# SI 330: Data Manipulation

JSON, APIs and AWS

# Course Roadmap

| Week | Topics |
|---|---|
| 1 | Course introduction & review of python basics |
| 2 | Basic and compound data structures |
| 3 | Extracting patterns from text with regular expressions |
| 4 | Natural Language Processing |
| 5 | JSON, APIs, AWS |
| 6 | Pandas I |
| 7 | Pandas II |
| 8 | Pandas III |
| 9 | Break |
| 10 | SQL |

## Overview of Today

- Announcements
- Questions, comments, concerns
- JavaScript Object Notation (JSON)
- Application Programming Interfaces (APIs)
- Amazon Web Services (AWS)
    - Simple Storage Service (S3)
    - Lambda (not python lambda)

# Overview of Today

- **Announcements**
- Questions, comments, concerns
- JavaScript Object Notation (JSON)
- Application Programming Interfaces (APIs)
- Amazon Web Services (AWS)
    - Simple Storage Service (S3)
    - Lambda (not python lambda)

# Please sign up for an AWS account TODAY!

- 2 options:
  - Standard AWS account, using only free tier (requires credit/debit card)
    - Advantages: get account right away, no restrictions, persists after graduation
    - Disadvantage: need credit/debit card (but extremely unlikely you'll get charged
  - AWS Educate
    - Advantage: no need for credit/debit card
    - Disadvantages: can take several days to activate, some restrictions on account, disappears after you graduate

# Overview of Today

- **Announcements**
- Questions, comments, concerns
- JavaScript Object Notation (JSON)
- Application Programming Interfaces (APIs)
- Amazon Web Services (AWS)
  - Simple Storage Service (S3)
  - Lambda (not python lambda)

## Overview of Today

- Announcements
- **Questions, comments, concerns**
- JavaScript Object Notation (JSON)
- Application Programming Interfaces (APIs)
- Amazon Web Services (AWS)
  - Simple Storage Service (S3)
  - Lambda (not python lambda)

# Overview of Today

- Announcements
- Questions, comments, concerns
- **JavaScript Object Notation (JSON)**
- Application Programming Interfaces (APIs)
- Amazon Web Services (AWS)
  - Simple Storage Service (S3)
  - Lambda (not python lambda)

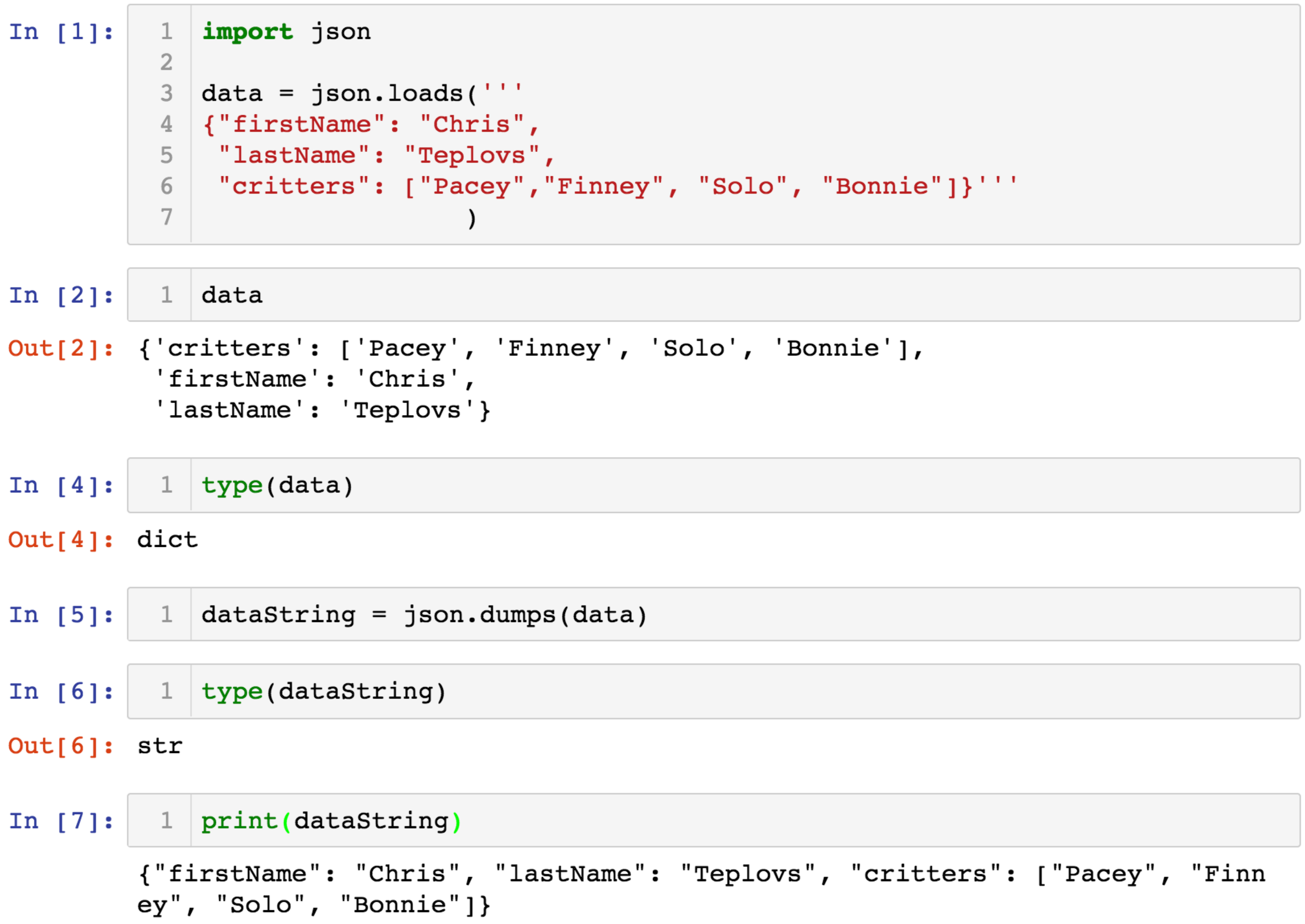
## JSON: JavaScript Object Notation

- 2 basic structures:
  - (unordered) name: value pairs (a.k.a. python dictionaries)
  - (ordered) lists (a.k.a. python lists)

# JSON Example

```
{
        "id": "0001",
        "type": "donut",
        "name": "Cake",
        "ppu": 0.55,
        "batters":
                {
                        "batter":
                                [
                                        { "id": "1001", "type": "Regular" },
                                        { "id": "1002", "type": "Chocolate" },
                                        { "id": "1003", "type": "Blueberry" },
                                        { "id": "1004", "type": "Devil's Food" }
                                ]
                },
        "topping":
                [
                        { "id": "5001", "type": "None" },
                        { "id": "5002", "type": "Glazed" },
                        { "id": "5005", "type": "Sugar" },
                        { "id": "5007", "type": "Powdered Sugar" },
                        { "id": "5006", "type": "Chocolate with Sprinkles" },
                        { "id": "5003", "type": "Chocolate" },
                        { "id": "5004", "type": "Maple" }
                ]
}
```

# Working with JSON in Python



```
In [1]: import json

        data = json.loads('''
        {"firstName": "Chris",
         "lastName": "Teplovs",
         "critters": ["Pacey","Finney", "Solo", "Bonnie"]}'''
                            )
```

```
In [2]: data
```

```
Out[2]: {'critters': ['Pacey', 'Finney', 'Solo', 'Bonnie'],
         'firstName': 'Chris',
         'lastName': 'Teplovs'}
```

```
In [4]: type(data)
```

```
Out[4]: dict
```

```
In [5]: dataString = json.dumps(data)
```

```
In [6]: type(dataString)
```

```
Out[6]: str
```

```
In [7]: print(dataString)
```

```
{"firstName": "Chris", "lastName": "Teplovs", "critters": ["Pacey", "Finn
ey", "Solo", "Bonnie"]}
```

## Working with JSON in Python

- json.loads(aString): takes a string and converts it to a python data structure

- json.dumps(aDataStructure): takes a data structure and converts it to a string

## Application Programming Interfaces (APIs)

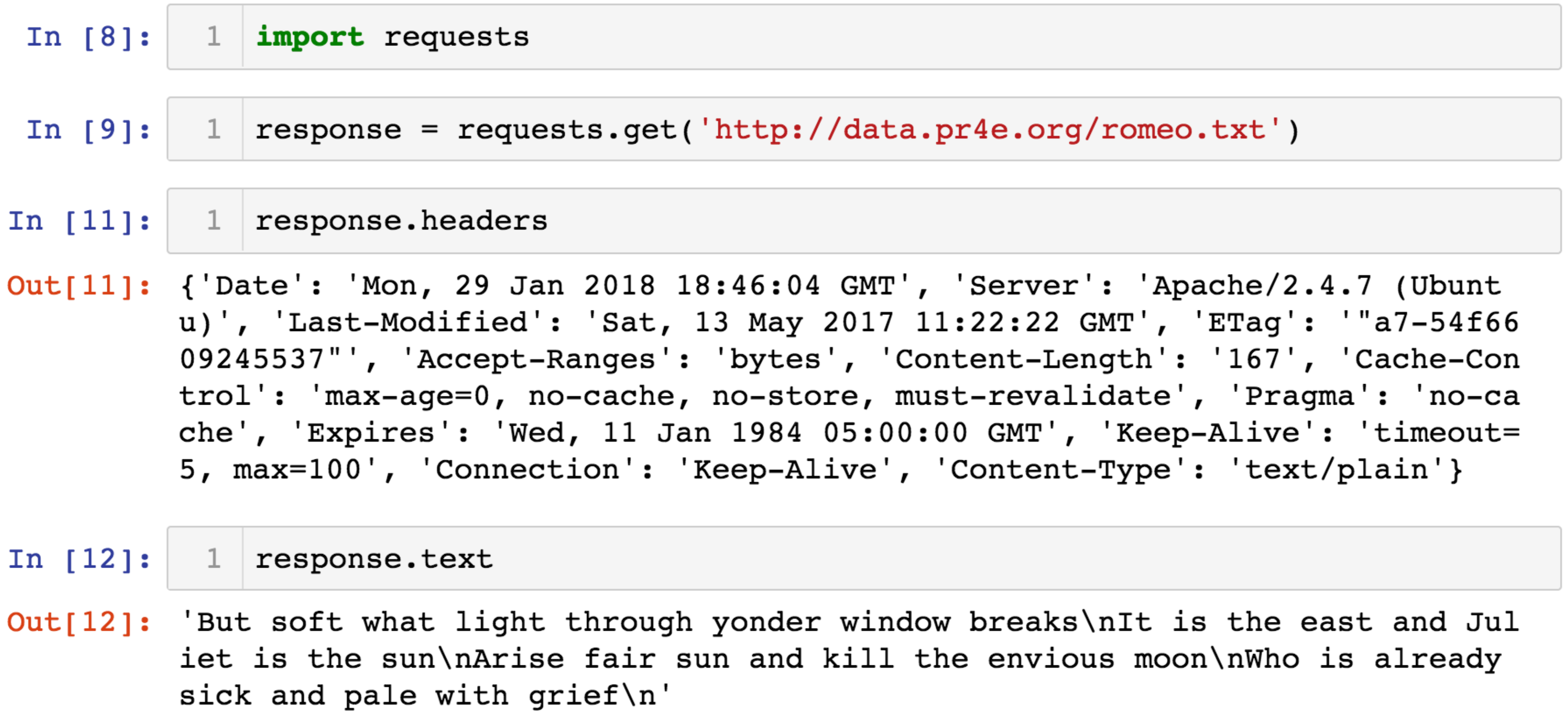
- data interchange between different applications
- you've worked with APIs before (in 106 and 206):
  - Twitter
  - Facebook
  - iTunes

  - Google Maps
  - Google Maps Geocoding
  - Slack

  - Others?

# Interacting with APIs via python

- in the old days: sockets, HTTPClient, urllib, urllib2
- now: **requests**



```
In [8]:  import requests
```

```
In [9]:  response = requests.get('http://data.pr4e.org/romeo.txt')
```

```
In [11]: response.headers

Out[11]: {'Date': 'Mon, 29 Jan 2018 18:46:04 GMT', 'Server': 'Apache/2.4.7 (Ubunt
         u)', 'Last-Modified': 'Sat, 13 May 2017 11:22:22 GMT', 'ETag': '"a7-54f66
         09245537"', 'Accept-Ranges': 'bytes', 'Content-Length': '167', 'Cache-Con
         trol': 'max-age=0, no-cache, no-store, must-revalidate', 'Pragma': 'no-ca
         che', 'Expires': 'Wed, 11 Jan 1984 05:00:00 GMT', 'Keep-Alive': 'timeout=
         5, max=100', 'Connection': 'Keep-Alive', 'Content-Type': 'text/plain'}
```

```
In [12]: response.text

Out[12]: 'But soft what light through yonder window breaks\nIt is the east and Jul
         iet is the sun\nArise fair sun and kill the envious moon\nWho is already
         sick and pale with grief\n'
```

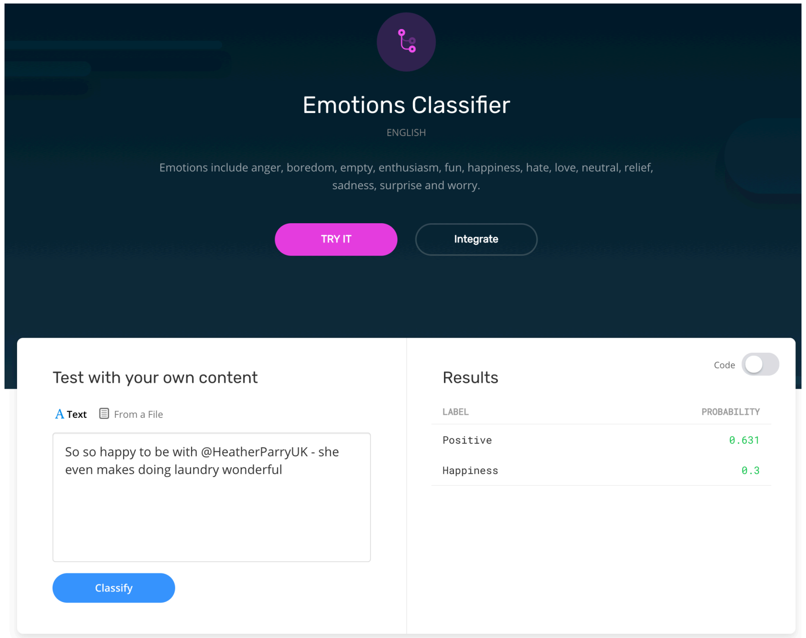
## Two basic things that APIs do

- retrieve data according to some query (like what you've done with Twitter, Facebook, iTunes)

- perform some computations on data that you send to the API

# Computationally-oriented APIs

- Monkeylearn



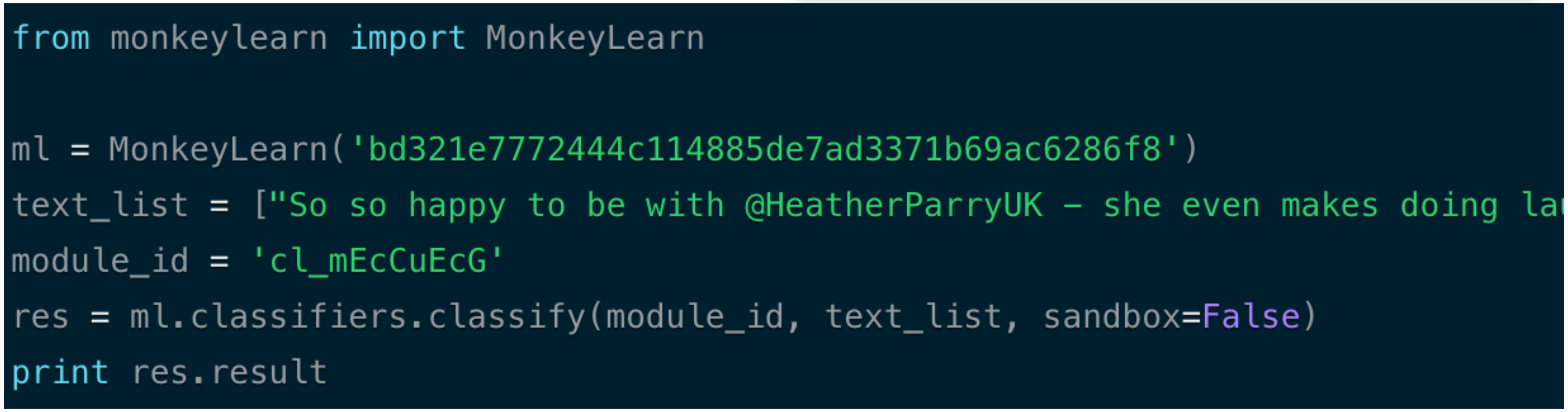
Emotions include anger, boredom, empty, enthusiasm, fun, happiness, hate, love, neutral, relief, sadness, surprise and worry.

Test with your own content

So so happy to be with @HeatherParryUK - she even makes doing laundry wonderful

Results

| LABEL | PROBABILITY |
|---|---|
| Positive | 0.631 |
| Happiness | 0.3 |



```
from monkeylearn import MonkeyLearn

ml = MonkeyLearn('bd321e7772444c114885de7ad3371b69ac6286f8')
text_list = ["So so happy to be with @HeatherParryUK - she even makes doing la
module_id = 'cl_mEcCuEcG'
res = ml.classifiers.classify(module_id, text_list, sandbox=False)
print res.result
```

## So... you're an API consumer

- you've queried databases
- you've done some computing via APIs

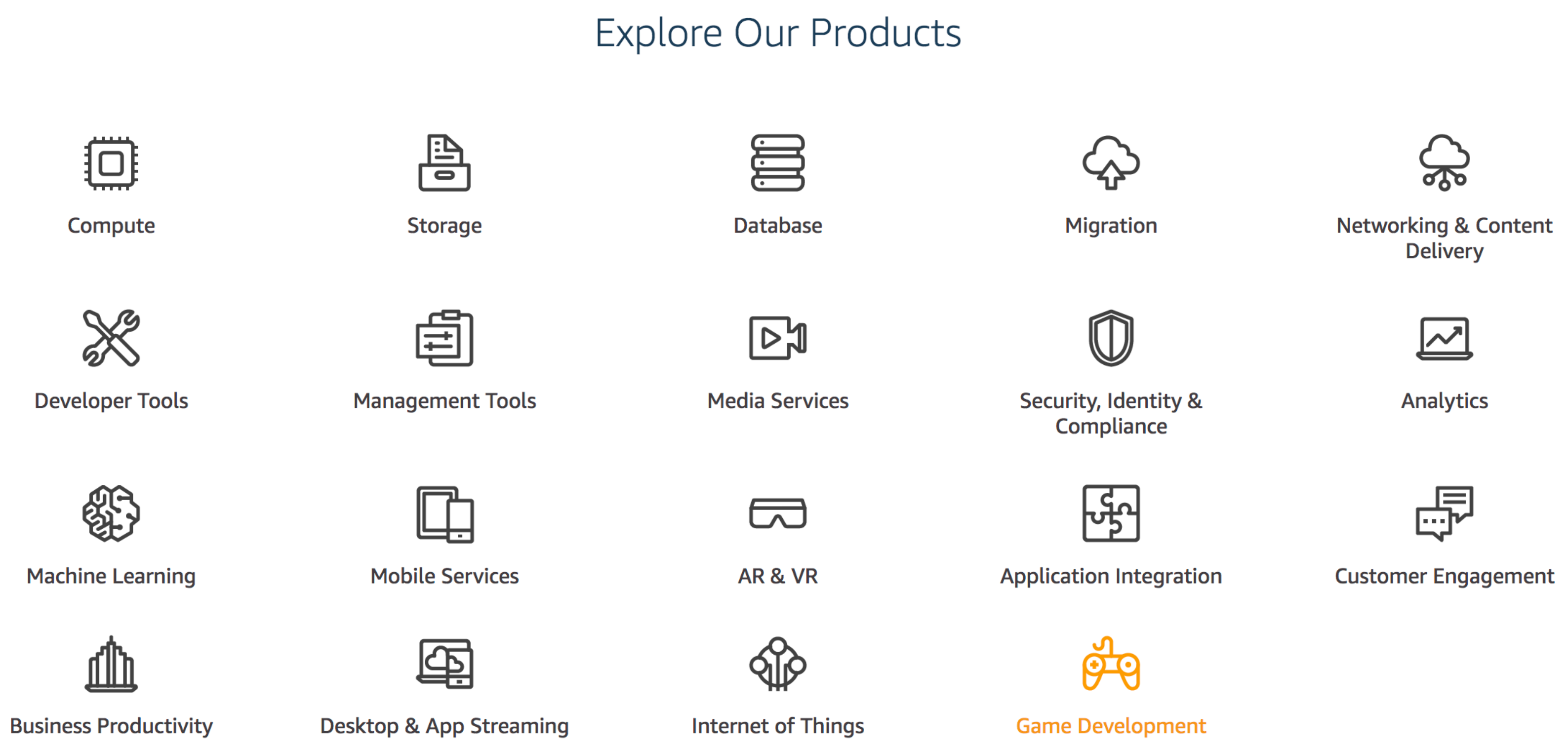
## It's time to become an API creator!

- Why?
- Single-page apps (e.g. Angular), AJAX (Asynchronous JavaScript and XML)

## How?

- In the old days:
  - spin up a machine (real or virtual), typically running Linux, set up a web server, get a framework (e.g. Flask or Django), write your API, deploy it on your server
  - what's wrong with this?
    - heavyweight
    - need to worry about security (of machine)
- Now:
  - serverless solutions

# Introducing Amazon Web Services (AWS)



Explore Our Products

Compute

Storage

Database

Migration

Networking & Content Delivery

Developer Tools

Management Tools

Media Services

Security, Identity & Compliance

Analytics

Machine Learning

Mobile Services

AR & VR

Application Integration

Customer Engagement

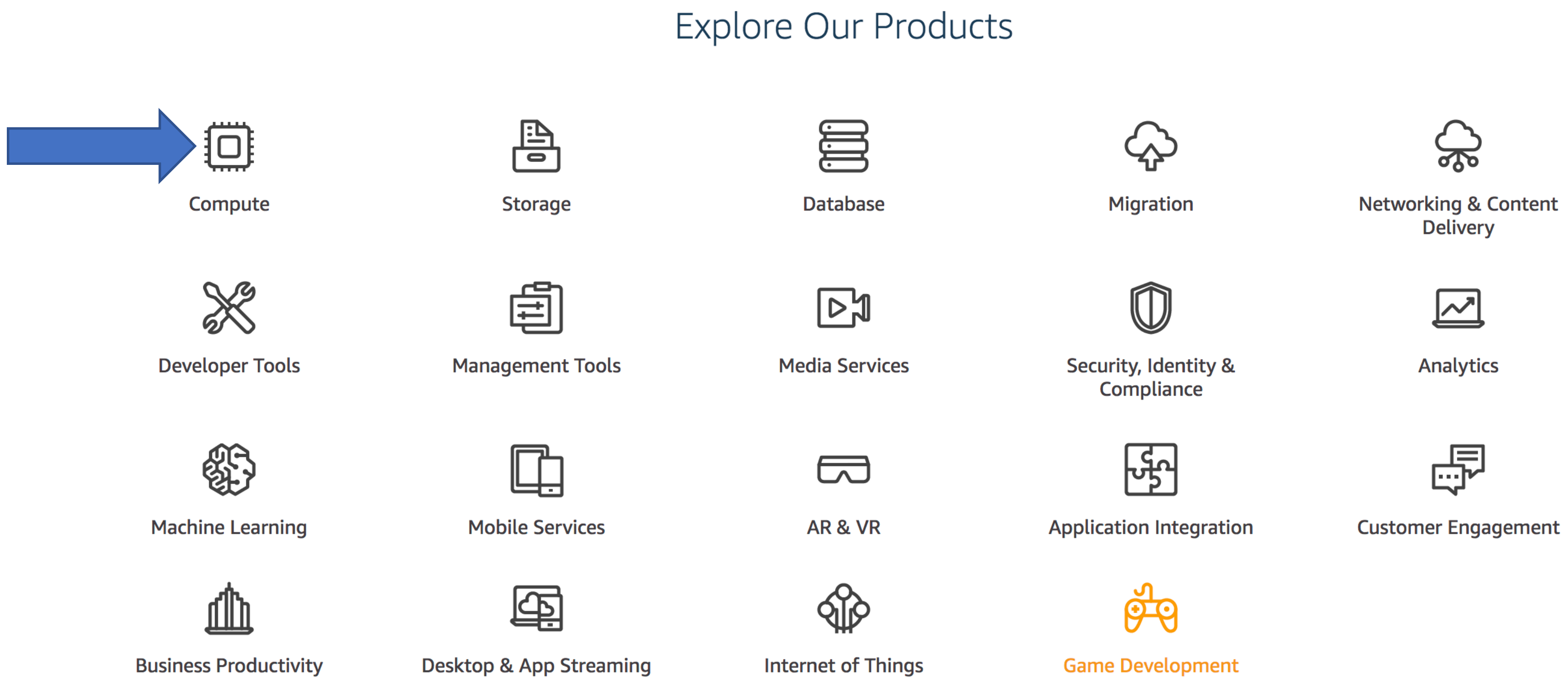
Business Productivity

Desktop & App Streaming

Internet of Things

Game Development

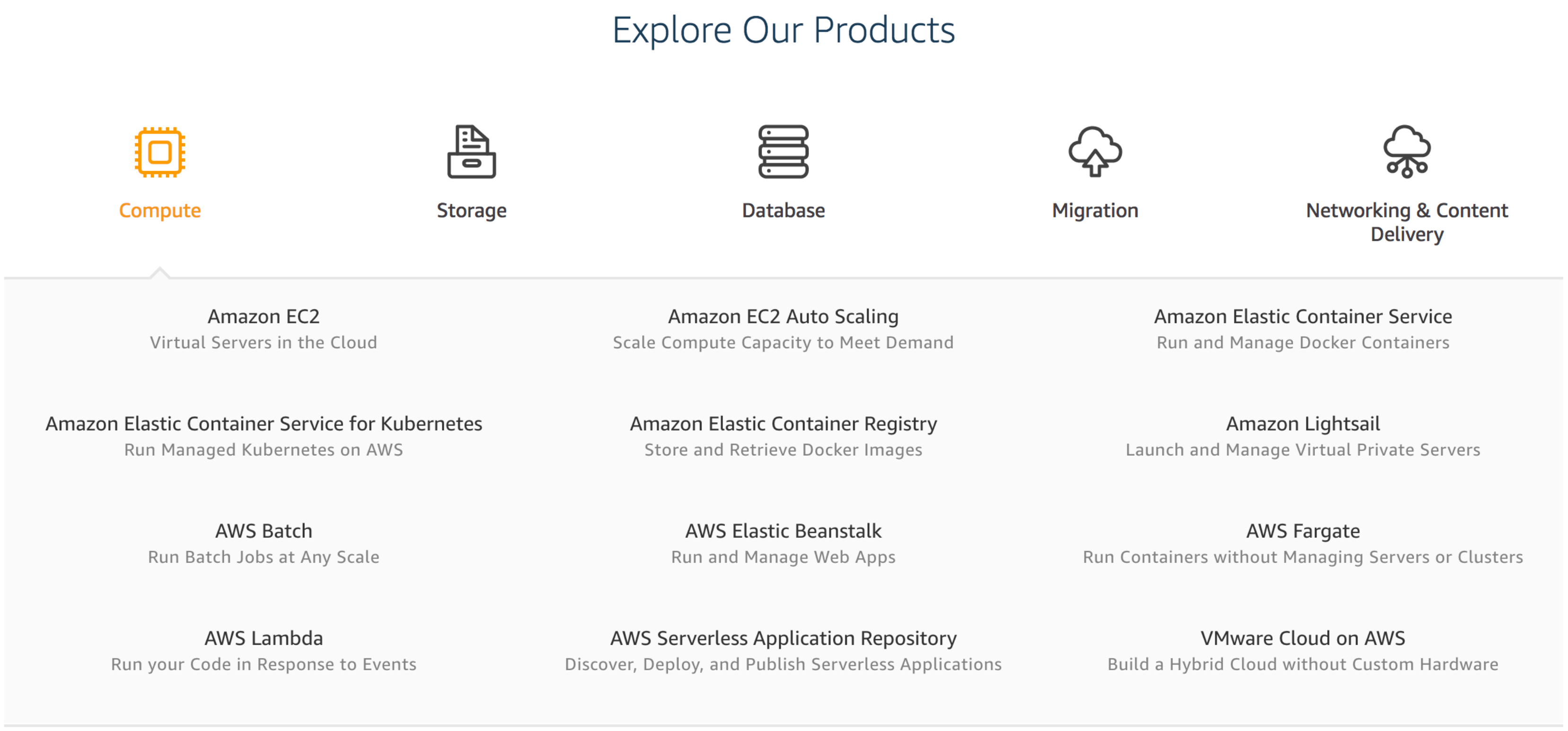
# Introducing Amazon Web Services (AWS)



Explore Our Products

| | | | | |
|---|---|---|---|---|
| Compute | Storage | Database | Migration | Networking & Content Delivery |
| Developer Tools | Management Tools | Media Services | Security, Identity & Compliance | Analytics |
| Machine Learning | Mobile Services | AR & VR | Application Integration | Customer Engagement |
| Business Productivity | Desktop & App Streaming | Internet of Things | Game Development | |

# Introducing Amazon Web Services (AWS)



Explore Our Products

Amazon EC2
Virtual Servers in the Cloud

Amazon EC2 Auto Scaling
Scale Compute Capacity to Meet Demand

Amazon Elastic Container Service
Run and Manage Docker Containers

Amazon Elastic Container Service for Kubernetes
Run Managed Kubernetes on AWS

Amazon Elastic Container Registry
Store and Retrieve Docker Images

Amazon Lightsail
Launch and Manage Virtual Private Servers

AWS Batch
Run Batch Jobs at Any Scale

AWS Elastic Beanstalk
Run and Manage Web Apps

AWS Fargate
Run Containers without Managing Servers or Clusters

AWS Lambda
Run your Code in Response to Events

AWS Serverless Application Repository
Discover, Deploy, and Publish Serverless Applications

VMware Cloud on AWS
Build a Hybrid Cloud without Custom Hardware

## The AWS Ecosystem

- Cloud-based
- Extensive
- Huge infrastructure
- Mostly reliable

## Our Focus: S3 and Lambda

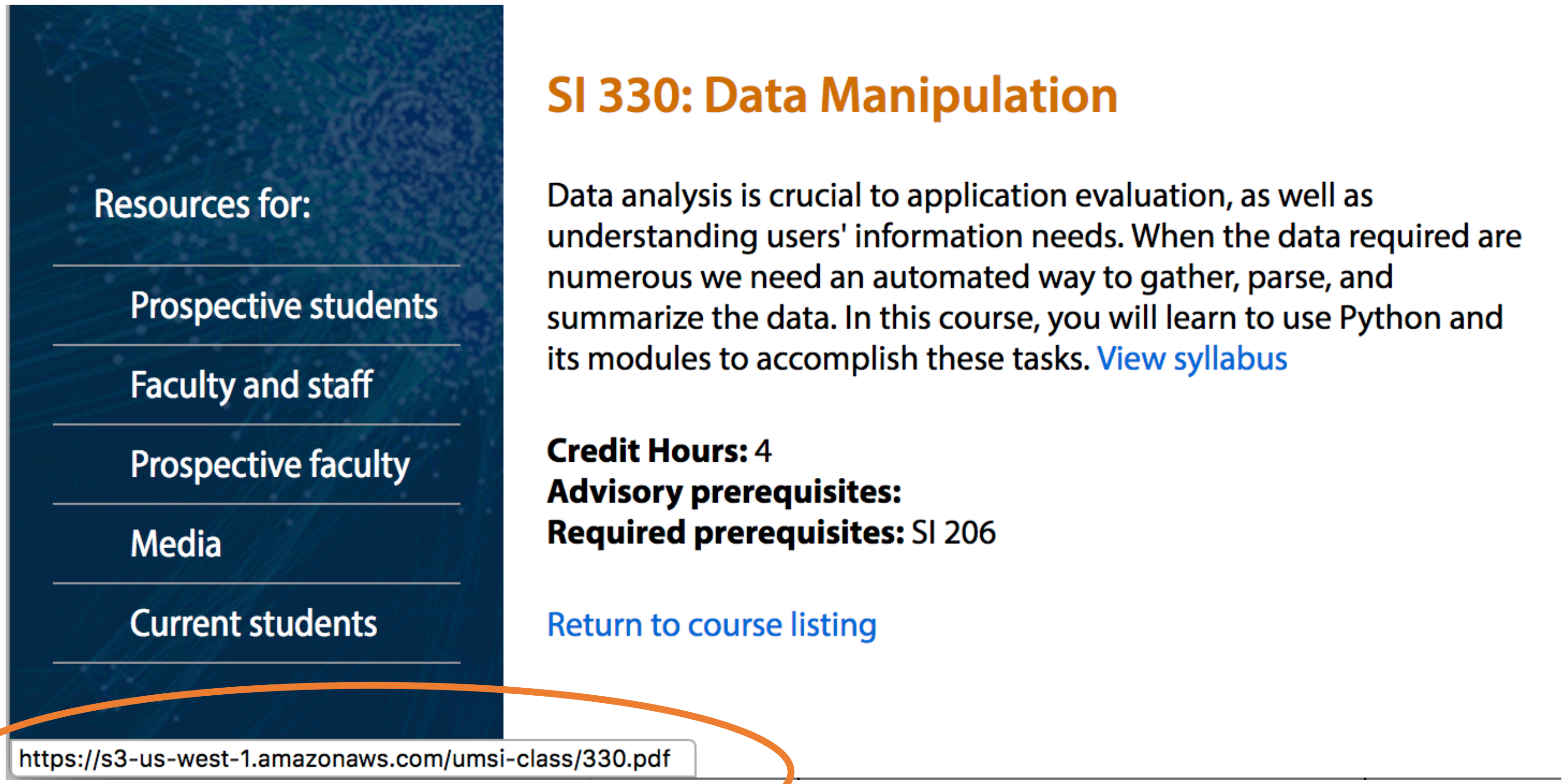
- S3: Simple Storage Service
  - basic data store
  - not a database

