



# Python Programming for Blackboard Learn System Administrators

**Michael Bechtel Jean-Max Davis Brett Stephens**

## LMS Specialist

# Academic Educational Technologist

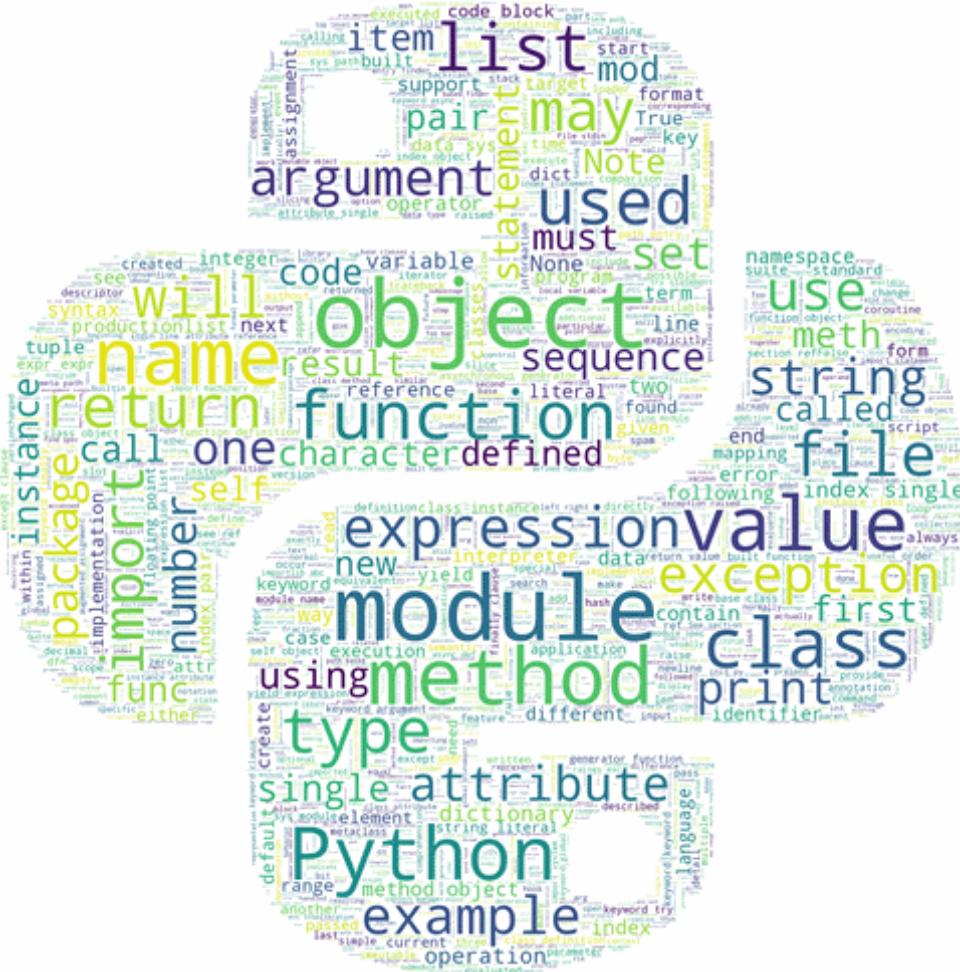
## Systems Analyst

Indian River State  
College

University of Miami

University of Miami

July 23, 2019



DEVCON  
austin 19

Fortran & COBOL

~~Python~~ Programming  
for Blackboard Learn  
System Administrators

Michael Bechtel Jean-Max Davis Brett Stephens

LMS Specialist

Academic Educational  
Technologist

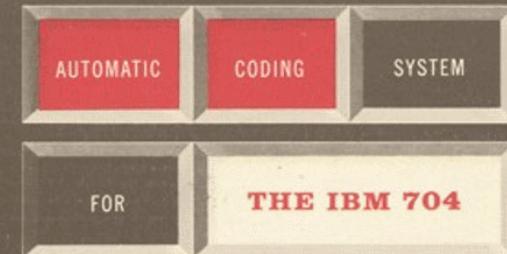
Systems Analyst

Indian River State  
College

University of Miami

University of Miami

Fortran  
& COBOL



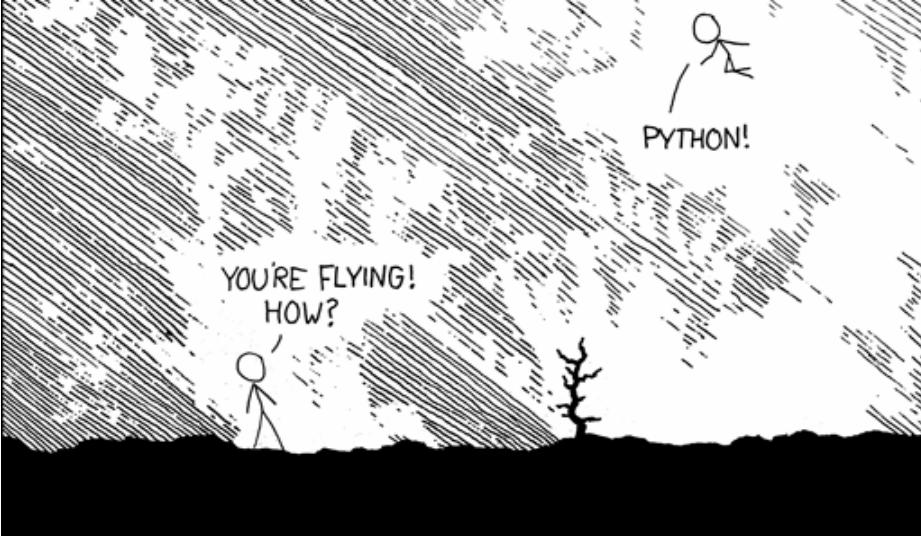
July 23, 2019

Blackboard

#BbWorld19

Slides and Sample Code available at...

<https://tinyurl.com/bb19python>



## What? Why Python?

- Dynamically-typed
- Interpreted (no compiling)
- OOP Capable
- Well supported by community
- Python is designed to be readable

```
for n in range(10):  
    print(n)
```

0  
1...  
9

<https://xkcd.com/353/>

*Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Readability counts.*

--The Zen of Python (PEP 20)

Easter egg >>> import this

## Learning Python (it takes a bit longer than 45 minutes)

Python.org  
Documentation  
& Beginner's Guide



Dr. Chuck's PY4E.com

Dr. Charles Severance  
University of Michigan

He was here yesterday,  
so go ask him to teach  
you then...



# Useful Libraries and stuff

## I/O & Data

- Requests
- bbRest
- SQLAlchemy
- Selenium

## I/O & Data

- Google apiclient
- json
- email, smtplib
- BeautifulSoup

## Misc

- DocOpt & Schema
- Matplotlib
- Jupyter Notebook/Lab

# SQL in Python...with Alchemy!

There is a hard way and an easy way. Ain't nobody got time for the hard way.

## SQL Alchemy basic example

```
1 from sqlalchemy import create_engine
2 db_string = "postgres://postgres:postgres@somehost:9879/BBdbName"
3 db = create_engine(db_string)
4 userid = 'stu_bs007'
5 results = db.execute("select u.user_id, cm.course_id, cu.role \
6                         from users u join course_users cu on \
7                             u.pk1 = cu.users_pk1 \
8                         join course_main cm on \
9                             cu.crsmain_pk1 = cm.pk1 \
10                        where u.user_id = '%s' " % userid)
```

SeeeeeQuiiil

```
1 for row in results:  
2     print(  
3         f'User: {row.user_id} Course: {row.course_id} \  
4         Role: {row.role}')  
5  
6 #output User: stu_bs007 Course: dev-course Role: S
```

# List and Dictionary Comprehensions

You thought you could read before...

# What is a List? A Dictionary?

- List is a list...
- Dictionaries contain definitions... or key/value pairs.
- Lists can contain anything as items
- Dictionaries can too, but as values. Keys can be strings, numbers, (or tuples)
- Referencing item in a list: myList[3]
- Referencing value in a dict: myDict['keyname']

# List comprehension

```
1 # using a for loop
2
3 numbsq = []
4
5 for num in range(8):
6     numbsq.append(num**2)
7
8 print(numbsq)
9
10 # with list comprehension
11
12 numbsqlc = [num**2 for num in range(8)]
13
14 print(numbsqlc)
```

# Dictionary example

```
1 names = ['Hal', 'Clark', 'Charles']
2 supernames = ['Green Lantern', 'Superman', 'Professor X']
3
4 hero_dict = {}
5 for name, super in zip(names, supernames):
6     hero_dict[name] = super
7 print(hero_dict)
8
9 # {'Hal': 'Green Lantern', 'Clark': 'Superman', 'Charles': 'Professor X'}
10
11 #Dictionary Comprehension
12
13 hero_dict = dict(zip(names, supernames))
14
15 hero_dict2 = {value:key for key,value in hero_dict.items()}
16 print (hero_dict2)
17
18 # {'Green Lantern': 'Hal', 'Superman': 'Clark', 'Professor X': 'Charles'}
```



Don't forget  
to rate this session  
in the BbWorld app.



