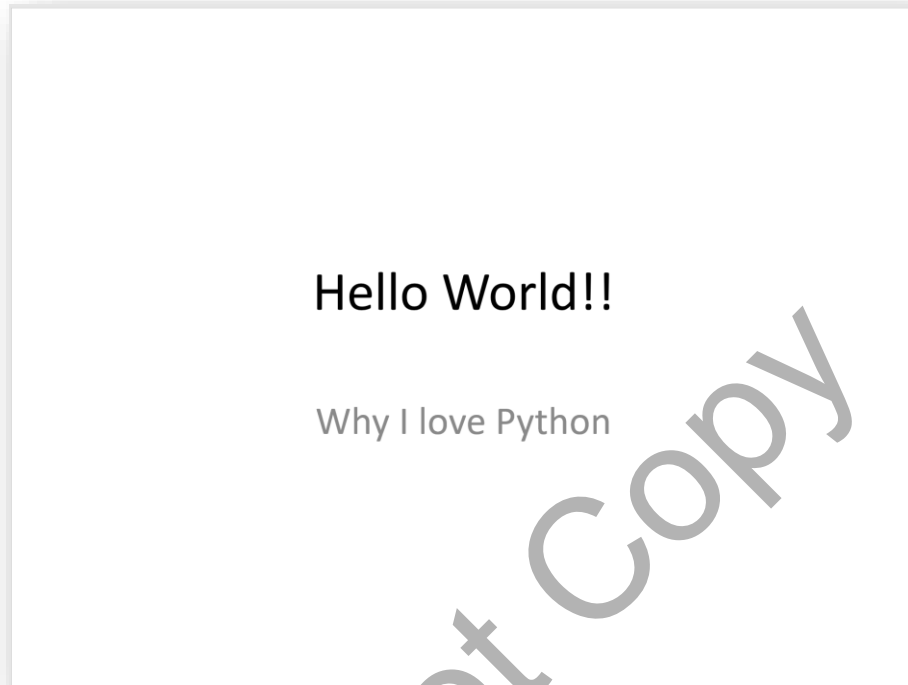
```
print(len(pres.slides[0].shapes))
print(len(pres.slides[1].shapes))
```

The result should be identical.

Next let's edit the content on the title slide:

```
pres.slides[0].shapes[0].text = 'Hello World!!'  
pres.slides[0].shapes[1].text = 'Why I love Python'
```

Save the presentation to see the results. The first slide should look something like this:



Let's add a text box to the blank slide:

```
shape = ( pres.slides[1]  
          .shapes  
          .add_textbox(left = pptx.util.Inches(0.5),  
                       top  = pptx.util.Inches(0.5),  
                       width = pptx.util.Inches(8),  
                       height = pptx.util.Inches(0.5)) )  
  
shape.text = 'This is a table'
```

The syntax is relatively straightforward. Access the second slide in the deck using `slides[1]`, access the `.shapes` element, and use the `.add_textbox` coding. This method returns the shape, and we then access the `.text` element of the shape to edit its content, just as we did for the title slide.

One unusual aspect of the code is that it uses the `pptx.util.Inches()` method to specify the dimensions of the text box. The package does this to give you the flexibility to use other measurement units if desired (see the documentation for details).

Now add a table to the slide:

```
shape = ( pres.slides[1]
          .shapes
          .add_table(rows = 3,
                    cols = 4,
                    left = pptx.util.Inches(0.5),
                    top  = pptx.util.Inches(1.5),
                    width = pptx.util.Inches(8),
                    height = pptx.util.Inches(2)) )

shape.table.cell(2,2).text = 'Sample content'

for i in range(4):
    shape.table.cell(0, i).text = f'Header {i+1}'
```

Let's dive into what's happening here. The first line of code simply adds a table in the same way we added a text box above. The function looks similar. The only difference is that it specifies the number of rows and columns.

Note that confusingly, the table gets added inside a shape. So to access a specific cell in the table, we first need to access the table itself using `shape.table`. We can then access the cell in the third row and third column using `shape.table.cell(2,2)`. The remaining lines of code simply add headers to the table, and sample text in one cell. Keep in mind that the item you put in a cell must be a string. If you want a cell to contain a number, convert it to a string using `str`.

We can now save the presentation. The resulting slide should contain our single line of text and a basic table.

Finally, let's add a slide with a graph. First, import the `pptx` chart data module like this:

```
import pptx.chart.data
```

Then add a new blank slide on which to put our graph:

```
slide = pres.slides.add_slide(pres.slide_layouts[6])
```

Adding the graph itself is simple. First, create a chart data object:

```
chart_data = pptx.chart.data.ChartData()
```

Next, define each category (x-axis labels) and add any series to plot:



```
chart_data.categories = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
chart_data.add_series('Series 1', (1, 2, 5, 8, 3, 5))
```

Finally, plot it:

```
chart =
slide.shapes.add_chart(chart_type=pptx.enum.chart.XL_CHART_TYPE.LINE,
                        x=pptx.util.Inches(2),
                        y=pptx.util.Inches(2),
                        cx=pptx.util.Inches(6),
                        cy=pptx.util.Inches(4.5),
                        chart_data=chart_data).chart
```

Notice how we were able to specify the chart type (in this case, we're creating a line graph).

Enter this line to remove the legend:

```
chart.has_legend=False
```

And here's our result:

