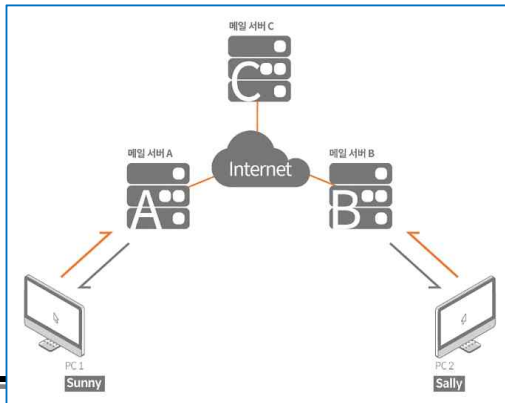


이메일 시스템 구조의 이해

- <https://m.post.naver.com/viewer/postView.nhn?volumeNo=26957131&memberNo=2521903> 참조
- 이메일 송신 프로토콜 : SMTP, 이메일 수신 프로토콜 : POP3, IMAP



파이썬으로 이메일 보내기

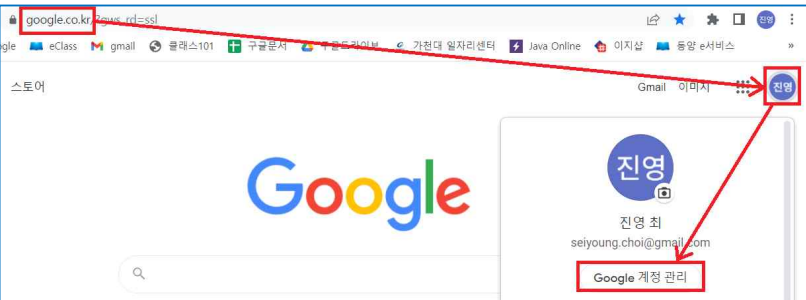
■ Gmail 이용 정보

받는 메일 서버 (POP3) – SSL 필요	주소: pop.gmail.com SSL 사용 : Yes 포트 : 995
보내는 메일 서버 (SMTP) – TLS(암호화) 필요	주소: smtp.gmail.com 인증 사용: Yes STARTTLS 사용: Yes 포트: 465 혹은 587
계정 이름:	Gmail 가입된 계정의 이메일 주소 전체 (@gmail.com가 포함된 문자열)
이메일 주소:	받는 사람 이메일 주소 (username@gmail.com)
비밀번호:	Gmail 계정 비밀번호

- TLS (Transport Layer Security) : TCP/IP 같은 통신에서 사용하는 암호 규약
- STARTTLS : 텍스트에 대한 암호화를 업그레이드하고 확장한 버전

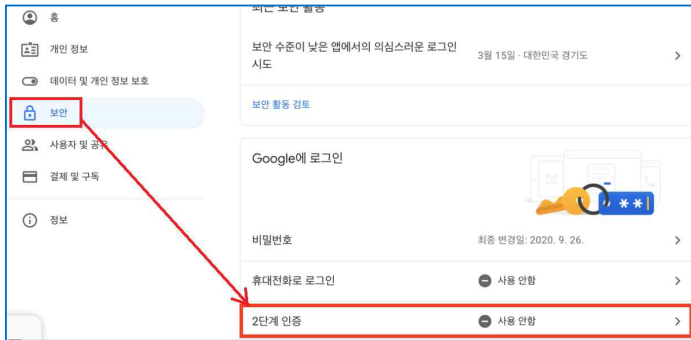
gmail : 앱 비밀번호 생성하기

- 구글 페이지(<https://www.google.co.kr/>) 열고
- “Google 계정 관리” 들어가기
- 창을 가로로 넓게 열기 (왼편 사이드바 보이게)



gmail : 앱 비밀번호 생성하기

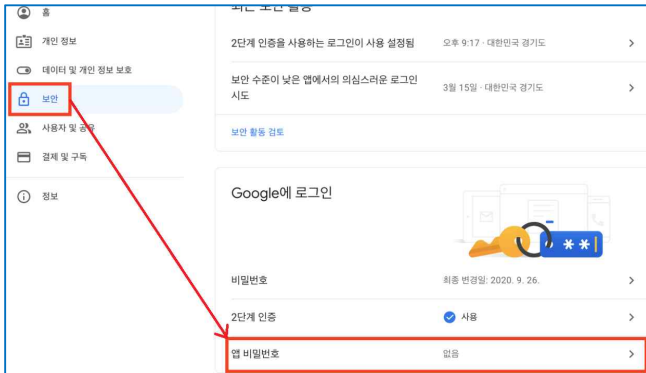
- “보안” -> “2단계인증” -> “시작하기”