# REST

Everyone take a nap. Mike is going to talk now.

# REST with the Requests library

```
1 import requests
2 import json
3
4 key = 'KeyFromBbDevAppRegistration'
5 secret = 'SecretSquirrel'
6 target_domain = 'mybb.institution.edu'
7
8 api_base_path = '/learn/api/public/'
9 api_default_version = 'v1'
10
11 rest_api = {
12     'token': f'{api_base_path}{api_default_version}/oauth2/token',
13     'users': f'{api_base_path}{api_default_version}/users/:userId',
14     'courses': f'{api_base_path}{api_default_version}/courses/:courseId',
15     'memberships': f'{api_base_path}{api_default_version}/courses/:courseId/users',
16     'enrollment': f'{api_base_path}{api_default_version}/courses/:courseId/users/:userId',
17     'enrollments': f'{api_base_path}{api_default_version}/users/:userId/courses',
18 }
```

# REST with the Requests library

```
1 payload = {"grant_type": "client_credentials", "token": None}
2 # lets the auth and headers here
3 auth = ''
4 headers = ''
5
6
7 def set_token():
8     global auth
9     global headers
10    oauth_url = f"https://{{target_domain}}{{rest_api['token']}}"
11    print(f"[POST] OAuth2 Set Token URL: {oauth_url}")
12    oauth_res = requests.post(oauth_url, data=payload, auth=(key, secret))
13    print(f"==> Token Response ==\n{{oauth_res.text}}\n")
14    payload['token'] = oauth_res.json()['access_token']
15    auth = f"Bearer {{payload['token']}}"
16    headers = {'Authorization': auth, 'Content-Type': 'application/json'}
```

# REST with the Requests library

```
1 def post_bb_data(version=None, **kwargs):
2     bb_object = kwargs['api'].title()
3     url = f"https://{{target_domain}}{{rest_api[{{kwargs['api']}}]}}"
4
5     try:
6         url = url.replace(kwargs['remove'], kwargs['replace'])
7     except KeyError:
8         pass
9
10    data = kwargs['data']
11    print(f"[POST] {bb_object} URL: {url}")
12    rest_res = requests.post(url, json=data, headers=headers)
13    print(f"==== {bb_object} Response ===\n{{rest_res.text}}\n")
14    return rest_res.json()
```

# REST with the Requests library

```
1 def get_bb_data(version=None, **kwargs):
2     bb_object = kwargs['api'].title()
3     url = f"https://{{target_domain}}{{rest_api[{{kwargs['api']}]]}}"
4
5     if version:
6         url = url.replace(api_default_version, version)
7
8     if kwargs['api'] == 'users' or kwargs['api'] == 'enrollments':
9         url = url.replace(':userId', kwargs['userId'])
10    elif kwargs['api'] == 'courses' or kwargs['api'] == 'memberships':
11        url = url.replace(':courseId', kwargs['courseId'])
12    elif kwargs['api'] == 'enrollment':
13        url = url.replace(':courseId', kwargs['courseId'])
14        url = url.replace(':userId', kwargs['userId'])
15
16    print(f"[GET] {bb_object} URL: {url}")
17
18    res = requests.get(url, headers=headers)
19
20    print(f"== {bb_object} Response ==\n{res.text}\n")
21    return res.json()
```

# REST with the Requests library

```
1 def main():
2     set_token()
3     print(auth)
4     print(headers)
5     userId = 'userName:mbechtel'
6     courseId = 'externalId:LOR-CC-mbechtel'
7     user = get_bb_data(api='users', userId=userId)
8     print(f"user ID: {user['userName']} email: {user['contact']['email']}\n")
9     course = get_bb_data(api='courses', courseId=courseId)
10    print(f"course ID: {course['id']} external ID: {course['externalId']}\n")
11    memberships = get_bb_data(api='memberships', courseId=courseId)
12    print('memberships:', memberships, '\n')
13    enrollments = get_bb_data(api='enrollments', userId=userId)
14    print('user enrollments:', enrollments, '\n')
15    enrollment = get_bb_data(
16        api='enrollment', courseId=courseId, userId=userId)
17    print('course user enrollment:', enrollment, '\n')
```

# REST with the Requests library

```
1     new_user_data = {
2         'externalId': 'devcon19_user',
3         'userName': 'devcon19user',
4         'name': {
5             'given': 'DevCon19',
6             'family': 'User'
7         },
8         'password': '1234',
9         'contact': {
10             'email': 'no-reply@no.email.here'
11         }
12     }
13
14     new_user = post_bb_data(api='users', data=new_user_data,
15                             remove='/:userId', replace='')
16     print('Created new user:', new_user)
17
18
19 if __name__ == '__main__':
20     main()
```

# bbRest

You've seen the hard way, now check out the eas...less hard way!