- Lambda (not python's lambda)
  - serverless computing
  - good for microservices (do one thing and do it well)

## S3: Simple Storage Service

- data store
- "buckets"
- up to 5 TB per bucket (!)
- powerful access control
- key benefits:
  - simplicity
  - durability
  - scalability
  - security

# You've been using S3 buckets!



Resources for:

Prospective students

Faculty and staff

Prospective faculty

Media

Current students

SI 330: Data Manipulation

Data analysis is crucial to application evaluation, as well as understanding users' information needs. When the data required are numerous we need an automated way to gather, parse, and summarize the data. In this course, you will learn to use Python and its modules to accomplish these tasks. View syllabus

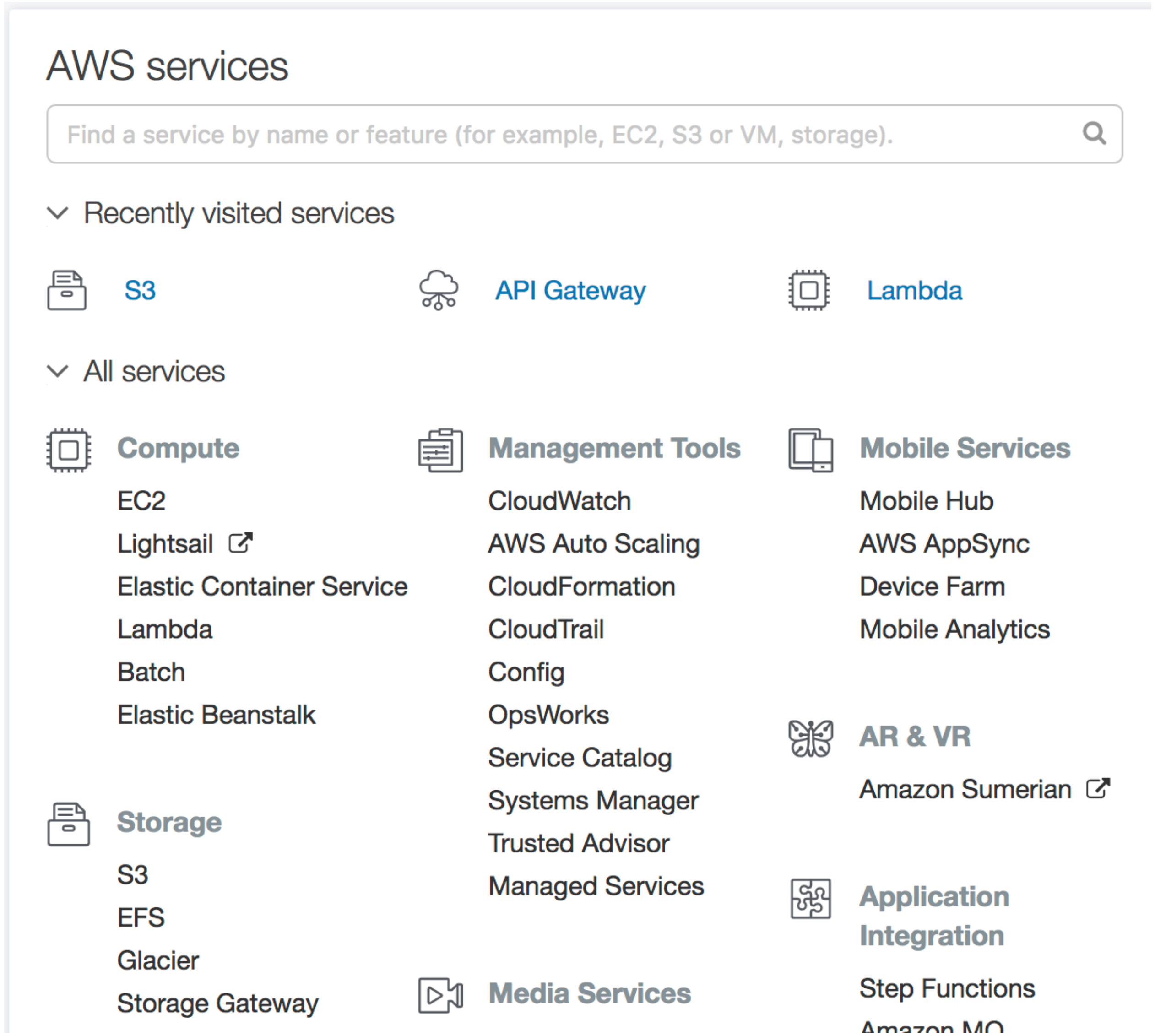
**Credit Hours:** 4
**Advisory prerequisites:**
**Required prerequisites:** SI 206

Return to course listing

https://s3-us-west-1.amazonaws.com/umsi-class/330.pdf

## So... how does this work? (Demo)

- AWS Console
  - create bucket
  - create folder
  - upload file
  - get link
  - access link



AWS services

Find a service by name or feature (for example, EC2, S3 or VM, storage).