- 비번 입력 -> 휴대폰 인증받기
- “2단계 인증을 사용하도록 설정하시겠습니까?” -> “사용”

gmail : 앱 비밀번호 생성하기

- 화면 왼편 위의 **Google 계정** 눌러서 “Google 계정” 첫 화면으로 돌아가기.
- “보안” -> “앱 비밀번호” -> 비번 입력



gmail : 앱 비밀번호 생성하기

- “메일”, “Windows 컴퓨터” 선택 -> “생성”

← 앱 비밀번호

앱 비밀번호를 사용하면 2단계 인증을 지원하지 않는 기기의 앱에서 Google 계정에 로그인할 수 있습니다. 비밀번호를 한 번만 입력하면 기억할 필요가 없습니다. [자세히 알아보기](#)

앱 비밀번호가 없습니다.

앱 비밀번호를 생성할 앱 및 기기를 선택하세요.

메일 ▼

Windows 컴퓨터 ▼

생성


gmail : 앱 비밀번호 생성하기

- 프로그램에서 사용할 수 있는 16자리 비번 취득.

생성된 앱 비밀번호

Windows 컴퓨터용 앱 비밀번호

파이썬 프로그램에서 사용할
16자리 비밀번호.



사용 방법

1. '메일' 앱을 엽니다.
2. '설정' 메뉴를 엽니다.
3. '계정'을 선택한 뒤 내 Google 계정을 선택합니다.
4. 비밀번호를 위에 표시된 16자리 비밀번호로 교체합니다.

Add your Google account

Enter the information below to connect to your Google account.

Email address

securesally@gmail.com

Password

2-7

<https://docs.python.org/ko/3/library/smtplib.html> 참조

```
from email.mime.text import MIMEText # MIMExe 생성에 사용
```

```
def sendMail(fromAddr, toAddr, msg):  
    import smtplib # 파이썬의 SMTP 모듈  
    # 메일 서버와 connect하고 통신 시작  
    s = smtplib.SMTP("smtp.gmail.com", 587) # SMTP 서버와 연결  
    s.starttls() # SMTP 연결을 TLS (Transport Layer Security) 모드로 전환  
  
    # 앱 password 이용  
    s.login('메일 계정', '앱 비밀번호')  
    s.sendmail(fromAddr, [toAddr], msg.as_string())  
    s.close()
```

재사용을 위해
함수로 준비!

```
msg = MIMEText('본문: 한 번 보내봅니다')  
msg['Subject'] = '제목: 파이썬으로 gmail 보내기'  
sendMail('보내는 메일', '받는 메일', msg)
```



```
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
```

```
def sendMail(fromAddr, toAddr, msg):
    .....
```

```
senderAddr = "보내는 메일 주소"
```

```
recipientAddr = "받는 메일 주소"
```

```
# HTML 전달을 위해 컨테이너 역할을 할 수 있는 "multipart/alternative" 타입 사용
```

```
msg = MIMEMultipart('alternative')
```

```
msg['Subject'] = "Test email in Python 3.10"
```

```
msg['From'] = senderAddr
```

```
msg['To'] = recipientAddr
```

```
# 파일로부터 읽어서 MIME 문서를 생성.
```

```
htmlFD = open("logo.html", 'rb')
```

```
HtmlPart = MIMEText(htmlFD.read(), 'html', _charset = 'UTF-8')
```

```
htmlFD.close()
```

```
# 만든 mime을 MIMEBase에 첨부.
```

```
msg.attach(HtmlPart)
```

```
# 메일 발송.
```

```
sendMail(senderAddr, recipientAddr, msg)
```

도서관리 프로그램에서 이메일 보내기

- 사용자가 책을 선택하면 책의 내용을 html로 바꿔 Gmail 서버를 통해 원하는 사람에게 메일 보내는 기능을 구현해 보자.