# REST, the less difficult way with bbRest

```
1 from bbrest import BbRest
2
3 bbURL = 'https://bb.yourinst.edu'
4 restKey = 'RESTapiKeyFromDeveloper.blackboard.com'
5 restSecret = 'SetecAstronomy'
6
7
8 def main():
9     bb = BbRest(restKey, restSecret, bbURL)
10    userName = 'bstephens'
11    courseId = 'dev-bs'
12    user = bb.GetUser(userId=userName).json()
13    print(f"user ID: {user['userName']} \
14          email: {user['contact']['email']}")
15    course = bb.GetCourse(courseId=courseId).json()
16    print(f"course ID: {course['id']} \
17          external ID: {course['externalId']}")
18    memberships = bb.GetCourseMemberships(courseId=courseId).json()
19    print('memberships:', memberships)
20    enrollments = bb.GetUserMemberships(userId=userName).json()
21    print('user enrollments:', enrollments)
22    enrollment = bb.GetMembership(courseId=courseId,
23                                    userId=userName).json()
24    print('course user enrollment:', enrollment)
```

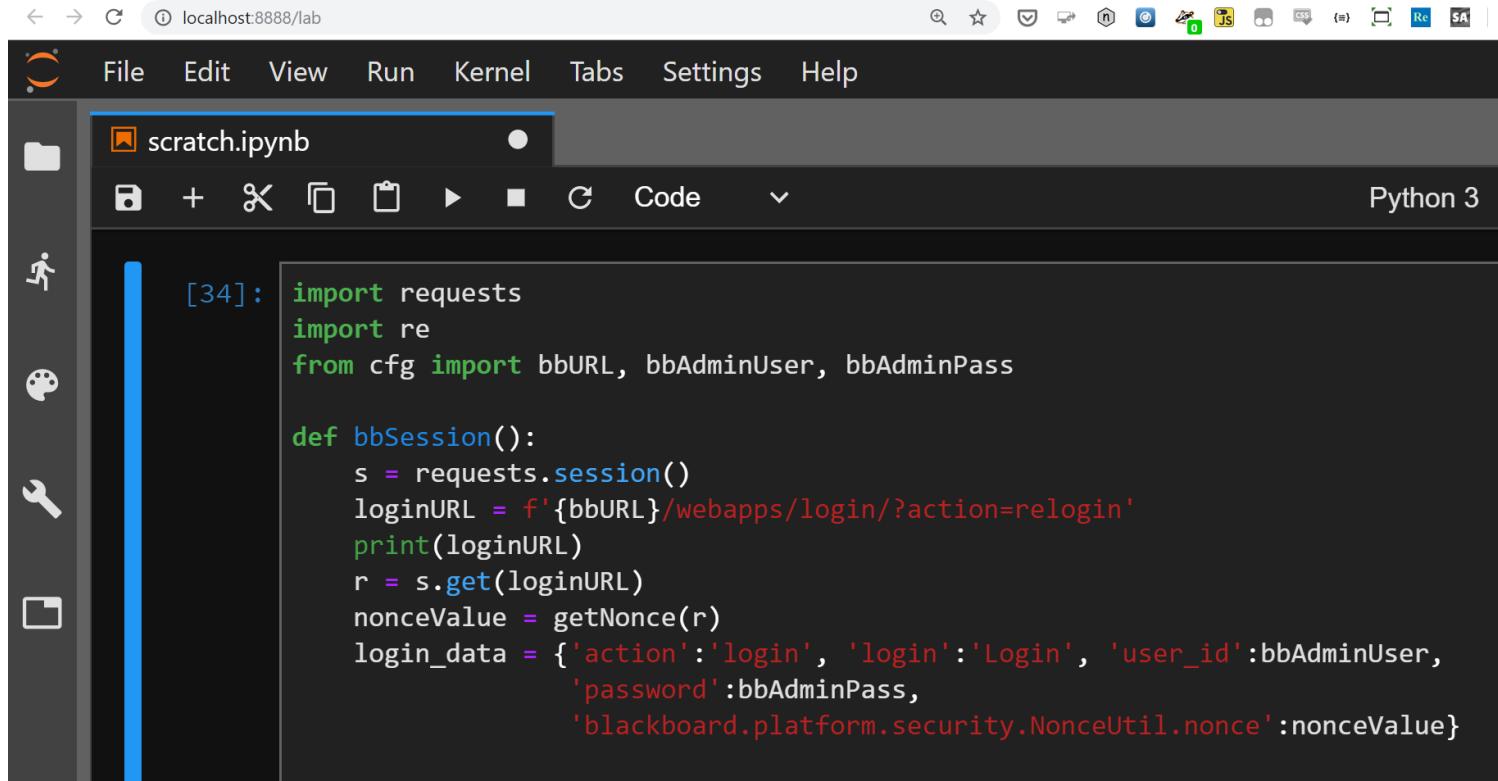
# REST, the less difficult way with bbRest

```
1 new_user_data = {  
2     'externalId': 'devcon19_user',  
3     'userName': 'devcon19user',  
4     'name': {  
5         'given': 'DevCon19',  
6         'family': 'User'  
7     },  
8     'password': 'p455w0rd',  
9     'contact': {  
10        'email': 'email@example.edu'  
11    }  
12 }  
13  
14 new_user = bb.CreateUser(payload=new_user_data)  
15 print('Created new user:', new_user.json())  
16  
17  
18 if __name__ == '__main__':  
19     main()
```