Recently visited services

S3 | API Gateway | Lambda

All services

**Compute**
EC2
Lightsail
Elastic Container Service
Lambda
Batch
Elastic Beanstalk

**Storage**
S3
EFS
Glacier
Storage Gateway

**Management Tools**
CloudWatch
AWS Auto Scaling
CloudFormation
CloudTrail
Config
OpsWorks
Service Catalog
Systems Manager
Trusted Advisor
Managed Services

**Media Services**

**Mobile Services**
Mobile Hub
AWS AppSync
Device Farm
Mobile Analytics

**AR & VR**
Amazon Sumerian

**Application Integration**
Step Functions

## Amazon S3 Summary

- S3 is a datastore
- inexpensive, reliable, web-accessible
- simple to create and add data
- simple to access

## Amazon Lambda: serverless computing

- most computing doesn't really need a dedicated server (with concomitant support requirements)
- focus on creating functions (hence "Lambda", which has nothing really to do with python lambdas)
- think of them as "microservices": do one tiny thing and do it well
  - Note: we're not going to get too hung up on the strict definition of microservices – it's the concept that's important

## Microservices: do one thing and do it well

- loosely coupled architecture
- can be combined with other microservices to create complex apps
- allows for evolution of technology stack

- contrast with monolithic architecture pattern

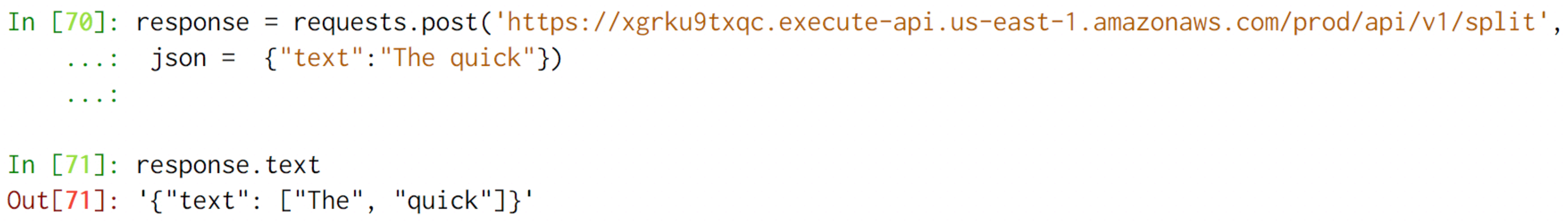
## That sounds great, but how do we make one?

- start with a real-world example:
  - create a microservice that takes a (JSON) string as input and returns a list of words (as JSON)
  - "The quick brown fox jumped over the lazy dog." ->

    ["The", "quick", "brown", "fox", "jumped", "over", "the", "lazy", "dog"]

- [https://xgrku9txqc.execute-api.us-east-1.amazonaws.com/prod/api/v1/split?text=The%20quick%20brown%20fox](https://xgrku9txqc.execute-api.us-east-1.amazonaws.com/prod/api/v1/split?text=The%20quick%20brown%20fox)

# Programmatically from python:



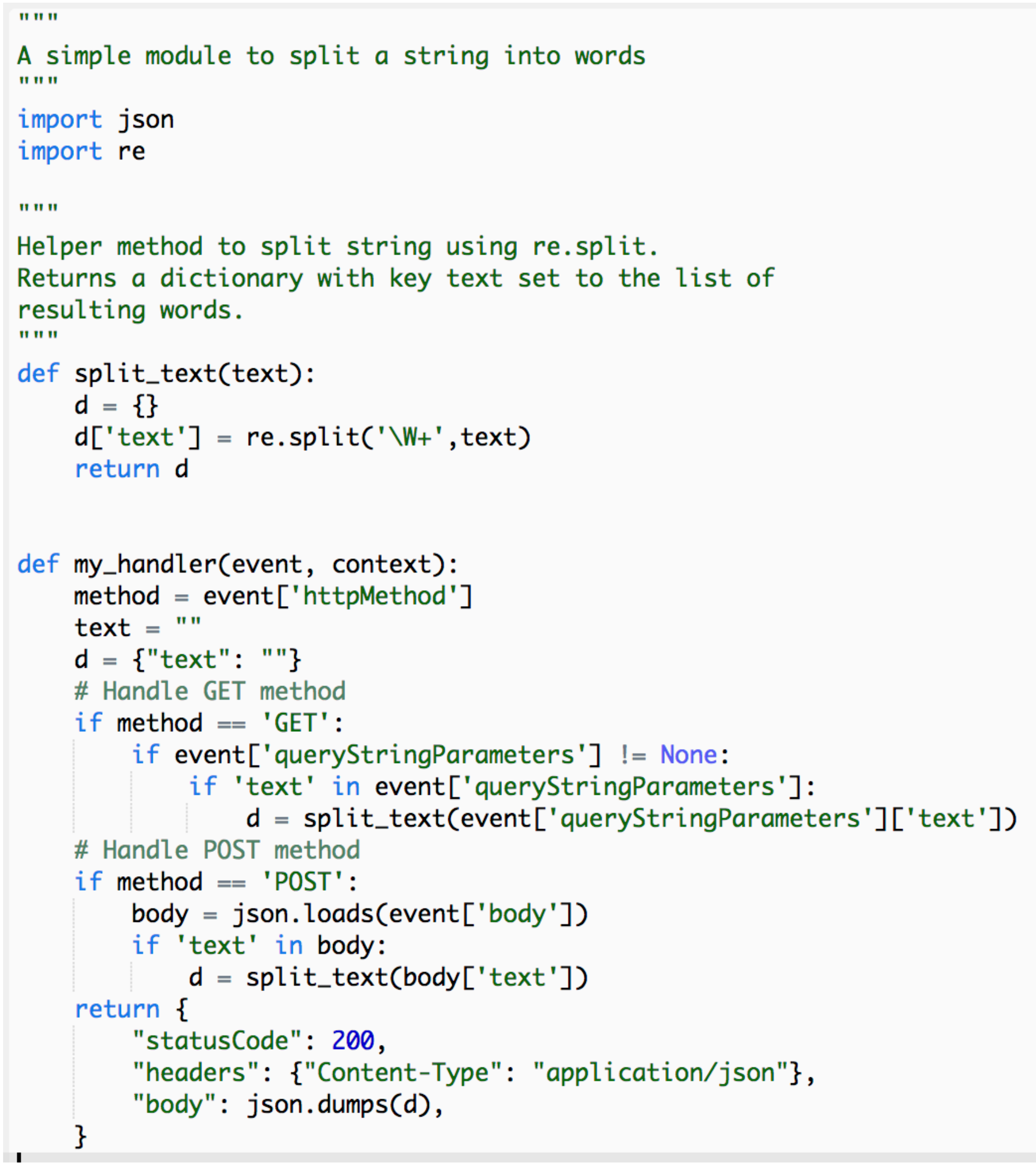
```
In [70]: response = requests.post('https://xgrku9txqc.execute-api.us-east-1.amazonaws.com/prod/api/v1/split',
    ...:  json =  {"text":"The quick"})
    ...:

In [71]: response.text
Out[71]: '{"text": ["The", "quick"]}'
```

## How does it work?

- AWS Microservice = Lambda + API Gateway + ?