=====Menu=====

Load xml: |

...

Get book data from isbn: g

send mail : i

=====Menu=====

select menu : i

Title : Book List Email

sender email address : 보내는 이메일

recipient email address : 받을 이메일

write message : TU test email

input your app password : 앱 비밀번호

Do you want to include book data (y/n): y

input keyword to search: The

connect smtp server ...

Mail sending complete!!!

실행 모습!

```
def printMenu():
```

```
    print("\n\nWelcome! Book Manager Program (xml version)")
    print("====Menu====")
    print("Load xml: l")
    print("Print dom to xml: p")
    print("Quit program: q")
    print("print Book list: b")
    print("Add new book: a")
    print("sEarch Book Title: e")
    print("Make html: m")
    print("-----")
    print("Get book data from isbn: g")
    print("send mail : i")
    print("sTart Web Service: t")
    print("====Menu====")
```

```
def launcherFunction(menu):
```

```
    .....
```

```
    elif menu == 'i':
        sendBookMail()
```

```
def sendMail(fromAddr, toAddr, msg):
```

```
.....
```

```
def sendBookMail():
```

```
    html = ""
```

```
    # 사용자 입력 받기
```

```
    title = input('Title :')
```

```
    senderAddr = input('sender email address :')
```

```
    recipientAddr = input('recipient email address :')
```

```
    msgtext = input('write message :')
```

```
    passwd = input('input your app password :')
```

```
    msgtext = input('Do you want to include book data (y/n):')
```

```
    if msgtext == 'y':
```

```
        keyword = input('input keyword to search:')
```

```
        html = MakeHtmlDoc(SearchBookTitle(keyword))
```

```
    ...
```

```
.....  
from email.mime.multipart import MIMEMultipart #MIMEMultipart MIME 생성  
from email.mime.text import MIMEText  
  
# HTML 전달을 위해 컨테이너 역할을 할 수 있는 "multipart/alternative" 타입 사용  
msg = MIMEMultipart('alternative')  
  
msg['Subject'] = title      #set message  
msg['From'] = senderAddr  
msg['To'] = recipientAddr  
  
msgPart = MIMEText(msgtext, 'plain')  
bookPart = MIMEText(html, 'html', _charset = 'UTF-8')  
  
# 메시지에 생성한 MIME 문서를 첨부  
msg.attach(msgPart)  
msg.attach(bookPart)  
  
print ("connect smtp server ... ")  
sendMail(senderAddr, recipientAddr, msg) # send mail  
print ("Mail sending complete!!!")
```

이메일 주소 popup

(9주) 서울시근린시설앱.py

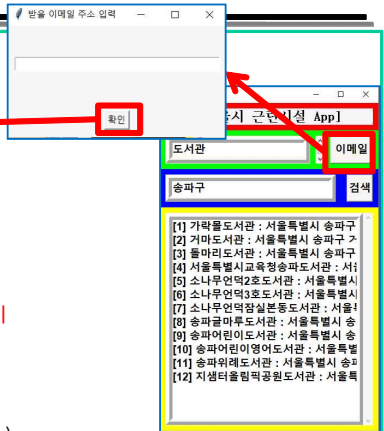
```
popup = inputEmail = btnEmail = None  
addrEmail = None
```

```
def onEmailInput():  
    global addrEmail  
    addrEmail = inputEmail.get()  
    popup.destroy() # popup 내리기
```

```
def onEmailPopup():  
    global g_Tk, addrEmail, popup  
    addrEmail = None  
    popup = Toplevel(g_Tk) # popup 띄우기  
    popup.geometry("300x150")  
    popup.title("받을 이메일 주소 입력")
```

```
global inputEmail, btnEmail  
inputEmail = Entry(popup, width = 200,  
inputEmail.pack(fill='x', padx=10, expand=True)
```

```
btnEmail = Button(popup, text="확인", command=onEmailInput)  
btnEmail.pack(anchor="s", padx=10, pady=10)
```