# Jupyter Lab & Google Colaboratory

Working on the command line just isn't as fun as it used to be...

# Jupyter Lab



The screenshot shows the Jupyter Lab interface running at localhost:8888/lab. The top navigation bar includes File, Edit, View, Run, Kernel, Tabs, Settings, and Help. The toolbar features icons for search, star, download, copy, paste, and various file operations. A status bar at the bottom shows 0, JS, CSS, {=}, Re, and SA.

The main area displays a file named scratch.ipynb. The code cell [34]: contains the following Python 3 code:

```
import requests
import re
from cfg import bbURL, bbAdminUser, bbAdminPass

def bbSession():
    s = requests.session()
    loginURL = f'{bbURL}/webapps/login/?action=relogin'
    print(loginURL)
    r = s.get(loginURL)
    nonceValue = getNonce(r)
    login_data = {'action':'login', 'login':'Login', 'user_id':bbAdminUser,
                  'password':bbAdminPass,
                  'blackboard.platform.security.NonceUtil.nonce':nonceValue}
```

# Google Colaboratory

The screenshot shows a Google Colaboratory notebook titled "Copy of 0. DevCon.ipynb". The notebook interface includes a toolbar with various icons for file operations, a sidebar with "COMMENT" and "SHARE" buttons, and a code editor with tabs for "CODE" and "TEXT". The main content area displays two sections:

- BbRest : Blackboard for Humans**
  - Presenter : Matt Deakyne**
  - [m.d@ku.edu](mailto:m.d@ku.edu)**
- Setting up the environment variables.**

You don't want to ever publish your key or secret - which is why these are stored in a .env file that is not shared in your git repo / or visible when presenting. Python has two functions to get these values in a secure way:

`load_dotenv` - adds the contents of .env file to your environment. `getenv` - reads the environment variables into local python variables.

*NOTE: You can skip this step, and hardcode in the values - but you should avoid committing these anywhere, or displaying them for others.*

```
[ ] from dotenv import load_dotenv
from os import getenv

if load_dotenv():
    kev. secret. url = getenv('akev'). getenv('asecret'). getenv('aurl')
```

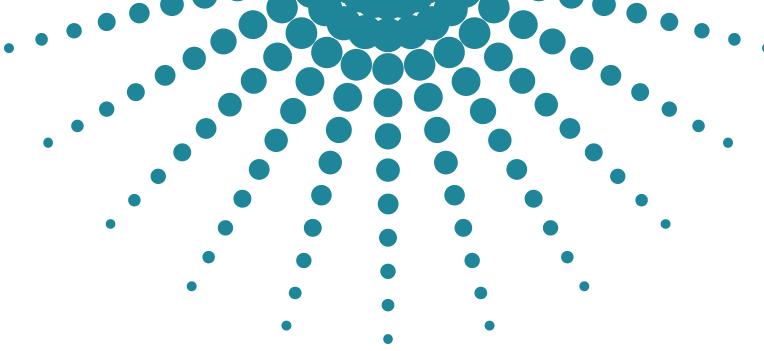
# QUESTIONS?

We can keep talking until they kick us out.



Don't forget  
to rate this session  
in the BbWorld app.





# DEVCON austin