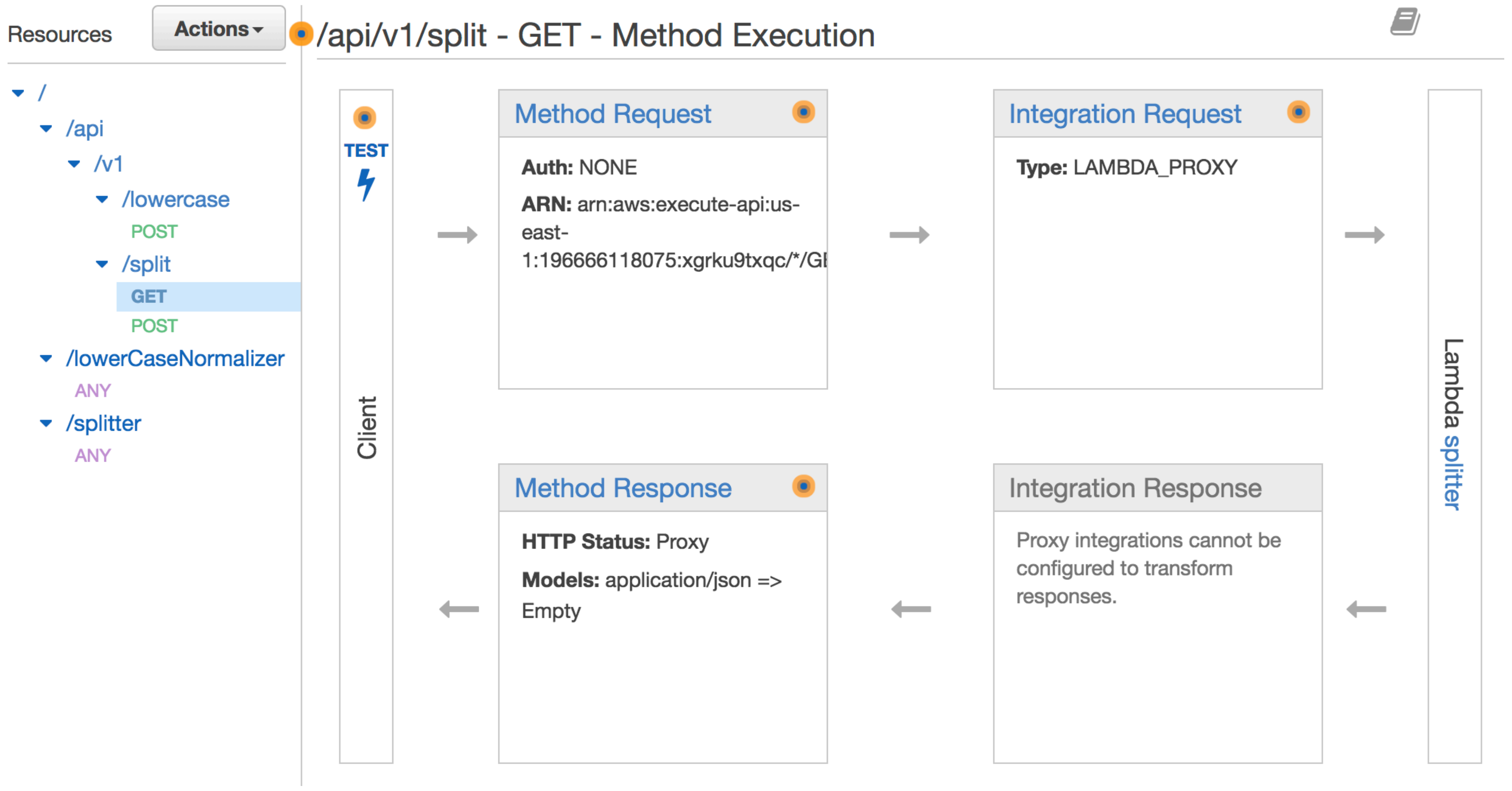
- Lambda: does the actual computations
- API Gateway: provides an HTTP(S) endpoint
- (plus optional noSQL database and logging)

## What does an AWS Lambda look like?



```python
"""
A simple module to split a string into words
"""
import json
import re

"""
Helper method to split string using re.split.
Returns a dictionary with key text set to the list of
resulting words.
"""
def split_text(text):
    d = {}
    d['text'] = re.split('\W+',text)
    return d


def my_handler(event, context):
    method = event['httpMethod']
    text = ""
    d = {"text": ""}
    # Handle GET method
    if method == 'GET':
        if event['queryStringParameters'] != None:
            if 'text' in event['queryStringParameters']:
                d = split_text(event['queryStringParameters']['text'])
    # Handle POST method
    if method == 'POST':
        body = json.loads(event['body'])
        if 'text' in body:
            d = split_text(body['text'])
    return {
        "statusCode": 200,
        "headers": {"Content-Type": "application/json"},
        "body": json.dumps(d),
    }
```

# What about the API Gateway?



/api/v1/split - GET - Method Execution

Method Request

**Auth:** NONE

**ARN:** arn:aws:execute-api:us-east-1:196666118075:xgrku9txqc/*/GE

Integration Request

**Type:** LAMBDA_PROXY

Method Response

**HTTP Status:** Proxy

**Models:** application/json => Empty

Integration Response

Proxy integrations cannot be configured to transform responses.

## Quick refresher: GET vs. POST

- GET is usually used to conduct a query
  - parameters are sent via the URL, typically with "?param1=foo&param2=bar" added to the end of the URL
  - limitations on length of URL
  - access via browser or via requests.get(...)
- POST is usually used to send data to the endpoint
  - data can be in virtually any format; we will be using JSON to send data
  - access via HTML pages that specify <FORM METHOD="POST"> or via requests.post(...)

## Let's check out AWS...

- (don't forget to get an account!)
- S3 buckets
- Creating a microservice
- https://aws.amazon.com

## Thursday's lab

- have your AWS account
- you will be creating an S3 bucket and a microservice (!)