Chapter 10

문자열 이야기

- str 클래스
- re 모듈

str 클래스

- 문자열을 다루는 기본 클래스 (모듈 import 필요 없음)
- str 클래스와 bytes 클래스
 - str 클래스: unicode 값의 모임.
 - bytes 클래스: byte 값의 모임.
 - bytes 클래스: immutable
 - bytearray 클래스: mutable
- 인코딩과 디코딩
 - str -> bytes로 변환 : encode()
 - bytes -> str로 변환 : decode()

```
>>> s1 = '한글'
>>> b1 = s1.encode('cp949')
>>> s2 = b1.decode('cp949')
```


str 클래스 주요 메소드

- **capitalize()** 첫 문자를 대문자로, 나머지 문자를 소문자로 변경
- **count(keyword, [start, [end]])** keyword가 포함된 횟수를 반환
- **encode([encoding, [errors]])** 해당 인코딩으로 변경 (기본적으로 모두 유니코드)

```
>>> s1 = "python is poserful. IT IS GREAT!"
>>> s2 = s1.capitalize()
>>> s1
'python is poserful. IT IS GREAT!'
>>> s2
'Python is poserful. it is great!'

>>> s1.count("p")
2
>>> "가나다".encode('cp949') #윈도우에서 사용하는 'CP949'
b'\xb0\xa1\xb3\xaa\xb4\xd9'
```

str 클래스 주요 메소드

- **endswith**(postfix, [start, [end]]) postfix로 문자열이 끝나면 True를 반환
- **expandtabs**([tabsize]) 탭을 공백으로 치환
- **find**(keyword, [start, [end]]) 문자열 keyword가 나타나는 첫 번째 인덱스를 반환

```
>>> "python is powerful".endswith('ful')
True
>>> "python\tis\tpowerful".expandtabs()
'python is    powerful'
>>> "python is powerful".find('p')
0
>>> "python is powerful".find('p', 5, -1)
10
```

str 클래스 주요 메소드

- **index(keyword, [start, [end]])** find() 메소드와 동일하게 동작, keyword를 찾지 못하는 경우 ValueError 예외 발생
- **isalnum()** 알파벳과 숫자로 이루어져 있으면 True를 반환
유사 함수들: isalpha(), islower(), isspace(), isupper(), isdecimal(), isdigit(), isnumeric()
- **strip([chars])** 문자열의 양쪽 끝을 잘라 냄, chars가 지정되지 않으면 공백 문자를 제거, 지정되어 있을 경우에는 chars의 모든 조합을 제거

```
>>> "python is powerful".index('p', 5, -1)
10
>>> "python is powerful".index('pa', 5, -1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>> "python is powerful".isalpha()
False
>>> "\t python is great \t".strip()
'python is great.'
```

str 클래스 주요 메소드

- **join(sequence)** 순회가능한 시퀀스 형 변수를 지정된 문자열로 연결해서 반환, split과 정반대 기능
- **partition(separator)** 문자열을 separator로 나눔. 결과로 앞부분, separator, 뒷부분 세 튜플이 반환됨
- **split(separator,[maxsplit]), rsplit** : 문자열을 separator로 분리. separator가 생략되면 공백 문자를 구분자로 사용. maxsplit 이 있으면 그 값 만큼만 분리. rsplit()은 오른쪽부터 maxsplit번 분리.

```
>>> ".".join("HOT")
```

```
'H.O.T'
```

```
>>> s1 = "12 555 67"
```

```
>>> l1 = s1.split()
```

```
>>> l1
```

```
['12', '555', '67']
```

```
>>> s2 = '&'.join(l1)
```

```
>>> s2
```

```
'12&555&67'
```

```
>>> s2 = "".join(l1)
```

```
>>> s2
```

```
'1255567'
```

```
>>> "python is powerful".partition("is")
```

```
('python ', 'is', ' powerful')
```

```
>>> "python is powerful".split(' ', 1)
```

```
['python', 'is powerful']
```

```
>>> "python is powerful".split()
```

```
['python', 'is', 'powerful']
```

문자열 뒤집기

- `reversed()`은 반대방향으로 순회하는 iterator를 반환.

```
>>> l1=[1,2,3]
>>> for i in reversed(l1):
...     print(i)
...
3
2
1
```

- `join` 을 이용하여 `reversed`가 반환하는 iterator를 문자열로 만들어 뒤집을 수 있다.

```
>>> s1 = 'Hello!'
>>> reversed_s1 = "".join(reversed(s1))
>>> print(reversed_s1)
!olleH
```

re 정규표현식 모듈

- re 모듈이란?
 - 특정한 규칙을 가진 문자열을 표현하는데 사용되는 형식 언어
 - 주어진 패턴으로 문자열을 검색/치환하는데 사용
- 정규표현식 (Regular Expression) 메타문자 (특별한 의미를 가진 문자)

특수 문자	의미
^	문자열의 시작을 의미
\$	문자열의 종료를 의미
	OR 연산 'A B'와 같은 경우 'A' 혹은 'B'를 의미
*	문자가 0회 이상 N회 반복됨을 의미
+	문자가 1회 이상 N회 반복됨을 의미
?	문자가 0회 또는 1회 출현
.	모든 문자
[]	문자 또는 문자 범위(a-z)내 문자 출현

re 정규표현식 예제

- 정규식 `^app` 는 'apple and orange'는 매치되지만 'orange and apple'는 매치되지 않음
- 정규식 `ple$`는 'orange and apple'는 매치되지만, 'apple and orange'는 매치되지 않음
- 정규식 `ap*le`는 'ale'(p가 없어도 된다), 'apple', 'apppple'와 같이 a와 le 사이에 p가 0회 이상 반복되는 모든 경우와 매치됨

re 모듈의 주요 함수들

- 주요 모듈 함수 (pattern은 정규식, string은 대상 문자열)
 - re.search(pattern, string[, flags])** string 전체에 대해서 pattern이 존재하는지 검사
 - re.match(pattern, string[, flags])** string 시작부분부터 pattern이 존재하는지 검사. match 객체 또는 None을 돌려줌. 중간에서 찾을 때: `bool(re.match('.*th', ' 35th'))` 식으로 사용 가능. (search보다 빠름)
 - re.split(pattern, string[, maxsplit=0])** pattern을 구분자로 string을 분리하여 리스트로 반환

```
>>> import re
```

```
>>> bool(re.match('[0-9]*th', ' 35th')) #문자열 시작부터 검색, Boolean로 변환
False
```

```
>>> bool(re.search('[0-9]*th', ' 35th')) #문자열 전체 검색, Boolean로 변환
True
```

```
>>> re.split('[:. ]+', 'apple Orange:banana tomato') #구분자로 ':',' ',' ' 사용
['apple', 'Orange', 'banana', 'tomato']
```

```
>>> re.split('[:. ]+', 'apple Orange:banana tomato', 2) #maxsplit 2회
['apple', 'Orange', 'banana tomato']
```


정규 표현식 객체

- `compiler()` : 정규식으로 표현된 문장을 매번 다시 분석하지 않고, 컴파일하여 정규표현식 객체를 생성하여 재사용
→ 동일 패턴을 반복적으로 검색하는 경우 성능 향상
- `r"..."` (raw 표기법): `\` (backslash)는 정규표현식의 메타 문자이므로 이를 일반 문자열에 표현할 때는 `'\\'`, 정규식에 표현할 때는 `'\\\\'` 사용.
→ raw 표기법에서는 `\` (backslash)를 이스케이프 표현으로 해석하지 않고 일반 문자로 해석하므로 일반 문자열에 표현할 때는 `'\'`, 정규식에 표현할 때는 `'\\'` 로써 사용 가능.

```
>>> c = re.compile(r"app\w*") #정규식을 분석해서 객체로 만듦. raw 표기법, 'w'는 아스키문자
```

- `findall()` : 단어를 각각 정규식과 매치하여 패턴에 맞는 케이스를 전부 찾아서 리스트를 반환.

```
>>> c.findall("application orange apple banana") #분석없이 검색  
['application', 'apple']  
>>> c.findall("There are so many apples in the basket")  
['apples']
```